

Práctica 2: Interacción con sistemas críticos

El objetivo de esta práctica es profundizar en los conceptos explicados en las clases de teoría. En concreto, utilizaremos los conceptos explicados en el tema 4.

La práctica está formada por dos ejercicios y en cada uno se simula un escenario diferente. Cada escenario puede resolverse de múltiples formas, no habiendo una solución única en ninguno de los ejercicios.

Ante cualquier duda durante la resolución de la práctica, escribir un email a sergio.perez.pelo@urjc.es y raul.martin@urjc.es. En caso de no poderse resolver la duda vía mail, se puede concertar una tutoría, siempre y cuando se concierte en un período de **hasta 48 horas antes** de la fecha de entrega de la práctica.

Normas y evaluación

- La realización de la práctica se realizará de forma **individual**.
- Se debe realizar una memoria donde se describirán los pasos realizados y el procedimiento utilizado en cada uno de los ejercicios. Incluye capturas en todas aquellas partes que ayuden a la comprensión o aporten información interesante. Si has resuelto algún ejercicio de varias formas, puedes aportar detalles sobre las diferentes alternativas, pero con una es suficiente.
- La memoria será entregada en formato pdf. Puedes utilizar Word, LaTeX / Overleaf, Markdown o cualquier otra herramienta que consideres para generar la memoria, pero el entregable **debe ser un fichero PDF**.
- **La extensión máxima de la memoria será de 10 páginas.** Este límite **NO incluye los anexos** que consideres relevantes, como por ejemplo el código desarrollado para resolver los escenarios propuestos.
- La fecha límite para entregar la práctica será el **12 de mayo a las 23:55**.
- Esta práctica se corresponde con un 20% de la nota final. **La nota mínima para considerar la práctica superada es un 5.**

1 Entorno para la práctica

Para facilitar el despliegue de las herramientas que vamos a utilizar, hemos preparado un contenedor Docker con la aplicación completa. Antes de nada, es necesario instalar Docker para el sistema operativo que se esté utilizando, siguiendo las instrucciones oficiales:

- Para Windows: <https://docs.docker.com/desktop/install/windows-install/>
- Para Mac: <https://docs.docker.com/desktop/install/mac-install/>
- Para Linux: <https://docs.docker.com/engine/install/#supported-platforms>

Una vez instalado y configurado Docker, se puede arrancar la aplicación para la práctica utilizando el comando `start-practica2.sh`, el cual se encargará de configurar tanto el contenedor como la red, exponiendo los puertos necesarios. Un resumen del entorno se puede ver en la Fig. 1.

