VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

POTLAČENÍ DDOS ÚTOKŮ S VYUŽITÍM IDS/IPS

MITIGATION OF DDOS ATTACKS USING IDS/IPS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN LITWORA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN KUČERA**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Litwora Martin**
Program: Informační technologie
Název: **Potlačení DDoS útoků s využitím IDS/IPS**
Mitigation of DDoS Attacks Using IDS/IPS
Kategorie: Počítačové sítě

Zadání:

1. Nastudujte dostupnou literaturu o DDoS útocích a způsobech jejich potlačení. Seznamte se se systémy IDS/IPS pro detekci bezpečnostních hrozeb v síti.
2. Analyzujte možnosti využití existujících IDS/IPS systémů k potlačení DDoS útoků. Zaměřte se především na systém Suricata.
3. Navrhněte způsob potlačení vybraných DDoS útoků s využitím prostředků systému Suricata.
4. Navržený přístup implementujte a ověřte v laboratorním prostředí.
5. Vyhodnoťte vytvořené řešení z hlediska dosažených vlastností.
6. Diskutujte dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr 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 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Tato bakalářská práce se zaměřuje na detekci a následné potlačení útoků typu DDoS (útoky typu odepření služby). Hlavním záměrem bylo analyzovat a prakticky ověřit možnosti využití existujících IDS/IPS právě k potlačení DDoS útoků. Práce se soustředí především na open-source systém Suricata. V rámci práce jsou analyzovány tři významné skupiny DDoS útoků. Jsou to útoky záplavové, amplifikační a tzv. slow útoky. Pro jednotlivé zástupce těchto útoků a každý specifický typ útoku byla navržena řada pravidel pro systém Suricata, pomocí nichž lze dané útoky detekovat a následně i efektivně potlačit. Práce dále implementuje sadu nástrojů a skriptů pro ověření funkčnosti a účinnosti navržených pravidel. Nástroje integrují generátory vybraných útoků a umožňují jak konfiguraci jejich parametrů, tak i dalších parametrů systému Suricata a vyhodnocují tak dosažené vlastnosti navržených pravidel a systému. Testování probíhalo ve virtualizovaném prostředí. Sada nástrojů byla navržena tak, aby byla bez dalších úprav snadno přenositelná do reálného prostředí, kde lze testovat větší síťové zátěže, různé variace a kombinace skutečných systémů.

Abstract

This This bachelor's thesis focuses on the detection and mitigation of DDoS attacks (Distributed Denial of Service). The main goal is to analyze and practically verify the capabilities of various IDS/IPS, especially the open-source tool Suircata, to mitigate DDoS attacks. Three main DDoS attack groups are analyzed in this thesis. These groups are flood attacks, amplification attacks, and slow attacks. A set of rules has to be created for each attack type from these groups in order for Suricata to mitigate those DDoS attacks. This thesis also implements a set of tools and scripts to check the functionality and effectiveness of the created rules. These tools are used to generate selected DDoS attacks with different parameters. Testing took place in a virtual environment where special nodes had to be created which represent real subjects during a real DDoS attack. The set of tools and scripts was designed in a way that it can easily be used outside this virtual environment where it is possible to have larger network loads, various variants and combinations of systems, and more.

Klíčová slova

IDS, IPS, Suricata, DDoS, DoS, Amplifikační útoky, Volumetrické útoky, Pomalé útoky, Potlačení DDoS

Keywords

IDS, IPS, Suricata, DDoS, DoS, Amplification attacks, Volumetric attacks, Slow attacks, DDoS mitigation

Citace

LITWORA, Martin. *Potlačení DDoS útoků s využitím IDS/IPS*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kučera

Potlačení DDoS útoků s využitím IDS/IPS

Prohlášení

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

.....
Martin Litwora
10. května 2021

Poděkování

Chtěl bych poděkovat vedoucímu práce panu inženýrovi Janu Kučerovi za ochotu, trpělivost, odbornou pomoc a čas, který mi věnoval během tvorby této bakalářksé práce.

Obsah

Kapitola 1

Úvod

V dnešní době je internet všude kolem nás. Je na něm postaveno fungování mnoha služeb, firem i společnosti jako celku, zvláště pak v době, kdy celý svět zmítá pandemie okolo koronaviru. Odjakživa se různé skupiny lidí snažily narušit fungování webových potažmo síťových služeb a ani dnes tomu není jinak.

Útoky typu odepření služeb (DoS, DDoS) jsou jednou z nejčastějších voleb, jak zamezit, nebo alespoň ztížit přístup k internetovému obsahu. Tyto útoky mají za cíl zahltit danou službu natolik, aby se stala nepoužitelnou pro ostatní legitimní uživatele. Jako obrana proti DDoS útokům se používá celá řada nástrojů, například firewall, proxy servery, distribuce obsahu skrze CDN nebo použití detekčních systémů IDS/IPS. Tyto systémy slouží k monitorování síťového provozu a detekci různých hrozeb a škodlivého obsahu.

Hlavním cílem této práce je analyzovat IDS/IPS open-source nástroj Suricata v kontextu jeho využití pro potlačení DDoS útoků. Pro každý útok je nutné vytvořit seznam pravidel, podle nichž se bude testovat, zda Suricata dokáže eliminovat daný DDoS útok. V této práci jsou blíže popsány jednotlivé útoky typu odepření služby a jakými dalšími způsoby (kromě Suricaty) se lze proti nim bránit. Tyto útoky se dají rozdělit do tří významných skupin. První skupinu tvoří populární záplavové útoky, jejichž cílem je zaneprázdnit server velkým počtem nesmyslných dotazů. Do této skupiny patří především SYN, ACK, RST a UDP flood. Druhou skupinu tvoří amplifikační útoky, které se zaměřují především na zahlcení síťové linky velkým objemem dat. Mezi významné zástupce této skupiny se řadí amplifikační NTP, DNS, CLDAP, OpenVPN, Memcached a SSDP útok. Poslední skupinu tvoří tzv. pomalé útoky, které se snaží udržet spojení s cílovým serverem co nejdéle otevřené, a zamezit tak legitimním uživatelům využívat služby serveru. Mezi zástupce slow útoků patří například Slowloris a Slowread. Taktéž jsou v práci zmíněny alternativy k Suricatě a možnosti jejich různého umístění a nasazení.

K ověření správnosti a funkčnosti navržených pravidel je nutné vytvořit testovací prostředí. Rozhodl jsem se vytvořit testovací prostředí pomocí virtuálních počítačů. Tyto počítače představují jednotlivé subjekty, jež se vyskytují během skutečného DDoS útoku. Subjekty tvoří útočník, oběť útoku a zařízení se Suricatou, které se snaží ochránit síť oběti. Pro amplifikační útoky je potřeba vytvořit také server, který amplifikuje útok útočníka a směřuje ho na počítač oběti. Dále jsem vytvořil sadu nástrojů, jejichž důležitá funkce spočívá v generování zmíněných DDoS útoků a sbírání statistik. Dané útoky je možné spouštět s různými parametry. Nástroje umožňují mj. i konfigurovat jednotlivá zařízení. Sada nástrojů je navržena tak, aby ji bylo možné snadno a beze změn přenést do reálného fyzického testovacího prostředí, které může disponovat například větším výkonem nebo různou konfigurací strojů a sítě.

Poslední kapitola se zaměřuje na využití této sady nástrojů ve virtuálním prostředí pro odsimulování jednotlivých DDoS útoků a ověření správné funkčnosti navržených pravidel k potlačení DDoS útoků využitím IDS/IPS Suricata.

Kapitola 2

Útoky typu odepření služby

Tato kapitola se věnuje úvodu do problematiky síťových hrozeb, které se zaměřují na útoky tzv. odepření služeb, neboli DoS a DDoS. Zabývá se popisem mnoha známých typů útoků a jejich výskytem v současnosti. Vycházím zde z informací dostupných zde [5, 15].

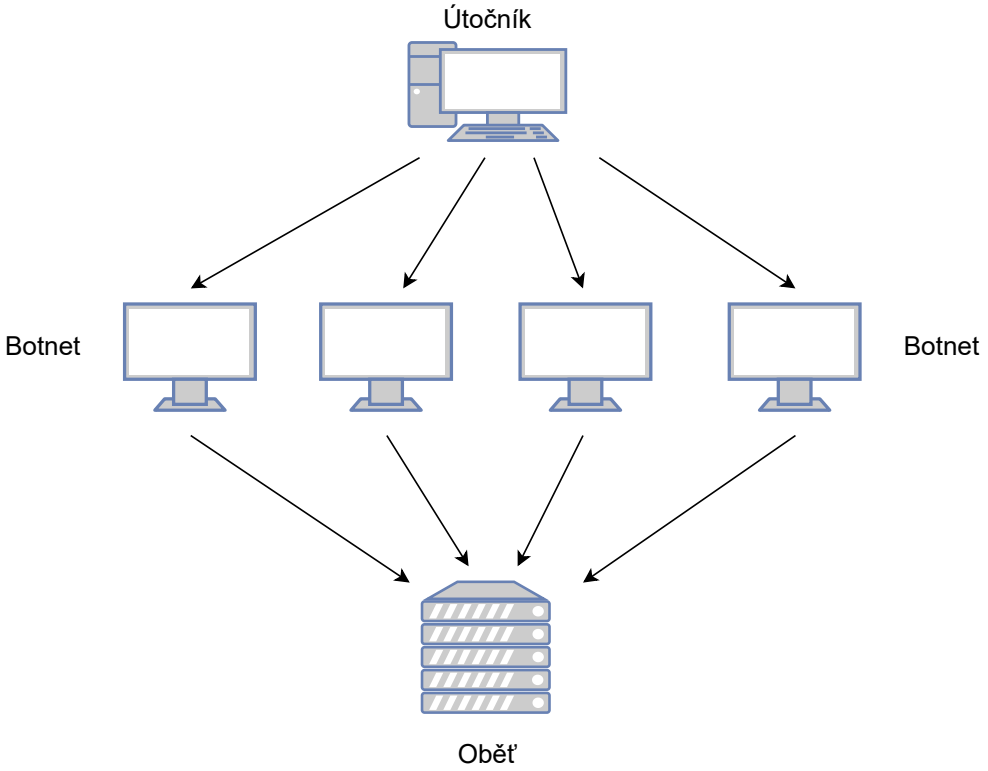
Útok odepření služby je takový typ útoku, který zahlcuje službu enormním množstvím požadavků v krátkém čase, jenž už daná služba není schopna zvládnout obsloužit, a dochází tak k jejímu kolapsu a nedostupnosti. To může být způsobeno přetížením linky, kdy šířka pásma nezvládá tak velké množství požadavků. Nebo může dojít k přetížení serveru, například vypotřebováním paměti, procesor nemusí stíhat obsloužit veškeré požadavky, může být zároveň v jeden moment připojeno vícero falešných uživatelů, kteří znemožní využít služby serveru pro legitimní uživatele.

Pokud se jedná o útok vedený pouze z jednoho zařízení, jedná se o DoS útok (Denial of Service). Variantou tohoto útoku je DDoS (Distributed Denial of Service). DDoS útok je znázorněn na obrázku 2.1, útok je veden z vícero zařízení na jeden konkretní cíl (na obrázku dole). Často jsou tyto stroje součástí tzv. botnetu (uprostřed), někdy přezdívané jako zombie. Botnet je kolekce strojů, jež byly napadeny škodlivým kódem – malwarem a jsou ovládány z jednoho centrálního počítače (nahoře). Cílová oběť je zahlcena mnoha nežádoucími pakety. Jeden z největších DDoS útoků využívající botnet, jaký se kdy povedlo uskutečnit, je Mirai Botnet [51] v roce 2016.

V současnosti je více rozšířený DDoS útok, jelikož dokáže vyvolat větší objem dat. Mimoto většina moderních nástrojů má již zabudovanou ochranu proti obyčejným DoS útokům. Mnoho útoků typu odepření služby má za cíl získat peníze od napadené společnosti za příslibu ukončení útoku. Objevují se však motivy i čistě z osobních důvodů, ať už třeba skupina hacktivistů, kteří kritizují současné dění (skupina Anonymous), nebo jen skupina znuděných teenagerů, kteří mají rádi adrenalin nebo si chtějí vybít zlost na nějaké instituci (například na škole) [15]. Taktéž se DDoS využívá jako prostředek konkurenčního boje mezi společnostmi. Firma, která se stane obětí útoku, může mít své služby nefunkční třeba i několik dní a tratit na tom velké množství peněz nebo přijít o zákazníky.

Důležitým pojmem, se kterým se můžeme setkat, je vektor útoku. Jedná se o pojmenování typu útoku, jak se projevuje, čeho zneužívá a jak ho specifikujeme.

Ať už jsou motivy jakékoliv, je velice důležité se umět těmto útokům bránit a v ideálním případě jim předcházet dříve, než vzniknou. V dnešní době je dokonce možné si objednat skupinu lidí, kteří mohou mimo jiné otestovat infrastrukturu a bezpečnost webových služeb vůči DDoS útokům. Jedná o tzv. penetrační testy [33].



Obrázek 2.1: Schéma DDoS útoku.

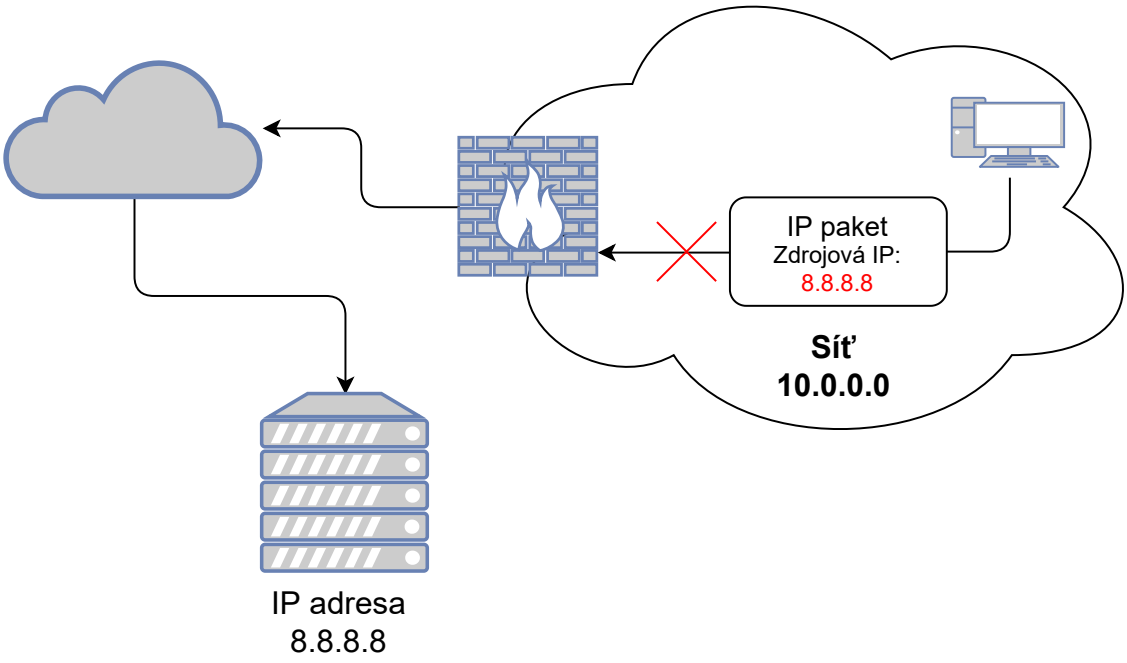
2.1 IP spoofing

Posílání IP paketů je primární způsob, jakým mezi sebou počítače, potažmo všechna chytrá zařízení, komunikují po síti. IP pakety obsahují hlavičku a tělo. Zatímco v těle zprávy je umístěn obsah zprávy, hlavička obsahuje různá metadata. Jsou zde také informace o příjemci a o odesílateli paketu. V normálním případě zdrojová IP adresa obsahuje skutečnou adresu odesílatele.

Útočníci používají řadu nástrojů k tomu, aby zdrojovou adresu změnili (podvrhli) za jinou. Příjemce si tak bude myslet, že se jedná o věrohodného odesílatele, a začne zprávu zpracovávat, odpověď pak dorazí oběti útoku. Útočník se tak může schovat za IP adresu někoho jiného. Dalším způsobem jak tohoto zneužít je, když útočník pošle velké množství požadavků různým příjemcům, kteří odpoví na jednu a tu samou podvrženou adresu a tím jsou schopni zahltit nic netušící oběť velkým množstvím dat.

Man-in-the-Middle je metoda [17], kdy útočník stojí mezi dvěma komunikujícími objekty a je schopen odposlouchávat jejich vzájemnou komunikaci a popřípadě podvrhovat falešné informace.

Bránit se IP spoofingu je poměrně náročné a z hlediska koncových uživatelů takřka nemožné. Častou a doporučovanou metodou je ingress filtering [7]. Jedná se o způsob ochrany, která je implementována na okrajích sítě. Všechny IP pakety jsou kontrolovány, zda se jejich zdrojové adresy opravdu shodují s tou z jaké sítě přicházejí. Tento způsob ochrany by měl provádět poskytovatel internetu, který je zodpovědný za příjem paketů z cizí sítě do té své. Doporučován je také egress filtering, zobrazen na obrázku 2.2 – zkou-



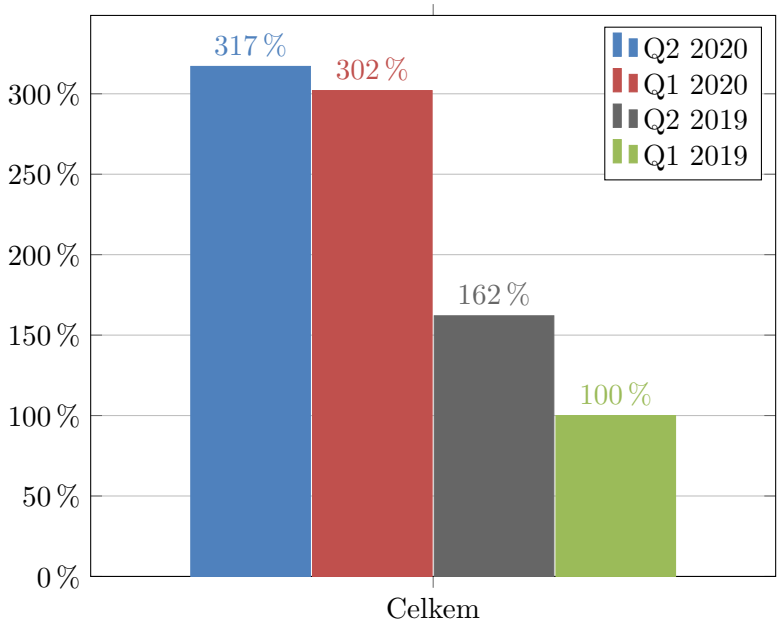
Obrázek 2.2: Příklad egress filtrování.

mání paketů ještě před opuštěním privátní sítě. Kontroluje se, zda jejich zdrojová adresa pochází opravdu z dané dané sítě.

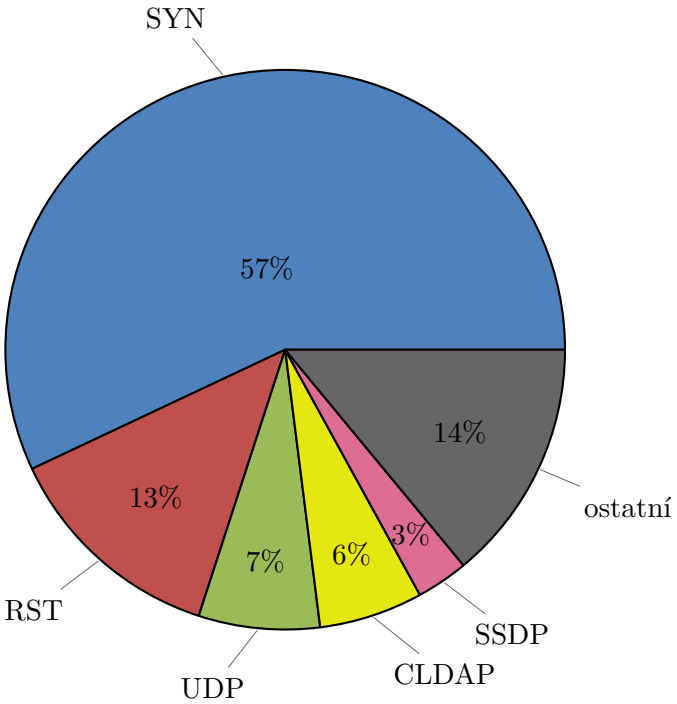
2.2 Četnost útoků

Zpravodajské weby, které informují o kyberútocích a hrozbách na internetu, vydávají pravidelné čtvrtletní zprávy, ve kterých analyzují, k jakým útokům docházelo, jaké jsou nové hrozby atd. Zpráva za druhé čtvrtletí [20] je zajímavá především z toho důvodu, že došlo až k trojnásobnému počtu DDoS útoků oproti minulým rokům, viz graf 2.3. Toto je způsobeno především pandemickou krizí, která přinutila vlády celého světa přijmout speciální opatření. Mnoho lidí bylo nuceno zůstat a pracovat z domova. Vše se začalo řešit přes internet a velká část zábavních internetových služeb (jmenovitě Netflix) zaznamenala rekordní množství současně připojených uživatelů. Někteří poskytovatelé těchto služeb museli dokonce přikoupit další servery nebo nějakým způsobem omezit své služby, aby nápor tolika lidí zvládli [24]. V očích kyberzločinců to upoutalo velkou pozornost.

Na obrázku 2.4 je znázorněn graf, kde je možné vidět zastoupení jednotlivých typů vektorů za druhé čtvrtletí 2020. Je zde jednoznačně vidět, že mezi nejpopulárnější útoky patří SYN flood, jež zneužívá tzv. three-way-handshake, blíže vysvětleno v části o SYN flood útocích 2.3.1.



Obrázek 2.3: Porovnání počtu útoků za druhý kvartál roku 2020 oproti roku 2019. Data vycházejí z údajů zde [20, 19].



Obrázek 2.4: Typy DDoS útoků za druhý kvartál roku 2020. Položka ostatní obsahuje mj. DNS, NTP, Mirai... Data pocházejí z údajů od společnosti Cloudflare [8].

Nejvíce útoků se podle zprávy [20] odehrálo v Číně (65,12 %), následované Spojenými státy (20,28 %) a Hongkongem (6,08 %). Většina útoků (85,97 %) netrvala déle než tři hodiny. To se nemusí zdát jako příliš vysoké číslo, je ale nutné si uvědomit, že následky

mohou trvat i několik dalších hodin, než se systém podaří vrátit do původního stavu. Během prvního čtvrtletí roku 2020 byl zaznamenán útok, který trval 21 dní. Podle informací společnosti Cloudflare [8] pokračoval trend v relativně malých útocích. Sílu pod 1 Gbps mělo 51,5 % všech DDoS útoků za druhé čtvrtletí. Útok v rozmezí 1 – 10 Gbps pak 38,3 %. Zajímavým faktem je, že v březnu 2020 se uskutečnilo 88 % všech útoků, které se detekovaly za první polovinu roku 2020 a které měly sílu více než 100 Gbps.

2.3 Typy útoků

Následující sekce se budou věnovat různým druhům DDoS útoků. Tyto útoky jsem si vybral na základě analýzy nejčastějších typů. Je zde popsán průběh útoku, jak se projevuje, čeho zneužívá a jaké jsou možnosti, aby k podobným útokům nedocházelo. Vycházím především z informací dostupných zde [5, 2, 38].

Útoky se dále mohou dělit podle toho, na jakou síťovou ISO/OSI vrstvu se zaměřují, často se jedná o pokusy zneužít třetí, čtvrtou a sedmou vrstvu. V reálném světě však útočníci často kombinují různé typy útoků dohromady, aby zvýšili jeho sílu.

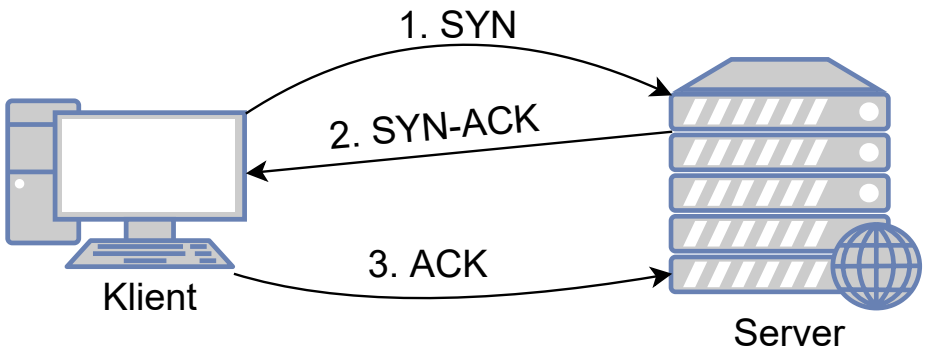
2.3.1 Volumetrické útoky

Volumetrický útok, někdy označován jako záplavový (angl. flood útok) je typ útoku, který lze popsat jako útok hrubou silou. Oběť je zahlcena tak velkým množstvím datového provozu, že již není schopna řádně pracovat. Jedná se o nejběžnější a nejznámější typ útoku, především proto, že není nikterak složité tento typ útoku provést.

