VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

PROSTŘEDÍ PRO TESTOVÁNÍ ZAŘÍZENÍ UMOŽŇUJÍCÍCH OCHRANU PŘED DOS ÚTOKY

ENVIRONMENT FOR TESTING OF DOS ATTACK PROTECTION DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **DOMINIK TRAN**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN KUČERA**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Tran Dominik, Bc.**
Program: Informační technologie Obor: Počítačové a vestavěné systémy
Název: **Prostředí pro testování zařízení umožňujících ochranu před DoS útoky**
Environment for Testing of DoS Attack Protection Devices
Kategorie: Počítačové sítě
Zadání:

1. Seznamte se s problematikou útoků typu odepření služby (DoS) a dále nastudujte možnosti hardwarově akcelerovaného zařízení vyvíjeného v rámci sdružení CESNET pro ochranu před těmito útoky.
2. Analyzujte, především z pohledu použitelné funkcionality a dosažitelné výkonnosti, dostupné nástroje pro generování síťového provozu a vybraných typů DoS útoků.
3. Navrhněte rozšiřitelné prostředí, které umožní generování legitimního i závadného síťového provozu a automatizované testování zařízení z bodu 1 zadání. V rámci návrhu uvažujte také potřebu stavového generování síťových toků a způsob vyhodnocování správné funkcionality zařízení.
4. Prostředí implementujte jako sadu SW nástrojů. V rámci implementace vytvořte také nezbytnou sadu testů ověřující funkční i výkonnostní parametry zařízení dle specifikace.
5. V závěru diskutujte vlastnosti vytvořeného řešení, dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kučera Jan, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 3. června 2020
Datum schválení: 25. října 2019

Abstrakt

Tato práce se zabývá návrhem a implementací rozšiřitelného prostředí a nezbytné sady testů pro testování zařízení DDoS Protector z pohledu funkce i výkonnosti. Sdružení CESNET vyvíjí zařízení DDoS Protector pro ochranu proti útokům typu odepření služby (DDoS) se zaměřením na volumetrické a TCP SYN flood útoky. Vývojové prostředí DDoS Protectoru v současné době neumožňuje generování stavového (TCP) provozu a je náročné vytvořit složitější testování zařízení z pohledu interakce různých částí. Cíl práce je vytvořit prostředí, které umožní komplexnější testování DDoS Protectoru s možností generování stavového i nestavového provozu včetně multi-vector DDoS útoků a tím se přiblížit k simulaci reálného provozu na síti. Po nastudování dostupných generátorů provozu byl zvolen Cisco TRex. Následně byla vytvořena sada testů generující různé kombinace legitimního provozu a různých typů útoků, která úspěšně ověřila funkci i výkonnost DDoS Protectoru.

Abstract

This thesis deals with the development of an environment and necessary set of tests for an evaluation of the DDoS Protector device in terms of functionality and performance. CESNET is developing device called DDoS Protector for protection against denial of service (DDoS) attacks with focus on volumetric and TCP SYN flood attacks. Current development environment does not support generation of stateful (TCP) network traffic and it's difficult to create complex evaluation tests in terms of interaction between various parts of the device. Goal of this work is to create an environment which enables complex evaluation of the device, including generation of both stateful and stateless network traffic combined with multi-vector DDoS attack, thus approaching real network traffic. Cisco TRex was chosen after examination of available traffic generators. Finally set of tests generating various combination of legitimate traffic and attacks was created and DDoS Protector was successfully evaluated.

Klíčová slova

DoS, DDoS, CESNET, DDP, Protector, TRex

Keywords

DoS, DDoS, CESNET, DDP, Protector, TRex

Citace

TRAN, Dominik. *Prostředí pro testování zařízení umožňujících ochranu před DoS útoky.* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kučera

Prostředí pro testování zařízení umožňujících ochranu před DoS útoky

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kučery. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Dominik Tran
3. června 2020

Poděkování

Velmi rád bych poděkoval vedoucímu práce Ing. Janu Kučerovi za odborné vedení, ochotu a trpělivost. Také bych rád poděkoval Ing. Pavlu Krobotovi a Ing. Tomáši Podermańskimu za další pomoc a rady při vypracování této práce.

Obsah

Kapitola 1

Úvod

Využití a objem dat přenášených po počítačové síti roste každým rokem. Souměrně s tím ale také roste nutnost zabývat se bezpečností sítí a její ochranou. Útoky typu odepření služby DoS (Denial of Service) stále nabývají na síle ale i na popularitě [35]. V roce 2016 překonal DDoS útok pomyslnou hranici 1 Tbps a to s pomocí zneužitých IoT (Internet of Things) zařízení [52]. Na internetu se dají dohledat webové stránky, které nabízejí DDoS útoky jako službu za finanční částku. Útok si tak může objednat téměř kdokoliv. Také roste počet politicky či jinak myšlenkově motivovaných DDoS útoků – jedná se o tzv. hacktivism.

Sdružení CESNET reagovalo vývojem zařízení DDoS Protector umožňující ochranu před DoS útoky se zaměřením na volumetrické a TCP SYN flood útoky. Hardwarová část DDoS Protectoru je akcelerovaná FPGA síťovými kartami COMBO a NFB a dosahuje propustnosti 100 Gbps. Softwarová část realizuje hlavní řídící logiku a implementuje jednotlivé mitigační algoritmy. Detekce útoků probíhá na základě administrátorem stanovených pravidel. Vývojové prostředí zařízení DDoS Protector v současné době neumožňuje generování stavového (TCP) provozu a těžce se vytváří náročnější testování zařízení z pohledu interakce různých částí. Tato práce má za cíl návrh testovacího prostředí, které umožní komplexnější testování DDoS Protectoru s možností generování stavového i nestavového provozu a tím se přiblížit k simulaci reálného provozu na síti. Dále umožní generovat tzv. multi-vector útoky, tedy útoky složené ze současně různých typů DDoS útoků. Takovou kombinací vznikne komplexní provoz složený jak z legitimních, tak i závadných (útočných) paketů. Další části práce je návrh a implementace testů, které v tomto prostředí otestují DDoS Protector z pohledu funkčního i výkonnostního.

Práce je rozdělena do několika kapitol. První kapitola 2 se věnuje problematice DDoS útoků. Obsahuje popis vybraných typů DDoS útoků a poté se věnuje zařízení DDoS Protector a jeho principům mitigace útoků. Kapitola 3 popisuje dostupného generátory provozu se zaměřením na generátory Cisco TRex a Juniper WARP17. Další kapitola 4 popisuje návrh prostředí, volbu generátorů provozu/útoku a návrh testovacích případů pro ověření funkčnosti DDoS Protectoru. Kapitola 5 navazuje implementační části prostředí a jednotlivých testovacích případů. Kapitola 6 popisuje testování zařízení, výsledky a zhodnocení jednotlivých testovacích případů. Poslední kapitola 7 shrnuje dosažené výsledky práce.

Kapitola 2

Problematika DoS útoků

Tato kapitola popisuje úvod do problematiky útoků odepření služby a možnosti ochrany proti nim v podobě dedikovaného zařízení. První podkapitola 2.1 popisuje útoky typu DoS a jejich rozdělení. Druhá podkapitola 2.2 se zabývá zařízením DDoS Protector, zejména jakým způsobem funguje detekce a mitigace DoS útoku.

2.1 Útoky odepření služby (DoS)

DoS (Denial of Service) lze definovat jako útok, který má za cíl vyčerpat prostředky oběti [51]. Oběť se okolnímu světu jeví buďto jako nedostupná, nebo je její připojení výrazně zpomaleno. Legitimní uživatelé poté mají potíže s obětí komunikovat a využívat její služby. DoS útok je inicializován jedním zařízením. Protože se jedná o jediné zařízení, útok typicky není příliš silný. Zároveň nebývá složité toto jediné zařízení identifikovat. DDoS (Distributed Denial of Service) je podmnožina DoS útoku a liší se od něj tím, že k útoku využívá více zařízení zároveň. To znamená větší počet dostupných prostředků, tím pádem silnější útoky a složitější identifikace útočících zařízení. Pro dosažení vyššího počtu zařízení útočník využije škodlivý kód – malware, který napadne různá zařízení bez vědomí jejich uživatelů. Útočník následně dokáže taková zařízení vzdáleně ovládat a zneužít je k útoku. Takto napadeným zařízením se říká zombie a tvoří tzv. botnet [51].

Důvody k vedení (D)DoS útoků bývají různé. Jedna z hlavních motivací jsou finanční prostředky. Těch mohou útočníci dosáhnout více způsoby [51, 38]:

- *Vydírání* – útočník může vyhrožovat budoucím útokem, pokud oběť nezašle požadovanou částku. Také může nejdříve zaútočit a až poté požadovat finanční částku, aby s útokem přestal.
- *Poskytování DDoS útoků jako služby* – na internetu existují různé stránky, které umožňují komukoliv zaplatit si DDoS útok na vybraný cíl.
- *Krádež dat* – ačkoliv samotný DDoS útok nemá za cíl krádež dat nebo citlivých informací, může být využit jako kamufláž. Zatímco správci věnují svoji pozornost DDoS útoku, útočník nenápadně odcizí data z databáze a ty následně prodá.

Platba bývá požadována v kryptoměnách jako Bitcoin, kdy je vzhledem k anonymitě účtu velmi těžké vypátrat, kdo za účtem stojí. Dalším důvodem k vedení DDoS útoku je tzv. hacktivism. Ten slouží k propagaci určité myšlenky, typicky politické nebo náboženské a také ke zviditelnění se. Širší populaci je tímto známá skupina Anonymous stojící například za útoky na webové stránky turecké vlády [8]. Poslední motivací může být jednoduše

způsobení škod. Může se jednat o pomstu propuštěného zaměstnance nebo o konkurenční boj. Výpadek služeb oběti často způsobí finanční ztráty. V případě e-shopu se jedná o ztracené zákazníky, poškození reputace a nutnost znovuzprovoznění služby. V případě hostování služby u poskytovatele cloudové infrastruktury by si poskytovatel mohl účtovat vyšší poplatek za zvýšené využití zdrojů.

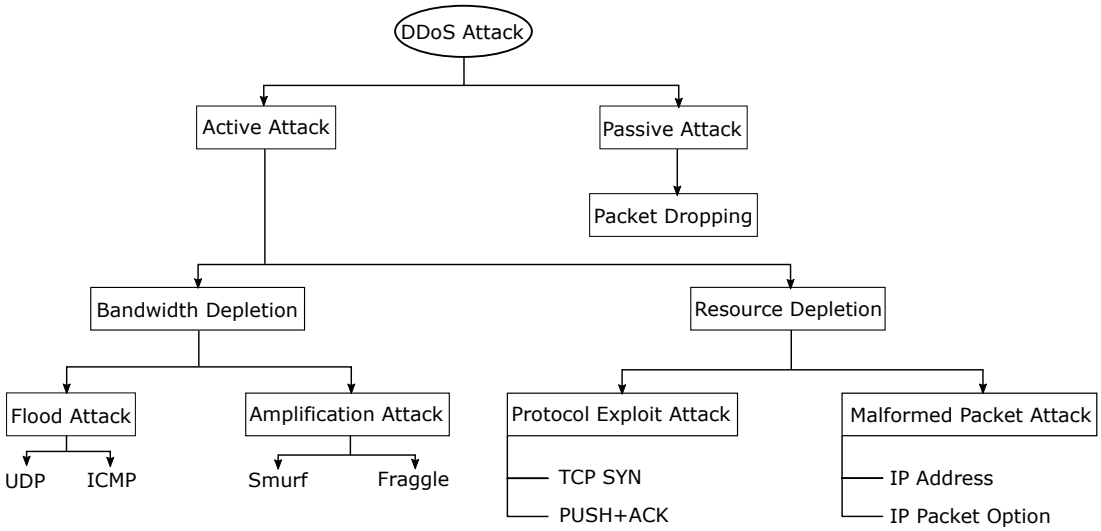
2.1.1 Rozdělení DoS útoků

Různé zdroje uvádí mírně odlišná rozdělení (D)DoS útoků do kategorií. Podle [9] (na obrázku 2.1) lze DoS útoky dělit na aktivní a pasivní. Zahazování paketů (packet dropping) je jediný zmíněný zástupce pasivních útoků. Jedná se o situaci, kdy směrovací či přepínací zařízení (obecně síťový uzel přeposílající pakety) zahazuje pakety i přesto, že síť ani zařízení nejsou nijak vytíženy. Z pohledu této práce nejsou pasivní útoky zajímavé a zbytek práce se tak bude zabývat pouze aktivními DoS útoky. Aktivní útoky lze dělit na dva základní typy: vyčerpání konektivity a vyčerpání zdrojů.

Útoky, které cílí na konektivitu generují velký objem provozu a snaží se o zahlcení přípojného spoje oběti. Množství paketů se i přes zahlcený spoj dostane k oběti a dodatečně vyčerpává její prostředky. Je-li síť dobře dimenzovaná a úzkým hrdlem je samotné síťové rozhraní oběti, pak útok zasáhne pouze cílovou oběť. Naopak pokud je útok dostatečně silný, tak ovlivní i další zařízení v koncové síti zahlcením koncového směrovače. Tyto útoky lze dále dělit na tzv. flood (záplavové) a amplifikační útoky. U flood útoků je provoz generován pouze zařízeními, které útočník přímo ovládá (botnet). Typickým příkladem jsou UDP nebo ICMP flood útoky. U amplifikačních útoků jsou zneužita i další, nezávislá zařízení. Z botnetu jsou jim zasílány dotazy s podvrženou zdrojovou IP adresou oběti. Na tyto dotazy zařízení odpoví podle pravidel použitého protokolu. Odpověď zařízení je ovšem typicky mnohem větší než přijatý dotaz. Dochází tím pádem k amplifikaci (zesílení), kdy se z odeslaného paketu malé velikosti stává paket velikosti mnohem větší. Často bývají zneužity otevřené DNS resolvery. Dotazový paket je také možné směrovat jako broadcast, pak odpoví více než jedno zařízení. Příkladem je Smurf útok využívající protokol ICMP. Odpověď sice není větší než dotaz, ale protože odpovídá více zařízení, tak ve výsledku stále dochází k zesílení. Amplifikační útoky lze také označit jako odrazové (reflected), protože útočné pakety jsou odraženy od neutrálních zařízení na oběť.

Útoky, které cílí na vyčerpání zdrojů mívají menší celkový objem než útoky na konektivitu. Zaměřují se na hardwarové a softwarové prostředky oběti jako je paměť, výpočetní výkon nebo různé limity, ať už systémové nebo uživatelem definované. Útoky je možné rozdělit na další dvě podkategorie. První jsou protokolové útoky, které využívají slabin návrhu protokolů. Nejčastější představitel je útok TCP SYN flood. Druhá podkategorie jsou útoky s poškozenými či jinak modifikovanými pakety, které cílí na chyby v implementaci protokolů. Sem patří například útok Teardrop, který využíval chybnou implementaci sestavení fragmentovaných paketů v IP protokolu.

Firma Cisco dělí DoS útoky do tří kategorií [10]: volumetrické, útoky na aplikační vrstvě a tzv. low-rate DoS útoky. Volumetrické DoS útoky odpovídají útokům vyčerpání konektivity podle [9]. Aplikační útoky cílí na protokoly pouze na L7 vrstvě a snaží se zahltit jejich dostupné zdroje (tabulky, paměť). Low-rate DoS se zaměřují na slabiny návrhu a implementace protokolů a s velmi malým počtem paketů vyčerpává zdroje oběti. Typický představitel je Slowloris. Aplikační i low-rate DoS útoky lze zařadit do kategorie útoků vyčerpání zdrojů podle [9].



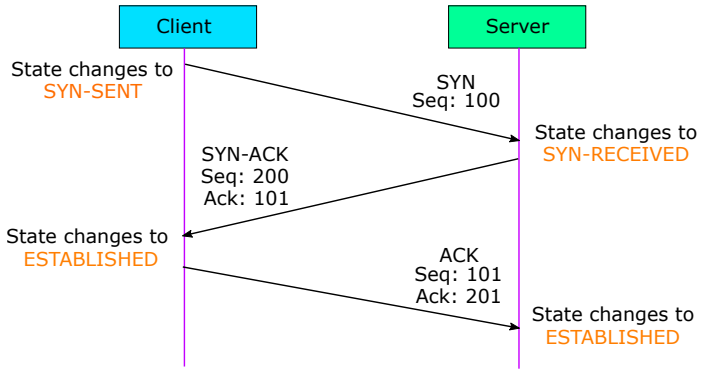
Obrázek 2.1: Taxonomie DoS útoků. [9].

Firma Imperva také dělí DoS útoky do tří kategorií [25]: volumetrické, útoky na aplikační vrstvě a útoky na protokoly. Mezi útoky na aplikační vrstvě ale řadí i low-rate DoS útoky. Protokolové útoky se v tomto rozdělení zaměřují na vrstvu L3, L4 a patří mezi ně TCP SYN flood. Identické rozdělení se nicméně dá dohledat i od firmy Cisco, ovšem na jiné stránce [12]. Rozdělení DoS útoků skutečně nemusí být jednostranné a některé útoky se dají zařadit do více kategorií.

2.1.2 Vybrané (D)DoS útoky

Tato podkapitola detailněji popisuje některé konkrétní typy DoS útoků.

TCP SYN flood je podle dat společnosti Kaspersky Lab nejčastější druh DoS útoku se zastoupením až 80 % [35]. Zaměřuje se na protokol transportní vrstvy TCP, který před zahájením samotné komunikace dvou zařízení vyžaduje, aby zařízení ustanovila spojení a provedla tzv. třícestný handshake [26]. Ten je zobrazen na obrázku 2.2 a probíhá následovně:



Obrázek 2.2: Diagram s jednotlivými kroky ustanovení spojení TCP handshake. [23].

1. *SYN*: Klient zašle serveru TCP segment s příznakem SYN (Synchronize) a náhodně vygenerovaným sekvenčním číslem Seq_c.

2. *SYN-ACK*: Server klientovi odpoví TCP segmentem s příznaky SYN a ACK (Acknowledge). Příznak ACK potvrzuje zatím jednosměrné navázání spojení ze strany klienta. Spojení musí být obousměrné, proto je ještě nutné navázat spojení ze strany serveru. Kvůli tomu je v TCP segmentu ještě příznak SYN. Potvrzovací číslo Ack_c nastaví jako přijaté sekvenční číslo zvýšené o jedničku ($Ack_c = Seq_c + 1$) a zvolí náhodné sekvenční číslo Seq_s.

3. *ACK*: Klient odpoví serveru TCP segmentem s příznakem ACK a potvrzuje tak spojení ze strany serveru. Sekvenční číslo Seq_c nastaví jako přijaté potvrzovací číslo zvýšené o jedničku ($Seq_c = Ack_c + 1$) a potvrzovací číslo Ack_s nastaví jako přijaté sekvenční číslo zvýšené o jedničku ($Ack_s = Seq_s + 1$).

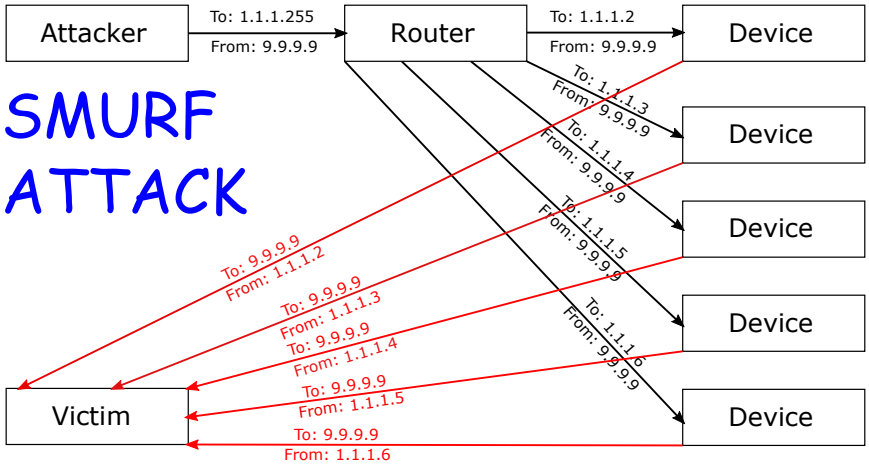
Potom, co server přijme potvrzovací paket od klienta, je spojení plně navázáno a oba účastnící mohou komunikovat. TCP SYN flood spočívá v technice, kdy útočník na oběť zasílá velké množství SYN paketů, na které již neodpoví. Při zpracování příchozího SYN paketu se v paměti vytvoří záznam TCB (Transmission Control Block), který uchovává stav a další informace o spojení. Oběť následně odpoví paketem s příznaky SYN-ACK a čeká na potvrzení od klienta. Potvrzení ovšem nikdy nedorazí a oběť tak v paměti musí držet mnoho tzv. napůl otevřených spojení [26]. Jádro operačního systému se snaží chránit před vyčerpáním veškeré paměti a tak záznamy TCB ukládá do tzv. backlogu [22], který výrazně omezuje jejich množství. Pokud je naplněn limit backlogu, pak oběť nebude moci navazovat nová TCP spojení s legitimními uživateli.

UDP flood je druhý nejčastěji používaný typ DoS útoku se zastoupením kolem 10 % [35]. UDP nevyžaduje ustanovení spojení, zařízení spolu mohou komunikovat okamžitě. Při přijetí UDP datagramu zařízení zkontroluje, zdali v systému existuje uživatelská aplikace, která naslouchá na cílovém portu. Pokud ne, tak vygeneruje ICMP zprávu *Destination unreachable* a zašle ji zpět. Obecný UDP flood generuje velké množství UDP paketů s náhodnými cílovými porty [17]. Hledání v datových strukturách, generování a odesílání ICMP zpráv stojí oběť určitý procesorový čas. Je-li UDP paketů příliš, pak oběť nebude schopná zpracovat všechny příchozí pakety a tím pádem se pro ostatní zařízení bude jevit buď jako nedostupná, nebo bude komunikovat velmi pomalu a nestabilně.

UDP nevyžaduje ustanovení spojení a neověřuje, s kým komunikuje. To jej dělá snadný cíl pro podvržení falešných IP adres a tím pádem i k odrazovým a amplifikačním útokům. Častý je amplifikační útok zneužívající službu DNS (Domain Name System) běžící na portu 53. Ta poskytuje mj. překlad doménových jmen na IP adresy [13] (a také může využívat protokol TCP, ale použití UDP je častější). Útočník pošle požadavek o záznamy na DNS server s podvrženou zdrojovou IP adresou oběti. Požadavek typicky obsahuje typ záznamu *ANY*, na který server vrátí všechny záznamy, které zná. Zatímco požadavek je malý (cca 70 B), odpověď směřující na oběť bývá mnohem větší (např. 3500 B) a dochází tak k amplifikačnímu faktoru kolem 50. Maximální možný amplifikační faktor u DNS je 179 [46]. Podobným způsobem lze zneužít i služby jako SSDP (Simple Service Discovery Protocol) nebo NTP (Network Time Protocol, opraveno v novějších verzích). Problémem DNS amplifikačních útoků jsou otevřené DNS resolvery [13], které odpověď poskytují komukoliv, kdo si o ni zažádá. Správci DNS resolverů je mohou nakonfigurovat tak, aby odpověď poskytly pouze zařízením z ověřených domén a snížili tak jejich zneužití.

ICMP flood je jednoduchý útok, při kterém útočník zahlcuje oběť pakety typu ICMP *echo request*, na které oběť odpovídá pakety typu ICMP *echo reply* [15]. ICMP je protokol

zapouzdřený v IP protokolu a poskytuje diagnostické informace. Smurf útok ilustrovaný na obrázku 2.3 je speciální případ ICMP floodu, kdy útočník odešle ICMP *echo request* paket s broadcast cílovou IP adresou a podvrženou zdrojovou IP adresou oběti. Všechny zařízení v cílové síti směřují odpověď ICMP *echo reply* na oběť. Fraggle attack je UDP varianta smurf útoku, která cílí na UDP port 7 (Echo) [24].



Obrázek 2.3: Princip smurf útoku. [45].