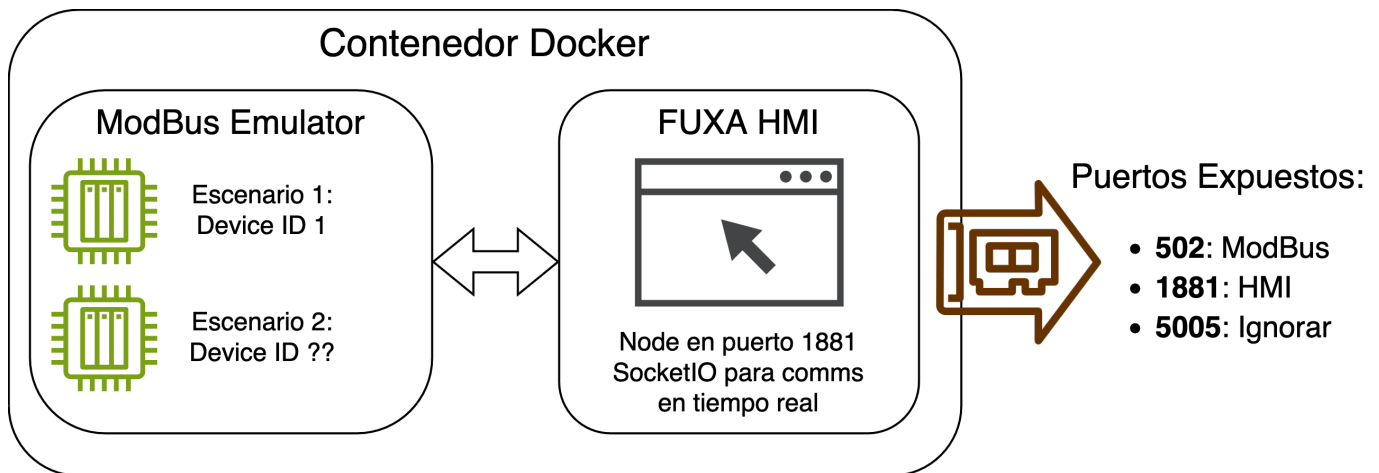


Fig. 1: Contenido del contenedor Docker utilizado en la práctica 2.

El puerto 502 expone una conexión directa a los dispositivos a manipular durante la práctica, utilizando el protocolo ModBus. De esta forma, es posible utilizar herramientas externas a Docker (por ejemplo, Metasploit), para interactuar con ellos fácilmente.

El puerto 1881 expone la interfaz de usuario para visualizar el estado del sistema (HMI). La interfaz de usuario está implementada con FUXA (<https://github.com/frangoteam/FUXA>), y se puede acceder a ella utilizando cualquier navegador web moderno con la URL <http://localhost:1881>. Por último, el puerto 5005 debe ser ignorado durante la realización de la práctica, salvo instrucciones de lo contrario por parte de los docentes. Este puerto permite conectar un depurador al emulador de modbus, en caso de que surjan problemas o comportamientos inconsistentes.

Para navegar entre el primer y el segundo escenario en el visor web, es suficiente con acceder al menú lateral, en la parte superior izquierda de la interfaz, como se muestran en la Fig. 2.

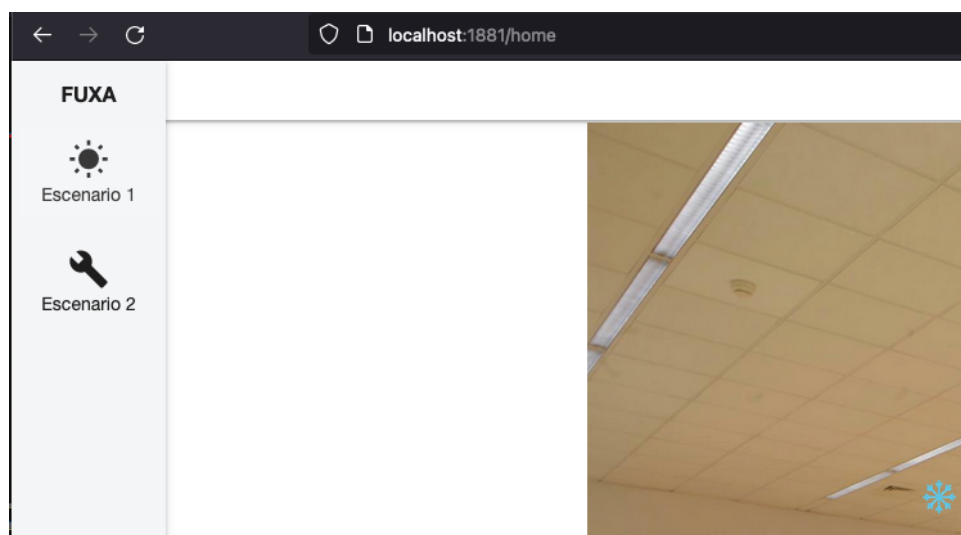


Fig. 2: Ubicación del menú lateral en la interfaz web de FUXA, que permite alternar entre los escenarios disponibles.

