

# TFG Modbus

Antonio Pérez Sánchez

Miguel Oleo Blanco



Tabla de contenido

[TFG Modbus 1](#_Toc63612910)

[Modbus 3](#_Toc63612911)

[Introducción 3](#_Toc63612912)

[Ataques 3](#_Toc63612913)

[Impersonating del Máster 7](#_Toc63612914)

[Man in the Middle 9](#_Toc63612915)

[Packet replay 12](#_Toc63612916)

[Bibliografía 12](#_Toc63612917)

# Modbus

## Introducción

Modbus es otro protocolo de comunicación industrial. Este protocolo fue creado a finales de los años 70 por el actual Schneider Electric para comunicación entre máquinas y PLCs. No es un protocolo moderno, por lo que se basa en los protocolos existentes en aquella época, como son el RS232, RS485 y RS422. Debido a esto, es un protocolo muy simple y fácil de implementar. Se basa en una arquitectura Maestro y Esclavo. En este tipo de arquitectura, el control de la comunicación la lleva a cabo el maestro. Por ejemplo, en ningún caso el esclavo proporcionará información sin que un maestro se lo solicite.

Debido a su simpleza y a su arquitectura, es un protocolo que cogió fuerza para poder mandar la información entre los sensores y los distintos equipos para poder monitorear los dispositivos de campo como, por ejemplo, en sistemas SCADA.

Hay varias versiones de Modbus, según las necesidades y según el canal sobre el cual quiere implementarse. Las dos versiones más empleadas son Modbus TCP para comunicaciones sobre Ethernet y Modbus RTU para comunicaciones serie. En este trabajo nos centraremos en el TCP, ya que es más actual.

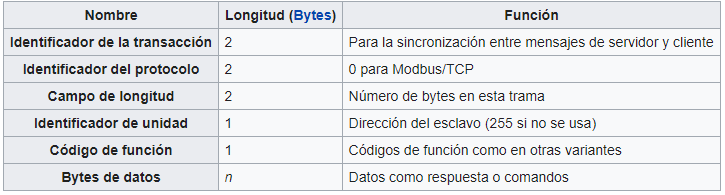
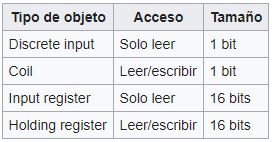
 El funcionamiento del protocolo es idéntico independientemente de la versión empleada. Modbus se basa en registros, en los cuales los esclavos y maestros escriben y leen. En el caso de los esclavos, como ya he citado anteriormente, algunas de las acciones solo las podrá llevar a cabo cuando el maestro correspondiente se lo indique. Cada esclavo tiene sus registros y su indicador (ID) con el cual el maestro le mandará peticiones, junto al número o números de registros a los que quiere acceder. Los registros son de varios tipos, con distintas políticas de acceso y tamaño. A continuación, se muestran estos datos en la tabla 1 y el formato de la trama TCP en la tabla 2:

Tabla 1: Tipos de registros

Tabla 2: Trama Modbus TCP

## Ataques

Antes de realizar un ataque sobre Modbus, hay que investigar qué medidas de seguridad implementa el protocolo. En este caso, Modbus es muy simple ya que no incorpora ningún tipo de cifrado (manda los datos en texto plano) y no incorpora grandes medidas de seguridad. En concreto Modbus lo único que aporta es un número de transacción que sigue una secuencia, el resto de la seguridad que incorpora es la propia que tiene por estar basado en TCP.

Para realizar los ataques vamos a emplear un simulador de Modbus. En concreto para el cliente voy a emplear ModbusPal que es un cliente basado en java, lo cual nos aporta la posibilidad de ejecutarlo en distintos sistemas operativos indistintamente. Para descargarlo en la referencia [3]. Con esta aplicación podemos crear esclavos y sus registros. También nos permite modificar manualmente los registros, lo cual puede venir bien para llevar a cabo pruebas. Este programa se muestra en la figura 1. Para simular el master, utilizaremos QModMaster un programa que nos permite conectarnos al ModbusPal para leer y escribir datos. Este programa se muestra en la figura 2. A continuación se muestra el proceso de configuración.

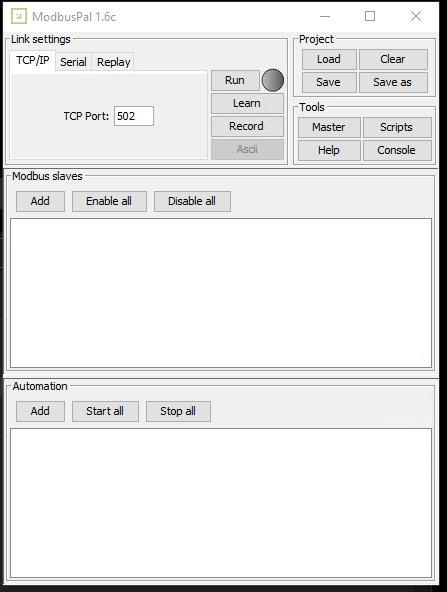
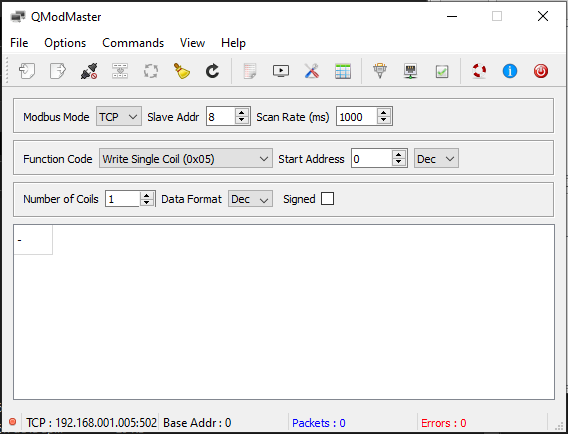


Figura 2: QModMaster

Figura 1: ModbusPal

El proceso de conexión es muy simple. Primero crearemos un esclavo en el ModbusPal en el botón Add del apartado Modbus slaves. Le asignaremos un ID y un nombre. Una vez hecho esto, podemos crear un registro y poner valores. Para este ejemplo, en los registros 1 y 10, he puesto el valor 1 para que veamos que funciona todo.

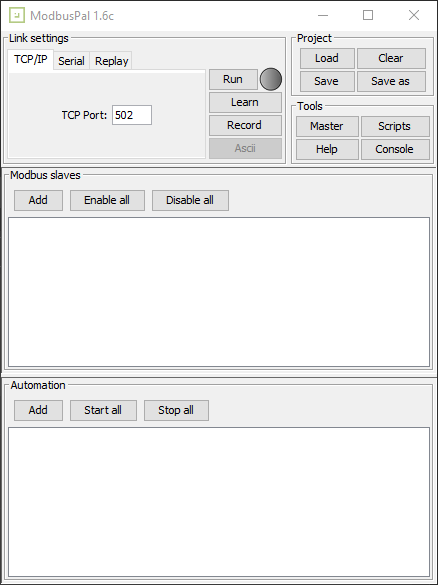
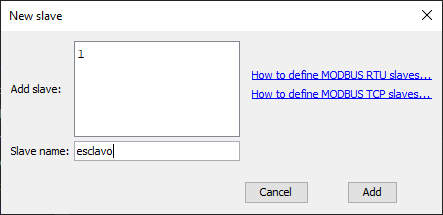