SYN flood

SYN flood je takový typ DDoS útoku, který se snaží vypotřebovat všechny zdroje serveru tím, že posílá nesčetné množství SYN paketů, kterými se v TCP spojení zahajuje tzv. *three-way-handshake*. Princip [45] three-way-handshake je popsán na obrázku 2.5. Klient prvně zašle SYN paket, kterým naznačuje začátek spojení. Server odpoví SYN-ACK paketem, kterým souhlasí s navázáním spojení, načež klient pošle ACK paket potvrzující spojení se serverem. Teprve potom jsou vyměňována data mezi serverem a klientem.

Útočník, který chce zneužít tohoto principu, začne posílat velký objem SYN paketů na server oběti, často s podvrženou IP adresou (více o IP spoofingu v této kapitole 2.1).



Obrázek 2.5: Three-way-handshake je způsob, jakým se zahajuje TCP spojení.

Server poté na každý obdržený paket odpoví SYN-ACK zprávou a otevřený port nechá připraven na odpověď. Zatímco server čeká na odpověď, která však nepřijde, protože zařízení na podvržené IP adrese nemá zájem o navázání spojení, útočník nadále posílá další SYN pakety. Postupně tak dojde k vyčerpání paměti serveru (backlog), jež je určena pro uchování stavové informace pro ustanovení spojení, tzv. Transmision Control Block [30]. Tomuto se někdy říká *half-open* útok, jelikož na jedné straně spojení zůstává otevřené po určitou dobu, zatímco na druhé straně je spojení uzavřeno. K útoku je možné také použít *botnet*, tedy síť infikovaných počítačů, jež jsou ovládané útočníkem. V tomto případě nemusí ani útočník používat podvržené IP adresy, pouze mu stačí zahltit server oběti dostatečnou silou, aby zkolaboval. Vystopovat zpětně útočníka, který používá například Mirai botnet, je zpětně poměrně obtížné [51].

Jedním z možných řešení je zvýšit počet současně otevřených spojení [45]. Každý operační systém má povoleno mít pouze určité množství spojení, která mohou být *half-open*. Ke zvýšení tohoto počtu je potřeba více paměti. Dalším řešením je jednoduše přepisovat *half-open* informace pro ustanovení spojení v momentě, kdy je vyčerpána kapacita. Tady je však nutné, aby legitimní požadavky byly vyřízeny dříve, než dojde k zahlcení.

Lepší způsob jsou tzv. SYN cookies [14]. Jakmile server dostane SYN paket, tak vytvoří kryptografický hash z IP adresy, čísla portu a dalších unikátních identifikátorů klienta. Ten pak vloží do SYN-ACK paketu. Legitimní klient zpětně odešle ACK, který obsahuje daný hash, z něhož se poté vytvoří původní SYN paket a alokuje se paměť na daném portu. RST cookies je taktéž validní způsob potlačení tohoto typu DDoS útoku. Zařízení, starající se o bezpečnost serveru, záměrně odešle nevalidní SYN-ACK odpověď, na kterou legitimní klient odpoví RST paketem, ve kterém sděluje, že je něco v nepořádku. V tuto chvíli zařízení pozná, že danému klientovi může věřit a uskuteční s ním spojení. Více o RST cookies je možné se dočíst v této práci [9].

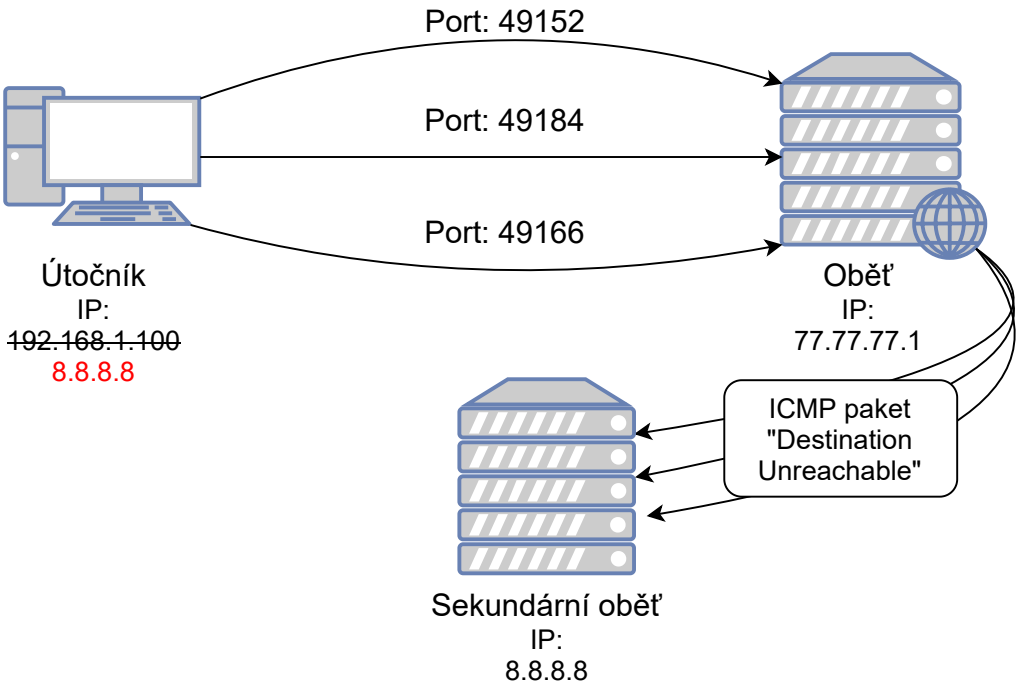
RST/FIN a ACK flood

Na konci každého TCP spojení, které je zahájeno *three-way-handshakem* (obrázek 2.5), dochází k výměně RST nebo FIN paketů, značící ukončení spojení. Při RST flood útoku je server zaplaven enormním počtem RST paketů sloužících k ukončení spojení, jež však neexistují. Server musí vyplýtvat velké množství výpočetního výkonu, aby prošel všemi těmito pakety a porovnal je s každým aktuálně navázaným spojením. Což může vést ke značnému zpomalení serveru. Stejně funguje i ACK flood, kdy je server zahlcen požadavky, které se používají k potvrzovaní navázaného spojení, spolehlivého doručení dat apod. Kromě serverů tyto útoky také negativně ovlivňují činnost firewallu, který musí všechny pakety kontrolovat.

Na podobném principu je vystaven útok „spoofed TCP reset“, taktéž známý jako „forged TCP reset“. Útočník odposlouchává komunikaci mezi dvěma subjekty a snaží se narušit jejich spojení a zabránit mu, aniž by o tom oba dva věděli. Tohoto typu útoku využívá Čínská vláda k cenzuře internetu, tzv. Great Firewall of China [10]. Metodou, jak se bránit proti RST flood útoku, je omezit počet aktuálně zpracovávaných RST paketů [34].

UDP flood

UDP flood je označován jako volumetrický útok [26]. UDP pakety nepoužívají žádný způsob počáteční legitimizace jako v případě TCP a jeho *three-way-handshake* (podrobněji vysvětleno v části o SYN flood 2.3.1). Obecně je známo, že UDP vyžaduje méně režie na fungování, čehož se využívá v případech, kdy je nutná rychlá odezva, např. VoIP, chatovací služby nebo



Obrázek 2.6: Příklad volumetrického UDP flood útoku.

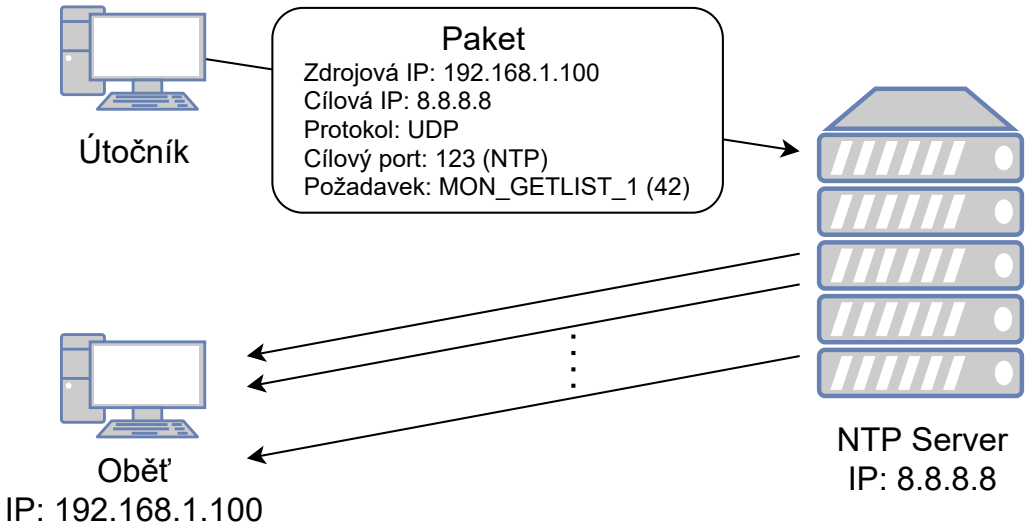
video streaming. Jednoduchost UDP spojení lze však také snadno zneužít. Pomocí UDP paketů je možné zasílat pakety ohromné velikosti nic netušící oběti.

Princip UDP flood útoku je znázorněn na obrázku 2.6. Útočník zahájí útok zasláním UDP paketu oběti s určitým číslem portu. Server přijme paket a podívá se, zda na daném portu naslouchá nějaká služba. V případě, že nenalezne takovou službu, systém vygeneruje ICMP zprávu o nedostupnosti služby zpět odesílateli. Zde však často útočníci používají podvržené IP adresy a zpráva tak dorazí zcela jinému příjemci. Mezitím co server zpracovával první UDP paket, obdržel od útočníka další desítky paketů stejného typu. Postupně tak může dojít k celkovému zahlcení serveru, který nemusí odpovídat na legitimní dotazy.

Většina dnešních operačních systémů se snaží omezovat počet generovaných ICMP zpráv v případě, že se jich najednou začne objevovat velké množství [16]. Pokud je útok dostatečně silný, nemusí stačit ani firewall systému. Útoku je možné předcházet tím, že server bude filtrovat všechny UDP pakety a povolí pouze DNS zprávy, jež používají UDP komunikaci. Poslední možností je filtrovat všechny UDP pakety nějakým IPS/IDS systémem, který bude zkoumat obsah a ty s podezřelým obsahem bude zahazovat.

2.3.2 Amplifikační útoky

Amplikfační útoky jsou jistou variací na volumetrické útoky. Podstata amplifikačních útoků spočívá ve zneužití určitých typů UDP protokolů, které byly špatně navrženy. Útočník zašle dotaz s podvrženou zdrojovou IP adresou oběti na server, na němž běží služba s tímto protokolem. Server na dotaz odpoví mnohonásobně větší zprávou než původní dotaz a tu směřuje na oběť nežádaného DDoS útoku. Dojde tedy k tzv. *reflexi* útoku. Potenciální efekt amplifikačního útoku se dá změřit jako poměr velikosti vygenerované odpovědi serveru ku velikosti UDP dotazu útočníka. Označujeme to zkratkou BAF (bandwidth amplification factor) [48]. Výhodou amplifikačních útoků je, že útočník sám nepotřebuje vyprodukovat



Obrázek 2.7: Amplifikační NTP útok.

příliš mnoho síťového provozu, stačí mu tedy menší šířka pásma, než jaká je na straně oběti. Tyto útoky se tedy spíše snaží zahltit linku vedoucí k serveru.

NTP amplifikační útok

NTP je jeden ze základních síťových protokolů. Slouží k synchronizaci času. Je používán mnoha servery, síťovými a koncovými zařízeními. Jeho nejnovější verze 4 je popsána v RFC 5905[1]. Starší NTP servery podporují příkaz `monlist` (`MON_GETLIST_1`). Tento dotaz slouží primárně k monitorování. Odpověď na něj je posledních 600 klientů, kteří se přihlásili k serveru. Tato zpráva je mnohonásobně větší než `monlist` dotaz. Podle dat společnosti Imperva [13] je BAF, tedy amplifikační faktor, v rozmezí kolem 20 – 200. NTP útok je nastíněn na na následujícím obrázku 2.7. Útočník zašle UDP pakety NTP serverům, které umožňují `monlist` příkaz s podvrženou IP adresou oběti. Jelikož se jedná o UDP komunikaci, není vyžadována žádná autentizace. Každý server na přijatý dotaz generuje mnohonásobně větší odpověď. Oběť je ve výsledku zahlcena velkým množstvím síťového provozu.

Na straně NTP serveru se dá bránit tím, že se zablokuje příkaz monlist. Toho lze mimo jiné dosáhnout aktualizováním na nejnovější NTP verze, kde je tahle zranitelnost opravena a příkaz je v základní konfiguraci zakázán, nebo nahrazen. Na straně klienta/oběti je dobré sledovat síťový provoz. Pokud by se najednou začalo objevovat velké množství NTP paketů, může to značit probíhající DDoS útok na infrastrukturu sítě. Taktéž je dobré se řídit zásadami bezpečnosti pro ingress filtering, aby se minimalizovalo podvrhování IP adres (více o tom v kapitole IP spoofing 2.1). Pokud zařízení nevyžaduje synchronizaci času, je dobré zavřít UDP port 123, na kterém funguje služba NTP.

DNS amplifikační útok

DNS (Domain Name System) je služba, jejíž hlavní cíl je mapování doménových jmen, např. www.example.com na IP adresu a naopak. DNS amplifikační útok je další velice známý vektor útoku používaný pro DDoS. Podobně jako v případě NTP amplifikačního útoku

[1] https://tools.ietf.org/html/rfc5905

i zde se útočníci snaží zneužít trhliny v DNS, aby z mála objemových dat vyprodukovali mnohonásobně větší datový tok směřující k oběti útoku. Útočník zašle dotaz na veřejně přístupný DNS server. Tento dotaz typu „ANY“ značí, že odpověď má kromě IP adresy obsahovat všechny záznamy, které o této doméně zná. Což mohou být informace o poddoménách, záložních serverech, serverech pro mailovou komunikaci, aliasy a další. Více o tomto dotazu je možné se dočíst v oficiální dokumentaci RFC 8482[2]. Najednou tak útočník vygeneruje odpověď mnohonásobně větší. BAF (amplifikační faktor) se u tohoto útoku pohybuje okolo 50 [50]. Aby útočník ještě navýšil velikost odpovědi, může použít protokolové rozšíření klasického DNS dotazu, tzv. `EDNS(0)` [4]. Paradoxně DNSSEC, jenž má chránit strukturu a obsah DNS odpovědi, může být také zneužit ke zvětšení odpovědi, jelikož bezpečnostní klíče a položky jsou objemově poměrně velké [31].

Ideální způsob, jak zabránit DNS amplifikaci, je, aby všechny volně dostupné DNS servery měly dostatečné zabezpečení. V rámci privátní sítě by měla všechna zařízení, od serverů až po IoT[3], zpracovávat pouze požadavky od lokálních DNS serverů. Z vnitřní sítě by neměl odcházet žádný DNS provoz. Taktéž všechen příchozí DNS tok by měl být zpracováván prvně DNS serverem určeným pro tuto síť. Aby nedošlo k přetížení tohoto serveru, je doporučeno využít funkce DNS Anycast, která rozdistribuuje všechen DNS provoz mezi dostupné DNS servery. Poskytovatelé internetu by měli aktivně předcházet IP spoofingu v jejich síti, viz kapitola 2.1.

CLDAP

CLDAP (Connection-less Lightweight Directory Access Protocol) je definován v RFC 3352[4]. Jedná se o alternativu k LDAP, což je protokol navržený firmou Microsoft. Slouží k připojení, vyhledávání a modifikování adresářů na síti. Oba protokoly fungují na portu 389, zatímco LDAP používá TCP, CLDAP byl navržen pro rychlejší komunikaci, a proto používá UDP. LDAP je jeden z nejpoužívanějších protokolů pro přístup k uživatelským jménům a heslům v *Active Directory*, jež jsou zakomponovány v mnoha serverech. Active Directory je služba od Microsoftu, která zajišťuje ukládání dat a přístup k nim po síti [3]. Špatně nakonfigurovaný CLDAP server může být snadno použit k amplifikačnímu DDoS útoku.

Tento typ útoku byl objeven v roce 2016. Podle zprávy společnosti Akamai [1] může amplifikační faktor BAF útoku dosáhnout až 70, průměrně se však pohybuje kolem 57. Největší zaznamenaný útok byl v roce 2017, jeho síla byla 24 Gbs. Útok probíhá podobně jako u většiny amplifikačních DDoS útoků. Paket s podvrženou IP adresou je zaslán CLDAP serveru, který místo útočníkovi odpoví oběti útoku, dojde tedy k reflexi útoku. Pro potřebnou amplifikaci je v požadavku dotaz na vrácení seznamu všech uživatelů zaregistrovaných v Active Directory.

Ochranou může být implementování ingress a engress filtrování paketů na hranicích sítě s jinou. Jedná se o jeden z nejefektivnějších způsobů, jak se bránit IP spoofingu. Nebezpečí spočívá v CLDAP serverech, které naslouchají a odpovídají na portu 389. V současnosti jich je podle serveru Shodan bezmála kolem 100 000 [37]. V dubnu 2017 jich však bylo přes 250 000, což svědčí o tom, že některé servery se podařilo zabezpečit. Předcházet tomuto útoku lze také použitím pravidel pro IPS/IDS zařízení.

[2] https://tools.ietf.org/html/rfc8482
[3] IoT – všechna fyzická zařízení, která jsou schopna se připojit a komunikovat po síti
[4] https://tools.ietf.org/html/rfc3352

Memcached DDoS

Memcached je distribuovaná síť serverů, které jsou využívány webovými stránkami ke cachování dat, tedy ukládání do vyrovnávací paměti. Slouží k rychlejšímu načítaní a zobrazování dat uživateli serveru. Tyto servery nemají žádný způsob autentizace. Je to způsobeno tím, že tyto Memcached servery neměly nikdy být přístupny z internetu pro nelegitimní uživatele. Ovšem jejich špatná konfigurace zapříčinila, že jsou volně dosažitelné z internetu. Podle dat serveru Shodan jich je momentálně kolem 56 tisíc a všechny mohou být zneužity k DDoS útoku [36]. Pokud jsou takto veřejně přístupné, je možné, aby na ně útočník nahrál vlastní data velkého objemu. DDoS je podobný ostatním amplifikačním útokům. Útočník pošle `HTTP GET` požadavek serveru na data, která si tam sám uložil. Protože útočník použil podvrženou IP adresu, data jsou zaslána oběti, u které může dojít k odepření služeb. BAF tohoto typu útoku je v rozmezí 10 000 až 52 000, což z něj dělá jednu z největších hrozeb internetu [21]. Jeden z největších útoků, jaký byl kdy podniknut, se odehrál v roce 2018. Útočníci zneužili Memcahed servery a zaútočili na vývojářskou platformu GitHub. Datový tok dosáhl síly přes 1,3 Tbs, více o tomto útoku zde [27]. O pár dní později se útočníci pokusili o útok znovu, tentokrát byl zaznamenán datový tok 1,7 Tbs.

Memcached je možné zneužít, protože velké množství IT administrátorů nebylo schopno tyto servery správně nakonfigurovat a zabezpečit. Pokud organizace nebo firma vlastní Memchached server, měla by zajistit, že není přístupný veřejně z internetu použitím firewallu k zablokování požadavků z vnější sítě na UDP port 11211. Pokud je třeba, aby byl přístupný i z jiné sítě, tak zablokovat komunikaci přes UDP. Pro Memcached existují také příkazy `shutdown\r\n` a `flush_all\r\n`, které může administrátor zaslat na server a tím ukončit probíhající DDoS útok na svoji síť z daného zneužitého serveru.

SSDP útok a WS-Discovery

SSDP (Simple Service Discovery Protocol) je typ reflexivního útoku, který zneužívá UPnP (Universal Plug and Play). Služba UPnP slouží k automatické konfiguraci a propojení zařízení na lokální síti. Útočník prvně skenuje seznam všech zařízení na síti, která poslouchají na portu 1900 pro UDP. Na tato zařízení pak pošle UDP pakety (použitím botnetu) s podvrženou IP adresou oběti. SSDP protkol trpí stejným nedostatkem jako Memcached servery. Protokol nebyl nikdy navržen pro to, aby se používal volně na internetu, ale jen v rámci lokální LAN sítě. Protokol pracuje pouze nad LAN multicast IP adresou 239.255.255.255. Pakety s touto IP adresou se nemají mimo LAN síť vůbec vyskytovat. SSDP však nikterak neřeší, zda pakety přicházejí ze stejné sítě, ve které se vyskytuje dané zařízení. Během amplifikačního útoku jsou zaslány dotazy s obsahem `ssdp:all`, nebo `ssdp:rootdevice`, které vrací největší odpověď. Oběť tak může dostat pakety, které jsou až 30krát větší než původní dotaz [41].

Na začátku roku 2019 se objevil podobný typ útoku. Zneužívá Web-Service-Dynamic-Discovery protokol, zkráceně WSD, který funguje velice obdobně a má podobné chyby jako SSDP. Na internetu je mnoho těchto IoT zařízení, ke kterým se dá normálně dostat, 87,7 % z nich tvoří bezpečnostní kamery [11]. Amplifikační faktor se v průměru pohybuje kolem 200 [32].

OpenVPN

Mezi další zneužívané služby patří OpenVPN. OpenVPN je software používaný k vytváření šifrovaných kanálů pro virtuální privátní sítě. Zde byla chyba na straně softwaru[5]. Uživatel poslal `P_CONTROL_HARD_RESET_CLIENT_V2` paket serveru, který odpověděl `P_CONTROL_HARD_RESET_SERVER_V2`. Pokud však uživatel neodpověděl `P_ACK_V1` paketem, server neustále posílal další `P_CONTROL_HARD_RESET_SERVER_V2` pakety. Dělo se tak do té doby, než došlo k timeoutu u socketu na straně serveru. Timeout byl ve výchozím nastavení 30 sekund. Pokud server měl nastavený exponenciální růst timeoutu, tak došlo k odeslání maximálně pouze 5 paketů ze strany serveru. Pokud byla však použita fixní hodnota, tak server posílal paket každých 0,5 sekundy, odeslal tak 60 paketů během 30 sekund na jeden `P_CONTROL_HARD_RESET_CLIENT_V2` paket. Dokud tato chyba nebyla opravena, bylo k dispozici přes 700 000 serverů, jež mohly být použity k tomuto útoku [28].

2.3.3 Pomalé útoky

Pomalé útoky neboli slow útoky, jsou poměrně náročné na detekování, protože nefungují na principu použití hrubé síly [38]. Snaží se cílit na vícevláknové servery, kdy každé vlákno je zaneprázdněno požadavkem, který záměrně trvá dlouho dokončit. Útočník posílá data po síti velmi pomalu, ale tak akorát, aby mu nevypršel čas spojení. Jelikož se tyto útoky nevyznačují velkým provozem, může stačit útočníkovi i jeden stroj k tomu, aby vypotřeboval všechny zdroje na serveru. Pomalé útoky si lze představit tak, že útočník naváže spojení se serverem a poté mu sdělí, že má pomalé spojení a posílá data velice pomalu, ale dostatečně rychle, aby server nezavřel spojení.

Slow útokům se dá bránit poměrně těžko. Jednou z možností je, že navýšíme kapacity serveru, aby zvládal vícero spojení. To však nezabrání útočníkovi zesílit svůj útok. Daleko účinnější je použití reverse proxy serverů, které zajišťují, že žádný klient nebude nikdy přímo komunikovat se serverem. Taktéž je dobré na straně serveru nastavit nějaký absolutní timeout na požadavky, který by uzavřel spojení v případě, kdy komunikace probíhá příliš dlouho. Doporučené je též neustále monitorovat kapacitu a zatíženost serveru, např. vytížení procesoru, paměť, sledovat tabulku připojení, počet vláken.

Slowloris

Slowloris naváže spojení se serverem a poté posílá velice pomalu hlavičky HTTP paketu. Snaží se tak udržet server ve stavu, kdy čeká na klienta, až zašle všechna potřebná data. Útočník se takto snaží zaneprázdnit všechna vlákna serveru, aby nebyl schopen odpovídat legitimním uživatelům.

R.U.D.Y.

R.U.D.Y. (R-U-DEAD-YET?) pracuje na podobném principu. Útok generuje `HTTP POST` požadavky, ve kterých sdělí serveru, kolik dat má očekávat. Poté však posílá tato data velice pomalu. Server udržuje spojení otevřené, protože očekává další data.