HTTP flood je útok na protokol aplikační (L7) vrstvy HTTP. Útočník generuje velké množství HTTP dotazů, které mohou být náročné na zpracování a vyčerpají zdroje oběti. Podle typu HTTP dotazu lze útok dělit na dva typy [14]:

- *HTTP GET* – žádá HTTP server o zaslání objektů statických jako soubory, obrázky, text apod. Dotazy bývá nenáročně vyřídit.
- *HTTP POST* – odesílá data na server, například vyplněný formulář. Server data může ukládat do databáze, většinou provádí komplexní kontrolu vstupních dat, která může být náročná na zdroje.

HTTP flood útoky bývá těžší odhalit, protože se skládají z legitimně vypadajících dotazů a zároveň nemusí generovat tak vysoký objem provozu, aby oběť přetížili. To platí zejména pokud zpracování dotazů na straně oběti není dobře optimalizované a může generovat stovky databázových dotazů.

Slowloris je další typ útoku na protokol HTTP a zároveň typický představitel low-rate DoS útoků [16]. Slowloris otevírá mnoho HTTP spojení se serverem a v každém spojení posílá částečný požadavek na server, který ovšem nikdy zcela nedokončí. V pravidelných časových intervalech (např. 15 sekund) pošle další část HTTP požadavku, aby server neukončil spojení na základě timeoutu. Server má nastaven určitý maximální limit současných HTTP spojení, které eventuálně Slowloris vyčerpá. Webová stránka oběti pak bude ostatním uživatelům nedostupná. Některé webové servery pro každé spojení vytvořili nové vlákno. Počet hardwarových vláken (jader procesoru) je omezen a vytváření softwarových vláken nad hardwarový limit není dobře škálovatelné.

Vzhledem k velmi malému množství provozu, které Slowloris generuje, je těžké ho odhalit konvenčními metodami, které sledují celkový objem provozu směřující na server. Správci

webového serveru ovšem na rozdíl od volumetrických DoS útoků mají určité možnosti, jak se útoku Slowloris bránit sami. V konfiguraci serveru mohou například [16]:

- snížit timeout čas,
- snížit maximální počet současných spojení z jedné IP adresy,
- snížit čas, jak dlouho klienti mohou udržovat jedno otevřené spojení,
- zvýšit počet současně otevřených spojení.

Existují další útoky založené na podobném principu jako Slowloris [34]. Zmíněn bude útok R-U-Dead-Yet, který na server odesílá data, ale velmi pomalu. Opakem je útok Slow Read, který ze serveru čte data, opět velmi pomalu a drží tak otevřené spojení po velmi dlouhou dobu.

2.2 DDoS Protector

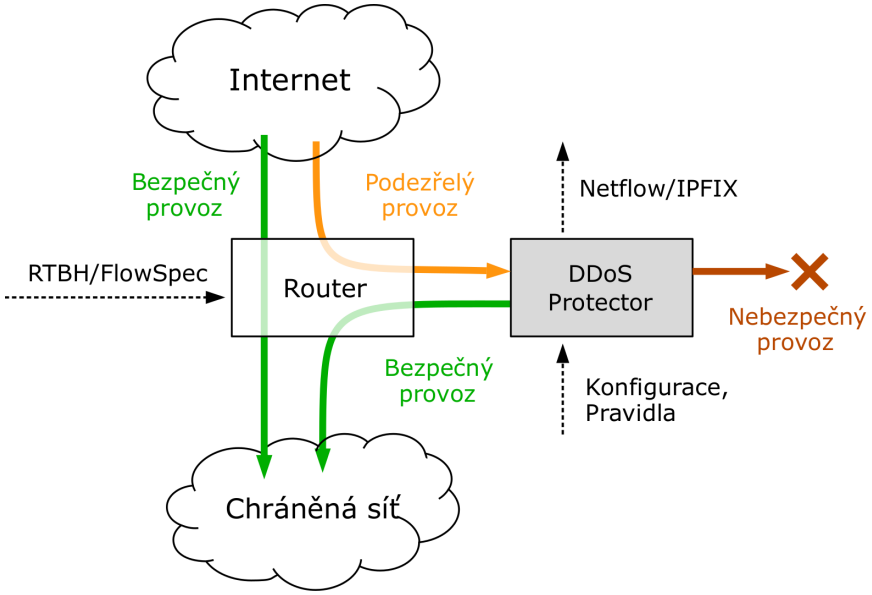
Tato část vychází z uživatelského manuálu zařízení DDoS Protector [7], není-li uvedeno jinak.

DDoS Protector je aktivní síťové zařízení, které chrání vybrané cílové sítě před příchozím DDoS útokem. Je vyvíjen zájmovým sdružením právnických osob CESNET. DDoS Protector (dále už jen Protector) se zaměřuje na volumetrické DDoS útoky, které cílí na přípojný spoj oběti, ale i na protokolové útoky TCP SYN flood. Volumetrickým útokům se oběť sama nemůže nijak bránit, ochranu musí zajistit poskytoval připojení. Proto předpokládané umístění Protectoru je před chráněnou sítí, ve které se oběť nachází. Ukázka zapojení Protectoru je na obrázku 2.4. Zde je Protector zapojen v kombinaci se směrovačem, který do něj zasílá provoz směřující do chráněné sítě. Provoz je v Protectoru analyzován a pokud není detekován DDoS útok, je provoz zaslán zpět na směrovač. Ten jej následně směruje k cílovým zařízením v chráněné síti. Naopak je-li detekován probíhající DDoS útok, jsou pakety útočících zařízení zahozeny. Ke směrovači zpětně přichází odfiltrovaný provoz o menším objemu, efektivita DDoS útoku je tím pádem snížena.

Protector disponuje propustností 100 Gbps, resp. 150 Mpps (150 miliónů paketů za sekundu). K dosažení vysoké propustnosti využívá hardwarovou akceleraci ke zpracování paketů v podobě FPGA (Field Programmable Gate Array) síťových karet COMBO (CESNET) a/nebo NFB (Netcope). Karty COMBO využívají konkrétně FPGA čipy Virtex7 a Virtex UltraScale+ od firmy Xilinx [36] a jsou dále doplněny o paměti, PCI-Express, síťové konektory apod. Softwarová část Protectoru zajišťuje celkové ovládání chování celého zařízení, logování, hlavní logiku analýzy paketů a algoritmy mitigace útoku. Detekci DDoS útoku Protector vyhodnocuje na základě pravidel, které nastaví administrátor sítě. Pravidlo je vždy zadáno pro konkrétní cílovou síť – prefix a udává vlastnosti, které paket musí splňovat, aby byl započítán do statistik daného pravidla. Detekce probíhá v periodických časových intervalech označených jako pozorovací okno (observation window). Protector je stále aktivně rozvíjen a jedním z připravovaných rozšíření je integrace IDS (Intrusion Detection System) systému pro blokování útoků i na vyšších vrstvách komunikace, tj. aplikační L7 vrstvě.

2.2.1 Volumetrické útoky

Mitigaci volumetrických útoku zajišťuje amplifikační modul. V případě volumetrických DDoS útoků je hlavní myšlenka v nastavení tzv. *threshold* a *limit* hodnot, které se zadá-



Obrázek 2.4: Ukázka zapojení DDoS Protectoru v síti se směrovačem [7].

vají v bajtech nebo paketech za sekundu. Překročí-li objem provozu směřující do chráněné sítě hodnotu *threshold* daného pravidla (detekce útoku), pak Protector začne provoz mitigovat tak, aby objem provozu klesl na/pod hodnotu *limit*. Mitigaci provede tak, že začne zahazovat pakety od těch IP adres, které poslali největší objem provozu (tzv. Top N) za dané pozorovací okno. Každý přijatý paket je zkontrolován na shodu se všemi pravidly. Pokud nastane shoda, je připočítán do statistik daného pravidla. Pravidla se v novější verzi Protectoru zadávají do databáze pomocí specializovaného nástroje `ddpctl`. Zde bude představen starší textový formát pravidel, jelikož je vhodnější pro popis jednotlivých položek. Sémantika je ovšem identická.

```
<RULE> <DST-NET> <PROTOCOL> <SRC-PORT> <DST-PORT> <LENGTH> <FRAGMENTATION>
<TCPFLAGS>  <THRESHOLD> <LIMIT> <WHITELIST>
```

Kde jednotlivé položky mají následující význam:

- `RULE`: Nepovinný textový komentář k pravidlu.
- `DST-NET`: Chráněná cílová síť (prefix) ve standardním formátu (např. 147.229.0.0/16). Může být zadán IPv4 i IPv6 prefix. Tato položka je povinná.
- `PROTOCOL`: Udává protokol paketu podle hodnoty položky *Protocol* v IP hlavičce (TCP, UDP, ICMP...). Nepovinná položka.
- `SRC-PORT`: Zdrojový port paketu. Může být zadán jako menší než, větší než, rozsah (od-do) nebo přesná shoda. Nepovinná položka.
- `DST-PORT`: Cílový port paketu. Může být zadán jako menší než, větší než, rozsah (od-do) nebo přesná shoda. Nepovinná položka.
- `LENGTH`: Délka paketu. Může být zadána jako menší než, větší než, rozsah (od-do) nebo přesná shoda. Nepovinná položka.

- `FRAGMENTATION`: Nepovinná položka. Určuje fragmentaci paketu a může nabývat následujících hodnot:
  - *YES*: Pouze fragmentované pakety včetně první a poslední části paketu.
  - *NO*: Pouze nefragmentované pakety.
  - *FIRST*: Pouze první část fragmentovaného paketu.
  - *LAST*: Pouze poslední část fragmentovaného paketu.
  - *MID*: Pouze prostřední části fragmentovaného paketu, tzn. bez první a poslední části.
  - *NOFIRST*: Všechny části fragmentovaného paketu s výjimkou první části.
- `TCPFLAGS`: Nepovinná položka. Určuje TCP příznaky v paketu a může nabývat následujících hodnot:
  - *S*: Příznak SYN.
  - *A*: Příznak ACK.
  - *F*: Příznak FIN.
  - *P*: Příznak PUSH.
  - *R*: Příznak RESET.

  Zároveň je možné příznak invertovat, což pak znamená, že příznak se v paketu nesmí vyskytovat. Tuto položku lze nastavit pouze v případě, kdy je položka `PROTOCOL` nastavena na TCP.
- `THRESHOLD`: Povinná položka. Udává hodnotu v bajtech nebo paketech za sekundu, kdy po překročení této hodnoty začne Protector mitigovat provoz spadající do tohoto pravidla.
- `LIMIT`: Povinná položka. Udává hodnotu v bajtech nebo paketech za sekundu (musí se shodovat s položkou `THRESHOLD`), na kterou se provoz sníží v případě, že překročí hodnotu danou položkou `THRESHOLD`.
- `WHITELIST`: Nepovinná položka. Udává seznam bezpečných IP adres, od kterých se pakety nebudou zahazovat.

Pokud administrátor sítě chce zadat pravidlo, které bude chránit prefix 147.229.0.0/16 před například amplifikačním DNS útokem popsaným v kapitole 2.1.2, pak může nastavit položky pravidla třeba následovně (za předpokladu, že typický maximální objem DNS provozu v dané síti je 5 Gbps):

```
RULE: "DDoS: DNS amplification attack rule"
DST-NET: 147.229.0.0/16
PROTOCOL: UDP
SRC-PORT: 53
THRESHOLD-BITS: 10000000000
LIMIT-BITS: 6000000000
```

Hodnota `THRESHOLD` je nastavena na dvojnásobek typické špičky pro neobvyklé, ale pravděpodobně legitimní situace. Pokud je překročena, pak se s vysokou pravděpodobností jedná o DDoS útok. Protector začne zahazovat pakety od těch zařízení, které posílají největší objem provozu. Zablokuje tolik zařízení, aby se celkový objem snížil pod 6 Gbps (rezerva 1 Gbps). V současné verzi je Protector schopen zablokovat až 32768 unikátních zdrojových IP adres.

2.2.2 TCP SYN flood útoky

Původně byl Protector navržen pouze pro ochranu proti volumetrickým, amplifikačním útokům. TCP SYN flood je ovšem stále častější se zastoupením až 80 % [35]. Později tak byla funkčnost rozšířena o synflood modul, který je dále dělen na momentálně tři podmoduly, které se snaží mitigovat TCP SYN flood třemi různými strategiemi. Jelikož tyto útoky vyžadují odlišný přístup pro jejich efektivní odražení, mají i pravidla odlišný formát. Ten je následující:

```
<RULE> <DST-NET> <SRC-PORT> <DST-PORT> <THRESHOLD> <STRATEGY>
<HOST-SYN-SOFT-THRESHOLD> <HOST-SYN-HARD-THRESHOLD> <WHITELIST>
```

Kde jednotlivé položky mají následující význam:

- `RULE`: Nepovinný textový komentář k pravidlu.
- `DST-NET`: Chráněná cílová síť (prefix) ve standardním formátu (např. 147.229.0.0/16). Může být zadán IPv4 i IPv6 prefix. Tato položka je povinná.
- `SRC-PORT`: Zdrojový port paketu. Může být zadán jako menší než, větší než, rozsah (od-do) nebo přesná shoda. Nepovinná položka.
- `DST-PORT`: Cílový port paketu. Může být zadán jako menší než, větší než, rozsah (od-do) nebo přesná shoda. Nepovinná položka.
- `THRESHOLD`: Povinná položka. Udává počet paketů s příznakem SYN za sekundu, po jehož překročení začne Protector mitigovat SYN pakety.
- `STRATEGY`: Povinná položka. Udává typ strategie pro mitigaci TCP SYN flood útoků. Protector podporuje tři odlišné strategie:
  - *SYN-DROP*: Syn Drop (SYN-Drop) strategie.
  - *RST-COOKIE*: Reset cookies (RST-Cookies) strategie.
  - *ACK-SPOOF*: Acknowledge spoofing (ACK-Spoofing) strategie.
- `HOST-SYN-SOFT-THRESHOLD`: Nepovinná položka. Udává relativně nízký počet přijatých SYN paketů za sekundu. Platí pouze pro SYN-Drop strategii.
- `HOST-SYN-HARD-THRESHOLD`: Nepovinná položka. Udává relativně vysoký počet přijatých SYN paketů za sekundu. Platí pouze pro SYN-Drop strategii.
- `WHITELIST`: Nepovinná položka. Udává seznam bezpečných IP adres, od kterých se pakety nebudou zahazovat.

RANGE		SYN			
		1	<2, SOFT>	(SOFT, HARD)	<HARD, ∞)
ACK	0	DROP	ALLOW	DROP	DROP
	0<	ALLOW	ALLOW	ALLOW	DROP

Tabulka 2.1: Rozhodovací tabulka algoritmu SYN-Drop. Podle poměru přijatých SYN a ACK paketů pro každou zdrojovou IP adresu určuje, zdali paket přepošle nebo zahodí [7].

SYN-Drop je strategie založena na poměru mezi SYN a ACK pakety. Pro každou zdrojovou IP adresu uchovává počet samostatných SYN paketů a ACK paketů. Rozhodovací tabulka 2.1 určuje, zdali je paket zahozen nebo přeposlán dál. V této tabulce hodnota `SOFT` odpovídá položce `HOST-SYN-SOFT-THRESHOLD` a hodnota `HARD` položce `HOST-SYN-HARD-THRESHOLD`.

SYN paket je zahozen v případě, že:

- se jedná o první paket dané IP adresy (filtrování podvržených adres),
- počet SYN paketů je větší než relativně nízká hodnota `HOST-SYN-SOFT-THRESHOLD` a zároveň nedorazil žádný ACK paket (IP adresa nekomunikuje správně, možný TCP SYN flood),
- počet SYN paketů je vyšší než relativně vysoká hodnota `HOST-SYN-HARD-THRESHOLD` bez ohledu na počet odeslaných ACK paketů (IP adresa je příliš agresivní).

V ostatních případech je paket propuštěn dále, protože zdrojová IP adresa buďto komunikuje (odeslala nenulový počet ACK paketů), nebo počet SYN paketů je pod hranicí relativně nízké hodnoty `HOST-SYN-SOFT-THRESHOLD`. Čítače SYN a ACK paketů se periodicky resetují.

RST-Cookies je strategie založená na faktu [26], že legitimní zařízení by mělo poslat paket s příznakem RST (reset) v případě, že spojení je v nesesynchronizovaném stavu a od protější strany přijme paket s příznakem ACK, který potvrzuje něco, co ještě nebylo odesláno. Na základě toho je budován seznam legitimních zdrojových IP adres (whitelist), jejichž pakety nebudou znovu ověřovány (po určitou dobu). Modul implementující tuto strategii funguje tak, že se první SYN paket zatím neviděné zdrojové IP adresy zahodí a zároveň odpoví nevalidním SYN+ACK paketem. Pokud dostane odpověď v podobně RST paketu, jedná se o legitimní zařízení a jeho IP adresa je přidána do whitelist seznamu. Pokud zařízení neodpoví vůbec nebo neodpoví správně, jeho SYN pakety se nedostanou k cíli. Podobně jako u SYN-Drop strategie je whitelist seznam perodicky resetován. Velice podrobně se této strategii věnovala bakalářská práce Patrika Goldschmidta [22].

ACK-Spoofing strategie funguje jako firewall/proxy. Pokud zařízení v chráněné síti (server) dostane žádost o navázání TCP spojení – SYN paket, odpoví paketem SYN+ACK a očekává ACK paket. Pokud Protector zachytí SYN+ACK paket od serveru, okamžitě mu odpoví podvrženým ACK paketem. Hlavní smysl této strategie je, aby server mohl spojení označit jako navázané a uvolnil tak místo v limitovaném backlogu. Protector čeká na pravý ACK paket od klienta (iniciátora TCP spojení) po určitou dobu. Pokud ho do té doby nezachytí, pak serveru odešle podvržený RST paket, což u serveru způsobí uzavření spojení a tím uvolnění zdrojů. Implementací této strategie se zabývá bakalářská práce Tomáše Odehnala [44].

Kapitola 3

Analýza generátorů síťového provozu

Tato kapitola se zabývá analýzou dostupných nástrojů pro generování síťového provozu a DoS útoků. První podkapitola 3.1 popisuje rozhraní pro rychlý příjem a generování paketů. Patří sem AF_PACKET, PF_RING a DPDK. Další dvě podkapitoly 3.2 a 3.3 představují volně dostupné stavové generátory provozu Cisco TRex a Juniper WARP17. Podkapitola 3.4 jen okrajově vystihuje dva vybrané zástupce komerčních generátorů provozu. Poslední podkapitola 3.5 velmi stručně uvede programy, které je možné použít jako generátory DoS útoků.

3.1 Rozhraní pro příjem a generování paketů

V linuxových systémech je příjem a odesílání paketů z uživatelské aplikace standardně zpracováno síťovým zásobníkem jádra. Ten byl původně vytvořen s důrazem na obecnou použitelnost a množství funkcí (protokolů), které je schopen zpracovávat. Nebyl tedy navržen s důrazem na co nejvyšší propustnost. Jedno z výrazných zpomalení spočívá v tom, že aplikace (například generátor provozu) běží v uživatelském prostoru, zatímco síťový zásobník v prostoru jádra. Z důvodu bezpečnosti jádra se pakety mezi dvěma prostory kopírují a kopírování dat je poměrně drahá operace. Kromě kopírování dat se při příjmu paketu také vykoná systémové volání, což vyžaduje přepnutí kontextu [21]. Postupem času tak začaly vznikat různá rozhraní, která se toto úzké hrdlo snažila odstranit.

3.1.1 AF_PACKET

AF_PACKET je typ socketu (schránky) v linuxovém systému, který umožňuje příjem a odesílání tzv. *raw* paketů na linkové (L2) vrstvě a obchází tak L3 a L4 vrstvu síťového zásobníku jádra [37]. Je vhodný při testování vlastní implementace protokolů v uživatelském režimu, nebo pro zachytávání paketů, které přicházejí na síťové rozhraní zařízení. Vlastní vytváření raw paketů umožňuje použít například falešnou zdrojovou IP adresu, v případě příjmu zas číst pakety, které jsou určeny pro jiné aplikace (podle čísla portů). Proto aplikace, která využívá socket typu AF_PACKET musí disponovat oprávněním CAP_NET_RAW. Kopírování paketů mezi uživatelskou aplikací a jádrem ve výchozím stavu stále zůstává, avšak linuxové jádro od verze 2.4 poskytuje PACKET_MMAP. PACKET_MMAP je API pro minimalizaci kopírování dat a systémových volání [2]. PACKET_MMAP umožňuje

vytvořit kruhový buffer pro ukládání paketů a ten následně namapovat do uživatelského prostoru aplikace.

Libpcap je známá, otevřená knihovna poskytující C/C++ API pro zachytávání a ukládání příchozích paketů do souborů [59]. Pro zachytávání paketů v linuxových systémech interně využívá právě typ socketu AF_PACKET (resp. PF_PACKET – jsou identické). Existuje její verze i pro operační systém Windows pod názvem WinPcap (již neudržovaná), případně její vylepšená varianta Npcap. Uložené soubory mají typickou koncovku PCAP a jsou hojně používané, protože umožňují zpětné zkoumání příchozího provozu za účelem například bezpečnosti nebo odstraňování problémů s konektivitou. Zároveň mohou být použity pro přehrání zachyceného provozu zpět do sítě. Libpcap využívají programy jako analyzátor paketů Wireshark nebo přehrávač PCAP souborů tcpreplay.

3.1.2 PF_RING

PF_RING firmy ntop je typ socketu, který umožňuje příjem a odesílání paketů na vyšších rychlostech [40]. Na rozdíl od AF_PACKET se nejedná o standardní linuxový typ socketu a do systému se proto musí přidat pomocí zavedení jaderných modulů. V prostoru jádra vytváří kruhový buffer, kam jádro nakopíruje pakety na úrovní L2 vrstvy. Tento buffer je pak namapován uživatelským aplikacím, které s pakety pracují skrze rozhraní socketu. Buffer může využívat několik aplikací současně. Stejně jako v případě AF_PACKET příchozí pakety zároveň dále procházejí protokolovým zásobníkem jádra do uživatelských aplikací, kterým jsou určeny. Jinak by totiž např. spuštění nástroje Wireshark zastavilo síťovou komunikaci již spuštěných aplikací, což se neděje.

Firma ntop dále nabízí vylepšenou variantu PF_RING ZC (Zero Copy). Nad standardními ovladači síťové karty pracuje podobně jako standardní (vanilla) verze PF_RING, ale při použití upravených ovladačů dokáže pakety číst přímo ze síťové karty a kompletně tak obchází jádro systému, tedy včetně vanilla PF_RINGu [41]. Pakety již standardně nejsou doručeny cílovým aplikacím a nevzniká žádné kopírování paketů. Avšak PF_RING ZC na rozdíl od jiných rozhraní, které kompletně obcházejí jádro, umí takto zachycené pakety zaslat „zpět“ do síťového zásobníku jádra pro standardní zpracování [42]. ZC verze dosahuje rychlosti zpracování 1 až 100 Gbps. Upravené ovladače poskytli výrobci jako Intel, Fiberblaze nebo Netcope [39]. Její nevýhoda oproti vanilla verzi je, že je placená – až 249.95 Euro na MAC adresu [43].

3.1.3 DPDK

DPDK (Data Plane Development Kit) je otevřený framework, který poskytuje knihovny a prostředí pro rychlé zpracování paketů [18]. Původně byl vydán firmou Intel v roce 2010 a postupně se rozvíjel s vývojovou podporou od více než 25 firem. V současnosti je spravován organizací Linux Foundation. DPDK vytváří vlastní prostředí s knihovnami skrze abstraktní vrstvu EAL (Environment Abstraction Layer) [19]. Tato vrstva specifikuje například architekturu (i686, x86_64, arm64) nebo překladač. Po vytvoření EAL knihoven je DPDK aplikace může využít pro zpracování paketů. Tímto způsobem DPDK odstiňuje aplikaci od specifických systémů a podporuje její přenositelnost. DPDK podporuje obcházení jádra, k čemuž potřebuje speciální ovladače síťových karet, které ovšem běží v uživatelském režimu a implementují API pro nastavení a komunikaci se síťovým zařízením [20]. Ovladače

pracují výhradně v režimu polling, kdy aktivně kontrolují stav kruhových bufferů a nečekají na přerušení. Dosud byly ovladače poskytnuty více než 15 výrobci síťových karet.