Figura 4: Asignar ID y nombre al esclavo

Figura 3: Añadir un esclavo

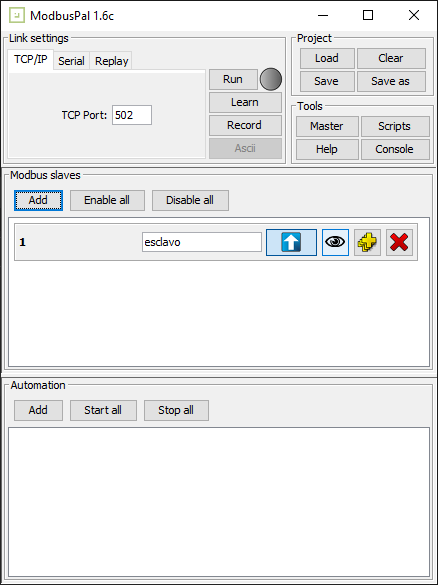
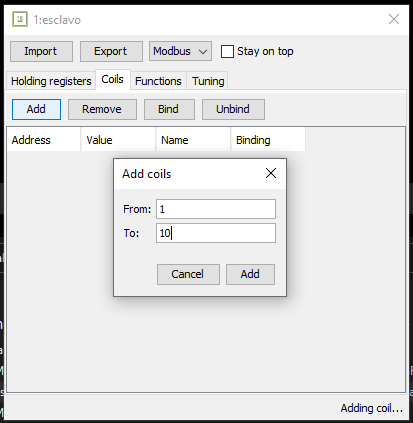
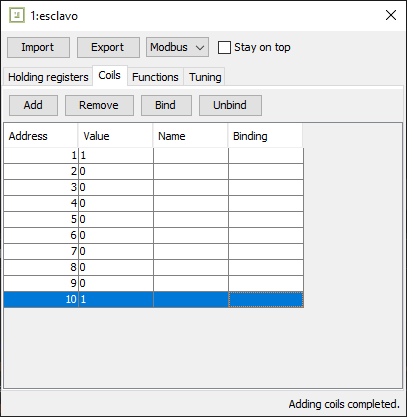


Figura 5: Acceder a los registros

Figura 6: Tabla de coils y valores

Figura 6: Acceder a coils y crear tabla

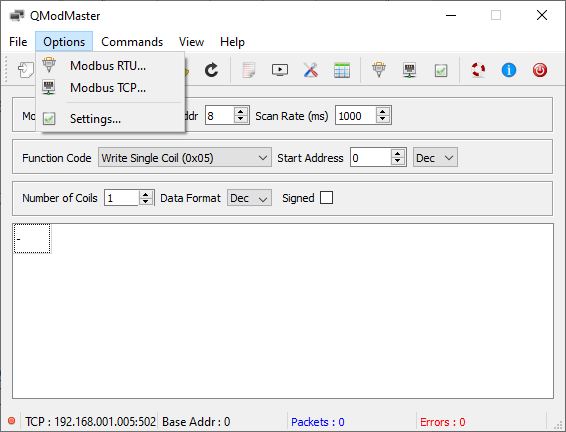
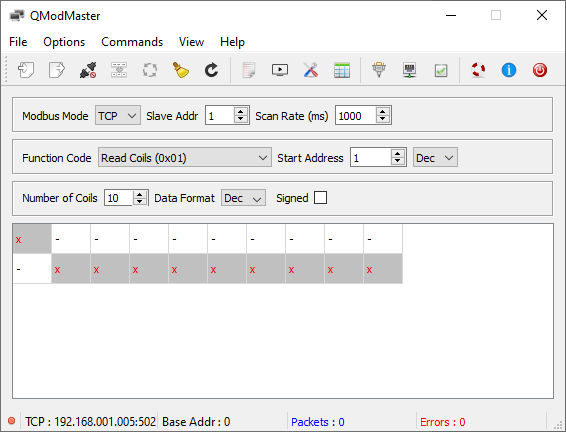
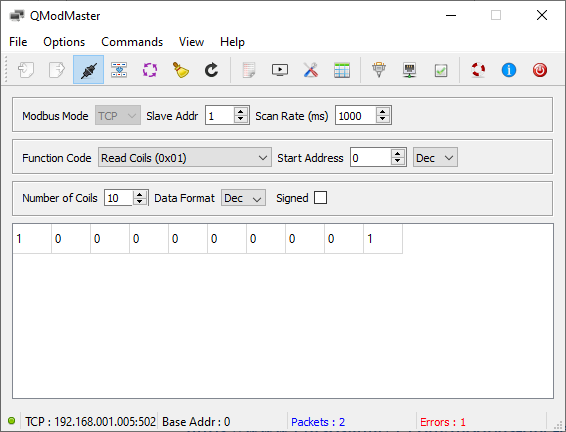
 Una vez hecho este proceso, solo hace fala ir a la pantalla principal de ModbusPal y darle a Run. Ahora procedemos a configurar el QModMaster para que se conecte con el ModbusPal. Lo primero es configurar la dirección del slave al que nos queremos conectar, para ello, vamos a opciones y a modbus tcp. Ahí introducimos la ip y el puerto del ModbusPal. Una vez hecho esto, debemos de introducir el id del esclavo y nos conectamos. QModMaster nos ofrece varias operaciones, en este proyecto nos centraremos en la de escritura y lectura de Coils. Para esto también hay que indicarle el número o números del coils a los que queremos acceder para realizar la operación indicada.

Figura 9: Resultado de la lectura

Figura 8: Configuración Lectura de Coils

Figura 7: Conexión QModMaster

Al leer o escribir datos, se comunican los dos programas utilizando Modbus. Este tráfico lo podemos analizar usando Wireshark y veremos que es un protocolo bastante simple y confirmamos que no está cifrado. Un descubrimiento importante que se aprecia en este punto es que el checksum no se valida en ningún momento. En este punto también vemos los distintos mecanismos de seguridad que implementa. Al estar basado en TCP, se ve que cada petición se valida por un ACK, por lo que hay unas secuencias TCP que sigue. También se ve que sigue una secuencia de transacción. En la figura 10 se puede ver tráfico de lectura y de escritura y en la figura 11 se ve los datos que añade Modbus sobre TCP en una query de escritura de un valor 0 en el primer registro, los cuales se aprecia que se transmiten en texto plano.