2 ModBus

El protocolo ModBus es uno de los protocolos más utilizados en la industria dentro de los conocidos sistemas SCADA. Existen varias formas de transmitir o encapsular mensajes en el contexto de ModBus (Serial, TCP/IP, UDP/IP, ASCII, etc.), en esta práctica, nos vamos a centrar en la variante TCP/IP, la cual encapsula los mensajes a transmitir utilizando el protocolo TCP. De esta forma, es posible interactuar fácilmente y analizar el tráfico con las unidades remotas utilizando herramientas conocidas, utilizando librerías como pwntools en el caso de Python ¹, o Wireshark² para el análisis de tráfico. Los mensajes transmitidos tienen la siguiente estructura:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Transaction Identifier																Protocol Identifier															
Length																Unit Identifier								Function Code							
Reference Number																Word Count															
...																															

Los argumentos dependen de la función

La descripción de los campos es la siguiente:

- Transaction Identifier (2 bytes): número aleatorio que será replicado en la respuesta. De esta forma, es posible saber a qué mensaje responde cada paquete.
- Protocol Identifier (2 bytes): siempre vale 0x0000. En teoría valdría para expandir Modbus en el futuro.
- Length (2 bytes): longitud en bytes del resto de campos del paquete.
- Unit Identifier (1 byte): identificador del dispositivo al que va dirigido el mensaje. En algunos manuales de Modbus, este campo se llama *Slave ID*. **En el primer escenario, el ID del dispositivo con el que queremos comunicarnos es 1, en el segundo escenario, el id del dispositivo se desconoce, y debemos averiguarlo.**
- Function Code: Función que queremos ejecutar. Modbus tiene estandarizadas varias, como pueden ser leer o escribir un *coil*, leer uno o varios *holding register*, etc. Se puede encontrar un listado detallado en [este enlace](#):
- Reference Number: argumentos para la función invocada. Por ejemplo, para la función 0x03 (leer registros). Indica la dirección del registro a partir de la cual queremos leer. Empieza en 0, y cada registro tiene 16 bits de capacidad.
- Word Count: argumentos para la función invocada. Por ejemplo, como segundo argumento para la función 0x03 (leer registros), indica la cantidad de registros a leer.

Las respuestas, por su parte, siguen la siguiente estructura:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Transaction Identifier																Protocol Identifier															
Length																Unit Identifier								Function Code							
...																}															
...																															

El contenido de la respuesta depende de la función invocada

¹<https://docs.pwntools.com/en/stable/>

²<https://www.wireshark.org/download.html>

Como se puede observar, la estructura de la respuesta es muy parecida a una petición. **En caso de que se produzca un error o el dispositivo remoto se niegue a realizar la petición, el contenido de la respuesta estará vacío, y el código de función será > 128 (0x80 en hexadecimal).** Las peticiones más útiles a lo largo de la práctica serán:

- 0x01 Read Coil: Permite leer el estado de un actuador. Referencia: https://ozeki.hu/p_5876-mobdbus-function-code-1-read-coils.html.
- 0x03 Read Holding Register: permite leer el contenido de varios bloques de memoria contiguos, normalmente usado para recuperar el estado del dispositivo y datos reportados por los sensores. Referencia: https://ozeki.hu/p_5878-mobdbus-function-code-3-read-multiple-holding-registers.html.
- 0x05 Write Single Coil: permite cambiar el estado de un actuador, o bien encenderlo o apagarlo. Referencia: https://ozeki.hu/p_5880-mobdbus-function-code-5-write-single-coil.html.

