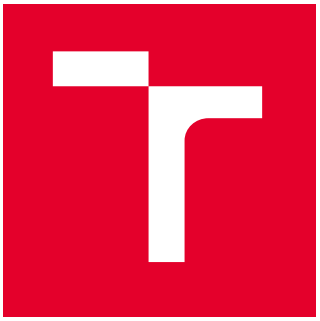


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

VIRTUALIZOVANÉ PROSTŘEDÍ PRO TESTOVÁNÍ DDOS ÚTOKŮ

VIRTUALIZED ENVIRONMENT FOR TESTING THE DDOS ATTACKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Lukáš Král**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Michael Jurek**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Lukáš Král ***ID:*** 230601

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

NÁZEV TÉMATU:

Virtualizované prostředí pro testování DDoS útoků

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem práce je návrh a vývoj ovladatelného testovacího prostředí, které umožní simulaci DDoS útoků. Součástí bude webová aplikace umožňující ovládání botnetu a vizualizaci vhodných metrik (vytížení botů, sítě, serveru). Tato aplikace bude umožňovat jak generování botů a webového serveru (v podobě docker kontejnerů), tak i samotnou simulaci vybraných DDoS útoků s možností jejich konfigurace (pomocí formulářových prvků). V teoretické části se zaměřte na možnosti tvorby virtualizovaného kontejnerizovaného prostředí, jeho monitoring, automatické ovládání a následnou konfiugraci a exekuci DoS útoků. V praktické části vytvořte testovací prostředí a webovou aplikaci, která bude umožňovat tvorbu a ovládání jednotlivých kontejnerů a vizualizaci metrik (pomocí aplikace Grafana). Dále proveďte testování vytvořené aplikace pomocí vybraných DDoS útoků.

DOPORUČENÁ LITERATURA:

[1] SIMON, Marek a Ladislav HURAJ. DDoS testbed based on peer-to-peer grid. 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES) [online]. IEEE, 2016, 2016, 1181-1186 [cit. 2022-09-09]. ISBN 978-1-5090-4620-1. Dostupné z: doi:10.1109/SCOPES.2016.7955627.,
[2] HURAJ, Ladislav a Marek SIMON. Realtime attack environment for DDoS experimentation. 2019 IEEE 15th International Scientific Conference on Informatics [online]. IEEE, 2019, 2019, 000113-000118 [cit. 2022-09-09]. ISBN 978-1-7281-3180-1. Dostupné z: doi:10.1109/Informatics47936.2019.9119271.

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

Vedoucí práce: Ing. Michael Jurek

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

ABSTRAKT

Bakalářská práce se zabývá tvorbou virtualizovaného kontejnerizovaného prostředí pro testování útoků na dostupnost služby. V rámci teoretické části práce jsou rozebrány útoky typu DDoS, dále pak téma virtualizace a platforma Docker. V praktické části práce je navrhnuta a implementována webová aplikace umožňující testování vybraných DDoS útoků. Dopad útoků je následně vizualizován pomocí vhodných metrik prostřednictvím monitorovacího systému v podobě nástrojů Prometheus a Grafana. Útočníci, oběť i komponenty monitoringu jsou generováni pomocí Docker kontejnerů.

KLÍČOVÁ SLOVA

DDoS, Docker, Flask, Grafana, monitoring, Prometheus, virtualizace, vizualizace

ABSTRACT

The bachelor thesis focuses on the creation of a virtualized containerized environment for testing denial of service attacks. Theoretical part of the thesis analyses DDoS attacks, virtualization and Docker platform. In the practical part of the thesis, a web application is designed and implemented to test selected DDoS attacks. The impact of the attacks is then visualized using appropriate metrics through a monitoring system consisting of Prometheus and Grafana tools. Attackers, victim and monitoring components are generated using Docker containers.

KEYWORDS

DDoS, Docker, Flask, Grafana, monitoring, Prometheus, virtualization, visualization

KRÁL, Lukáš. *Virtualizované prostředí pro testování DDoS útoků*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 72 s. Bakalářská práce. Vedoucí práce: Ing. Michael Jurek

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Lukáš Král
VUT ID autora:	230601
Typ práce:	Bakalářská práce
Akademický rok:	2022/2023
Téma závěrečné práce:	Virtualizované prostředí pro testování DDoS útoků

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Michaelu Jurkovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Tato bakalářská práce se zabývá tvorbou virtualizovaného kontejnerizovaného prostředí pro testování útoků na dostupnost služby. Součástí práce je rozebrání problematiky těchto útoků, techniky virtualizace a tvorba kontejnerizovaného prostředí, jeho ovládání a monitoring. Práce si klade za cíl vytvořit webovou aplikaci schopnou provádět více druhů útoků na dostupnost služby (anglicky *Distributed Denial of Service* – DDoS) pomocí kontejnerů představující útočníky, kteří útočí na kontejner webového serveru. Výsledná aplikace by měla umožňovat generování libovolného počtu konfigurovatelných botů a jedné oběti v podobě zvoleného webového serveru. Následně by měla být umožněna konfigurace a exekuce vybraných útoků na dostupnost služby. Nedílnou součástí práce je nasazení monitorovacího systému, v němž bude uživatel schopen pomocí vhodné vizualizace sledovat dopad útoků na jeho jednotlivé účastníky.

Celá práce je rozdělena do sedmi kapitol. V první z nich je základně popsána problematika distribuovaných útoků na dostupnost služby včetně rozdělení těchto útoků do kategorií a uvedení příkladů útoků pro jednotlivé kategorie. Druhá kapitola objasňuje virtualizaci platforem, v třetí kapitole je poté popsána platforma jednoho druhu z těchto virtualizací *Docker*, která je posléze využívána v praktické části práce. Obsahem čvrté kapitoly je vytvoření návrhu aplikace splňující vytyčené cíle, jsou zde představeny jednotlivé funkcionality aplikace a popsáno očekávané chování aplikace. Pátá kapitola se zabývá výběrem vhodného monitorovacího systému pro potřeby této práce. Jsou zde popsány a srovnány tři monitorovací nástroje a následně je vyhodnocen ten nejvhodnější. Kapitola šestá se zabývá realizací tohoto návrhu, rozděluje implementaci do dvou etap, v rámci kterých je popsán proces tvorby celého nástroje a fungování dílčích funkcionalit. V poslední kapitole je provedeno testování a demonstrace vytvořené aplikace.

1 Problematika DDoS útoků

Pro uvedení do problematiky útoků DDoS (Distributed Denial of Service) je nejprve vhodné objasnit útok DoS (Denial of Service) (česky *odepření služby*). Při útoku DoS utočník využívá jediné zařízení ke zneužití zranitelnosti softwaru nebo k zahlcení cíle falešnými požadavky ve snaze vyčerpat výpočetní zdroje oběti. Útok DDoS se snaží o znedostupnění služby z mnoha připojených zařízení rozmístěných po internetu společně tvořící takzvaný „botnet“. Součástí botnetu můžou být mobilní telefony, počítače nebo prakticky jakékoliv IoT zařízení infikovaná malwerem, který dovoluje útočníkovi jejich vzdálené ovládání [1]. Specálně IoT zařízení poskytují vhodný terč pro napadení, většina z nich má totiž obvykle mizivé nebo žádné zabezpeční. Nejčastější díry v zabezpečení zahrnují slabá či defaultní hesla, neaktualizovaný firmware nebo slabé zabezpečení sítě, ve které se zařízení nachází [2].

Na rozdíl od jiných druhů kybernetických útoků se útoky DoS a DDoS nesnaží o narušení bezpečnostního perimetru[1], nýbrž o narušení normálního provozu serverů a webů s cílem je znedostupnit legitimním uživatelům. Odepření služby může také sloužit pouze jako odpoutání pozornosti od jiné škodlivé aktivity.

Motivace k útoku bývá různá, může se jednat o způsob jak poškodit konkurenci v podnikání, způsob jak někoho vydírat nebo může jít o pouhou „zábavu“. Úspěšný útok může vést ke ztráte příjmů, podkopat důvěru spotřebitelů a celkově poškodit pověst oběti. Útoky jsou také prováděny jako nástroj takzvaného „hackativismu“, kdy se daná skupina aktivistů snaží vyjádřit nesouhlas s aktuálním děním. Státem sponzorované útoky jsou využívány k umlčení opozice nebo jako prostředek k narušení kritické infrastruktury nepřátelského státu. [1]

Projevy přetížení webové služby

Z pohledu uživatele snažícího se o legitimní spojení s některým webovým serverem, který je pod útokem DoS či DDoS, bude cílová služba projevovat známky zahlcení. Základní projevy přetížení webového serveru [4], které mohou, ovšem nemusí znamenat útok, zahrnují:

- zobrazení HTTP chybových kódů v rozsahu 500 až 599 – značí problémy na straně serveru,
- zpoždění vyřízení požadavků,
- obnovení nebo odmítnutí TCP (Transmission Control Protocol) připojení před doručením obsahu,
- server doručuje pouze částečný obsah.

[1] Bezpečnostním perimetrem se rozumí hranice mezi jednou a druhou sítí. Vytvoření takové hranice lze definovat jako umístění nezbytných ochranných opatření k zabránění nepovoleného přístupu [3].

1.1 Typy komunikace botnetů

Jak již bylo naznačeno výše, slovo botnet označuje síť zařízení infikovaných malwerem umožňující neoprávněné vzdálené ovládaní. Konkrétní zařízení se nazývá „bot“, ale je možné se také setkat s označením „zombie“, obdobně pak „zombie army“ k botnetu [5].

Ačkoliv oblastí zájmu této práce jsou útoky DoS, potažmo DDoS, je důležité zmínit, že se nejedná o jedinou škodlivou aktivitu, ke které se využívá botnet. Útočník může botnet využít například k různým formám spamu, kdy rozesílá malware nebo phishing[2]. Dále se může jednat o monitorování aktivity uživatelů kompromitovaných zařízení za účelem krádeže citlivých dat . [7]

Jednotliví boti dostávají po převzetí kontroly od útočníka příkazy skrze takzvaný C&C (Command–and–Control) server. V závislosti na konkrétní architektuře botnetu se funkce botů a C&C serveru můžou, ale také nemusí nacházet na stejném zařízení. Níže jsou popsány dva základní typy komunikace botnetů, centralizovaná architektura klient–server a decentralizovaná peer–to–peer.

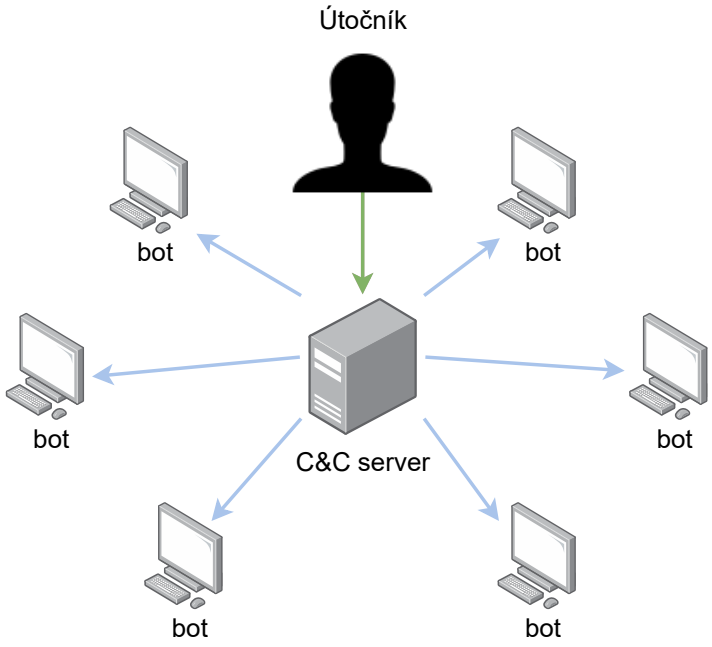
1.1.1 Klient–server

Základním a nejjednodušším spojením botnetu je centralizovaná topologie klient–server. Klientem je v tomto případě bot, který dostává příkazy od přímo připojeného C&C serveru (viz obr. 1.1). Jednoduchá struktura této topologie zajišťuje nízkou latenci a vysokou škálovatelnost. Naproti tomu hlavní nevýhodou je právě centralizace, kdy stačí vyřadit C&C server k ochromení botnetu. Větší botnety samozřejmě disponují více než jedním C&C serverem, stále se však jedná o velkou slabinu celého botnetu. K zabránění odhalení C&C serveru ze síťového provozu bota se používá algoritmus generování domén (Domain Generation Algorithm – DGA). Server na základě měnícího se vstupu za pomocí DGA generuje v pravidelný časový interval jiné doménové jméno, ze kterého přicházejí instrukce. Vstupem je většinou aktuální čas, je tedy důležité, aby měl celý botnet správně synchronizovaný systémový čas.

K řízení botnetu můžou být využity například snadno implementované protokoly HTTP (Hypertext Transfer Protocol) nebo IRC (Internet Relay Chat). HTTP byl navržen pro doručování webových stránek ze serverů, jedná se o protokol typu požadavek–odpověď, takže není možná skupinová komunikace. Hlavní výhodou použití protokolu HTTP je to, že je ve velkém používán na internetu, tudíž komunikace botnetu prostřednictvím tohoto protokolu splývá s běžným provozem. Protokol IRC

[2]Phishing – kybernetický trestný čin, při kterém je oběť kontaktovaná ve snaze přesvědčit ji o legitimnosti protějšku a následné získání jejích citlivých údajů [6].

byl původně vyvinut pro aplikace určené k internetové komunikaci. Jedná se o textový protokol umožňující klientům komunikovat se serverem, který je zodpovědný za předávání zpráv ostatním klientům a serverům. Protokol podporuje přenos souborů a komunikační kanály mohou být chráněny heslem, což umožňuje zabránit převzetí kontroly nad botnetem. Nevýhodou protokolu IRC je, že může být snadno blokován firewallem[3] a není běžný v podnikových sítích, což u HTTP neplatí. [8]



Obr. 1.1: Centralizovaná architektura botnetu

1.1.2 Peer-to-peer

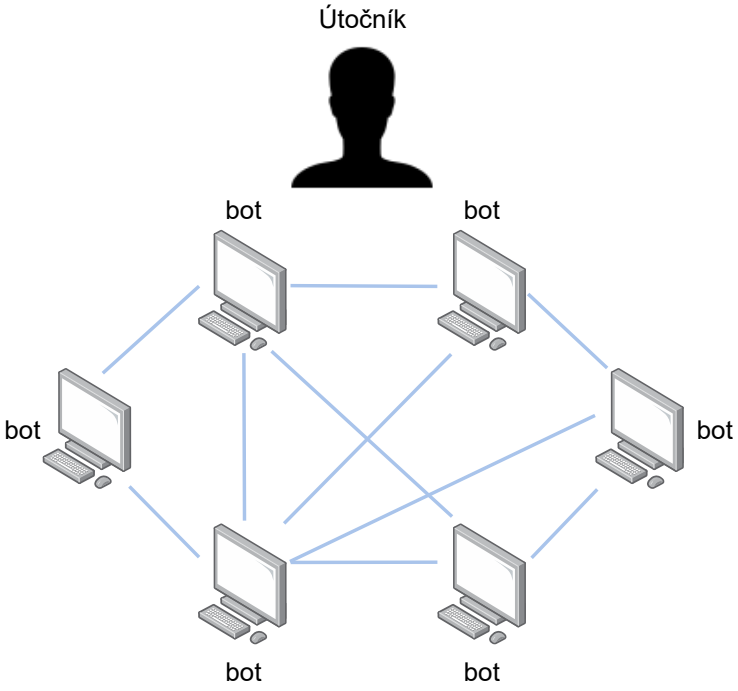
Novější botnety využívají decentralizovanou architekturu peer–to–peer (P2P), postrádající centralizovaný bod v podobě C&C serveru. Tato architektura se tedy skládá pouze z botů (viz obr. 1.2), přičemž každý bot potenciálně zastává roli C&C serveru. Tuto roli jsou schopni zastat pouze boti (označováni také jako *peer*) přijímající příchozí komunikaci [11]. Jedná se o zařízení, která nejsou schována za firewallem nebo proxy serverem[4], ani nevyužívají metodu překladu adres neboli NAT[5]. Zatímco boti schopni pouze odchozí komunikace zastávají roli klienta.

[3]Firewall – síťový systém, fungující jako filtr, který určuje pravidla komunikace v rámci sítě.

[4]Proxy server zastává funkci prostředníka mezi klientem a serverem. Požadavek klienta nejprve putuje na proxy server, který jej poté přepošle na cíl a sám se přitom vydává za klienta.

[5]NAT (Network Address Translation – Překlad síťových adres) je technika, kdy se množství soukromých IP adres „schová“ za jednu veřejnou, skrze kterou komunikují se zbytkem internetu.

Každé zařízení v botnetu je propojeno alespoň s jedním dalším botem, avšak čím více botů je vzájemně propojeno, tím lépe. Ideální plná propojenost botnetu by zajišťovala velice nízkou komunikační latenci a vysokou robustnost botnetu, neboť vyřazení jakéhokoliv počtu botů by nenarušilo schopnost komunikace zbytku. V realitě ovšem není většina botnetů plně propojena z důvodu obtížné implementace, omezení počtu spojení použitého protokolu a zvyšující se viditelnosti botnetu s roustoucím počtem spojení. Obtížná implementace je způsobena především problémem s nalezením peerů a spolehlivou distribucí příkazů. K distribuci příkazů mohou být využity P2P protokoly původně vyvinuté pro sdílení souborů či spolupráci zařízení jako je například protokol *WASTE*[6] nebo protokoly založené na hashovací tabulce *Kademlia*[7]. Jedním z možných řešení, jak naleznout peery, je pak náhodné prohledávání internetu, což ovšem vede ke snadnějšímu odhalení aktivity botnetu. Dalším způsobem je vložení seznamu známých peerů do spustitelného souboru bota, to však nese riziko v podobě odhalení peerů v případě zachycení bota obránci. Zmíněné slabiny musí útočník při formování útoku vzít v potaz a omezit je vhodnou implementací. [8]



Obr. 1.2: Princip peer–to–peer komunikace botnetu

[6]WASTE je protokol pro sdílení souborů nebo zpráv, komunikace je šifrována pomocí RSA. Nevýhodou je malý počet možných uzlů (okolo 50), pro větší botnety tudíž není nejvhodnější [9].

[7]Distribuovaná hashovací tabulka Kademlia je systém zajišťující automatickou organizaci zařízení do strukturované sítě a jejich komunikaci [10].

1.2 Typy útoků na dostupnost služby

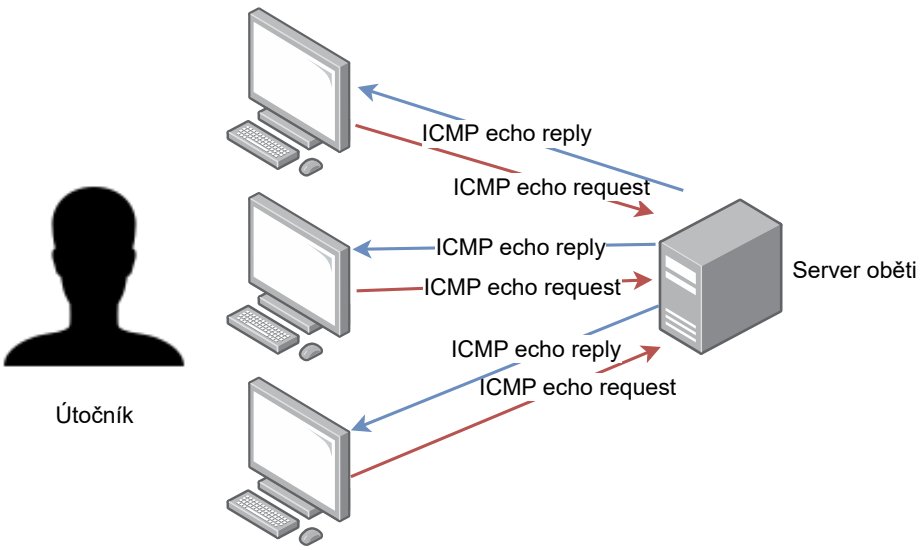
Tato práce rozděluje útoky do tří základních kategorií, a to útoky volumetrické, protokolové a útoky na aplikační vrstvě.

1.2.1 Volumetrické útoky

Volumetrické útoky se snaží o zpomalení nebo úplné zastavení cílové služby. Útočník vysílá na cíl velké množství paketů ve snaze zahltit dostupnou šířku pásma oběti. Velikost útoku se standardně měří v bitech za sekundu, obvykle se jedná o útoky v řádech desítek až stovek *Gb/s*, ačkoliv se objevují i útoky přesahující 1 *Tb/s*. Volumetrické útoky jsou rozšířeny díky relativní jednoduchosti škálování útoku, přičemž bývá obtížné zmírnit útoky přicházející z mnoha zdrojů. [12, 13]

ICMP Flood

ICMP (Internet Control Message Protocol) je protokol sloužící k odesílání informací o službách a zpráv jako jsou echo požadavek, echo odpověď, časová známka nebo chybových zpráv. Echo požadavek a odpověď tvoří dohromady funkcionalitu ping, která slouží k ověření funkčnosti spojení mezi dvěma zařízeními. Zdrojové zařízení pošle na cíl echo požadavek a očekává odpověď. Cílem útoku ICMP (Ping) Flood je tedy zahltit cílové zařízení echo požadavky v takové míře, že kvůli posílaní obrovského počtu echo odpovědí vyčerpá své dostupné zdroje, a tím bude narušena jeho běžná síťová činnost. [14]



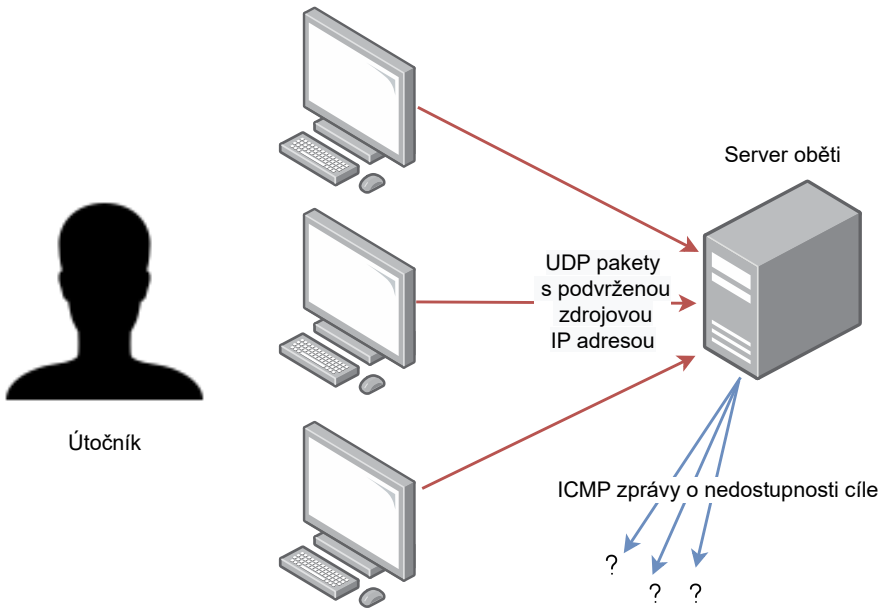
Obr. 1.3: Zahlcení oběti ICMP echo požadavky

UDP Flood

Protokol UDP (User Datagram Protocol) je na rozdíl od protokolu TCP (Transmission Control Protocol) bezestatový, což znamená, že nenavazuje obousměrnou komunikaci, pouze odesílá datové pakety, aniž by čekal na odpověď. Tato vlastnost UDP protokolu je dobrou příležitostí k jeho využití na provedení DDoS útoku.