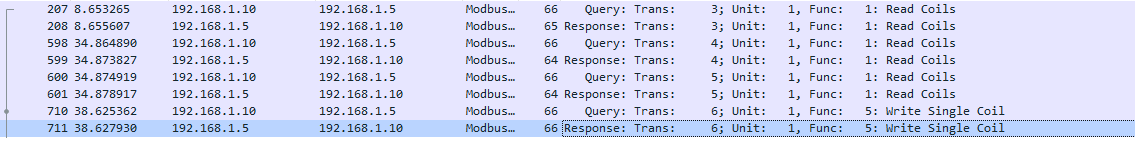


Figura 10: Tráfico Modbus

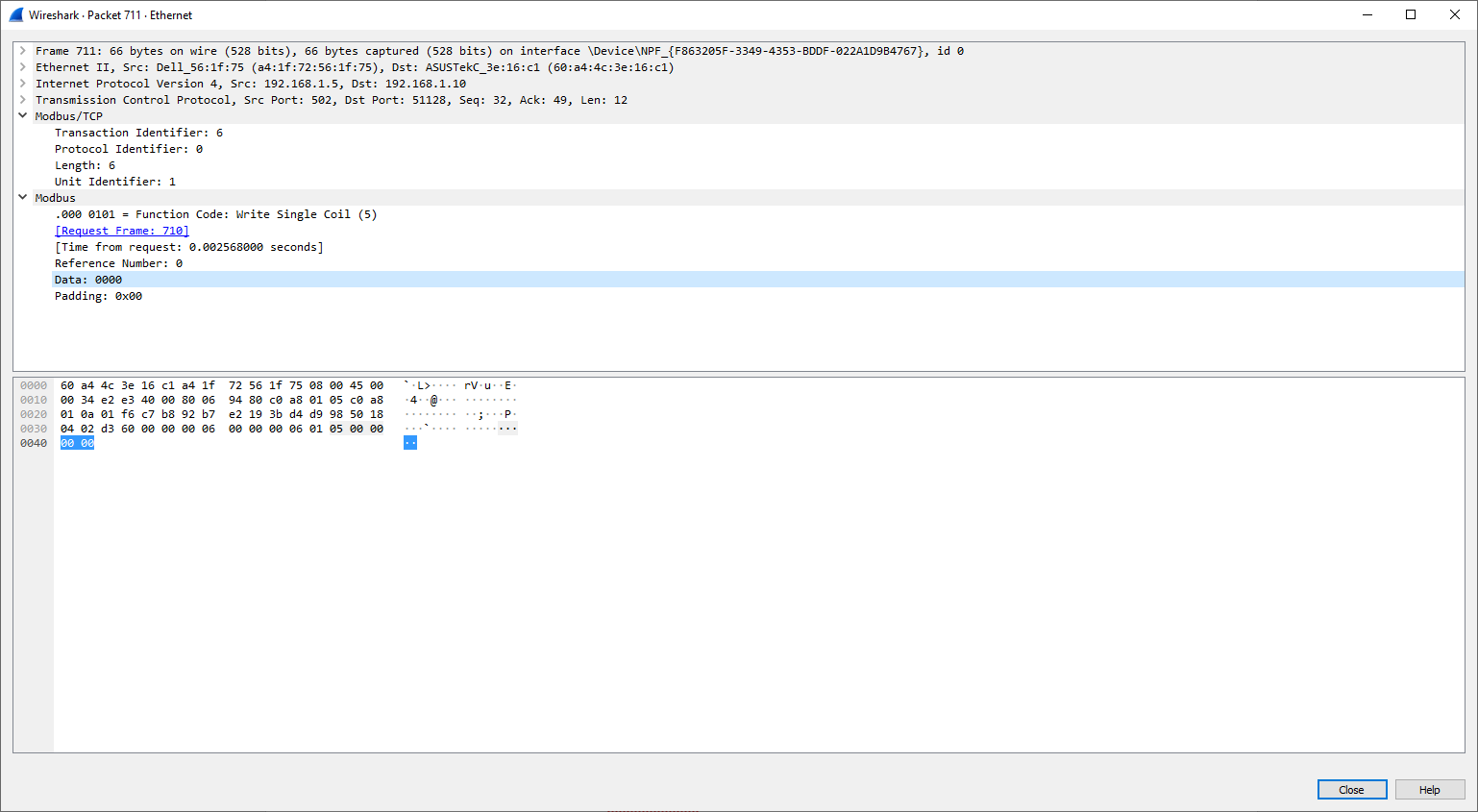


Figura 11: Datos de Modbus en query de escritura

Para realizar estos ataques voy a emplear 3 equipos para tener una topología de red más realista. Por un lado, tenemos dos equipos Windows (pueden ser también MAC o Linux). Uno de ellos es el que contiene el ModbusPal simulando el esclavo y el otro ejecuta el QModMaster que simula el maestro. Estos dos equipos simulan las estaciones de ingeniería presentes en una red industrial. Para realizar los ataques yo empleo una máquina con Kali Linux (una Raspberry pi 4), esta es la que simula el atacante y la que correrá las distintas herramientas para llevar a cabo los ataques. A continuación, en la figura 12 se muestra la topología de red con todos los datos importantes.

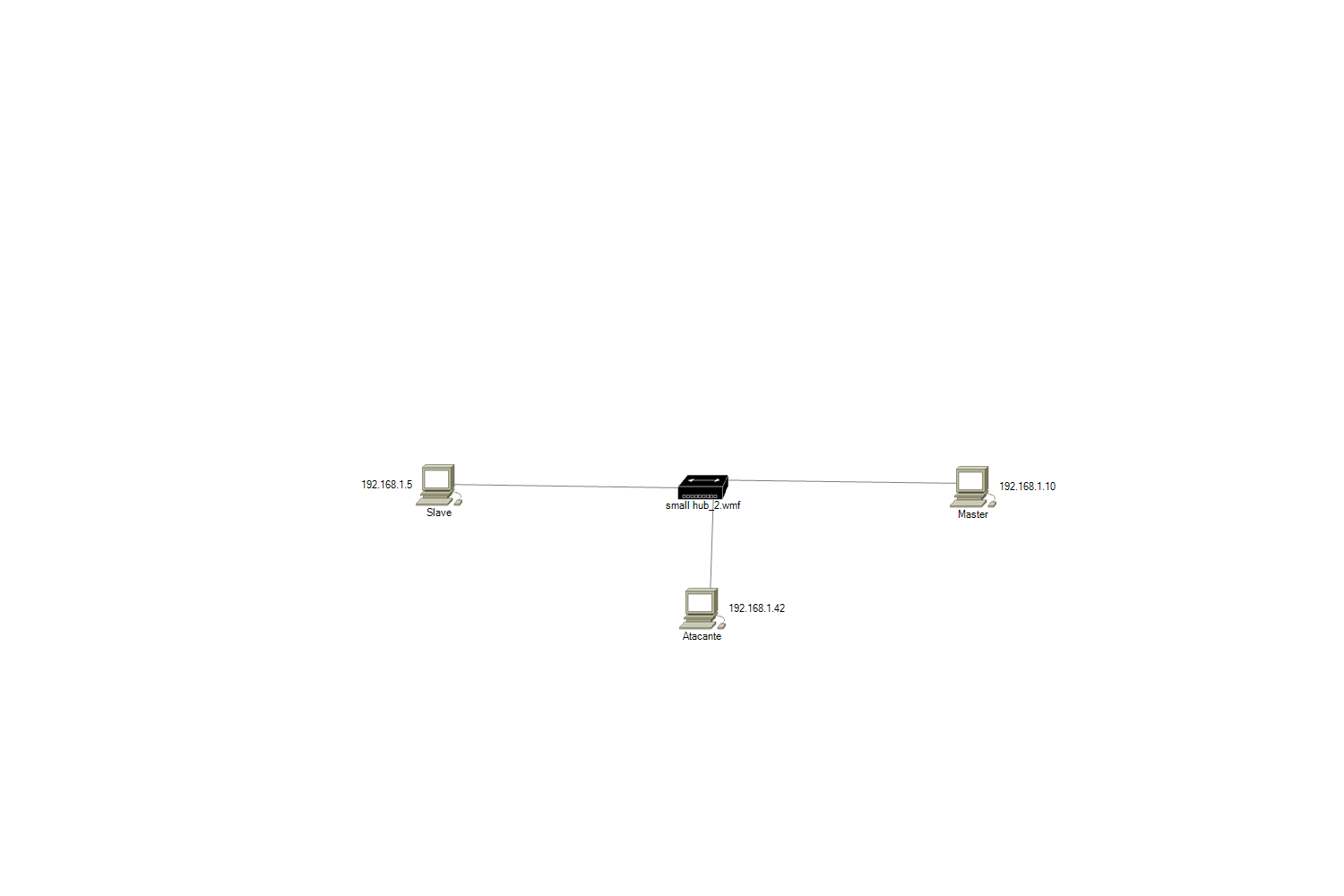


Figura 12: Topología de red

También es importante conocer las MACs de los distintos equipos, ya que nos serán útiles para comprobar que los ataques se están realizando correctamente. En la tabla 3 se muestra las MACs y las IPs.

Tabla 3: Direcciones de las estaciones

|  |  |  |
| --- | --- | --- |
| Máquina | Dirección IP | Dirección MAC |
| Máster | 192.168.1.10 | 60-A4-4C-3E-16-C1 |
| Slave | 192.168.1.5 | A4-1F-72-56-1F-75 |
| Atacante | 192.168.1.42 | dc:a6:32:67:24:38 |

### Impersonating del Máster

Los ataques de Impersonating (imitar en español) son los que, con un software específico, podemos hacernos pasar por el sistema que imita. En este caso, utilizaremos la herramienta de Metaesploit que está incluido en Kali Linux para utilizar unos paquetes que nos proporciona listos para realizar este ataque. Para arrancar este framework basta con escribir el siguiente comando en el terminal de Linux: *sudo msfconsole.* Esperamos a que se abra la interfaz de terminal y hay que continuar buscando los módulos que vamos a emplear. Para buscarlos escribimos *search modbus* y se nos muestran los distintos paquetes preparados relacionados con modbus. Como nosotros partimos del hecho de que conocemos la topología de red (quien es el esclavo y quien el máster) no empleamos ningún paquete de scanner. Esta fase es fácil de hacer con Wireshark una vez que el atacante se conecta a la red y arranca el programa. Solo hay que filtrar por protocolo modbus y vemos quien es el que escucha en el puerto 502 que es el esclavo y el que escribe a ese puerto es el máster. Por ello, vamos a seleccionar el paquete de modbusclient. Este paquete se puede seleccionar escribiendo *use 5* o con *use auxiliary/scanner/scada/modbusclient.* Este proceso se muestra en la figura 13. Una vez seleccionado, hay que ver que parámetros debemos establecer antes de ejecutar el paquete. Para ver estos parámetros, usamos el comando *show options*, como se muestra en la figura 14. Por último, hay que establecer todos los parámetros y ejecutarlo. Este último paso se muestra en la figura 15, que está establecido para leer el dato 2 de la coil del esclavo 1. Para establecer los valores se hace poniendo *set [opción] [valor]* y se ejecuta con *run.*