3.2 Cisco TRex

Podkapitola vychází z webových stránek a dokumentace TRex generátoru [11, 58, 57, 56], není-li uvedeno jinak.

TRex je otevřený softwarový generátor realistického provozu pro linuxové systémy (CentOS/RHEL) vyvíjený známým výrobcem síťových zařízení Cisco. Cisco uvádí, že je vhodný pro výkonné testování zařízení provádějící mechanismy jako DPI (Deep packet inspection), NAT, Firewally nebo IPS (Intrusion Prevention System). Mezi hlavní body podporované funkcionality patří:

- DPDK rozhraní o propustnosti od 1 až do 100 Gbps.
- Vysoce škálovatelné generování realistického provozu (stavový i bezstavový).
- Emulace L7 provozu s vlastním plně implementovaným TCP/IP zásobníkem.
- Chytré zpracování PCAP souborů.
- Téměř neomezená tvorba vlastních paketů a možnost modifikace existujících paketů, IPv4 i IPv6.
- Měření latence paketů a rozsáhlé statistiky o generovaném i příchozím provozu, GUI.
- Automatizace díky API v jazyce Python.

Na podporovaných DPDK síťových kartách je schopný generovat velký objem provozu o celkové propustnosti do 100 Gbps, avšak při použití specializovaného systému UCS (Cisco Unified Computing System) výrobce uvádí propustnost až 200-400 Gbps. Realistický provoz se snaží simulovat generováním bezstavového i stavového provozu na základě předem zvolených šablon. Profil provozu je také možné nadefinovat manuálně v jazyce Python pomocí připraveného API.

3.2.1 Bezstavový provoz

Bezstavový provoz je realizován ve formě stream paketů. Stream může nabývat plynulé (continuous), dávkové (burst) nebo vícedávkové (multi-burst) povahy. Profil provozu se definuje v jazyce Python. Při ručním sestavování paketu je možné použít API z nástroje `Scapy`, který je již v TRex balíku standardně obsažen. Scapy poskytuje jednoduchý způsob pro vytvoření jakéhokoliv paketu (včetně nevalidního), kdy se paket definuje postupně po jednotlivých síťových vrstvách L2-L7 hlavičkami jednotlivých protokolů dané vrstvy [1]. TRex dále poskytuje tzv. *Field Engine* pro modifikaci paketů za chodu. To je vhodné například pro zadání velikosti paketu nebo rozsahu zdrojových a cílových IP adres paketů. Přes rozhraní Scapy se vytvoří struktura jediného paketu a TRex následně aplikací programu Field Engine zařídí, aby každý odeslaný paket měl různou IP adresu či velikost v zadaném rozsahu. Kód 3.1 popisuje ukázku tvorby jednoduchého profilu provozu skládajícího se z UDP streamu o 1 paketu. Je vytvořena třída `STLS1`, která definuje strukturu streamu. Její metoda `create_stream()` definuje strukturu paketu skrze Scapy a nastavuje typ streamu na continuous. Další metoda `get_streams()` je pro povinná a zde pouze vrací výsledek

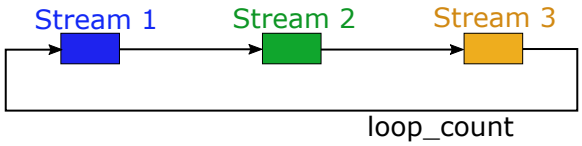
metody `create_stream()`. Profil provozu se nakonec musí zaregistrovat skrze povinnou funkci `register()`. Ta vrací třídu `STLS1`, ve které byl definován zmíněný UDP stream.

```
from trex_stl_lib.api import *                          # TRex API
class STLS1(object):
    def create_stream (self):
        return STLStream(
            packet = STLPktBuilder(
                        # definice struktury paketu skrze rozhrani Scapy
                        pkt = Ether()/IP(src="16.0.0.1",dst="48.0.0.1")/
                              UDP(dport=12,sport=1025)/(10*'x')),
             mode = STLTXCont())                        # tok je typu "Continuous"
    def get_streams (self, direction = 0, **kwargs):  # tato funkce je povinna
        return [ self.create_stream() ]                 # vytvor 1 stream
def register():                           # funkce register() je povinna pro kazdy
    profil provozu
    return STLS1()
```

Výpis 3.1: Ukázka vytvoření jednoduchého profilu provozu – UDP stream o 1 paketu (per stream) bez aplikace Field Engine v jazyce Python [58].

Dále je možné bezstavový provoz vytvořit částečně automatizovaně na základě standardně používaných PCAP souborů. V TRexu je použití PCAP souborů možné dvěma způsoby:

Local PCAP push U Local push metody je PCAP soubor nahrán Python klientem a transformován na seznam streamů, kde každý stream obsahuje pouze **jeden** paket. Počet jednotlivých streamů se pak bude rovnat počtu paketů v PCAP souboru. Každý dokončený stream (paket) spustí odeslání následujícího streamu (paketu). Toto je znázorněno na obrázku 3.1. Metoda podporuje Field Engine pro dodatečnou uživatelem definovanou manipulaci paketů před jejich odesláním. Vzhledem k režii spravování velkého množství streamů (maximální podporovaný limit je kolem 10 až 20 tisíc současných paketu) je velikost PCAP souboru u této metody omezena na 1 MB.



Obrázek 3.1: Vytvořené streamy pro PCAP soubor obsahující 3 pakety. Každý stream odkazuje na další, čímž ho spustí [58].

Server-based push Server-based push metoda pouze načte PCAP soubor a postupně odesílá pakety ze souboru bez zvláštní dodatečné režie. Na rozdíl od Local push metody není velikost PCAP souboru omezena, ale také nepodporuje Field Engine pro dodatečnou modifikaci paketů, pak jakékoliv úpravy budou muset být provedeny do PCAP souboru předem. Tato metoda více odpovídá klasickému přehrávání PCAP souborů v programech jako `tcpreplay`.

3.2.2 Stavový provoz

Stavový provoz TRex rozděluje na tzv. *Stateful* a *Advanced Stateful* režim.

Stateful Základní Stateful režim byl první stavový režim, který TRex implementoval a i přes svůj název není skutečně plně stavový, tedy ve smyslu, že nepoužívá úplnou implementaci TCP/IP zásobníku. Určité stavovosti však dosahuje díky předzpracování a chytrému přehrávání předem vytvořených šablon provozu. Šablona provozu se definuje v souboru ve formátu YAML. Ukázka jednoduché šablony je popsána (včetně komentářů k jednotlivým řádkům) v kódu 3.2, kde je definován zejména rozsah IP adres klienta a serveru a cesta k PCAP soubor s pakety. TRex na základě této konfigurace dokáže *chytře* přehrát obsah PCAP souboru podle zadaných pravidel. Například i při pouhé výměně IP adres musí TRex udržovat, které IP adresy paketů z PCAP souboru odpovídají nově přiřazeným IP adresám z rozsahu zadaného uživatelem, aby dokázal správně přehrát komunikaci mezi různými klienty a servery, když komunikace probíhá obousměrně. S tím souvisí i schopnost TRexe podporovat NAT – překlad adres. Pokud DUT (Device Under test, testované zařízení) provádí překlad adres a TRexový server příjme paket od jiné IP adresy či portu, než z jaké ji TRexový klient odeslal, dokáže se na to za běhu adaptovat. Pokud uživatel chce testovat stavové TCP spojení mezi klientem a serverem, PCAP soubor musí toto ustanovení, udržení a nakonec ukončení spojení obsahovat. TRex totiž, jak již z výše uvedeného popisu vyplývá, pouze chytře přehrává obsah PCAP souboru, ale sám negeneruje dodatečné pakety. Protože Stateful režim neobsahuje plný TCP/IP zásobník, nedokáže se správně vypořádat například se situací, kdy DUT vynutí ukončení TCP spojení.

```
- duration : 10.0                          # Delka testu. Lze prepsat parametrem -d
    prikazove radky.
  generator :
          distribution : "seq"             # IP adresy ze zadaneho rozsahu prirazuj
    postupne v sekvencnim poradi
          clients_start : "16.0.0.1"       # Rozsah IPv4 adres klientu (celkove 254)
          clients_end   : "16.0.0.254"
          servers_start : "48.0.0.1"       # Rozsah IPv4 adres serveru (celkove 254)
          servers_end   : "48.0.0.254"
  cap_info :
     - name: cap2/dns.pcap # Obsahuje pouze 2 pakety: DNS Query od klienta and DNS
    Response od serveru
        cps : 1.0           # Generovani 1 spojeni za sekundu
        ipg : 10000         # Mezipaketova mezera v mikrosekundach (Inter-packet gap)
```

Výpis 3.2: Ukázka konfigurace v YAML souboru [57].

Advanced Stateful Advanced Stateful režim byl do TRexe přidán až později (2017, stále ve vývoji) a oproti základnímu Stateful režimu už implementuje svůj vlastní TCP/IP zásobník pracující v uživatelském prostoru (kompletně obchází TCP/IP zásobník v linuxovém jádře) a dá se tak považovat za plně stavový[54]. Profil provozu se definuje v jazyce Python, stejně jako u bezstavového provozu. Pokud je použit PCAP soubor, TRex ho dokáže analyzovat a vytvořit z něho interní reprezentaci skládající se z jednotlivých TCP operací. V průběhu analýzy je dále schopen odvodit některé TCP parametry jako velikost okna nebo MSS (Maximum segment size). Profil provozu je také možné nadefinovat manuálně. K tomu TRex poskytuje poměrně bohaté API, které poskytuje jak podporu pro L4 vrstvu (navázání TCP spojení na straně klienta, přijetí TCP spojení na straně serveru, vynucený reset TCP spojení...) tak i L7 vrstvu zejména pro protokol HTTP (request, response). Krátký úryvek s využitím tohoto

API je popsán ve výpisu 3.3. Jedná se o komunikaci klienta a serveru vyměňující si zprávy HTTP request (od klienta) a HTTP response (od serveru).

```
http_req = b'GET /3384 HTTP/1.1\r\n...'
http_response = 'HTTP/1.1 200 OK\r\n...'

# prikazy pro klienta
prog_c = ASTFProgram()            # vytvor emulaci L7 programu
prog_c.send(http_req)             # posli serveru HTTP request, implicitni navazani
    TCP spojeni se serverem (metoda send=TCP; metoda send_msg=UDP)
prog_c.recv(len(http_response))   # prijmi odpoved od serveru
# implicitni  uzavreni spojeni

# prikazy pro server
prog_s = ASTFProgram()
prog_s.recv(len(http_req))        # prijmi HTTP request od klienta,
prog_s.delay_rand(100,1000)       # simuluj nahodne zpozdeni odpovedi v mikrosekundach
prog_s.send(http_response)        # posli HTTP response
prog_s.wait_for_peer_close();     # cekej na ukonceni spojeni klientem
```

Výpis 3.3: Úryvek Python kódu ruční definice HTTP provozu [56].

Podporovaný operační systém je CentOS/RHEL 7.6, ale měl by fungovat i na dalších linuxových systémech, pokud mají zkompilované potřebné ovladače. Z uživatelského pohledu je dokumentace TRexe poměrně bohatá a dobře čitelná. Instalace je podle dokumentace velmi jednoduchá, jedná se totiž o stažení a rozbalení jediného archivu. V něm se celý systém nachází již zkompilovaný, není tedy potřeba řešit různé závislosti. Před prvotním spuštěním je nutné nastavit konfigurační soubor informující o seznamu rozhraní (interfaces), které budou použity jako porty pro klienta a server. V případě DPDK kompatibilní síťové karty se nastaví systémové PCI adresy fyzických portů karty. V archivu je přibalen nástroj `dpdk_setup_ports.py` pro přehledný výpis připojených síťových zařízení. Nástroj kromě výpisu umí i konfigurační soubor vygenerovat podle interaktivních vstupů od uživatele. Není-li v systému DPDK kompatibilní síťová karta, je možné použít standardní linuxový síťový interface – tedy jak běžnou fyzickou síťovou kartu (`eth0`), tak i čistě virtuální interface (`veth0`). Zajímavý je fakt, že konfigurace umožňuje povolit tzv. *low end* režim pro nevýkonné stroje. Tato volba podporuje vyzkoušení TRexe i na vlastních nevýkonných strojích, například ve virtuálním prostředí. Po úspěšné konfiguraci lze TRex spustit z příkazové řádky s parametry a cestou k šabloně provozu. Zde je určitě na místě ocenit velké množství předpřipravených šablon (60 YAML souborů a 30 PCAP souborů jen pro Stateful režim).

V průběhu generování provozu je možné sledovat různé statistiky. Samozřejmostí je počet odeslaných a přijatých paketů/bytů pro každý port a z nich vypočtenou jeho celkovou propustnost. Dále zde jsou zahrnuty čítače o chybách při odeslání/příjmu paketů nebo o vytížení procesoru. Posledním zajímavým ukazatelem je celkový počet toků (flows) a počet *aktivních* toků. V případě advanced stateful provozu (TCP) se jedná o celkový počet navázaných spojení a počet momentálně *aktivních* spojení. Na začátku testu se počet aktivních a celkových TCP spojení bude rovnat. Po určité době budou aktivní spojení ukončeny a jejich počet se v průběhu testu ustálí kolem určité hodnoty (za předpokladu, kdy délka každého spojení je přibližně stejná), zatímco celkový počet TCP spojení bude pouze růst. Po uplynutí časového limitu testu pak TRex přestane generovat nová spojení, ale zároveň musí počkat, než stále aktivní spojení nebudou ukončena. Po ukončení testu je v případě advanced stateful režimu vypsána dodatečná statistika obsahující položky jako počet po-

kusů o navázání spojení (klient), přijaté požadavky o navázání spojení (server), odeslané kontrolní TCP pakety (příznaky SYN, FIN, RST), pakety s TCP příznakem ACK nebo pakety přijaté v jiném pořadí (out of order). Celá ukázka takových statistik je uvedena ve výpisu 3.4.

```
                  |     client   |        server |
-------------------------------------------------------------------------------
  m_active_flows  |      39965   |        39966  |  active flows
     m_est_flows  |      39950   |        39952  |  active est flows
    m_tx_bw_l7_r  | 31.14 Mbps   |    4.09 Gbps  |  tx bw
    m_rx_bw_l7_r  |  4.09 Gbps   |   31.14 Mbps  |  rx bw
      m_tx_pps_r  |140.36 Kpps   |  124.82 Kpps  |  tx pps
      m_rx_pps_r  |156.05 Kpps   |  155.87 Kpps  |  rx pps
      m_avg_size  |    1.74 KB   |     1.84 KB   |  average pkt size
                -  |        ---   |          ---  |
              TCP  |        ---   |          ---  |
                -  |        ---   |          ---  |
tcps_connattempt  |      73936   |            0  |  connections initiated
    tcps_accepts  |          0   |        73924  |  connections accepted
   tcps_connects  |      73921   |        73910  |  connections established
     tcps_closed  |      33971   |        33958  |  conn. closed (includes drops)
  tcps_segstimed  |     213451   |       558085  |  segs where we tried to get rtt
 tcps_rttupdated  |     213416   |       549736  |  times we succeeded
     tcps_delack  |     344742   |            0  |  delayed acks sent
    tcps_sndtotal |     623780   |       558085  |  total packets sent
     tcps_sndpack |      73921   |       418569  |  data packets sent
     tcps_sndbyte |   18406329   |   2270136936  |  data bytes sent
     tcps_sndctrl |      73936   |            0  |  control (SYN,FIN,RST) packets sent
     tcps_sndacks |     475923   |       139516  |  ack-only packets sent
     tcps_rcvpack |     550465   |       139502  |  packets received in sequence
     tcps_rcvbyte | 2269941776   |     18403590  |  bytes received in sequence
  tcps_rcvackpack |     139495   |       549736  |  rcvd ack packets
  tcps_rcvackbyte |   18468679   |   2222057965  |  tx bytes acked by rcvd acks
    tcps_preddat  |     410970   |            0  |  times hdr predict ok for data pkts
   tcps_rcvoopack |          0   |            0  |  *out-of-order packets received
                -  |        ---   |          ---  |
       Flow Table  |        ---   |          ---  |
                -  |        ---   |          ---  |
   redirect_rx_ok  |          0   |            1  |  redirect to rx OK
```

Výpis 3.4: Ukázka statistik dostupných po ukončení TRex testu využívající advanced stateful režim [56].

3.3 Juniper WARP17

WARP17 je otevřený softwarový generátor stavového provozu pro linuxové systémy (Ubuntu) vyvíjený dalším výrobcem síťových zařízení Juniper. Stejně jako TRex podporuje DPDK a je zaměřen primárně na plně stavový L5-L7 provoz (HTTP) běžící nad TCP. Obsahuje vlastní, plně implementovaný TCP/IP zásobník pracující v uživatelském prostoru [32]. Umožňuje generovat i provoz běžící nad UDP, avšak v omezené formě. Dle výrobce dosahuje propustnosti 40 Gbps, což je méně než TRex, avšak dle popisu použitého referenčního stroje byly použity pouze dvě 40 Gbps síťové karty Intel XL710-QDA1 [30]. Maximální propustnost na výkonnějších kartách tak může být i vyšší. Z pohledu výkonnosti navazování a okamžitého ukončování TCP spojení je schopen generovat 17 M spojení každou sekundu.

Profil provozu se definuje pouze ručně specifickými příkazy pro interpret WARP17. Neexistuje tudíž podpora pro generování provozu na základě PCAP souborů. V profilu provozu se definují se dvě strany – klient a server. Oběma se přiřadí jejich IP adresy. Zde platí dvě omezení – první je podpora pouze IPv4 adres a druhá je limit pouze 10 IP adres na fyzický port. V případě loopback zapojení na dvouportové síťové kartě pak nelze využít více jak 20 IP adres v průběhu testu. Dále se definuje L4 protokol (TCP/UDP) a zdrojové/cílové porty (pro server pouze zdrojový port, na kterém bude naslouchat). Díky implementaci vlastního TCP zásobníku umožňuje konfigurovat některé jeho parametry, jako například velikost okna, počet pokusů o znovuodeslání paketů a různé timeout hodnoty. Ve výchozím stavu bude obsah L5-L7 vrstev náhodný (tzv. *RAW TCP*). WARP17 také obsahuje příkazy pro podporu L7 protokolu HTTP verze 1.1 s požadavky GET/HEAD od klienta a odpověďmi 200 OK/404 NOT FOUND od serveru. V obou případech je nutné určit velikost L5-L7 obsahu (payload).

Pro průběh testu se definují 3 typy časů:

- *initial_delay* – určuje, jak dlouho (v sekundách) klient bude čekat, než se pokusí navázat spojení se serverem.
- *conn_uptime* – určuje, jak dlouho klient bude udržovat spojení se serverem a posílat data.
- *conn_downtime* – určuje, jak dlouho bude klient po ukončení spojení čekat, než se pokusí o znovunavázání spojení se serverem.

Z popisu těchto čítačů vyplývá, že provoz mezi konkrétním klientem a serverem se periodicky opakuje. Výpis 3.5 obsahuje ukázku konfigurace periodické TCP komunikace mezi jedním klientem (10.0.0.1:10000) a serverem (10.0.0.253:6000). Požadavek klienta má velikost 100 B, odpověď serveru 200 B a obsah těchto paketů je náhodný. Klient a server komunikují 5 sekund, poté spojení ukončí. Po 15 sekundách je komunikace zopakována.

```
# CLIENT port configuration - Add the L3 interface on the client side
add tests l3_intf port 0 ip 10.0.0.1 mask 255.255.255.0

# Configure a TCP client test case (ID 0) on PORT 0 from 10.0.0.1:10000 to
    10.0.0.253:6000
# src 10.0.0.1 10.0.0.1 means source IP range from 10.0.0.1 to 10.0.0.1 = only 1 IP
    address
add tests client tcp port 0 test-case-id 0 src 10.0.0.1 10.0.0.1 sport 10000 10000
    dest 10.0.0.253 10.0.0.253 dport 6000 6000

# Configure the RAW application values for test case ID 0 on PORT 0 - request-size
    100, response-size 200:
set tests client raw port 0 test-case-id 0 data-req-plen 100 data-resp-plen 200
################################################################################
# SERVER port configuration
add tests l3_intf port 1 ip 10.0.0.253 mask 255.255.255.0

# Configure a TCP server test case (ID 0) on PORT 1 accepting connections on
    10.0.0.253:6000
add tests server tcp port 1 test-case-id 0 src 10.0.0.253 10.0.0.253 sport 6000 6000

# Configure the RAW application values for test case ID 0 on PORT 1:
set tests server raw port 1 test-case-id 0 data-req-plen 100 data-resp-plen 200
################################################################################
# Configure the timeout profile for test case ID 0 on PORT 0:
set tests timeouts port 0 test-case-id 0 init 0        # start immediately
set tests timeouts port 0 test-case-id 0 uptime 5
set tests timeouts port 0 test-case-id 0 downtime 15
```

Výpis 3.5: Ukázka jednoduché konfigurace testu s *RAW TCP* příkazy WARP17 [31].

Podporovaný operační systém je Ubuntu Server 16. Fungovat by měl i na dalších linuxových systémech, avšak můžou se vyskytnout potíže s různými závislostmi balíčků a instalací vzhledem k tomu, že instalační skripty napevno pracují s balíčkovacím systémem `apt` (a balíčky typu `.deb`) [29]. CentOS/RHEL (který je naopak nativně podporovaný TRexem) typicky využívá balíčkovací systém `yum` a balíčky typu `.rpm`. Uživatelům na CentOS/RHEL pak instalační skripty nepůjdou bez dodatečné modifikace správně spustit. Dokumentace je vcelku dostatečná, ale některé části jsou zpracovány hůře a některé lépe. Před spuštěním je nutné nastavit rozhraní, které budou použity jako porty pro klienta a server. Na rozdíl od TRexe se toto nastavuje přímo přes nástroj samotného DPDK (které je spolu s WARP17 instalované samostatně) `dpdksetup.sh`. To znamená nutnost dodatečného nastudování relevantních materiálů o platformě DPDK. Dále existuje podpora pro linuxový síťový interface a tzv. *InMemoryRingBased* interface, který je čistě virtuální. Po úspěšné konfiguraci lze WARP17 spustit z příkazové řádky s parametry a cestou k souboru obsahující příkazy pro generování provozu. Výchozí instalace obsahuje 15 příkladů pro generování provozu. V průběhu a po ukončení generování provozu je možné sledovat statistiky, které jsou podobné těm z TRexe.

3.4 Komerční nástroje pro generování síťového provozu

V této podkapitole jsou velmi stručně uvedeny dva zástupci komerčních nástrojů – Spirent TestCenter a Ixia IxNetwork/IxLoad. Jedná se typicky o řešení složené z hardware a software, tudíž cena takových řešení, zejména u vysokých propustností může být až v řádu statisíc dolarů.

3.4.1 Spirent TestCenter

Spirent TestCenter je flexibilní platforma pro funkční i výkonnostní testování síťových zařízení na vrstvách L2 až L7. Je složen z hardwarové a softwarové části. Podle typů nainstalovaných hardwarových modulů je schopen generovat a přijímat pakety o rychlosti od 10 Mbps až 400 Gbps [47]. Také poskytuje modul pro testování bezdrátových sítí Wi-Fi. Ovládání a vytvoření testů je možné manuálně skrze grafické uživatelské rozhraní aplikace nebo automatizovaně skrze API (Tcl, C, Java, Python, REST, ...) [48, 50]. Kromě generování bezstavového provozu poskytuje funkcionalitu i pro generování stavového provozu [49]. Obsahuje vlastní TCP/IP zásobník a podporu pro L7 protokoly jako HTTP(S), FTP, POP3 či SMTP. Funkcionalitu platformy Spirent TestCenter lze rozšiřovat ať už HW moduly nebo SW balíčky, ale celková cena výsledného řešení bude výrazně růst.