Slowread

Slowread nebo Sockstress, jak je tento útok někdy označován, se vyznačuje tím, že po navázání spojení klienta se serverem a legitimním požadavku, začne klient číst odpověď

[5] https://github.com/SoftEtherVPN/SoftEtherVPN/pull/1014

velice pomalu. Někdy klidně i jeden bajt za sekundu. Klient posílá serveru TCP paket `Zero window`, čímž sděluje, že nestíhá číst, a vyžaduje po serveru, aby na chvíli přestal posílat data, ale neuzavíral spojení. Čímž může dojít k postupnému vypotřebování všech zdrojů serveru [25].

Kapitola 3

Systémy IDS a IPS

Tato část se zaměřuje na to, jakými způsoby jsme schopni pomocí různých IDS/IPS zařízení detekovat a následně eliminovat DDoS útoky. Odhalit amplifikační útok není příliš náročné, jelikož produkuje velký objem dat. Mitigace tohoto typu DDoS není zdaleka tak snadná, jelikož pakety přichází často od legitimních zdrojů a pochází od služeb, jež jsou často používané (například DNS, NTP). Často tak musíme chirurgicky rozebrat každý paket a ten, který se zdá být škodlivý, zahodit.

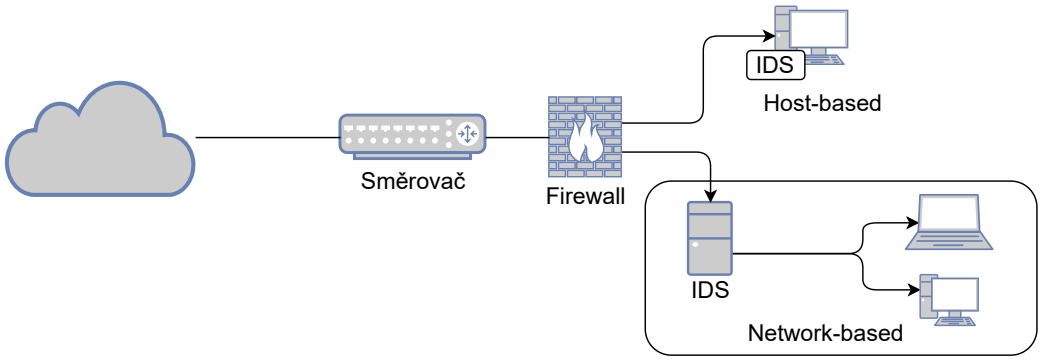
IDS/IPS je obecně systém pro detekci hrozeb, detekci nežádoucího provozu, monitorování síťového provozu a detekci různých typů útoku, mezi které patří i DDoS útoky. Cílem této práce je ověřit použitelnost IDS/IPS pro detekci a mitigaci DDoS útoků. Proto se u těchto systémů zaměřím na to, jak je toho možné docílit.

IDS (angl. Intrusion Detection System) pro potlačovaní útoků používá seznam pravidel, na jejichž základě je systém schopen odhalit potenciální DDoS útok. Nedokáže mu však samostatně zabránit. V případě podezření útoku systém vygeneruje zprávu (může se jednat o e-mail, SMS, zápis do logovacího souboru) a zašle ji správci sítě. Na něm je, aby útoku zabránil [47].

IPS (angl. Intrusion Prevention System) je často považován za vylepšení/nadstavbu IDS systému. IPS provádí podobnou sadu úkonů jako IDS, jeho hlavním úkolem je odhalit škodlivé pakety a aktivně zablokovat DDoS útok. IPS systém však může chybně vyhodnotit legitimní pakety jako potenciální hrozbu, a zamezit tak fungování služby, aniž bychom chtěli. Je proto nutné tento systém správně nakonfigurovat, aby nepáchal více škody než užitku. Bezpečnostní experti se snaží využít oba systémy dohromady, zkráceně IDPS (Intrusion Detection and Prevention System) [47].

Detekce signatur je jeden z nejběžnějších způsobů, jakým se dá odhalit škodlivý software. Nemusí se přitom jednat konkrétně o DDoS útok, většina antivirů je postavena na tomto principu. Je tak možné odhalit viry typu trojský kůň, malware, worms, atd [40]. V případě IDS/IPS systémů má aplikace v sobě předem nakonfigurované vzory. Vzor může být sekvence bajtů. Ty jsou pak porovnávány s datovým tokem, pokud se nalezne shoda, systém vygeneruje zprávu. Tento přístup vyžaduje pravidelně aktualizované databáze vzorů. Slabinou tohoto stylu je, že funguje spíše reaktivně a pro každý nový typ útoku je nutné vytvořit odpovídající vzor.

Detekce anomálií je druhý způsob detekce. Je navržen tak, aby byl schopen odhalit i nové typy útoků. Zde se užívá schopnosti machine-learningu [47], kdy se stroj postupně učí chování jednotlivých útoků a snaží se předcházet novým. Snaží se odhalit jakoukoliv nesrovnalost, nenormálnost na síti a reagovat na ni. Zde je však opět obrovský prostor pro plané poplachy, a tak s sebou tento systém přináší jisté riziko.



Obrázek 3.1: Schéma zapojení IDS/IPS systému. Host-based, kdy software běží samostatně na každém stroji, nebo Network-based, kdy je IDS/IPS systém nainstalován na jednom stroji, který zkoumá veškerou příchozí komunikaci.

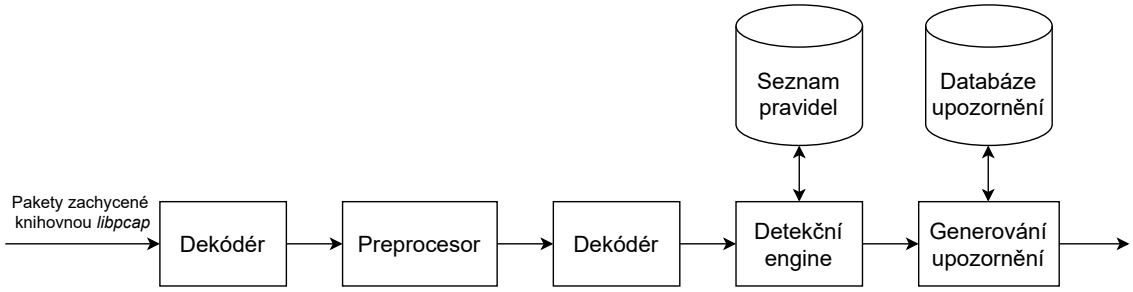
Poslední možností je tzv. stavová analýza protokolu [29], což je způsob, kdy se udržují informace o spojení dvou hostů a ty se porovnávají se stavovou tabulkou. V tabulce se udržují informace o cílových a zdrojových IP adresách, portech a protokolech. Hledají se jakékoliv nesrovnalosti, výchylky od normálního chování na síti. Tento způsob vyžaduje vysoké nároky na výpočetní výkon a není schopen odhalit útoky, které svým chováním odpovídají legitimním charakteristikám daného protokolu.

IDS/IPS systém můžeme zapojit dvojím způsobem, jak je možné vidět na obrázku 3.1. Host-based detection system je způsob, kdy aplikace běží přímo na zařízení, které má ochraňovat. Tato možnost je logisticky náročná na údržbu a aktualizaci softwaru a všech potřebných komponent v případě, kdy spravujeme vícero zařízení. Výhodou je, že nezpomalujeme tok dat pro zbytek sítě analýzou všech paketů. Druhá možnost je Network-based detection system. Zde pracuje aplikace pro monitorování a případně prevenci proti útoku na jednom stroji. Je nutné zapojit stroj s aplikací tak, aby byl zachycen a zpracováván veškerý datový tok, který prochází lokální sítí. Ideálně hned za směrovač v síti. Takto se docílí toho, že případný útok je mitigován hned na začátku vstupu do sítě a nezasáhne zbytek infrastruktury. Ovšem musíme počítat s tím, že při vetším objemu dat může dojít ke zpoždění na základě zpracovaní všech paketů vstupujících do sítě. Zařízení, na kterém IDS/IPS běží, musí proto být dostatečně výkonné.

V následující části se zaměřím na konkrétní programy, které se používají pro detekci hrozeb, nežádoucího provozu, monitorování síťového provozu a detekci různých typů útoku, mezi které patří i DDoS útoky.

3.1 Snort

Snort je jeden z nejznámějších a nejpoužívanějších IDS/IPS systémů. Stejně jako Suricata používá seznam pravidel pro odhalení nebezpečného internetového provozu. Původně Snort vytvořil Martin Roesch v roce 1998, ale v současnosti je vyvíjen především firmou Cisco [35]. Snort nemá žádné vlastní grafické prostředí (GUI), k dispozici je ale velké množství open-source nástrojů pro webový front-end pro analýzu dat vyprodukovaných Snortem (např. BASE, Sguil). Snort je velice populární, především díky pravidelným aktualizacím databázových pravidel a aktivní komunitě. Dlouhou dobu ovšem Snort neumožňoval vícevláknové zpracování, což dělalo ze Suricaty výkonnější nástroj. Od verze 3.0 již Snort



Obrázek 3.2: Architektura Snortu. Aby paket prošel do zbytku sítě, musí projít jednotlivými bloky, kde je zkontrolován.

také podporuje vícevláknové zpracování paketů [39]. Architektura Snortu je zobrazena na obrázku 3.2.

Dekodér slouží k uchovávání paketů, které zachytí pcap rozhraní. Pakety se následně dekódují a získají se z nich užitečné informace, jako například čísla portů, cílová IP adresa, protokol. Pokud Snort narazí na neznámou hlavičku paketu, vygeneruje chybovou hlášku.

Preprocesor je první filtr, kterým paket projde. Zde se mohou filtrovat potenciálně nebezpečná TCP, UDP spojení.

Detekční engine používá seznam pravidel k tomu, aby zkontroloval daný paket a rozhodl, zda je nebezpečný, či nikoliv. *Seznam pravidel* je soubor, ve kterém jsou popsána pravidla, podle kterých se hledají podobné signatury v paketech.

Generátor upozornění slouží pro tvorbu a logování záznamů, které jsou pak zobrazeny správci sítě. Teprve poté je paket případně puštěn dále do zbytku sítě.

Snort může běžet ve třech hlavních režimech:

- **Sniffer Mode** – program pouze čte pakety, které mu přijdou na vstup, a zobrazuje je do terminálu konzole.
- **Packet Logger Mode** – program loguje záznamy o přijatých paketech na disk.
- **Network Intrusion Detection System Mode** – program monitoruje a analyzuje síťový provoz. Porovnává pakety se sadou předem nadefinovaných pravidel a chová se podle příkazů v těchto pravidlech.

Příklad Snort pravidla, kterým se generuje upozornění, kdykoliv se zaznamená ICMP Echo request (ping) nebo Echo reply.

```
alert icmp any any -> $HOME_NET any (msg:"ICMP test";
    sid:1000001; rev:1; classtype:icmp-event;)
```

alert – typ akce, kterou má Snort provést. V tomto případě pouze vygeneruje upozornění.

icmp – specifikace protokolu, pro který má být pravidlo určeno.

any any – zdrojová IP adresa a zdrojový port. Poté následuje šipka (->) značící směr ze zdrojové adresy do cílové. Za šipkou se nachází cílová adresa (zde je použita proměnná `$HOME_NET`) a číslo portu.

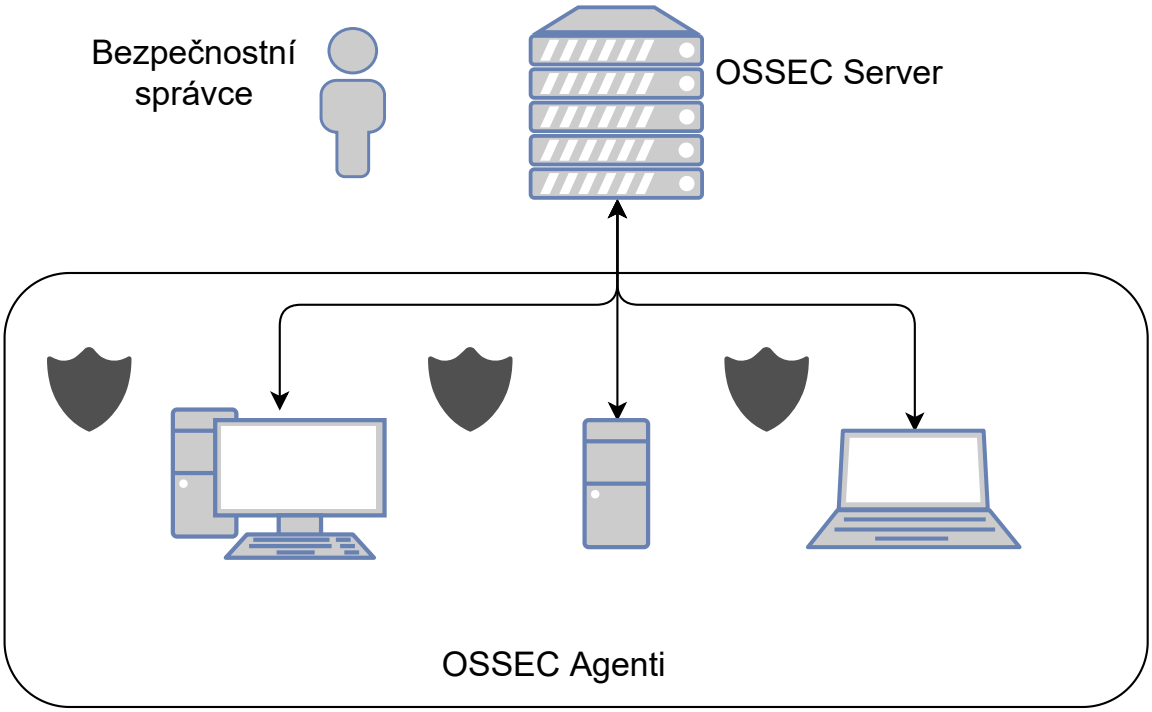
Poslední část jsou vlastnosti pravidla, jaké vzory, příznaky a obsah se hledá v paketu.

3.2 Suricata

Suricata je vyvíjená neziskovou organizací OISF. Nástroj Suricata může fungovat v obou režimech, tedy IDS a IPS. Její knihovny a engine jsou použity pod GPLv2[1] licencí. Používá seznam pravidel, která se používají pro detekci nebezpečné aktivity na síti. Tyto signatury (pravidla) jsou kompatibilní s pravidly u Snortu, Suircata ovšem umožňuje větší konfigurační možnosti. Suricata byla postavena jako konkurence Snortu, má však lehce odlišnou architekturu a podporuje vícevláknové zpracování. Taktéž Suricata umožňuje oproti Snortu použít skriptovací jazyk Lua. Suricata je nástroj, který budu používat v této práci pro odhalení a mitigaci DDoS útoků, je proto více rozebrána v samostatné následující kapitole 4.

3.3 OSSEC

OSSEC je open-source program typu host-based (běží tedy pouze na stroji, více na obrázku 3.1) [35]. Slouží k analýze logů, ke kontrole integrity a registrů. Má schopnost aktivně reagovat na nebezpečné hrozby v reálném čase. Skládá se ze dvou hlavních částí, jak je možné vidět na obrázku 3.3.



Obrázek 3.3: Schéma OSSEC se skládá z centrálního serveru a z agentů, která posílají data na server, kde dochází k jejich vyhodnocení.

Server (nahoře vpravo na obrázku) je hlavní výpočetní jednotka OSSEC systému. Jsou na něm uloženy veškeré soubory, logy a záznamy událostí. Také jsou zde soubory s pravidly a je zde uložena dekódovací jednotka. Agent (dole) je pak každý stroj v síti, na kterém je nainstalován OSSEC v režimu agent. Ten přeposílá veškerá data na centrální server,

[1] GPLv2 je licence pro svobodný software. Vyžaduje, aby odvozená díla byla dostupná pod stejnou licencí.

kde dochází k jejich podrobnější analýze. Správce sítě může kontrolovat jednotlivé stroje z jednoho zařízení a zároveň má stále informace ze stroje, který mohl být napaden. OSSEC je podporován na všech Unixových systémech a Windows. Je možné také nastavit režim, kdy monitoruje server, na kterém je nainstalován.

3.4 Zeek

Zeek (původně pojmenovaný Bro) funguje trochu odlišně než Suricata či Snort. Určitým způsobem detekuje hrozby, jak pomocí signatur, tak pomocí anomálií [35]. Analyzátor Zeeku transformuje přijaté pakety na sérii událostí. Událost může představovat přihlášení uživatele na FTP server, zobrazení stránky v prohlížeči apod. Síla Zeeku spočívá v jeho vlastním tzv. Policy Script interpretu. Tento interpret používá vlastní skriptovací jazyk, který má daleko obsáhlejší možnosti, než co představují pravidla u Suricaty nebo Snortu. Zeek také umožňuje velkou dávku automatizace, což usnadňuje práci bezpečnostním technikům. Je však časově náročné ho zprovoznit a vyžaduje kvalifikovanou obsluhu. Neobsahuje žádné vestavěné GUI.

3.5 DDoS Protector

DDoS Protector je kombinací hardwarové FPGA akcelerační karty a softwarového řízení, kdy část paketů je zpracovávaná přímo na kartě a část pomocí SW. Softwarová část je řešena pomocí Suricaty. Zařízení je možné zapojit ve stylu network-based IDS/IPS (obrázek 3.1, kde je jeho potenciál největší. Dokáže filtrovat pakety při šířce pásma až 100 Gb/s. Je možné ho tedy připojit na páteřní body sítě, nemusí tak filtrovat pouze menší sítě, ale klidně i síť celé organizace nebo síť ISP. Je vyvíjen firmou CESNET. Více o tomto zařízení a jeho propojení se Suricatou je možné se dočíst v bakalářské práci [22].

Tohle jsou jedny z nejznámějších a nejpoužívanějších systémů detekce a prevence proti DDoS útokům. Podrobnější porovnaní výkonu jednotlivých aplikací je možné nalézt v této diplomové práci [23]. V následující části je popsáno, jak funguje Suricata. Odzkoušení funkčnosti Suricaty a jejích pravidel je jedním z cílů této práce. Pravidla použitá pro Suricatu lze použít i pro DDoS Protector, jelikož Suricata je integrována právě v DDoS Protectoru.

Kapitola 4

Suricata

Tato kapitola se věnuje výhradně softwarovému nástroji Suricata. Úvod do těchto IDS/IPS systémů je nastíněn v předchozí kapitole 3, kde jsou více rozebrány odlišné aplikace. Suricata je jeden z nejvíce používaných nástrojů pro potlačení hrozeb, útoků a monitorování síťového provozu. Jedná se o open-source nástroj, který vyvíjí nezisková organizace Open Information Security Foundation (OISF). Suricata podporuje multivláknové zpracovaní, což jí dává ohromnou výhodu. Je napsána v jazyce C a v Rustu a běží na mnoha operačních systémech, včetně Windows, Linux, FreeBSD a macOS [44].

Suricata používá pravidla pro odhalení nebezpečného toku na síti. Je proto důležité, aby byla tato databáze pravidelně aktualizovaná a udržovaná. Jedna z nejznámějších databází pravidel nese název *Emergency Threats* [18]. Jedná se o projekt, kde mnoho bezpečnostních expertů provádí výzkum a snaží se vytvářet signatury pro mitigaci nebezpečných pokusů o narušení sítě. Pokud Suricata odhalí nějaký pokus o útok, vygeneruje hlášení do svého logovacího souboru. Je také možné si nechat vygenerovat hlášení ve formátu JSON, pro lehčí další zpracování a automatizaci. Mimo jiné Suricata podporuje skriptovací jazyk Lua, který lze použít buď v pravidlech pro detekci signatur, nebo pro úpravu výstupních souborů. Suricata též umožňuje extraktovat soubory a ukládat je na disk. Taktéž podporuje software třetích stran, například pro uživatelské prostředí (GUI). Suricata podporuje Unixový socket. Díky tomu je do určité míry možné provádět změny konfigurace nebo monitorovat běh Suricaty, aniž by bylo třeba ji jakkoliv restartovat.

4.1 Pravidla

Formát pravidel (signatur) je popsán níže na příkladu pravidla. Toto konkrétní pravidlo je převzato přímo z databáze *Emergency Threats* [6]:

```
drop tcp $HOME_NET any -> $EXTERNAL_NET any (
msg: "ET TROJAN Likely Bot Nick in IRC (USA +..)";
flow:established,to_server; flowbits:isset, is_proto_irc;
content:"NICK "; pcre:"/NICK .*USA.*[0-9]3,/i";
reference:url,doc.emergingthreats.net/2008124;
classtype:trojan-activity; sid:2008124; rev:2;)
```

Akce
: je činnost, která se má provést, pokud se najde shoda s paketem. Akce může být:

- *alert* – vygeneruje se pouze hlášení do záznamového souboru.

- *pass* – paket se dále prozkoumávat nebude a pošle se dál.
- *drop* – zahodí paket a vygeneruje zprávu do logu.
- *reject* – zašle `RST/ICMP` zprávu zpět na zdrojovou adresu o nedosažitelnosti cílové adresy.
- *rejectdst* – zašle `RST/ICMP` chybovou zprávu na cílovou IP adresu.
- *rejectboth* – zašle `RST/ICMP` chybovou zprávu pro příjemce i odesílatele.

V IPS (Intrusion Prevention System) režimu každá z *reject* akcí současně zahazuje přijatý paket. Priorita pravidel je následující: *pass, drop, reject, alert*, toto pořadí však lze modifikovat.

Hlavička

se skládá z několika částí:

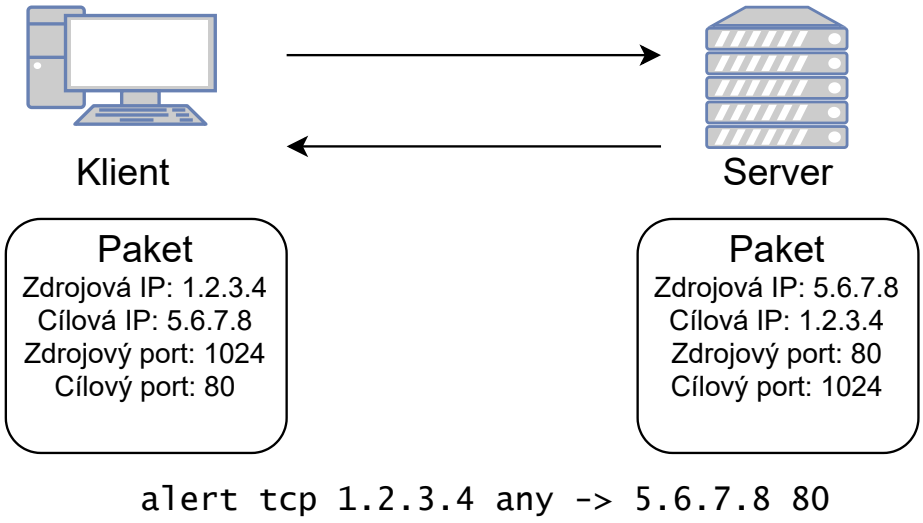
- `tcp` – jedná se o protokol transportní vrstvy. Suricata krom základních TCP, UDP, ICMP a IP (IP zde znamená „any“, tedy jakýkoliv transportní protokol) podporuje také protokoly na sedmé (aplikační) vrstvě, např. HTTP, FTP, NTP, DNS...
- `$HOME_NET` – proměnná, která v sobě zahrnuje seznam zdrojových IP adres. Proměnnou lze změnit v konfiguračním souboru. Je také možné použít přímo konkrétní číslo (podporován je IPv4 i IPv6) nebo rozsah IP adres.
- `$EXTERNAL_NET` – stejně jako `$HOME_NET`, jen pro cílové IP adresy. Povšimnout si lze šipky, která naznačuje směr.
- `any` – slouží pro specifikaci zdrojových a cílových čísel portů. Je nutné si však uvědomit, že číslo portů neznamená protokol, ale spíše aplikaci, která přijímá daný paket. Pro HTTP totiž zdrojový port je náhodné číslo, ale HTTP server většinou běží na portu 80 (`any -> 80`).
- směr se označuje pomocí šipky `->`. Znamená to, že pouze pakety v tomto směru jsou kontrolovány. Lze použít i `<>`, což určuje oba směry. Pravidlo pro směr `<-` nelze udělat. Názorně je tohle zobrazeno na obrázku 4.1.

Suricata pracuje s globálními proměnnými. Může to být již zmíněný `$HOME_NET`, `$EXTERNAL_NET`, ale i třeba `$HTTP_PORTS` a další. Tyto proměnné slouží k lepší specifikaci pravidel a k informování Suricaty o prostředí, ve kterém se pracuje. Je možné je nalézt v konfiguračním souboru `suricata.yaml`.