3 Escenario 1: Análisis sistema climatización

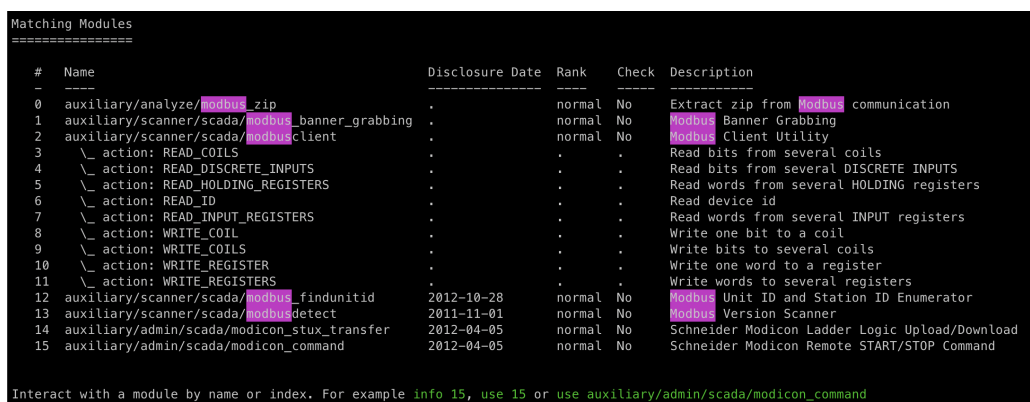
Los estudiantes de la Universidad Príncipe Carlos Juan se han quejado de un comportamiento anómalo en el sistema de climatización del aula. Parece que el control de temperatura, que permite elegir si queremos aire acondicionado o calefacción, funciona de forma errónea. Al manipular el interruptor que controla si se activa la calefacción o el aire condicionado dentro del aula, los estudiantes han observado que solo cambia la temperatura en una de las zonas del aula. El resto de salidas de aire siempre expulsan aire frío, independientemente de la posición del interruptor.

Los estudiantes de ciberseguridad están cansados de pasar frío, y se han propuesto hacer ingeniería inversa al sistema, averiguar cómo funciona, e intentar implementar su propio control de temperatura para suplir al actual. ¿Serán capaces?

3.1 Interacción con el sistema

Escaneando la red, hemos descubierto que podemos interactuar con el sistema a través del puerto 502. Vamos a probar a interactuar con el sistema utilizando Metasploit.

3.1.1 Busca en Metasploit los módulos relacionados con Modbus, utilizando el comando search modbus, y resume para qué sirve cada uno.



#	Name	Disclosure Date	Rank	Check	Description
0	auxiliary/analyze/modbus_zip	.	normal	No	Extract zip from modbus communication
1	auxiliary/scanner/scada/modbus_banner_grabbing	.	normal	No	modbus Banner Grabbing
2	auxiliary/scanner/scada/modbus_client	.	normal	No	modbus Client Utility
3	action: READ_COILS	.	.	.	Read bits from several coils
4	action: READ_DISCRETE_INPUTS	.	.	.	Read bits from several DISCRETE INPUTS
5	action: READ_HOLDING_REGISTERS	.	.	.	Read words from several HOLDING registers
6	action: READ_ID	.	.	.	Read device id
7	action: READ_INPUT_REGISTERS	.	.	.	Read words from several INPUT registers
8	action: WRITE_COIL	.	.	.	Write one bit to a coil
9	action: WRITE_COILS	.	.	.	Write bits to several coils
10	action: WRITE_REGISTER	.	.	.	Write one word to a register
11	action: WRITE_REGISTERS	.	.	.	Write words to several registers
12	auxiliary/scanner/scada/modbus_findunitid	2012-10-28	normal	No	modbus Unit ID and Station ID Enumerator
13	auxiliary/scanner/scada/modbus_detect	2011-11-01	normal	No	modbus Version Scanner
14	auxiliary/admin/scada/modicon_stux_transfer	2012-04-05	normal	No	Schneider Modicon Ladder Logic Upload/Download
15	auxiliary/admin/scada/modicon_command	2012-04-05	normal	No	Schneider Modicon Remote START/STOP Command

Interact with a module by name or index. For example info 15, use 15 or use auxiliary/admin/scada/modicon_command

Fig. 3: Resultado de buscar en Metasploit utilizando el comando search modbus

3.1.2 Utilizando el módulo `scanner/scada/modbusdetect`, comprueba si realmente el puerto 502 se corresponde con un servicio Modbus.