Server naslouchá na svých portech a po přijetí UDP požadavku zkontroluje, zdali se na daném portu nachází nějaká služba. V případě, že k portu žádná služba přidělena není, odpoví server paketem `ICMP HOST UNREACHABLE`.

Při UDP Flood útoku posílá útočník na různé porty serveru obrovské množství paketů v co nejmenším možném časovém intervalu ve snaze vyčerpat dostupné zdroje serveru kvůli jeho snaze o identifikaci služeb na daných portech. Server však posílá zpět pakety o nedosažitelnosti služby, takže používání vlastní IP adresy jednotlivých botů způsobuje postupné vyčerpávání vlastních zdrojů. Při útocích o menším množství útočících zařízení je tedy vhodné podvrhnout zdrojovou IP adresu (viz 1.4), při velkých útocích už však toto nehraje příliš velkou roli. [15]



Obr. 1.4: Zahlcení oběti pakety s podvrženou zdrojovou IP adresou

ICMP Fragmentation Flood

ICMP Fragmentation Flood je typ ICMP Flood útoku, kdy útočník namísto kompletních ICMP paketů posílá pouze podvržené fragmenty paketů. Cílové zařízení se poté snaží o rekonstrukci paketů do validní formy, přičemž postupně vyčerpá své dostupné výpočetní zdroje. [12]

DNS Amplification

Amplifikující (zesilovací) útoky patří do skupiny DDoS útoků snažící se o vyčerpání dostupné šířky pásma pomocí velkého objemu dat na cílový server. Principem těchto útoků je generování větší odpovědi na relativně malý požadavek a tuto odpověď směřovat na server oběti [16]. Jedním takovým útokem je útok zvaný *DNS Amplification*. Tento útok využívá fungování DNS (Domain Name System) protokolu k zahlcení cílového serveru. Protokol DNS je používán k překladu lidsky čitelného doménového jména na strojově srozumitelné IP adresy. Pokud uživatel zadá doménu do webového prohlížeče (např. *www.vut.cz*) a prohlížeč nemá příslušnou IP adresu ve své mezipaměti, je odeslán dotaz na rekurzivní DNS server, který v případě neznalosti překladu adresy posílá dotaz dále v architektuře DNS serverů[8] až do získání korespondujícího překladu. Při tomto útoku zasílá útočník velké množství DNS dotazů s podvrženou zdrojovou IP adresou (tedy adresou oběti) na rekurzivní DNS server, dotazujících se na všechny záznamy pro určitou doménu. V případě mnoha záznamů spadajících pod jednu doménu tak může být odpověď DNS serveru mnohem větší než zaslaný dotaz, což zesiluje dopad útoku. Server oběti tak přijímá velké množství dat, jimiž je postupně přetížen. [17]

NTP Amplification

Útok NTP Amplification využívá, obdobně jako u útoku DNS Amplification, fungování síťového protokolu NTP (Network Time Protocol) k zahlcení cílového serveru násobně větším síťovým provozem než je vysílán. NTP protokol slouží k distribuci přesných časových informací do zařízení připojených k síti. Napříč internetem existuje mnoho veřejných NTP serverů, používaných legitimními klientskými systémy k synchronizaci času. Jedním z možných požadavků, který může být NTP serverem zpracován, je takzvaný *MONLIST* příkaz. Tento příkaz vrací v odpovědi seznam až 600 posledních IP adres, které byly připojeny k NTP serveru. Tato odpověď je proto obvykle datově mnohonásobně objemnější než samotný požadavek, což z NTP serveru dělá ideální zdroj pro zesilující útoky DDoS. [18]

[8]DNS architektura se skládá z root, top–level doménových a autoritativních DNS serverů, kdy dotaz putuje postupně touto strukturou až ke konečnému vyřešení autoritativním serverem, který odešle odpověď zpět rekurzivnímu serveru.

1.2.2 Útoky na aplikační vrstvě

Velký počet útoků DDoS, kterými jsou například již popsaný ICMP nebo UDP Flood, se zaměřuje na šířku pásma sítě kvůli jejímu relativně snadnému vyčerpání. Při těchto útocích zasílá utočník na server velké množství síťových paketů, čímž efektivně vyčerpá šířku pásma. K tomuto účelu se používají protokoly síťové vrstvy jako UDP a ICMP. V dnešní době jsou však servery více odolnější při identifikaci těchto útoků a také mají mnohdy dostupnou velkou šířku pásma, čímž se znedostupnění serverů pomocí těchto útoků stalo obtížnějším. Naproti tomu útoky na aplikační vrstvě nemají za cíl vyčerpat šířku pásma sítě, místo toho se pokoušejí vyčerpat zdroje serveru, jako je procesor, databáze, paměť nebo soketová spojení. Dalším rozdílem oproti jiným útokům na dostupnost služby, je jejich legitimní projev. Mezi útočníkovým a legitimním požadavkem není prakticky žádný rozdíl, liší se pouze v záměru. Tyto útoky jsou také vysoce cílené, snaží se vyčerpat konkrétní zdroj jako například procesor, paměť nebo soketová spojení, zatímco zbylé zdroje nejsou dotčeny. Vyčerpání jednoho ze zdrojů má poté za následek nefunkčnost celého systému. [19]

Slowloris attack

Název útoku je odvozen od nástroje *Slowloris*, který se využívá pro provedení útoku na dostupnost služby. Vyznačuje se svou jednoduchostí a nutností použítat jen minimální šířku pásma, ačkoliv prakticky ovlivňuje pouze web cílového serveru bez větších efektů na ostatní služby či porty. Slowloris se ukázal jako vysoce účinný proti mnoha populárním typům softwaru webových serverů, včetně Apache 1.x a 2.x.

Slowloris funguje na principu otevření několika spojení k cílovému webovému serveru a poté udržení těchto spojení po co nejdelší dobu. Útočník tohoto dosahuje nepřetržitým odesíláním dílčích HTTP požadavků, kdy žádný není dokončen. Napadený server tedy otevírá další a další spojení, přičemž postupně vyčerpává maximální počet možných spojení a následně jsou legitimní pokusy o připojení zamítnuty.

Odesíláním částečných, avšak na první pohled legitimních paketů, bývá na rozdíl od paketů se škodlivým obsahem obtížné detekovat nežádoucí aktivitu na serveru. Útok slowloris je tedy pomalejší než můžou být ostatní útoky na dostupnost služby, vzhledem k jeho projevu podobajícímu se běžnému provozu ovšem může být velmi účinným. [20]

Slow Read

Při útoku Slow Read vytvoří útočící zařízení legitimní spojení s cílovým serverem. Po navázání spojení začne každé zařízení požadovat po serveru soubor. Jakmile je zahájeno stahování souboru, začne hostitel útočníka zpomalovat čtení přijatých

paketů. Stav čtení poté přetrvává, čímž je udrženo spojení po dlouho dobu. V případě dostatečného množství útočníků jsou tak zabrány všechny dostupné prostředky oběti, a tím je dosaženo znemožnění navázání dalších, legitimních spojení. Útoky Slow Read nepoužívají podvrhnuté IP adresy, naopak pro udržení dlouhodobého spojení je potřeba využít adresu reálnou. Tyto útoky mohou být velmi účinné, neboť požadovaný soubor nemusí být nikterak velký, operuje se zde pouze s rychlostí čtení kvůli co nejdelšímu udržení spojení, tudíž detekce takového útoku může být poměrně obtížná. [21]

HTTP Flood

Útok HTTP Flood zneužívá protokol aplikační vrstvy HTTP k útoku na server. Při tomto útoku utočník při komunikaci se serverem nebo konkrétní aplikací manipuluje s požadavky HTTP GET a HTTP POST. Pro provedení tohoto útoku je nutné navázat TCP spojení s platnou zdrojovu IP adresou. Útočník tedy naváže spojení s cílovým serverem bez podvržení zdrojové IP adresy jednotlivých botů a odešle požadavek HTTP GET za účelem stažení velmi velkého souboru. V reakci na takový požadavek oběť provede řadu akcí. Nejprve musí načíst soubor ze svého uložiště a uložit jej do své operační paměti. Poté soubor podle potřeby rozdělí na více paketů a odešle je na dané zařízení. Odpověď na tento požadavek tedy zahrnuje využití paměti i výpočetního výkonu oběti. Velké množství požadavků tohoto typu zahltí veškeré dostupné zdroje oběti a znemožní jí odpovídat na legitimní požadavky. [16]

1.2.3 Protokolové útoky

Protokolové DDoS útoky využívají slabin internetových komunikačních protokolů. Vzhledem k tomu, že mnoho těchto protokolů se používá globálně, bývá změna jejich funkčnosti komplikovaná a pomalá. Mimo jiné složitost většiny protokolů způsobuje, že po opravě již známé slabiny se objeví slabina nová, využitelná k jiné formě útoku. [22]

Ping smrti

Ping smrti (anglicky *Ping of Death*) je typ útoku, kdy dochází k destabilizaci zařízení či služby tím, že přijme nadměrně velké datové pakety. Správný paket internetového protokolu verze 4 (IPv4) je tvořen 65 535 bajty a většina starších zařízení nedokáže zpracovat větší pakety. Odeslání většího paketu porušuje standard internetového protokolu, takže útočníci posílají pakety ve fragmentech, které při pokusu o opětovné složení cílovým serverem mohou způsobit jeho pád. Jedná se o poměrně zastaralý útok, objevený v první polovině devadesátých let, od roku 1998 je většina moderních

zařízení proti těmto typům útoku chráněna, mnoho serverů je navíc nastavena na výchozí blokaci ICMP ping zpráv. Jak ukazují události z roku 2013 a 2020, i přes zastaralost tohoto útoku je stále relevantní brát tento útok v potaz. V roce 2013 byl zaznamenán návrat pingu smrti kvůli chybné implementaci ICMP protokolu IPv6 v rámci některých operačních systémů Windows, jenž útočník mohl využít ke generování velmi velkých ICMPv6 paketů, které byly schopny způsobit pád cílového zařízení. V roce 2020 pak byla objevena chyba v ovladači jádra opět v systému Windows, kdy útočník při správně zvoleném zaslání ICMP ping zpráv mohl způsobit pád a následný restart cílového počítače. Zneužití této zranitelnosti bylo ovšem vcelku obtížné, nicméně lze vidět, že tento typ zranitelností se stále může objevit. [23]

TCP SYN Flood

Útok TCP SYN Flood je typ útoku DDoS zneužívající běžného TCP třícestného potřesení rukou (anglicky *three–way handshake*) ke spotřebování dostupných zdrojů oběti. Klasické třícestné potřesení rukou vypadá následovně:

- klient požádá o připojení odesláním zprávy SYN (synchronize) serveru,
- server potvrdí spojení odesláním SYN-ACK (synchronize–acknowledge) zprávy zpět klientovi,
- klient odpoví zprávou ACK (acknowledge) a spojení je navázáno.

TCP SYN flood útok spočívá v opakovaném odesílání SYN paketů na všechny porty cílového serveru, často s použitím falešné IP adresy. Server, nevědomý útoku, přijímá zdánlivě legitimní požadavky na navázání spojení a odpovídá každému pokusu SYN-ACK paketem. Útočník poté buď neodesílá očekávané ACK, nebo, v případě že podvrhl zdrojovou adresu, vůbec SYN-ACK paket neobdrží. V každém případě bude server pod útokem čekat na potvrzení svého SYN-ACK paketu. Takto zatížený server je poté zpomalen nebo úplně odstaven pro legitimní uživatele. [16]

2 Virtualizace platforem

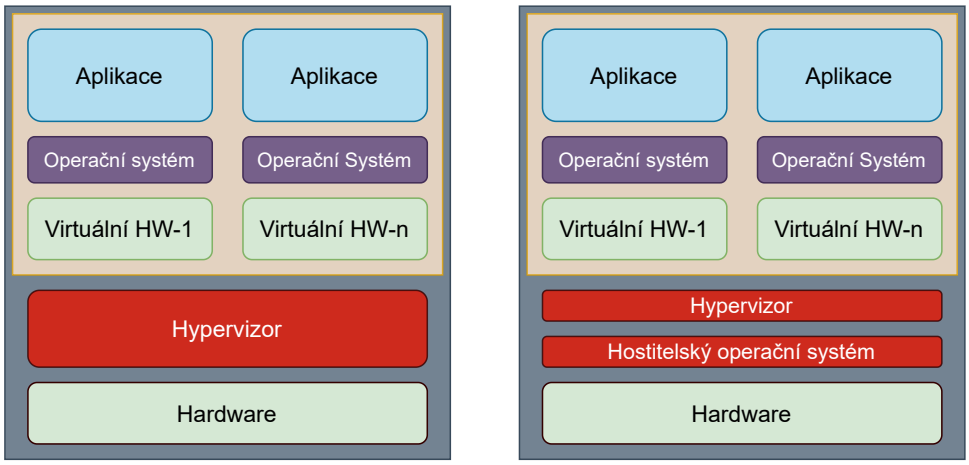
Virtualizace je v informatice technika rozdělení dostupných zdrojů zařízení do jiného uspořádání než fyzicky existují, a tím umožňující uživateli využívat tyto zdroje způsobem, kterým by bez možnosti virtualizace nebyl schopen. Virtualizaci [24] je možno rozdělit na dva typy. Prvním typem je virtualizace zdrojů. Do té kategorie spadá například síťová virtualizace nebo virtualizace uložiště. Druhým typem je virtualizace platforem, která je zodpovědná za virtualizaci procesů a aplikací.

Objektem zájmu této bakalářské práce je primárně typ druhý, tedy virtualizace platformem. Platformu lze obecně definovat jako hardwarovou nebo softwarovou architekturu sloužící jako základní struktura, na níž jsou vyvíjeny a provozovány aplikace, procesy a technologie umožňující dosažení požadovaných výsledků [25].

Níže jsou popsány dva základní přístupy pro virtualizaci platforem.

2.1 Využití Hypervizoru

Hypervizor je software umožňující komunikaci virtuálního stroje s fyzickým zařízením. Hypervizory se dělí na dva typy. První typ je nainstalován přímo na hardware zařízení, což vede k přímé komunikaci virtuálních strojů s hardwarem. Naproti tomu druhý typ hypervizoru běží nad operačním systémem, který je na daném zařízení nainstalován. [24]



Obr. 2.1: První a druhý typ Hypervizoru

2.1.1 Paravirtualizace

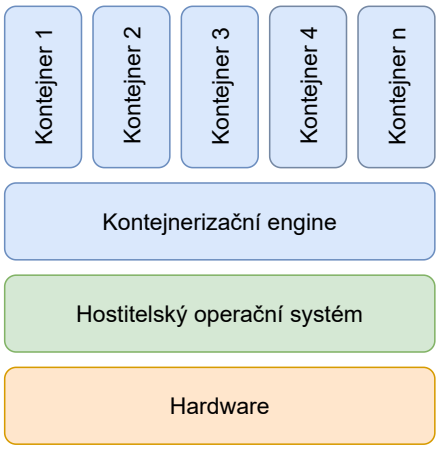
Paravirtualizace využívá první typ hypervizoru, který je nainstalován přímo na hardwaru. Virtualizované systémy jsou si vědomy vzájemné přítomnosti ostatních systémů a pracují jako celek. Tento typ virtualizace je díky přímější komunikaci skrze příkazy hypervizoru, zvané *Hypercalls*, rychlejší než plná virtualizace, je však zapotřebí upravit jádro jednotlivých operačních systémů, díky čemuž jsou jednotlivé virtuální stroje hůře přenosné a ne vždy kompatibilní. [24, 26]

2.1.2 Plná virtualizace

Plná virtualizace je založena na druhém typu hypervizoru. V tomto případě je hypervizor nainstalován v hostovském operačním systému. Virtuální stroje jsou plně izolovány jeden od druhého, fungují nezávisle každý s vlastním operačním systémem a konfigurací. Plná virtualizace také nevyžaduje úpravu jednotlivých operačních systémů, což poskytuje lepší portabilitu. [24, 26]

2.2 Virtualizace na úrovni operačního systému

Virtualizace na úrovni operačního systému (také zvaná *kontejnerizace*) je druh virtualizace, při níž se virtualizuje pouze potřebný kus softwaru společně se všemi závislostmi nezbytnými k jeho bezproblémovému chodu. Výsledná virtualizovaná jednotka se nazývá kontejner. Kontejnery umožňují vývojářům vytvářet aplikace bez ohledu na prostředí, ve které bude následně používána, navíc díky tomu, že obsahují pouze nezbytné součásti, zabírají oproti virtualizaci pomocí hypervizoru podstatně méně uložiště. [27]



Obr. 2.2: Virtualizace na úrovni operačního systému

3 Platforma Docker

Docker je platforma umožňující virtualizaci na úrovni operačního systému vydána v roce 2013. Ačkoliv technologie kontejnerů byla známá již před vydáním Dockeru, nebyl tento druh virtualizace zdaleka tak populární jako s touto platformou. V dnešní době je Docker stále hojně využíván pro většinu kontejnerových projektů díky svým možnostem, jednoduchosti, kompabilitě a velké komunitě [28]. Docker je pro osobní užití, výukové účely a menší komerční užití zdarma. Podporuje operační systémy Linux, macOS i Windows [29].

V následujícím textu je rozebrána architektura této platformy, její dílčí části a možnosti práce s nimi. Na závěr jsou zmíněny aktuální dostupné alternativy pro Docker a jejich konkurenceschopnost. Vzhledem k obsahu praktické části práce a také k určité přehlednosti popisu zacházení s Dockerem je tento popis omezen pouze na Linuxovou verzi této platformy. Obsah této kapitoly, pokud není uvedeno jinak, je čerpán především z oficiální dokumentace Dockeru viz [29], kde je také možno nalézt detailnější informace o zácházení s níže popsanou platformou.

3.1 Architektura Dockeru

Docker používá architekturu klient-server. Klient Docker komunikuje s démonem[1], který vytváří a spravuje docker objekty jako jsou image, kontejnery, sítě a svazky. Klient a démon mohou běžet na stejném systému nebo může být klient připojen ke vzdálenému démonu. Architektura Dockeru (viz obrázek 3.1) se skládá z tří základních komponent – Docker klient, Docker hostitel a registr.

Docker klient

Klient je hlavní komponentou pro interakci uživatele s Dockerem. Docker klient komunikuje s démonem a poskytuje příkazový řádek, skrze který uživatel zadává příkazy pro ovládání platformy. Docker klient je schopen komunikace s více než jedním démonem.

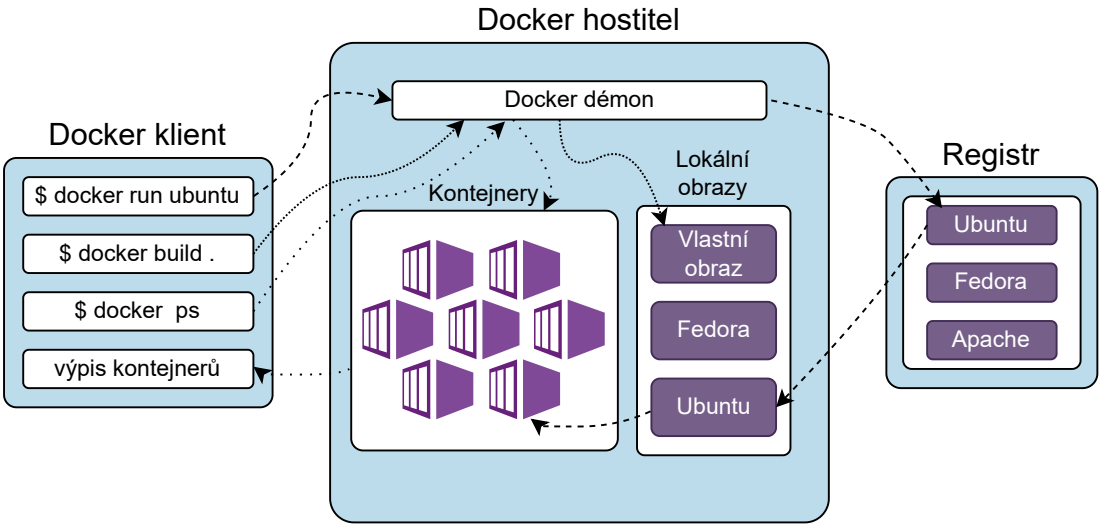
Docker hostitel

Docker hostitel poskytuje potřebné prostředí pro spouštění a provoz kontejnerů. Skládá se z Docker démona, Docker objektů a uložiště.

[1]Démon (anglicky *daemon*) je program bežící na pozadí bez přímé interakce s uživatelem, dohlíží na správný chod systému a poskytuje určité funkce. Název démon programu obvykle končí písmenem *d* (například *dockerd*).

Registr

Registr ukládá Docker obrazy. Docker Hub je veřejný registr obrazů, který je ve výchozím stavu využíván Dockerem. Je však možné vytvořit registr vlastní, případně využít registr z neoficiálních zdrojů.



Obr. 3.1: Architektura Dockeru

3.2 Docker démon

Z poznatků již zmíněných by se dalo označit Docker démona za správce celého Dockeru. Klient s démonem spolu komunikují pomocí rozhraní REST API[2]. Ve výchozím nastavení poslouchá Docker démon skrze unix soket vytvořený v adresáři `/var/run/docker.sock` a jeho vlastníkem je root, ostatní uživatelé k němu mohou přistupovat pouze pomocí sudo nebo je nutné vytvořit skupinu s názvem *docker* a vybrané uživatele do ní vložit. V některých Linux distribucích je tato skupina automaticky vytvořena při instalaci Dockeru. Vytvoření skupiny a přidání uživatele do ní je možno udělat následující posloupností příkazů.