3.4.2 Ixia IxNetwork a IxLoad

Ixia podobně jako Spirent nabízí modulární řešení pro funkční i výkonnostní testování síťových zařízení na vrstvách L2 až L7. IxNetwork je řešení zabývající se primárně vrstvou L2/3 [28]. Slouží tedy hlavně pro testování zařízení jako přepínače nebo směrovače a jejich protokoly. Je schopné generovat pakety o rychlosti od 1 Gbps do 400 Gbps. IxLoad je software pro generování bezstavového i stavového a realistického provozu na vrstvě L4-L7 [27]. Při provozování na doporučeném hardware dokáže generovat provoz od 1 Gbps do 100 Gbps. Podporuje emulaci bohatého množství protokolů jako je HTTP, SSL, TLS, FTP, MySQL nebo Adobe Flash player. Ovládání je možné skrze GUI nebo API (Tcl, Python, Perl, REST). Ixia ve svých materiálech k produktu IxLoad uvádí, že se jedná o jediné řešení na trhu, které dokáže modelovat dynamickou povahu chování uživatele na internetu [27].

3.5 Generátory DoS útoků

Tato podkapitola popisuje tři programy, které umožňují generovat různé typy DoS útoky. Některé z nich lze najít v Kali Linuxu, což je distribuce speciálně určena pro testování různých aspektů bezpečnosti a etického hackerství.

3.5.1 hping3

je program umožňující analýzu a vytváření TCP/IP paketů z prostředí příkazové řádky [33]. Již z názvu je patrné, že je inspirovaný standardním nástrojem `ping`, ale poskytuje mnohem více funkcionality. Podporuje protokoly TCP, UDP, ICMP a RAW-IP (bez protokolu vyšší vrstvy). Obecně umožňuje měnit některá políčka v hlavičkách protokolů. U TCP jsou to např. zdrojový a cílový portu, TCP příznaky nebo datový offset (na nevalidní hodnotu). U IP protokolu jsou to zdrojová adresa, TTL nebo příznaky pro fragmentaci. Samozřejmostí je záplavový režim, kdy pakety odesílá co nejrychleji může, bez čekání na odpověď. Program `hping3` by bylo možné použít pro generování UDP flood útoků i SYN flood útoků.

3.5.2 SlowHTTPTest

je program umožňující generování DoS útoků na aplikační vrstvě z prostředí příkazové řádky [34]. Zaměřuje se výhradně na low-rate DoS útoky typu Slowloris. Dále podporuje útoky R-U-Dead-Yet and Slow Read, jejichž cílem je vyčerpat limit současných HTTP

spojení. Program `SlowHTTPTest` umožňuje nastavit počet současných spojení, periodický interval pro komunikaci se serverem nebo délku testu.

3.5.3 pfsend

je součástí rozhraní PF_RING (popsané v podkapitole 3.1.2) jako ukázkový program. Program `pfsend` umí přehrávat obsah PCAP souborů a generovat pakety s náhodným (nedefinovaným) obsahem. Bylo by možné realizovat útoky UDP flood nebo TCP SYN flood při vhodně zvoleném PCAP souboru. Výhoda oproti nástroji `hping3` spočívá ve vyšší propustnosti díky využití rychlejšího rozhraní pro generování paketů.

Kapitola 4

Návrh prostředí

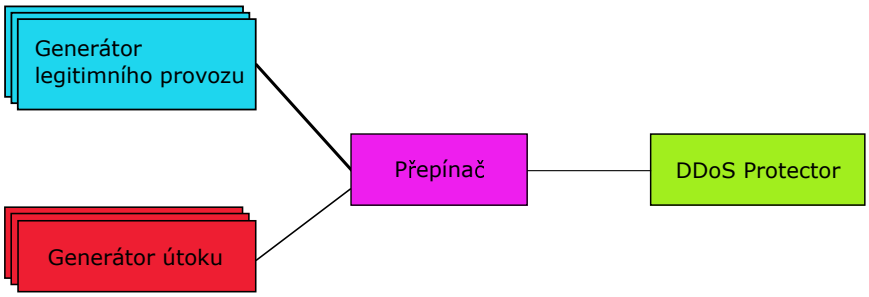
Cílem je návrh prostředí, které umožňuje generování legitimního i závadného síťového provozu za účelem automatizovaného testování zařízení DDoS Protector. Legitimní provoz by se měl podobat reálnému provozu na síti, ve smyslu kombinace různých stavových (TCP) i nestavových (UDP) spojení nebo komunikace na protokolech jako HTTP, DNS, DHCP či ICMP. Závadný síťový provoz bude nabývat podoby různých typů (D)DoS útoků. Prostředí by mělo umožňovat komplexní testování zařízení DDoS Protector, zejména vyhodnocování úspěšnosti mitigace současně různých druhů DDoS útoků, propouštění (nezahazování) legitimního provozu a výkonnost zařízení. Dosavadní vývojové testování Protectoru se soustředilo na postupné ověřování konkrétních funkcí/modulů, jako například korektní funkce hardwarových filtrů nebo mitigace DDoS útoku – nejdříve amplifikační modulu a poté synflood modulu. Zde byl Protector a generátor paketů – základní verze Spirent TestCenter zapojen přímo. Ve Spirentu se nastavily vlastnosti paketů (protokol, zdrojový port atd.) a spustilo s generování o zadaném objemu. Následovala ruční kontrola statistik Spirentu, zdali je objem přijatých paketů nižší než objem vygenerovaný paketů a zdali tento objem odpovídá očekávání dle použitých mitigačních pravidel Protectoru. Automatizaci některých testů implementovala bakalářská práce Matúše Burzaly [3]. Stále se ovšem jedná o testy ověřující jednu konkrétní funkci/modul v daném čase, bez možnosti generování stavového provozu. To je dáno základní verzí Spirenta, která nepodporuje stavové prokoly a protokoly vyšších vrstev. Pokročilé verze Spirenta zmíněné protokoly podporují (podkapitola 3.4.1), ale cena takové licence (a případně nového hardware) je příliš vysoká. Proto je třeba najít jiné řešení. Navrhované prostředí by mělo rozšířit současné testy o možnosti jako generování multi-vector útoků zkombinovaných s realisticky generovaným provozem pro lepší simulaci reálných scénářů. To znamená například současné použití amplifikačního a synflood modulu a vyhodnocení interakce několika různých pravidel z pohledu mitigace jednotlivých typů útoku a jejich dopad na účastníky legitimního provozu.

4.1 Způsob zapojení

Z pohledu zapojení budou nutná minimálně tři zařízení – DDoS Protector, generátor(y) legitimního provozu a generátor(y) DDoS útoků. Pro ověření úspěšnosti mitigace útoků bude třeba přístup k čítačům přijatých paketů/bajtů na rozhraní cílového zařízení. Popsané generátory provozu TRex a WARP17 tyto základní čítače obsahují a také disponují dalšími čítači spojenými s typem generovaného provozu, jako třeba počet pokusů o navázání TCP spojení ze strany klienta nebo počet přijatých pokusů o navázání TCP spojení na straně serveru.

Proto jako vhodné cílové zařízení se jeví server emulovaný v generátoru provozu – obsahuje totiž všechny potřebné čítače pro vyhodnocení testů. Klient a server se vždy nacházejí na odlišných síťových rozhraních (ať už fyzických na síťové kartě nebo čistě virtuálních) a mají samozřejmě vlastní MAC a IP adresy. TRex navíc v plně stavovém režimu podporuje běh klienta a serveru na dvou odlišných fyzických zařízeních. Tento server tak bude cílem generátoru útoku, zatímco Protector se ho bude snažit chránit.

Varianta zapojení s přepínačem je ilustrována na obrázku 4.1. Pokud jsou generátory dvě různá fyzická zařízení, je k jejich propojení s Protectorem nutný přepínač. Úzkým hrdlem zde může být propustnost přepínače, jelikož 100 Gbps přepínač pro testování maximální propustnosti Protectoru není levný. Odchozí pakety z generátorů provozu a útoku budou muset být nastaveny na MAC adresu Protectoru. Pokud jsou zařízení propojená přes přepínač, pak také v principu nic nebrání v přidání dalších fyzických zařízení, které mohou plnit funkci buďto generátoru útoku (silnější a potenciálně komplexnější útoky) nebo generátoru provozu. Takové zapojení umožňuje snadnější rozšiřitelnost prostředí v budoucnosti.



Obrázek 4.1: Návrh zapojení DDoS Protectoru s přepínačem.

V úvahu připadá ještě druhá, jednodušší varianta zapojení bez přepínače. Protector je přímo propojen se zařízením, na kterém běží jak generátor provozu, tak generátor útoku. Například TRex umožňuje spustit více paralelních instancí, kdy jedna instance může generovat provoz a druhá útok. Hrozí ovšem maximální vytížení procesoru stroje před dosažením maximální propustnosti. Při použití různých softwarových programů by se mohl vyskytnout problém přístupu ke sdílenému rozhraní. Při použití linuxových síťových rozhraní je přístup sdílený, avšak byl by naprosto nedostatečný pro jakékoliv výkonnostní testování. Jako lepší řešení jeví varianta zapojení s přepínačem.

4.2 Volba generátoru provozu a generátoru útoku

Volba generátoru provozu je celkem jednoznačná ve prospěch TRexe. Oproti komerčním generátorům má samozřejmě obrovskou výhodu v bezplatném použití. Oproti generátoru WARP17 má výrazně lepší dokumentaci a funkcionalitu, je aktivně vyvíjen, má lepší podporu pro řešení problémů a uživatelská práce s ním je mnohem pohodlnější.

V bezstavovém provozu umožňuje generovat jakýkoliv stream paketů díky tvorbě vlastních paketů v nástroji Scapy. To lze vhodně využít i pro tvorbu volumetrických útoků jako obecný UDP flood, DNS (amplifikační) flood, NTP flood či ICMP flood ale i pro tvorbu TCP SYN flood útoku. S ohledem na jeho vysokou propustnost až 100 Gbps s DPDK rozhraním se tak zároveň stává i vhodnou volbou pro generátor DDoS útoků. Umožňuje využít PCAP soubory, v plně stavovém režimu poskytuje možnost vytvoření šablony vlastního profilu provozu v jazyce Python a obsahuje bohaté množství čítačů pro vyhodnocení

testů. Nakonec poskytuje velké množství již předdefinovaných šablon realistického provozu a Python API pro automatizaci jeho ovládání. Použití TRexe pro útoky na aplikační vrstvě není jednoznačné. Útok typu HTTP flood by měl být realizovatelný bez většího úsilí. Jedná se totiž o posílání paketů se stále stejnou HTTP hlavičkou. Stavové TCP spojení řeší TRex automaticky skrze dodané API. Low-rate útoky jako Slowloris a R-U-Dead-Yet by nejspíše měly být technicky realizovatelné kvůli metodě `send_chunk()` [55]. Ta buffer rozdělí na menší části, které postupně odesílá a mezi každou část vloží časové zpoždění. Dobrá implementace by však pravděpodobně byla náročná a navíc útok Slow Read by nebylo možné realizovat, jelikož TRex nenabízí metodu pro pomalé čtení. U low-rate útoků se mi tudíž jeví výhodnější použít program `SlowHTTPTest`, který by běžel na vyhrazeném zařízení. Při navržené variantě zapojení s přepínačem není složité přidat další zařízení.

4.3 Framework pro testování zařízení DDoS Protector

Framework pro automatizované testování Protectoru byl implementován bakalářskou prací Matúše Burzaly [3], ze které tato podkapitola čerpá. Navržený systém je aktivně používán a dále rozvíjen. Jeden z hlavních cílů systémů byla rozšiřitelnost a proto se implementace této práce jako rozšíření stávajícího frameworku jeví jako ideální volba. Veškeré testy Protectoru budou sjednocené v jediném testovacím systému. Rozšiřitelnost použitého frameworku znamená rozšiřitelnost vytvořených testů i možnost vytvoření nových testů v budoucnosti. Další velká výhoda současného frameworku spočívá v použití jazyka Python, jelikož vybraný generátor TRex poskytuje aplikační rozhraní také v jazyce Python.

Framework běží na stejném stroji jako Protector, aby s ním mohl komunikovat a ovládat ho skrze dostupné nástroje pro potřeby testů. Oba TRex generátory pak musí být ovládány vzdáleně příkazy poslanými po síti. TRex takové ovládání podporuje (více v podkapitole 5.2). Ve frameworku jsou jednotlivé testy implementované jako třídy, které dědí ze základní třídy StcTest (SpirentTestCenter Test). Vytvoření nových testů využívající TRex bude implementovat podobnou hierarchii, kdy se nejdříve vytvoří základní třída TRexTest, z níž poté budou dědit jednotlivé třídy implementující jednotlivé testy. Testy jsou logicky rozdělené do tří fází:

- *Fáze před začátkem testování*– Příprava testovacího prostředí. Jedná se například o inicializaci Protectoru nebo navázání spojení se Spirentem/ TRexem.
- *Fáze testování*– Provádění jednotlivých testovacích případů. Tato fáze je dále dělena do tří podfází. Patří sem například nahrání pravidel do databáze Protectoru, definice provozu a útoku, spuštění TRex generátorů a vyhodnocení testovacích případů.
- *Fáze po skončení testování*– Ukončení spojení s generátory, uvolnění zdrojů.

4.4 Návrh testovacích případů

V podkapitole 4.1 bylo navrženo zapojení do přepínače. Každé zařízení, resp. síťové rozhraní musí mít přiřazenou IP adresu. V návrhu se bude předpokládat zapojení do lokální sítě a bude využit rozsah privátní IPv4 sítě 10.0.0.0/8. IP adresy klienta budou spadat do podsítě 10.0.0.0/24 (rozsah adres 10.0.0.1 - 10.0.0.254). IP adresy serveru budou spadat do podsítě 10.0.1.0/26 (rozsah adres 10.0.1.1 - 10.0.1.62). Tato volba umožní v testech simulovat až 254 klientů a 62 serverů. Počet serverů není příliš důležitý. Z pohledu TRexe všechny servery sdílí jedno fyzické síťové rozhraní a čítače všech serverů jsou agregovány do jednoho.

Využití více serverů ale lépe simuluje scénář, kdy se v chráněné síti vyskytuje více různých serverů (například HTTP server, FTP server, DNS server...). Agregace statistik platí i pro klienta, ovšem na rozdíl od serveru může Protector blokovat i provoz některých klientů. Proto je vhodné mít určitý minimální počet klientů, aby zjišťování procenta zablokovaných klientů/útočníků bylo dostatečně jemné. IP adresy útočníka budou ve výchozím stavu spadat do podsítě 10.0.100.0/24 (rozsah adres 10.0.100.1 - 10.0.100.254). Počet útočníku je pak shodný s počtem klientů, což je výhodné z pohledu vyhodnocování testů. V určitých případech může být rozsah IP adres útočníka navýšen do rozsahu 10.0.100.1 - 10.0.199.254, tedy až 25598 adres. Připomeňme, že Protector je schopný blokovat maximálně 32768 adres. V případě budoucího zvětšení blokovací kapacity ovšem není problém dále rozšířit adresový prostor pro útočníky a umožnit generování útoku z výrazně většího počtu zdrojů. Například RST-Cookies strategie si uchovává seznam legitimních hostů, ale neuchovává žádný seznam útočníků, proto dokáže odolávat útoku i z velkého množství IP adres. IP adresa Protectoru bude nastavena na 10.0.42.1. Je především důležité znát MAC adresu Protectoru, aby všechny generované pakety klienta, serveru i útočníka procházely přes Protector. Protector provádí jednoduché směrování, kdy na základě cílové IP adresy přepíše cílovou MAC adresu propuštěného paketu. Záznamy do směrovací tabulky se musí zadat staticky.

TRex i Protector podporují IPv6. IPv6 varianty podsítí budou využívat stejný počet hostů. Klient bude spadat do podsítě 2001:db8::/122 (2001:db8::1 - 2001:db8::fe), server do podsítě 2001:db8::100/120 (2001:db8::101 - 2001:db8::1fe) a útočník do podsítí 2001:db8::6400/120 (2001:db8::6401 - 2001:db8::64fe) či 2001:db8::6400/112 (2001:db8::6401 - 2001:db8::c7fe). Celá podpora IPv6 ale není hlavní náplní těchto testů, proto ji budou využívat jen některé testy. Rekapitulace přiřazení IP adres je uvedena v tabulkách 4.1 a 4.2.

	IPv4
Klienti	10.0.0.0/24 (10.0.0.1 - 10.0.0.254)
Servery	10.0.1.0/26 (10.0.1.1 - 10.0.1.62)
Útočníci	10.0.100.0/24 (10.0.100.1 - 10.0.100.254) 10.0.0.0/16 (10.0.100.1 - 10.0.199.254)
DDoS Protector	10.0.42.1

Tabulka 4.1: Tabulka obsahující přehled přiřazení IPv4 adres.

	IPv6
Klienti	2001:db8::/122 (2001:db8::1 - 2001:db8::fe)
Servery	2001:db8::100/120 (2001:db8::101 - 2001:db8::1fe)
Útočníci	2001:db8::6400/120 (2001:db8::6401 - 2001:db8::64fe) 2001:db8::6400/112 (2001:db8::6401 - 2001:db8::C7fe)
DDoS Protector	2001:db8::4201

Tabulka 4.2: Tabulka obsahující přehled přiřazení IPv6 adres.

V testech se předpokládá rovnoměrné zastoupení IP adres ve vygenerovaných paketech. Pokud je například vygenerováno 254000 klientských paketů, předpokládá se, že každý klient vygeneroval 1000 paketů. Dále se předpokládá současný start generátoru provozu i generátoru útoku. Jednotlivé testovací případy by měly Protector otestovat v různých kombinacích typů útoků a typů legitimního provozu. Provoz může být buď nestavový (UDP) nebo stavový (TCP). Následuje seznam navržených testovacích případů.

4.4.1 Testovací případ 1: UDP flood [1:5]

Testovací případ je schématicky znázorněn na obrázku 4.2. Protector aplikuje strategii Top N, kdy jednotlivé IP adresy seřadí podle počtu vygenerovaného objemu (či paketů) a ty nejvíce aktivní blokuje, dokud nedosáhne stanoveného limitu. Označení 1:5 zde symbolizuje poměr mezi objemem útoku a objemem klientského provozu.

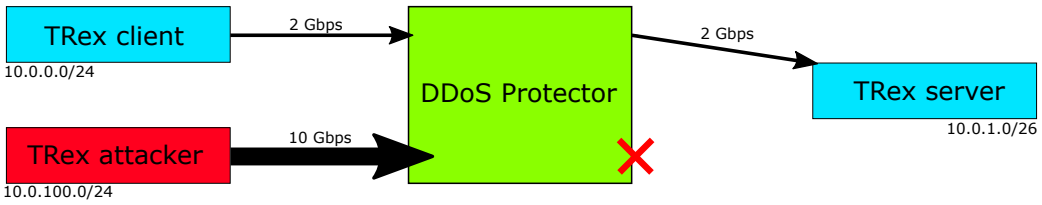
Parametry testu

- **Typ útoku:** UDP flood
- **Síla útoku:** 10 Gbps
- **Typ provozu:** UDP (bezstavový)
- **Síla provozu klienta na server:** 2 Gbps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26 (2001:db8::100/122), Protocol: UDP, Threshold: 4 Gbps, Limit: 2 Gbps`

Očekávaný výsledek Protector sníží celkový objem na 2 Gbps. Provoz klienta by neměl být nijak ovlivněn, protože jeden útočník generuje 5x tolik objemu na jednoho klienta.



Obrázek 4.2: Testovací případ 1: UDP flood [1:5].

4.4.2 Testovací případ 2: UDP flood [1:1]

Testovací případ je schématicky znázorněn na obrázku 4.3. Objem provozu klienta a útočníka je identický.

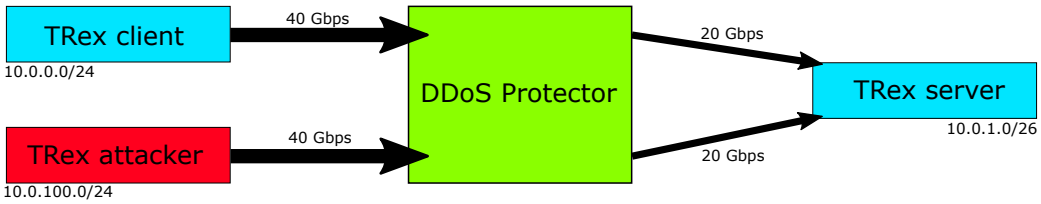
Parametry testu

- **Typ útoku:** UDP flood
- **Síla útoku:** 40 Gbps
- **Typ provozu:** UDP (bezstavový)
- **Síla provozu klienta na server:** 40 Gbps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26 (2001:db8::100/122), Protocol: UDP, Threshold: 60 Gbps, Limit: 40 Gbps`

Očekávaný výsledek Protector sníží celkový objem na 40 Gbps. Očekávané zahození je 50 % provozu klienta a 50 % provozu útočníka.



Obrázek 4.3: Testovací případ 2: UDP flood [1:1].

4.4.3 Testovací případ 3: UDP flood [25000 adres]

Testovací případ je schématicky znázorněn na obrázku 4.4. Tento případ demonstruje nežádoucí chování v případě špatně nastaveného pravidla. Parametry jsou podobné prvnímu případu.

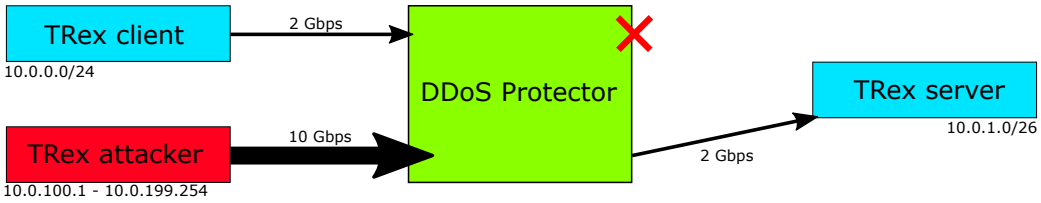
Parametry testu

- **Typ útoku:** UDP flood
- **Síla útoku:** 10 Gbps
- **Typ provozu:** UDP (bezstavový)
- **Síla provozu klienta na server:** 2 Gbps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26 (2001:db8::100/122), Protocol: UDP, Threshold: 4 Gbps, Limit: 2 Gbps`

Očekávaný výsledek Protector sníží celkový objem na 2 Gbps. 80 % útoku (8 Gbps) bude zahozeno, 20 % útoku (2 Gbps) bude propuštěno. Veškerý provoz klienta by měl být zahozen. Jeden klient generuje ∼7,8 Mbps, zatímco jeden útočník ∼0,4 Mbps.



Obrázek 4.4: Testovací případ 3: UDP flood [25000 adres].

4.4.4 Testovací případ 4: TCP SYN flood [SYN-Drop]

Testovací případ je schématicky znázorněn na obrázku 4.5. Případ aplikuje SYN-Drop strategii pro mitigaci TCP SYN flood útoku.

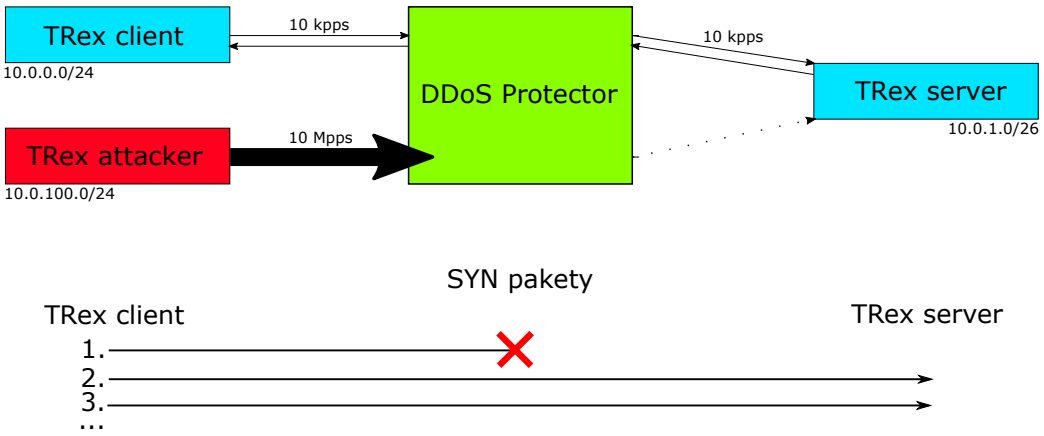
Parametry testu

- **Typ útoku:** TCP SYN flood
- **Síla útoku:** 10 Mpps
- **Typ provozu:** HTTP (stavový)
- **Síla provozu klienta na server:** 10 kpps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26 (2001:db8::100/122), Strategy: SYN-DROP, Threshold: 2 Mpps, Host-syn-soft-threshold: 5, Host-syn-hard-threshold: 100`

Očekávaný výsledek Protector výrazně sníží počet SYN paketů, které k serveru doputují. Závisí na nastavení časovače resetování čítačů SYN a ACK paketů. Pokud se čítač SYN paketů resetuje každých 15 sekund, pak je každých 15 sekund propuštěno $4 * 254$ SYN paketů útočníka. První pokus o navázaní spojení klienta se serverem selže, další pokusy budou úspěšné. Po resetu čítače ACK paketů (typicky 30 sekund) se situace znovu opakuje. Je nutné si dát pozor na hodnotu host-syn-hard-threshold, aby nebyla příliš nízká, nebo by klient mohl být zablokován, pokud by navazoval spojení příliš často (byl příliš agresivní).