3.1.3 ¿Existe algún mecanismo en metasploit para enumerar dispositivos que respondan al protocolo Modbus? En caso afirmativo, ¿cuál? En caso contrario, ¿qué herramienta utilizarías y cómo?

3.2 Análisis de tráfico

Utilizando Wireshark y observando el tráfico de la interfaz de red local, responde a las siguientes preguntas:

3.2.1 ¿Cuál es el ID de transacción que utiliza el módulo de enumeración de Modbus de Metasploit? ¿Crees que podría tener alguna consecuencia o implicación de OPSEC?

3.2.2 Realiza alguna interacción con el sistema a través de Metasploit, observa el tráfico que se genera e identifica los bytes más relevantes del paquete, estableciendo la correspondencia entre lo que ves en la traza y el formato de paquetes explicado al inicio del enunciado.

3.3 Implementa el protocolo

Como además queremos tener el máximo control sobre las pruebas que vamos a realizar, hemos decidido implementar nosotros el protocolo. Para ello, utilizaremos la librería `pwntools` en Python 3. Existen librerías en Python que ya implementan el protocolo modbus, como `pymodbus`, **pero no las usaremos durante esta práctica**. En concreto, se pide implementar los siguientes puntos:

3.3.1 Leer estado *coil*

Implementa una función en Python similar a la siguiente, que permita recuperar el estado de un *coil*, conociendo su dirección de memoria y el ID del dispositivo. El ID de la función Modbus a utilizar es 1.

```
def read_coil(conn: remote, uid: int, start_address: int):
    """
    Read a value from a coil
    :param conn: connection to modbus device
    :param uid: device id
    :param start_address: address to read
    :return: True if the coil is on, False if not,
    an exception if the address cannot be read or the modbus device an error
    """
```

Comprueba que puedes leer los *coils* que controlan el estado del aire, y que el valor del primer *coil* cambia al accionar el interruptor.

3.3.2 Escribir estado *coil*

Implementa una función en Python similar a la siguiente, que permita modificar el estado de un *coil*, conociendo su dirección de memoria y el ID del dispositivo. El ID de la función Modbus a utilizar es 5.

```
def write_coil(conn: remote, uid: int, start_address: int, value: bool):
    """
    Read a value from a coil
    :param conn: connection to modbus device
    :param uid: device id
    :param start_address: address to read
```

```
:param value: value to write to the coil
"""
```

Comprueba que puedes modificar los *coils* que controlan el estado del aire, y que, **tanto a través de la función de lectura como de la interfaz web (HMI)**, se observa cómo se reflejan los valores en tiempo real.

3.3.3 Leer valores de registros o *holding registers*

Implementa una función en Python similar a la siguiente, que permita leer el valor de un *holding register*, conociendo su dirección de memoria y el ID del dispositivo. El ID de la función Modbus a utilizar es 3.

```
def read_hregister(conn: remote, uid: int, start_address: int):
    """
    Read a value from a holding register
    :param conn: connection to modbus device
    :param uid: device id
    :param start_address: address to read
    :return: numeric value read from holding register, or an exception
    if the address cannot be read or the modbus device an error
    """
```

3.4 Verificar control sistema climatización

Utilizando las funciones previamente implementadas, y sabiendo que el ID del primer actuador que controla una de las salidas de aire es el 17, implementa un script en Python que permita cambiar entre aire caliente o frío para todas las salidas de aire del aula.

4 Escenario 2: Hidrata al becario

Se acerca el verano, y con el calor se hace especialmente importante mantenerse hidratado. Varios becarios de la universidad han construido un grifo de líquido hidratante controlado por un dispositivo Arduino que les permite permanecer frescos e hidratados sin necesidad de interaccionar con el grifo. Simplemente llegan, llenan su vaso y se lo beben.