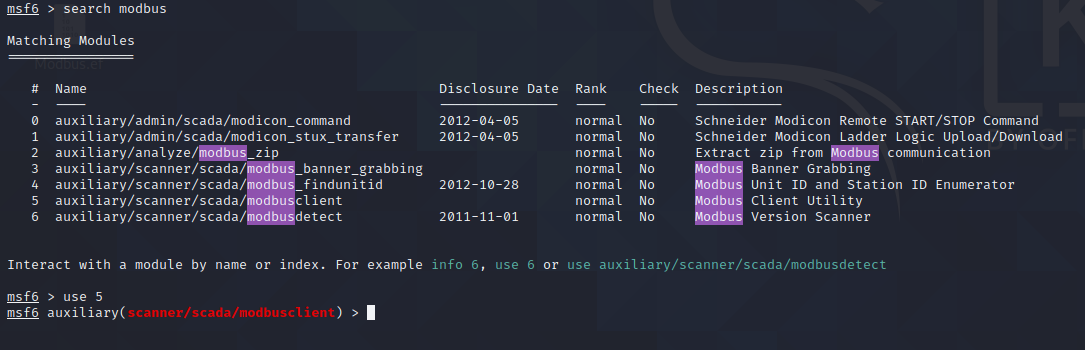
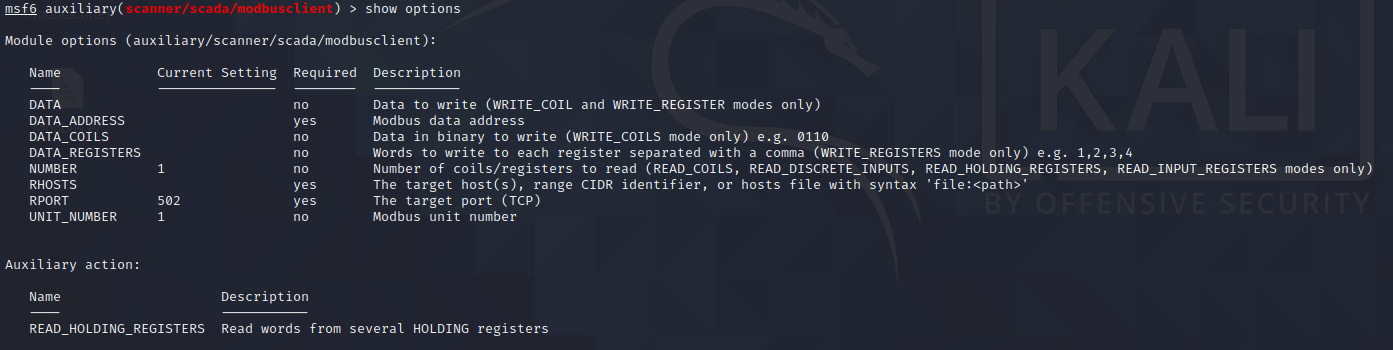


Figura 13: Busqueda modbus en msfconsole



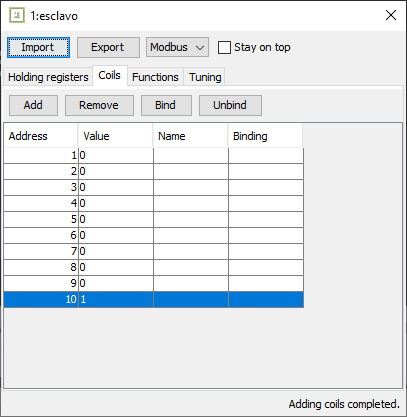
Figura 14: show options de msfconsole

Figura 16: Tabla de coils

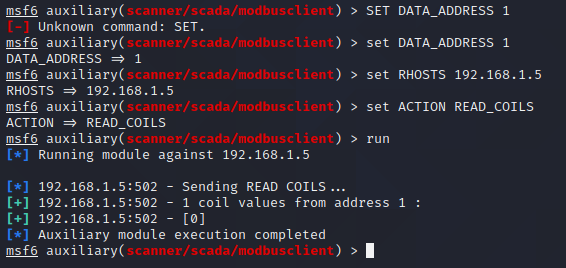


Figura 15: Establecimiento de parámetros y ejecución

Como se puede observar en la figura 15, el programa nos devuelve el valor del coil 2 del slave número 1. El valor que nos devuelve es correcto como se puede comprobar en la figura 16. Ahora vamos a emplear el mismo paquete, pero para escribir un valor. En la figura 17 se muestra la configuración y el resultado del run.

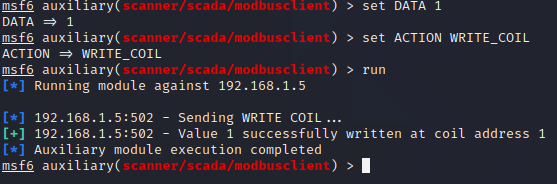
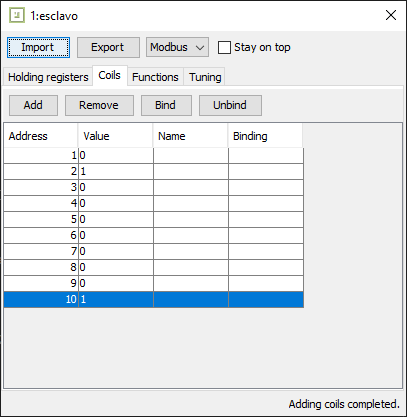


Figura 18: Resultadod e escritura

Figura 17: Establecimiento de parámetros y escritura

Como se puede ver en la figura 17 y 18, la escritula del valor 1 en el registro 2 se ha realizado con éxito. Con esto queda demostrado lo fácil que es imitar al máster. Si miramos el tráfico que hemos generado (me centro en este último de escritura), podemos confirmar con el wireshark que todo es correcto, ya que la dirección de origen es la de la máquina del atacante, como se aprecia en la figura 19.