Obrázek 4.5: Testovací případ 4: TCP SYN flood [SYN-Drop].

4.4.5 Testovací případ 5: TCP SYN flood [RST-Cookies]

Testovací případ je schématicky znázorněn na obrázku 4.6. Případ aplikuje RST-Cookies strategii pro mitigaci TCP SYN flood útoku.

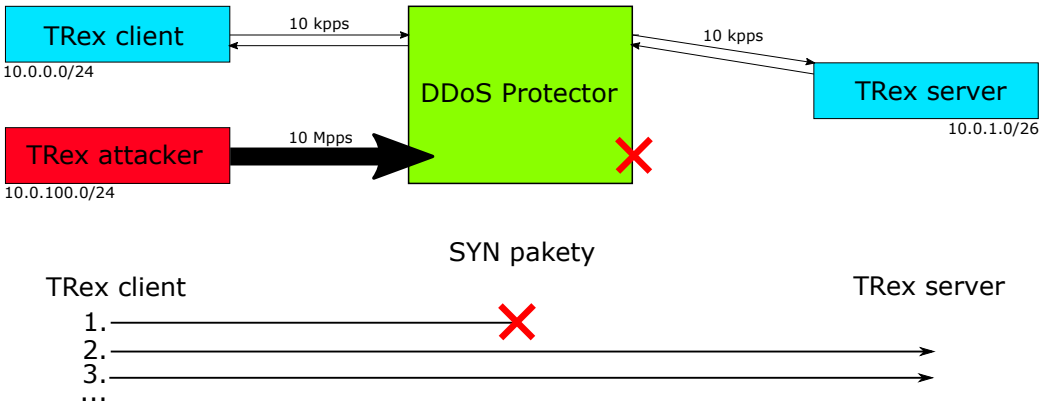
Parametry testu

- **Typ útoku:** TCP SYN flood
- **Síla útoku:** 10 Mpps
- **Typ provozu:** HTTP (stavový)
- **Síla provozu klienta na server:** 10 kpps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26 (2001:db8::100/122), Strategy: RST-COOKIES, Threshold: 2 Mpps`

Očekávaný výsledek Protector výrazně sníží počet SYN paketů, které k serveru doputují. Závisí na nastavení whitelist časovače. Po aktivaci strategie bude každý SYN paket útočníka zahozen. První pokus o navázání spojení klienta selže, ale další pokus bude úspěšný, dokud nebude resetován whitelist čítač (typicky 30 sekund).



Obrázek 4.6: Testovací případ 5: TCP SYN flood [RST-Cookies].

4.4.6 Testovací případ 6: TCP SYN flood [ACK-Spoofing]

Testovací případ je schématicky znázorněn na obrázku 4.7. Případ aplikuje ACK-Spoofing strategii pro zeslabení TCP SYN flood útoku.

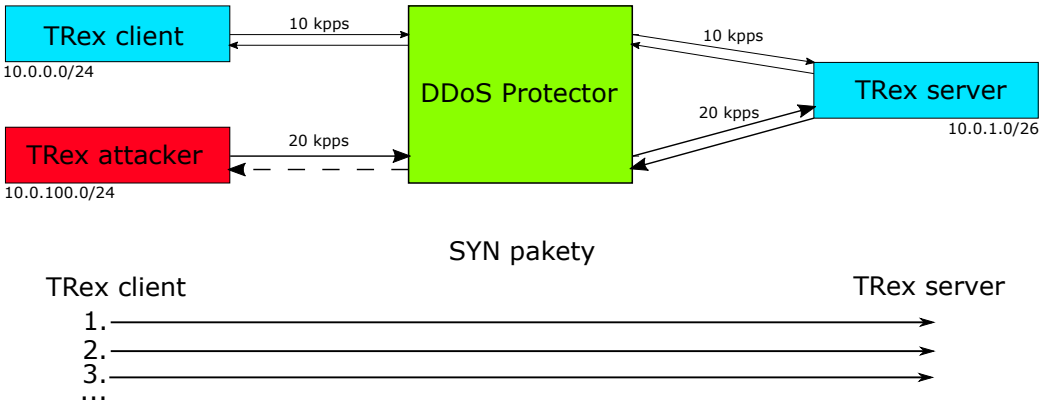
Parametry testu

- **Typ útoku:** TCP SYN flood
- **Síla útoku:** 10 Mpps
- **Typ provozu:** HTTP (stavový)
- **Síla provozu klienta na server:** 10 kpps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26 (2001:db8::100/122), Strategy: ACK-SPOOFING, Threshold: 2 Mpps`

Očekávaný výsledek Protector vůbec nesníží počet SYN paketů, které k serveru doputují. Naopak, pokud server pokus spojení přijme, pak Protector zajistí jeho úspěšné navázání. První pokus o navázání spojení klienta bude úspěšný a Protector by neměl nijak ovlivnit komunikaci klienta a serveru.



Obrázek 4.7: Testovací případ 6: TCP SYN flood [ACK-Spoofing].

4.4.7 Testovací případ 7: TCP SYN flood [Zátěžové testy]

Stresové varianty pro všechny tři TCP SYN flood mitigační strategie. Vzhledem k tomu, že syn-flood modul není na rozdíl od amplifikačního modulu hardwarově akcelerovaný, je cílem zjistit, jak velký útok jsou jednotlivé strategie schopny softwarově zpracovat.

Parametry testu

- **Typ útoku:** TCP SYN flood
- **Síla útoku:** 20-100 Mpps
- **Typ provozu:** HTTP (stavový)
- **Síla provozu klienta na server:** 10 kpps

4.4.8 Testovací případ 8: Multi-vector útok [2 útoky]

Testovací případ je schématicky znázorněn na obrázku 4.8. Jedná se o první multi-vector útok složený ze dvou útoků.

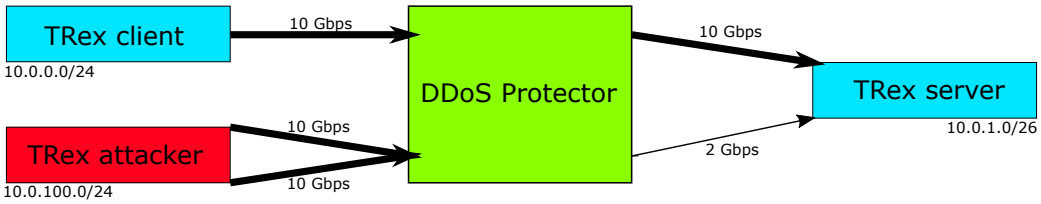
Parametry testu

- **Typ útoku:** Multi-vector útok – DNS amplifikační útok a ICMP flood
- **Síla útoku:** $2 * 10$ Gbps
- **Typ provozu:** UDP (bezstavový)
- **Síla provozu klienta na server:** 10 Gbps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26, Protocol: UDP, Dst port: 53, Threshold: 3 Gbps, Limit: 1 Gbps`
- `Dst net: 10.0.1.0/26, Protocol: ICMP, Threshold: 3 Gbps, Limit: 1 Gbps`
- `Dst net: 10.0.1.0/26, Protocol: UDP, Threshold: 15 Gbps, Limit: 10 Gbps`

Očekávaný výsledek Protector sníží DNS amplifikační útok na 1 Gbps, ICMP útok na 1 Gbps. Do obecného UDP pravidla spadají i útočící DNS pakety, ty jsou ale zachyceny DNS pravidlem a většina z nich (9 Gbps) zablokována. Do UDP pravidla pak spadá 10 Gbps (klient) + 1 Gbps (zbylý DNS útok), celkově 11 Gbps, což není dostatek objemu pro aktivaci mitigace na základě 15 Gbps threshold hodnoty a toto pravidla zůstane neaktivní. Provoz klienta by měl zůstat nedotčený a celkový objem provozu bude 12 Gbps.



Obrázek 4.8: Testovací případ 8: Multi-vector útok [2 útoky].

4.4.9 Testovací případ 9: Multi-vector útok [4 útoky]

Testovací případ je schématicky znázorněn na obrázku 4.9. Jedná se o rozšíření předchozího případu o dva další typy útoku.

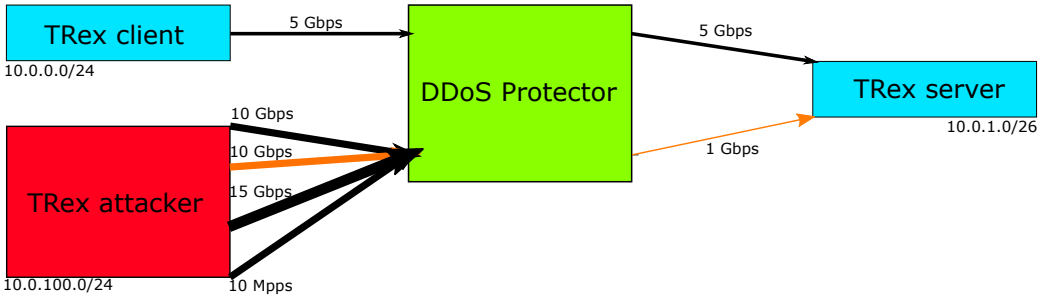
Parametry testu

- **Typ útoku:** Multi-vector útok – DNS amplifikační útok, ICMP flood, UDP flood, TCP SYN flood
- **Síla útoku:** 10 Gbps, 10 Gbps, 15 Gbps, 10 Mpps
- **Typ provozu:** UDP (bezstavový)
- **Síla provozu klienta na server:** 5 Gbps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26, Protocol: UDP, Dst port: 53, Threshold: 3 Gbps, Limit: 1 Gbps`
- `Dst net: 10.0.1.0/26, Protocol: ICMP, Threshold: 3 Gbps, Limit: 1 Gbps`
- `Dst net: 10.0.1.0/26, Protocol: UDP, Threshold: 10 Gbps, Limit: 5 Gbps`
- `Dst net: 10.0.1.0/26, Strategy: RST-COOKIES, Threshold: 2M pps`

Očekávaný výsledek Protector sníží DNS amplifikační útok na 1 Gbps, ICMP útok na 1 Gbps. Obecný UDP flood sníží na 5 Gbps, přesně tak, aby byly propuštěny jen pakety od klienta. V rámci snižování veškerého UDP provozu na 5 Gbps je blokován i 1 Gbps objem DNS amplifikačního útoku, který je propuštěn prvním pravidlem (DNS útočník generuje dvojnásobek objemu oproti klientovi). RST-Cookies strategie nepropustí žádný SYN paket. Celkový objem putující na server by měl být 6 Gbps.



Obrázek 4.9: Testovací případ 9: Multi-vector útok [4 útoky].

4.4.10 Testovací případ 10: Multi-vector útok [stavový provoz]

Testovací případ je schématicky znázorněn na obrázku 4.10. Je podobný předchozímu případu, ale místo bezstavového provozu se generuje stavový provoz.

Parametry testu

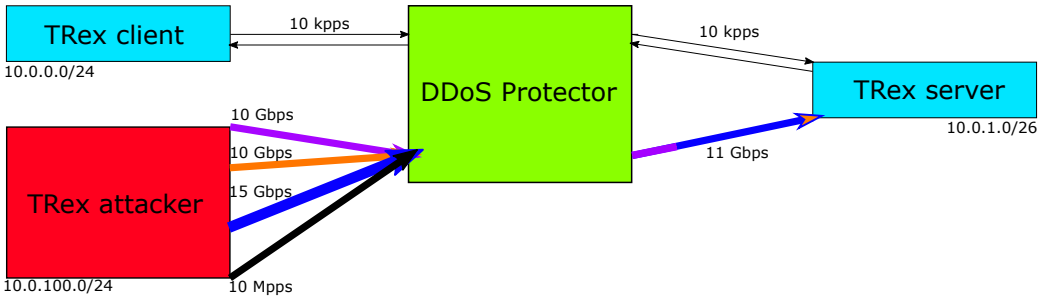
- **Typ útoku:** Multi-vector útok – DNS amplifikační útok, ICMP flood, UDP flood, TCP SYN flood
- **Síla útoku:** 10 Gbps, 10 Gbps, 15 Gbps, 10 Mpps
- **Typ provozu:** HTTP (stavový)
- **Síla provozu klienta na server:** 10 kpps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26, Protocol: UDP, Dst port: 53, Threshold: 3 Gbps, Limit: 2 Gbps`

- `Dst net: 10.0.1.0/26, Protocol: ICMP, Threshold: 3 Gbps, Limit: 1 Gbps`
- `Dst net: 10.0.1.0/26, Protocol: UDP, Threshold: 12 Gbps, Limit: 10 Gbps`
- `Dst net: 10.0.1.0/26, Strategy: RST-COOKIES, Threshold: 2 Mpps`

Očekávaný výsledek Protector sníží DNS amplifikační útok na 2 Gbps, ICMP útok na 1 Gbps. Propustí 10 Gbps UDP útoku. Do obecného UDP flood pravidla se počítá i 2 Gbps propuštěného DNS amplifikačního útoku. Kapacita na obecný UDP flood (ne DNS) pak ve skutečnosti je 8 Gbps. RST-Cookies strategie nepropustí žádný SYN paket. První pokus klienta o navázání spojení selže každých 30 sekund (výchozí hodnota pro whitelist timeout), jinak provoz probíhá v pořádku.



Obrázek 4.10: Testovací případ 10: Multi-vector útok [stavový provoz].

4.4.11 Testovací případ 11: HTTP flood

Testovací případ je schématicky znázorněn na obrázku 4.11. Jedná se o jediný test na HTTP flood útok.

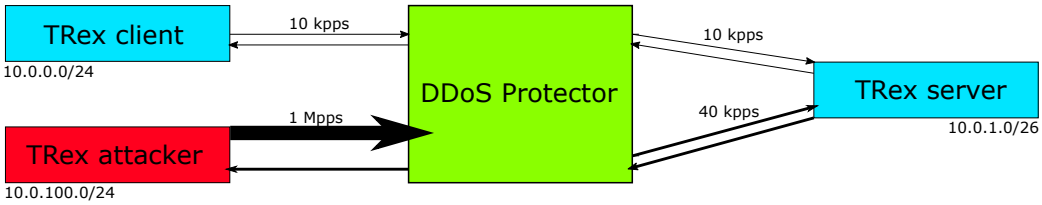
Parametry testu

- **Typ útoku:** HTTP flood
- **Síla útoku:** 1 Mpps
- **Typ provozu:** HTTP (stavový)
- **Síla provozu klienta na server:** 10 kpps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26, Protocol: TCP, Threshold: 100 kpps, Limit: 50 kpps`

Očekávaný výsledek Protector sníží celkový objem na 50 kpps, z čehož je 40 kpps útoku. Provoz klienta by neměl být nijak ovlivněn.



Obrázek 4.11: Testovací případ 11: HTTP flood.

4.4.12 Testovací případ 12: Slowloris

Jedná se o jediný test na Slowloris útok.

Parametry testu

- **Typ útoku:** Slowloris
- **Síla útoku:** 1000 současných spojení
- **Typ provozu:** HTTP (stavový)
- **Síla provozu klienta na server:** 10 kpps

Pravidla Protectoru

- `Dst net: 10.0.1.0/26, Protocol: TCP, Threshold: 20 kpps, Limit: 15 kpps`

Očekávaný výsledek Současná verze Protectoru není schopna útok zaznamenat. Vývojový tým pracuje na nové verzi, která v sobě integruje IDS (Intrusion Detection System). Pak bude Protector moci podporovat filtraci pro nízkoobjemové útoky i na vyšších vrstvách provozu.

4.4.13 Testovací případ 13: Propustnost

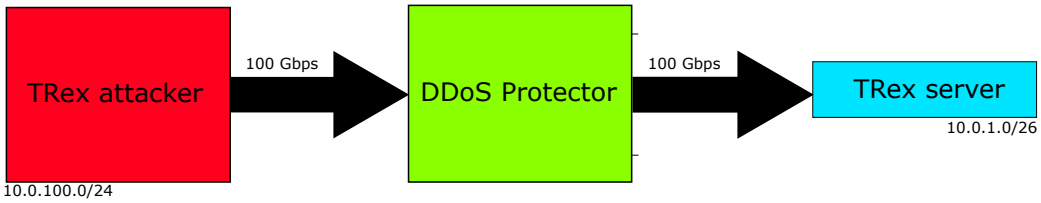
Testovací případ je schématicky znázorněn na obrázku 4.12. Jedná se o test propustnosti, který ověří, že TRex i Protector jsou schopny generovat a zpracovávat 100 Gbps provoz. V tomto případě postačí, když bude pakety generovat pouze útočník. Na straně serveru se ověří, že počet přijatých bitů (Gbps) odpovídá generované hodnotě, tedy že Protector propustnost nijak nesnižuje. Pokud by provoz generoval i klient, došlo by k překročení maximální kapacity Protectoru 100 Gbps.

Parametry testu

- **Typ útoku:** UDP flood
- **Síla útoku:** 100 Gbps
- **Typ provozu:** Žádný

Pravidla Protectoru

- Žádná



Obrázek 4.12: Testovací případ 13: Propustnost.

4.4.14 Testovací případ 14: Latence ping

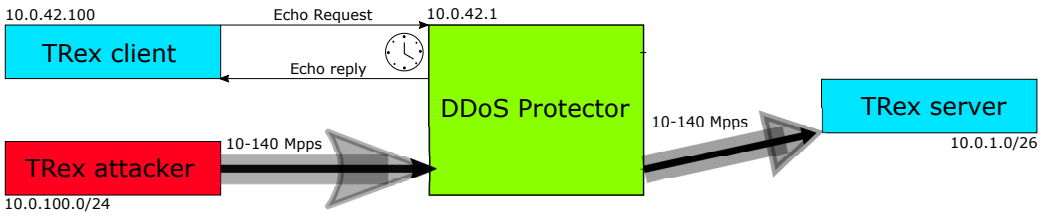
Testovací případ je schématicky znázorněn na obrázku 4.13. Jedná se o test latence, který měří, jak moc se čas odpovědi prodlouží, když je Protector pod zátěží různých intenzit útoku.

Parametry testu

- **Typ útoku:** UDP flood
- **Síla útoku:** 10-140 Mpps
- **Typ provozu:** ICMP Echo Request-Reply

Pravidla Protectoru

- Hodnota threshold těsně nad útokem.



Obrázek 4.13: Testovací případ 14: Latence ping.

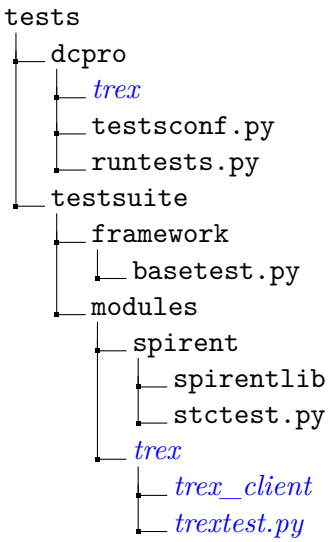
Kapitola 5

Implementace

Kapitola se zabývá implementací navržených testovacích případů. První podkapitola 5.1 popisuje strukturu současně používaných testů a její rozšíření o nové testy. Druhá podkapitola 5.2 se zabývá implementací základní třídy `TRexTest`, která zajišťuje komunikaci a metody pro práci s TRexem. Poslední podkapitola 5.3 popisuje vybrané implementační detaily z jednotlivých testů.

5.1 Struktura současných testů

Významná část aktuální struktury používaných testů je znázorněna na obrázku 5.1 [4].



Obrázek 5.1: Zkrácená struktura testovacího systému.

Soubor `basetest.py` implementuje bázovou třídu `BaseTest`, ze které všechny další třídy dědí. BaseTest definuje obecnou posloupnost vykonání testu a základní kostru výpisu o průběhu testu. Třída předpokládá, že třídy z ní dědící implementují jednotlivé fáze testu a tím test samotný. Každý test prochází různými fázemi. Seznam všech fází testu je následující [6]:

- *setup* – Setup fáze je určena k nastavení v rámci testovacího frameworku. Řeší se zde vytváření adresářů, nastavení cest k souborům, se kterým test pracuje apod.. V této

fázi jsou definovány jednotlivé případy (scénáře) konkrétního testu. Scénáře definují například konkrétní pravidla pro Protector nebo objem a strukturu generovaného provozu. Test může definovat například 10 scénářů, kdy počet generovaného objemu s každým dalším scénářem vzrůstá, zbytek parametrů však zůstává stejný napříč scénáři. Dále se v této fázi inicializuje komponenta `logger`, která má na starosti logování průběhu testu.

- *prolog* – Prolog fáze je určena pro přípravu samotného testu. Jsou zde umístěny příkazy pro konfiguraci prostředí a komponent, které se mezi jednotlivými scénáři nemění (např. rezervace generátoru, povolení IPv6 apod.).
- *testing* – Testing fáze provádí spouštění jednotlivých scénářů nastavených uvnitř třídy testu. Běhy jednotlivých scénářů se pak skládají ze tří fází:
  - *pre-test* – Obsahuje konfiguraci spojenou s daným testovacím scénářem (nahrání pravidel do databáze Protectoru, konfigurace paketů/provozu pro generátor...).
  - *test* – Implementuje průběh testu samotného. Jedná se o generování a zpracování provozu a následné vyhodnocení výsledků. Každý testovaný scénář by měl před ukončením zavolat speciální metodu, kterou oznámí svůj úspěch či neúspěch, v případě neúspěchu i důvod.
  - *post-test* – Provádí úklid konfigurace po daném testovacím scénáři. Typicky se jedná o reverzní operace k pre-test fázi (odstranění pravidel z databáze Protectoru, reset generátoru...).
- *epilog* – Fáze epilog slouží k úklidu prostředí a komponent společných pro celý test. Typicky se jedná o reverzní operace k prologu (ukončení rezervace generátoru...).

Soubor `stctest.py` v adresáři `spirent` implementuje třídu `StcTest`, která rozšiřuje základní třídu `BaseTest` o práci s generátorem provozu Spirent. Adresář `spirentlib` obsahuje soubory tvořící API pro práci se Spirentem. Z této třídy pak dědí jednotlivé testy Protectoru, které se nacházejí v adresáři `dcpro`. Implementace testů založených na generátoru provozu TRex bude používat stejnou hiearchii. Adresář `modules` bude doplněn o nový adresář `trex`, ve kterém se bude nacházet soubor `trextest.py`. Ten bude implementovat třídu `TRexTest`, která rozšíří základní třídu `BaseTest` o práci s generátorem provozu TRex. Adresář také bude obsahovat adresář `trex_client` obsahující soubory tvořící API pro práci s TRexem [53]. Nakonec adresář `dcpro` bude doplněn o nový adresář `trex`, ve kterém se budou nacházet konkrétní testy.

Spuštěním souboru `runtests.py` se postupně spustí všechny nakonfigurované testy. V souboru jsou definovány výchozí argumenty pro testy. Patří sem například IP adresa generátoru provozu. Seznam testů, které se mají spustit, je nakonfigurován v souboru `testsconf.py`.

