

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR



Escuela Politécnica Superior



Grado en Ingeniería de Tecnologías y
Servicios de Telecomunicación

TRABAJO FIN DE GRADO

**DESARROLLO DE UN SISTEMA
EFICIENTE DE ANÁLISIS DE
TRÁFICO MODBUS TCP PARA LA
DETECCIÓN DE ANOMALÍAS EN
REDES SCADA**

Autor: Celia Pascual Casado

Tutor: Jorge E. López de Vergara Méndez

Febrero 2020

DESARROLLO DE UN SISTEMA EFICIENTE DE ANÁLISIS DE TRÁFICO MODBUS TCP PARA LA DETECCIÓN DE ANOMALÍAS EN REDES SCADA

Autor: Celia Pascual Casado

Tutor: Jorge E. López de Vergara Méndez

High Performance Computing and Networking
Research Group

Departamento de Tecnología Electrónica y de las
Comunicaciones

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Febrero 2020

Resumen

Durante los últimos años, se ha producido un aumento exponencial del número de ataques en redes informáticas, causando que la seguridad se convierta en un tema de gran relevancia para empresas e industrias.

En la actualidad, las redes SCADA (*Supervisory Control and Data Acquisition*) tienen un papel imprescindible a nivel industrial, ya que se encargan del control, supervisión y adquisición de datos en procesos industriales y se encuentran presentes en una gran cantidad de estos procesos, desde centrales eléctricas a sistemas de transporte, suponiendo un gran avance en la automatización industrial. Estas redes no quedan exentas de ciberataques, lo cual podría traer graves consecuencias al controlar procesos de gran importancia para la humanidad. Por esta razón es necesario implementar mecanismos de defensa ante cualquier amenaza.

A lo largo de este TFG se muestra el desarrollo de un IDS (*Intrusion Detection System*) que se encarga de capturar el tráfico de tipo Modbus TCP, protocolo ampliamente utilizado en redes SCADA, analizarlo y extraer información de relevancia sobre este. El objetivo es crear un sistema capaz de detectar distintos tipos de amenazas que pueden darse en estas redes generando avisos en caso de que llegaran a producirse. También se han llevado a cabo diferentes pruebas con el fin de validar el correcto funcionamiento del IDS.

Palabras Clave

SCADA, IDS, Modbus TCP

Abstract

During the last years, there has been an exponential increase in the number of attacks on computer networks, causing security to become a major issue for enterprises and industries.

Nowadays, SCADA networks (Supervisory Control and Data Acquisition) have an essential role in the industry, since they are in charge of the control, supervision and acquisition of data in industrial processes and are present in a large number of these processes, from power plants to transport systems, assuming a great advantage in industrial automation. These networks are not exempt from cyber attacks, which could have serious consequences as they control processes of great importance for humanity. For this reason it is necessary to implement defense mechanisms against any threat.

Throughout this project, the development of an IDS (Intrusion Detection System) is shown, which is responsible for capturing Modbus TCP traffic, a protocol widely used in SCADA networks, analyze it and extract relevant information about it. The objective is to create a system capable of detecting different types of threats that can occur in these type of networks, generating warnings in case they occur. Different tests have also been carried out in order to validate the correct functioning of the IDS.

Key words

SCADA, IDS, Modbus TCP

Agradecimientos

Quiero comenzar dando las gracias a mi tutor, Jorge E. López de Vergara, por haberme dado la oportunidad de realizar este TFG con el que he podido dar un paso más en el conocimiento del mundo de las redes. También le quiero agradecer el gran interés mostrado y seguimiento llevado a cabo durante todo este tiempo.

Del mismo modo, a la Cátedra UAM-Naudit por proporcionarme la posibilidad de probar el detector en un entorno real.

A mis padres, Jesús e Isabel, por el cariño, apoyo y cada valor que me han inculcado y me ha hecho ser quien soy. Sin ellos no habría sido posible llegar a donde estoy hoy.