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

[2]REST (Representational State Transfer) označuje architekturu rozhraní splňující určitá kritéria. API (Application Programming Interaface) je rozhraní umožňující dvěma aplikacím vzájemně komunikovat. REST API (také označované jako RESTful API) je pak rozhraní splňující omezení architektury REST.

Mimo unix socket může Docker démon naslouchat požadavkům skrze síťový TCP soket nebo Deskriptor souboru (anglicky *file descriptor*). Pokud se klient potřebuje připojit k démonu vzdáleně, je nutné nejprve TCP soket povolit. Je však důležité brát v potaz, že základní nastavení nevyžaduje autentizaci a poskytuje nešifrovaný přístup k Docker démonu. Ten by měl být zabezpečen pomocí šifrovaného soketu HTTPS (Hypertext Transfer Protocol Secure) nebo umístěním zabezpečené proxy před něj. Více o nastavení soketů a další možnosti konfigurace démonu lze dohledat v již zmíněné oficiální Docker dokumentaci [29].

3.3 Docker objekty

3.3.1 Obraz

Obraz (častěji nazýván anglicky *image*) označuje v Dockeru šablonu pouze pro čtení obsahující sadu pokynů pro vytvoření kontejneru. Obrazy často bývají založeny na jiném obrazu s určitým vylepšením.

Obraz je také možno vytvořit pomocí takzvaného *Dockerfile* souboru, což je textový soubor obsahující všechny příkazy potřebné k jeho sestavení. Každý pokyn v tomto souboru vytvoří nad obrazem vrstvu. Pokud tedy uživatel stahuje z registru například obraz založený na Linuxové distribuci Ubuntu, který již má ve svém lokálním uložišti, stáhnou se z registru pouze chybějící vrstvy. Tato vlastnost Dockeru je jedním z důvodů, proč se jedná o tak rychlou a ve smyslu uložiště lehkou virtualizační technologii.

Tvorba Dockerfile souboru

Obrazy jednotlivých Linux distribucích jsou velmi osekané, kupříkladu obraz Ubuntu neobsahuje ani nástroj *ping*[3]. Tvorbou jednoduchého souboru (viz výpis 3.1), jež se dle konvence nazývá *Dockerfile*, lze ovšem vytvořit obraz Ubuntu, jehož pozdějším použitím lze ušetřit čas instalováním tohoto nástroje pro každý kontejner zvlášť.

Výpis 3.1: Příklad jednoduchého *Dockerfile* souboru.

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y iputils-ping
```

Příkazem `FROM` začíná každý soubor pro tvorbu obrazu, určuje, na jakém obrazu bude ten tvořený stavět. V tomto případě se jedná o nejnovější obraz Ubuntu. Většina Dockerfile souborů se vytváří z již existujícího obrazu, pokud chce však tvůrce obrazu

[3]Ping je jednoduchý program, sloužící k otestování spojení mezi dvěmi síťovými zařízeními.

mít plnou kontrolu nad jeho obsahem, má možnost vytvořit takzvaný základní obraz, v tom případě by místo `ubuntu:latest` bylo `scratch`. Příkaz `RUN` je krok stavění obrazu, po jeho vykonání se k obrazu přidá vrstva s tímto stavem. Ve výpisu 3.1 je tedy vytvořena nová vrstva, ve které se aktualizují balíčky a následně nainstaluje utilita *ping*.

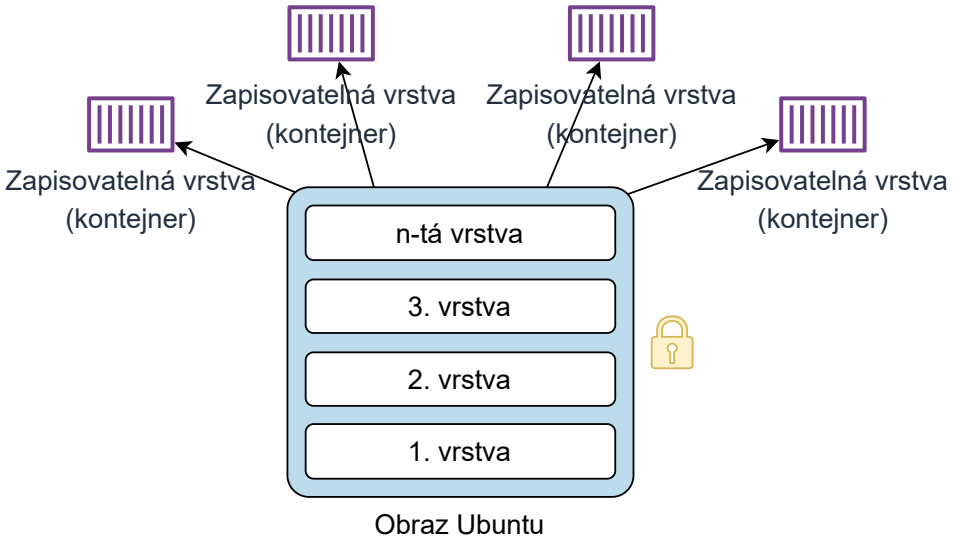
Vytvoření obrazu z Dockerfile souboru lze pak pomocí příkazu.

```
$ docker build -t ubuntu_ping .
```

Příznak `-t` u příkazu `docker build` nastavuje jméno tvořeného obrazu, v tomto případě *ubuntu_ping*. Tečka na konci příkazu pak značí, že Dockerfile se nachází v aktuálním adresáři.

3.3.2 Kontejner

V předchozí části byl popsán obraz jako šablona pro vytváření kontejnerů, kontejner je tedy instance obrazu. Kontejner může být vytvořen, spuštěn, zastaven, vymazán nebo přesunut. Obraz se skládá z vrstev určených pouze pro čtení. Kontejner je pak poslední vrstvou obrazu, která už je však zapisovatelná. Všechny změny provedené v běžícím kontejneru jako je zapisování nových dat, modifikace nebo smazání již existujících dat, se ukládají do této zapisovatelné vrstvy. Jakmile je kontejner smazán, je smazána i tato vrstva. Každý vytvořený kontejner představuje separátní vrstvu, takže nic nebrání vytvoření mnoha kontejnerů, které přistupují ke stejnému obrazu, každý však s odlišným stavem dat. Na obrázku 3.2 je znázorněno množství kontejnerů, sdílející stejný obraz Ubuntu.



Obr. 3.2: Více kontejnerů sdílejících jeden obraz Ubuntu

Ve výchozím nastavení je kontejner poměrně dobře izolován od ostatních kontejnerů a svého hostitelského počítače. Docker nicméně umožňuje změnit izolaci kontejneru a jeho subsystémů, kterými jsou například síť nebo uložiště.

Spuštění kontejneru

Základním příkazem pro práci s Dockerem by se dal označit příkaz `docker run`, kterým se vytvoří a spustí nový kontejner. Následující příkaz spustí kontejner z obrazu Ubuntu, interaktivně se připojí k aktuální relaci příkazového řádku, přidělí mu jméno ubuntu_container a spustí /bin/bash.

```
$ docker run -it --name ubuntu_kontejner ubuntu /bin/bash
```

Jakmile je tento příkaz odeslán, Docker démon zajistí následující:

1. Pokud není obraz Ubuntu již přítomen v lokálním uložišti, bude stáhnut z definovaného registru (výchozím registrem je Docker Hub). Manuálně lze tento krok provést příkazem `docker pull ubuntu`.
2. Vytvoří se nový kontejner, stejným způsobem by jej uživatel Dockeru vytvořil příkazem `docker create ubuntu`[4].
3. Docker přidělí kontejneru souborový systém a přidá k obrazu zapisovatelnou vrstvu obsahující vytvořený kontejner.
4. Vzhledem k nespecifikování sítě v zadaném příkazu bude vytvořeno síťové rozhraní pro připojení kontejneru k výchozí síti zvané bridge a bude mu přiřazena IP adresa (více o sítích viz následující část kapitoly 3.3.3).
5. Následně bude kontejner spuštěn a zahájí se proces definovaný v příkazu, v tomto případ se jedná o `/bin/bash`.

Po zadání výše vypsaných kroků může uživatel zadávat vstupy skrze terminál, ke kterému je interaktivně připojen vytvořený kontejner. K interaktivnímu připojení došlo díky příznaku `-it`, za kterým stojí dva oddělené příznaky `-i` a `-t`. Přiznak `-i`, celým názvem `--interactive`, zajišťuje reakci kontejneru na uživatelův vstup. Přiznakem `-t` (`--tty`) je kontejneru přidělena relace terminálu.

V příkazu `docker run` bylo také definováno jméno kontejneru. Docker při vytváření kontejnerů přiděluje automaticky každému z nich takzvané *docker id* sloužící k identifikaci kontejneru. Definování jména kontejneru poskytuje další možnost jak pracovat s kontejnerem a především lepší přehlednost pro správce kontejnerů. Praktičnost lze předvést například na příkazu `docker inspect` sloužícímu k vypsání detailního popisu kontejneru ve formátu JSON. Bez definování názvu kontejneru by příkaz vypadal následovně.

[4]Zde je třeba upozornit na funkci příkazu `docker create`, který kontejner pouze vytvoří a o spuštění se nestará, jako je tomu v případě `docker run`. Manuální spuštění již existujícího kontejneru zajišťuje příkaz `docker start`.

```
$ docker inspect c9df564b66e5
```

Nevýhodou práce s id je nutnost jej vyhledat ve výpisu kontejnerů nebo se k němu dobrat jiným způsobem, zapamatovatelnost tohoto formátu je velice obtížná. Pokud je ovšem definován název kontejneru, nejlépe podle nějaké předem definované konvence, je práce s ním o mnoho přijemnější.

```
$ docker inspect ubuntu_kontejner
```

3.3.3 Síť

Při vytváření kontejneru existuje možnost připojit kontejner do předem definované sítě pomocí parametru `--network`. Při instalaci Dockeru jsou automaticky vytvořeny tři sítě: *none, host* a *bridge*. Výpis existujících sítí lze zobrazit příkazem `docker network ls`. Výchozí sítí je poslední z vyjmenovaných (tedy bridge) a pokud při tvoření kontejneru není definováno jinak, je kontejner automaticky připojen do této sítě. Bridge síť zajišťuje izolaci kontejnerů, nacházejících se na stejném hostitelském zařízení, díky čemuž jsou kontejnery navzájem schopny komunikovat, pouze pokud se nacházejí ve stejné síti. Při práci s Dockerem je však, než využívání výchozí bridge síťe, vhodnější vytvořit vlastní bridge síť, která je té výchozí nadřazena a přináší několik výhod. Uživatelem definované bridge síťě poskytují lepší izolaci, protože používání pouze výchozí sítě vede k povolené komunikaci mezi kontejnery, které jinak spojené nejsou, a tudíž by správně neměli sdílet stejnou síť. Dále je možné odpojit za běhu kontejner od této sítě a připojit jej k jiné, kdežto kontejner ve výchozí síti je nejprve nutné zastavit a poté jej znovu vytvořit s jinou nadefinovanou sítí. Uživatelsky definované sítě také automaticky poskytují rozlišování kontejnerů pomocí DNS[5] (Domain Name System), takže kontejnery mohou mezi sebou komunikovat pomocí doménového aliasu namísto IP adresy. Výpis sítí, jež Docker umožňuje vytvořit, je zobrazen v tabulce 3.1.

Vytvoření bridge sítě, následné vytvoření kontejneru a zobrazení detailu sítě by bylo možné provedením posloupností příkazů zobrazených níže.

```
$ docker network create -d bridge bridge_sit
$ docker create -itd --name ubuntu_kontejner \
> --network bridge_sit ubuntu /bin/bash
$ docker network inspect bridge_sit
```

[5]DNS je síťový prokol zajišující překlad z IP adresy na doménové jméno a opačně.

První z této série příkazů vytvoří síť s názvem *bridge_sit*, přičemž přepínač `-d` slouží k definování typu vytvářené sítě[6]. Dále je vytvořen kontejner *ubuntu_kontejner* na základě obrazu ubuntu a připojen do sítě *bridge_sit*. Posledním příkazem je pak vypsán detail této sítě ve formátu JSON.

Při vytváření kontejneru je použit přiznak `-td`, `-t` již byl vysvětlen dříve (viz 3.3.2). Příznak `-d` (`--detached`) rozběhne kontejner na pozadí, tudíž se v příkazovém řádku po vytvoření kontejneru zobrazí pouze jeho id. Rozběhnout kontejner na pozadí je vhodné, pokud tvůrce nechce zadávat vstupy do kontejneru hned po jeho vytvoření. Vyvolání příkazového řádku kontejneru by pak bylo možné následujícím příkazem.