Vlastnosti pravidla

jsou uzavřeny v závorkách a odděleny středníkem. Jednotlivé vlastnosti mají své pořadí a změna jejich pořadí by znamenala změnu jejich významu. Jde zároveň o nejsložitější část, proto se často používají nebo modifikují pravidla, která již byla vytvořena. Jednotlivé vlastnosti se označují klíčovými slovy. Klíčová slova se mohou lišit i od použitého transportního protokolu. Příklad některých klíčových slov:

- `msg` – vlastnost, která zde slouží pro informaci o signatuře a pro zprávu, která se vygeneruje v logovacím souboru při shodě pravidla.
- `flow:established` – znamená, že pouze pakety, které jsou již součástí ustanoveného spojení, budou kontrolované (popřípadě naopak). Záleží i na použitém



Obrázek 4.1: Obrázek zobrazuje příklad HTTP komunikace mezi klientem (IP 1.2.3.4) se serverem (IP 5.6.7.8). Signatura pro Suricatu: `alert tcp 1.2.3.4 any -> 5.6.7.8 80` by znamenala, že pouze pakety klienta směrem k serveru budou zkoumány.

transportním protokolu, u TCP ustanovené spojení znamená po navázaní *three-way-handshake*, u UDP to představuje moment, kdy jsou zaznamenány pakety z obou stran komunikace. Lze tak zabránit tomu, že bude narušena legitimní komunikace.

- `content:"NICK"` – v paketu se musí objevit tento obsah. Lze specifikovat i například přímo hexadecimální obsah (`content:"|61 61|"` znamená "aa"). Tato část signatury poskytuje největší možnosti, jak blíže specifikovat nebezpečný provoz.
- `pcre:"/NICK.*USA.*[0-9]3,/i"` – Suricata umožňuje prohledávat pakety i pomocí regulárních výrazů. Jedná se ovšem o výkonnostně náročnou operaci, proto se ještě používá v kombinaci s jiným klíčovým slovem. Například `content`, pro který bude Suricata hledat shodu dříve, než začne kontrolovat shodu s regulárním výrazem.
- `flowbits: isset, is_proto_irc` – kontroluje, zda je nastaven stav `is_proto_irc`. Tento stav je možné nastavit jiným pravidlem při shodě. Můžeme tak specifikovat, že se má kontrolovat pravidlo až poté, co byla shoda s jiným pravidlem.

Vlastnosti lze různorodě specifikovat až do nejmenších detailů, aby byly filtrovány pouze pakety, jež mohou být nebezpečné, a legitimní provoz nebyl narušen. Bližší informace o těchto pravidlech a všech klíčových slovech je možné nalézt v oficiálních dokumentacích [42, 43].

Zajímavou možností je nastavit práh (angl. threshold) pro dané pravidlo. Znamená to, že se dané pravidlo začne aplikovat pouze, když dosáhne určitého počtu shod za daný časový interval. V Suricatě se tohle nastavuje klíčovým slovem `threshold`. Můžeme použít také možnost `limit`, který nám zajišťuje, aby administrátor, jenž obdržuje upozornění, nebyl zahlcen velkým množstvím zpráv. Například, když chceme vygenerovat upozornění pouze jednou za 3 minuty. Je možné použít i kombinaci `threshold` a `limit`. V souboru

threshold.conf je možné upravit tuto vlastnost globálně pro dané pravidlo. Je zde taky možné definovat nový typ akce pro pravidlo, když dosáhne určitého počtu shod za čas a jak dlouho má tato nová změna platit.

```
alert tcp any any -> $MY_SSH_SERVER 22 (msg:"Connection to SSH server";
                flow:to_server; flags:S,12; sid:888;)
```

Je pravidlo generující upozornění, když se někdo snaží připojit pomocí SSH na server. Příklad omezení na pouze 10 připojení za 60 sekund je znázorněno níže:

```
rate_filter gen_id 1, sig_id 888, track by_src, count 10,
        seconds 60, new_action drop, timeout 300
```

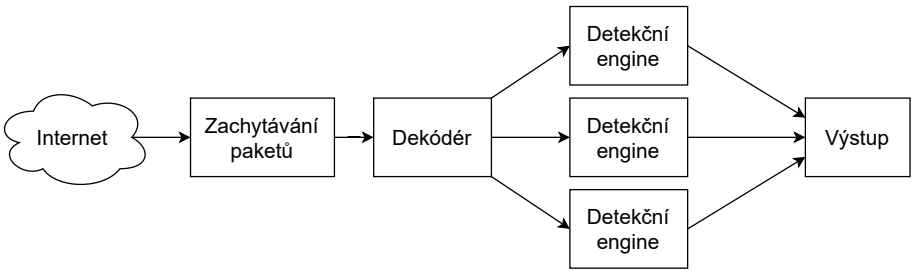
Pokud by bylo dosaženo tohoto prahu, tak všechny následující pokusy o připojení jsou zahozeny. Omezení platí pouze po dobu 300 sekund. Můžeme taky potlačit daná pravidla, například pokud chceme výjimku pro jednu konkrétní IP adresu, můžeme to specifikovat klíčovým slovem `suppress`. Pravidla, která jsou zaznamenaná v souboru *threshold.conf* přepisují ta, jež jsou nastavena v souboru s pravidly.

4.2 Architektura

V následujících podkapitolách vycházím především z oficiální dokumentace Suricaty [42, 43]. Suricata se skládá z několika bloků, které jsou navzájem propojeny a zajišťují její funkčnost. Suricata je schopna využívat vícero jader procesoru současně, v čemž spočívá její největší výhoda oproti konkurenci. Pakety mohou být zpracovány různými vlákny, rozdělení paketů k jednotlivým vláknům zajišťují fronty. Každé vlákno se skládá z modulů, znázorněných na obrázku 4.2. Modul *zachytávání paketů* (angl. *Packet acquisition*) se stará o příjem paketů, *dekodér* dekóduje paket a zkoumá ho z pohledu jednotlivých OSI vrstev. Paket poté postupuje do *detekčního enginu*, který je výpočetně nejnáročnější. V tomto modulu se musí porovnat paket se všemi pravidly, která jsou uložena v databázi, což může představovat i přes tisíc pravidel. *Výstupní* modul se stará o generování zpráv o paketech. Pokud je Suricata v režimu IPS, stará se tento modul také o zahození škodlivých paketů.

Suricata nabízí tři *Runmódy*:

- *workers* – režim, který je většinou nejlepší volbou. Uvnitř každého vlákna je paket zpracován všemi moduly. Vlákno se tedy stará o všechno, od příjmu paketů z fronty, přes jeho dekódovaní a detekci, až k výstupu. Distribuce paketů do front je starostí síťové karty, ta je také zodpovědná za správné vytížení jednotlivých vláken. Je nutné



Obrázek 4.2: Znázornění modulů uvnitř Suricaty.

aby pakety patřící k jednomu toku byly zasílány do stejného vlákna. Pokud by tomu tak nebylo, bylo by nutné, aby vlákna mezi sebou komunikovala, což je drahá operace.

- *autofp* – tento režim se používá při zpracování PCAP souborů nebo při použití NFQ pro IPS režim. Je zde přítomno jedno nebo více speciálních vláken pro zachytávání a dekódování paketů. Ta se starají o správnou paralelní distribuci, pokud síťová karta tohoto není schopna docílit. Pakety jsou poté posílány do druhé skupiny vláken, které se starají o detekci pravidel. Jelikož je mezi těmito dvěma skupinami vláken nutná komunikace, nejedná se o nejlepší řešení z hlediska výkonu.
- *single* – všechny pakety jsou zpracovávané pouze jedním vláknem, nedochází tak k paralelnímu zpracovávání. Tento režim je určen především pro vývojové účely.

4.3 Konfigurace

Na Linuxových systémech je konfigurační soubor uložen na výchozí lokaci: */etc/suricata/-suricata.yaml* ve formátu YAML. YAML (YAML Ain't Markup Language) je formát pro serializaci dat, je dobře čitelný i člověkem, ne jenom strojem, a dobře přenosný. Suricata umožňuje upravit řadu vlastností, mj. runmód, kdo je oprávněn spouštět Suricatu, prioritu pravidel, lokace výstupních souborů atd.

Jedna z důležitých vlastností je možnost upravit počet zároveň zpracovávaných paketů. Jedná se o kompromis mezi výkonem a náročností na paměť (RAM). Velký počet současně zpracovávaných paketů znamená větší výkon, ale i větší nároky na paměť. Na druhou stranu, pokud nemáme k dispozici dostatek paměťového prostoru, Suricata nebude schopna tak výkonně zpracovávat velké množství paketů a může dojít k latenci při větší síťové zátěži. V konfiguračním souboru je toto možné ovlivnit položkou:

```
max-pending-packets: 1024
```

Další věc, která lze upravit, je chování detekčního modulu. Detekční modul porovnává pakety s pravidly. Za účelem optimalizace a zvýšení rychlosti zpracování, detekční modul rozděluje pravidla do skupin. Pakety jsou poté zkoumány pouze vůči určité skupině pravidel, které potenciálně odpovídají shodě s paketem. Například nemá smysl, aby paket, který používá UDP transportní protokol, byl zkoumán vůči pravidlům pro TCP protokoly. Nicméně, při velkém počtu pravidel se může stát, že si detekční modul vytvoří velké množství skupin, což povede k větším nárokům na paměť. V konfiguračním souboru je možné upravit maximální počet takto vytvořených skupin.

Suricata umožňuje nastavit, jakým způsobem probíhá kontrola pravidel. Při inspekci pravidel se nejprve vezme jeden vzor z pravidla. Ten je určen např. použitím klíčového slova `fast_pattern`. Pokud není toto klíčové slovo použito, Suricata vybere nejsilnější vzor (zpravidla ten, jenž je nejdelší a nejvíce variabilní). Tyto vzory se poté porovnávají s obsahem paketu. Pouze ta pravidla, jejichž vzory se shodovaly s pravidlem, jsou dále porovnávána. Tento systém se nazývá v angličtině *Multi-Patern-Matcher*. V konfiguraci Suricaty je možné změnit algoritmus pro tento systém. Výchozí algoritmus je `Aho-Corasick`, jedná se o druh slovníkového vyhledávacího algoritmu, který ve vstupním textu hledá prvky konečné množiny řetězců. Pro výkonnější stroje je doporučeno použít `Hyperscan` algoritmus, který využívá hybridních automatů pro současné porovnání velkého počtu regulárních výrazů (až desítek tisíc) napříč daty [12].

Další vlastnost, kterou lze změnit, se vztahuje k defragmentovaným paketům. Pakety, které dorazí ve fragmentované podobě, musí být před další inspekcí sestrojeny dohromady

do kompletní podoby. Fragmentované pakety jsou drženy v paměti do té doby, než dorazí zbytek, který k němu patří. Aby se zabránilo tomu, že jsou tyhle pakety drženy v paměti donekonečna v případě, že zbytek fragmentovaného paketu nedorazí, je možné nastavit několik zásad. Například nastavení časovače, jak dlouho budou fragmenty uloženy v paměti, nastavení maximálního počtu fragmentů, zda se má před alokovat nějaká paměť atd.

Je několik možností, jak přeposílat pakety do Suricaty. Například AF_PACKET, AF_RING nebo pomocí NFQ. NFQ je rozhraní uvnitř Linuxového kernelu, sloužící k filtrování paketů, překladu síťových adres, realizaci firewallu apod. Umožňuje odklonit část síťového provozu do různých aplikací, jako je například Suricata. Používá se proto pro IDS režim. Ve výchozím nastavení nejsou pakety, které byly zaslány do Suricaty pomocí IP tables s využitím NFQ, kontrolovány zbytkem pravidel v IP tables. Je ovšem možné tohle změnit, aby poté, co jsou pakety kontrolovány Suricatou, byly zkontrolovány pomocí IP tables nebo pomocí další jiné aplikace. Místo `iptables` je možné použít například `nftables`, který je přímočařejší. Principem je vytvoření řetězce paketů vyhrazeného pro IPS (Suricatu), který prošel pravidly firewallu. Obě tyto varianty pracují nad třetí (síťovou) OSI vrstvou.

Můžeme však pakety směrovat do Suricaty i na druhé (linkové) vrstvě. K tomu nám slouží AF_PACKET režim. V tomto režimu nám stačí pouze, aby fungovala síťová rozhraní. Suricata se postará o předávání paketů z jednoho rozhraní na druhé. Nepotřebujeme tedy konfigurovat žádný `nftables` nebo `iptables`. Zátěž mezi jednotlivá vlákna pak Suricata v tomto režimu provádí na základě toku, kde jsou pakety v rámci stejného toku předávány stejným vláknům. Stejně funguje i AF_RING, který umožňuje rozdělovat pakety i pomocí tzv. Round Robin mechanismu. Tedy, že pakety se rozdělují vláknům podle toho, jak přicházejí (podobně jako při rozdávání karet hráčům).

Suricata též umožňuje tzv. *Multi Tenancy*. Jedná se o možnost, kdy se používají na síti různé VLANy. Pro každou VLAN je možné nastavit samostatný seznam pravidel a proměnných.

Pro šifrovanou komunikaci neexistuje způsob dešifrování paketů. Zkoumání paketů oproti vzorům nemá v případě šifrované komunikace příliš smysl. Po *three-way-handshake* je inspekce pravidel limitována ve výchozím nastavení, Suricata si například bude udržovat informace o probíhajících spojeních SSL/TLS nebo bude kontrolovat jakékoliv zvláštnosti komunikačního protokolu. Je možné nastavit, aby inspekce veškerých šifrovaných paketů neprobíhala vůbec, nebo naopak aby byl obsah paketů kontrolován i přesto, že je zašifrován.

Kapitola 5

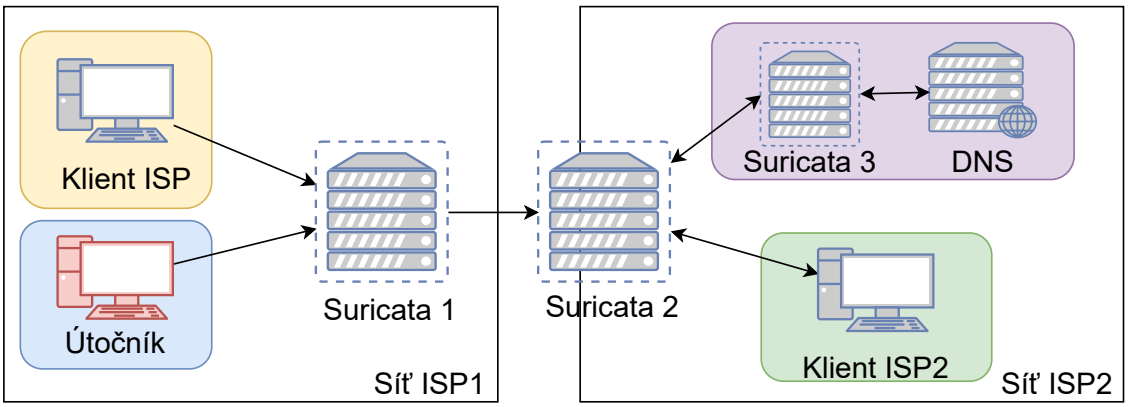
Návrh potlačení DDoS útoků

Tato kapitola se zaměřuje na návrh, jak by se daly jednotlivé útoky mitigovat. Jaké jsou možnosti v rámci Suricaty pro konkrétní DDoS útoky. U daných útoků je zmíněný návrh, jak by mělo pravidlo, které by se mělo snažit ho eliminovat, vypadat a jaké jsou jeho výhody popřípadě nevýhody.

5.1 Umístění Suricaty

Hned ze začátku je důležité si ujasnit, na jakém místě v infrastruktuře sítě bude IDS/IPS (Suricata v tomto případě) zařízení umístěno. Od tohoto se dále určuje, jakým způsobem budou jednotlivá pravidla konstruována a jaké náležitosti je potřeba splnit. Je nutné vždy zohlednit vlastnosti a chování sítě. V lokální síti, ve které je připojeno pár zařízení rychlostí 1 Gb/s, si lze dovolit zkoumat pakety více do hloubky, více detailně. Na velkých sítích poskytovatele internetu a páteřních sítích s kapacitou a rychlostí přes 100 Gb/s je nutné zohlednit latenci. Vždy je třeba určit kompromis mezi kvalitou detekce/filtrace a rychlostí zpracování. Já při návrhu pravidel nezohledňuji náročnost jednotlivých pravidel na výpočetní prostředky. Mým cílem při návrhu pravidel je co nejvíc zablokovat nebezpečný provoz a povolit pouze legitimní datový tok.

Různá umístění Suricaty je možné vidět na následujícím obrázku 5.1:



Obrázek 5.1: Příklad možných umístění Suricaty

Suricata 1 – v ideálním případě by Suricata běžela hned za útočníkem, tedy v místě, kde nebezpečné pakety opouštějí lokální síť útočníka. To by vyžadovalo, aby každý ISP (poskytovatel internetu) filtroval pakety od svých klientů ve svém přístupovém bodě. Poskytovatele internetu by mělo zajímat, zda se některý z jeho klientů vědomě či nikoliv podílí na nějakém globálním masivním DDoS útoku. Poskytovatel internetu však musí mít vyhrazené speciální zdroje na udržování aktuálnosti pravidel a musí mít jednotlivá pravidla dobře navržena, aby nezahazovala a nezpomalovala legitimní provoz, jelikož se může skrz něj směrovat prakticky jakýkoliv provoz. U menších ISP z ekonomických důvodů nemusí být příliš výhodné, aby provozovali nějaký systém pro detekci nebezpečného provozu, u větších by to nicméně bylo na místě.

Suricata 2 – v místě, kde jeden ISP dostane pakety od druhého poskytovatele, by se mohlo provádět například ingress/egress filtering jako základní obrana proti IP spoofingu (více v kapitole o podvržených IP adresách 2.1).

Suricata 3 – umístění IDS/IPS zařízení hned před nějakou službu, která by potenciálně mohla být zneužita k DDoS útokům. Jedná se především o amplifikační/reflexivní útoky. Pokud provozuji nějakou takovou službu (např. NTP, DNS, Memcached, ...) a zároveň je můj server dostupný volně na internetu, tedy má přiřazenou veřejnou IP adresu a je se na něj možné dotázat, tak bych ho měl mít řádně zabezpečen. Pokud nevyžaduje server žádnou autentizaci, tak by měl být veškerý příchozí provoz kontrolován, aby nebyla služba zneužita. Popřípadě na nebezpečné dotazy nereagovat, tedy například negenerovat žádnou odpověď.

Filtrováním paketů ještě před odrazem od zranitelného serveru dosáhneme lepšího výkonu pro Suricatu, ve smyslu, že dotazy nejsou tak velké jako odpovědi a Suricata tedy teoreticky zvládne vyfiltrovat větší objem paketů za jednotku času. Suricatu je také možné umístit přímo za server-odražeč, nevidím v tom však tolik smysl, jako umístit IDS/IPS před potenciálně zneužitý server. Většinou nechceme, aby se služba zbytečně zaobírala dotazy, na které bude odpověď později stejně zahozena. Jediný užitek vidím v tom, když využiji oba přístupy dohromady (tedy před a za server-odražeč), kdy chci mít jistotu, že nedojde k vygenerování nebezpečné odpovědi.

Poslední volbou je umístění IDS/IPS zařízení přímo do své lokální sítě. Tato možnost už je lehce nastíněna v kapitole 3, obrázek 3.1. Ve zkratce, může Suricata běžet buď přímo na stroji, který má chránit, nebo v lepším případě ji můžeme umístit na hraniční přístupové body do své lokální sítě, kde očekáváme příchozí provoz.

5.2 Návrhy pravidel

U jednotlivých útoků se snažím navrhnout pravidla, která by sloužila k eliminování daného typu útoku. Pro amplifikační útoky se snažím zahrnout dvě možnosti. První, kdy je Suricata umístěna přímo před serverem, který by mohl být zneužit k amplifikaci, a filtrovat tak přímo dotaz od útočníka. Druhá možnost je, kdy se pravidlo soustřeďuje na vygenerovanou odpověď. Při návrhu jednotlivých pravidel je důležité si ujasnit několik věcí. Kolik očekáváme příchozího a odchozího provozu, jak moc komplexní máme síť, zda bude stačit jedno IDS/IPS zařízení v síti, nebo jich bude třeba víc. Jaký typ provozu je pro nás důležitý a očekávaný a jaký by se naopak neměl vůbec objevit. Toto nám pomůže určit, jak výkonný hardware budeme potřebovat, a také nám to umožní nakonfigurovat jednotlivá pravidla, aby

nenarušovala legitimní provoz. V síti se mohou objevit dva druhy paketů, buď jsou pakety součásti útoku, nebo se jedná o legitimní pakety. Mohou nastat tyto situace:

- true positive – daný paket je součástí útoku a Suricata jej zahodí.
- true negative – daný paket je legitimní a Suricata jej nezahodí.
- false positive – daný paket je legitimní a Suricata jej zahodí.
- false negative – daný paket je součástí útoku, ale Suricata jej nezahodí.

Chceme docílit toho, aby míra false positive a false negative byla co nejmenší. Z hlediska klasifikace paketů se u nich typicky hledá nějaký kompromis. Fasle positive je varianta, které se chceme co nejvíce vyhnout, ideálně aby byla nulová. Jinak hrozí, že následky útoku mohou být ještě horší, než kdyby vůbec žádná filtrace neprobíhala. Pro false negative jsou následky při nejhorším takové, jako by provoz nebyl nikterak filtrován.

Níže následují návrhy pravidel pro jednotlivé typy útoků. Pokud se nejedná o převzaté pravidlo, tak pro jednoduchost u pravidel neuvádím `sid` a `rev`. Stejně tak opomíjím použití `classtype:attempted-dos`, které slouží pro klasifikaci pravidel. Každá klasifikace má určitou prioritu, která se projeví při generování upozornění do logovacího souboru. V reálném použití pravidel jsou však tato klíčová slova nutná pro správné fungování Suricaty.

5.3 Volumetrické útoky

Záplavové útoky není příliš snadné filtrovat, jelikož je lze snadno zaměnit legitimním provozem. Lze tak spíše jistým způsobem pravidla použít pro limitování provozu. Toho se dá docílit použitím klíčového slova `threshold`:

```
threshold: type both, track by_dst, count <numb>, seconds <numb>;
```

Specifikací `count` a `seconds` pravidlo sleduje počet přijatých paketů, které odpovídají danému pravidlu, za počet sekund. Pokud se překročí daná hranice, dané pravidlo se aplikuje. Pokud chceme pouze detekovat příchozí spojení, pak lze použít sledování podle cílové IP adresy (`track by_dst`). Z hlediska filtrace je toto ovšem velice nebezpečné, neboť bychom takto zablokovali i legitimní provoz, prakticky tak dojde přesně k tomu, co chce útočník. Na druhou stranu zabráníme tomu, že by daný server byl zahlcen a nebyl by tak schopen reagovat na jakékoliv dotazy. Při použití `track by_src` zase hrozí, že útočník může snadno podvrhnout velké množství IP adres, a nemusí tedy k žádné filtraci vůbec dojít. Je proto nutné zvolit variantu, která nejvíc odpovídá požadavkům a specifikaci sítě.

TCP flood

Návrh pravidla pro jednotlivé záplavové útoky používající TCP transportní protokol.

```
drop tcp $EXTERNAL_NET any -> $HOME_NET 0:1023 (
      msg:"Possible TCP Flood incomming";
flow:not_established, to_server; flags:<letter>;)
```

- `0:1023` – útočníci se většinou zaměřují na porty běžných služeb, nicméně je možné použít hodnotu `any`.
- `flow:not_established,to_server;` – pakety nejsou součástí žádného navázaného spojení a jsou součásti toku od klienta k serveru.
- flags:<letter> – pravidlo pro SYN flood útok by zde mělo nastavenou hodnotu `S`, pro ACK flood hodnotu `A` a pro RST flood hodnotu `R`.

Pomocí pravidla je dost obtížné mitigovat záplavové útoky používající SYN a obecně se pro tento typ záplavového útoku nehodí. Jak jsem uvedl na začátku, pravidlo pro SYN lze použít buď pouze k detekci, nebo k limitování provozu na určitý počet paketů za jednotku času.

Pro RST/ACK flood útoky však již pravidlo má smysl. Použitím klíčového slova `flow:not_established` totiž jasně říkáme, že Suricata může filtrovat tyto pakety, pokud nebylo prvně pro daný datový tok ustanoveno spojení. RST a ACK pakety by se totiž neměly objevit bez předchozí zahájené komunikace.