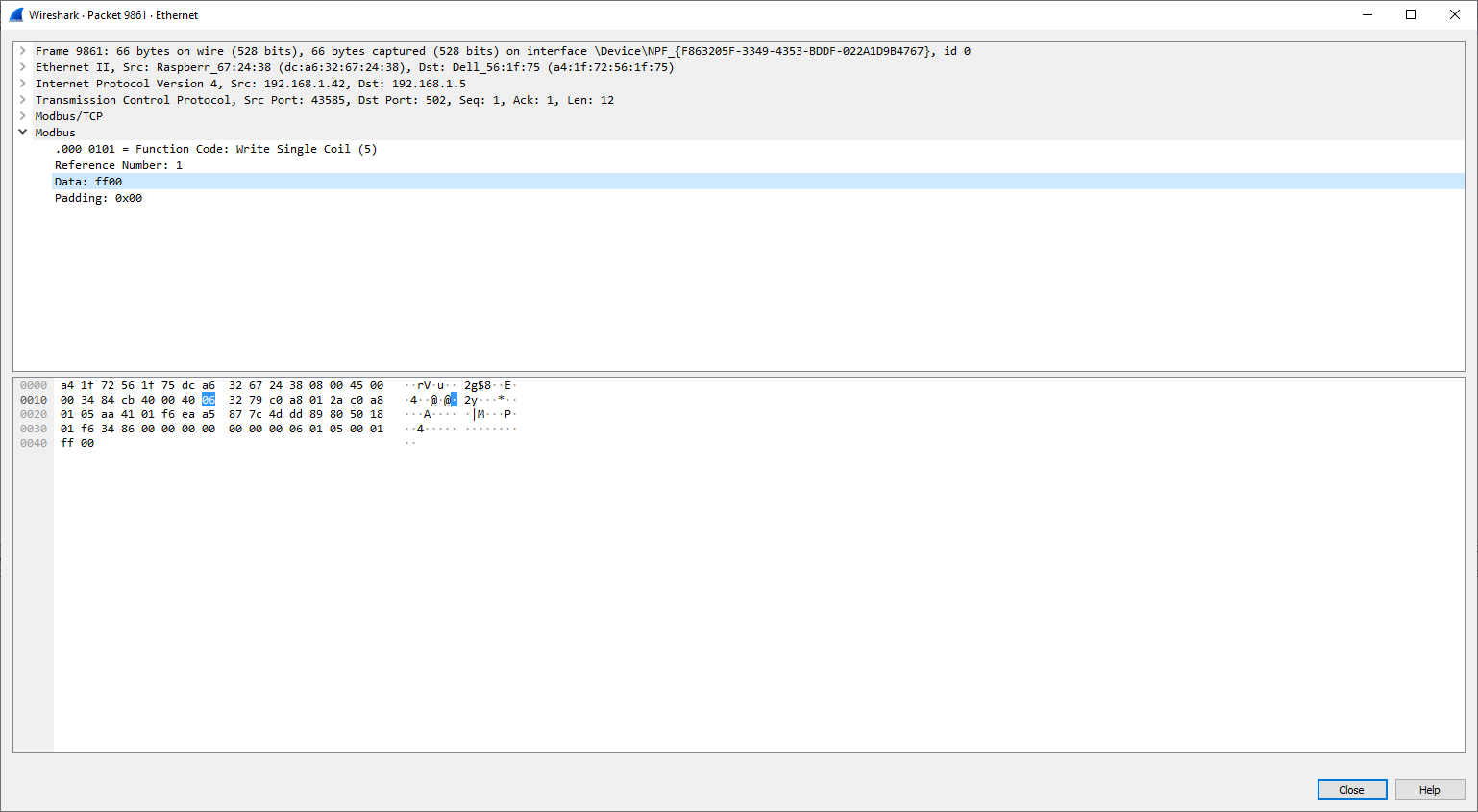


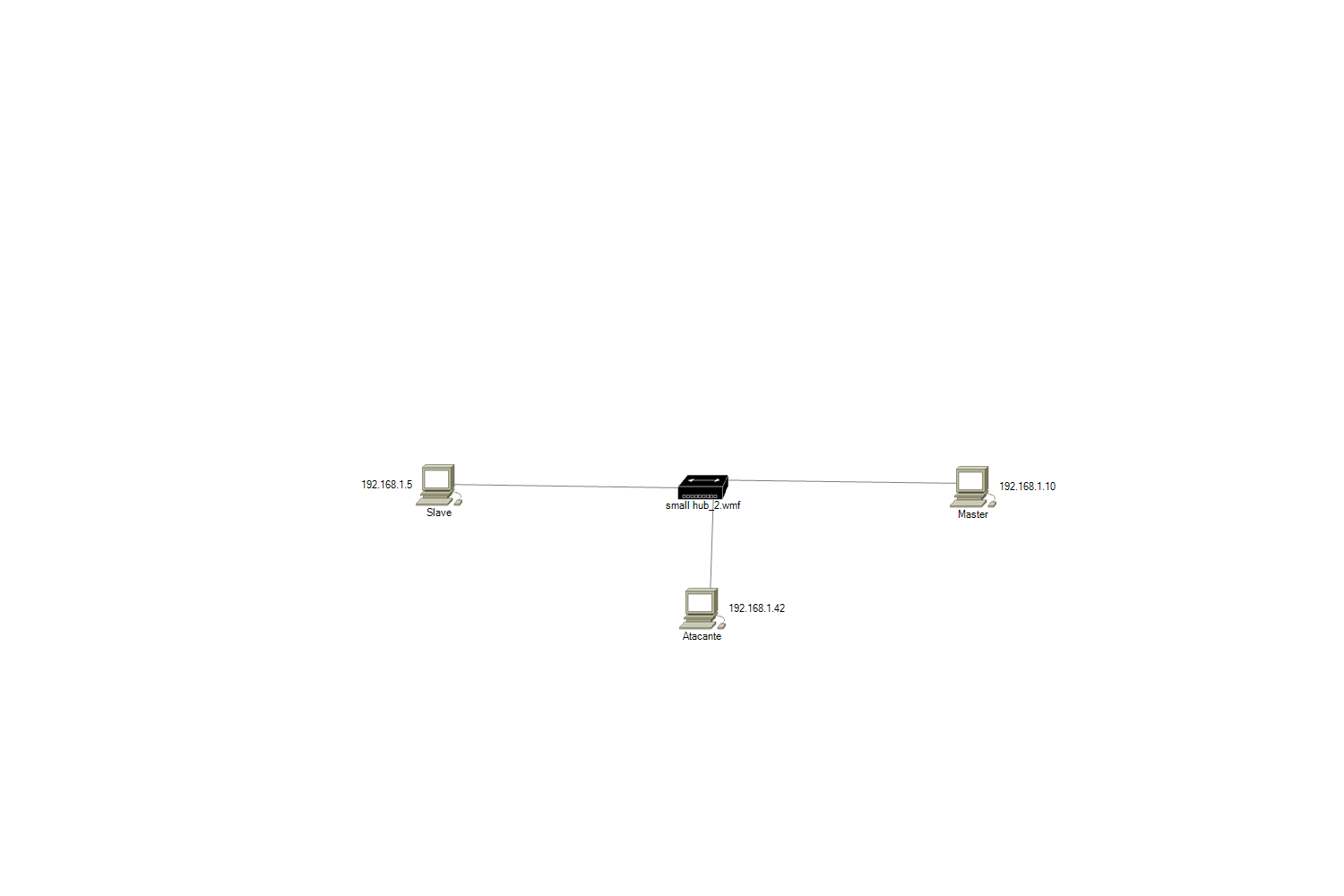
Figura 19: Tráfico del atacante

Este programa es muy fácil de usar, pero si queremos realizar ataques más complejos o automatizarlos, podemos también emplear una librería de Python y picar nuestro código. Esta librería se llama PyModbus y es muy fácil de usar. En la referencia [5] dejo la documentación sobre este paquete.

### Man in the Middle

Este tipo de ataques son de los más conocidos, ya que se puede aplicar a un motón de ámbitos. Este ataque se basa en el concepto de que hay una comunicación entre dos partes (en nuestro caso, el máster y el esclavo) y una tercera persona (el atacante) se pone entre medias de los dos. El atacante es capaz de controlar el flujo de toda la información y modificarla sin que las partes se enteren (a no ser que lo comprueben). En nuestro protocolo, se va a implementar este Man in the Middle que alterará el flujo normal de comunicación y aprovecharemos esta alteración para implementar un filtro que nos modifique automáticamente los datos que se envían.