```
$ docker exec -it ubuntu_kontejner /bin/bash
```

Tab. 3.1: Druhy Docker sítí

Host	Síť odstraňující síťovou izolaci mezi kontejnerem a hostitelem Dockeru, tzn. kontejner se stává součástí hostitelovy sítě.
Bridge	Výchozí síť určená pro kontejnery komunikující se stejným Docker démonem.
Overlay	Tento typ sítě dokáže propojit více Docker démonů, a tím umožnit spojení kontejnerů, které se nacházejí na jiných zařízení.
IPvlan	Poskytuje uživateli plnou kontrolu nad IPv4 i IPv6 adresací.
Macvlan	Umožňuje přiřazení kontejneru MAC (Media Access Control) adresu[7], díky čemuž se pak tváří v síti jako fyzické zařízení.
None	Definicí sítě „none“ je zakázáno veškeré síťování.

3.4 Docker Compose

Vytváření objektů s mnoha parametry může vést k relativní nepřehlednosti jejich tvorby. Mnohdy je takových objektů k dosažení požadovaného cíle potřeba více a vypisování jednotlivých příkazů do příkazového řádku se stává velice neefektivní. K usnadnění tohoto procesu slouží program *Docker Compose*, díky kterému lze pomocí konfiguračního YAML[8] souboru přehledně nastavit jednotlivé objekty a zadáním příkazu `docker-compose up` v adresáři, kde se tento soubor nachází spustit všechny naráz.

[6] V případě tvorby bridge sítě není třeba používat přepínač `-d` pro její definování, v příkladu je pouze pro demonstraci.

[7] MAC adresa je jednoznačný síťový identifikátor, někdy zvaný *fyzická adresa*. Je přidělena síťové kartě při výrobě zařízení.

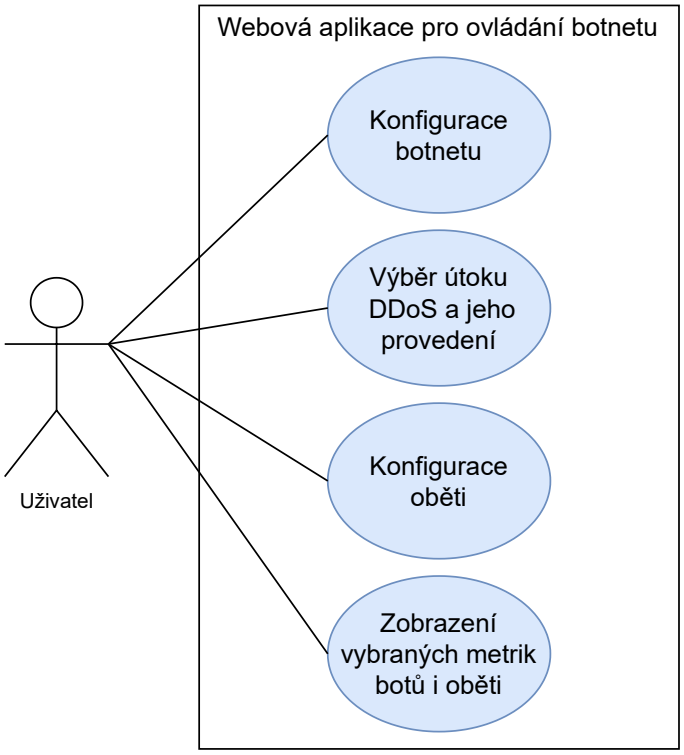
[8] YAML (YAML Ain't Markup Language) je jazyk pro serializaci strukturovaných dat, populární díky své poměrně dobré čitelnosti pro člověka.

4 Návrh webové aplikace

Obsahem této kapitoly je návrh budoucí webové aplikace. Návrh je rozdělen do dílčích částí, přičemž výsledkem by měla být kompletní představa o podobě aplikace, jejíž implementací se zabývá kapitola 6. Proces návrhu a definice funkcionalit budoucí aplikace je důležitou součástí při vývoji programů. Tento proces, mnohdy opomíjený, vede k lepší vizualizaci řešeného problému, a tím usnadňuje a urychluje následný proces implementace.

4.1 Diagram případů užití

V první fázi vývoje programu je pro lepší vizualizaci vhodné vytvořit diagram případu užití. Diagram případu užití se používá k určení očekávaného chování vyvíjeného programu z pohledu koncového uživatele. Většinou se jedná o velmi jednouchý diagram zobrazující stěžejní možnosti užití budoucího programu. Jak lze zpozorovat na obrázku 4.1, jeho vytvoření neslouží ke specifikaci řešení daného problému, nýbrž pouze k vizualizaci chování, jež od budoucího systému očekáváme [34].



Obr. 4.1: Diagram případů užití

4.2 Funkcionalita aplikace

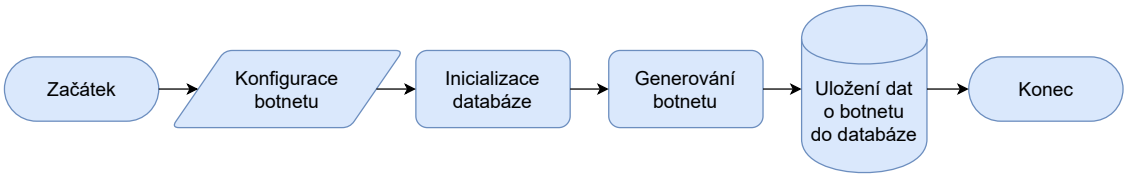
Po vytvoření diagramu případů užití je vhodné rozšířit vizualizaci možností uživatele do praktického seznamu. Tento seznam slouží k určení průchodu aplikací opět z pohledu uživatele, přičemž je však detailněji objasněno chování vyvíjeného systému. Průchod lze shrnout takto:

1. Jakmile je aplikace spuštěna, automaticky se vygeneruje server oběti a zobrazí se domovská stránka.
2. Na této stránce je uživatel schopen nakonfigurovat a následně vygenerovat botnet.
3. Uživateli se poté zpřístupní možnost výběru typu útoku DDoS, jeho konfigurace a následné provedení.
4. Na domovské stránce se nachází tlačítko, které uživatele přesměruje do prostředí, kde jsou monitorovány vybrané metriky botnetu i serveru.
5. Uživatel je kdykoliv schopen změnit konfiguraci botů, serveru i útoků.

Z doposud popsaných funkcionalit programu je možné aplikaci rozdělit do tří logických celků, a to: generování botnetu, konfigurace útoku DDoS a monitoring výpočetních zdrojů kontejnerů.

4.2.1 Generování botnetu

První aktivita uživatele po spuštění aplikace před umožněním provedení útoku je specifikace botnetu a jeho následné generování. Boti i automaticky vygenerovaný server, na který bude prováděn útok, jsou generováni pomocí Docker kontejnerů. Uživatel je schopen nastavit počet botů a vybrané specifikace kontejnerů. V momentě, kdy je uživatel spokojen s volbou specifikace botnetu, jej tlačítkem vygeneruje. Proces generování zahrnuje vytvoření již zmíněných Docker kontejnerů s nakonfigurovanými specifikacemi, přičemž potřebná data k další práci v aplikaci jsou uložena do vytvořené databáze. Vývojový diagram algoritmu generování botnetu je uveden na obrázku 6.2.

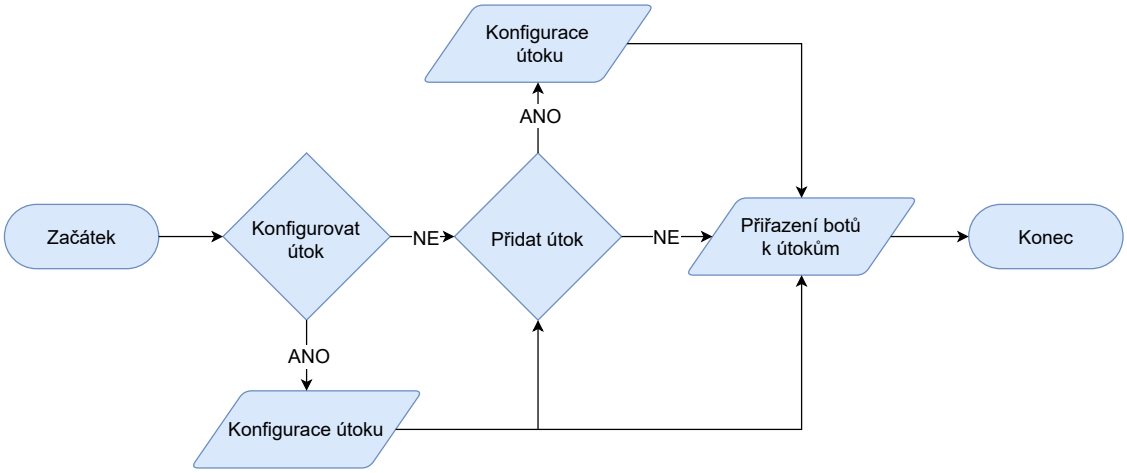


Obr. 4.2: Vývojový diagram algoritmu pro generování botnetu

4.2.2 Konfigurace útoku

Po vygenerování botnetu se uživateli zpřístupní možnost konfigurace útoku. Na výběr je z více druhů útoků na dostupnost služby, pričemž může celý botnet provádět shodný útok, nebo se útok může skládat z více typů útoků.

Ve výchozím stavu je vybrán jeden útok, ke kterému jsou připojeni všichni dostupní boti. Uživatel je schopen v tomto bodě útok spustit s přednastavenými parametry nebo tuto konfiguraci změnit. Dále má možnost přidat další z dostupných útoků a ty taktéž konfigurovat, přičemž možnosti konfigurace záleží na konkrétním typu útoku. Uživatel má možnost přesouvat své dostupné zdroje v podobě botů mezi těmito útoky, či některé boty vůbec do útoku nezahrnout.



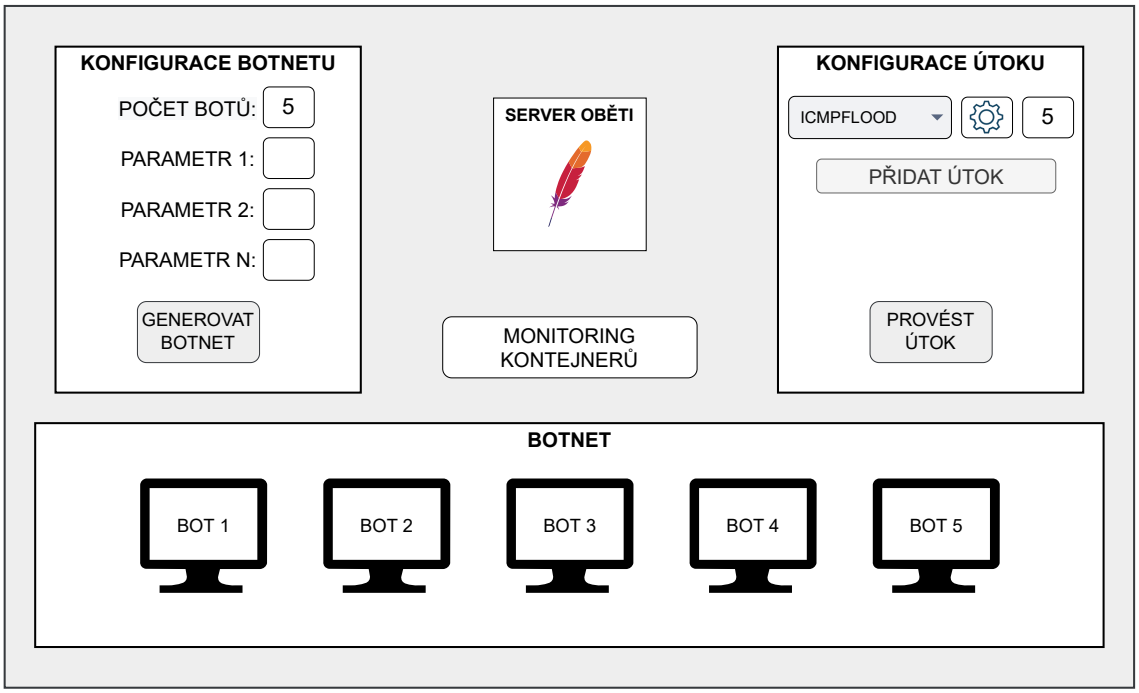
Obr. 4.3: Vývojový diagram algoritmu pro konfiguraci útoku

4.2.3 Monitoring výpočetních zdrojů kontejnerů

Důležitým bodem v testování DDoS útoků je monitoring dopadu prováděných útoků jak na oběť, tak i na útočníka. Samotná aplikace obsahuje možnost zobrazení detailu útočníků i oběti, nástroj ovšem umožňuje vizualizaci vhodných metrik kontejnerů v reálném čase pomocí monitorovacího systému vybraného v následující kapitole (viz 5). Tento systém obsahuje přehledný dashboard zobrazující dopad útoků pomocí vybraných metrik. Uživatel je schopen dostat se na stránku obsahující tento dashboard pomocí tlačítko přítomného na domovské stránce aplikace.

4.3 Vizuální podoba aplikace

V tomto bodě již bylo definováno vše potřebné ke kompletní představě o tom, jak má vypadat vizuální podoba stránky vyvíjené aplikace. Na obrázku 4.4 je tato idea přenesena do návrhu, přičemž je třeba brát v potaz, že proces implementace jednotlivých částí aplikace může vést k určitému odchýlení od této prvotní představy.



Obr. 4.4: Návrh vizuální podoby domovské stránky aplikace

Pro kompletní představu o grafickém zobrazení této stránky je návrh uveden v bodě, kde je botnet již vygenerován. Botnet se zde skládá z pěti botů, kteří jsou graficky znázorněni na spodní části stránky. Na levé vrchní straně je dostupná sekce s konfigurací botnetu, na té protější pak konfigurace útoku. Vybrán je útok ICMP Flood a přiřazeno mu je všech 5 botů, vedle tohoto počtu se nachází možnost útok konfigurovat a pod vybraným útokem volba pro přidání dalšího typu útoku. Mezi nastavovacími okny je znázorněn server oběti[1] a přesně uprostřed stránky se nachází tlačítko pro přesměrování na stránku s monitoringem kontejnerů.

[1]Oficiální logo Apache serveru je pero, proto je v návrhu znázorněn server oběti právě tímto obrázkem.

5 Monitorovací systém Docker kontejnerů

Při práci s Docker kontejnery je mnohdy potřebné monitorovat jejich metriky, jakými jsou například vytížení procesoru, operační paměti nebo síťový provoz. Nástrojů, pomocí kterých lze tohoto dosáhnout, existuje v dnešní době mnoho. Výběr řešení závisí na mnoha faktorech, které je třeba brát v potaz. Některé nástroje nabízí širokou škálu možností práce s nimi, nicméně mohou představovat neúměrně velkou náročnost implementace vzhledem k nízkým požadavkům na monitoring. Na druhou stranu výběr příliš jednoduchého řešení může vést k nenaplnění potřeb osob, jež potřebují metriky vhodným způsobem monitorovat a k nárůstu času strávéného nad implementací monitorovacího systému.

Rozvaha nad výběrem správného monitorovacího řešení je proto nedílnou součástí efektivní implementace zájmu této práce. Tato kapitola obsahuje představení třech systémů pro monitoring Docker kontejnerů a následný výběr nejvhodnějšího řešení.

5.1 Dynatrace

Dynatrace [30] je platforma pro správu a monitoring digitálních služeb, včetně kontejnerového prostředí. Dynatrace využívá umělou inteligenci pro diagnostiku problémů v reálném čase. Díky strojovému učení a dalším technikám umělé inteligence dokáže tato platforma nejen lépe odhalovat problémy v infrastruktuře, ale taktéž poskytnout pravděpodobnou příčinu problému a jeho možná řešení. Monitoring pomocí této platformy je realizován pomocí takzvaného *Dynatrace OneAgent* agenta. Tento agent je po nainstalování schopen získat relevantní informace o aplikacích běžících na monitorovaném zařízení, jako jsou metriky, logy, chování uživatelů, výkonnost a jiné. Toto se, především díky technologiím umělé inteligence, děje bez toho, aniž by bylo nutné manuálně nastavovat specifické nastavení pro každou aplikaci zvlášť. Tento proces automatizuje a zjednodušuje monitorování a diagnostiku aplikací.

Dynatrace nabízí řešení pro monitorování Docker kontejnerů, pričemž existují dvě možnosti jak kontejnery monitorovat. Jedním ze způsobů je nainstalování agenta na hostitelský systém Dockeru. Nainstalováním agenta na systém hostující Docker je umožněno monitorovat základní metriky jako je vytížení procesoru, paměti nebo síťového procesu. Druhým způsobem je nainstalovat agenty do jednotlivých kontejnerů, což umožní monitorovat všechny úrovně metrik a vlastností kontejneru včetně interních chyb a výkonostních problémů. Tento přístup umožňuje získat podrobnější informace o stavu aplikace a kontejneru a umožňuje efektivnější diagnostiku problémů v aplikaci. Dynatrace je navíc velice uživatelsky přívětivý a po nainstalování agenta na hostitelský systém je možné jej nastavit tak, aby při každém novém

vytvořeném kontejneru automaticky zavedl podřízeného agenta do každého takového kontejneru, a tím umožnil monitorování detailnějších metrik.

Celkově se řešení monitorování Docker kontejnerů pomocí Dynatrace dá považovat za velice profesionální a je určeno především pro větší podniky zabývající se výkonností a dostupností svých aplikací a infrastruktury. Dynatrace nabízí širokou škálu funkcí a nástrojů pro monitorování výkonnosti aplikací, analýzu dat a řízení incidentů, které usnadňují řešení komplexních problémů. Nevýhodou Dynatrace je především nutnost platby za použití této platformy, což v případě malých projektů či použití v menších podnicích může být neúměrné k požadovaným schopnostem monitoringu.

Vzhledem k potřebám implementované aplikace v rámci této práce toto řešení poskytuje mnohem více než je potřeba, rozhodně by tak mohla být zvolena právě tato platforma. Na druhou stranu nepřítomnost i kupříkladu velice osekané verze platformy zdarma a existence jiných, volně dostupných řešení, které sice neposkytují tak komplexní nástroje jako právě Dynatrace, avšak pro potřeby vyvíjené aplikace plně dostačujících, zařazuje tuto platformu spíše níže v žebříčku výběru.

5.2 Zabbix

Zabbix [31] je open-source[1] platforma pro monitorování sítí, serverů a dalších komponent IT. Monitorování pomocí Zabbixu funguje na základě agentů a serveru, který zpracovává data z agentů. Zabbix agenti jsou nainstalováni na zařízeních, jejichž aplikace, výpočetní zdroje apod. je potřeba monitorovat. Zabbix server je pak jakási centrální jednotka zpracovávající přichozí data z jednotlivých agentů. Na základě nastavených kritérií zasílá tato platforma, například pomocí e-mailu nebo sms, upozornění o možných problémech. Díky své schopnosti monitorovat a sledovat výkonnost a stav IT infrastruktury v reálném čase je Zabbix hojně využíván v operačních centrech, jakými jsou NOC[2] nebo SOC[3].

Zabbix nabízí vlastní šablonu pro monitorování Docker kontejnerů, tato šablona obsahuje předdefinované prvky pro monitorování CPU, paměti, disku a dalších metrik Docker kontejnerů. Tyto prvky lze snadno upravit nebo rozšířit podle potřeb uživatele. Kromě této oficiální šablony lze však také nalézt spoustu komunitních

[1]Open-source je termín pro softwarové projekty publikované spolu s volně dostupným zdrojovým kódem. Uživatelé těchto projektů mají možnost zdrojový kód prohlížet a upravovat ho dle svých potřeb obvykle bez licenčních poplatků.

[2]Network Operational Center (NOC) je služba poskytující síťový dohled IT infrastruktur organizací. Cílem NOC týmů je sledování stavu sítě a síťových zařízení a odhalování jejich problémů.

[3]Security Operational Center (SOC) se oproti NOC týmům zaměřuje na kybernetické hrozby v rámci sledovaných sítí.

šablon, nebo vytvořit šablonu vlastní. Zajímavostí je také možnost nasazení jednotlivých Zabbix komponentů ve formě Docker kontejnerů. Jednou z hlavních výhod nasazení Zabbixu pomocí kontejnerů je možnost jednoduchého a vcelku rychlého škálování monitorovacího prostředí. Nasazení monitorovacího systému ve formě kontejnerů taktéž poskytuje izolaci monitorovacího prostředí od zbytku infrastruktury, což zajišťuje vetší bezpečnost a stabilitu tohoto prostředí.

Vzhledem k volně dostupnému použití této platformy může být volba Zabbixu pro monitorování různých prostředí skvělým řešením pro prakticky jakoukoliv organizaci ať už s vyššími či nižšími nároky. Nicméně konfigurace a následná údržba agentů, serverů a dalších komponent nezbytných pro monitorování prostředí vyžaduje určitou úroveň znalostí správce dohledu. Celkově lze tuto možnost považovat za solidního kandidáta výběru pro potřeby této práce, výtkou mohou být poměrně omezené možnosti vizualizace, kdy Zabbix sice umožňuje vlastní tvorbu dashboardů a různých vizualizací, existují však nástroje s obsáhlejšími možnostmi. Pokud vezmeme v potaz testovací povahu řešeného problému, kdy je vhodné přizpůsobit vizuální stránku monitoringu pro přehlednost dopadu jednotlivých útoků, může být toto rozhodujícím faktorem pro výběr jiného nástroje.

5.3 Prometheus + Grafana

Při hledání vhodného monitorovacího systému pro Docker kontejnery se nejčastěji lze setkat s řešením v podobě nástroje Prometheus, který sbírá data z jednotlivých kontejnerů společně s vizualizačním nástrojem Grafana.

Prometheus

Prometheus [33] je open-source nástroj pro monitoring sbírající data pomocí takzvaných eportérů, které jsou připojeny k aplikacím nebo službám, které je potřeba monitorovat. Data jsou s časovým razítkem ukládána do jednotlivých metrik, na které se lze dotazovat pomocí jazyku PromQL[4] skrze takzvaný *expression browser*. V případě monitorování Docker kontejeru je možné využít exportér od společnosti Google zvaný cAdvisor. Tento exportér shromažďuje, zpracovává a následně exportuje informace o spuštěných kontejnerech. V konfiguračním souboru Promethea je pak nastaven cAdvisor jako zdroj dat, společně s intervalem vyzvedávání dat. Výsledek lze poté zobrazit v prostředí Promethea pomocí tabulky nebo jej vykreslit pomocí grafu. Monitorování dat o kontejnerech je tedy možné i pouze s použitím samostatného Promethea, nejedná se však o uživatelsky přívětivou možnost z důvodu

[4]Prometheus Query Language (PromQL) je jazyk navrhnutý speciálně pro dotazování se na jednotlivé metriky monitorované Prometheem.

potřeby zadávání dotazů na jednotlivé metriky a nemožnost sestavit si přehledný dashboard[5] chtěných metrik pomocí vizuálně uspokojivých zobrazení.

Grafana

Grafana [32] je open-source platforma pro vizualizaci a analýzu dat. Grafana umožňuje připojit široké spektrum zdrojů dat, kterým je právě například Prometheus nebo různé databázové zdroje jako InfluxDB, PostgreSQL a další. V případě neexistující defaultní podpory zdroje dat je však možné vytvořit vlastní plugin pro konkrétní zdroj dat a ten použít. Obdobně jako u již zmíněných monitorovacích systémů je možné nastavit upozornění na chybové stavy a výpadky monitorovaných systémů. V případě implementovaného testovacího prostředí je brán v potaz Prometheus jako zdroj dat pro Grafanu.

Grafana umožňuje vizualizovat přijaté metriky na základě importovaných dashboardů. Pro vizualizaci lze použít již existující dasboardy[6], nebo vytvořit dashboard dle vlastních požadavků. Grafana tedy splňuje požadavky vyvíjené aplikace a výsledný monitoring se odehrává právě uvnitř tohoto nástroje.

Celkově lze označit řešení v podobě Promethea a Grafany za velmi obstojné, náročnost nasazení tohoto monitorovacího systému pro monitorování Docker kontejnerů není složitá a díky jeho velké oblibě existuje rozsáhlá podpora s potencionálními problémy ve formě různých fór napříč celým internetem. Na druhou stranu pokud je potřeba kombinovat více zdrojů a vytvářet komplexní dashboardy, tak nároky na technické znalosti velmi rostou. Za další nevýhodu může být považována nutnost používat exportéry na jednotlivé služby či aplikace, jež chceme monitorovat. Taktéž je toto řešení určeno primárně pro vizualizaci a analýzu dat, nikoliv pro ukládání velkého množství dat na dlouhodobé období.

5.4 Srovnání systémů

V této kapitole byly popsány tři potencionální systémy pro monitorování dopadu implementovaných DDoS útoků na jednotlivé Docker kontejnery. Dynatrace, Zabbix i Prometheus s Grafanou splňují požadavky pro tuto práci, nicméně každý z těchto nástrojů se v určitých aspektech více či méně liší. Dynatrace je velice pokročilá profesionální platforma pokrývající prakticky jakoukoliv potřebu co se monitorování a diagnostiky problémů týče. Je ovšem náročnější na nasazení, správu i finanční zdroje, což vzhledem k potřebám práce vede k vyřazení tohoto nástroje z výběru.

[5] Dashboard je typem grafického rozhraní, které zobrazuje více druhů informací na jednom místě.

[6] Existující dashboardy lze získat na oficiálních stránkách Grafany viz `https://grafana.com/grafana/dashboards/`.

Platforma Zabbix je na druhou stranu zdarma a taktéž poskytuje dostatečné možnosti, avšak ve srovnání s Prometheem a Grafanou je stále náročnější na nasazení a údržbu. Co se týče monitorování Docker kontejnerů, poskytuje tento nástroj méně robustní řešení než Dynatrace nebo Grafana s Promeethem. Možnosti vizualizace a tvorby dashboardů v tomto nástroji nejsou tak pestré jako v Grafaně. Poslední zmíněné řešení poskytuje všechny potřebné možnosti. Grafana i Prometheus je volně distribuován a i přes potencionální problémy v podobě potřeby více nástrojů pro funkční monitorování a omezení dlouhodobějšího ukládání dat se zdá být toto řešení tím nejvhodnějším. Oba z prvních dvou popsaných nástrojů jsou spíše řešením pro organizace nežli projekty jako je tento, Prometheus s Grafanou více sedí k požadovaným účelům. Srovnání základních možností, vztahujících se k monitorování Docker kontejnerů je zaneseno do tabulky níže.

	Dynatrace	Zabbix	Prometheus + Grafana
Vhodné pro	Organizace potřebující komplexní nástroj pro správu IT infrastruktury	Projekty/organizace s nízkými nároky na vizualizaci	Projekty/organizace potřebující dobře škálovatelný systém
Podpora monitorování Docker kontejnerů	Ano, automatizovaná konfigurace a sběr metrik	Ano, pracnější na konfiguraci	Ano, velmi dobrá, nutnost nasazení více nástrojů
Možnosti vizualizace	Široké možnosti	Méně obsáhlé možnosti	Široké možnosti
Složitost nasazení	Náročnější	Střední	Poměrně jednuduchá
Open-source	Ne	Ano	Ano
Cenová dostupnost	Poplatky za licence v závislosti na počtu hostů a metrik	Zdarma	Zdarma

Obr. 5.1: Srovnání potencionálních nástrojů pro monitorování a vizualizaci

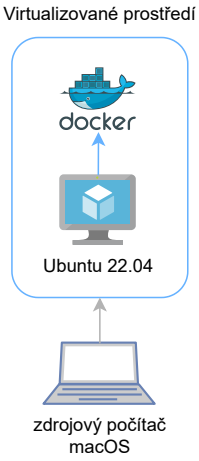
V této fázi je tedy již vybrán nástroj pro monitorování Docker kontejnerů a následného dopadu implementovaných útoků. Pro své možnosti vizualizace dat, schopnost kombinování datových zdrojů a vcelku snadného nasazení bylo zvoleno řešení v podobě vizualizačního nástroje Grafana a nástroje Prometheus pro sběr dat. Nyní je možné přejít z návrhu nástroje pro testování DDoS útoků k jeho implementaci.

6 Implementace webové aplikace

Implementace aplikace je rozdělena do dvou etap, pričemž první etapa má za cíl vytvořit pomyslnou funkční kostru celé aplikace, etapa druhá se poté zabývá kompletací vytvořeného nástroje jako logického celku a implementací vhodných rozšíření.

6.1 Testovací prostředí

Vývoj webové aplikace probíhá na operačním systému macOS s procesorem založeným na architektuře ARM[1]. Jako testovací prostředí byla zvolena linuxová distribuce Ubuntu ve verzi 23.04. Virtualizaci tohoto prostředí zajišťuje software VMware Fusion od společnosti VMware. Virtuálnímu stroji Ubuntu jsou přidělena čtyři procesorová jádra a 8192 *MB* operační paměti. Hlavním programovacím jazykem byl zvolen Python, konkrétně Python 3.10.6, přičemž pro tvorbu webové aplikace byl vybrán framework[2] Flask. Úprava zdrojového kódu pak probíhá pomocí editoru Visual Studio Code, vyvíjeného společností Microsoft. K tvorbě jednotlivých kontejnerů je použita dříve popsaná platforma Docker (viz kapitola 3) prostřednictvím knihovny pro Python. Postup tvorby webové aplikace byl průběžně nahráván do git[3] repozitáře platformy GitHub.



Obr. 6.1: Architektura vývojového prostředí

[1]Procesory založené na architektuře ARM (Advanced RISC Machines) představují odlišný přístup k hardwaru systému, dříve se používaly převážně pro mobilní telefony či tablety, nyní postupně pronikají do světa počítačů.

[2]Webový framework je nádstavba programovacího jazyka, jež ulehčuje implementaci aplikace.

[3]Git je systém sloužící ke sledování změn v souborech počítače, nejčastěji používán při vývoji softwaru.

6.1.1 Tvorba webové aplikace

Aplikace je tvořena pomocí webového frameworku psaného v jazyce Python – Flask. Pro vytvoření dynamického obsahu je použita kombinace HTML, Pythonu a Javascriptu. HTML (Hypertext Markup Language) poskytuje základní syntaxi pro tvorbu webových stránek. Python je programovací jazyk s širokým množstvím využitelnosti, jedná se například o vývoj webových stránek, softwaru nebo vizualizace a analýza dat [35]. JavaScript je skriptovací jazyk pro vytváření dynamického obsahu webových stránek. Slouží k vytváření a ovládání prvků pro zlepšení interakce uživatele s webovou stránkou [36].

Pro vizuální úpravy webové aplikace byl zvolen open source framework Bootstrap. Tento framework je postaven na HTML, kaskádových stylech CSS a jazyku JavaScript. Použití Bootstrapu funguje na vkládání kódu do předem definovaného systému mřížek, což urychluje vytváření webové stránky [37].

Aplikace ve Flasku stojí na takzvaných cestách (anglicky *route*)„ kdy pomocí dekorátoru `@app.route()` je označena funkce, která spadá pod specifickou URL[4] adresu. Například pro odstranění konkrétního bota (viz výpis 6.1) je použita cesta */remove_bot*, kdy je zavolána funkce *remove_bot()*, v níž je získána hodnota kontejneru z formuláře a následně provedena a vrácena návratová hodnota funkce *remove_bot()*.

Výpis 6.1: Demonstrace tvorby cesty ve Flasku

```
@app.route("/remove_bot", methods=["POST"])
def remove_bot():
    bot_id = request.form.get('container_id')
    return bot_management.remove_bot(bot_id)
```

Zavoláním této cesty je ovšem uživatel přesměrován na danou URL adresu, nicméně ne vždy je toto požadovaným chováním, někdy je pouze potřeba provést funkci definovanou touto cestou. Toto zajišťuje skupina technologií AJAX (Asynchronous JavaScript and XML). AJAX umožňuje odeslat na server pouze potřebná data a svižněji je zpracovat bez nutnosti neustálého znovunačítání stránky [38]. Příklad využití této technologie je zobrazen na výpisu 6.2, kdy po potvrzení formuláře pro generování botnetu je nejprve pomocí `e.preventDefault()` zabráněno změně URL adresy a následně je proveden AJAX požadavek, kdy je v případě úspěchu zavolána další série funkcí definovaných JavaScriptem, v opačném případě je do konzole vypsána příslušná chyba (error).

[4]URL (Uniform Resource Locator) je řetězec znaků sloužící k identifikaci umístění informací na internetu.

Výpis 6.2: Použití technologie AJAX pro funkci generující botnet