5.2 Nastavení TRex generátoru, třída TRexTest

V podkapitole 3.2 byla popsána manuální práce s TRex generátorem, zejména nutnost nastavit konfigurační soubor, spuštění TRexe z příkazové řádky a sledování aktuálního stavu (počet odesílaných paketů, vytížení procesoru...) na textovém terminálu. Pro automatizované testy je vhodnější, aby se TRex spustil ve formě démona, který bude běžet na pozadí a jediná interakce s ním bude přes poskytnuté API. Samotný program TRex mód démona

nepodporuje, ale v archivu je přibalen program `trex_daemon_server`, který funkcionalitu démona zajišťuje. Tento démon se stará o správu pouze jedné instance TRex generátoru. Mezi jeho poskytované funkce patří zejména spuštění TRexe a ukončení TRexe, tzn., že pokud neběží testy, nemusí běžet ani TRex, pouze na prostředky nenáročný démon.

Veškeré navržené testovací případy (s výjimkou Slowloris) pracují se dvěmi instancemi TRexe (TRex generující útok a TRex generující legitimní provoz) a tudíž bude nutné mít spuštěné dva démony, kdy každý bude naslouchat na jiném portu. Dvě instance TRexe vyžadují dva konfigurační soubory. Ty definují cílovou MAC adresu (toto bude MAC adresa Protectoru) a případně VLAN ID. Konfigurační soubory se budou lišit zejména PCI adresami fyzických portů síťové karty a síťovými porty pro naslouchání. TRex generující provoz potřebuje jeden fyzický port pro klienta a druhý fyzický port pro server, zatímco TRex generující útok pouze jeden fyzický port. Port by se neměl sdílet více instancemi TRexe. Síťové porty pro naslouchání poskytují uživatelským programům možnost se připojit na danou instanci TRexe a zaslat ji příkazy přes definované API. Toto je důležité, protože pro provedení testů je nutné provést dva kroky. První krok je připojit se na démona, který spustí instanci TRexe. Poté se test připojí na tuto spuštěnou instanci a s ní poté primárně komunikuje.

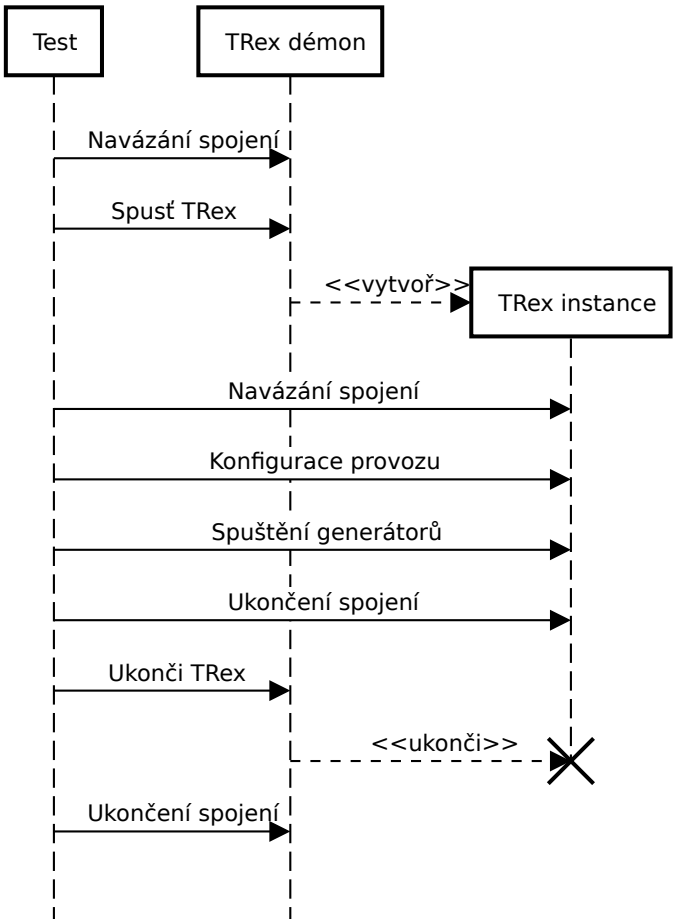
Třída `TRexTest` byla implementována se snahou být obecná. Proto nepočítá s pevným počtem dvou TRex instancí, ale dovoluje jednotlivým testům vytvořit libovolné množství instancí (uživatel by samozřejmě měl respektovat omezené zdroje zařízení, zejména počet fyzických portů DPDK-kompatibilních síťových karet). Vytvoření nové instance zajišťuje metoda `_setup_trex_instance()`. Ta přijímá 6 argumentů:

- *server* – IP adresa zařízení, na kterém se TRex nachází.
- *daemon_port* – Port TRex démona.
- *sync_port* – Synchronní port instance TRex generátoru.
- *async_port* – Asynchronní port instance TRex generátoru.
- *config_file* – Jméno konfiguračního souboru. Soubor na zařízení již musí existovat.
- *statefulness* – Stavovost TRex generátoru. Může být nestavový (stl) nebo plně stavový (astf).

Metoda vrací objekt, který přijímají další metody jako vstupní argument. Předpokládané volání této metody je ve fázi setup konkrétního testu (samotná třída `TRexTest` fázi setup nijak nerozšiřuje od základní `BaseTest` implementace). Metoda zároveň ukládá další potřebné informace do interních struktur. Když pak test přejde na další fáze prolog, tak třída automaticky zajistí navázání spojení se všemi instancemi TRexe a zabrání fyzických portů dle konfiguračního souboru. Poslední fáze epilog naopak zajistí ukončení spojení se všemi instancemi TRexe a poté jejich ukončení přes jejich démona. Fázi testing implementují konkrétní testy. Třída `TRexTest` jim pro tuto fázi poskytuje některé užitečné metody.

První z metod jsou metody pro nahrání profilu provozu. Profil se vytváří pomocí předdefinovaného rozhraní v jazyce Python. V případě bezstavového provozu je to tvorba paketů nástrojem Scapy, v případě stavového se použijí příkazy poskytované TRexem. Při implementaci jsem chtěl oddělit definici provozu od samotných testů. Jeden profil totiž může být využit více testy. Zvolené řešení vede na definici profilů v samostatných `*.py` souborech. Obsah souboru je nahrán jako lokální kód do metody použitím funkce `exec()`, tedy jako

kdyby se kód souboru vložil přímo do těla metody, která zpracovává profily. Další jsou metody pro start generování. Zde je zajímavé, že metody umožňují zvolit velikost generovaného provozu ať už v paketech za sekundu (pps) nebo bitech za sekundu (bps) včetně prefixů jako *15kpps* nebo *1.455 Gbps*. Zbytek jsou metody jako získání statistik pro vyhodnocení testů nebo vymazání všech statistik pro čistý start nového scénáře. Dosavadní text graficky shrnuje zjednodušený diagram komunikace na obrázku 5.2. Na obrázku je znázorněna komunikace pouze s jednou instancí TRexe.



Obrázek 5.2: Zjednodušený diagram komunikace mezi testem, TRex démonem a TRex generátorem.

Implementované metody vychází z možností, které definuje API TRexe. To může být v některých případech celkem bohaté a třída `TRexTest` neimplementuje vše, co TRex nabízí. Tvůrce testů ovšem může využít veškeré metody, které TRex poskytuje. Stačí k tomu využít objekt, který vrací metoda `_setup_trex_instance()`. Není tudíž ani povinné využívat metody třídy `TRexTest`, lze použít metody TRexe přímo. Nicméně metody třídy `TrexTest` většinou pracují se všemi vytvořenými instancemi a při větším počtu těchto instancí mohou ušetřit duplikaci kódu.

5.3 Vybrané implementační detaily navržených případů

Struktura nových testů je ilustrována na obrázku 5.3. Testy jsou děleny do dvou hlavních kategorií. První kategorie (prefix *stl*) označuje testy, které pracují pouze s bezstavovými instancemi TRexe. Druhá kategorie (prefix *astf*) označuje testy, které využívají alespoň jednu stavovou instanci TRexe. Důvod pro takové rozdělení je kvůli statistikám, které se výrazně liší v závislosti na stavovosti generátoru.

```
tests
└── dcpro
    └── trex
        ├── stl_udp_flood
        ├── stl_udp_flood_bps
        ├── stl_udp_flood_ipv6
        ├── stl_multivector_flood
        ├── astf_syn_flood
        ├── astf_multivector_flood
        ├── astf_http_flood
        ├── astf_syn_drop_stress_testing
        ├── astf_rst_cookies_stress_testing
        ├── stl_throughput
        ├── stl_ping_latency
        └── trex_common_test.py
```

Obrázek 5.3: Adresářová struktura TRex testů.

Při implementaci se ukázalo, že mnoho testů sdílí stejný kód. Proto byla vytvořena nová třída `TRex_common_test` (soubor `trex_common_test.py`), která často používaný kód implementuje. Veškeré testy jsou tak odvozeny z této třídy. Třída obsahuje nahrávání a odstraňování pravidel z/do databáze Protectoru, nastavení třídních proměnných a metody pro vyhodnocení zejména *stl* testů.

První test `stl_udp_flood` implementuje částečně pozměněné testovací případy 1, 2 a 3 (z podkapitoly 4.4). Pro každý případ jsou nastaveny hodnoty jako pravidla Protectoru, velikost generovaného provozu (zde jsou oproti navrženým testům použity jednotky v paketech za sekundu, nikoliv bitech za sekundu), cesta k souboru definující profil provozu nebo očekávané procento zablokování útočných paketů. Toto procento blokace je získáno tak, že se v průběhu testu zachytávají pakety, které dorazí na server. Ty jsou následně analyzovány na zdrojovou IP adresu. Podle ní se dá jednoduše určit, jestli se jedná o paket útočníka nebo klienta. Podle poměru přijatých paketů útočníka vůči celkovému počtu analyzovaných paketů se pak získá procento zablokování útočných paketů. Po ukončení generování se také provádí kontrola na počet přijatých paketů na základě čítačů serveru.

Výpis 5.1 obsahuje ukázku části souboru, který definuje profil legitimního provozu, konkrétně profil (resp. strukturu paketů) klienta. Všechny pakety mají pevně danou délku. Při definici hlaviček paketů kód automaticky přidá Dot1Q hlavičku pro podporu technologie VLAN. V samotném souboru není definována žádná třída, odkud se tedy bere *self* na řádcích 5 a 6?

V popisu implementace třídy `TRexTest` bylo vysvětleno, že soubor je nahrán metodou `exec()`, tedy jako kdyby se kód souboru vložil přímo do těla metody třídy `TRexTest`. A protože daná metoda je součást třídy TRexTest, pak i tento kód má přístup ke všem

objektům třídy `TRexTest` (resp. třídy konkrétního testu). Kód dále obsahuje ukázku Field Engine, který paketům přiřazuje IP adresy a UDP porty náhodně v rozsahu 1025 až 65000.

```
pkt_size = 220   # pevna delka paketu
udp_traffic_pkt = Ether()

# pridej VLAN, pokud je treba
if self._TREX_TRAFFIC_INSTANCE.get_port_attr(0)['vlan'] != "-":
    udp_traffic_pkt = udp_traffic_pkt/Dot1Q(vlan = self._TREX_TRAFFIC_INSTANCE.
    get_port_attr(0)['vlan'])

udp_traffic_pkt = udp_traffic_pkt/IP()/UDP()
padding = (pkt_size - len(udp_traffic_pkt)) * 'x'

# Vyuziti Field Engine - manipulace s hodnotami paketu za behu
vm_client = STLScVmRaw([
    # STLVmFlowVar - definice hodnot
    STLVmFlowVar("ip_dst", min_value="10.0.1.1", max_value="10.0.1.62", size=4, step
    =1, op="random"),
    # Dle zdrojove IP adresy se zrejme jedna o FE pro klientske pakety
    STLVmFlowVar("ip_src", min_value="10.0.0.1", max_value="10.0.0.254", size=4, step
    =1, op="random"),
    STLVmFlowVar(name="src_port", min_value=1025, max_value=65000, size=2, op="random"
    ),
    STLVmFlowVar(name="dst_port", min_value=1025, max_value=65000, size=2, op="random"
    ),
  # STLVmWrFlowVar - zapis hodnot do paketu
    STLVmWrFlowVar(fv_name="ip_src", pkt_offset= "IP.src"),
    STLVmWrFlowVar(fv_name="ip_dst", pkt_offset= "IP.dst"),
    STLVmWrFlowVar(fv_name="src_port", pkt_offset= "UDP.sport"),
    STLVmWrFlowVar(fv_name="dst_port", pkt_offset= "UDP.dport"),
  # Protoze se hodnoty v paketu zmenili, je treba prepocitat kontrolni soucet
    STLVmFixChecksumHw(l3_offset="IP",l4_offset="UDP",l4_type=CTRexVmInsFixHwCs.
    L4_TYPE_UDP)
]);

client_streams = STLStream(packet = STLPktBuilder(pkt = udp_traffic_pkt/padding, vm =
    vm_client), mode = STLTXCont())
```

Výpis 5.1: Vybraný obsah souboru definující klientské pakety.

Test `stl_udp_flood_bps` je téměř identický s prvním testem `stl_udp_flood`, liší se však v použití jednotek bps místo pps. Protector totiž počet bitů za sekundu počítá na L3 vrstvě, zatímco TRex je počítá na vrstvě L2 (je možné nastavit i L1). Každý odeslaný paket je pak z pohledu Protectoru o 18B (22B v případě použití technologie VLAN) menší kvůli absenci 6 B zdrojové MAC adresy, 6 B cílové MAC adresy, 2 B EtherType a 4 B kontrolního součtu. V případě pevně stanovené velikosti generovaných paketů (viz 5.1) je možné provést přepočet. Pevná velikost paketů ale omezuje možnosti při definování profilu provozu a proto jsou ostatní testy, oproti původnímu návrhu, definovány v paketech za sekundu (pps). Z pohledu očekávaných výsledků se ovšem nic nemění při dodržení poměru velikostí mezi legitimním provozem a útokem. Test `stl_udp_flood_ipv6` je podobný, liší se však v použití IPv6 adres. Úprava IPv6 adres přes Field Engine podporuje pouze spodních 32 bitů adresy. Zbylé horní bity (96) jsou pevně dané uživatelem. Zadávání spodních 32 bitů je identické se zadáváním IPv4 adres (viz 5.1). Uvažujme pevně zvolenou síť 2001:db8::. Pak IPv4 adresa 0.0.0.1 bude převedena na 2001:db8::1, adresa 0.0.3.255 na 2001:db8::3ff apod..

Test `stl_multivector_flood` implementuje případy 8 a 9. Původně se, na rozdíl od testu `stl_udp_flood`, při analýze zachycených paketů nelze spoléhat na zdrojovou IP adresu, protože jeden útočník generuje více typů útoku. Je pak nutné pakety analyzovat hlouběji na základě typů jednotlivých útoků. Při experimentech se ale zjistilo, že pokud je jedna IP adresa obsažena ve dvou nebo více aktivních pravidlech, pak se zablokuje veškerý její provoz. Jedná se o omezení z důvodu specifické implementace FPGA firmware Protectoru. Test by pak selhal na kontrole očekávaného procenta blokace útočných paketů. Proto se u multi-vector útoků musí jednotlivé typy útoků generovat z disjunktního adresního prostoru.

Test `astf_syn_flood` implementuje případy 4, 5 a 6. Test kontroluje, že počet útočných SYN paketů, které dorazí na server je snížen podle očekávání použité strategie. Zároveň kontroluje, že počet úspěšně navázaných spojení mezi klientem a serverem je (téměř, z důvodu nepřesného TRex plánovače) identický s počtem pokusů o navázání spojení. Při vyhodnocení sleduje i počet opakovaných pokusů o spojení (znovuodeslání SYN paketů). Soubor s profilem provozu se příliš neliší od ukázky 3.3. Důvod použití manuální implementace HTTP provozu je kvůli jednoduchosti vyhodnocení testů. Při použití složitého PCAP souboru by se nejdříve muselo zjistit, jaké jsou typické výsledky při přehrávání bez útoku a z těch poté vycházet při vyhodnocování testu. Některé PCAP soubory totiž mohou obsahovat záměrné selhání spojení či jiné chyby, které, ačkoliv více aproximují realistický provoz, dělají vyhodnocení těžší. Vlastní implementace provozu je také lépe čitelná a rozšiřitelná. Test `astf_http_flood` implementuje případ 11 a jedná se jediný test, který využívá dvě stavové instance TRexu.

Testy `astf_syn_drop_stress_testing` a `astf_rst_cookies_stress_testing` implementují případ 7. Jedná se tedy o zátěžové testy TCP SYN flood strategií. Test pro ACK-Spoofing strategii nebyl implementován, protože se tato strategie v testech ukázala jako nefunkční. Z původního návrhu nebyl implementován útok typu Slowloris. Protector takový typ útoku zatím nepodporuje, ale díky použitému modulárnímu systému je možné test v budoucnu doplnit.

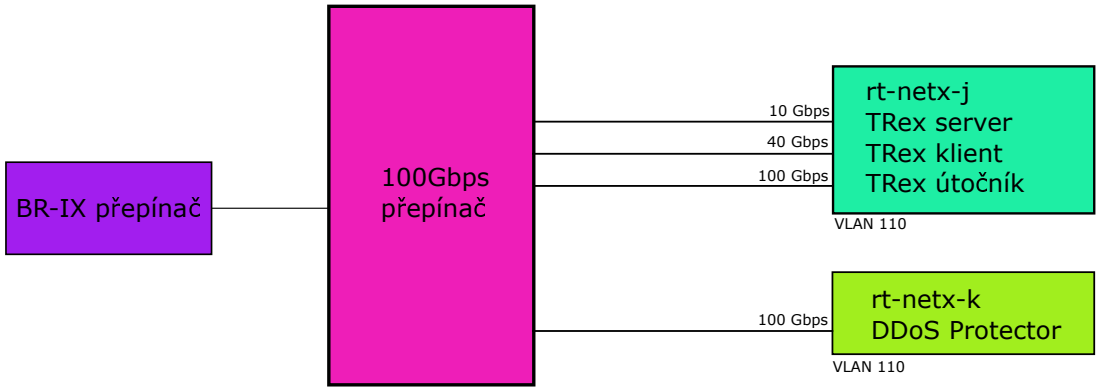
Kapitola 6

Testování Protectoru

Tato kapitola popisuje testování zařízení DDoS Protector pomocí navržených testovacích případů a jejich vyhodnocení. V první podkapitole 6.1 je popsáno testovací prostředí. Následující kapitoly se zabývají popisem a výsledky jednotlivých testů. Poslední podkapitola 6.9 shrnuje dosažené výsledky v tabulce.

6.1 Testovací prostředí

Prostředí pro instalaci TRexe a vývoj testů bylo realizováno v testovacím prostředí (laboratoři) Centra výpočetních a informačních služeb VUT (CVIS VUT). Fyzické zapojení všech potřebných zařízení a zajištění vzdáleného přístupu na stroje bylo zajištěno pracovníky CVIS. Prostředí je propojeno s brněnským peeringovým centrem BR-IX. Pro logické oddělení provozu je využita technologie VLAN. Provoz v použité laboratoři má přiřazenou VLAN ID 110. Schéma zapojení je na obrázku 6.1. Protector běží na stroji rt-netx-k, obě instance TRexe běží na stroji rt-netx-j. Obě zařízení disponují procesorem Intel Xeon Gold 6210U s frekvencí 2510 MHz, 20 fyzickými jádry a více než 16 GB paměti RAM.



Obrázek 6.1: Schéma zapojení testovacího prostředí.

Použitá verze TRexe je 2.74. Protector používá FPGA síťovou kartu NFB-200G2QL firmy Netcope, která dosahuje plných 100 Gbps. TRex využívá dvě síťové karty od firmy Mellanox. Obě karty jsou dvouportové z řady ConnectX-5 podporující 100 Gbps na každém portu. V prostředí jsou zapojeny do přepínače optickými spoji o rychlostech 100 Gbps, 40 Gbps a 10 Gbps. Konfigurační soubor pro TRex generující útok byl nastaven tak, aby využil port s 100 Gbps propojem a z pohledu softwaru 8 vláken. TRex generující legitimní

provoz používá 40 Gbps port pro klienta, 10 Gbps port pro server a 20 vláken pro vykonání programu. Této instanci bylo přiděleno více vláken z důvodu vyšší náročnosti na procesor zejména při generování stavového provozu. Z toho vyplývá, že testy musejí být nastaveny tak, aby celkový objem na server nepřesáhl 10 Gbps. Je pak nutná menší úprava velikosti generovaného objemu oproti původnímu návrhu testovacích případů (např. u případu 8 se předpokládal objem 12 Gbps směřující na server). Zároveň v podkapitole 5.3 bylo popsáno, proč se přešlo z původních bitů za sekundu na pakety za sekundu. Jedná se o současné omezení dané dostupným vybavením v době implementace této práce. Do budoucna je plánováno vylepšení přepínače a daných propojů tak, aby podporovaly plnou kapacitu 100 Gbps. Testy potom bude možné snadno přizpůsobit.

6.2 UDP flood

Test realizuje celkem 4 testovací případy. Příklad výpisu z první testovacího případu vypadá následovně:

```
== UDP flood 1: 1Mpps client + 5Mpps attack; 1.5Mpps threshold, 1Mpps limit; 254 clients, 254 attackers ==

Preparing common test environment...
        Clearing database AMP&SYN rules...
        Adding AMP rule: network="10.0.1.0/26" protocol=17 threshold_packets=1500000 limit_packets=1000000
Common environment prepared successfully.

Preparing test environment...
        Clearing TRex ports and stats ...
        Adding TRex attack file trex/basic_udp_attack.py...
        Adding TRex traffic file trex/basic_udp_traffic.py...
Environment prepared successfully.

Running the test scenario...
        Starting both traffic and attack generators ...
        Waiting until generators stop (~20 seconds) ...
Results: Expected to receive: 20000k-30000k packets, actually received: 24146k packets.
Percentual block of attack packets: Expected: 100.00%-100.00%, actually blocked: 100.00%.

--------------------------------
-- Test case result: SUCCESS  --
--------------------------------

Cleaning up common test environment...
        Clearing database AMP&SYN rules...
Common environment cleaned up successfully.
```

Výpis 6.1: Textový výpis prvního testovacího případu UDP flood testu

Na prvním řádku je jméno testovacího případu. Jména byla zvolena tak, aby uživatele informovala o hlavních parametrech testovacího případu a bylo z nich bylo možné odvodit očekávaný výsledek. Toto platí pro veškeré testy. Následuje fáze pre-test. Ta zařídí nahrání pravidel do databáze Protectoru a nahrání profilů provozu do TRex generátorů. Fáze test spouští generátory provozu na dobu definovanou testem. Po ukončení generování se zkontrolují výsledky. První se provede kontrola, jestli generátory vygenerovaly očekávané množství

paketů. Pokud ne, nemá smysl test dále vyhodnocovat. Tato kontrola se standardně na výstup nevypisuje, pokud proběhla v pořádku. Jako další následuje kontrola na počet přijatých paketů. Ta má spodní a horní povolenou hranici. Spodní hranice je vypočítána jako $(delka\ testu) * (limit\ pravidla\ v\ paketech\ za\ sekundu)$ a zaručuje, že pakety nejsou po cestě ztraceny. Všechny testy předpokládají, že hranice threshold bude překročena a tudíž započne mitigační proces, který celkový provoz k serveru sníží na (či těsně pod) hodnotu limit. Horní hranice je vypočítána jako $(dolni\ hranice) + ((celkovy\ provoz\ klienta\ a\ utoku) * (maximalni\ reakcni\ doba\ Protectoru))$ a zaručuje, že Protector je schopen útok detekovat a reagovat v očekávané době. Tato doba je stanovena jako dvojnásobek pozorovacího okna (viz 2.2), což jsou ve výchozím nastavení 2 sekundy. Protector z důvodu šetření zdrojů nezaznamenává statistiky pro jednotlivé zdrojové IP adresy, dokud provoz nedosáhne alespoň 70 % hodnoty threshold [5]. Před testem byl na síti klid, proto když se z nenadání objeví velké množství generováno provozu, tak Protector na konci časového okna nemá údaje o IP adresách, podle kterých by mohl mitigaci spustit. Spustí tedy sběr informací a až na konci druhého pozorovacího okna může provést mitigaci. Poslední je kontrola na procento propuštěných (resp. zablokovaných) paketů od útočníka. Ta má také spodní a horní hranici. V testech, kdy se očekává 0 % nebo 100 % je interval zbytečný. V ostatních testech je ovšem nutná určitá tolerance kolem očekávané hodnoty. IP adresy jsou u generovaných paketů vybírány náhodně ze zadaného rozsahu. Tato náhodnost způsobí, že se procento blokovaných paketů bude v každém běhu mírně lišit. Tato poslední kontrola je vlastnost, kterou současné testy využívající generátor Spirent neumí. Pokud jsou všechny tři kontroly splněny, je test prohlášen za úspěšný. Následuje poslední fáze post-test, která odstraní pravidla z databáze Protectoru.