UDP flood

Návrh pravidla pro UDP flood.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 0:1023 (
       msg:"Possible UDP flood incomming";
         flow:not_established, to_server;
threshold: type both, track by_dst, seconds <numb>, count <numb>;)
```

Zde nám velice pomáhá použití klíčového slova flow:not_established. Pokud druhá strana (server) neodpovídá, tak se lze na tyto UDP pakety od útočníka zaměřit a filtrovat je. Nastává zde ovšem stejný problém s IP adresami, které může útočník snadno podvrhnout, proto použití `track by_dst` nemusí být dostatečné. Při sledování adres podle cílové adresy serveru můžeme dosáhnout určité míry limitace datového toku, díky čemuž by nedošlo k úplnému zahlcení serveru.

Nabízí se i možnost například zahodit vždy první paket od klienta, a pokud klient zašle UDP dotaz znovu (klient předpokládá, že byl paket ztracen po cestě), můžeme ho zařadit na seznam povolených spojení. V dnešní době jsou však už i útočící boti dostatečně chytří na to, aby se chovali obdobně jako legitimní klient. Pravidlo je tedy vhodné spíše pro monitorování síťového provozu.

5.4 Amplifikační útoky

Pro amplifikační útoky vždy uvádím dvě varianty pravidel: pravidla pro dotazy útočníka směrem k zranitelnému serveru a pravidla pro odpovědi serveru. Pro amplifikační útoky je Suricata uzpůsobena daleko více než v případě záplavových útoků. Lze tady totiž využít hledání vzorů uvnitř paketů. Ty totiž často obsahují na aplikační vrstvě detekovatelné příkazy daného protokolu, které se snaží útočník zneužít k amplifikaci.

Lze proto také snadno určit vlastnost `threshold` u pravidel:

```
threshold: type both, track by_src, count <numb>, seconds <numb>;
```

Díky tomuto pravidlu lze specifikovat, že při zvoleném počtu paketů (`count`) za počet sekund (`seconds`) se aplikuje daná akce. Dojde také k vygenerování pouze jednoho upozornění za tuto specifikovanou dobu. Pro většinu amplifikačních útoků lze tuto hodnotu nastavit nejvýše jeden paket a tím zcela eliminovat nebezpečné dotazy. Pokud bychom i přesto chtěli danou komunikaci povolit, je možné vytvořit speciální pravidlo `pass` pro danou IP adresu serveru, kterému důvěřujeme. Pro jednoduchost tuto vlastnost pro amplifikační útoky neuvádím, pokud se nejedná o nějakou výjimku, kde by dané pravidlo mohlo ohrozit i legitimní provoz.

NTP

Velká řada dnešních NTP serverů používá aktualizované verze, ve kterých už `monlist` příkaz není podporován. I přesto je možné nalézt na internetu servery, kterých se už delší dobu nikdo nedotknul. Popřípadě samotný útočník si může vytvořit vlastní NTP server, který danou chybu obsahuje.

V databázi *Emerging Threats* [6] je možné nalézt pravidla pro mitigaci NTP amplifikace, například pravidlo s označením SID: 2017918. U tohoto pravidla je možné vidět, že existují dvě různá čísla implementace (`02` a `03`). Je proto nutné zahrnout obě verze pravidla. Pro jednoduchost zde dále uvádím pouze hodnotu `03`, jelikož zbytek pravidla je identický. Dále jsou pro tyto pravidla v databázi uvedeny i testy na nastavení příznaků (flags) u paketů. Jedná se o testování, zda je testovaný paket odpovědí, nebo nikoliv, a zda jsou správně nastavené rezervované bity. Pokud chceme více specifikovat pravidlo a předejít *false positive*, měli bychom tuto kontrolu zahrnout také.

Pravidlo pro dotaz

Dotazy směřující na server obsahují požadavek `MON_GETLIST_1`, pravidlo jej tedy musí obsahovat.

```
drop udp $EXTERNAL_NET any -> $HOME_NET 123 (
msg:"Possible NTP DDoS Multiple MON_LIST Request (IMPL 0x03)";
content:"|03 2a|"; offset:2; depth:2;
byte_test:1,=,0x07,0,bitmask 0x87;)
```

- `123` – NTP služba má přiřazenou hodnotu čísla portu 123.
- `03 2a` – hodnota `2a` v šestnáctkové soustavě odpovídá číslu 42, což je kód pro `MON_GETLIST_1` příkaz. Hodnota `03` označuje číslo implementace (nejedná se o verzi NTP). V případě NTP amplifikace se může ještě objevit číslo `02` (viz výše).
- `offset:2; depth:2;` – udává, kde se má `content` vyskytovat. Suricata bude hledat od druhého bajtu a pouze do hloubky dvou bajtů.
- `byte_test:1,=,0x07,0,bitmask 0x87;` – testuje se, zda se jedná o odpověď nebo dotaz serveru. Suricata vezme první bajt od začátku UDP payloadu, provede bitový AND s touto hodnotou oproti nastavené masce (`0x87`). Výsledek poté porovná s hodnotou `0x07`. Dotaz má nastaven MSB bit příznaku na hodnotu 0. Zároveň se testují tři poslední (LSB) bity zda jsou nastaveny na hodnotu 1 (jedná se o rezervované bity). Během `monlist` dotazu jsou tyto bity nastaveny.

Dotazy typu `monlist` by se dle mého názoru neměly šířit volně po internetu. A pokud je to nutné, tak pouze v lokální síti.

Pravidlo pro odpověď

Odpovědi NTP serveru obsahují informace o typu původního dotazu, proto se na to lze snadno zaměřit při návrhu pravidla.

```
drop udp $EXTERNAL_NET 123 -> $HOME_NET any (
msg:"Possible NTP DDoS Multiple MON_LIST Response (IMPL 0x03)";
content:"|03 2a|"; offset:2; depth:2;
byte_test:1,=,0x87,0,bitmask 0x87;)
```

Pravidlo je velice podobné jako u pravidla pro dotaz, liší se v umístění čísla portu, které je ve směru paketu od serveru. Další odlišnost je v části, kde se testují bitové příznaky. Oproti dotazu se nyní testuje, zda je příznak odpovědi nastaven na hodnotu 1. V databázi *Emerging Threats* [6] je u pravidel také uvedeno testovaní na nastavený bit značící `more bits`. Na jeden dotaz server dokáže vygenerovat až 100 paketů a všechny kromě posledního mají tento příznak nastaven. Z mého úhlu pohledu je to zbytečné, jelikož to neeliminuje poslední paket. Možné je pravidlo specifikovat přidáním `content:"|00 48|"; offset:6; depth:2;`. Tímto se dotazujeme na část paketu, kde je uvedena velikost jednoho záznamu v odpovědi na dotaz. Jeden tento záznam má pokaždé velikost 72 bitů (jeden paket může obsahovat až šest těchto záznamů).

DNS

Jak je nastíněno v dřívější kapitole 2.3.2, DNS útok spočívá v zaslání požadavku `ANY` na DNS servery. Při tomto typu dotazu lze taktéž specifikovat informace o konkrétní doméně, tato možnost ovšem zvyšuje velikost samotného dotazu na DNS server a následná odpověď není tak velká, jako když se nechá tato část prázdná, což útočník většinou nechce. Když se nespecifikuje doména, automaticky se předpokládá dotaz `<root>`. Na tento typ dotazu jsem se zaměřil při návrhu pravidel.

Pravidlo pro dotaz

Při návrhu pravidla pro dotaz je důležité nalézt typ požadavku `ANY`.

```
drop udp $EXTERNAL_NET any -> $HOME_NET 53 (
msg:"DNS DDoS Response (QTYPE=<ROOT> ANY)";
content:"|00 00 ff 00 01|"; offset:12; depth:5;
byte_test:1,&,128,2;)
```

- `53` – očekáváme odpověď DNS služby, která má přiřazené číslo portu 53.
- `00 00 ff` – první dvojice nul značí `<root>`. Dotaz `ANY` má přiřazenou hodnotu 255, v šestnáctkové soustavě to odpovídá hodnotě `00 ff`.
- `00 01` – udává, že se jedná o třídu IN (Internet).
- `offset:12; depth:5;` – sděluje Suricatě, že má začít hledat až od dvanáctého bajtu UDP payloadu a má zkontrolovat pouze prvních 5 bajtů. Tímto se určuje přesná poloha, kde se typ `ANY` má vyskytovat. U DNS paketu se však tato část nachází v části, která může mít variabilní velikost, proto určení přesné polohy funguje pouze za předpokladu, kdy při dotazu není specifikovaná doména.

- `byte_test:1,!&,128,2;` – testuje, zda se jedná o DNS odpověď nebo DNS dotaz. Suricata má provést bitový AND nad jedním bajtem, offset je nastaven na 2 bajty od začátku UDP payloadu. Pokud je první bit nastaven na hodnotu 1, jedná se o odpověď, v opačném případě se jedná o dotaz. Testuje se tedy celá hodnota bajtu vůči číslu 128 (v desítkové soustavě).

DNS dotazy typu `ANY` nejsou příliš běžné a obecně platí zásada, aby se nepoužívaly. Objevit se mohou v ojedinělých případech například když by si chtěl někdo obstarat všechny (MS, A, AAAA, ...), které má server k dispozici, a zároveň chtěl minimalizovat počet dotazů.

Pravidlo pro odpověď

Odpovědi generované DNS serverem na dotaz typu `ANY` mají v těle uloženy také původní dotaz (angl. v části questions).

```
drop udp $EXTERNAL_NET 53 -> $HOME_NET any (
msg:"Possible DDoS DNS Response (QTYPE=<ROOT> ANY)";
content:"|00 00 ff 00 01|"; offset:12; depth:5;
byte_test:1,&,128,2;)
```

Pravidlo je proto stejné jako v případě dotazu. Rozdíl je v `byte_test:1,&,128,2;`, které testuje, zda jsou příznaky DNS paketu nastaveny pro odpověď. Stejně jako v případě dotazu i zde považuji tento provoz za nežádoucí, v případě nutnosti ho můžeme povolit pouze z důvěryhodného a zabezpečeného serveru.

CLDAP

CLDAP útok se soustředí na požadavek `objectclass=0` (někdy uváděno `objectclass=*`). Jedná se o filtr, který vrátí veškeré záznamy dané struktury, jež má server k dispozici.

Pravidlo pro dotaz

Pravidlo pro dotaz je možné nalézt v databázi *Emerging Threats* [6]. Identifikátor pravidla je 2024585. Nachází se v souboru *emerging-dos.rules*. Zde uvádím pouze zkrácenou verzi bez referencí a metadat.

```
drop udp $EXTERNAL_NET any -> $HOME_NET 389 (
msg:"ET DOS Potential CLDAP DOS Querry";
content:"objectclass0"; fast_pattern;
```

- `389` – CLDAP/LDAP má přiřazené číslo portu 389.
- `objectclass0` – typ žádosti, která generuje amplifikační útok.
- `fast_pattern;` – specifikuje, že předešlý `content` má být použit jako první pro *multi-pattern-matcher* při hledání shody s pravidlem.

Dle mého názoru je dotaz, který obsahuje tzv. *wildcard* pro filtr (zástupný znak hvězdička), velice nebezpečný a neměl by se v kombinaci s CLDAP, které neposkytuje žádnou autentizaci, používat, nebo jen v omezené míře na lokální síti.

Pravidlo pro odpověď

Podle zprávy zveřejněné společností Akamai [1] je pro každou odpověď společný znak `searchResEntry(1) "ROOT"`. Při tvorbě pravidla je tedy možné se na toto zaměřit.

```
drop udp $EXTERNAL_NET 389 -> $HOME_NET any (
    msg:"Possible CLDAP DOS Response";
content:"|30 84 00 00|"; offset:0; depth:4;)
```

- `389` – číslo portu pro CLDAP/LDAP službu je 389.
- `30 84 00 00` – při bližším zkoumání se zmíněný příznak `searchResEntry(1)-"ROOT"` opakuje v paketu pod zmíněnou hexadecimální hodnotou.
- `offset:0; depth:4;` – content je hned na začátku CLDAP payloadu.

Memcached

Memcached útok se většinou skládá z vícero částí. Ze začátku útočník nahraje na zranitelný server data, na který se posléze dotazuje s podvrženou adresou oběti. Druhá varianta útoku spočívá v zaslání požadavku `stats` na server, který generuje velkou odpověď.

Pravidlo pro dotaz

Pravidlo pro nahrání dat na server je možné nalézt v databázi *Emerging Threats* [6] s označením `SID: 2025401`. Toto pravidlo se zaměřuje na požadavek `set`, který jak název napovídá umisťuje obsah na server. Na tento obsah se poté útočník dotazuje příkazem `get`. Návrh pravidla pro požadavek `get`:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 11211 (
    msg:"Possible Memcached DDoS Query GET";
    content:"get"; nocase; depth:12;)
```

- `11211` – číslo portu pro Memcached službu je 11211.
- `get` – požadavek, na který se má Suricata zaměřit.
- `nocase; depth:12;` – Suricta nebude hledět, jestli je content malými nebo velkými písmeny. Zároveň se tento obsah musí objevit v prvních 12 bajtech.

Toto pravidlo je ovšem velice obecné a může zahazovat legitimní provoz, proto jeho samostatné použití není úplně nejlepší. Zde je dobré ho použít v kombinaci s pravidlem pro dotaz `set` za použití stavové podmínky `flowbits`. Pravidlo pro `set` nastaví podmínku a teprve poté může být zkoumáno pravidlo `get`. K těmto dvěma pravidlům je dobré uvést také pravidlo pro odpověď, jež se aktivuje poté, co byly zaznamenány předchozí požadavky. Pravidlo zde slouží spíše pro monitorování. Nastavení `threshold` je nutné u tohoto pravidla mít takové, aby teprve při větších počtech za jednotku času generovalo upozornění. Pokud bychom místo akce `alert` použili `drop`, dosáhneme toho, že můžeme zahazovat legitimní dotazy.

Druhá možnost, jak může útočník vygenerovat velikou odpověď, je zaslat požadavek `stats`. Pravidlo pro tento požadavek:

```
drop udp $EXTERNAL_NET any -> $HOME_NET 11211 (
  msg:"Possible Memcached DDoS Query STATS";
    content:"stats"; nocase; depth:13;)
```

- `11211` – číslo portu pro Memcached službu je 11211.
- `stats` – požadavek ukrytý uvnitř paketu, na který se má Suricata zaměřit.
- `nocase; depth:13;` – Suricata nebude hledět, jestli je content malými nebo velkými písmeny. Zároveň se tento obsah musí objevit v prvních 13 bajtech.

Tento provoz není běžný a slouží spíše pro zjištění stavu serveru, popřípadě pro zjištění problému se serverem. Proto by nemělo přicházet z cizí sítě.

Pravidlo pro odpověď

Pro odpověď na jediný `get` požadavek může server vygenerovat vícero odpovědí (paketů), jelikož se obsah jedné hodnoty nemusí vlézt do jednoho paketu. Je proto nutné vytvořit dvě pravidla. Jedno pro prvotní paket, který obsahuje hodnotu `VALUE`, a druhé pravidlo pro následující pakety, u Memcached protokolu se jedná o tzv. *Continuation*.

```
drop udp $EXTERNAL_NET 11211 -> $HOME_NET any (
 msg:"Possible Memcached DDoS Response VALUE";
          content:"VALUE"; depth:13;
     flowbits: set, ddos.memcached.value;)
```

- `11211` – Memcached využívá 11211 pro číslo portu.
- `VALUE` – odpověď serveru na dotaz `get` uložena v paketu.
- `depth:13;` – content se nachází v prvních 13 bajtech.
- `flowbits:set,ddos.memcached.value;` – nastavení stavové podmínky pro následující pravidlo.

Pravidlo pro *Continuation* nelze bohužel nikterak blíže specifikovat.

```
drop udp $EXTERNAL_NET 11211 -> $HOME_NET any (
msg:"Possible Memcached DDoS Response Continuation";
     flowbits: isset, ddos.memcached.value;)
```

U tohoto pravidla se kontroluje, zda byl viděn předchozí paket `VALUE`, pomocí stavové podmínky `flowbits: isset, ddos.memcached.value;`.

U těchto pravidel je důležité, zda je Suricata umístěna u Memcached serveru, kdy dochází k filtraci odchozího provozu, nebo je umístěna tak, že filtruje provoz směrem do lokální sítě. Pokud se filtruje odchozí provoz, je dobré nastavit `threshold` vysoko pro obě pravidla, aby se začal zahazovat provoz až při velkém počtu paketů. S tím souvisí nastavení sledování pravidla podle cílové adresy (`by_dst`). Není dobré zahltit nějakou adresu potenciálně velkým počtem paketů.

V případě, kdy dochází k filtraci příchozího provozu, je nutné nastavit sledování podle zdrojové adresy (`by_src`) a mitigovat požadavky pouze jednoho serveru. Nedojde tak k zablokování serveru, který posílá požadavky v povolené míře.

Pravidlo pro odpověď na požadavek `STATS` můžeme nalézt v databázi *Emerging Threats* [6], jeho označení je `SID:2025402`. Lehce upravené pravidlo má následující tvar:

```
drop udp $EXTERNAL_NET 11211 -> $HOME_NET any (
msg:"ET DOS Possible Memcached DDoS Response STAT";
        content:"STAT|20|pid"; offset:8;)
```

- `11211` – číslo portu pro Memcached službu je 11211.
- `STAT|20|pid` – v paketech se na začátku objevuje sekvence `STAT pid`, hexadecimální hodnota 20 znamená mezeru.
- `offset: 8;` – content je nutné hledat po prvních 8 bajtech od začátku memcached payloadu.

Stejně jako v případě pravidla pro dotaz `stat`, tento provoz by neměl přicházet z vnější sítě.

SSDP

V databázi *Emerging Threats* [6] je možné nalézt několik variant pravidel pro eliminování SSDP DDoS útoku. Konkrétně jde o tato pravidla `SID: 2101917, 2008094, 2019102`. Nejdůležitější je pravidlo, které se zaměřuje na generování dotazu `ssdp:all`, jelikož zde dochází k největší amplifikaci.

Pravidlo pro dotaz

Následující návrh pravidla je modifikací pravidla pro `ssdp:all`, upravil jsem obsah druhého `content`, aby odpovídal paketům, jež jsem zkoumal.

```
drop udp $EXTERNAL_NET any -> $HOME_NET 1900 (
    msg:"ET DOS Possible SSDP Querry ALL";
content:"M-SEARCH|20|*|20|HTTP/1.1"; nocase; fast_pattern;
                content:"ST:ssdp:all";)
```

- `1900` – protokol SSDP používá 1900 jako číslo portu.
- `M-SEARCH|20|*|20|HTTP/1.1` – jedná se o požadavek `M-SEARCH`.
- `nocase; fast_pattern;` – content není závislý na tom, zda jsou použita malá nebo velká písmena. Tento vzor bude hledán jako první v paketu.
- `ST:ssdp:all` – ST (angl. Search Target), specifikace požadavku na odpověď všech zařízení, která podporují SSDP protokol.

Pokud požadavky na SSDP protokol pocházejí z vnější sítě, nemělo by se na ně vůbec odpovídat.

Pravidlo pro odpověď

Pro odpovědi SSDP útoku není jednoznačný vzor, na který by se v paketu dalo spolehlivě a jednoznačně zaměřit, jediný společný vzor je `HTTP/1.1 200 OK`.

```
alert udp $EXTERNAL_NET 1900 -> $HOME_NET any (
        msg:"Possible SSDP Response";
content:"HTTP/1.1|20|200|20|OK"; nocase;)
```

- `1900` – protokol SSDP používá 1900 jako číslo portu.
- `HTTP/1.1|20|200|20|OK` – informace, která se objevuje v odpovědi na SSDP dotaz.
- `nocase;` – content není závislý na tom, zda jsou použita malá nebo velká písmena.

Pokud tyto požadavky přicházejí z vnější sítě, neměli bychom na ně vůbec reagovat. Musíme mít dobře nastavenou proměnnou `$EXTERNAL_NET`, aby obsahovala všechno, kromě lokálních IP adres (obsah `$HOME_NET`). Účinné může být také zablokování všech příchozích paketů na UDP port 1900.

OpenVPN

Při tomto útoku dochází ke generovaní `P_CONTROL_HARD_RESET_SERVER_V2` paketů serverem. OpenVPN spojení zahajuje klient zasláním zprávy `P_CONTROL_HARD_RESET_CLIENT_V2`, která zároveň může spustit tento útok. Jelikož se jedná o legitimní provoz, nelze ho bezmyšlenkovitě odfiltrovat. Při návrhu pravidla se tudíž soustředím pouze na vygenerované odpovědi serverem. Těch může být až 60 za 30 sekund na jediný dotaz.

Pravidlo pro odpověď

Pravidlo se zaměřuje na odpověď `P_CONTROL_HARD_RESET_SERVER_V2`.

```
drop udp $EXTERNAL_NET 1194 -> $HOME_NET any (
msg:"DDoS OpenVPN Response - Hard Reset Server V2";
content:"|40|"; startswith;
flow:not_established;)
```

- `1194` – OpenVPN protokol používá číslo portu 1194.
- `40` – typ odpovědi `P_CONTROL_HARD_RESET_SERVER_V2` má hexadecimální hodnotu 40.
- `startswith` – typ OpenVPN paketu je uložen v hlavičce a nachází se na prvním místě, `content` se proto musí vyskytovat hned jako první.
- `flow:not_established` – určení, že pakety nejsou součástí žádného ustanoveného spojení. Pro UDP to znamená, že Suricata nezaznamenala pakety z obou stran komunikace, tedy jak ze strany serveru, tak i klienta.

Vlastnost `flow:not_established` je zde velice důležitá. Díky této podmínce jsme schopni rozlišit legitimní nebezpečný provoz. Legitimní provoz, totiž již bude zahájen ze strany klienta.

Můžeme vytvořit i pravidlo typu `pass` pro konkrétní OpenVPN server, se kterým je komunikace považována za legitimní. Pokud tato varianta nepřipadá v úvahu, například pokud jako ISP nemám tušení, které servery budou klienti potenciálně využívat, tak nám může pomoct právě nastavení `threshold`.

5.5 Pomalé útoky

Pomalé útoky bývá velice těžké odhalit, jelikož jejich chování je velice podobné běžnému spojení. Nejlepší by bylo, kdyby se dalo uvnitř Suricaty zjistit, jak dlouho trvá dané spojení. Suricata sice umožňuje upřesnit pravidlo podle celkové velikosti datového toku (klíčové slovo `stream_size`), ale s jednotkou času se to příliš efektivně spojit nedá. Pro filtrování slow útoků jsem proto vytvořil vždy sérii 4 pravidel.

Pravidlo 1: inicializuje proměnnou, která je vytvořena pro každý datový tok:

```
flowint: slowattack.count, notset; flowint: slowattack.count, =, 1;
```

Pravidlo 2: inkrementuje tuto proměnnou pro každou shodu pravidla:

```
flowint: slowattack.count, notset; flowint: slowattack.count, +, 1;
```

Pravidlo 3: provádí akci `rejectboth`, které zašle RST pakety na obě strany komunikace (pro server i klienta) v případě, že dojde k překročení zadané hodnoty proměnné. Zároveň toto pravidlo nastavuje tzv. `xbits`, který si poznačí danou zdrojovou IP adresu a vloží ji do tabulky hostů na dobu stanovenou vlastností `expire`:

```
slowattack.count, isset; flowint: slowattack.count, >, 4;
  xbits:set, slowattack.ban, track ip_src, expire 180;
```

Pravidlo 4: blokuje veškeré nové pokusy o spojení pro IP adresy nacházející se v tabulce hostů, do které se dostaly na základě předchozího pravidla. Celý návrh pravidla:

```
drop tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (
     msg:"Drop balcklisted IP - reason: slowatack";
       xbits: isset, slowlorisban, track ip_src;
```

Všechny tyto části pravidel jsou společné pro následující pomalé útoky. Rozdílný je pouze vzor, který se bude v paketech hledat, aby byl klasifikován jako slow útok. Pro první dvě pravidla je nutné nastavit akci `alert`. Pokud nechceme, aby tyto dvě pravidla generovala upozornění do logovacího souboru, můžeme u nich uvést klíčové slovo `noalert`. Můžeme použít i klíčové slovo `threshold`, a omezit tak tento počet generovaných upozornění.

Slowloris

V první fázi útočník zašle úvodní dotaz na server obsahující metodu, tedy `GET`. Nedojde však k ukončení dotazu. Útočník se následně odmlčí na menší časový interval, aby simuloval fungování pomalého stroje. Poté zašle další část dotazu, ale data neobsahují ukončení spojení. Toto se neustále opakuje.

Co může být obsahem těchto částečných dotazů, nelze příliš specifikovat. Já jsem se nicméně zaměřil na vzory, které generují nástroje `slowhttptest` a `slowhttptest`. Tyto nástroje slouží ke generování slow útoků.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (
        msg:"Possible Slowloris Attack Detected";
flow:to_server, established; content: "X-"; distance:0; fast_pattern;
        pcre:"/.*\: .*/"; content:"|0d 0a|"; endswith;)
```

- `$HTTP_PORTS` – globální proměnná obsahující čísla portu pro HTTP servery (obvykle 80 a 443).
- `flow:to_server, established;` – důležité jsou pouze pakety mající datový tok směrem k serveru a pro daný datový tok byl zaznamenán úvodní TCP `three-way-handshake`.

- `content: "X-"; distance:0; fast_pattern;` – nástroje **slowhttptest** a **slowloris** posílají pomalá data, která začínají hodnotou `X-`. Jelikož pravidlo obsahuje více klíčových slov **content**, je dobré označit pro *MPM* některé z nich. Já jsem zvolil toto, aby se nebral jako první vzor k porovnání regulární výraz, jež následuje.
- `pcre:"/.*\: .*/";` – regulární výraz představující řetězec libovolných znaků. V řetězci se musí alespoň jednou objevit kombinace znaku dvojtečka a mezera.
- `content:"|0d 0a|"; endswith;` – na konci datové části se objevují znaky pro přesun kurzoru na začátek (angl. Carriage return) a znak nového řádku.

Slowread

Během Slowread útoku dojde k odeslání celého **GET** dotazu směrem na server. Server následně odpoví, ovšem útočník simuluje pomalé spojení a tedy mění *TCP Window*. TCP Window slouží k určení maximálního počtu bajtů, které mohou být zaslány, aniž by byly nuceny být potvrzeny ACK paketem. Takto útočník čte data co nejvíce pomalu a zbytečně plýtvá zdroje serveru.

Je tedy možné se zaměřit na počet nulových oken, jež zasílá klient serveru:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (
msg:"Possible slowread DDoS - large amount of TCP Zero Window";
flags:A; window:0;)
```

- `$HOME_NET` – globální proměnná obsahující čísla portu pro HTTP servery (obvykle 80 a 443).
- `flags:A; window:0;` – cílem pravidla jsou pakety s nulovou velikostí okna, zároveň se jedná o ACK typ pakety zasílané klientem.