```
$("#generate_botnet_form").on("submit", function (e) {
    e.preventDefault();
    var formData = $(this).serialize();
    $.ajax({
      type: "POST",
      url: "/generate_botnet",
      data: formData,
      success: function (data) {
        console.log(data);
        alert(data);
        updateBotCount(); },
      error: function (xhr, status, error) {
        console.log("Error: " + error); },
    });
  });
```

6.2 První etapa implementace

V první etapě jsou implementovány funkce pro dosažení požadovaného chování aplikace. Aplikace umožňuje vytvořit a konfigurovat libovolný počet kontejnerů představující botnet, výběr a provedení tří různých DDoS útoků, zobrazení detailu jednotlivých botů i serveru. Zprovozněny jsou také všechny potřebné komponenty pro funkční monitoring. Tyto komponenty, stejně jako kontejner představující oběť (viz 6.2.3), jsou generovány pomocí programu *Docker Compose* (popsán v kapitole 3.4). Mimo zmíněné objekty je pomocí tohoto programu vygenerována síť typu *bridge* (viz Docker sítě *3.3.3*) s názvem *testbed*, do které jsou přiřazovány všechny Docker objekty podílející se na testování. Vizuální podoba domovské stránky aplikace v první etapě implementace je zobrazena na obrázku 6.3.

6.2.1 Systém ukládání dat

V rámci aplikace je z mnoha důvodů nutné ukládat data jak o botech, cílovém serveru tak i konfiguraci jednotlivých útoků. Pro tuto potřebu byla zvolena databáze SQLite. SQLite [39] je databázový systém psaný v jazyce C, jedná se o lehké a jednoduché databázové řešení, snadno použitelné v aplikacích. SQLite nepoužívá databázový server, není tedy potřeba nic nastavovat a samotná data se ukládají do jednoho souboru, jedná se proto o ideální řešení pro potřeby vyvíjené aplikace. SQLite je do aplikace vložena formou knihovny *sqlite3*, přičemž na začátku chodu aplikace je

automaticky vytvořena databáze *database.db* a v ní tabulky pro data o botech, oběti a pro konfiguraci jednotlivých útoků. Mimo vytvoření tabulek jsou také vložena data o automaticky vygenerovaném kontejneru oběti a o výchozí konfiguraci jednotlivých útoků.

6.2.2 Tvorba kontejnerů představující botnet

Pro potřebu tvorby kontejnerů představujících jednotlivé boty byl zvolen obraz linuxové distribuce Ubuntu jako základní operační systém. Oficiální Ubuntu obraz je však velmi omezený, pro potřebu testování byl proto vytvořen nový obraz vycházející z Ubuntu a nahrán na oficiální registr obrazů Docker Hub pod názvem *kralluk/ubuntu_for_ddos*. Tvorba obrazu je nastíněna již dříve v podkapitole o Docker obrazech (viz 3.3.1). Pro možné použití aplikace i na bežné architektuře procesorů *x86-64* byl tento obraz jako multiplatformní[5]. Způsobů, jak vytvořit multiplatformní obraz je více, nicméně z důvodu dostupnosti zařízení právě s potřebnou architekturou x86-64 byl zvolen způsob vytvoření obrazu pro obě architektury přímo na jednotlivých architekturách a následné vytvoření manifestu[6]. Sekvence příkazů vedoucího k vytvoření obrazu použitého v této práci je zobrazeno na výpisu 6.3. Tvorba (příkaz *build*) a nahrání (*push*) je provedeno odděleně na každé z příslušných architektur, vytvoření a nahrání manifestu je následně provedeno pouze jednou na jednom ze zařízení.

Výpis 6.3: Sekvence příkazů k vytvoření multiplatformního obrazu

```
docker build -t kralluk/ubuntu_for_ddos:v1.2-arm64 .
docker build -t kralluk/ubuntu_for_ddos:v1.2-x86_64 .

docker push kralluk/ubuntu_for_ddos:v1.2-arm64
docker push kralluk/ubuntu_for_ddos:v1.2-x86_64

docker manifest create kralluk/ubuntu_for_ddos:v1.2
--amend kralluk/ubuntu_for_ddos:v1.2-x86_64
--amend kralluk/ubuntu_for_ddos:v1.2-arm64

docker manifest push kralluk/ubuntu_for_ddos:v1.2
```

Co se týče vytvořeného obrazu, jedná se o rozšíření obrazu Ubuntu o čtyři utility a to *ping*, *slowhttptest*, *hping3* a *iproute2*. Utilita ping byla nainstalována z důvodu snadnějšího testování chování kontejnerů, slowhttptest a hping3 pro samotné

[5]Docker obraz může podporovat různé architektury, přičemž při stažení daného obrazu je automaticky Docker démonem vybrána verze pro přítomnou architekturu.

[6]Manifest je soubor metadat, který popisuje dostupné varianty obrazu pro různé architektury.

spuštení implementovaných útoků a iproute2 pak pro možnost síťového omezení jednotlivých botů. Tvorba kontejnerů se provádí pomocí formulářových prvků, přičemž uživateli je poskytnuta možnost zvolit počet jader procesoru, množství operační paměti, procentuální ztráta paketů, omezení přenosové rychlosti a zpoždění přenosu dat v síti. Aplikace umožňuje zvolit libovolný počet botů a v případě potřeby vygenerovat boty další. Všechna potřebná data ohledně generovaných botů se ukládají do SQLite databáze pro další manipulaci s nimi.

Výpočetní zdroje botů

Výběr počtu jader je v rámci knihovny *docker* řešen pomocí volby `cpu_period` a `cpu_quota`. Tato volba je možná při tvorbě kontejneru v Dockeru také volby `cpu_count`, nicméně tato volba je v použité knihovně pro Python zpřístupněna pouze pro operační systém Windows. Jak uvádí oficiální dokumentace pro Docker (viz [29]), alternativou pro statické nastavení počtu jader procesoru je určení periody a kvóty procesoru. Nastavením periody na hodnotu *100000* a kvóty na *150000* je dosaženo stejného výsledku jako nastavením počtu jader na *1,5*. Konfigurace jader ve vyvíjené aplikaci je tedy založena na statickém nastavení periody na hodnotu *100000* a v závislosti na počtu jader zvolených uživatelem se mění hodnota kvóty, čímž je dosaženo požadovaného počtu jader. Operační paměť botů je nastavena pomocí volby `mem_limit`. Uživatel je schopen zvolit, zdali chce zadávat operační paměť v megabajtech (*MB*) nebo v gigabajtech (*GB*). Pro zajištění opravdové limitace operační paměti je současně automaticky nastavena hodnota výměnné paměti[7] (anglicky *swap memory*) na stejnou jako zvolená operační paměť pomocí volby `memswap_limit`. Tímto je dosaženo toho, že generovaní boti mají k dispozici pouze zvolenou paměť i v případě, že již nebude žádná volná k dispozici. Před vygenerováním jednotlivých kontejnerů je také provedeno porovnání dostupných zdrojů hostovského zařízení s požadovanou limitací procesoru a paměti na jednotlivý kontejner, v případě, že uživatel požaduje přidělit větší počet jader nebo operační paměti než má k dispozici, je aplikací informován a botnet není vygenerován.

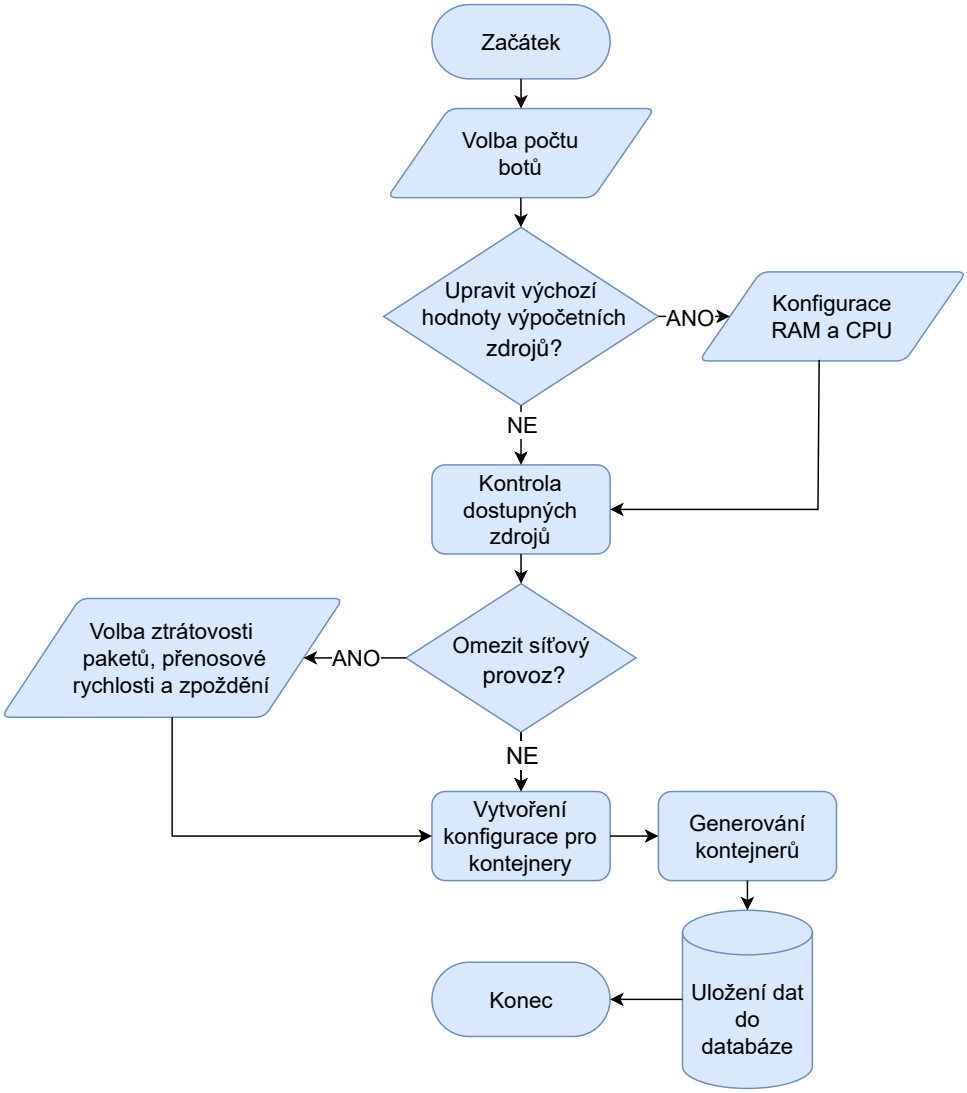
Síťová omezení botů

Jak již bylo uvedeno výše, uživatel má možnost nastavit procentuální ztrátu paketů, omezení přenosové rychlosti a zpoždění přenosu dat v síti. Ztrátovost paketů je procentuální, u omezení přenosové rychlosti (jinak řečeno šířka pásma, anglicky

[7]Swap memory (neboli výměnná paměť) je druh paměti v počítačích, která slouží k dočasnému ukládání dat, když je operační paměť (RAM) plná. Pokud operační systém potřebuje více paměti RAM než má k dispozici, přesune neaktivní část RAM na disk, aby uvolnil místo pro aktivní aplikace.

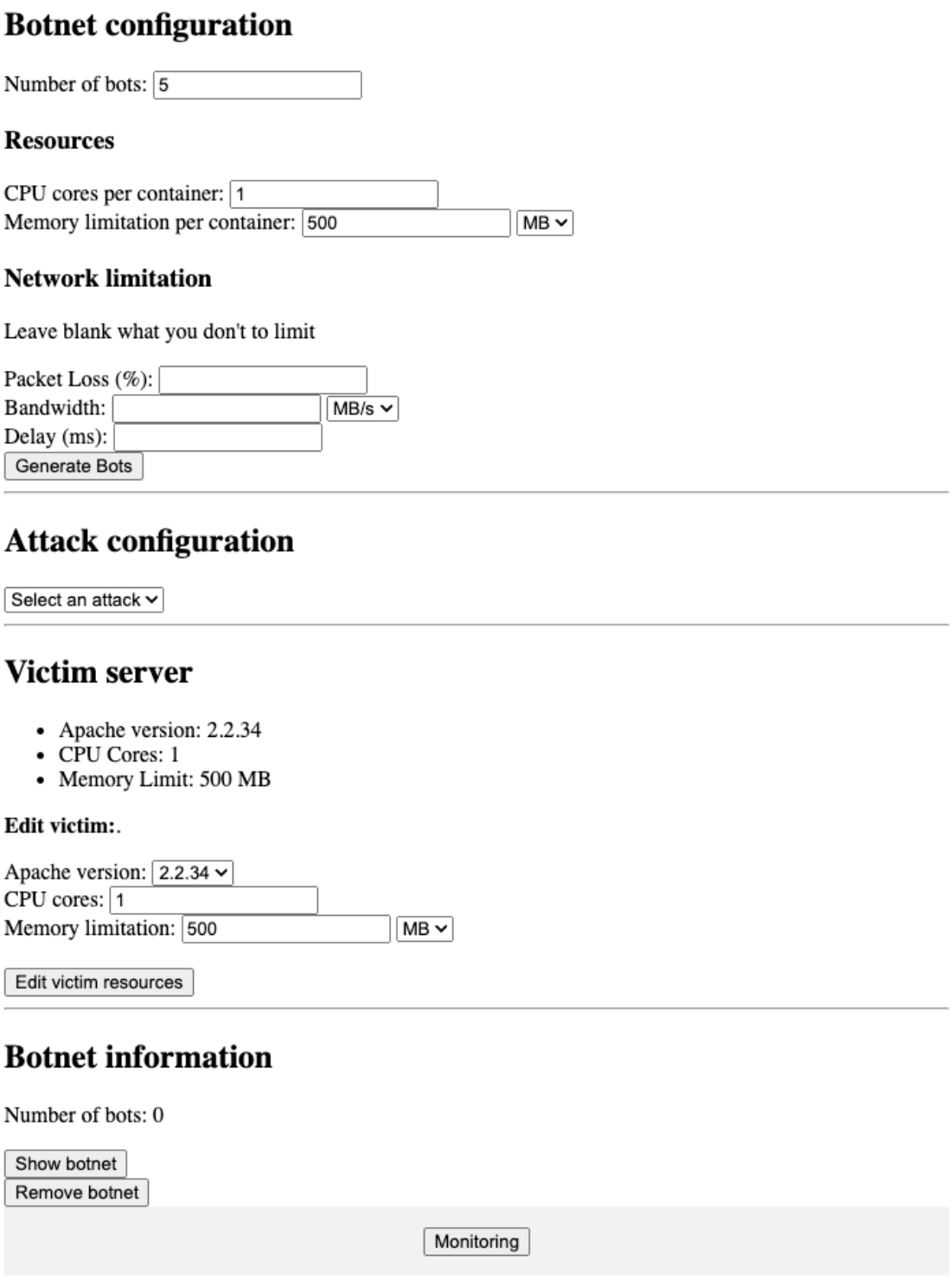
bandwidth) se nachází možnost výběru mezi omezením v kilobajtech, megabajtech nebo gigabajtech, zpoždění je udáváno v milisekundách. Docker tato omezení neumožňuje, z toho důvodu byla přidána utilita *iproute2* do obrazu na kterém jsou jednotlivé kontejnery představující boty založeny. K těmto omezením v programu iproute2 slouží příkaz `tc` (traffic control), konkrétně s volbou `qdisc netem`. Tento příkaz umožňuje nastavit různé typy front[8], které se používají k řízení toku dat v síti. Na základě uživatelova výběru síťového omezení při tvorbě botů je na pozadí aplikace vytvořen příkaz, který je předán do konfigurací jednotlivých kontejnerů.

Průchod generováním botnetu je zanesen do následujícího vývojového diagramu.



Obr. 6.2: Vývojový diagram algoritmu pro generování botnetu

[8]Fronta je datová struktura používána k řízení toku dat v síti, uchovává pakety, které nejsou aktuálně odesílány, ale mají být přeneseny.



Obr. 6.3: Vizuální podoba domovské stránky aplikace v první etapě implementace

6.2.3 Kontejner představující oběť

Cílem testovaných útoků byl zvolen kontejner založený na obrazu Apache. Aplikace umožňuje výběr mezi třemi verzemi tohoto webového serveru, a to: *2.2.34*, *2.4.29* a *2.4.56*. Po spuštění aplikace je automaticky pomocí *Docker Compose* vygenerován Apache kontejner první zmíněné verze a pro následnou lepší manipulaci mu je přidělen název *victim*.

V rámci vytvoření tohoto kontejneru je namapován port 80 hostitele na port 80 kontejneru, Apache je poté dostupný na adrese *http://localhost:80*. Pro potřebu monitorování Apache serveru (viz zprovoznění monitoringu v kapitole 4.2.3) je taktéž potřeba zpřístupnit modul *mod_status*. Toto je dosaženo vytvořením správného configuračního souboru *httpd.conf* pro příslušné verze Apache a namapování jej pomocí takzvaného *datového svazku*[9] (anglicky *volume*) do kontejneru na umístění */usr/local/apache2/conf/httpd.conf*. Po uspěšné konfiguraci modulu je možné jej zobrazit na adrese *http://localhost:80/server-status*. Část konfiguračního souboru pro Apache 2.2.34 zprovozňující daný modul je zobrazen na výpisu 6.4. Co se výpočetních zdrojů týče, kontejneru je přiděleno jedno jádro procesoru a 500 *MB* operační paměti. Verze Apache a výpočetní zdroje jsou uloženy do databáze, tato data jsou pro přehlednost zobrazena na domovské stránce a případě editace kontejneru představující tento server jsou data aktualizována.

Výpis 6.4: Sekce souboru httpd.conf zprovozňující modul mod_status

```
<VirtualHost *:80>

<Location /server-status>
      SetHandler server-status
      Order allow,deny
      Allow from all
</Location>

</VirtualHost>

ServerName localhost
ExtendedStatus On
```

[9]Datové svazky jsou v Dockeru způsob jakým sdílet umístění na hostitelském zařízení s umístěním v kontejneru. To umožňuje kontejnerům pracovat s různými daty bez nutnosti jejich kopírování a celkově usnadňuje práci s daty.

6.2.4 Konfigurace a provedení útoků

Pro potřeby testování byly vybrány a implementovány tři útoky na dostupnost služby, jež jsou popsány dále. Exekuce těchto útoků probíhá pomocí již zmíněných utilit *slowhttptest* a *hping3*. Uživatel je v této fázi schopen v sekci *Attack configuration* pomocí rozbalovací nabídky vybrat jeden útok a ten provést ze všech botů ve stejnou chvíli.

ICMP Flood

ICMP Flood (viz 1.2.1) spočívá v posílání velkého množství *echo request* zpráv na cílové zařízení. Obdobně jako při ostatních útocích je zde staticky nastaven cíl na kontejner představující server oběti a ke spuštění útoku je použit program *hping3* (viz [40]). Nabídka konfigurace tohoto útoku zahrnuje nastavení doby útoku a možnost podvržení IP adresy zdrojového zařízení. K provedení útoku ICMP Flood v rámci použitého programu slouží příznak `--icmp` společně s příznakem `--flood`. Pro podvržení zdrojové IP adresy je zde použit příznak `-a`, za nímž následuje zvolená IP adresa.

Slowloris

Útok Slowloris (1.2.2) je v aplikaci prováděn pomocí utility *slowhttptest*, přičemž veškeré možnosti této utility lze nalézt v příslušné dokumentaci viz [41]. Zmíněný útok je v tomto programu nastaven jako výchozí mód, případně jej lze nastavit přepínačem `-H`. Uživatel má k dispozici zvolit počet spojení, které každý bot naváže s obětí a také počet spojení navázaných za sekundu. Toto je v dané utilitě provedeno příznaky `-c` a `-r`. Výchozí hodnotou pro počet navázaných spojení i počet spojení za sekundu je hodnota 50.

Slow Read

Slowread (1.2.2), stejně jako útok předchozí, je prováděn taktéž pomocí utility *slowhttptest*, zvolením přepínače `-X`. Uživatel je schopen nastavit počet spojení, počet spojení za sekundu, počet opakování stejného požadavku v rámci spojení, interval mezi operacemi čtení, počet bajtů ke čtení, začátek a konec rozsahu inzerované velikosti okna TCP. První dvě možnosti jsou nastaveny stejným způsobem jako při útoku Slowloris, výchozí hodnoty jsou taktéž stejné. Opakování požadavku v rámci jednoho TCP spojení je voleno příznakem `-k`, výchozí hodnota je 1 a maximální možná 10. Interval mezi jednotlivými čteními ze serveru (příznak `-n`) je volen v sekundách s výchozí hodnotou jedna sekunda. Počet bajtů, které klient ze serveru v tomto intervalu čte je pak nastaven pomocí `-z` s výchozí hodnotou 5 *B*. Poslední

možností je nastavení začátku a konce inzerovaného TCP okna, neboli rozsah přijímajícího okna. Příznaky `-w` a `-y` lze nastavit počátek a konec bajtového okna, ze kterého je poté náhodně vybrán objem dat, který je klient schopen zpracovat před potvrzením. Pokud je okno příliš malé, server bude muset čekat na potvrzení od klienta pro každý datový segment, který odesílá, což může způsobit zpomalení komunikace. Velikost okna je ve výchozím nastavení volena z rozsahu 1–512 *B*.

6.2.5 Monitoring kontejnerů

V kapitole zabývající se výběrem vhodného monitorovacího systému pro potřeby této práce (viz 5) byla vybrána kombinace nástrojů Grafana a Prometheus. První etapa implementace testovacího prostředí se zabývá zprovozněním daného monitoringu. Všechny potřebné komponenty jsou nasazeny pomocí Docker kontejnerů prostřednictvím programu *Docker Compose*. Uživatel je následně schopen překliknout se do prostředí monitoringu pomocí tlačítka umístěného na domovské stránce aplikace.

Prometheus

Prometheus, v rámci zvolené architektury monitorování, je nástroj sloužící především pro sběr dat pomocí exportérů. Ke sběru všech potřebných metrik je zapotřebí nasazení dvou exportérů. Prvním z exportérů je nástroj *cAdvisor* starající se o shromáždění, zpracování a exportování základních informací o spuštěných kontejnerech. Stejně jako zbylé komponenty tvořící monitorování je tento exportér nasazen ve formě kontejneru, přičemž je použit neoficiální obraz cAdvisoru *zcube/cadvisor*. Důvodem změny z oficiálního obrazu[10] je jeho neexistující podpora architektury procesoru zařízení, na němž je tvořeno testovací prostředí. Kvůli potřebě vizualizace metrik Apache, které nejsou součástí základních metrik, jenž sbírá cAdvisor je nutné nasadit ještě jeden exportér. Tento druhý exportér musí zajistit sběr informací z modulu *mod_status*. Na základě více než milionu stažení a tím relativně velké popularity a pravděpodobné spolehlivosti byl vybrán exportér s názvem *lusotycoon/apache-exporter*, který splňuje nároky na potřebný nástroj. Pro sběr dat z modulu Apache je potřeba spustit exportér s příkazem, který definuje adresu sběru dat, v případě *docker-compose.yml* souboru se jedná o následující příkaz.