A mis amigas, porque cada momento juntas me llena de energía para afrontar cualquier situación. En especial me gustaría nombrar a Bea y a Lorena, por estar ahí desde siempre y ser un gran apoyo para mi.

A Julio, uno de los pilares fundamentales de mi vida, ser la fuerza que a veces me falta y tener siempre las palabras correctas para animarme y calmarme en los momentos en los que lo he necesitado.

Y por último me gustaría también dar las gracias a todas las personas que me han acompañado durante esta carrera, y con las que espero seguir contando por mucho tiempo. Gracias por hacer estos años mucho más amenos y llevaderos.

Glosario de acrónimos

- **SCADA:** *Supervisory Control and Data Acquisition* (Supervisión, Control y Adquisición de Datos).
- **IDS:** *Intrusion Detection System* (Sistema de Detección de Intrusiones).
- **NIDS:** *Network-Intrusion Detection System* (Sistema de Detección de Intrusiones en una Red).
- **HIDS:** *Host-Intrusion Detection System* (Sistema de Detección de Intrusiones en un Host).
- **TCP:** *Transmission Control Protocol* (Protocolo de Control de Transmisión).
- **IP:** *Internet Protocol* (Protocolo de Internet).
- **MTU:** *Master Terminal Unit* (Unidad Terminal Maestra).
- **RTU:** *Remote Terminal Unit* (Unidad Terminal Remota).
- **PLC:** *Programmable Logic Controller* (Controlador Lógico Programable).
- **PAC:** *Programmable Automation Controller* (Controlador de Automatización Programable).
- **HMI:** *Human-Machine Interface* (Interfaz Humano Máquina).
- **MBAP:** *Modbus Application Protocol Header* (Cabecera del Protocolo de Aplicación Modbus).
- **PDU:** *Protocol Data Unit* (Unidad de Datos de Protocolo).
- **OSI:** *Open System Interconnection* (Interconexión de Sistemas Abiertos).
- **VLAN:** *Virtual Local Area Network* (Red de Área Local Virtual).

Índice general

| | |
|--|-------------|
| Glosario de acrónimos | VII |
| Índice de figuras | XI |
| Indice de tablas | XIII |
| 1. Introducción | 1 |
| 1.1. Motivación | 1 |
| 1.2. Objetivos | 2 |
| 1.3. Fases de realización | 2 |
| 1.4. Estructura de la memoria | 3 |
| 2. Estado del arte | 5 |
| 2.1. Introducción | 5 |
| 2.2. SCADA | 5 |
| 2.2.1. Arquitectura | 6 |
| 2.3. Modbus TCP | 7 |
| 2.3.1. Modelo cliente-servidor | 8 |
| 2.3.2. Estructura de la trama Modbus TCP | 9 |
| 2.3.3. Vulnerabilidades | 11 |
| 2.4. Sistema de Detección de Intrusiones | 12 |
| 2.5. Conclusiones | 13 |
| 3. Desarrollo | 15 |
| 3.1. Introducción | 15 |
| 3.2. Entorno de desarrollo | 15 |
| 3.3. Análisis del tráfico | 16 |
| 3.4. Mecanismos de comprobación de anomalías | 20 |
| 3.5. Visualización de los resultados | 22 |
| 3.6. Conclusiones | 22 |

| | |
|--|-----------|
| 4. Validación | 23 |
| 4.1. Introducción | 23 |
| 4.2. Validación de funcionalidad | 23 |
| 4.2.1. Pruebas iniciales | 23 |
| 4.2.2. Pruebas con Modpoll, Metasploit y ModbusPal | 26 |
| 4.2.3. Pruebas con tráfico real | 35 |
| 4.3. Validación de rendimiento | 36 |
| 4.4. Conclusiones | 36 |
| 5. Conclusiones y trabajo futuro | 37 |
| 5.1. Conclusiones | 37 |
| 5.2. Trabajo futuro | 38 |
| Bibliografía | 40 |