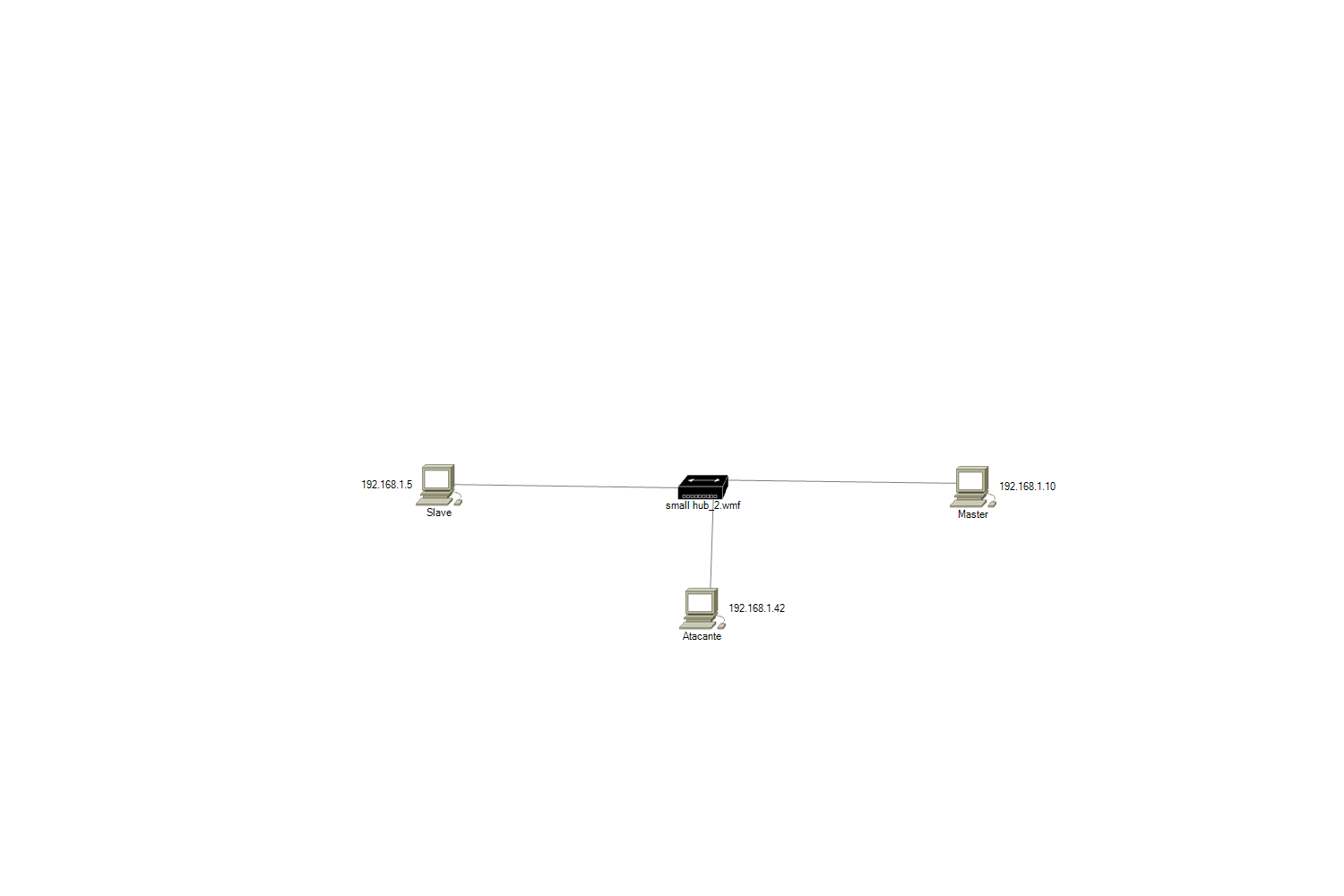
El flujo normal de la información es la que genera el Máster, va directamente al esclavo y este le responde directamente al máster con un ACK confirmando la petición. Al implementar este ataque, el máster pensará que el slave es el ordenador del atacante, por lo que le envía al equipo del atacante la petición. El equipo del atacante puede o bien, no modificar nada y mandárselo al slave real (con esto podría analizar el tráfico) o aprovechar y modificar los datos que se envían. Este último es el que se implementará en este proyecto. Una vez modificado el paquete, se lo envía al esclavo. El ack que genera el esclavo, seguirá también este mismo camino. A continuación, en la figura 20 y 21, se muestra cómo es el tráfico normal y el tráfico con el MiTM implementado.



ACK

Petición

Figura 20: Flujo de tráfico normal



ACK 1.2

ACK 1.1

Petición 1.2

Petición 1.1

Figura 21: Flujo de tráfico con MiTM

Para realizarlo, vamos a emplear varios programas ya citados, como pueden ser ModbusPal, QModMaster y Wireshark, y otro nuevo que es Ettercap, que ya viene incluido en Kali Linux. Este programa es el que nos va a permitir implementar el MiTM. El funcionamiento de este programa y su implementación es muy sencillo. Al arrancarlo, escaneará el tráfico y nos mostrará una lista de las IPs encontradas en la red. Llegados a este punto, añadiremos la IP del esclavo como target 1 y la IP del máster como target 2. Ahora debemos activar la opción de arp poisoning. El arp poisoning es un ataque que solo se puede realizar en redes locales (LANs) y se basa en enviar paquetes de ARP (sirven para obtener la IP de una MAC) a un Gateway y modificar la asociación MAC e ip. En nuestro ataque esto lo que hará es que el master cuando realiza una petición al slave, en vez de tener en su tabla que esa IP corresponde a la del slave, tendrá que se corresponde al del atacante.

Llegados a este punto, el man in the middle está implementado, pero vamos a ir un paso más allá. Vamos a crear un filtro en el ettercap que intercepte este tráfico que proviene del protocolo Modbus y vamos a indicarle que nos invierta los valores. Para ello hay que escribir un código en un archivo de texto y luego compilarlo. Para más información sobre este paso, consultar la página de ettercap en la referencia [6]. En la figura 25 se puede ver el. El archivo del filtro no está sujeto a una extensión (yo usare filter), pero a la hora de compilarlo, si debemos decirle que el output debe tener extensión ef. Para compilar se emplea el siguiente comando en el terminal:

*etterfilter [input\_file] -o [output\_name].ef*

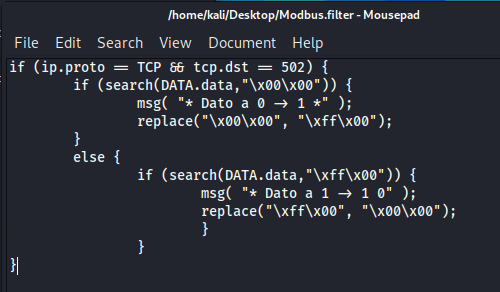
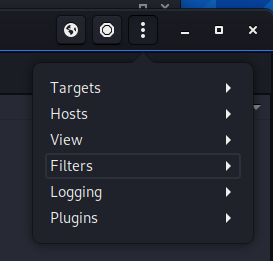
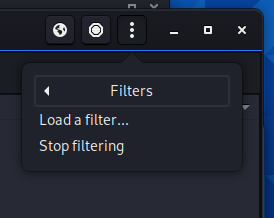
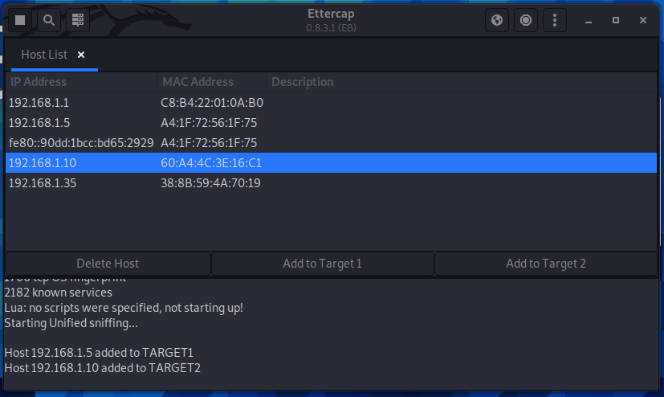
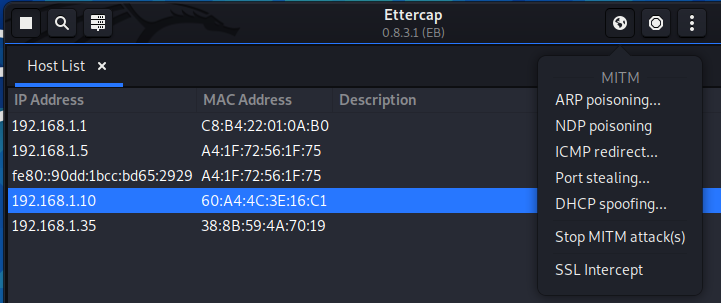
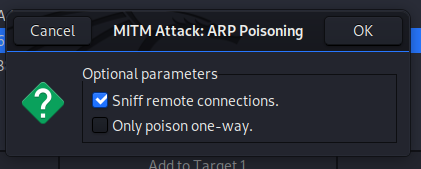
**** Una vez creado y compilado el filtro, solo debemos añadirlo a ettercap y generar tráfico de escritura en QModMaster mientras analizamos el tráfico con Wireshark. El paso de analizar el tráfico es muy importante, ya que podremos ver si el proceso de MiTM se está realizando correctamente. Si todo funciona correctamente, deberíamos de ver en la parte inferior de ettercap mensajes cada vez que escribimos algún valor desde el máster. En este ejemplo se va a emplear escritura del coil 2 del esclavo 1. A continuación, se muestran capturas de wireshark del proceso del MiTM. Se puede observar que la petición de escritura del valor 1 se manda del máster al equipo del atacante y del atacante se manda la misma petición al esclavo, pero con el valor modificado (pasa a ser 0). Es importante destacar que la comunicación de escritura entre el atacante y el esclavo se realiza en TCP, no en modbus (pero el contenido del paquete es el mismo). En la figura 28 y 29 se aprecia como se han cambiado las direcciones para que el tráfico entre maestro y esclavo pase por el equipo del atacante. También se observa cómo se ha modificado el valor a escribir. Para el ACK que nos devuelve el esclavo, se sigue el mismo proceso.