Pro tyto pakety je ovšem nutné nastavit hodnotu proměnné, při jejímž překročení Suricata zašle RST pakety. Můžeme zde totiž i ukončit legitimní spojení.

5.6 Shrnutí

Při návrhu pravidel pro SYN flood útok se lze bohužel zaměřit pouze na určitou formu limitování provozu. Nelze tady jednoznačně určit pomocí pravidla, zda se jedná o legitimní pokus o navázání spojení, nebo zda se jedná o útok. Pro zbylé útoky se můžeme opřít alespoň o uchování stavu datového toku.

Pro amplifikační útoky se pravidla navrhovala nejlépe. Je vždy nutné zjistit informace o chybě, kterou nebezpečný dotaz zneužívá. Následně tento dotaz lokalizovat uvnitř paketu, a pokud se tam nachází, tak tento paket filtrovat. Navíc se řada těchto dotazů (NTP, SSDP, DNS...) má správně objevit pouze na lokální síti, kde operuje daný server, neměly by se proto příliš objevovat mimo lokální síť. Lze tak snadno blokovat všechny tyto pakety přicházející zvenčí do naší sítě.

Pro potlačení pomalých útoků jsem navrhl vždy sérii 4 pravidel. Především proto, aby bylo možné dočasně zablokovat IP adresu útočníka, a tím mu znemožnit provést daný útok znovu. Tato pravidla využívají akce **rejectboth**, kdy Suricata zašle RST pakety na obě strany komunikace, čímž tím by mělo dojít k ukončení spojení. Nelze tady totiž jednoduše pouze zahazovat pakety jako u předešlých útoků, ale musíme i ukončit dané spojení,

a uvolnit tak zdroje na serveru. Taktéž je nutné si ujasnit, kdy budeme chtít daná pomalá spojení ukončit. Pokud hranici nastavíme příliš nízko, ohrozí se tím i legitimní spojení, pokud necháme útok běžet delší dobu, plýtváme tím zdroji serveru.

Nutno podotknout, že mnou navržená pravidla nemusí pokrýt veškeré modifikace a zvláštnosti, kterými se dané útoky mohou projevovat.

Kapitola 6

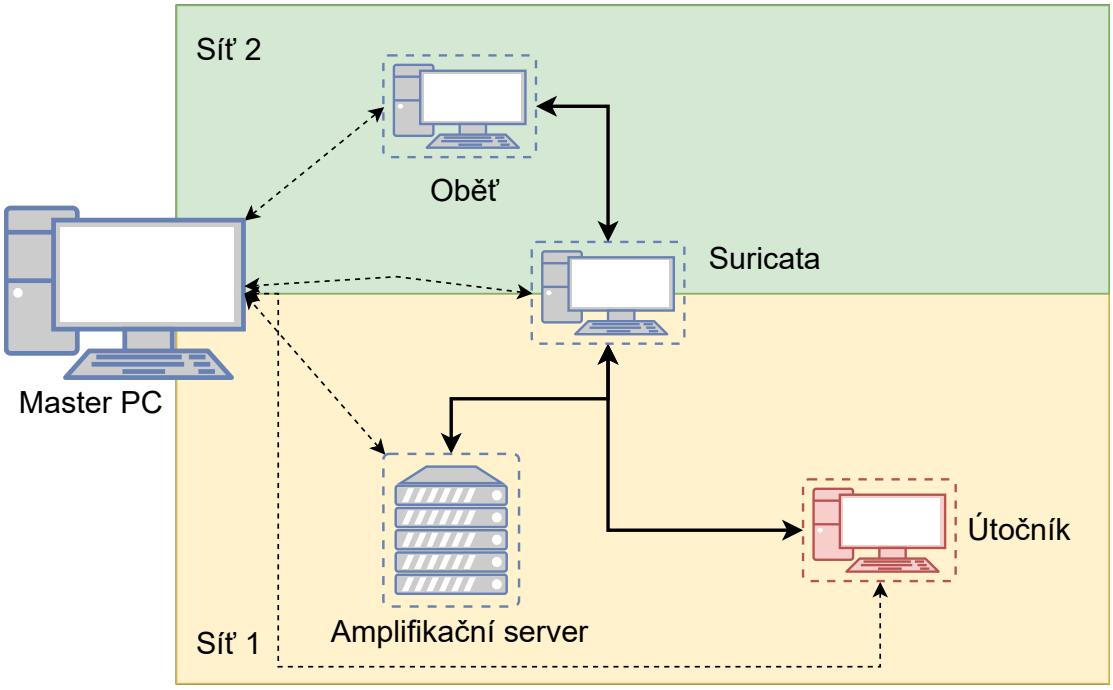
Implementace testovacího prostředí

Testovací prostředí hraje podstatnou roli v odzkoušení jednotlivých pravidel a ověření, zda Suricata dokáže správně odfiltrovat nebezpečný útok. V této kapitole je blíže popsaná konfigurace jednotlivých uzlů testovacího prostředí. Tyto uzly představují subjekty, které se během skutečného útoku typicky objevují. Snahou testovacího prostředí tak bude napodobit reálný útok.

Pro testovaní jsem zvolil i vzhledem k probíhající pandemické situaci virtuální prostředí s tím, že veškerá konfigurace, testování a simulace útoků je co nejvíce automatizovaná a dá se tedy lehce přenést do reálného laboratorního prostředí, které nabízí lepší výkon a větší možnosti, než v případě virtuálního prostředí. Je také daleko vhodnější pro simulování DDoS útoků. Jako první bylo třeba zvolit software pro správu virtuálních strojů. Jelikož můj primární OS je Fedora, zvolil jsem *Virtual Machine Manager*, který je na tomto systému primárně podporován. Jedná se o aplikaci s uživatelským rozhraním, která používá *libvirt*, což je open-source API a knihovna pro virtualizaci. Lze ji taktéž ovládat pomocí příkazové řádky v terminálu. Byl použit *KVM/QEMU* modul Linuxového kernelu jako hypervizor (emulátor virtuálního prosředí) [49].

Na obrázku 6.1 je znázorněno zapojení jednotlivých strojů ve virtuálním prostředí. Vytvořil jsem dvě sítě, abych oddělil zařízení oběti od zbytku. Pro oběť je výchozí bránou Suricata, přes kterou prochází veškerý provoz od a k oběti. Plnými čárami je znázorněno spojení mezi jednotlivými stroji ve virtuálním prostředí. Přerušovaná čára znázorňuje, že PC, na kterém běží virtuální prostředí, je spojeno se všemi stroji. Je tak učiněno, aby tento počítač fungoval jako velící stroj, ze kterého je poté možné konfigurovat všechna ostatní zařízení pomocí SSH. Z hlediska přenositelnosti do fyzického prostředí je toto velice výhodné. Dá se poté veškerá činnost stále ovládat a kontrolovat z jediného management (master) uzlu. Ve virtuálním prostředí je výchozí nastavení, že má hostitelský stroj přístup ke všem ostatním. V laboratorním prostředí je nutné jednotlivé cesty nastavit.

Záplavové útoky jsou generovány s falešnou zdrojovou adresou. Docílí se tak toho, že útok se bude chovat, jako by se jednalo o distribuovaný útok z vícero zařízení. Pro amplifikační útoky je možné generovat pakety, které směřují na potenciálně zranitelný server, a tak ověřit funkčnost pravidel pro tento případ. Pro generováni amplifikovaných odpovědí se v testovacím prostředí nachází Amplifikační server. Ten dokáže pakety pro vybrané útoky (DNS, NTP, Memcached) amplifikovat a odrazit směrem k oběti. Chová se tedy stejně jako skutečný server, který by mohli útočníci zneužít. U zbylých amplifikačních



Obrázek 6.1: Schéma virtuálního prostředí

útoků se používají pro amplifikované odpovědi PCAP soubory. V nich je zachycena reálná komunikace, která se objevuje při daných DDoS útocích. Během pomalých útoků je nutné, aby na uzlu oběti běžel HTTP server.

6.1 Automatizace

Aby bylo možné jednoduše přenést veškeré odzkoušení DDoS útoků v laboratorním nebo jakémkoliv jiném prostředí, rozhodl jsem se veškerou činnost co nejvíce automatizovat. Vytvořil jsem řadu skriptů, které jsou napsány v Bashi, předpokladem je, že veškerá činnost bude prováděna na operačních systémech Linux. Nejdůležitější jsou dva hlavní skripty, jeden pro konfiguraci jednotlivých zařízení a druhý pro simulaci DDoS útoků. Jak tyto skripty použít je zmíněno v příloze A. Oba dva skripty využívají konfigurační soubor, kde jsou uloženy proměnné a hodnoty specifické pro dané prostředí. Před použitím těchto skriptů je nutné upravit tento soubor, aby odpovídal konkrétnímu prostředí, ve kterém je použit. Pro oba skripty je zásadní, aby bylo možné se přihlásit na jednotlivé stroje pod uživatelem `root` pomocí SSH. K autentizaci se používá kombinace veřejného a privátního klíče. Privátní klíč je uložen na master PC. Na ostatních zařízeních je uložena kopie veřejného klíče. Není tak potřeba žádného hesla. Pokud stejné prerekvizity budou splněné i ve fyzickém prostředí, je možné jednotlivé skripty jednoduše použít.

Pro automatizaci konfigurace je vytvořen jednoduchý skript `set_up.sh`, který nastaví všechny potřebné záležitosti pro daný stroj. Uživatel vždy vybere, který z počítačů chce nakonfigurovat. Konkrétní kroky jsou zmíněny pro každé zařízení zvlášť v následujících sekcích. Tento skript využívá dalších pomocných skriptů, které jsou uloženy v adresářové struktuře odpovídající názvům zařízení, kterých se týkají. Jelikož chci, aby skript co nejméně modifikoval zařízení, tak nedochází k nainstalování potřebných balíčků, nástrojů

a závislostí, ty je nutné nainstalovat manuálně. Jejich konkrétní seznam je v příloze A. Názvy se mohou lišit od použitého operačního systému.

Pro simulaci DDoS útoků se využívá skript `launch_test.sh`. Uživatel zvolí typ útoku a následně se spustí DDoS útok podle parametrů. Tento skript je možné použít až poté, co byl úspěšně zavolán konfigurační skript `set_up.sh`. Jeho fungování je rozděleno na několik částí (fází). Jako první se nahraje nejaktuálnější soubor se seznamem pravidel pro konkretní typ útoku na Suricatu. Pouze tento seznam pravidel je brán jako aktivní, ostatní soubory s pravidly nejsou brány v potaz. Poté dojde k resetování Suricaty, děje se tak jednak, aby se použila pouze pravidla pro konkrétní útok, ale především, aby se vynulovaly čítače Suricaty. Tyto čítače zaznamenávají různé statistiky, například počet přijatých a zamítnutých paketů nebo celkový objem příchozího provozu. Následuje část, kdy se skript připojí pomocí SSH na útočníka a spustí z něho daný DDoS útok. Některé útoky je možné ještě různě modifikovat nebo upravit pomocí proměnných, jež jsou uloženy v souboru `attacker/attack_spec.sh`. I tento soubor musí být modifikován, aby odpovídal hodnotám konkrétního prostředí. Poté co útok skončí, dojde k výpisu statistik útočníka a Suricaty. Pokud se jedná o pomalý útok (slow), skript zajistí, že se před útokem spustí HTTP server na oběti a po útoku dojde zase k jeho vypnutí.

Pro získávání statistik ze Suricaty se využívá `suricatasc`, tedy připojení na socket Suricaty, přes který jí lze zadávat příkazy bez nutnosti restartovat Suricatu. Tento způsob umožňuje taktéž vypsat aktuální stav různých čítačů. Výstup je ve formátu JSON, ze kterého jsou následně vytaženy potřebné informace.

Pro získání statistik o počtu vygenerovaných paketů útočníkem a jejich celkové velikosti se využívá nástroj `iptables`. Jedná se o nástroj, který umožňuje linuxovému nebo unixovému systému plně pracovat se síťovou komunikací. Zde je využit pouze pro monitorování a pro sběr informací o odchozích paketech útočníka.

6.2 Suricata

Pro počítač, na kterém běží Suricata, jsem zvolil operační systém Ubuntu 16.04, aby byl systém stabilní. Suricata funguje zároveň jako směrovač a výchozí brána mezi obětí a zbytkem sítě. Je tedy nutné nastavit dvě síťová rozhraní, pro každou síť jedno. V mém prostředí jsem nainstaloval Suricatu ve verzi 6.0.2, v době provádění testovaní se jednalo o poslední dostupnou verzi. Na Suricatě je nutné nastavit směrování podle obrázku výše 6.1, dále nastavit IP forwarding u Linuxového jádra, aby zařízení fungovalo jako směrovač a přeposílalo pakety, které nejsou určeny pro koncové zařízení, na kterém běží Suricata. Vzhledem k tomu, že Suricata má fungovat ne jenom jako detekční, ale i mitigační zařízení (tedy IPS), je nutné zajistit, aby pakety byly směrovány do Suricaty. K tomu můžeme využít NFQ (angl. NetfilterQueue), pro potřeby virtuálního prostředí je tato možnost zcela dostačující z hlediska výkonu. Nastavení NFQ se dá docílit pomocí `iptables`:

```
iptables -I FORWARD -j NFQUEUE
```

Suricata se poté musí spustit s parametrem `-q 0`. Suricata poté z NFQ začne odebírat pakety a začne je zpracovávat. Po inspekci se musí Suricata postarat o to, aby přeposlala legitimní provoz dále. Je dobré nastavit, aby Suricata běžela jako démon[1]. K tomu je nutné upravit soubor `/etc/default/suricata` tak, aby se Suricata automaticky zapínala v NFQ režimu. O veškerou tuto konfiguraci se stará konfigurační skript `set_up.sh`.

[1] démon – obecně se jedná o proces nebo službu běžící na pozadí

6.3 Oběť

Oběť, stejně jako Suricata, běží na Ubuntu 16.04. Konfigurace zde zahrnuje pouze nastavení směrovací tabulky tak, aby odkazovala na Suricatu jakožto výchozí bránu. V případě pomalých (slow) útoků je nutné, aby na oběti běžel HTTP server. Pro testovací účely ve virtuálním prostředí bohatě stačí Python modul `ComplexHTTPModul`, který podporuje vícevláknové zpracování a je jednoduché ho nastavit. Abych docílil toho, že server poběží i po skončení SSH relace, využil jsem nástroj `tmux`. Jedná se o terminálový multiplexer, který obdobně jako `screen` umožňuje mít otevřeno vícero oken v rámci jednoho terminálu. Já jsem ho nicméně využil k tomu, abych pomocí něj zahájil relaci, ve které se spustí HTTP server, jehož činnost nebude narušena i přesto, že dojde k ukončení SSH spojení. Po provedeném slow útoku je možné se k této relaci vrátit a server bezpečně ukončit. Konfigurační skript nastavuje Suricatu jako výchozí bránu a kontroluje, zda jsou nainstalovány nástroje pro slow útok. Poté nahraje textový soubor o velikosti 1 MB na uzel oběti. Tento soubor slouží pro Slowread útok, během něhož se útočník dotazuje na tento soubor.

6.4 Útočník

Jako OS pro útočníka jsem zvolil Kali Linux 2020.4, především proto, že tento operační systém je mj. určen pro různá bezpečnostní a penetrační testování. Má proto řadu nástrojů již nainstalovaných. Z hlediska směrování je nutné ve virtuálním prostředí nastavit cestu do sítě oběti skrz Suricatu, jinak by útočník posílal pakety skrz hostitelské PC. Konfigurační skript přetáhne na stroj útočníka veškeré skripty, soubory a programy, které se používají během DDoS útoku. Nad těmi programy, které to vyžadují, je zavolán příkaz `Make`, aby se zjistilo, zda je možné daný program spustit, nebo je nutné doinstalovat závislosti, knihovny a podobně. Veškeré závislosti jsou zmíněny v příloze A. Během spuštění skriptu pro DDoS útok se master PC připojí na útočníka a spustí zvolený DDoS útok. Útoky se generují rozdílně, každý je blíže popsán v následující kapitole 7.

6.5 Amplifikační server

Pro reflexivní útoky se používá amplifikační server, který na dotazy útočníka odpovídá daleko většími pakety. Pro simulaci NTP, DNS a Memcached amplifikačních útoků je využito lokálního serveru, na kterém je nakonfigurována zneužitá služba. Pro své testovací prostředí jsem stejně jako v případě oběti a Suricaty využil OS Ubuntu 16.04. Na této verzi je dostupná řada starších balíčků, které mohou obsahovat neopravené verze služeb. Zároveň to do jisté míry simuluje dlouho neaktualizovaný software, kdy se na řadu serverů dlouhá léta nesáhlo. Během konfiguračního skriptu je nastaveno směrování do sítě oběti skrz Suricatu, jinak by byly ve virtuálním prostředí pakety zasílány skrz hostitelské PC. Další kroky jsou uvedeny zvlášť pro každý typ serveru.

NTP

Pro NTP amplifikaci je nutné nainstalovat NTP balíček ve verzi nejvýše `ntp-4.2.7p26`, já jsem použil konkrétně `ntp-4.2.6p2`. Vyšší verze již nepodporují příkaz `MON_GETLIST_1`, který je k amplifikaci potřeba. Dále je nutné upravit konfigurační soubor `/etc/ntp.conf`, aby povolil tento typ dotazu. Během konfiguračního skriptu pro NTP se mimo jiné z útočníka spustí skript, který zasílá informace na NTP server a vytvoří na něm 600 falešných

záznamů. Na tyto záznamy se poté během amplifikačního DDoS útoku útočník dotazuje pomocí příkazu `MON_GETLIST_1`. Šest set záznamů proto, jelikož tento dotaz vrací informace o nejvýše 600 záznamech.

Memcached

Pro Memcached server jsem použil verzi balíčku `1.4.25`. Pro konfiguraci je nutné pouze upravit soubor `/etc/memcached.conf`, aby umožňoval UDP spojení na portu 11211, ve výchozím nastavení je toto od verze `1.5.6` vypnuté. Konfigurační skript přetáhne potřebný soubor s upravenou konfigurací na server a restartuje službu.

DNS

Pro DNS server na Ubuntu je třeba nainstalovat balíčky `bind9, dnsutils`. DNS servery mají ve výchozím nastavení do jisté míry ochranu proti IP spoofingu. Je proto nutné povolit dotazy z cizích sítí/domén, například:

```
allow query { any; };
```

Dále je nutné upravit soubor `named.conf.default-zones`, aby informace o root záznamech hledal v lokálním souboru, a zároveň, aby se k těmto záznamům choval jako typ `master`. Tento soubor s root záznamy lze stáhnout z webu organizace IANA[2]. Soubor je dost obsáhlý a tedy silně amplifikuje odpověď serveru. Konfigurační skript nahraje všechny tyto potřebné soubory na server. Jména balíčků a umístění konfiguračních souborů se mohou lišit pro různé operační systémy.

[2] https://www.iana.org/domains/root/files

Kapitola 7

Ověření filtrace DDoS útoků

V této části se věnuji praktickému odsimulování jednotlivých DDoS útoků a následnému ověření, že filtrování Suricatou podle navržených pravidel je opravdu funkční. U každého útoku je uvedeno, jakým způsobem je v testovacím prostředí generován. Pro většinu amplifikačních útoků jsem využil skripty a programy volně dostupné na internetu. Jejich seznam je uveden v příloze B, včetně zdrojů k těmto programům. Snažím se ověřit funkčnost pravidel, jež jsem navrhl pro každý útok v kapitole 5. Po útoku se sbírají statistiky především o tom, kolik bylo vygenerováno paketů útočníkem, kolik paketů bylo Suricatou akceptováno a kolik zablokováno. V případě amplifikačních útoků, jaký objem dat vygeneroval server. I přesto, že se každý útok generuje jinak, snažím se, aby simulace trvala stejnou dobu, tedy 30 sekund. Simulaci jsem vždy 5krát zopakoval, abych měl jistotu, že nedojde k nějaké neočekávané chybě.

Jak je zmíněné v předchozí kapitole 6.1, pro získávání statistik ze Suricaty jsem využil `suricatasc`. Pro monitorování statistik útočníka jsem využil nástroj `iptables`. Tyto informace zobrazí nástroj pro generování útoků (`launch_test.sh`) na standardní výstup po dokončení útoku. Pokud jsou hodnoty v `iptables` vysoké, jsou zobrazeny zaokrouhleně na tisíce.

7.1 Volumetrické útoky

Záplavové útoky jsou simulovány v testovacím prostředí pomocí nástroje `hping3`. Nástroj umožňuje testovat řadu věcí, například firewall pravidla, skenování portů, určování MTU na linkách mezi dvěma hostiteli a mnoho dalšího. Nástroj slouží ke generování síťového provozu. Generování je možné ovlivnit mnoha přepínači, kterými `hping3` disponuje. Například možnost nastavit rychlost posílání paketů, velikost TCP okna, sekvenční čísla atd.

Během testování se uvnitř skriptu spouštěl `hping3` následovně:

```
timeout 30 hping3 --rand-source --flood -p 80 -L 0 --<type> <VICTIM_IP>
```

- `timeout 30` – jelikož je nástroj spuštěn s parametrem `flood`, je nutné generování paketů uvnitř skriptu ukončit po dané době, zde 30 sekund.
- `flood` – pakety jsou zasílané směrem k oběti co největší rychlostí, tento režim ignoruje odpovědi serveru.
- `-p 80` – cílový port je 80, toto číslo lze změnit uvnitř konfiguračního souboru, který skript využívá.

- `-L 0` – nástroj `hping3` posílá SYN pakety s náhodnou ACK hodnotou. Při navazování legitimního spojení pomocí three-way-handshake je však pro první SYN paket hodnota ACK a SEQ rovna nule. Suricata ve výchozí konfiguraci již obsahuje ochranu proti nevalidním paketům. Proto je nutné u `hping3` specifikovat generování paketů s číslem ACK 0. Zahazování nevalidních paketů Suricatou je možné ovlivnit položkou `drop-invalid` v jejím konfiguračním souboru (`suricata.yaml`). Pro RST bylo nutné nastavit také přepínač `-b`, který do paketů vloží špatnou hodnotu checksum. Zde se mi nepodařilo přijít na to, proč tomu tak je, ale bez použití tohoto přepínače jsou pakety Suricatou považovány za nevalidní. Tyto položky se týkají pouze TCP flood útoků.
- `--<type>` – určuje typ zaslaného paketu. Podle útoku je to buď `syn`, `rst`, `ack`, nebo `udp`.

Prvně byl testován každý typ útoku zvlášť a poté byly testovány všechny čtyři varianty dohromady. U nastavení `thresholdu` pravidel jsem se zaměřil na limitování provozu. Zvolil jsem, že Suricata nesmí propustit více než 200 paketů (SYN, RST, ACK, UDP) za sekundu. Útočník v mém prostředí dokáže pomocí `hping3` generovat až 28 tisíc paketů za sekundu. Nechci zbytečně zatěžovat celý systém, na kterém mi funguje virtuální prostředí, tím, že nechám většinu těchto paketů projít až k oběti útoku. V reálném prostředí je však nutné maximální počet paketů omezit podle kapacity chráněného zařízení.

```
threshold: type both, track by_dst, count 201, seconds 1;
```

Bohužel zde nemá smysl nastavit sledování pravidla podle zdrojové adresy, neboť útočník si může vytvořit nesčetné množství zdrojových IP adres.

Výsledky testování

Suricata	SYN	ACK	RST	UDP	Zároveň
Celkem paketů	404k	559k	732k	537k	1956k
Přijato paketů	4 000	6 300	6 080	6 314	83 004
Zamítnuto paketů	400k	553k	726k	531k	1873k
Vygenerovaných upozornění	30	30	30	30	120
Útočník	SYN	ACK	RST	UDP	Zároveň
Celkem vygenerovaných paketů	435k	633k	788k	591k	2217k
Celkem bajtů	17M	25M	32M	17M	82M
Velikost paketu	40	40	40	28	3x40,28

Statistiky ohledně počtu paketů a celkového objemu dat, které jsem získal z útočníka a ze Suricaty se lehce liší. Při bližším zkoumání ve Wiresharku bylo vidět, že dané pakety dorazí až k Suricatě, ale ta je zahodí i v případě, kdy nejsou použita žádná pravidla. Přitom tyto pakety Suricata nezahrne do svých statistik. Nejedná se ani o nevalidní pakety. Bohužel jsem nedokázal přijít na to, co způsobuje tento efekt.