El sistema cuenta con un sensor que detecta el nivel de líquido en el vaso y lo rellena de forma apropiada. Sabemos que han implementado la comunicación con el sistema utilizando el protocolo Modbus. De lo que se han olvidado es de programar el sistema de control que monitoriza el nivel del vaso, y controla el grifo de forma apropiada, teniendo en cuenta:

- Para minimizar el desperdicio, solo se comienzan a llenar vasos que estén vacíos.
- Se debe evitar que el vaso rebose, o podrían ocurrir consecuencias desastrosas para el entorno.

Nuestra misión es implementar este sistema de monitorización y lógica de control precisamente para evitar que el vaso se quede vacío, y los becarios corran el riesgo de deshidratarse, o que el vaso rebose, y tengamos pérdidas desastrosas en el sistema de hidratación.

4.1 Enumera y encuentra el dispositivo

En el primer escenario, sabíamos que el dispositivo tenía el ID 1. En este segundo escenario, desconocemos el ID del dispositivo, por lo que tendremos que enumerar hasta descubrir el dispositivo correcto. Para realizar este escenario, es necesario haber completado el escenario anterior, porque reutilizaremos parte del código implementado.

4.2 Enumeración con metasploit

Realiza la enumeración de la red utilizando los módulos de metasploit explorados en el escenario anterior. En la Fig. 4 puedes encontrar un ejemplo de la salida esperada en una enumeración correcta.

```
msf6 auxiliary(scanner/scada/modbus_findunitid) > run
[*] Running module against 127.0.0.1

[+] 127.0.0.1:502 - Received: correct MODBUS/TCP from stationID 1
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 2 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 3 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 4 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 5 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 6 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 7 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 8 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 9 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 10 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 11 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 12 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 13 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 14 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 15 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 16 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 17 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 18 (probably not in use)
[+] 127.0.0.1:502 - Received: correct MODBUS/TCP from stationID 19
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 20 (probably not in use)
[*] 127.0.0.1:502 - Received: incorrect/none data from stationID 21 (probably not in use)
^C[-] 127.0.0.1:502 - Stopping running against current target...
```

Fig. 4: Ejemplo de enumeración unidades Modbus utilizando Metasploit

4.3 Enumeración con Python

Usa las funciones que has implementado en el ejercicio anterior, vamos a analizar el sistema y entender como funciona. El primer paso que debemos realizar es descubrir el ID del dispositivo a controlar, de forma similar a la sección anterior, pero utilizando nuestro código en vez de Metasploit.

4.3.1 ¿Qué error recibes cuando el ID del dispositivo no es correcto?

Compáralo con el error recibido cuando el dispositivo es correcto, pero la dirección de memoria no.

4.3.2 Diferencia con Metasploit

Analizando la diferencia entre el tráfico que genera metasploit y el generado por tu código, ¿se te ocurre alguna forma de detectar fácilmente el uso del módulo de Metasploit?

4.4 Implementar lógica de hidratación

Una vez que hemos averiguado el ID del dispositivo, queremos implementar un método que, utilizando los métodos creados en el primer escenario, rellene el vaso sin desperdiciar líquido. Sabemos que hay un sensor que nos indica el porcentaje de llenado del vaso, devolviendo una medición entre 0%, si el vaso está vacío, y 100% si el vaso está lleno, pero desconocemos su dirección.

Nos han avisado de que existen al menos dos actuadores, uno que sirve para detener el motor del grifo, utilizado en caso de emergencia, y otro que controla la propia llave de apertura del grifo.

Si el vaso supera su capacidad, se romperá y será necesario reiniciar el escenario. ¿Serás capaz de descubrir cómo funciona el sistema e implementarlo?

Pista: en los sistemas de control, cuando interactuamos con el mundo real, existen variaciones y rangos en las medidas, nunca vamos a medir exactamente la cantidad como 0.0. Por ejemplo, para considerar que el vaso está vacío, puedes mirar si el nivel reportado es < 5 en vez de $== 0$. Recuerda

que desconocemos los tiradores y las posiciones de memoria de los registros que contienen información, nuestra primera tarea será localizarlos.