Figura 27: Cargar el filtro (2)

Figura 26: Cargar el filtro (1)

Figura 24: Activar ARP poisoning (2)

Figura 25: Filtro ettercap

Figura 23: Activar ARP poisoning (1)

Figura 22: Añadir targets

Una característica importante de este tipo de ataques es que en el software del máster no salta ningún error. A pesar de esto, es fácil darnos cuenta de que se está produciendo un ataque de estas características. Si no filtramos el tráfico en Wireshark para que sólo nos muestre Modbus, se aprecian paquetes de retransmisiones que el programa nos indican que no son normales. Por otro lado, si filtramos por protocolo Modbus, todo parece normal, aparece una petición y un ACK, pero si vemos en campo de Data de estos dos, apreciamos como el máster manda un 1 y el ACK confirma un 0. Viendo esta anomalía, el ingeniero encargado podría darse cuenta de que hay algún fallo en la red o algún software que está provocando este comportamiento.

Este ataque puede ser más sofisticado y hacer que el ACK que devuelve, con otro filtro, restaurar el mismo dato que solicitó el máster.

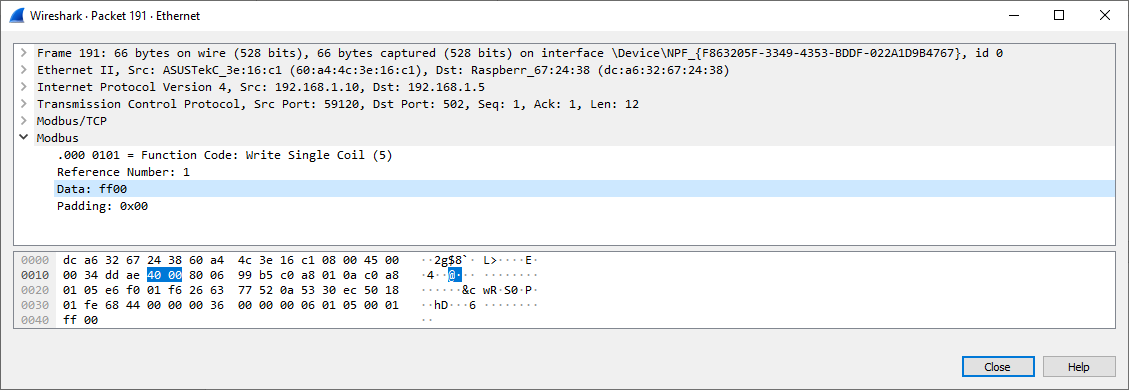


Figura 28: Petición de escritura interceptada

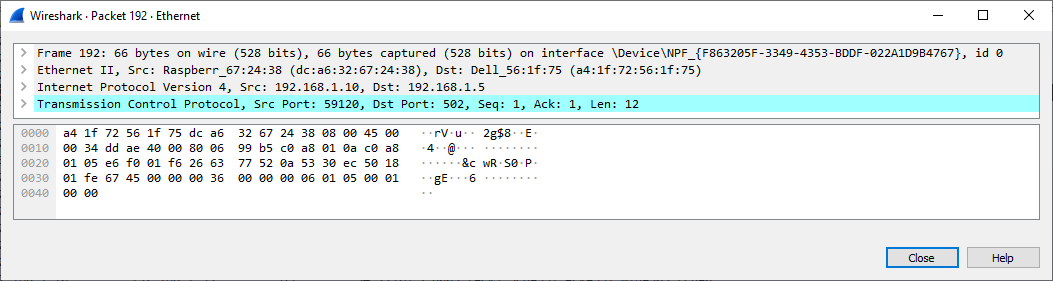


Figura 29: Petición de escritura interceptada y modificada

### Packet replay

Como ya se ha citado en anteriores protocolos, este ataque se basa en capturar tráfico durante el funcionamiento normal de la comunicación, editar el paquete y reenviarlo. Para poder editar y enviar estos paquetes, emplearé Scapy el cual es una herramienta dedicado exclusivamente a captación y edición de paquetes basado en Python.

Antes de nada, lo primero es capturar un paquete que nos interese para el ataque. Por ejemplo, para Modbus, es muy interesante modificar un paquete de escritura, ya que podemos cambiar la dirección del coil o el dato a escribir. Una parte clave de este proceso y muy interesante para el desarrollo del laboratorio de ciberseguridad, es descomponer el paquete a mano para identificar los distintos campos y más tarde recomponerlo. Para ello, hacemos click derecho sobre el paquete en wireshark y lo copiamos como Hex Stream. Para este ejemplo usaré el siguiente stream:

*a41f72561f7560a44c3e16c10800450000347531400080060233c0a8010ac0a80105c81c01f6214eca35ac7f79de501802004e57000000080000000601050001ff00*

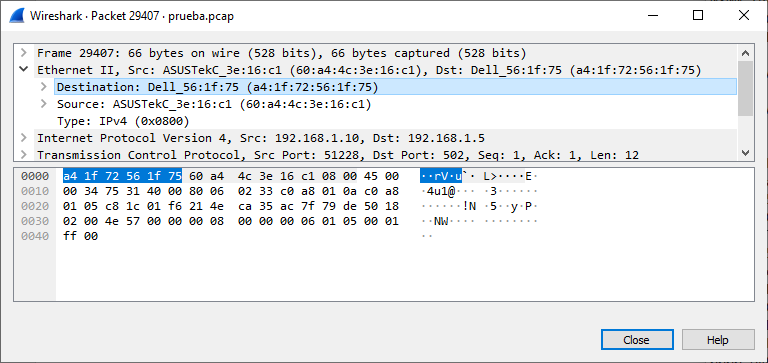
 Una vez tenemos el paquete como un stream en hexadecimal, lo recomendable es pegarlo en un editor de texto para empezar a su descomposición. El procedimiento es muy simple, pero hay realizarlo atentamente para fijarse que no cogemos un carácter que pertenezca a otro campo. Como ahora están todos los datos de todos los elementos del paquete seguidos, lo más recomendable es ir separándolo según encontramos un campo. Para ir realizando este proceso, nos vamos a ir apoyando en la información que nos proporciona Wireshark. Cuando seleccionamos una capa del protocolo, o un campo, se resalta en azul los datos hexadecimales correspondientes al campo seleccionado. Por ejemplo, el primer campo que encontramos es la MAC de destino en hexadecimal:

Figura 30: Mac en hexadecimal

En el la parte inferior nos indica wireshark los datos de esta dirección MAC, por lo que en nuestro stream podemos identificar estos datos y separalos:

*a41f72561f7560a44c3e16c10800450000347531400080060233c0a8010ac0a80105c81c01f6214eca35ac7f79de501802004e57000000080000000601050001ff00*

Este proceso lo vamos a repetir a lo largo de todos los campos que encontremos en el paquete y teniendo especial cuidado en aquellos campos que según wireshark parecen que comparten datos (solo subraya en azul cada 2 caracteres hexadecimales y si un campo acaba solo coge 1, subraya los dos para ese campo y para el siguiente). Una vez terminado este proceso, debería quedar algo parecido a lo siguiente:

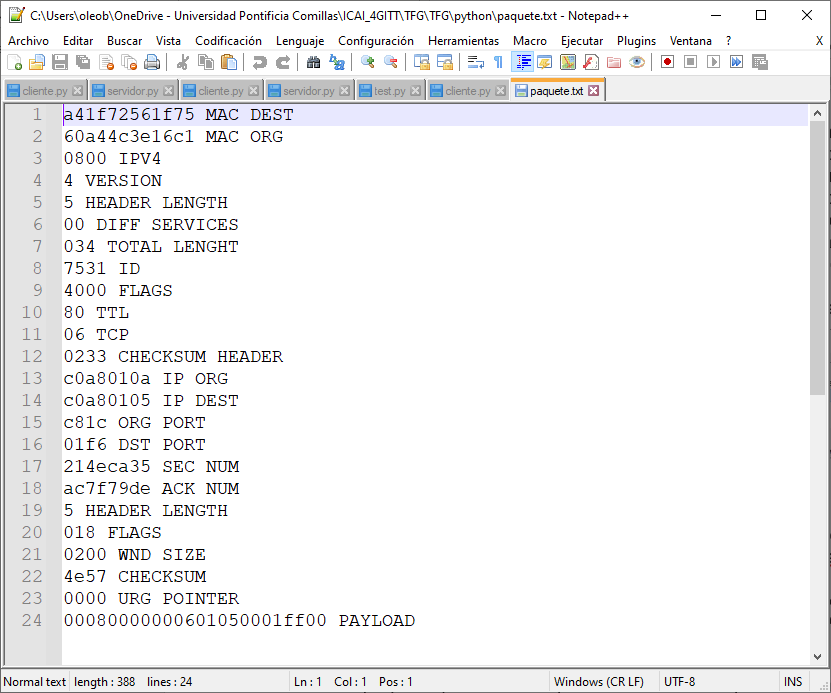


Figura 31: Paquete completamente descompuesto

Una vez ya hemos deshecho el paquete, es relativamente sencillo implementarlo en Scapy. Para editar el paquete, no vamos a hacerlo a través de la consola de Scapy, ya que requiere que pongamos muchos comandos. Para hacerlo de forma más clara y sencilla, se hará el mismo proceso que en la consola, pero en un script de Python implementando las librerías correspondientes. Para este proceso me baso en el documento de la referencia [7]. En este documento se trata sobre los elementos de los que se compone este protocolo y la base sobre la cual me baso en este proyecto para el script de Python. El script que se nos presenta en dicho documento no tiene todos los campos necesarios para este tipo de query que estamos intentando replicar (escritura de coil). Para adaptar este código a nuestro caso, podemos consultar la documentación de Modbus en Scapy [8]. En esta documentación se nos indica los tipos de datos y los campos que son siempre necesarios más aquellos que debemos de incluir para determinadas querys. Aplicando todo esto, nos queda un código como el siguiente:

from scapy.all import \*

import traceback

try:

    class Modbus(Packet):

        name = 'Modbus'

        fields\_desc = [

                XShortField("transId", int('8', 16)),

                XShortField("protoId",int('0000', 16)),

                ShortField("len", int('6', 16)),

                XByteField("unitId", int('1', 16)),

                XByteField("funcCode", int('5', 16)),

                XShortField("outputAddr", int('0001', 16)),

                XShortField("outputValue", int('ff00', 16))

                ]

    # Ether Fields

    src\_mac = '60:a4:4c:3e:16:c1'

    dst\_mac = 'a4:1f:72:56:1f:75'

    # IP Fields

    src\_ip = '192.168.1.10'

    dst\_ip = '192.168.1.5'

    leng = 52

    i\_d = 30001

    ip\_flags = 2

    ttl = 128

    ip\_checksum = int('0233', 16)

    # TCP Fields

    sport = 51228

    dport = 502

    seq = int('214eca35', 16)

    ack = int('ac7f79de', 16)

    tcp\_flags = 'PA'

    window = 512

    tcp\_checksum = int('4e57', 16)

    urgptr = 0

    pkt = Ether(src=src\_mac, dst=dst\_mac)/\

        IP(src=src\_ip, dst=dst\_ip, len=leng, id=i\_d, flags=ip\_flags, ttl=ttl, chksum=ip\_checksum)/\

        TCP(sport=sport, dport=dport, seq=seq, ack=ack, flags=tcp\_flags, window=window, chksum=tcp\_checksum, urgptr=urgptr)/\

        Modbus()

    hexdump(pkt)

    #wrpcap('filtered.pcap', pkt)

    sendp(pkt)

except:

    traceback.print\_exc()

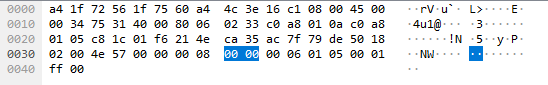
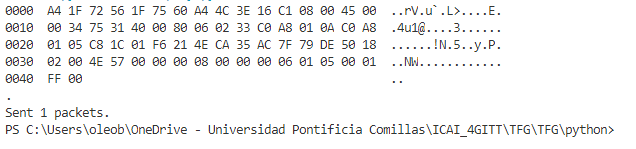
 En caso de que queramos cambiar de tipo de solicitud, habría que volver a consultar la documentación de Modbus Scapy y modificar la parte superior del código. También si quisiéramos guardar este paquete para analizarlo con wireshark, es tan fácil como descomentar el wrpcap(), pero para verificar que está bien creado, basta con ver la salida de la consola. En la consola, al ejecutar el hexdump(pkt), se mostrará en hexadecimal el paquete, el cual podemos comprobar que es idéntico al original:

Figura 32: Paquete en hexadecimal recompuesto con Scapy

Figura 33: Paquete original en hexadecimal

Esta es la forma típica de realizar un packet replay, pero dependiendo del programa de destino, nos podemos encontrar con algunos contratiempos. Por ejemplo, con el ModbusPal, no podemos enviar un paquete al destino y que este lo lea, sin previamente haber establecido una conexión con el socket de destino. Al igual que en el GitHub [9] está presente el código para generar el paquete de la figura 32, también está un código similar estableciendo un socket (PacketReplay.py). Este código se queda con la parte de la formación de los datos Modbus del paquete y una vez establecido el socket TCP, los manda. También podemos recoger la respuesta, la cual se muestra por consola como stream en hexadecimal. Utilizando este método de envío de datos basado en un socket TCP conseguimos modificar datos de los coils del esclavo. La clave de este método se basa en sacar la información sobre la ip y puerto que el esclavo utiliza usando wireshark y en saber qué tipo de mensaje Modbus queremos crear. Desde la documentación de Modbus de Scapy podemos ver que campos y tipos de datos son necesarios para distintos tipos de querys, ya que este método no solo sirve para escribir.

## Bibliografía

[1] Documentación Modbus

<https://www.cursosaula21.com/modbus-que-es-y-como-funciona/#:~:text=El%20protocolo%20Modbus%20RTU%20es,en%20serie%20utilizando%20este%20protocolo.>

[2] Documentación Modbus

<https://es.wikipedia.org/wiki/Modbus>

[3] Cliente Modbus: ModbusPal

<https://sourceforge.net/projects/modbuspal/files/modbuspal/>

[4] Ataques Modbus

<https://www.sans.org/reading-room/whitepapers/ICS/man-in-the-middle-attack-modbus-tcp-illustrated-wireshark-38095>

[4] Máster Modbus: QModMaster

<https://sourceforge.net/projects/qmodmaster/>

[5] PyModbus

<https://pymodbus.readthedocs.io/en/latest/readme.html>

[6] Ettercap

<https://www.ettercap-project.org/>

[7] Crafting Modbus Packets with Scapy

<https://westoahu.hawaii.edu/cyber/wp-content/uploads/2018/09/craft_modbus.docx>

[8] Documentación de Modbus Scapy

<https://scapy.readthedocs.io/en/latest/api/scapy.contrib.modbus.html>

[9] Repositorio de GitHub

<https://github.com/miguelob/TFG-Ciberseguridad>