Nicméně Suricata při všech testech správně vygenerovala 30 upozornění, jedno každou sekundu. Při zvoleném prahu maximálně 200 paketů za sekundu bych očekával nejvýše 6 000 přijatých paketů. To se podařilo splnit pouze v případě SYN flood útoku. Zbylé útoky tuto hranici pokořily. Pro tyto útoky jsem následně zkoušel odebrat kontrolu toku (vlastnost `flow`) u pravidel, nedošlo však k žádné změně. Může to být způsobeno zpožděním nástroje `timeout`, který ukončuje generování paketů.

7.2 Amplifikační útoky

Pro amplifikační útoky typu DNS, NTP a Memcached existuje ve virtuálním testovacím prostředí server, který je nakonfigurovaný tak, aby se dal zneužít k amplifikaci a reflexi útoku na oběť. Před útokem byl vždy zavolán konfigurační skript pro daný typ serveru. Pro ostatní typy jsem využil PCAP soubory obsahující zachycenou komunikaci reálného útoku. Abych byl tento PCAP soubor schopen zreprodukovat, použil jsem open-source nástroj Tcpreplay. Slouží k editaci a znovu přehrání zachycené síťové komunikace. Při použití Tcpreplay ovšem nedochází k zaznamenání provozu uvnitř `iptables`. Je to dáno tím, že Linuxový kernel jiným způsobem vkládá a jiným způsobem čte rámce pro `iptables` [46]. V daných případech se odkazuji na statistiky vygenerované nástrojem Tcpreplay.

V tabulkách s hodnotami jsou výsledky pro filtrování nebezpečných dotazů na server a výsledky pro filtrování amplifikovaných odpovědí ze serveru. Testování odpovědí a dotazů probíhalo zvlášť.

Pro amplifikační útoky vyjma Memcached-get jsem použil následující nastavení:

```
threshold: type both, track by_src, count 2, seconds 60;
```

Zvolil jsem, aby se propustil pouze jeden paket za minutu, protože chci mít jistotu, že daný útok bude potlačen mnou navrženými pravidly. Vyvaruji se tak situace, kdy by teoreticky něco jiného mohlo potlačit daný útok. V reálném nasazení bych však zakázal veškerý provoz.

Pro Memcached-get útok je nutné mít jiné nastavení prahu, jelikož pravidla mohou zahazovat legitimní provoz, a není proto možné mít nastaveny hodnoty prahu tak nízko.

7.2.1 NTP

NTP útok je simulován skriptem napsaným v jazyce Perl. Skript generuje dotaz `MON_GETLIST_1`. Pro testovací amplifikační server je nutné před útokem nahrát na server falešné záznamy. V případě dotazu jsou generované pakety směrovány přímo na oběť, zdrojová adresa je pro všechny pakety stejná. V případě odpovědi jsou pakety zasílané na amplifikační server s podvrženou adresou oběti, která je následně příjemcem paketů.

Výsledky testování

Suricata	Dotazy	Odpovědi
Celkem paketů	100	10 000
Přijato paketů	1	1
Zamítnuto paketů	99	9 999
Vygenerovaných upozornění	1	1
Celkem bajtů	3 600	4 680 000
Útočník	Dotazy/Odpovědi	
Celkem vygenerovaných paketů	100	
Celkem bajtů	3 600	
Velikost dotazu	36	

Testování dopadlo podle očekávání. Suricata správně odhalila nebezpečné pakety a začala je zahazovat. Podle nastaveného `threshold` byl správně propuštěn vždy pouze první paket. Díky tomu, že bylo na serveru uloženo přes 600 záznamů, na které se bylo možné dotázat, se podařilo mnohonásobně amplifikovat dotaz.

7.2.2 DNS

Simulace DNS útoku využívá program napsaný v jazyce C, jenž je volně dostupný pod GNU GPL License v3.0. Skript dokáže generovat reálné dotazy `ANY` na DNS servery. Vyžaduje seznam odražečů/serverů, na které zašle dotazy s podvrženou adresou oběti, kterou lze specifikovat. Pro testování dotazů jsem použil IP adresu oběti pro tento seznam, všechny dotazy tedy mají stejnou zdrojovou IP adresu. V případě testování odpovědí došlo k zaslání dotazů na DNS amplifikační server v testovacím prostředí. Tento server byl předtím nakonfigurován, aby vracel velké odpovědi o kořenových záznamech.

Výsledky testování

Suricata	Dotazy	Odpovědi
Celkem paketů	562 808	1 079 200
Přijato paketů	1	388 115
Zamítnuto paketů	562 807	691 085
Vygenerovaných upozornění	1	1
Průměrná velikost paketu	61	1 317
Maximální velikost paketu	89	1 500
Celkem bajtů	34 331 377	947 951 641
Celkem fragmentů	0	719 499
Složených fragmentů	0	359 700
Využitá paměť (v bajtech)	24 354 304	21 410 304
Útočník	Dotazy	Odpovědi
Celkem vygenerovaných paketů	563k	372k
Celkem bajtů	34M	23M
Velikost dotazu	75	75

Během útoku DNS server některé odpovědi fragmentoval. Tyto fragmenty, které se nepodařilo zařadit do žádného toku, byly propuštěny směrem k oběti. Jedná se o bezmála 380 tisíc paketů. Při bližší analýze zachyceného provozu za využití Wiresharku bylo vidět, že dané odpovědi mají průměrnou velikost 1514 bajtů. Nejedná se tedy o něco, co by se mělo brát na lehkou váhu. Na základě jejich zkoumání jsem navrhl další pravidlo speciálně pro tyto fragmenty:

```
drop ip any any -> $HOME_NET any (
msg:"Possible DDoS DNS (FRAGMENT)";
ip_proto: 17; fragbits: M; flow:no_stream, no_frag;
threshold: type both, track by_src, count 100, seconds 60;
classtype:attempted-dos;)
```

Suricata bohužel nenabízí větší míru specifikace, při použití tohoto pravidla tedy hrozí, že začne filtrovat i legitimní provoz. Proto by měl být `threshold` nastaven optimálně pro danou síť. Mnou zvolené hodnoty jsou pouze ilustrační. Pravidlo se soustředí na fragmenty, které mají v hlavičce paketu uveden UDP transportní protokol a zároveň mají nastavený příznak *more fragments*. Dále nebyly složeny stream-enginem Suricaty (`no_stream`) a nebyly ani sestrojeny z fragmentovaných paketů (`no_frag`). Obecně se fragmentované útoky

těžko eliminují. Suricata v současné chvíli nenabízí lepší řešení. Zavedením tohoto pravidla můžeme ovlivnit legitimní provoz, a zvýšit tak míru false positive. Při použití tohoto pravidla došlo k úspěšnému eliminování i těchto fragmentů a potlačení daného DNS útoku.

7.2.3 CLDAP

Pro testování CLDAP útoku jsem použil PCAP soubor se zachycenou komunikací pokusu o amplifikaci dotazů. Tento soubor jsem vytvořil pomocí skriptu, jenž zasílá dotaz `objectclass=*` na zranitelné servery. Jako seznam serverů jsem využil informace z vyhledávače Shodan, který detekuje zranitelné servery. Skript jsem použil s vlastní zdrojovou IP adresou, proto veškerá komunikace serveru směrovala zpět ke mně. K následné úpravě PCAP souboru jsem použil Wireshark a nástroj Tcpreplay. Vytvořil jsem dva oddělené soubory. První obsahuje pouze dotazy a druhý pouze odpovědi serveru.

Pro simulování dotazů byl soubor s dotazy přehrán 13krát, pokaždé s jinou zdrojovou IP adresou. Pro simulování odpovědí byl PCAP soubor přehrán 53krát, taktéž vždy s jinou IP adresou. Dané hodnoty jsem určil na základě měření, kdy počet opakování odpovídá zhruba 30 sekundám trvání útoku.

Už během zachycené reálné komunikace bylo vidět, že odpovědi serverů jsou často fragmentovány podobně jako v případě DNS 7.2.2. Použil jsem tedy i pravidlo pro filtrování fragmentovaných odpovědí.

Výsledky testování

Suricata	Dotazy	Odpovědi
Celkem paketů	32 566	543 240
Přijato paketů	13	28 704
Zamítnuto paketů	32 552	514 536
Vygenerovaných upozornění	13	106
Průměrná velikost paketu	67	785
Maximální velikost paketů	67	1 492
Celkem bajtů	2 181 922	339 615 960
Celkem fragmentů	0	432 420
Složených fragmentů	0	110 820
Využitá paměť (v bajtech)	7 394 304	7 394 304

Suricata reportovala 13 upozornění pro testování dotazů, což nasvědčuje tomu, že bylo použito 13 různých zdrojových IP adres, čemuž odpovídá hodnota přijatých paketů. Suricata v tomto případě eliminovala útok podle očekávání.

V případě odpovědí je vygenerováno 106 upozornění. Vzhledem k tomu, že byla použita dvě pravidla, tak to činí 53 rozdílných zdrojových IP adres. Nicméně došlo k téměř 29 tisícům propuštěným paketům. Při zkoumání provozu ve Wiresharku na uzlu oběti bylo vidět, že počet přijatých paketů se liší od statistik, které zobrazila Suricata. Paketů, které dorazily až k oběti bylo kolem 5 tisíc. Nedokázal jsem vysvětlit, čím je toto způsobeno, nicméně se jedná pouze o fragmentované pakety, které tento způsobují problém.

Nicméně 5 tisíc paketů, které Suricata propustila, je správné chování. Pravidlo pro fragmenty má totiž nastavený `threshold` na maximálně 99 paketů za minutu od jedné IP adresy. Během útoku bylo použito až 53 zdrojových IP adres.

7.2.4 Memcached

Jak je zmíněno v kapitole pro návrh pravidel 5.4, Memcached útok může nastat buď zasláním dotazu `get` na uložená data nebo zasláním dotazu `stats`. Dotaz `get` netestuji, jelikož se jedná o legitimní komunikaci. Testuji pouze odpovědi generované serverem na tento dotaz.

Výsledky testování dotazu `stats`

Pro simulování tohoto útoku jsem využil program napsaný v jazyce C, který posílá dotazy `stats`. Při posílání dotazů na server je použita jedna zdrojová IP adresa.

Suricata	Dotazy	Odpovědi
Celkem paketů	12 060	11 744
Přijato paketů	1	1
Zamítnuto paketů	12059	11 743
Vygenerovaných upozornění	1	1
Celkem bajtů	518 580	14 263 925
Útočník	Dotazy	Odpovědi
Celkem vygenerovaných paketů	12 060	11744
Celkem bajtů	519k	505k
Velikost dotazu	43	43

Pro testovaní odpovědí i dotazů pravidlo fungovalo podle očekávání, Suricata propustila pouze jeden paket.

Výsledky testování na odpověď `get`

Během testování se na amplifikační server nahrají data o velikosti 3 545 bajtů. Pro ověření funkčnosti pravidel ve virtuálním prostředí je toto dostačující velikost. Maximální velikost dat, kterou lze uložit na server, záleží na konfiguraci serveru. Obecně platí, že je možné uložit 1 MB dat. Celková velikost vygenerovaných dat je tedy závislá na velikosti hodnoty, kterou si uložíme na server. Pro úpravu velikosti v implementovaném nástroji je nutné modifikovat Python skript, jenž nahrává data na server.

Dotazy `get` na danou hodnotu jsou zasílané pomocí nástroje `nping`. Jedná se o nástroj pro generování síťových paketů. Nastavil jsem, aby útočník zasílal dotazy rychlostí 900 paketů za sekundu. Pro pravidla jsem zvolil `threshold` následovně:

```
threshold: type both, count 100, seconds 10, track by_src;
```

Navržená pravidla mohou zahazovat i legitimní dotazy. Proto jejich funkce slouží spíše pro limitování provozu v případě, že by bylo zaznamenáno velké množství Memcached odpovědí za určitý čas. Mnou zvolená hodnota 100 je zde pouze ilustrační. V reálném prostředí je nutné zvolit hodnoty podle očekávané komunikace s Memcached servery.

Suricata	Odpovědi
Celkem paketů	108 000
Přijato paketů	297
Zamítnuto paketů	107 703
Vygenerovaných upozornění	6
Průměrná velikost paketu	930
Maximální velikost paketu	1 428
Celkem bajtů	100 494 000
Útočník	
Celkem vygenerovaných paketů	2 700
Celkem bajtů	1 242 000
Velikost dotazu	46

Testování zde proběhlo podle očekávání, každých 10 sekund bylo propuštěno nejvýše 99 paketů. Zajímavým postřehem je, že Memcached útoky (ať už při použití `stat` nebo `get`) neprodukují velké množství paketů, ale celkový datový objem je i přesto velký. Je to způsobeno tím, že Memcached servery vrací velké odpovědi, a potvrzuje to tedy fakt, že tento amplifikační útok patří k největším.

7.2.5 SSDP

Pro simulaci SSDP útoku jsem si vytvořil vlastní PCAP soubor, podobně jako v případě CLDAP. Využil jsem program napsaný v jazyce C, který posílá dotazy `ssdp:all` na dané IP adresy. Pro seznam zranitelných IP adres jsem použil vyhledávač Shodan, který detekuje zranitelná zařízení. Odpovědi byly zasílané zpět na moji IP adresu, komunikaci jsem zachytil ve Wiresharku a následně upravil a rozdělil do dvou souborů. První PCAP soubor obsahuje pouze SSDP dotazy a druhý soubor pouze odpovědi. Pro přehrání souboru je využit nástroj Tcpreplay.

Pro simulování dotazů jsem PCAP soubor přehrál 140krát, pro simulování odpovědí 109krát. Tyto hodnoty odpovídají 30 sekundám trvání v mém testovacím prostředí. Každé opakování generuje novou zdrojovou IP adresu paketů, ovšem v rozmezí dané sítě. Může se tedy vygenerovat pouze 254 adres a je možné, že se vygeneruje stejná adresa vícekrát. Jedná se o neduh mého nástroje, která by se do budoucna dala zlepšit.

Výsledky testování

Suricata	Dotazy	Odpovědi
Celkem paketů	14 700	130 800
Přijato paketů	113	97
Zamítnuto paketů	14 587	130 703
Vygenerovaných upozornění	113	97
Celkem bajtů	1 734 600	48 886 500

Pro dotazy bylo vygenerováno 113 různých upozornění, tudíž se během testování vytvořilo pouze 113 různých zdrojových IP adres. Zbylých 27 přehrání PCAP souboru vygenerovalo duplicitní zdrojové IP adresy, pro které již dané pravidlo platilo. Obdobná situace nastala i v případě odpovědí. Opět si lze povšimnout, že amplifikační útoky generují daleko větší odpovědi, než jaká je velikost původního dotazu.

7.2.6 OpenVPN

K simulaci OpenVPN útoku jsem použil PCAP soubor nalezený na GitHubu SoftEther VPN projektu[1], kde byla chyba umožňující amplifikaci reportována. V PCAP souboru je nastíněna celá reakce serveru, včetně počátečního paketu od klienta. Zachycenou komunikaci jsem upravil tak, aby obsahovala pouze odpovědi serveru. Jak je zmíněno u návrhu pravidla, generování dotazů zde postrádá smysl, jelikož se jedná o legitimní komunikaci a nejsme schopni ji jakkoliv rozlišit od útoku.

Během testování odpovědí je vždy daný soubor přehrán současně deseti procesy, čímž se simuluje distribuovanost útoku. Každý proces zašle se svojí IP adresou na oběť 60 paketů během 30 sekund.

Výsledky testování

Suricata	Odpovědi
Celkem paketů	600
Přijato paketů	10
Zamítnuto paketů	590
Vygenerovaných upozornění	10
Útočník	Odpovědi
Celkem vygenerovaných paketů	10
Celkem bajtů	560
Velikost dotazu	56

Testování dopadlo podle očekávání. Povolen byl vždy pouze první paket každého procesu. Zde si nejsem zcela jistý, zda vlastnost `flow:not_established` opravdu nezahrnuje legitimní pakety. Do budoucna by bylo dobré mít pro tento typ útoku i současně běžící legitimní spojení, které by tento problém pomohlo objasnit.

Ač se to nemusí zdát, tak OpenVPN může představovat velkou hrozbu. Útočník pomocí jednoho dotazu vygeneruje až 60 odpovědí během 30 sekund. Pokud má k dispozici velkou řadu serverů, může tento útok způsobit značné problémy.

7.3 Pomalé útoky

Pro simulování pomalých útoků je potřeba, aby na uzlu oběti fungoval HTTP server. V mém virtuální prostředí jsem se spokojil s použitím Python modulem ComplexHTTPServer. Ten na rozdíl od SimpleHTTPServer modulu zvládá vícevláknové zpracování požadavků. Server jsem spustil vždy před útokem a po provedené simulaci jej zase vypnul. Server funguje nad portem 80. Pro testování slow útoků jsem použil `slowhttptest` nástroj, který je určen právě k testování DoS zranitelností serverů. Nabízí několik typů útoků: Slowloris, R.U.D.Y., Apache killer a Slow read. U pomalých útoků nemá smysl testovat pouze 30 sekund. Rozhodl jsem se proto nástroj spustit vždy na dvě minuty. Pokud nástroj skončí dříve, je nutné zjistit, jestli se jedná o důsledek použitých pravidel, kdy došlo k ukončení spojení, nebo server již nezvládl obsloužit všechna spojení.

Jak je zmíněno při návrhu pravidel 5.5, pro mitigování slow útoků jsem použil 4 na sobě závislá pravidla. První pravidlo inicializuje proměnnou, má nastavený `threshold` tak,

[1] https://github.com/SoftEtherVPN/SoftEtherVPN/issues/1001

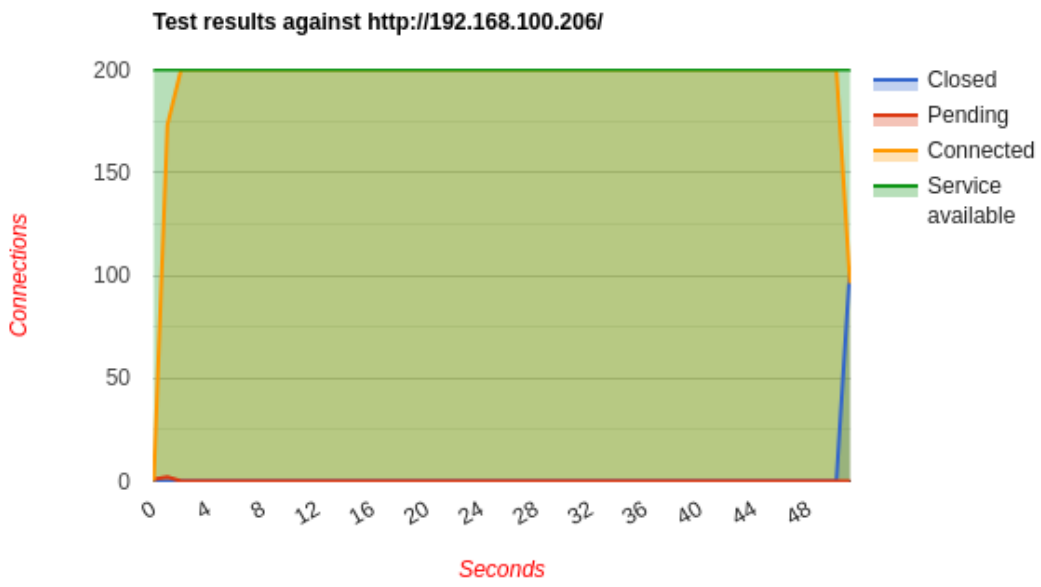
aby generoval upozornění po prvních dvou shodách za 30 sekund. Pravidlo by tedy pouze mělo reportovat potenciální začátek pomalých dat. Druhé pravidlo přičítá k proměnné +1, negeneruje žádná upozornění. Pro třetí a čtvrté pravidlo je nastaven limit generování pouze jednoho reportu za 30 sekund. Vlastnost `xbits`, která se nastavuje u třetího pravidla, má nastavenou hodnotu expirace pro IP adresu na 3 minuty.

7.3.1 Slowloris

Během testování bylo pomocí nástroje `slowhttptest` vytvořeno 200 současných připojení, kdy se posílá část dat dotazu každých 10 sekund. Počet spojení jsem zvolil tak, aby byl zvládnutelný serverem ve virtuálním prostředí. Pro testování skutečného serveru se počet připojení musí odvíjet od maximálního počtu současných připojení, které server zvládne.

Při nastavené podmínce > 4 pro třetí pravidlo očekávám po 50 sekundách simulace útoku, že dojde k ukončení spojení. Bez použití pravidel by tento útok trval stanovené dvě minuty.

Výsledky testování



Obrázek 7.1: Výsledek testování Slowloris pomocí `slowhttptest` nástroje.

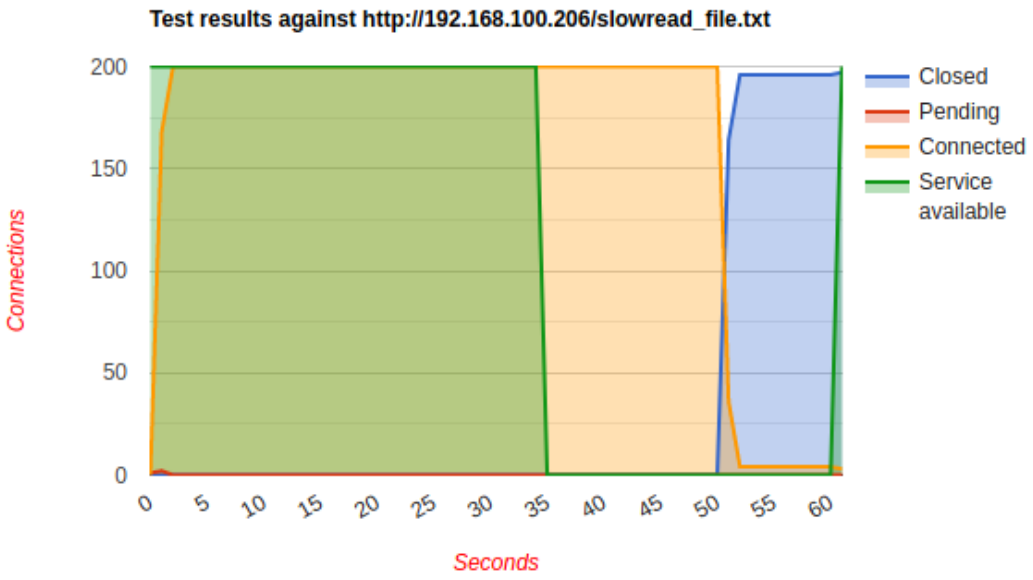
U pomalých útoků je více důležité, zda se podařilo pomalá spojení ukončit na straně serveru, a uvolnit tedy jeho zdroje. K tomu slouží graf 7.1, který po simulaci vygeneroval nástroj `slowhttptest`. Osa x znázorňuje dobu v sekundách, jak dlouho nástroj běžel, osa y zase počet připojení. Nástroj nezávisle na útoku kontroluje, zda je server stále dostupný. To je znázorněno zelenou křivkou. Žlutá křivka informuje o aktuálním počtu navázaných spojení. Modrá křivka znázorňuje řádně ukončená spojení.

Testování dopadlo podle očekávání. Po uplynutí 50 sekund byly zaslány RST pakety a nástroj `slowhttptest` hlásil, že došlo k ukončení spojení. Poté byl nástroj spuštěn znovu, čímž se úspěšně ověřila funkčnost posledního pravidla, které zabraňuje útočníkovi opět navázat spojení po dobu tří minut.

7.3.2 Slowread

Pomocí `slowhttptest` nástroje jsem vytvořil 200 současných připojení na server, které si příkazem `GET` vyžádaly 1MB textový soubor. Soubor se nahraje na server pomocí konfiguračního `set_up.sh` skriptu. Tento soubor následně útočník čte z vyrovnávacího bufferu rychlostí 256 B za 10 sekund. Zde jsem nastavil podmínku třetího pravidla na počet více jak 6 viděných TCP paketů s nulovým oknem. Tato hodnota závisí na tom, jak dlouho chceme nechat klienta/útočníka zasílat TCP pakety s nulovým oknem. V reálném nasazení by měla být daleko větší než pouze 6. Určení této hodnoty je tedy hledáním hranice mezi false positive a false negtive.

Výsledky testování



Obrázek 7.2: Výsledek testování Slowread pomocí `slowhttptest` nástroje.

Graf 7.2 vygenerovaný nástrojem obsahuje zelenou křivku značící dostupnost serveru (nástroj nezávisle na útoku kontroluje, zda je služba stále dostupná). Na grafu je vidět, že po 35 sekundách přestal být server dostupný. Při bližším zkoumání veškerého provozu jsem zjistil, že Suricata zaslala RST pakety již po 30 sekundách. Spojení ovšem zůstala otevřená dalších 20 sekund. Už ale začalo platit poslední pravidlo, které blokovalo spojení pro danou IP adresu, proto nástroj nemohl ověřit, zda je služba stále dostupná. Většina spojení se ukončila po 50. sekundě, zbylých pár zůstalo otevřených dalších 10 sekund.