Zkrácený výpis druhého testovacího případu zobrazuje toleranci u kontroly procenta blokovaných útočných paketů:

```
1 == UDP flood 2: 1Mpps client + 1Mpps attack; 1.9Mpps threshold, 1Mpps limit; 254
      clients, 254 attackers ==
2 ...
3 Results: Expected to receive: 20000k-22000k packets, actually received: 21503k packets
      .
4 Percentual block of attack packets: Expected: 45.00%-55.00%, actually blocked: 50.90%.
5
6 --------------------------------
7 -- Test case result: SUCCESS  --
8 --------------------------------
```

Výpis 6.2: Zkrácený textový výpis druhého testovacího případu UDP flood testu.

Čtvrtý testovací případ je jediný případ, který v UDP flood testu selhává.

```
== UDP flood 4: 1Mpps client + 5Mpps attack; 1.5Mpps threshold, 1Mpps limit; 254
    clients, 25.7k attackers ==
...
Results: Expected to receive: 20000k-30000k packets, actually received: 49180k packets
    .
Percentual block of attack packets: Expected: 0.00%-0.00%, actually blocked: 0.45%.

--------------------------------
-- Test case result: FAILED   --
-- Reason: Expected to receive 20000000-30000000 packets, but received 49180087
    packets instead.
--------------------------------
```

Výpis 6.3: Zkrácený textový výpis čtvrtého testovacího případu UDP flood testu.

Počet přijatých paketů na serveru je vyšší než maximální očekává hodnota. Procento blokovaných útočných paketů je také nepatrně vyšší než očekávaných 0 %. Jako důvod selhávání testu byl zjištěn fakt, že použitý firmware FPGA karty NFB-200G2QL má prostředky pro zablokování pouze 16384 IP adres. Pokud je počet útočících adres větší, jako v tomto případě, tak mitigace neproběhne správně a test podle očekávání selže. Třetí testovací případ je identický čtvrtému, ale útok generuje z pouze ∼10000 IP adres a proto končí úspěchem.

IPv6 a bps varianty UPD flood testu dosahují identických výsledků. Při testování IPv6 variant byla objevena chyba při zadávání specifických IPv6 prefixů do databáze pravidel. Chyba byla následně opravena.

6.3 TCP SYN flood

Test realizuje celkem 8 testovacích případů. Příklad výpisu z prvního testovacího případu pro strategii RST-Cookies vypadá následovně:

```
== RST-Cookies: 1k connections/seconds traffic + 1Mpps SYN flood; 100k threshold ==

Preparing common test environment...
        Clearing database AMP&SYN rules...
        Adding SYN rule: network="10.0.1.0/26" strategy=rst-cookies threshold_packets
    =100000
Common environment prepared successfully.

Preparing test environment...
        Clearing TRex ports and stats ...
        Adding TRex attack file trex/astf_syn_flood/config/attack.py ...
        Adding TRex traffic file trex/astf_syn_flood/config/traffic.py ...
Environment prepared successfully.

Running the test scenario...
        Starting both traffic and attack generators ...
        Waiting until generators stop (~20 seconds) ...
Client
connections initiated (retransmits not included): 20016
connections established (SYN+ACK received): 20016
SYN packets retransmited: 320
embryonic connections dropped (could not establish connection): 0
Server
connections accepted (SYN received): 1985106
   - attack SYNs that got through: 1965090
   - mitigated: 18034910, which is 90% of attack
connections established (ACK received): 20016
connections dropped (received RST or client does not respond): 1965090

--------------------------------
-- Test case result: SUCCESS  --
--------------------------------
```

Výpis 6.4: Textový výpis prvního testovacího případu TCP SYN flood testu využívající RST-Cookies strategii.

Provoz mezi klientem a serverem se sestává z navázání TCP spojení, výměny jedné HTTP GET a jedné HTTP OK zprávy a následného ukončení spojení. Z názvu testu se uživatel dozví, že tato komunikace proběhne 1000krát za sekundu (connections per second = CPS). Pre-test fáze je shodná s předchozím UDP flood testem. Hlavní je odlišný způsob vyhodnocení testu, který využívá více dostupných čítačů. Výsledky testu jsou děleny do dvou části podle čítačů na straně klienta a na straně serveru. Klient se za dobu testu pokusí se serverem navázat 20016 spojení. Každé spojení znamená minimálně jeden SYN paket. Dodatečných 16 pokusů, oproti očekávané hodnotě 20000 ($(delka\ testu) * (CPS)$) je způsobeno nepřesností plánovače TRex generátoru. Server naopak dostal téměř 2 miliony žádostí o navázání spojení. Test z dostupných informací odvodí, kolik žádostí bylo od útočníka a kolik útočných SYN paketů, tedy kolik procent celého útoku bylo Protectorem zablokováno. Celkový počet útočných SYN paketů je vypočten jako $(delka\ testu) * (sila\ toku)$. Klient úspěšně navázal 20016 spojení a byl tedy úspěšný ve všech pokusech o spojení. Z pohledu klienta je spojení úspěšně navázáno, pokud od serveru přijme paket s TCP příznaky SYN+ACK. Server z 2 miliónů pokusů úspěšně navázal 20016 spojení. Na straně serveru toto číslo znamená, že server od žadatele dostal ACK paket potvrzující navázání spojení (třetí krok třícestného handshaku). Poslední čítač na straně serveru informuje o počtu neúspěšně ukončených spojení, tzn. spojení, kdy žadatel buďto odeslal RST paket nebo neodpovídá na SYN+ACK paket. Toto číslo je identické s počtem SYN paketů od útoč-

níka, protože útočník z principu TCP SYN flood útoků neodpovídá. U klienta se objevuje čítač, který informuje o počtu, kdy klient muset opakovat odeslání SYN paketů z důvodu vypršení časovače pro odpověď serveru. Výpis informuje o 320 opakovaných odeslání. Strategie RST-Cookies zahazuje první SYN paket od každého klienta, dokud se neprokáže jako legitimní. Test operuje s 254 klienty, proto tento čítač musí být na hodnotě alespoň 254. Reset whitelist čítače je nastaven na 30 sekund, déle než délka testu (20 sekund) a není tak důvod očekávat dalších 254 znovuodeslání. Zbylých 66 (320-254) znovuodeslání je přirozený jev TRex generátoru. Při experimentech, kdy nebyl generován žádný útok bylo i tak naměřeno asi 100 znovuodeslání, proto hodnota 320 je z pohledu testu v pořádku. Poslední čítač u klienta informuje o počtu, kdy klient nebyl schopen spojení navázat i přes opakované znovuodeslání SYN paketů. Taková situace může nastat například pokud Protector neprovádí směrování správně, nebo pokud Protector neprovádí mitigaci. Server pak bude zahlcen pakety od útočníka a nebude schopen zpracovat pakety od klienta. Získané hodnoty odpovídají očekávanému výsledku a scénář skončí s úspěchem.

Další 2 případy využívají strategii SYN-Drop. Zkrácený výpis vypadá následovně:

```
== SYN-Drop: 1k connections/seconds traffic + 1Mpps SYN flood; 100k threshold ==
...
        Adding SYN rule: network="10.0.1.0/26" strategy=syn-drop threshold_packets
    =100000 host_syn_hard_threshold=30
...
Client
connections initiated (retransmits not included): 20016
connections established (SYN+ACK received): 20016
SYN packets retransmited: 606
embryonic connections dropped (could not establish connection): 0
...
== SYN-Drop: 4k connections/seconds traffic + 1Mpps SYN flood; 100k threshold ==
...
        Adding SYN rule: network="10.0.1.0/26" strategy=syn-drop threshold_packets
    =100000 host_syn_hard_threshold=119
...
Client
connections initiated (retransmits not included): 80010
connections established (SYN+ACK received): 80010
SYN packets retransmited: 293
embryonic connections dropped (could not establish connection): 0
...
```

Výpis 6.5: Zkrácený textový výpis druhého a třetího testovacího případu TCP SYN flood testu využívající SYN-Drop strategii.

První varianta generuje 1000 CPS, stejně jako u předchozího scénáře. Druhá varianta generuje 4000 CPS. U SYN-Drop strategie je třeba přizpůsobit parametr Host-syn-hard-threshold podle nastavené hodnoty CPS. Čím vyšší CPS, tím více SYN paketů za sekundu bude klient generovat. Protože klient nemá být zablokován, je hodnota Host-syn-hard-threshold vypočtena jako

$$(CPS/(pocet\ klientu) * (cas\ periodickeho\ resetu\ citace\ SYN\ paketu) + 1) * K$$

První část výrazu vypočítá průměrný počet SYN paketů generovaný jedním klientem za sekundu. Reset čítače SYN paketů je ve výchozím nastavení 5 sekund. Zahazování začne již v momentě, kdy se počet SYN paketů rovná hodnotě host-syn-hard-threshold (tzn. >=), proto je výsledná hodnota zvýšena o 1, aby klient nebyl blokován. Náhodnost výběru IP adres u generovaných paketů (viz 6.2) způsobí nedokonalé uniformní rozložení a proto

předpokládaný výpočet ($CPS/(pocet\ klientu)$) nebude úplně přesný. Kvůli tomu je celý výraz ještě násoben nějakou konstantou K (K > 1.0), která zaručuje míru tolerance. Experimentálně bylo K zvoleno jako 1.5, tedy tolerance 50 %. Strategie SYN-Drop podobně jako RST-Cookies zahazuje první SYN paket. Reset čítače ACK paketů je ve výchozím stavu nastaven na 20 sekund. Počet opakovaných vyslání SYN paketů tak může být až $((1 + (20/20)) * 254) + 150 = 658$. Číslo +150 je kvůli nepřesnosti TRex plánovače, jak bylo popsáno u RST-Cookies výše. Počet opakovaných vyslání ale může být i menší. Pokud reset čítače ACK paketů proběhne uprostřed probíhající komunikace (tzn. spojení již bylo navázáno), tak minimálně při ukončování aktuálního spojení klient pošle jeden ACK paket jako potvrzení FIN paketu od serveru. V takovém případě pak první SYN paket blokován nebude. Oba scénáře končí úspěchem.

Tento test odhalil dvě chyby v implementaci SYN-Drop submodulu. První se týkala datového typu hodnot Host-syn-soft-threshold a Host-syn-hard-threshold. V databázi, kam se pravidla nahrávají, jsou hodnoty typu `INT` a lze do nich zadat 32bitovou hodnotu. Ve zdrojových kódech Protectoru v jazyku C jsou hodnoty deklarovány jako typ `uint8_t`, jehož maximální hodnota je 255. Pokud se generuje velké množství CPS, pak hodnota host-syn-hard-threshold dle vztahu popsaném výše jistě překročí hodnotu 255. V databázi bude vše v pořádku, ale uvnitř Protectoru dojde k přetečení. Druhá chyba spočívala v práci s resetem čítačů ACK a SYN paketů. Každý čítač byl resetován, až když čas od posledního resetu byl větší než (>) nastavená hodnota v sekundách. Pokud se reset prováděl každých 20 sekund (případ ACK čítače), pak časová posloupnost resetů byla 0, 20, 41, 62 atd.. U SYN čítače pak 0, 5, 11, 17, 23 atd.. V čase 20 se resetuje čítač ACK paketů a do resetu SYN paketů zbývají 3 sekundy. Pokud je hodnota host-syn-soft-threshold nastavena nízko (nebo hodnota CPS vysoko), pak klient od času 17 poslal dostatek SYN paketů na to, aby jejich počet v čase 20 přesáhl hodnotu host-syn-soft-threshold. A podle SYN-Drop tabulky (2.1) pak takový host s nulovým počtem ACK paketů bude blokován, přestože se v testech jedná o legitimního klienta a veškerého hodnoty jsou nastaveny správně. Oprava této chyby spočívala v použití porovnání >= místo >.

Čtvrtý případ testuje ACK-Spoofing strategii. Zkrácený výpis vypadá následovně:

```
== ACK-Spoofing: 20 connections/seconds traffic + 10pps SYN flood; 1pps threshold ==
...
        Adding SYN rule: network="10.0.1.0/26" strategy=ack-spoofing threshold_packets
    =1
...
Client
connections initiated (retransmits not included): 414
connections established (SYN+ACK received): 414
SYN packets retransmited: 1
embryonic connections dropped (could not establish connection): 0
Server
connections accepted (SYN received): 614
   - attack SYNs that got through: 200
   - mitigated: 0, which is 0% of attack
connections established (ACK received): 414
connections dropped (received RST or client does not respond): 200

--------------------------------
-- Test case result: FAILED   --
-- Reason: Number of established server connections (414) did not match expected value
     (attack(200)+client(414)=614).
--------------------------------
```

Výpis 6.6: Zkrácený textový výpis čtvrtého testovacího případu TCP SYN flood testu využívající ACK-Spoofing strategii.

Síla generovaného provozu je v tomto testu výrazně snížena. Server přijal 614 SYN paketů, přičemž 200 bylo od útočníka. Úspěšně navázal jen 414 spojení, což nesouhlasí s očekávaným výsledkem. Strategie ACK-Spoofing totiž zařízením ve chráněné síti v reakci na zachycený ACK+SYN paket posílá jako odpověď podvržené ACK pakety. Očekávaný výsledek je, že server úspěšně naváže všech 614 přijatých pokusů o spojení. Test z tohoto důvodu selže a vypíše důvod. Po diskuzi s členy vývojového týmu DDoS Protectoru bylo usneseno, že strategie ACK-Spoofing je v současné verzi Protectoru nefunkční. Zbylé 4 případy jsou IPv6 verze předchozích případů a jejich výsledky jsou identické.

6.4 Multi-vector útoky

Testy implementují případy 8, 9 a 10 z podkapitoly 4.4. Jako zástupce těchto testů je prezentován výstup případu 10:

```
== Stateful multivector mitigation 1: 1k CPS traffic+16Mpps attack[DNS+ICMP+UDP+SYN,
    4:4:6:2 ratio]; Limits: 200k,100k,1M,rst-cookies ==
...
        Adding AMP rule: network="10.0.1.0/26" protocol=17 dst_to=53 threshold_packets
    =1000000 limit_packets=200000
        Adding AMP rule: network="10.0.1.0/26" protocol=1 threshold_packets=500000
    limit_packets=100000
        Adding AMP rule: network="10.0.1.0/26" protocol=17 threshold_packets=3000000
    limit_packets=1000000
        Adding SYN rule: network="10.0.1.0/26" strategy=rst-cookies threshold_packets
    =500000
...
Results: Expected to receive: 22080k-54080k packets, actually received: 27568k packets
    .
DNS amp flood: Percentual block of attack packets: Expected: 77.00%-87.00%, actually
    blocked: 84.90%.
ICMP flood: Percentual block of attack packets: Expected: 86.00%-96.00%, actually
    blocked: 93.40%.
UDP flood: Percentual block of attack packets: Expected: 22.00%-32.00%, actually
    blocked: 23.55%.
TCP SYN flood: Percentual block of attack packets: Expected: 100.00%-100.00%, actually
     blocked: 100.00%.
Client
connections initiated (retransmits not included): 20016
connections established (SYN+ACK received): 20016
SYN packets retransmited: 2023
embryonic connections dropped (could not establish connection): 0
Server
connections accepted (SYN received): 1361298
   - attack SYNs that got through: 1341282
   - mitigated: 38658718.0, which is 96% of attack
connections established (ACK received): 20016
connections dropped (received RST or client does not respond): 1341282

--------------------------------
-- Test case result: SUCCESS  --
--------------------------------
```

Výpis 6.7: Textový výpis multi-vector testu se stavovým provozem.

Při generování více typů útoku je třeba provést kontrolu blokace útočných paketů pro všechny útoky. Všechny pokusy o navázání spojení klienta se serverem byly úspěšně. Počet opakovaných znovuodeslání SYN paketů je výrazně vyšší, než u samotného testu strategie RST-Cookies. Důvodem je počáteční zahlcení serveru útokem o síle 16 Mpps, kvůli kterému server není schopen zpracovat počáteční pokusy o spojení. Reakce Protectoru může trvat 1 až 2 sekundy. Z výpisu si lze všimnout, že naměřené procento blokace SYN paketů je 100 %, ačkoliv se na server dostalo asi 1.3 miliónů útočných SYN paketů. Důvod je ten, že zachytávání paketů probíhá v době, kdy už je mitigace aktivní. Multi-vector útoky odhalily jednu chybu v implementaci Protectoru. Jednalo se o špatný výpočet dostupných zdrojů pro blokaci IP adres. Tato chyba se projevila celkem nečekaným způsobem: pořadí pravidel v databázi mělo vliv na mitigační proces. Při prohození pořadí dvou pravidel již mitigace nemusela správně fungovat.

6.5 HTTP flood útok

Test implementuje jediný testovací případ. Výstup vypadá následovně:

```
== HTTP-flood 1: 1k connections/seconds traffic + 50k CPS HTTP flood, 4.2kpps limit (1 CPS = 4 packets) ==
...
        Adding AMP rule: network="10.0.1.0/26" protocol=6 threshold_packets=5000 limit_packets=4200
...
Client
connections initiated (retransmits not included): 20016
connections established (SYN+ACK received): 20016
SYN packets retransmited: 3532
embryonic connections dropped (could not establish connection): 0
Server
connections accepted (SYN received): 82206
connections established (ACK received): 82062
connections dropped (received RST or client does not respond): 1401

--------------------------------
-- Test case result: SUCCESS  --
--------------------------------
```

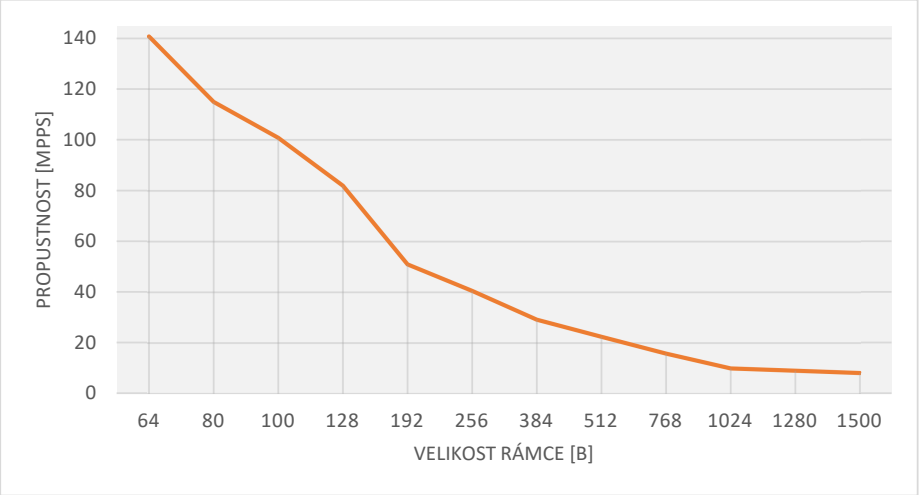
Výpis 6.8: Textový výpis HTTP flood testu.

Provoz mezi klientem a serverem je 1000 CPS. Útok generuje 50000 CPS. V testu je použito pravidlo amplifikačního modulu, které limituje veškerý TCP provoz na určený limit. Z analýzy generovaného provozu v programu Wireshark bylo zjištěno, že při každém spojení klient generuje 4 pakety (SYN, HTTP REQUEST (PSH+ACK), FIN, ACK). Proto je limit nastaven na hodnotu 4200 pps, kdy 200 dodatečných paketů je prostor pro případné opakované přenosy. Z čítačů serveru lze vyčíst, že přijal 82000 žádostí a úspěšně jich navázal téměř stejný počet, přičemž klient poslal pouze 20016 žádostí o spojení. Reakce Protectoru tak trvala asi 1.2 sekundy.

6.6 Měření propustnosti

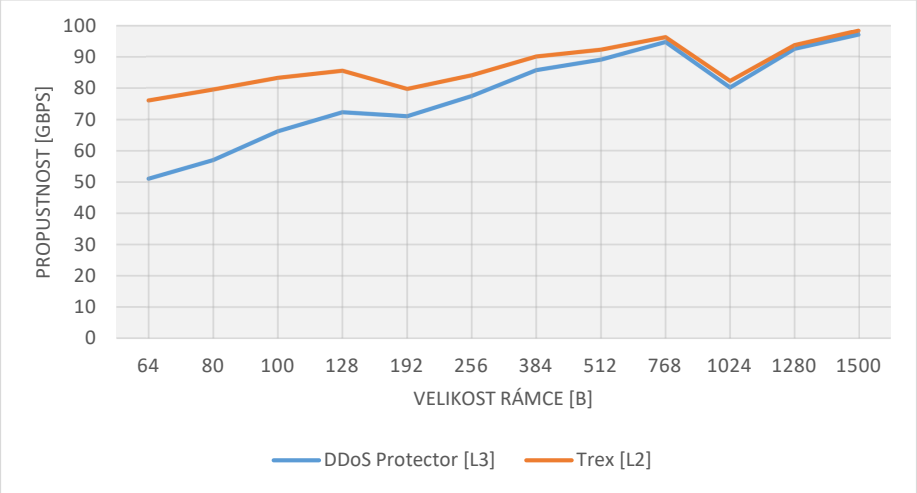
Použité testovací prostředí je momentálně limitováno pouze jedním 100 Gbps spojem (podkapitola 6.1). Měření propustnosti probíhalo mezi vysílacím rozhraním (TX) TRex útočníka a přijímacím rozhraním (RX) Protectoru. Přijaté pakety Protector odesílá ze svého TX rozhraní na RX rozhraní TRex serveru, ale protože TRex server je připojen pouze 10 Gbps spojem (případně 40 Gbps spojem, pokud se klient a server prohodí), není měření z této strany zajímavé z pohledu výkonnosti. Pro dosažení nejlepších výsledků muselo být TRex instanci generující útok přiřazeno 16 vláken oproti původním 8. Obě instance byly spuštěny v bezstavovém (*stl*) režimu. První graf 6.2 zachycuje počet přijatých paketů za sekundu v závislosti na velikosti rámce. Při nejmenší testované velikosti rámce 64 B je dosaženo hodnoty 140 Mpps. To je velmi blízko k hodnotě 150 Mpps – maximu, které je Protector schopný zpracovat. Zároveň to ukazuje skutečnost, že i TRex je schopný generovat 140 Mpps s použitou síťovou kartou Mellanox. S rostoucí velikostí rámce počet paketů přirozeně klesá, až na hodnotu ∼8 Mpps u délky paketu 1500 B.

Druhý graf 6.3 zachycuje počet odeslaných bitů (oranžová křivka, TRex) a počet přijatých bitů (modrá křivka, Protector) za sekundu v závislosti na velikosti rámce. V tomto grafu se projevuje rozdílný výpočet hodnoty bps (viz 5.3) mezi TRexem, který ji počítá



Obrázek 6.2: Graf propustnosti v paketech za sekundu v závislosti na velikosti rámce.