```
command: --scrape_uri="http://victim/server-status?auto"
```

Mohlo by se zdát, že odkazem by mělo být *http://localhost:80*, kontejner oběti se sice z pohledu hostitele kvůli mapování portů na této adrese nachází, nicméně komunikace mezi kontejnery stále stojí na jejich IP adrese nebo názvu.

[10] Oficiální obraz nese název *google/cadvisor*.

Aby Prometheus shromažďoval data z daných exportérů je potřeba vytvořit konfigurační soubor *prometheus.yml* (viz výpis 6.5) a vytvořit datový svazek do kontejneru Promethea na adresu */etc/prometheus/prometheus.yml:ro*.

Výpis 6.5: Konfigurační soubor prometheus.yml

```
scrape_configs:
- job_name: cadvisor
  scrape_interval: 5s
  static_configs:
  - targets:
    - cadvisor:8080
- job_name: apache
  scrape_interval: 5s
  - targets:
    - apache_exporter:9117
```

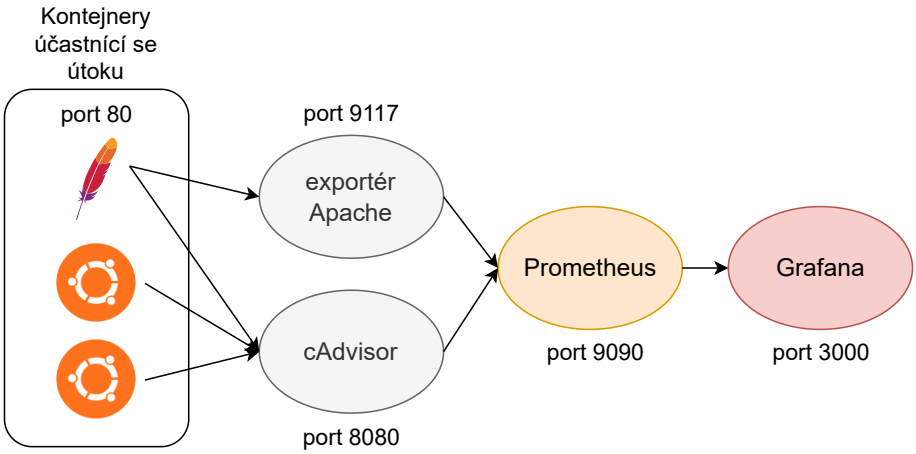
Grafana

Nyní jsou všechny nástroje pro sběr dat nasazeny, lze se tedy přesunout přímo ke konečnému nástroji zajišťující jejich vizualizaci. Pro Grafanu je použit její oficiální obraz, tedy *grafana/grafana* a dostupná je na adrese *http://localhost:3000*. Pro možnost vizualizace dat je potřeba připojit Prometheus jako zdroj dat. Toto je možné udělat přímo v grafickém prostředí Grafany, nicméně pro persistenci dat je v projektu vytvořen adresář *grafana_provisioning* obsahující dva další adresáře *datasources* a *dashboards*. Na zastřešující adresář je vytvořen datový svazek do adresáře v kontejneru Grafany */etc/grafana/provisioning*. Do adresáře „datasources“ je vložen konfigurační soubor *datasources.yaml* (viz 6.6), na základě kterého je v Grafaně automaticky nastaven Prometheus jako výchozí zdroj dat.

Výpis 6.6: Konfigurační soubor datasources.yaml

```
datasources:
    - name: Prometheus
      type: prometheus
      access: proxy
      url: http://prometheus:9090
      editable: true
      isDefault: true
```

Adresář „dashboards“ je vytvořen pro pozdější nahrání vlastního dashboardu, obsahujícího vizualizaci metrik na míru pro potřeby testování, jehož tvorbou se zabývá etapa druhá (viz 6.3.1). Princip sběru dat až po jejich konečnou vizualizaci je zobrazen na obrázku 6.4.



Obr. 6.4: Princip shromáždění dat až po jejich vizualizaci

6.3 Druhá etapa implementace

V rámci první etapy byla implementována podstatná část návrhu aplikace (4). Tato druhá etapa by se dala označit za jakési dotažení podoby testovacího prostředí do plnohodnotné formy. Cíle této etapy jsou následující:

- vytvoření Dashboardu pro vizualizaci vhodných metrik,
- implementace možnosti kombinování útoků,
- umožnění detailnější správy botnetu,
- vytvoření uspokojivé vizuální podoby aplikace.

6.3.1 Tvorba dashboardu

V tomto bodě jsou zprovozněny jednotlivé komponenty monitorovacího systému a nasbíraná data Prometheem jsou předávána do Grafany. Tvorba dashboardu v Grafaně není nikterak obtížná, v sekci *Dashboards* lze vytvořit nový dashboard a poté započít konfiguraci jednotlivých panelů. Tvorba panelu stojí na elementární znalosti jazyka PromQL s pomocí Prometheus dokumentace [33] a poměrně intuitivní vizuální úpravy panelu podle potřeb vizualizace dané metriky.

Co se týče obecných výpočetních metrik, jsou odděleně pro boty a Apache server vytvořeny panely zobrazující příchozí a odchozí síťový provoz, zatížení procesoru a využití operační paměti. Všechny tyto metriky poskytuje cAdvisor a za formu vizualizace jsou zvoleny takzvané časové řady (anglicky *time series*) v podobě grafu. Pomocí tohoto exportéru je také zobrazen počet botů, pro nějž je použita vizualizace statistika (*stat*). Jak lze pozorovat na výpisu 6.7, o toto zobrazení se stará metrika *container_last_seen*, jejímž argumentem je již zmíněný vytvořený obraz pro potřeby

uskutečnění útoků. Dotaz spočítá počet kontejnerů, které jsou založeny na daném obrazu a byly exportérem viděny v posledních 10 sekundách. V případě, že by dotaz nevrátil žádnou hodnotu, mohla by Grafana nesprávně zobrazit poslední vrácenou hodnotu nebo „no data“. Část dotazu `OR on () vector (0)` zajistí navrácení hodnoty 0 v případě, že samotný dotaz neobsahuje žádnou návratovou hodnotu [42].

Výpis 6.7: Dotaz na počet botů

```
count(container_last_seen{image="kralluk/ubuntu_for_ddos:v1.2"
}> (time() - 10)) OR on() vector(0)
```

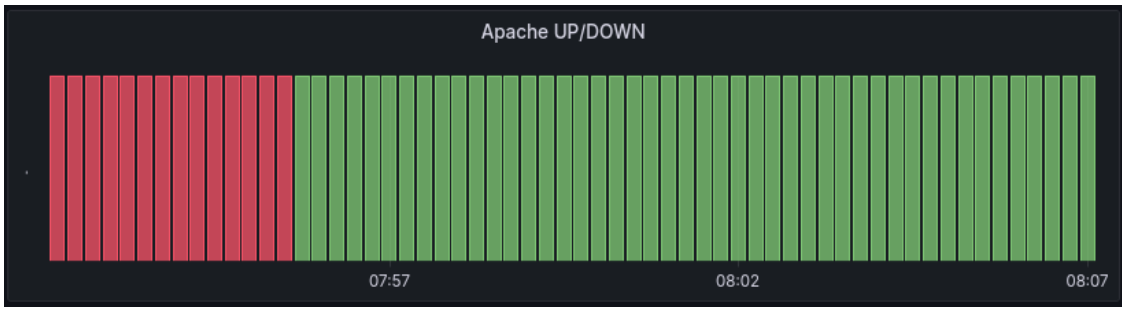
Exportér, sbírající metriky Apache z modulu *mod_status*, zajišťuje dvojici metrik `apache_up` a `apache_workers`. První metrika, jak název napovídá, vrací hodnotu 1 v případě, že Apache server běží, hodnotu 0 pak v případě opačném. Na základě těchto návratových hodnot je vytvořen panel využívající vizualizaci historie stavu (anglicky *status history*), který zobrazuje zelené svislé čáry pokud Apache běží a červené pokud ne. Pro metriku vracející počet zaneprázdněných a nečinných Apache pracovníku[11] (anlicky *busy and idle workers*) je zvolen koláčový graf. Po vytvoření uspokojivého dashboardu je pomocí Grafany vyexportován ve formátu JSON pod názvem *ddos.json* a uložen do adresáře *grafana_provisioning/dashboards*, na který je vytvořen datový svazek na příslušné místo do kontejneru Grafany (popsáno v 6.2.5). V tomto adresáři je také vytvořen soubor *default.yaml* (viz výpis 6.8), který zajišťuje, že Grafana bude v tomto adresáři dashboardy hledat. Ukázky panelů zobrazujících status Apache a jeho pracovníků jsou zobrazeny na obrázcích 6.5 a 6.6.

Výpis 6.8: Konfigurační soubor default.yaml

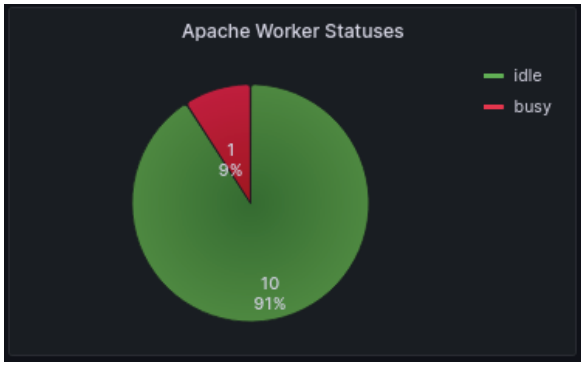
```
apiVersion: 1