Opakované pokusy potvrdily, že se nejedná o chybu, ale děje se to pokaždé. Všechna spojení se však úspěšně ukončí, a dojde tedy k potlačení daného útoku. Pokud bychom pravidla nepoužili, test by skončil až po stanovených 2 minutách.

7.4 Shrnutí

Záplavové útoky patří mezi nejoblíbenější DDoS útoky, zvláště pak SYN flood. Je tomu tak i proto, že je velice snadné tyto útoky generovat. Suricata SYN útoky nedokáže zcela potlačit, ale dokáže obstojně limitovat daný typ útoku. Samozřejmě toto vede i k zvýšení míry false positive, kdy můžeme tímto stylem limitovat i legitimní spojení. Nedojde však ale k totálnímu kolapsu serveru. Zbylé záplavové útoky lze potlačit více, respektive díky uchování stavu `flow:not_established` jsme schopni nastavit pravidlo víc striktně, aniž bychom výrazně ovlivnili legitimní provoz.

Jak se předpokládalo, amplifikační útoky dokáže Suricata potlačit nejlépe. Pokud se daný útok projevuje nebezpečným příkazem, je velice snadné jej detekovat. Jestli je to možné, tak je daleko lepší filtrovat dotazy, které ještě nebyly amplifikovány žádným serverem. Největší problém, a to nejen amplifikačních útoků, jsou fragmentované útoky. Tyto pakety Suricata nedokáže příliš efektivně filtrovat. Taktéž jsou s tím spojené nedokonalosti ve sběru statistik pomocí socketu Suricaty, které by bylo dobré prozkoumat.

Pomalé útoky Suricata překvapivě dobře zvládla. Pro potlačení těchto útoků je nejdůležitější si ujasnit, jak dlouho chceme nechat dané pomalé spojení otevřené. Může se totiž jednat buď o legitimního klienta, který má špatné spojení, nebo pokus o slow útok. Musíme tedy udělat nějaký kompromis. Suricata též umožňuje dynamicky zablokovat nebezpečnou IP adresu, čímž se potenciálně vyvarujeme novému pokusu o útok.

V mém virtuálním prostředí je těžké porovnávat jednotlivé útoky mezi sebou. Útoky generované pomocí PCAP souborů jsou daleko méně výpočetně náročné než útoky, které využívají amplifikační server. Z hlediska náročnosti na paměť je těžké odhadnout, zda amplifikační útoky budou náročnější než záplavové útoky. Pro amplifikační útoky hraje roli celková velikost dat, jakou daný útok vygeneruje. Záplavové útoky zase generují ohromné množství paketů, které musí Suricata zpracovat. Do budoucna se tak jedná o zajímavé porovnání.

Kapitola 8

Závěr

Záměrem této práce bylo analyzovat jednotlivé možnosti IDS/IPS z hlediska jejich využití pro detekci a následné potlačení nebezpečných DDoS útoků. Tyto útoky jsem rozdělil do tří skupin, které obsahují zástupce v současnosti nejznámějších DDoS útoků, ale také méně známé varianty. Skupiny se dělí na volumetrické neboli záplavové útoky, dále amplifikační a tzv. slow útoky. Jednotliví představitelé těchto útoků jsou v práci blíže představeni a jsou rozebrány různé možnosti jejich potlačení. Především se soustředím na využití Suricaty, jakožto open-source IDS/IPS nástroje pro filtrování daných DDoS útoků. K tomu bylo třeba vytvořit řadu pravidel (signatur), která Suricata používá pro detekování a potlačení síťového provozu.

Pro otestování funkčnosti Suricaty za využití jednotlivých pravidel pro DDoS útoky bylo nutné vytvořit testovací prostředí. Vytvořil jsem virtuální prostředí, které obsahuje stejné náležitosti a zařízení jako skutečné prostředí. Pro generování různých typů útoků a pro konfiguraci jednotlivých zařízení, včetně amplifikačních serverů, jsem vytvořil řadu skriptů a nástrojů. Byly navrženy s cílem, aby se veškeré toto testování dalo v budoucnu snadno přenést do reálného fyzického prostředí.

V závěru práce je samotné testování a sbírání výsledků simulovaných útoků. Mohu konstatovat, že Suricata nejlépe filtruje ty útoky, jež lze jednoznačně identifikovat. Takové útoky, které používají typ dotazu, který by se neměl vyskytovat na síti. Týká se to především amplifikačních útoků. Pro záplavové útoky Suricata spíše limituje provoz, s tím souvisí nutnost mít dobře nakonfigurovaná pravidla pro dané prostředí, aby byla míra false positive co nejmenší. Pro tyto útoky by se tak spíše hodil nějaký více specializovaný nástroj. Pomalé útoky se dají do jisté míry potlačit, je k tomu ovšem nutné vytvořit sérii vícero pravidel. Nejspíš by existovalo zajímavější řešení, které by spočívalo v monitorování doby datového toku v závislosti na velikosti přenesených dat pro daný datový tok. Takový způsob jsem u Suricaty nenašel. Taktéž fragmentované útoky jsou obecně velký problém a Suricata je v současné chvíli nedokáže příliš účinně eliminovat.

Do budoucna by bylo též zajímavé vylepšit implementovanou sadu nástrojů, aby současně s útokem generovala také odpovídající legitimní provoz. Dalo by se tak vyzkoušet, zda navržená pravidla neblokují kromě útoku i legitimní provoz, který chceme zachovat. Za zmínku taktéž stojí fakt, že navržená pravidla eliminují útoky, které vygeneruje mnou vytvořený nástroj. Nemusí proto plně pokrýt veškeré možnosti, modifikace a zvláštnosti daného útoku, které by se mohly na síti objevit. Je potřeba, aby pravidla byla v daném prostředí udržovaná a odpovídala nejnovějším hrozbám. V opačném případě pravidla postrádají účinnosti.

Také by se v budoucnu dalo zaměřit na výkonnost Suricaty z pohledu paměťové náročnosti pro amplifikační a záplavové DDoS útoky a zjistit, pro který z těchto typů je Suricata efektivnější. Popřípadě jak by se dal výkon Suricaty optimalizovat, například i použitím vlastností `bypass`, jež dovoluje přeskočit celé datové toky. Potenciál Suricaty může spočívat též v kombinaci s DDoS protectorem, který kombinuje vlastní algoritmy pro potlačení DDoS s možnostmi Suricaty. Zatímco DDoS protector by se mohl starat o flood útoky, Suricata by mohla potlačovat amplifikační a slow útoky.

Práce mi dala řadu zkušeností a nových poznatků. Přináší řadu nástrojů a pravidel, která lze využít nejen pro testování IDS/IPS zařízení Suricata v různých prostředích a systémech, ale lze je použít i pro reálné potlačení DDoS útoků.

Literatura

[1] Arteaga, J. a Mejia, W. *CLDAP Reflection DDoS* [online]. Akamai, březen 2017 [cit. 2020-11-10]. Dostupné z: https://www.akamai.com/uk/en/multimedia/documents/state-of-the-internet/cldap-threat-advisory.pdf.

[2] Chickowski, E. *Types of DDoS attacks explained* [online]. AT&T Cybersecurity, 08. července 2020 [cit. 2021-01-03]. Dostupné z: https://cybersecurity.att.com/blogs/security-essentials/types-of-ddos-attacks-explained.

[3] *Active Directory Domain Services Overview* [online]. Microsoft, 31. května 2017 [cit. 2021-01-03]. Dostupné z: https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview.

[4] Damas, J. a Graff, M. *Extension Mechanisms for DNS (EDNS(0))* [online]. IETF Tools, duben 2013 [cit. 2021-01-03]. Dostupné z: https://tools.ietf.org/html/rfc6891.

[5] *What is a DDoS Attack?* [online]. Cloudflare, 2020 [cit. 2021-01-03]. Dostupné z: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/.

[6] *Proofpoint Emerging Threats Rules* [online]. Proofpoint, 2021 [cit. 2021-04-28]. Dostupné z: https://rules.emergingthreats.net/open/suricata/rules/.

[7] Ferguson, P. a Senie, D. *Network Ingress Filtering* [online]. IETF Tools, březen 2010 [cit. 2021-01-03]. Dostupné z: https://tools.ietf.org/html/bcp38.

[8] Ganti, V. a Yoachimik, O. *Network-layer DDoS attack trends for Q2 2020* [online]. Cloudflare, 08. května 2020 [cit. 2021-01-03]. Dostupné z: https://blog.cloudflare.com/network-layer-ddos-attack-trends-for-q2-2020/.

[9] Goldschmidt, P. *TCP Reset Cookies – a heuristic method for TCPSYN Flood mitigation* [online]. Brno: Fakulta informačních technologií VUT v Brně, duben 2019 [cit. 2020-11-10]. Dostupné z: http://excel.fit.vutbr.cz/submissions/2019/057/57.pdf.

[10] *Great Firewall* [online]. Wikipedia, 2020 [cit. 2021-01-03]. Dostupné z: https://en.wikipedia.org/wiki/Great_Firewall.

[11] Hao, M. *A Look Into WS-Discovery Reflection Attacks for 2020 Q1* [online]. NSFOCUS, 5. května 2020 [cit. 2020-11-18]. Dostupné z: https://nsfocusglobal.com/a-look-into-ws-discovery-reflection-attacks-for-2020-q1/.

[12] *About Hyperscan* [online]. Hyperscan, 2020 [cit. 2021-01-04]. Dostupné z: https://www.hyperscan.io/about/.

[13] *NTP Amplification* [online]. imperva, 2020 [cit. 2020-11-14]. Dostupné z: https://www.imperva.com/learn/ddos/ntp-amplification/.

[14] *TCP SYN Flood* [online]. imperva, 2020 [cit. 2020-11-10]. Dostupné z: https://www.imperva.com/learn/ddos/syn-flood/.

[15] *What does DDoS Mean? | Distributed Denial of Service Explained | Imperva* [online]. imperva, 2020 [cit. 2020-11-09]. Dostupné z: https://www.imperva.com/learn/ddos/denial-of-service/.

[16] *UDP flood* [online]. IONOS Digital Guide, září 2020 [cit. 2020-11-13]. Dostupné z: https://www.ionos.com/digitalguide/server/security/udp-flood/.

[17] *What is IP spoofing?* [online]. Kaspersky, 2020 [cit. 2020-11-10]. Dostupné z: https://www.kaspersky.com/resource-center/threats/ip-spoofing.

[18] Jonkman, M. *About Emerging Threats* [online]. Emerging Threats, 03. března 2012 [cit. 2020-12-10]. Dostupné z: https://doc.emergingthreats.net/bin/view/Main/AboutEmergingThreats.

[19] Kupreev, O., Badovskaya, E. a Gutnikov, A. *DDoS attacks in Q2 2019* [online]. Kaspersky Securelist, srpen 2019 [cit. 2021-04-17]. Dostupné z: https://securelist.com/ddos-report-q2-2019/91934/.

[20] Kupreev, O., Badovskaya, E. a Gutnikov, A. *DDoS attacks in Q2 2020* [online]. Kaspersky Securelist, srpen 2020 [cit. 2020-11-09]. Dostupné z: https://securelist.com/ddos-attacks-in-q2-2020/98077/.

[21] *Memcached DDoS Attacks* [online]. Radware, 3. února 2018 [cit. 2020-11-18]. Dostupné z: https://security.radware.com/ddos-threats-attacks/threat-advisories-attack-reports/memcached-under-attack/.

[22] MJASOJEDOV, I. *Systém pro ochranu před DoS útoky s využitím IDS: Comparative study of Snort, Suricata and OSSEC*. Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kučera, J. Dostupné z: https://www.fit.vut.cz/study/thesis-file/23110/23110.pdf.

[23] Nagadevara, V. *Evaluation of Intrusion Detection Systems under Denial of Service Attack in virtual Environment: Comparative study of Snort, Suricata and OSSEC*. Karlskrona, SWE, 2017. Diplomová práce. Blekinge Institute of Technology, Faculty of Computing. Dostupné z: https://www.diva-portal.org/smash/get/diva2:1176622/FULLTEXT02.

[24] *Netflix to cut streaming quality in Europe for 30 days* [online]. BBC News, 19. března 2020 [cit. 2020-11-20]. Dostupné z: https://www.bbc.com/news/technology-51968302.

[25] *Slow Read DDoS Attack* [online]. NETSCOUT, 2020 [cit. 2021-04-03]. Dostupné z: https://www.netscout.com/what-is-ddos/slow-read-attacks.

[26] *UDP Flood Attacks* [online]. NETSCOUT, 2020 [cit. 2020-11-13]. Dostupné z: https://www.netscout.com/what-is-ddos/udp-flood.

[27] Newman, L. H. *GitHub Survived the Biggest DDoS Attack Ever Recorded* [online]. WIRED, 1. března 2018 [cit. 2020-11-18]. Dostupné z: https://www.wired.com/story/github-ddos-memcached/.

[28] null001. *OpenVPN service is used for UDP reflection to amplify DDoS attacks* [online]. FreeBuf, 26. září 2019 [cit. 2020-11-19]. Dostupné z: http://13.58.107.157/archives/8190.

[29] Palmer, G. *IDS/IDPS Detection Methods. Anomaly, Signature, and Stateful Protocol Analysis* [online]. Zymitri, 03. dubna 2017 [cit. 2020-12-07]. Dostupné z: https://zymitry.com/ids-idps-detection-methods/.

[30] *SYN Flood* [online]. Radware, 2020 [cit. 2021-01-03]. Dostupné z: https://security.radware.com/ddos-knowledge-center/ddospedia/syn-flood/.

[31] Rashid, F. Y. *Poorly configured DNSSEC servers at root of DDoS attacks* [online]. InfoWorld, 19. srpna 2016 [cit. 2020-11-14]. Dostupné z: https://www.infoworld.com/article/3109581/poorly-configured-dnssec-servers-at-root-of-ddos-attacks.html.

[32] Respeto, J. *New DDoS Vector Observed in the Wild: WSD attacks hitting 35/Gbps* [online]. Akamai, 18. září 2019 [cit. 2020-11-18]. Dostupné z: https://blogs.akamai.com/sitr/2019/09/new-ddos-vector-observed-in-the-wild-wsd-attacks-hitting-35gbps.html.

[33] Rouse, M. *What is penetration testing?* [online]. TechTarget, říjen 2018 [cit. 2021-01-03]. Dostupné z: https://searchsecurity.techtarget.com/definition/penetration-testing.

[34] *RST Or Fin Flood* [online]. DDoS-GUARD, 2020 [cit. 2020-11-13]. Dostupné z: https://ddos-guard.net/en/terminology/attack_type/rst-or-fin-flood.

[35] Schreiber, J. *Open Source IDS Tools: Comparing Suricata, Snort, Bro (Zeek), Linux* [online]. AT&T Cybersecurity Insights, 22. května 2020 [cit. 2020-12-08]. Dostupné z: https://cybersecurity.att.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview.

[36] *Port 11211* [online]. Shodan, 2020 [cit. 2021-01-03]. Dostupné z: https://www.shodan.io/search?query=port%3A11211.

[37] *Port 389* [online]. Shodan, 2020 [cit. 2021-01-03]. Dostupné z: https://www.shodan.io/search?query=port%3A389.

[38] *What is a low and slow attack?* [online]. Cloudflare, 2020 [cit. 2020-11-19]. Dostupné z: https://www.cloudflare.com/learning/ddos/ddos-low-and-slow-attack/.

[39] *Snort 3* [online]. Snort, 2021 [cit. 2021-04-18]. Dostupné z: https://www.snort.org/snort3.

[40] *What are Signatures and How Does Signature-Based Detection Work?* [online]. Sophos, 18. února 2018 [cit. 2020-12-07]. Dostupné z: https://home.sophos.com/en-us/security-news/2020/what-is-a-signature.aspx.

[41] *SSDP DDoS Attack* [online]. DDoS GUARD, 2020 [cit. 2020-11-19]. Dostupné z: https://ddos-guard.net/en/terminology/attack_type/ssdp-ddos-attack.

[42] *Rules Format* [online]. 2019 [cit. 2020-12-11]. Dostupné z: https://suricata.readthedocs.io/en/suricata-6.0.1/index.html.

[43] *Suricata Rules* [online]. 2011 [cit. 2020-12-11]. Dostupné z: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Rules.

[44] *Suricata (software)* [online]. Wikipedia, 2020 [cit. 2021-01-04]. Dostupné z: https://cs.wikipedia.org/wiki/Suricata_(software).

[45] *SYN Flood Attack* [online]. Cloudflare, 2020 [cit. 2020-11-10]. Dostupné z: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/.

[46] *Can I use IPTables/Traffic Control with tcpreplay?* [online]. Tcpreplay, 2021 [cit. 2021-04-26]. Dostupné z: https://tcpreplay.appneta.com/wiki/faq.html#can-i-use-iptablestraffic-control-with-tcpreplay.

[47] *How to Secure Your Network Using IDS/IPS Tools* [online]. Tek-Tools, 30. května 2020 [cit. 2020-12-07]. Dostupné z: https://www.tek-tools.com/security/best-ids-and-ips-tools.

[48] *UDP-Based Amplification Attacks* [online]. Cybersecurity & Infrastruture Security Agency CISA, 17. ledna 2014. Revidováno 18. 12. 2019 [cit. 2020-11-13]. Dostupné z: https://us-cert.cisa.gov/ncas/alerts/TA14-017A.

[49] *Virtual Machine Manager* [online]. Virtual Machine Manager, 2021 [cit. 2021-04-08]. Dostupné z: https://virt-manager.org/.

[50] Walkowski, D. *What is a DNS Amplification Attack?* [online]. F5 Labs, 26. července 2019 [cit. 2020-11-14]. Dostupné z: https://www.f5.com/labs/articles/education/what-is-a-dns-amplification-attack-.

[51] Woolf, N. *DDoS attack that disrupted internet was largest of its kind in history, experts say* [online]. The Guardian, říjen 2016 [cit. 2020-11-08]. Dostupné z: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet.

Příloha A

Použití automatizačních skriptů

V této příloze je popsáno, jak správně použít skripty napsané v Bashi. Skript `set_up.sh` nastavuje jednotlivé stroje a provádí na nich potřebnou konfiguraci, aby bylo možné následně spustit `launch_test.sh` a odsimulovat DDoS útok. K simulaci DDoS útoku se používá skript `launch_test.sh`. Oba tyto skripty používají `config.sh` pro různé proměnné a hodnoty. V tomto souboru se také nacházejí IP adresy strojů, na které se poté skripty připojují. Jednotlivé parametry DDoS útoků je možné také specifikovat v souboru `ddos_scripts/attack_spec.sh`, aby soubor odpovídal prostředí ve kterém se používá. V poslední sekci je seznam závislostí. Informace o projektu je taktéž možné nalézt na GitHubu[1].

Adresářová sktruktura

```
/
├── amp_server/
├── attacker/
│   └── attacker_steps.sh
├── config.sh
├── ddos_scripts/
│   ├── attack_spec.sh
│   ├── cldap/
│   ├── dns/
│   ├── memcache/
│   ├── ntpdos/
│   ├── openvpn/
│   └── ssdp/
├── launch_test.sh
├── LICENSE
├── README.md
├── rules/
├── set_up.sh
├── suricata/
│   └── stats_suricata.sh
└── victim/
```

[1] https://github.com/kokesak/suricata_ddos_protection

Konfigurační skript

Použití `set_up.sh`:

```
bash set_up.sh [target]
```

`target` může mít následující hodnoty: `victim, suricata, attacker, memcached, dns, ntp`. Nutno podotknout, že skript neinstaluje balíčky a nástroje, pouze je konfiguruje.

Skript pro spuštění DDoS

Použití `launch_test.sh`:

```
launch_test.sh [-a attack-vector] [-t attack-type] [-d victim-ip]
                [-l loops] [-s seconds] [-p packets]

    -a, --attack-vector: ntp, dns, openvpn, cldap, memcached-stat,
                         memcached-get, flood-syn, flood-ack,
                         flood-rst, flood-all, ssdp, slowloris,
                         slowread.

    -d, --victim-ip:     IPv4 adresa cíle útoku (oběti).

    -l, --loops:         Počet opakování pro útoky, kde se používá
                         tcpreplay (výchozí hodnota 1),
                         Platí pro útoky: OpenVPN, CLDAP, SSDP.

    -s, --time:          Počet sekund, jak dlouho útok
                         trvá (výchozí hodnota 10),
                         Platí pro útoky: DNS, Memcached-stat,
                         slowloris, slowread a všechny typy
                         flood útoků.

    -p, --packets:       Počet odeslaných paketů během
                         útoku (výchozí hodnota 10),
                         Platí pro útoky: NTP, memcached-get.

    -t, --attack-type:   Povinný argument pro amplifikační útok,
                         může být buď 'query' nebo 'response',
                         slouží ke specifikaci toho, jestli se máji
                         generovat dotazy na zneužitý server
                         nebo odpovědi z amplifikace
                         Amplifikační útoky: ntp, dns, cldap,
                         openvpn, ssdp, memcached.
                         Pro memcached-get tohle neni podporovano,
                         funguje pouze jako 'response'
```

Závislosti a knihovny

V této části je popsána většina důležitých nástrojů, knihoven a závislostí, které je nutné nainstalovat na dané stroje, aby bylo možné automatizované tooly použít. Názvy balíčků se mohou lišit podle použitého OS. Mnou použitá verze je uvedena v závorkách.

Oběť

- `ComplexHTTPServer (0.3)` – Python modul pro vytvoření HTTP serveru.
- `tmux (2.1)` – Linuxový nástroj pro tvorbu vícero oken jednoho terminálu.

Suricata

- `Suricata (6.0.2)` – balíček se samotnou Suricatou.
- `suricatasc (6.0.2)` – klient pro připojení k socektu Suricaty.

Amplifikační server

1. NTP
   - `ntp (4.2.6p2)` – balíček pro NTP server, verze musí být <= 4.2.7p26.
2. Memcached
   - `memcached (1.4.25)` – balíček pro Memcached server.
3. DNS
   - `bind9 (9.10.3.dfsg.P4-8ubuntu1.19)` – balíček pro vytvoření DNS služby na serveru.
   - `dnsutils (9.10.3.dfsg.P4-8ubuntu1.19)` – balíček pro testování správného fungovaní DNS.

Útočník

1. NTP
   - `bittwist-linux (2.0)` – Linuxový nástroj pro generování paketů.
   - `perl 5 (v5.32.0)` – programovací jazyk Perl, nutný ke spuštění NTP DoS skriptu.
   - `Socket.pm (2.031)` – modul pro Perl umožnující tvorbu socketů.
2. Memcached-STATS, DNS – tento útok používá následující knihovny jazyka C:
   - `time.h`
   - `pthread.h`
   - `unistd.h`
   - `stdio.h`
   - `stdlib.h`
   - `string.h`
   - `sys/socket.h`
   - `netinet/ip.h`
   - `netinet/udp.h`
   - `arpa/inet.h`

   Pro verzi Memcached útoku, kdy se zasílá dotaz `get` je nutné mít:
   - `nping (0.7.91)` – Linuxový nástroj pro generování paketů.
   - `memcache (0.2.0)` – modul pro Python.
3. CLDAP, OpenVPN, SSDP
   - `tcpreplay (4.3.3)` – sada nástrojů pro editovaní a přehrávaní zachycené komunikace (např. PCAP soubory).
4. Volumetrické útoky SYN flood, ACK, RST flood a UDP flood
   - `hping3 (3.0.0-alpha-2)` – nástroj pro generování a analýzu paketů.
5. Pomalé útoky Slowloris a Slowread
   - `slowhttptest (1.8.2)` – nástroj pro simulaci DoS útoků, které nepotřebují velkou šířku pásma (tzv. slow útoky).

Příloha B

Skripty pro generování útoků

Zde je seznam skriptů, které jsem použil pro generování útoků nebo pro tvorbu PCAP souborů. Jedná se o seznam skriptů, jejichž autorem nejsem já, jedná se o skripty volně dostupné na internetu. Jejich použití je omezeno pouze ke vzdělávacím účelům.

- NTP - skript napsaný v jazyce Perl, dostupný zde:
  https://www.exploit-db.com/exploits/37562.
- DNS - skript napsaný v jazyce C, dostupný zde:
  https://github.com/MrScytheLULZ/DDoS-Scripts/blob/master/dns.c.c.
- Memcached - skript napsaný v jazyce C, dostupný zde:
  https://github.com/responsibleD/memcached-PoC.
- CLDAP - skript napsaný v jazyce C použit pro generování PCAP souboru, dostupný zde:
  https://github.com/Phenomite/AMP-Research.
- SSDP - skript napsaný v jazyce C použit pro generování PCAP souboru, dostupný zde:
  https://github.com/MrScytheLULZ/DDoS-Scripts/blob/master/ssdp.c.c.