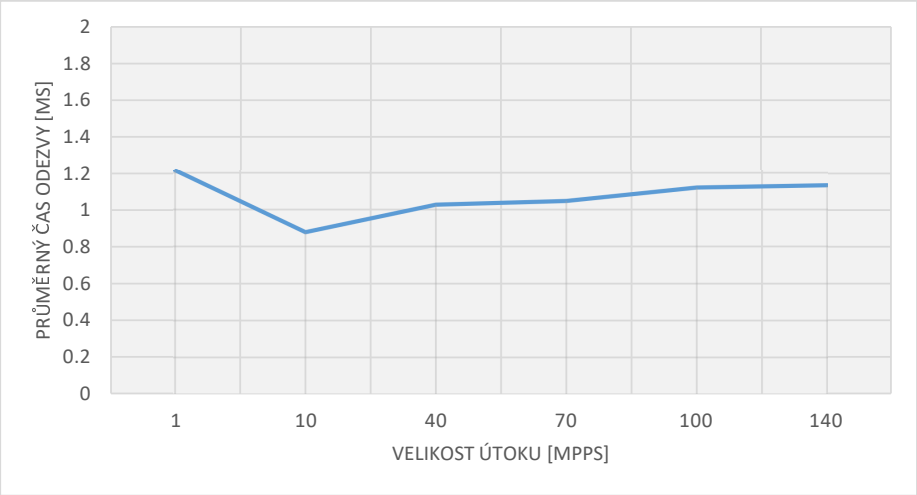
na L2 vrstvě a Protectorem, který ji počítá na L3 vrstvě. Při nejmenší velikosti rámce 64 B je rozdíl největší, kdy TRex generuje 76 Gbps a Protector detekuje 50 Gbps. Přepočet hodnoty bps z L2 na L3 se zde převede vynásobením hodnotou, která je dána vztahem $(64\,B - 22\,B)/64\,B = 0.65625$, kde 64 B je velikost rámce a 22 B je velikost L2 hlavičky (6 B zdrojové MAC adresy, 6 B cílové MAC adresy, 4 B VLAN, 2 B EtherType a 4 B kontrolního součtu). Pak $76\,Gbps * 0.65625 = 49,8\,Gbps$ což odpovídá naměřené hodnotě modré křivky v grafu. Tedy to, že se křivky v grafu liší **ne**znamená, že by se bity po cestě ztrácely, nebo že by je Protector nebyl schopen zpracovat. Generovaný objem a přijatý objem bitů je identický, pouze se liší síťová vrstva, na které se objem počítá. S rostoucí velikostí rámce se rozdíly zmenšují, protože počet paketů klesá a tudíž klesá i režie L2 hlavičky. U 768 B rámce je rozdíl už velice malý $((768\,B - 22\,B)/768\,B = 0.97)$ a zároveň se propustnost pohybuje velmi blízko maximální kapacity. V grafu je zaznamenán pokles propustnosti u velikosti rámce 1024 B. Důvod poklesu nebyl identifikován.



Obrázek 6.3: Graf propustnosti v bitech za sekundu v závislosti na velikosti rámce. Graf obsahuje dvě křivky. Oranžová křivka zobrazuje propustnost ze strany TRexe (vysílání) a modrá křivka ze strany Protectoru (příjem).

6.7 Měření latence

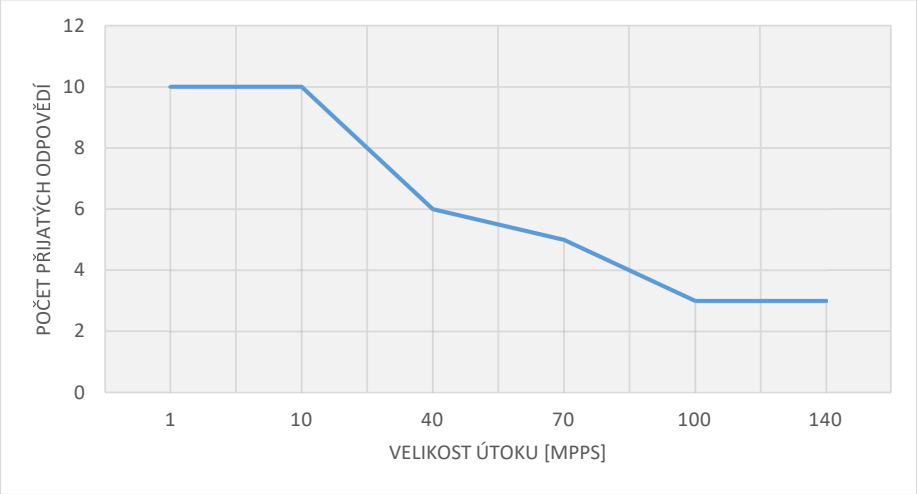
Implementace odpovídá původnímu návrhu. Výsledky jsou zobrazeny na grafu 6.4. Průměrná doba odezvy byla menší než dvě milisekundy pro všechny intenzity útoku od 1 Mpps až do 140 Mpps. Threshold pravidla je nastaven tak, aby byl spuštěn sběr informací o IP adresách (zahrnuje prohledávání a vkládání do hash tabulky), ale nebyla spuštěna mitigace. Protector tento test zvládl dobře.



Obrázek 6.4: Graf průměrné latence v závislosti na velikosti útoku.

Nad rámec původního testu byl experimentálně zvýšen počet pravidel z jednoho na 20. Pak se výrazně sníží počet paketů, které Protector dokáže zpracovat (např. z 140 Mpps na 47 Mpps) a zároveň pro většinu požadavků vyprší timeout pro odpověď. Ovšem odpovědi, které dorazí mají latenci stále pod dvě milisekundy. Graf 6.5 zobrazuje počet přijatých odpovědi (z 10 požadavků) v závislosti na velikosti útoku. Od velikosti útoku 40 Mpps

již některé odpovědi nedorazí a s rostoucí silou útoku se průměrně sníží počet přijatých odpovědí. Shrnutí tedy je, že v tomto testu si Protector vedl dobře, ale pokud se nad rámec původního návrhu zvýší nároky, například zvýšením počtu pravidel, pak se již objeví problémy s latencí.



Obrázek 6.5: Graf počtu odpovědí v závislosti na velikosti útoku při 20 pravidlech v databázi Protectoru (z 10 požadavků).

6.8 Zátěžové testy strategií RST-Cookies a SYN-Drop

Testy mají velmi podobnou strukturu s testy popsanými v podkapitole 6.3. Test nejdříve vygeneruje dostatečné množství útoku na to, aby aktivoval mitigaci. Cílem těchto testů není zahltit TRex server (to by naopak mohlo přinést potíže s vyhodnocováním), ale DDoS Protector. Poté se začne generovat útok o zadané velikosti. Po ukončení generování se scénář vyhodnotí standardním způsobem, tzn. jestli byl útok mitigován, jestli žádosti klientů nebyly blokovány a jestli odpovídá počet znovuodeslání SYN paketů. Každý test obsahuje 13 případů s postupně se zvyšující silou útoku. Výsledky jsou prezentovány v tabulce 6.1. Z výsledků je patrný rozdíl mezi výkonností obou strategií. Strategie RST-Cookies začne lehce selhávat od útoku velikosti 2.4 Mpps, od útoku velikosti 4 Mpps již zahazuje klientské pakety. Strategie SYN-Drop naopak úspěšně mitiguje všechny intezity útoku do 90 Mpps. Ačkoliv předchozí podkapitola naměřila až 140 Mpps, v tomto testu byl TRex útočník schopný generovat útok o velikosti maximálně 95 Mpps. Vliv má pravděpodobně odlišná definice profilu útoku. Důvod výrazného rozdílu mezi strategiemi nebyl přesně identifikován, ale za možnou příčinu se považuje počítání kryptograficky bezpečné funkce při vytváření paketu u strategie SYN-Cookies. Strategie SYN-Drop se rozhoduje pouze na základě tabulky a sama nemá žádnou režii s vytvářením a odesíláním paketů.

Síla útoku [Mpps]	Výsledek [RST-Cookies]	Síla útoku [Mpps]	Výsledek [SYN-Drop]
1	OK	1	OK
2	OK	2	OK
2.1	OK	3	OK
2.2	OK	5	OK
2.3	OK	10	OK
2.4	Počet opakovaných spojení (663)	20	OK
2.5	Počet opakovaných spojení (765)	30	OK
3	Počet opakovaných spojení (5948)	40	OK
4	Blokování klientů (úspěšnost 88 %)	50	OK
5	Blokování klientů (úspěšnost 81 %)	60	OK
10	Blokování klientů (úspěšnost 58 %)	70	OK
20	Blokování klientů (úspěšnost 55 %)	80	OK
50	Blokování klientů (úspěšnost 47 %)	90	OK

Tabulka 6.1: Naměřené výsledky výkonnosti strategií RST-Cookies and SYN-Drop.

6.9 Přehled výsledků

Tato podkapitola obsahuje tabulku 6.2 shrnující výsledky testů. Jednotlivé testy byly detailněji popsány v předchozích podkapitolách. Vytvořené prostředí v rámci CVIS laboratoře VUT je funkční. Při testování byly odhaleny některé chyby v implementaci DDoS Protectoru. Prezentované výsledky byly vytvořeny při použití interní, opravené verze DDoS Protectoru. Vydání nové, stabilní verze Protectoru s opravou chyb může zabrat nějaký čas.

Další pokračování práce by mohlo spočívat v implementování nových testů a rozšíření stávající implementace. První rozšíření by mohla být implementace Slowloris útoku. Dle původního návrhu by se použil generátor `SlowHTTPTest`. Program ovšem nemá nativní podporu vzdáleného ovládání, ta by se musela vytvořit manuálně. Dále by bylo třeba ověřit správnou interakci mezi programem a TRex serverem. Nové multi-vector testy by mohly obsahovat současně více než 4 typy útoků. Dále vytvoření testů pro ověření více položek pravidel, jako například délky paketu (generování paketů proměnlivé délky) nebo fragmentace. Podpora IPv6 variant pro všechny testy. Rozšíření prostředí o směrovače, aby se záznamy do směrovací tabulky Protectoru instalovaly dynamicky.

Test	Odpovídá návrhu 4.4	Výsledek
UDP flood [1:5]	1	OK
UDP flood [1:1]	2	OK
UDP flood [25000 adres]	3	Nedostatek HW zdrojů
TCP SYN flood [RST-Cookies]	4	OK
TCP SYN flood [SYN-Drop]	5	OK
TCP SYN flood [Ack-Spoofing]	6	Strategie nefunkční
Zátěžový test [RST-Cookies]	7	∼2.3 Mpps
Zátěžový test [SYN-Drop]	7	90+ Mpps
Multi-vector útoky [2 útoky]	8	OK
Multi-vector útoky [4 útoky]	9	OK
Multi-vector útoky [stavový provoz]	10	OK
HTTP flood	11	OK
Slowloris	12	Neimplementován
Propustnost	13	140 Mpps, 98 Gbps
Latence ping	14	Latence neovlivněna

Tabulka 6.2: Souhrn výsledků testů.

Kapitola 7

Závěr

Cílem práce byl návrh a implementace rozšiřitelného prostředí pro testování zařízení DDoS Protector včetně nezbytné sady testů ověřující funkci i výkonnost zařízení. V rámci teoretického rozboru byla rozebrána problematika DoS útoků, zejména jejich dělení a bližší popis vybraných typů útoků. Následoval popis zařízení DDoS Protector. Důraz byl kladen na způsob, kterým útok detekuje a mitiguje a na popis pravidel pro detekci útoků. Poslední část se věnuje analýze dostupných generátorů stavového provozu se zaměřením na Cisco TRex a Juniper WARP17. V rámci návrhu a implementace byl zvolen Cisco TRex jako generátor legitimního provozu i generátor útoku díky jeho flexibilitě, vysoké propustnosti a dobré dokumentaci. Bylo navrženo 14 testovacích případů, skládající se z různých typů útoků a legitimního provozu. Jejich implementace byla navržena jako rozšíření stávajícího frameworku automatizovaných testů DDoS Protectoru. Nakonec byly popsány implementační detaily prostředí i testů. V poslední fázi bylo provedeno testování a vyhodnocení DDoS Protectoru. Výsledky testů byly důkladně analyzovány. Velká většina testů prochází úspěšně. U neúspěšných testů byla zjištěna příčina. Z pohledu výkonnosti byla otestována propustnost, latence a modul provádějící detekci a mitigaci TCP SYN flood útoků. Díky testům byly odhaleny některé chyby v implementaci DDoS Protectoru. V rámci pokračování práce bude probíhat integrace vytvořených testů do interního repozitáře DDoS Protectoru. Vytvořené prostředí se bude prakticky používat při dalším vývoji DDoS Protectoru pro testování nových strategií mitigace a dále rozvíjet v rámci navazující spolupráce se sdružením CESNET.

Literatura

[1] 0xbharath. *Creating packets with Scapy* [online]. [cit. 2019-11-18]. Dostupné z: https://0xbharath.github.io/art-of-packet-crafting-with-scapy/scapy/creating_packets/index.html.

[2] Baudy, J. a Camaró, U. A. *PACKET_MMAP* [online]. [cit. 2020-01-13]. Dostupné z: https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt.

[3] Burzala, M. *Testování zařízení pro ochranu před DoS útoky*. 2019. [cit. 2020-01-19]. Bakalářská práce. Vysoké učení technické v Brně. Dostupné z: https://www.fit.vut.cz/study/thesis-file/21651/21651.pdf.

[4] CESNET. *Ddos-protector Gitlab repository* [online]. [cit. 2020-05-20]. Dostupné z: https://gitlab.liberouter.org/ddos-protector/dcpro/-/tree/devel/tests.

[5] CESNET. *Ddos-protector Gitlab repository* [online]. [cit. 2020-05-22]. Dostupné z: https://gitlab.liberouter.org/ddos-protector/dcpro/-/blob/devel/sw/daemon/amp-module.c.

[6] CESNET. *Framework pro tvorbu automatických testů* [online]. [cit. 2020-05-20]. Dostupné z: https://redmine.liberouter.org/projects/ddos-protector/wiki/Testov%C3%A1n%C3%AD_%C4%8Disti%C4%8Dky.

[7] CESNET. *DDoS Protector User's Manual*. 2019.

[8] Chang, L. *Anonymous is behind those massive cyberattacks in Turkey* [online]. [cit. 2020-01-15]. Dostupné z: https://www.digitaltrends.com/computing/anonymous-behind-turkey-cyberattakcs/.

[9] Chhabra, M., Gupta, B. a Almomani, A. *A Novel Solution to Handle DDOS Attack in MANET* [online]. Journal of Information Security, 2013 [cit. 2020-01-14]. DOI: 10.4236/jis.2013.43019. Dostupné z: http://dx.doi.org/10.4236/jis.2013.43019.

[10] Cisco. *A Cisco Guide to Defending Against Distributed Denial of Service Attacks* [online]. [cit. 2020-01-15]. Dostupné z: https://tools.cisco.com/security/center/resources/guide_ddos_defense.

[11] Cisco. *TRex* [online]. [cit. 2019-11-18]. Dostupné z: https://trex-tgn.cisco.com/.

[12] Cisco. *What Is a DDoS Attack?* [online]. [cit. 2020-01-15]. Dostupné z: https://www.cisco.com/c/en/us/products/security/what-is-a-ddos-attack.html#~types-of-ddos-attacks.

[13] Cloudflare. *DNS Amplification Attack* [online]. [cit. 2020-01-16]. Dostupné z: https://www.cloudflare.com/learning/ddos/dns-amplification-ddos-attack/.

[14] Cloudflare. *HTTP Flood Attack* [online]. [cit. 2020-01-16]. Dostupné z: https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/.

[15] Cloudflare. *Ping (ICMP) Flood DDoS Attack* [online]. [cit. 2020-01-16]. Dostupné z: https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/.

[16] Cloudflare. *Slowloris DDoS Attack* [online]. [cit. 2020-01-16]. Dostupné z: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/.

[17] Cloudflare. *UDP Flood Attack* [online]. [cit. 2020-01-16]. Dostupné z: https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/.

[18] DPDK Project. *About DPDK* [online]. [cit. 2020-01-13]. Dostupné z: https://www.dpdk.org/about/.

[19] DPDK Project. *Programmer's Guide - overview* [online]. [cit. 2020-01-13]. Dostupné z: https://doc.dpdk.org/guides/prog_guide/overview.html.

[20] DPDK Project. *Programmer's Guide - Poll Mode Driver* [online]. [cit. 2020-01-13]. Dostupné z: https://doc.dpdk.org/guides/prog_guide/poll_mode_drv.html.

[21] García Dorado, J. L., Mata, F., Ramos, J., Río, P. M. S. del, Moreno, V. et al. *High-Performance Network Traffic Processing Systems Using Commodity Hardware* [online]. Springer, Berlin, Heidelberg, 2013 [cit. 2019-01-13]. Dostupné z: https://doi.org/10.1007/978-3-642-36784-7_1.

[22] Goldschmidt, P. *Heuristické metody pro potlačení DDoS útoků zneužívajících protokol TCP*. 2018. [cit. 2020-01-18]. Bakalářská práce. Vysoké učení technické v Brně. Dostupné z: https://www.fit.vut.cz/study/thesis-file/21711/21711.pdf.

[23] Hsu, F.-H., Hwang, Y.-L., Tsai, C.-Y., Cai, W.-T., Lee, C.-H. et al. *TRAP: A Three-Way Handshake Server for TCP Connection Establishment* [online]. Appl. Sci., 2016 [cit. 2020-01-15]. Dostupné z: https://doi.org/10.3390/app6110358.

[24] IBM. *Attack events* [online]. [cit. 2020-01-16]. Dostupné z: https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzaub/rzaubeventattack.htm.

[25] Imperva. *DDoS Attacks* [online]. [cit. 2020-01-15]. Dostupné z: https://www.imperva.com/learn/application-security/ddos-attacks/.

[26] Information Sciences Institute, University of Southern California. *TRANSMISSION CONTROL PROTOCOL* [Internet Requests for Comments]. RFC 793. RFC Editor. Dostupné z: https://www.ietf.org/rfc/rfc793.txt.

[27] Ixia. *IXLOAD OVERVIEW — CONVERGED MULTIPLAY SERVICE VALIDATION* [online]. [cit. 2020-01-11]. Dostupné z: https://www.ixiacom.com/sites/default/files/2019-07/Ixia-T-DS-IxLoad-Overview.pdf.

[28] Ixia. *IXNETWORK OVERVIEW — L2/3 NETWORK INFRASTRUCTURE PERFORMANCE TESTING* [online]. [cit. 2020-01-11]. Dostupné z: https://www.ixiacom.com/sites/default/files/2019-10/Ixia-T-DS-IxNetwork.pdf.

[29] Juniper Networks. *build dpdk.sh* [online]. [cit. 2019-11-25]. Dostupné z: https://github.com/Juniper/warp17/blob/dev/common/build_dpdk.sh#L127.

[30] Juniper Networks. *Performance benchmarks* [online]. [cit. 2019-11-25]. Dostupné z: https://github.com/Juniper/warp17/blob/dev/common/doc/Performance.md.

[31] Juniper Networks. *test 1 raw tcp connection.cfg* [online]. [cit. 2019-11-25]. Dostupné z: https://github.com/Juniper/warp17/blob/dev/common/examples/test_1_raw_tcp_connection.cfg.

[32] Juniper Networks. *WARP17, The Stateful Traffic Generator for L1-L7* [online]. [cit. 2019-11-25]. Dostupné z: https://github.com/Juniper/warp17.

[33] Kali Linux Penetration Testing Tools. *hping3 Package Description* [online]. [cit. 2020-01-20]. Dostupné z: https://tools.kali.org/information-gathering/hping3.

[34] Kali Linux Penetration Testing Tools. *SlowHTTPTest Package Description* [online]. [cit. 2020-01-20]. Dostupné z: https://tools.kali.org/stress-testing/slowhttptest.

[35] Kupreev, O., Badovskaya, E. a Gutnikov, A. *DDoS attacks in Q2 2019* [online]. [cit. 2020-01-15]. Dostupné z: https://securelist.com/ddos-report-q2-2019/91934/.

[36] Liberouter. *Cards* [online]. [cit. 2020-01-18]. Dostupné z: https://www.liberouter.org/technologies/cards/.

[37] Linux man-pages. *packet(7)* [online]. [cit. 2020-01-13]. Dostupné z: http://man7.org/linux/man-pages/man7/packet.7.html.

[38] Marchbank, C. a Stone, B. *Criminal Motivations: One Big Reason DDoS Attacks Are Exploding in Popularity* [online]. [cit. 2020-01-15]. Dostupné z: https://staysafeonline.org/blog/criminal-motivations-one-big-reason-ddos-attacks-exploding-popularity/.

[39] ntop. *Drivers and Modules* [online]. [cit. 2020-01-13]. Dostupné z: https://www.ntop.org/guides/pf_ring/modules/index.html.

[40] ntop. *PF_RING* [online]. [cit. 2020-01-13]. Dostupné z: https://www.ntop.org/products/packet-capture/pf_ring/.

[41] ntop. *PF_RING ZC (Zero Copy)* [online]. [cit. 2020-01-13]. Dostupné z: https://www.ntop.org/guides/pf_ring/zc.html.

[42] ntop. *PF_RING ZC (Zero Copy)* [online]. [cit. 2020-01-13]. Dostupné z: https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/.

[43] ntop. *Shop* [online]. [cit. 2020-01-13]. Dostupné z: https://shop.ntop.org/.

[44] Odehnal, T. *Mitigace DoS útoků s využitím neuronových sítí*. 2018. [cit. 2020-01-18]. Bakalářská práce. Vysoké učení technické v Brně. Dostupné z: https://www.fit.vut.cz/study/thesis-file/21654/21654.pdf.

[45] Prince, M. *Deep Inside a DNS Amplification DDoS Attack* [online]. CLOUDFLARE, 2012 [cit. 2020-01-15]. Dostupné z: https://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack/.

[46] Rijswijk Deij, R. van, Sperotto, A. a Pras, A. *DNSSEC and its potential for DDoS attacks: a comprehensive measurement study* [online]. ACM Press, 2014 [cit. 2020-01-16]. Dostupné z: https://doi.org/10.1145%2F2663716.2663731.

[47] Spirent. *Spirent Test Modules* [online]. [cit. 2020-01-11]. Dostupné z: https://www.spirent.com/products/testcenter/platforms/modules.

[48] Spirent. *Spirent TestCenter API* [online]. [cit. 2020-01-11]. Dostupné z: https://www.spirent.com/-/media/datasheets/broadband/pab/spirenttestcenter/stc_api_tcl-perl-java-c-ruby_datasheet.pdf.

[49] Spirent. *Spirent TestCenter Enhanced L4-7* [online]. [cit. 2020-01-11]. Dostupné z: https://www.spirent.com/-/media/datasheets/broadband/pab/spirenttestcenter/ds-spirent-testcenter-enhanced-l4-7.pdf.

[50] Spirent. *Spirent TestCenter REST API* [online]. [cit. 2020-01-11]. Dostupné z: https://www.spirent.com/-/media/datasheets/broadband/pab/spirenttestcenter/spirent_testcenter_rest_api_datasheet.pdf.

[51] Sucuri. *What is a DDoS Attack?* [online]. [cit. 2020-01-15]. Dostupné z: https://sucuri.net/guides/what-is-a-ddos-attack/.

[52] Swati, K. *World's largest 1 Tbps DDoS Attack launched from 152,000 hacked Smart Devices* [online]. [cit. 2020-01-19]. Dostupné z: https://thehackernews.com/2016/09/ddos-attack-iot.html.

[53] TRex team. *Client Package* [online]. [cit. 2020-05-20]. Dostupné z: https://trex-tgn.cisco.com/trex/doc/cp_docs/index.html#client-package.

[54] TRex team. *Comparing TRex Advanced Stateful performance to Linux NGINX* [online]. [cit. 2019-11-25]. Dostupné z: https://trex-tgn.cisco.com/trex/doc/trex_astf_vs_nginx.html.

[55] TRex team. *Source code for trex.astf.trex_astf_profile* [online]. [cit. 2020-01-20]. Dostupné z: https://trex-tgn.cisco.com/trex/doc/cp_astf_docs/_modules/trex/astf/trex_astf_profile.html#ASTFProgram.send_chunk.

[56] TRex team. *TRex Advance stateful support* [online]. [cit. 2019-11-18]. Dostupné z: https://trex-tgn.cisco.com/trex/doc/trex_manual.html#_basic_usage.

[57] TRex team. *TRex Manual - Basic usage* [online]. [cit. 2019-11-19]. Dostupné z: https://trex-tgn.cisco.com/trex/doc/trex_astf.html.

[58] TRex team. *TRex Stateless support* [online]. [cit. 2019-11-18]. Dostupné z: https://trex-tgn.cisco.com/trex/doc/trex_stateless.html.

[59] Wikipedia contributors. *Pcap — Wikipedia, The Free Encyclopedia.* [cit. 2020-01-13]. Dostupné z: https://en.wikipedia.org/wiki/Pcap.