providers:
   - name: 'Default'
     options:
       path: /etc/grafana/provisioning/dashboards/
```

[11] Apache pracovníci slouží k efektivnímu zpracování velkého počtu požadavků na webový server s minimálním využitím systémových zdrojů. Počet pracovníků se mění dynamicky v závislosti na aktuálním zatížení serveru a nastavení parametrů jako MaxRequestWorkers a ThreadsPerChild [43].



Obr. 6.5: Ukázka panelu zobrazujícího status Apache



Obr. 6.6: Ukázka panelu pro vizualizaci Apache pracovníků

Automatizace Grafany

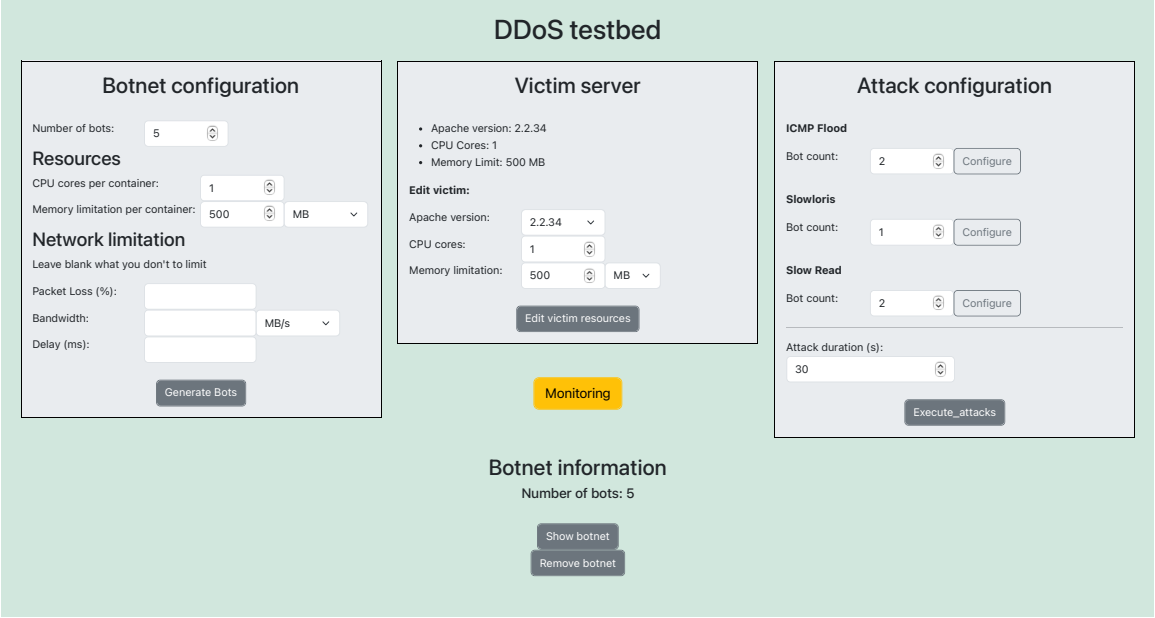
Platforma Docker umožňuje u jednotlivých obrazů definovat takzvané proměnné prostředí (anglicky *environment variables*). Tyto proměnné mohou obsahovat konfigurační informace, jako například adresu databáze nebo definici uživatelských údajů. Obraz Grafany umožňuje nastavit množství těchto proměnných, z nichž tato práce některé využívá za účelem automatizace určitých procesů. Ve výpisu 6.9 je zobrazena definice pěti proměnných prostředí v rámci definice kontejneru Grafany v konfiguračním souboru *docker-compose.yml*. První dvojice výtváří administrátorský účet s uživatelským jménem *admin* a heslem *1234*. Druhá dvojice zajišťuje přístup uživatele bez nutnosti přihlášení a přiděluje mu roli pozorovatele. Poslední proměnná definuje již uložený dashboard *ddos.json* jako domovský. Těmito proměnnými je zajištěno, že po spuštění aplikace a prokliknutí se do prostředí monitoringu se uživatel dostane ihned k pohledu na dashboard bez nutnosti jakéhokoliv zásahu. Pokud by ovšem nastala nutnost dashboard upravit, vytvořit nový, nebo jakkoliv jinak zasahovat do prostředí monitoringu, je uživatel stále schopen přihlásit se pod administrátorským účtem a toto provést.

Výpis 6.9: Definice proměnných prostředí pro Grafanu

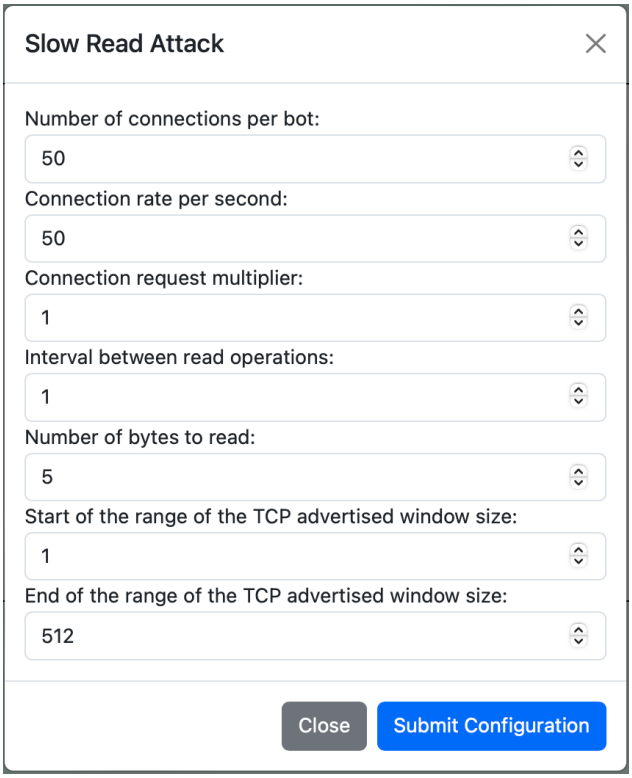
```
environment:
      GF_SECURITY_ADMIN_USER: admin
      GF_SECURITY_ADMIN_PASSWORD: 1234
      GF_AUTH_ANONYMOUS_ENABLED: "true"
      GF_AUTH_ANONYMOUS_ORG_ROLE: "Viewer"
      GF_DASHBOARDS_DEFAULT_HOME_DASHBOARD_PATH:
      /etc/grafana/provisioning/dashboards/ddos.json
```

6.3.2 Vizuální podoba aplikace

Pro vizuální úpravy aplikace byl zvolen framework Bootstrap. Instalace Bootstrapu je možná vícero způsoby, pro tuto práci je zvolena cesta stažení souborů Bootstrapu (viz [44]), jejich vložení do projektu a odkázání se na soubory kaskádových stylů CSS a skriptů Javascript. V době tvorby aplikace je zvolena nejnovější verze Bootstrapu, tedy verze *5.3.0*. Nasazení tohoto frameworku usnadňuje vizuální úpravu aplikace a také použití funkcionalit, které by jinak možné nebyly. Boostrap také zajišťuje konzistentní vzhled a chování aplikace napříč různými prohlížeči a zařízeními. Ve vztahu k vyvíjené aplikaci se jedná především o využití takzvaných modálních oken. Modální okno je interaktivní prvek webové stránky, který se zobrazuje před hlavním obsahem a dočasně s ním přerušuje interakci, dokud není okno uzavřeno. Modální okna jsou povětšinou využívána pro zobrazení dodatečných informací, formulářů, obrázků nebo videí, bez nutnosti opouštět aktuální stránku. Aplikace je rozdělena do dvou stránek, tedy domovská stránka konfigurace a stránka pro zobrazení detailu jednotlivých botů a jejich úprava (dále rozebíráno v 6.3.4). Pro obě tyto stránky byla zvolena obdobná barevná kombinace prvků. Na obrázku 6.7 je zobrazena domovská stránka po vytvoření vizuální stránky pomocí Bootstrapu. U každého z útoků se nachází tlačítko *Configure*, které po stisknutí vyvolá modální okno s konfigurací příslušného útoku. Na obrázku 6.8 je pro demonstraci zobrazeno modální okno pro konfiguraci útoku Slow Read.



Obr. 6.7: Vizuální podoba domovské stránky aplikace v druhé etapě implementace



Obr. 6.8: Modální okno pro konfiguraci útoku Slow Read

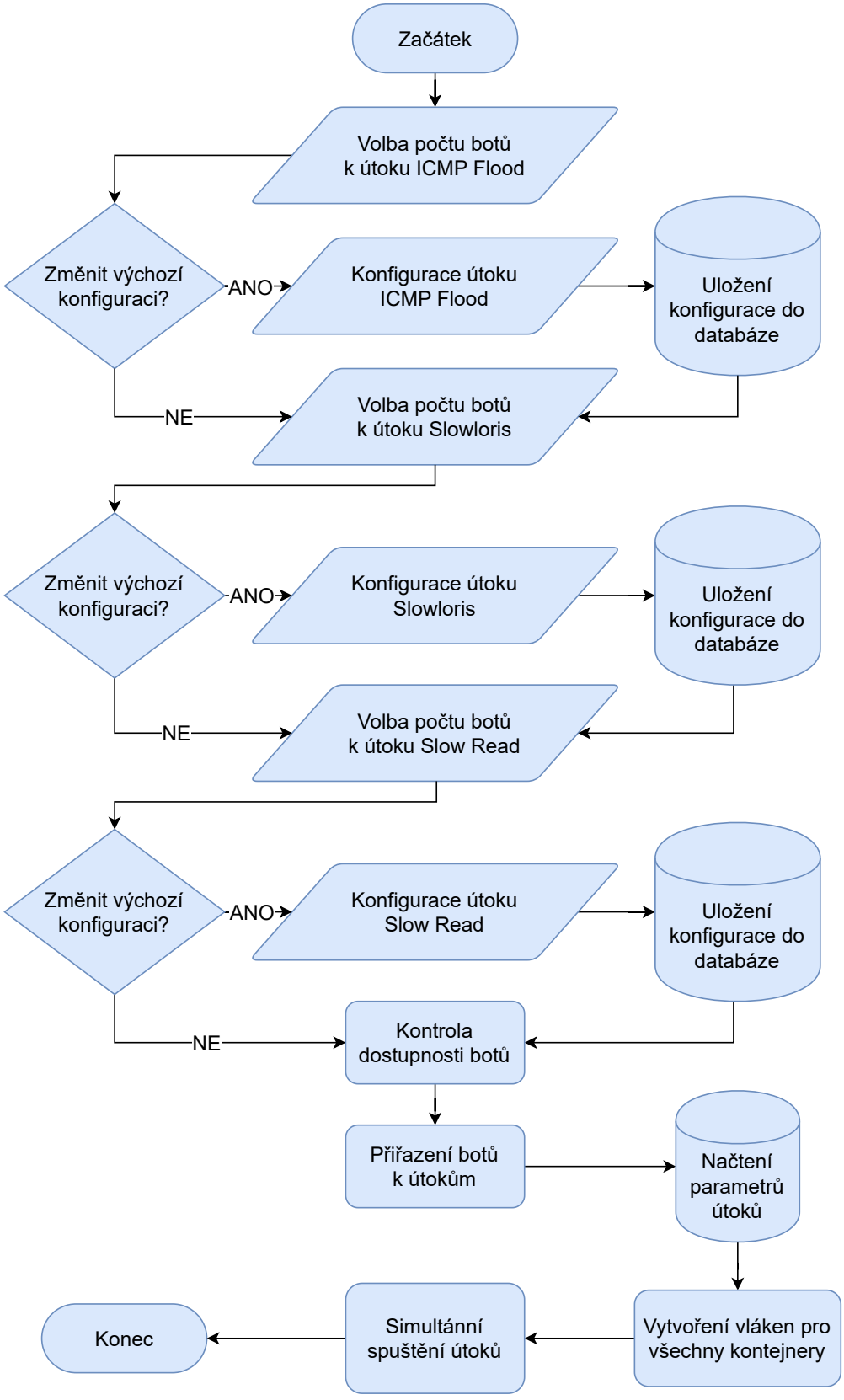
6.3.3 Kombinování útoků

V první etapě implementace bylo dosaženo možnosti vybrat jeden ze tří útoků na dostupnost služby a ten pak provést pomocí celého botnetu. Pro optimálnější možnosti testování je ovšem vhodné umožnit jednotlivým botům provádět různé útoky ve stejném čase a sledovat, jak se cílové zařízení chová pod náporem různých typů útoků. V této etapě je proto implementována možnost rozdělit boty mezi jednotlivé útoky, přičemž při překročení celkového množství dostupných botů nejsou útoky provedeny a uživatel je o této skutečnosti upozorněn. Jednotlivé útoky jsou po spuštění aplikace nakonfigurovány s výchozími hodnotami (viz kapitola o útocích 6.2.4), které jsou načteny z databáze. Uživatel má možnost pomocí již zmíněných modálních oken dle libosti útoky nakonfigurovat, přičemž data o útocích jsou následně v databázi přepisována. Ve spodní části sekce s konfigurací útoků se nachází formulářový prvek pro zvolení doby trvání útoku. Jakmile uživatel stiskne tlačítko *Execute attacks*, je předán počet botů u jednotlivých útoků funkci pro simultánní spuštění útoků. Tato funkce načte Docker indentifikátory všech botů z databáze a každému z útoků přidělí příslušný počet identifikátorů. Následně funkce ke každému útoku načte argumenty z databáze a pro každý kontejner vytvoří nové vlákno[12] spouštějící požadovaný útok. Poté jsou všechna vlákna spuštěna, čímž je proces exekuce kombinace útoků zakončen. Celý proces konfigurace útoku je zanesen do vývojového diagramu na obrázku 6.9.

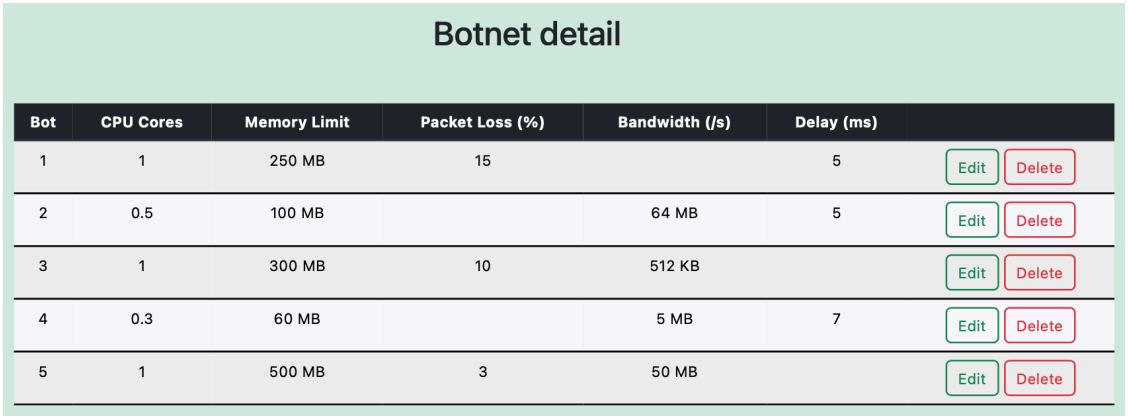
6.3.4 Zobrazení a úprava botnetu

Dalším možným vylepšením aplikace je umožnění detailnější správy botnetu. Po kliknutí na tlačítko *Show botnet* je možné dostat se do podrobného pohledu na jednotlivé boty. Ve vrchní středové části této stránky se nachází tabulka zobrazující jednolivé boty (viz obrázek 6.10), spolu s jejich specifikacemi, načtenými z databáze. Vedle každého záznamu se nachází dvojice tlačítek. Tlačítko *Edit* vyvolá modální okno, umožňující konfiguraci daného bota. Pomocí tlačítka *Delete* je tento bot, spolu se záznamem z databáze smazán. V druhé polovině stránky se nachází formulář (viz obrázek 6.11), jež umožňuje úpravu všech botů najednou. Pod tímto formulářem se, stejně jako na hlavní stránce, nachází tlačítko *Remove botnet* pro odstranění celého botnetu a tlačítko pro vrácení se na domovskou stránku.

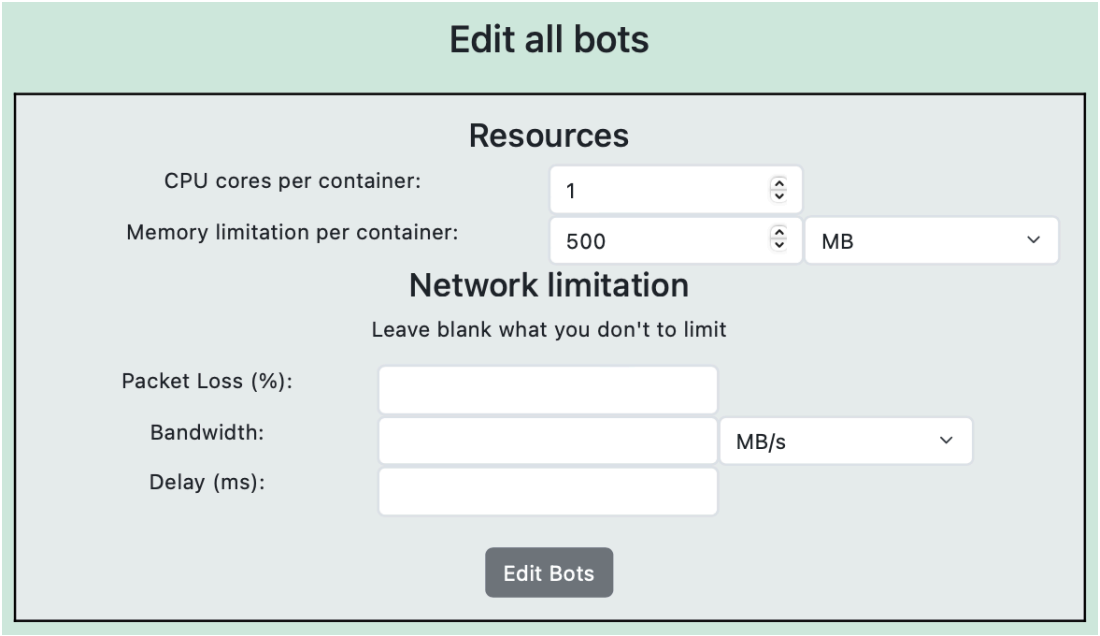
[12]Vláknování (anlicky *threading*) umožňuje souběžné spouštění různých částí programu, Python disponuje vestavěnou podporu pro vlákna pomocí modulu *threading* [45].



Obr. 6.9: Vývojový diagram konfigurace útoků



Obr. 6.10: Tabulka zobrazující podrobnosti o botnetu



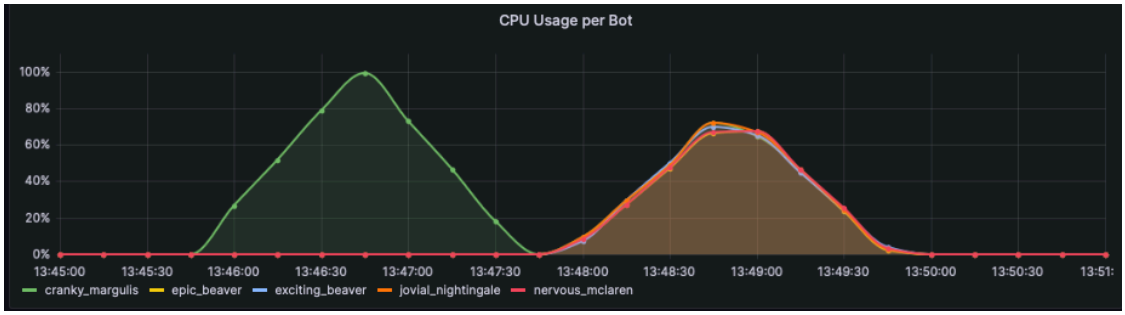
Obr. 6.11: Formulář pro jednotnou úpravu botů

7 Testování aplikace

Důležitou částí vývoje aplikace je taktéž její testování. Tato kapitola se zabývá právě touto činností, přičemž jsou nejprve otestovány jednotlivé útoky, následně pak jejich kombinace. Důležité je zde zmínit, že cílem testování není komplexní analýza jednotlivých verzí Apache serverů, nýbrž demonstrace funkčnosti jednotlivých funkcionalit a některých možných scénářů použití vytvořeného nástroje.

7.1 ICMP Flood

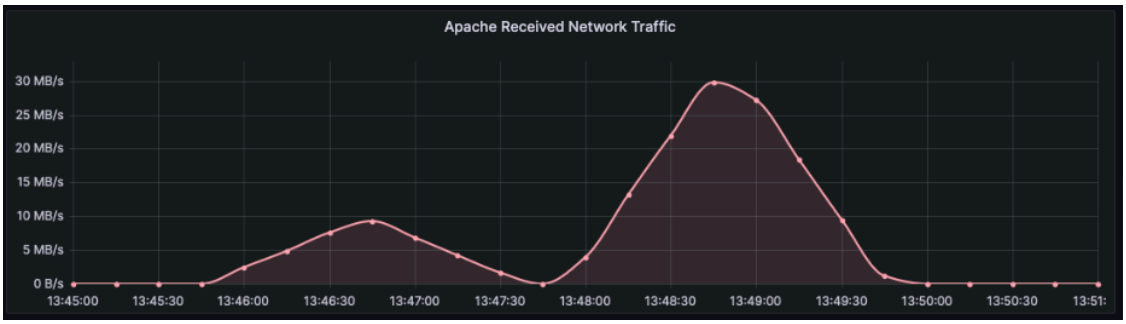
Útok ICMP Flood je v rámci implementovaných útoků ten méně komplexní. Výsledkem pozorování chování kontejnerů při exekuci tohoto útoku je velké zatížení procesoru kontejneru spouštějící tento útok. Útok ICMP Flood postupně zatíží procesor daného kontejneru na 100 % pro vygenerování co možná největšího množství paketů směřujících na server oběti. Virtuální stroj Ubuntu má k dispozici pouhá čtyři jádra procesoru, jakmile je tedy ICMP Floodu přiřazeno kupříkladu pět botů z nichž každému je přiděleno omezení na jedno jádro procesoru, nebudou výpočetní zdroje dostatečné. Toto se projeví v Grafaně jako využití procesoru na méně než 100 % jednoduše proto, že v celku není dostupná větší část procesoru. Porovnání, kdy je spuštěn minutový ICMP Flood pouze z jednoho kontejneru s jedním jádrem procesoru a poté z pěti takových kontejnerů, je zobrazeno na obrázku 7.1.



Obr. 7.1: Porovnání využití procesoru na základě jeho dostupnosti

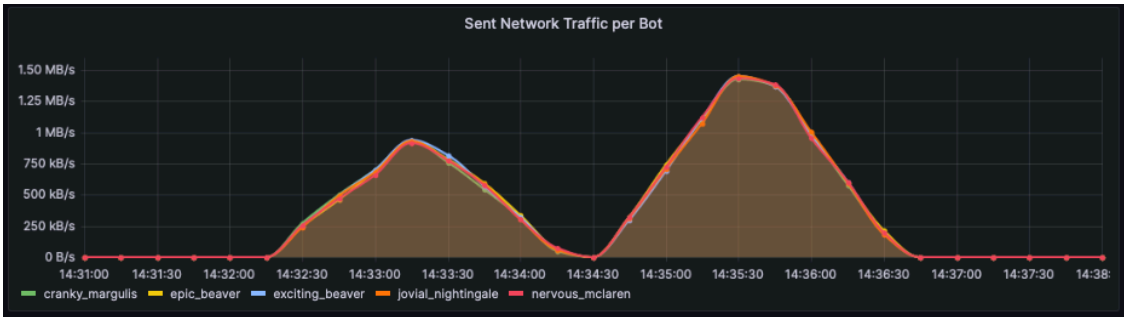
Co se týče dopadu na kontejner představující oběť, v případě útoku ICMP Flood není procesor tohoto kontejneru při limitaci na jedno jádro významně ovlivněn. Pozorovat dopad ICMP Floodu lze ovšem názorně pozorovat na síťovém provozu kontejneru. Na obrázku 7.2 je zobrazen přijatý síťový provoz Apache kontejner při stejných útocích jako na obrázku předchozím. Lze zde zpozorovat, že v případě útoku z jednoho bota bylo v jednu chvíli dosaženo téměř 10 *MB/s* příchozí síťové komunikace, nicméně při útoku z pěti botů to bylo pouhých 30 *MB/s*. Toto je

způsobeno opět nedostatečnými zdroji botů, kdy útok spuštěný z bota umožňuje botu dosáhnout až na svou limitaci v podobě jednoho procesoru, pro spuštění útoku z pěti takových botů už hostitelský systém nedisponuje dostatečnými prostředky, což se projeví menším množstvím odeslaných paketů v rámci utility *hping3* a tedy i menším vygenerovaným síťovým provozem každého z botů.



Obr. 7.2: Síťový provoz přijatý Apache

Aplikace umožňuje nastavení podvržení zdrojové adresy botů, ve výchozím nastavení je tato možnost vypnuta a boti tedy přijímají v podobě `ICMP ECHO` odpovědi stejný objem dat jako vysílají. Podvržením zdrojové adresy lze ovšem tyto zprávy přesměrovat na zvolenou adresu, což je následně pozorovatelné v podobě žádného přijatého síťového provozu u botů. Přijímání síťového provozu vyžaduje určitý výkon procesoru, odklonění tohoto provozu má proto za následek více dostupného výkonu procesoru pro bota a tím schopnost generovat větší množství paketů zahlcujících cílové zařízení. Porovnání vygenerovaného síťového provozu při ponechání zdrojové IP adresy a následně při jejím podvržení u botů s limitací na 0,2 CPU je zobrazen na obrázku 7.3.

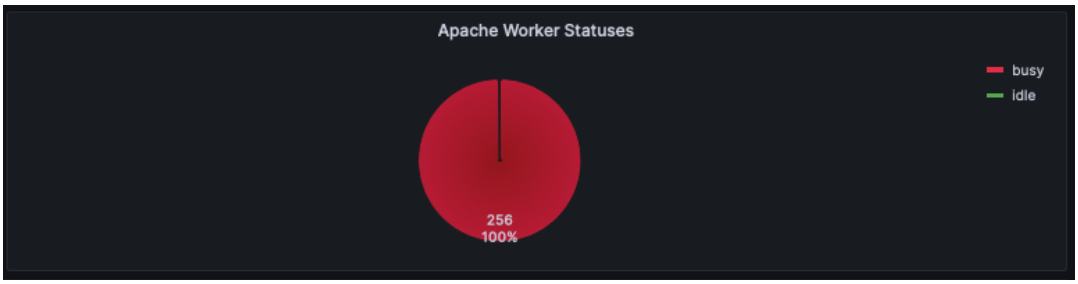


Obr. 7.3: Vygenerovaný síťový provoz bez a s podvrhnutím zdrojové IP adresy

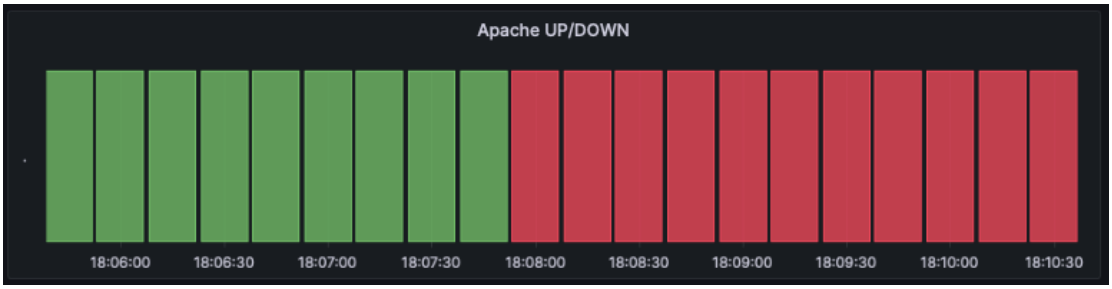
7.2 Slowloris

Útok Slowloris je testován pomocí pěti botů, každému z nich je přiděleno 250 *MB* paměti a 0,5 jádra procesoru, na stejné hodnoty je nastavena oběť, verze Apache zůstává výchozí (tedy *Apache 2.2.34*), není nastaveno žádné síťové omezení botů. Při těchto hodnotách dochází při minutovém útoku nastaveném na výchozí hodnoty, tedy pouhých 50 spojení s rychlostí 50 spojení za sekundu, opět k velkému zatížení procesoru ze strany botů, ostatní hodnoty nejsou nikterak významné. Ze strany Apache se pak jedná o vcelku vysoké zatížení paměti, zhruba 100 *MB*, nicméně počet ustanovených spojení ve výchozí konfiguraci útoku nevede k narušení jeho dostupnosti.

Druhým testem je nastavení počtu spojení na 600 a rychlosti spojení za sekundu 200, doba útoku je nastavena na 200 sekund. Tato kombinace již bez problému vede k nedostupnosti oběti, která obsluhuje tolik spojení, že další již nezvládne. V průběhu útoku je zaznamenáno stoprocentní vytížení Apache pracovníků a chvíli po spuštění útoku již Grafana zobrazuje Apache status jako *DOWN*.



Obr. 7.4: Plné vytížení Apache pracovníků



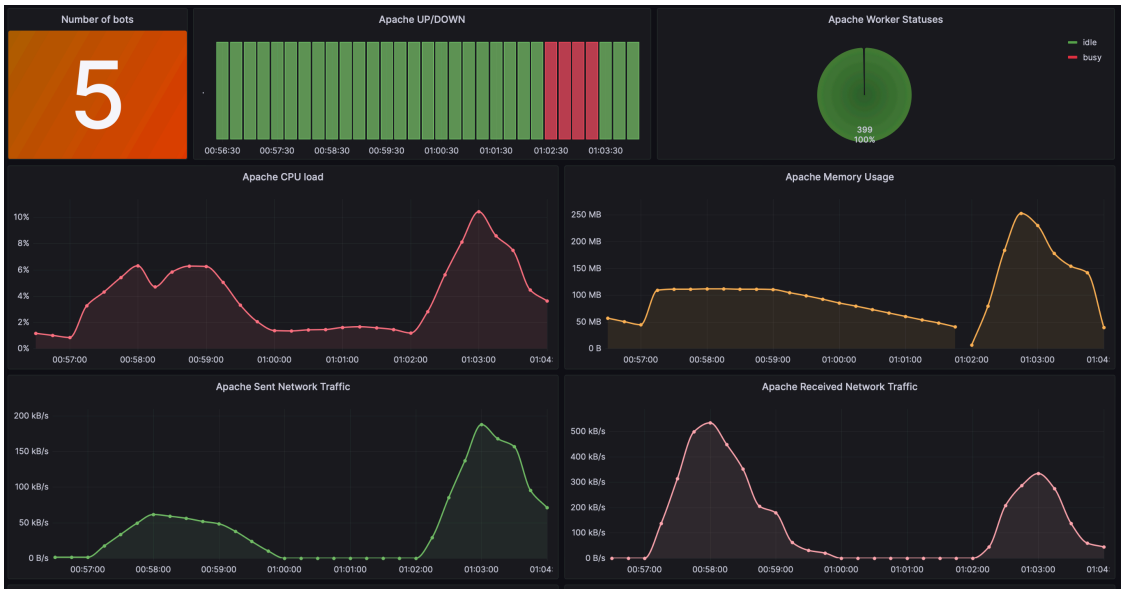
Obr. 7.5: Apache ve statusu DOWN po útoku Slowloris

7.3 Slow Read

Útok Slow Read obsahuje nejvíce možností konfigurace, což přináší obrovské množství možných kombinací k testování, níže je popsáno testování tohoto útoku demonstrující jeho konfiguraci a provedení.

7.3.1 Test rozdílu verze Apache

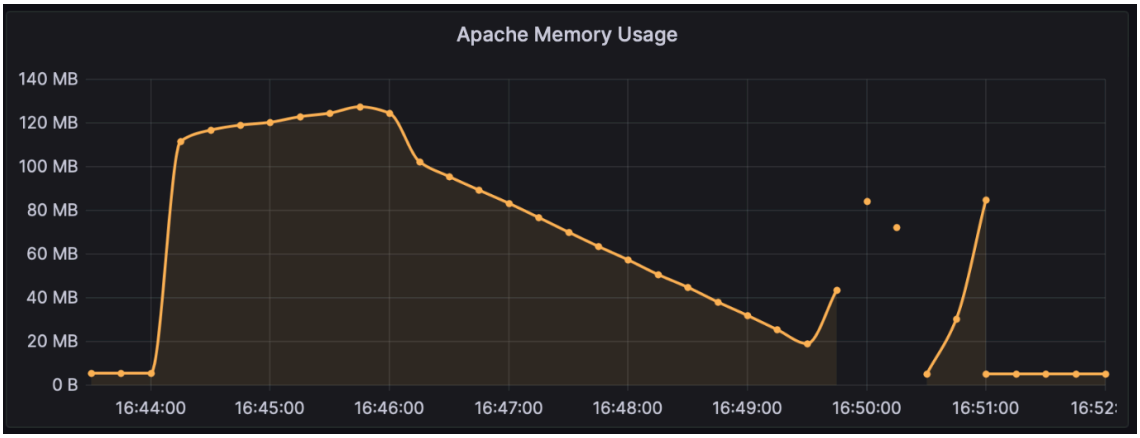
Prvním testem tohoto útoku je zjištění rozdílu chování oběti pod stejnými parametry útoku, ovšem s odlišnou verzí Apache. Vygenerováno je pět botů opět s přiděleným 0,5 jádra procesoru a 250 *MB* paměti, stejně tak oběti. Slow Read je při tomto testu nakonfigurován na 2000 spojení při rychlosti 200 spojení za sekundu, v rámci spojení je požadavek zopakován třikrát, interval mezi čtením je jedna sekunda, počet bajtů ke čtení je osm a TCP okno se pohybuje v rozsahu 512–1024 *B*, doba útoku jsou dvě minuty. Nejprve je otestována verze 2.2.34, poté je verze Apache změněna na 2.4.29 a proveden shodný útok. Dopad útoků na oběť je zobrazen na obrázku 7.6. Na tomto obrázku lze pozorovat, že prvním útoku na starší verzi Apache nedochází k výpadku služby ani k významnému zatížení výpočetních zdrojů. Průběh útoku také ukazuje počet vygenerovaných pracovníků. U prvního útoku se jednalo o 256 pracovníků, druhá testovaná verze již ovšem použila pracovníků 399, což pravděpodobně vedlo k většímu vytížení procesoru a spotřebování všech 250 *MB* dostupné operační paměti a následnému pádu serveru.



Obr. 7.6: Vizualizace dopadu útoku Slow Read s odlišnými verzemi Apache

7.3.2 Význam operační paměti

Při testování tohoto útoku lze demonstrovat důležitost dostatku operační paměti oběti. Vygenerováno je opět pět botů s 250 *MB* paměti a polovinou jádra procesoru. Konfigurace při tomto útoku je následující: 1200 spojení na bota rychlostí 150 spojení za sekundu, spojení je pětkrát opakováno, čteno je 32 *B* dat po pěti sekundách, TCP okno zůstává ve výchozím nastavení. Apache verze je zvolena 2.4.56, přiděleno je 0,5 jádra procesoru a pro první test je zvoleno 250 *MB* operační paměti, test trvá 120 sekund. Útok s touto konfigurací nevede k narušení chodu oběti, nicméně lze pozorovat využití paměti vetší než 100 *MB*, nabízí se tedy možnost otestovat chování oběti v případě, že nemá k dispozici více než těchto 100 *MB* paměti. Na obrázku 7.7 je zobrazeno porovnání těchto dvou situací. V zobrazeném intervalu probíhal nejprve útok s 250 *MB* paměti, poté byla pamět změněna na 100 *MB* a proveden shodný útok. Při nižší hodnotě již kontejener Apache nezvládal pracovat s dostupnou pamětí, což vedlo k jeho několikanásobnému restarování projevujícímu se v Grafaně jako skokové změny využití operační paměti.

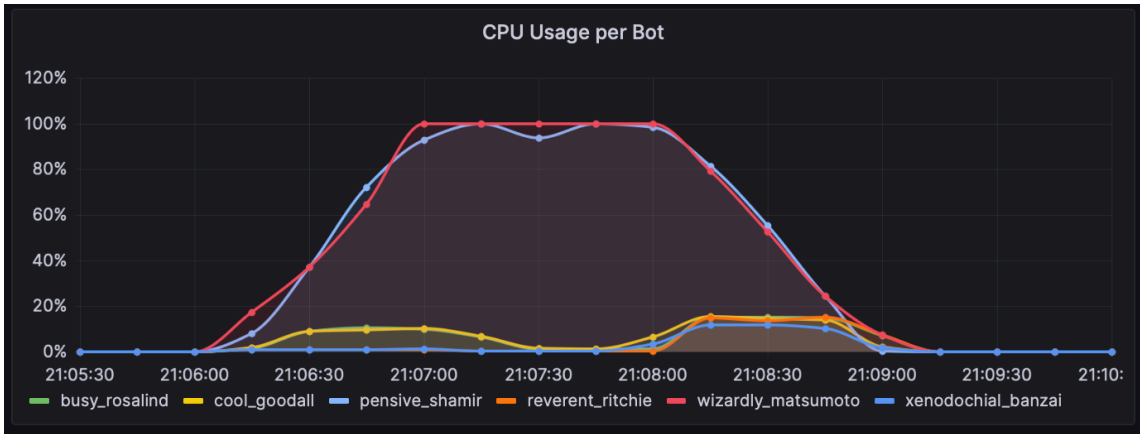


Obr. 7.7: Dopad útoku Slow Read při nedostatečné paměti

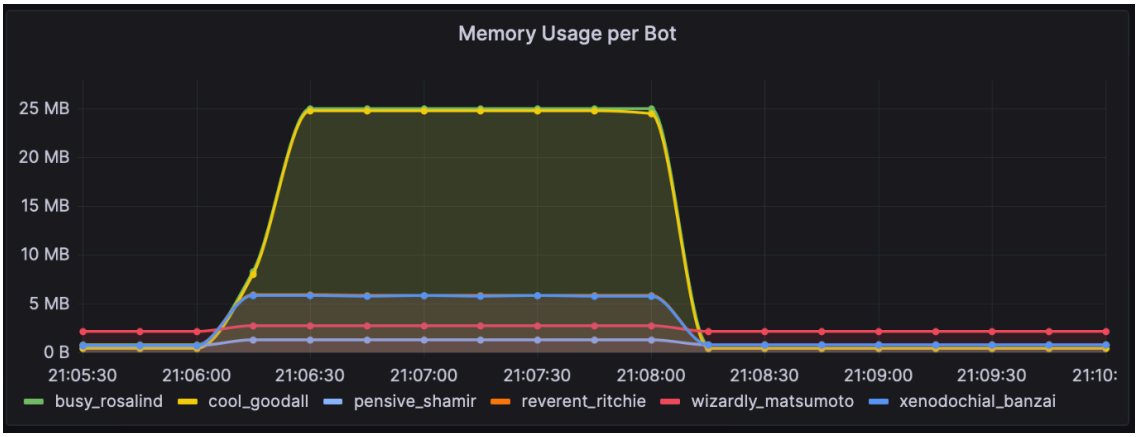
7.4 Kombinace útoků

Doposud byla popsána oddělená exekuce implementovaných útoků, aplikace nicméně umožňuje tyto útoky kombinovat. Kombinování různých útoků na dostupnost služby může vést k velice efektivnímu útoku, neboť takový útok vyčerpává různé zdroje cílového zařízení. Pro testování kombinovaného útoku je vygenerováno šest botů s 250 *MB* paměti a polovinou jádra procesoru. Ke každému z útoků jsou přiděleni dva boti. U útoku ICMP Flood je nakonfigurováno podvržení zdrojové IP adresy na adresu *1.2.3.4*, Slowloris je spuštěn s 600 spojeními při rychlosti 200 spojení za

sekundu. Slow Read ustanovuje 3000 spojení při rychlosti 200 za sekundu, každé spojení je opakováno dvakrát, čteno je 5 *B* po dvou sekundách, TCP okno je ponecháno na výchozí hodnotu. Doba útoku jsou dvě minuty a prakticky ihned po spuštění útoku dochází k pádu oběti, který přetrvává až do konce útoku. Dle dat v Grafaně nedochází na straně Apache k takové nárůstu využití procesoru nebo operační paměti, jež by měl za následek pád daného kontejneru, nicméně množství spojení s Apache pravděpodobně vede k jeho pádu. Pozorovat ovšem lze rozdílné využití zdrojů kontejnerů, které provádějí jednotlivé útoky. Například, jak bylo již otestováno dříve, útok ICMP Flood vytíží procesor bota na 100 %, nicméně využití operační paměti nijak signifikatní není. Naproti tomu u botů provádějících útok Slow Read lze pozorovat větší vytížení operační paměti než u ostatních útoků.



Obr. 7.8: Využití procesoru botů při kombinovaném útoku



Obr. 7.9: Využití operační paměti botů při kombinovaném útoku

Závěr

Cílem bakalářské práce bylo navrhnout a implementovat prostředí pro testování DDoS útoků. Prostředí se mělo skládat z webové aplikace umožňující generování botů a webového serveru v podobě Docker kontejnerů, konfiguraci a provedení vybraných DDoS útoků a následný monitoring dopadu útoků v nasazeném systému pro monitoring. Závěrečným cílem bylo uspěšné testování vytvořeného nástroje.

V rámci teoretické části práce byla nejdříve rozebrána problematika DDoS útoků, ve které byly vysvětleny základní typy komunikace botnetu, rozděleny typy těchto útoků a uvedeny zástupci těchto kategorií. Následně byl vysvětelen pojem virtualizace, konkrétně pak virtualizace platforem a její kategorie. Téma virtualizace platforem, respektive její podkategorie kontejnerizace, bylo poté rozšířeno v podobě popsání platformy Docker, nejznámějším zástupcem tohoto druhu virtualizace a nedílnou součástí praktické části práce.

V praktické části práce byl vytvořen návrh aplikace, vybrán vhodný systém pro monitoring potřebných metrik v podobě nástroju Prometheus a Grafana a následně implementována navržená aplikace. Implementace aplikace byla rozdělena do dvou etap, přičemž v etapě první byla vytvořena stěžejní část vyvíjeného nástroje v podobě vytvoření aplikace umožňující generování konfigurovatelného botnetu, konfigurace oběti i nastavení a exekuci vybraných útoků. Součástí první etapy bylo také zprovoznění systému pro monitoring v podobě nasazení jednotlivých komponent monitoringu ve formě Docker kontejnerů a jejich konfigurace. V rámci druhé etapy byl vytvořen přehledný dashboard pro vizualizaci zvolených metrik oběti i útočníků, za pomoci frameworku Bootstrap byla upravena vizuální podoba aplikace a byly implementovány rozšiřující funkce. Prvním rozšířením je umožnění kombinování útoků, tedy libovolně přiřazovat jednotlivé boty k vybraným útokům a následně spustit tyto útoky simultánně. Následně pak byla implementována možnost detailnější správy botnetu v podobě možnosti jednotlivě konfigurovat již vygenerované boty. Závěrem práce byla otestována funkčnost jednotlivých útoků i jejich kombinace v podobě demonstrace konečné vizualizace pomocí vytvořeného dashboardu v aplikaci Grafana.

Dalším směrem vývoje aplikace může být umožnění provedení více typů útoků a jejich komplexnější konfigurace. Možným rozšířením je také variabilnější výběr cílových služeb, popřípadě nasazení nástroje pro automatické ukládání testovaných dat pro možnost pozdější analýzy.

Literatura

[1] *Distributed Denial of Service (DDoS)* [online]. [cit. 2022-10-30]. Dostupné z: https://www.imperva.com/learn/ddos/denial-of-service/

[2] BORGINI, Julia. *Tackle IoT application security threats and vulnerabilities* [online]. 2021 [cit. 2022-10-30]. Dostupné z: https://www.techtarget.com/iotagenda/tip/Tackle-IoT-application-security-threats-and-vulnerabilities

[3] HUBBARD, Brian, 2022. *Defining Your IT Network Security* Perimeter [online]. January 27, 2022 [cit. 2022-11-23]. Dostupné z: https://blog.invgate.com/defining-your-it-security-perimeter

[4] RESONATE. *Web Server Overload: Causes, Signs, and Prevention* [online]. 2019 [cit. 2023-05-06]. Dostupné z: https://www.resonatenetworks.com/2019/01/11/web-server-overload-causes-signs-and-prevention/

[5] CROWDSTRIKE. *What is a botnet?* [online], 2022. [cit. 2022-10-31]. Dostupné z: https://www.crowdstrike.com/cybersecurity-101/botnets/

[6] KNOWBE4. *What is phishing* [online]. [cit. 2022-11-23]. Dostupné z: https://www.phishing.org/what-is-phishing

[7] DATADOME. *What is a botnet attack and how does it work?* [online]. [cit. 2022-11-23]. Dostupné z: https://datadome.co/learning-center/what-is-botnet-how-does-botnet-attack-work/

[8] VORMAYR, Gernot, Tanja ZSEBY a Joachim FABINI. Botnet Communication Patterns. *IEEE Communications surveys and tutorials* [online]. New York: IEEE, 2017, 19(4), 2768-2796 [cit. 2022-11-15]. ISSN 1553-877X. Dostupné z: https://doi.org/10.1109/COMST.2017.2749442

[9] SOURCEFORGE. *WASTE* [online]. 2003 [cit. 2022-12-08]. Dostupné z: https://waste.sourceforge.net

[10] LEFFEW, Kevin, 2019. *A Brief Overview of Kademlia, and its use in various decentralized platforms* [online]. [cit. 2022-12-10]. Dostupné z: http://bit.ly/3uIXcMo

[11] HUTCHINS, Marcus. *Peer-to-Peer Botnets for Beginners* [online]. 2013 [cit. 2022-11-21]. Dostupné z: https://www.malwaretech.com/2013/12/peer-to-peer-botnets-for-beginners.html

[12] KIME, Chad. *Complete Guide to the Types of DDoS Attacks* [online]. 2022 [cit. 2022-10-28]. Dostupné z: https://www.esecurityplanet.com/networks/types-of-ddos-attacks/

[13] CORERO. *Volumetric DDoS Attack* [online]. [cit. 2022-10-26]. Dostupné z: https://www.corero.com/resource-hub/volumetric-ddos-attack/

[14] CLOUDFLARE, INC. *Ping (ICMP) flood DDoS attack* [online]. [cit. 2022-10-27]. Dostupné z: https://bit.ly/3FlrJVs

[15] *DDOS UDP Flood Attack - explained and simulated.* In: Youtube [online]. 2022 [cit. 2022-10-27]. Dostupné z: https://www.youtube.com/watch?v=i5ezb54lEos.. Kanál uživatele Elia Halevy.

[16] MAHJABIN, Tasnuva, Yang XIAO, Guang SUN a Wangdong JIANG. *A survey of distributed denial-of-service attack, prevention, and mitigation techniques* [online]. 2017 [cit. 2023-04-21]. Dostupné z: https://doi.org/10.1177/1550147717741463

[17] SANJAY, Balaji RAJENDRAN a Pushparaj SHETTY. *DNS Amplification & DNS Tunneling Attacks Simulation, Detection and Mitigation Approaches*, IEEE [online]. 2020 [cit. 2023-04-18]. Dostupné z: https://doi.org/10.1109/ICICT48043.2020.9112413

[18] RUDMAN, L. a B. IRWIN. *Characterization and Analysis of NTP Amplification Based DDoS Attacks*, IEEE [online]. 2015 [cit. 2023-04-20]. Dostupné z: https://doi.org/10.1109/ISSA.2015.7335069

[19] PRASEED, Amid a P. Santhi THILAGAM. *DDoS Attacks at the Application Layer: Challenges and Research Perspectives for Safeguarding Web Applications*, IEE [online]. 2018 [cit. 2023-04-20]. Dostupné z: https://doi.org/10.1109/COMST.2018.2870658

[20] IMPERVA, Inc. *Slowloris* [online]. [cit. 2022-11-05]. Dostupné z: https://www.imperva.com/learn/ddos/slowloris/

[21] SUROTO, Suroto. *A Review of Defense Against Slow HTTP Attack* [online]. 2017 [cit. 2023-04-03]. Dostupné z: https://doi.org/10.30630/joiv.1.4.51

[22] A10 NETWORKS, INC. *What is a Protocol DDoS Attack?* [online]. [cit. 2022-10-28]. Dostupné z: http://bit.ly/3BqJbae

[23] FORTINET, Inc. *What Is a Ping of Death Attack?* [online]. [cit. 2023-04-21]. Dostupné z: https://www.fortinet.com/resources/cyberglossary/ping-of-death

[24] AGEYEV, Dmytro, Oleg BONDARENKO, Tamara RADIVILOVA a Walla ALFROUKH. Classification of existing virtualization methods used in telecommunication networks. In: *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)* [online]. Kyiv, Ukraine: IEEE, 2018 [cit. 2022-11-04]. ISBN 978-1-5386-5904-5. Dostupné z: https://doi.org/10.1109/DESSERT.2018.8409104

[25] *ROY, Siji. What is a platform?* [online]. 2021 [cit. 2022-11-05]. Dostupné z: https://www.webopedia.com/definitions/platform/

[26] JAVATPOINT. *Difference between Full Virtualization and Paravirtualization in Operating System* [online]. [cit. 2022-11-05]. Dostupné z: https://bit.ly/3MH7OFi

[27] IBM CLOUD EDUCATION. *Containerization* [online]. 2019 [cit. 2022-11-05]. Dostupné z: https://www.ibm.com/cz-en/cloud/learn/containerization

[28] CLOUDZERO. *The 11 Best Docker Alternatives In 2022* [online]. 2022 [cit. 2022-11-28]. Dostupné z: https://www.cloudzero.com/blog/docker-alternativesß

[29] DOCKER. *Docker documentation* [online]. [cit. 2022-11-28]. Dostupné z: https://docs.docker.com

[30] DYNATRACE LLC, 2023. *Dynatrace* [online]. [cit. 2023-04-02]. Dostupné z: https://www.dynatrace.com

[31] ZABBIX LLC. *Zabbix* [online]. 2023 [cit. 2023-04-08]. Dostupné z: https://www.zabbix.com

[32] GRAFANA LABS. *Grafana: The open observability platform* [online]. 2023 [cit. 2023-04-09]. Dostupné z: https://grafana.com

[33] PROMETHEUS. *Prometheus documentation* [online]. [cit. 2022-12-11]. Dostupné z: https://prometheus.io/docs/

[34] VISUAL PARADIGM. *What is Use case diagram* [online]. [cit. 2022-11-22]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

[35] CORBO, Anthony. What Is Python?. *Built In* [online]. 2022 [cit. 2023-05-07]. Dostupné z: https://builtin.com/software-engineering-perspectives/python

[36] JORDANA, A. What Is JavaScript?. *Hostinger* [online]. 2023 [cit. 2023-05-07]. Dostupné z: https://www.hostinger.com/tutorials/what-is-javascript

[37] ZOLA, Andrew. Bootstrap. *TechTarget* [online]. 2022 [cit. 2023-05-07]. Dostupné z: https://www.techtarget.com/whatis/definition/bootstrap

[38] IBM CORPORATION. *What is Ajax?* [online]. 2021 [cit. 2023-05-06]. Dostupné z: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=page-asynchronous-javascript-xml-ajax-overview

[39] PRIY, Surya. Introduction to SQLite. *GeeksforGeeks* [online]. 2023 [cit. 2023-05-08]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-sqlite/

[40] OFFSEC. Hping3. *Kali.org* [online]. 2022 [cit. 2023-05-10]. Dostupné z: https://www.kali.org/tools/hping3/

[41] OFFSEC. Slowhttptest. *Kali.org* [online]. 2022 [cit. 2023-05-10]. Dostupné z: https://www.kali.org/tools/slowhttptest/

[42] ANTIFEROV, Nicolai. PromQL / How to return 0 instead of 'no data'. *Medium.com* [online]. 2019 [cit. 2023-05-11]. Dostupné z: https://nklya.medium.com/promql-how-to-return-0-instead-of-no-data-9e49f7ccb80d

[43] THE APACHE SOFTWARE FOUNDATION. *Apache MPM worker* [online]. 2023 [cit. 2023-05-12]. Dostupné z: https://httpd.apache.org/docs/2.4/mod/worker.html

[44] BOOTSTRAP. *Bootstrap* [online]. 2023 [cit. 2023-05-12]. Dostupné z: https://getbootstrap.com

[45] ANDERSON, Jim. An Intro to Threading in Python. *Real Python* [online]. [cit. 2023-05-13]. Dostupné z: https://realpython.com/intro-to-python-threading/

Seznam symbolů a zkratek

AJAX Asynchronous JavaScript and XML – Technologie interaktivních webových aplikací

API Application Programming Interaface – Rozhraní pro programování aplikací

ARM Advanced RISC Machines – Architektura procesorů

C&C Command–and–Control – Komunikační středisko útočníka

CLI Command Line Interface – Příkazový řádek

CPU Central Processing Unit – Centrální procesorová jednotka

CSS Cascading Style Sheets – Kaskádové styly

DDoS Distributed Denial of Service – Distribuovaný útok na dostupnost služby

DGA Domain Generation Algorithm – Algoritmus generování domén

DNS Domain Name System – Protokol pro překlad adres

DoS Denial of Service – Útok na dostupnost služby

HTML HyperText Markup Language – Značkovací jazyk pro tvorbu webových stránek

HTTP Hypertext Transfer Protocol – Internetový protokol

HTTPS Hypertext Transfer Protocol Secure – Zabezpečený internetový protokol

HW Hardware – Fyzické vybavení počítače

IT Information Technology – Informační technologie

IoT Internet of Things – Internet věcí

ICMP Internet Control Message Protocol – Protokol pro odesílání informací o službách

IRC Internet Relay Chat – Protokol pro textovou komunikaci

JSON JavaScript Object Notation – Datový formát

MAC	Media Access Control – Identifikátor síťového zařízení
MacOS	Macintosh Operating System – Operační systém společnosti Apple
NAT	Network Address Translation – Překlad síťových adres
NOC	Network Operational Center – Síťové operační středisko
NTP	Network Time Protocol – Protokol pro synchronizaci času
RAM	Random Access Memory – Operační paměť
REST	Representational State Transfer – Architektura rozhraní
RISC	Reduced Instruction Set Computer – Architektura jednoduchých mikroprocesorů
OS	Operating System – Operační systém
P2P	Peer–to–Peer – Přímá komunikace mezi stanicemi
SOC	Security Operational Center – Bezpečnostní operační středisko
TCP	Transmission Control Protocol – Protokol transportní vrstvy
UDP	User Datagram Protocol – Protokol transportní vrstvy
URL	Uniform Resource Locator – Jednotný lokátor zdroje
YAML	YAML Ain't Markup Language – Jazyk pro serializaci dat

A Obsah elektronické přílohy

V příloze této práce se nachází všechny soubory potřebné ke spuštění vytvořeného nástroje pro testování vybraných DDoS útoků. Tytéž soubory lze nalézt v online repozitáři `https://github.com/kralluk/DDoS_testbed/`, v tomto repozitáři je také přítomen návod na instalaci a spuštění aplikace.

```
/ ..... kořenový adresář přiloženého archivu
├── webapp ..... zastřešující adresář webové aplikace
│   ├── app. ..... adresář s aplikací
│   │   ├── static ..... adresář se statickými soubory
│   │   │   ├── bootstrap ..... adresář s Bootstrap soubory
│   │   │   │   ├── css ..... adresář s Bootstrap kaskádovými styly
│   │   │   │   └── js ..... adresář s Bootstrap JavaScript soubory
│   │   │   ├── attacks.js
│   │   │   ├── style.css
│   │   │   ├── victim.js
│   │   │   └── main.js
│   │   ├── templates ..... adresář se šablonami
│   │   │   ├── index.html
│   │   │   └── show_botnet.html
│   │   ├── __init__.py
│   │   ├── attacks.py
│   │   ├── bot_management.py
│   │   ├── db.py
│   │   ├── initialize.py
│   │   ├── resource_utils.py
│   │   ├── routes.py
│   │   ├── setings.py
│   │   └── victim.py
│   ├── configurations ..... adresář s konfiguracemi
│   │   ├── apache_configs ..... adresář s konfiguračními soubory Apache
│   │   │   ├── httpd_2.2.34.conf
│   │   │   ├── httpd_2.4.29.conf
│   │   │   └── httpd_2.4.56.conf
│   │   ├── grafana_provisioning ..... adresář s konfiguračními soubory Grafany
│   │   │   ├── dashboards ..... adresář s dashboardy
│   │   │   │   ├── ddos.json
│   │   │   │   └── default.yaml
│   │   │   └── datasources ..... adresář s konfiguračním souborem datových zdrojů
│   │   │       └── datasources.yaml
│   │   ├── docker-compose.yml
│   │   └── prometheus.yml
│   ├── app.py
│   └── requirements.txt
└── README.md
```