Índice de figuras

| | |
|--|----|
| 1.1. Diagrama de Gantt. | 3 |
| 2.1. Arquitectura de una red SCADA. | 7 |
| 2.2. Capas del modelo OSI en Modbus TCP. | 8 |
| 2.3. Comunicación en Modbus TCP [1] | 9 |
| 2.4. Estructura de la trama Modbus TCP [2] | 9 |
| 2.5. Tabla con los códigos de función públicos más comunes. [3] | 10 |
| 3.1. Configuración modo NAT. | 16 |
| 3.2. Ejecución del programa mediante captura de tráfico. | 17 |
| 3.3. Ejecución del programa mediante la lectura de una traza. | 17 |
| 3.4. Tabla <i>hash</i> con listas enlazadas. | 17 |
| 3.5. Estructura para la muestra de tramas por dirección IP. | 18 |
| 3.6. Estructura para la muestra del número de clientes o maestros. | 18 |
| 3.7. Estructura para la muestra de IDs repetidos. | 19 |
| 3.8. Estructura para el cálculo de peticiones por unidad de tiempo. | 19 |
| 3.9. Fórmulas aplicadas para el suavizado exponencial. | 19 |
| 3.10. Captura de Wireshark con la estructura de un paquete. | 20 |
| 3.11. Ejemplo de paquete con varias tramas Modbus TCP. | 21 |
| 4.1. Resultado de Tshark para la IP 141.81.0.10 y código de función 1. | 25 |
| 4.2. Resultado de Tshark para la IP 141.81.0.10 y código de función 2. | 25 |
| 4.3. Resultado de Tshark para la IP 141.81.0.164 y código de función 15. | 25 |
| 4.4. Interfaz del simulador de servidor ModbusPal. | 26 |
| 4.5. Comando de ejecución de ModbusPal. | 26 |
| 4.6. Interfaz de captura de tráfico Vmnet8. | 27 |
| 4.7. Comando de Modpoll para petición de lectura. | 27 |
| 4.8. Comando de Modpoll para petición de escritura. | 27 |
| 4.9. Comprobación de tipos de paquete con la herramienta Tshark. | 29 |

| | |
|---|----|
| 4.10. Comprobación de IDs con la herramienta Tshark. | 29 |
| 4.11. Resultado tras generar anomalías en la tasa de paquetes. | 29 |
| 4.12. Módulos de Metasploit para Modbus TCP. | 30 |
| 4.13. Opciones del módulo <i>modbusdetect</i> | 30 |
| 4.14. Respuesta ante el exploit <i>modbusdetect</i> | 30 |
| 4.15. Resultado de la ejecución del tráfico generado con <i>modbusdetect</i> | 31 |
| 4.16. Captura de Wireshark con la excepción <i>Illegal Function</i> | 31 |
| 4.17. Respuesta ante el exploit <i>modbus_findunitid</i> | 32 |
| 4.18. Resultado de la ejecución del tráfico generado con <i>modbus_findunitid</i> | 32 |
| 4.19. Opciones del exploit <i>modbusclient</i> | 32 |
| 4.20. Ejemplo de escritura en bobina con <i>modbusclient</i> | 33 |
| 4.21. Resultado de la ejecución del tráfico generado con <i>modbusclient</i> | 33 |
| 4.22. Captura de Wireshark de las respuestas con excepción. | 33 |
| 4.23. Captura de Wireshark de un paquete con anomalía. | 34 |
| 4.24. Resultado de la ejecución del tráfico generado con <i>modicon_command</i> | 34 |
| 4.25. Alertas generadas tras la ejecución de <i>modicon_command</i> | 35 |
| 4.26. Captura de paquete con longitud superior a la permitida. | 35 |

Indice de tablas

| | |
|---|----|
| 4.1. Resultado de la ejecución de la traza de Github. | 24 |
| 4.2. Resultados tras la ejecución con Modpoll. | 28 |
| 4.3. Resultados de las medidas de rendimiento. | 36 |

1

Introducción

1.1. Motivación

En los últimos años, el uso de Internet ha ido ganando fuerza no sólo a nivel particular, sino que cada vez son más los dispositivos conectados a la red, desde móviles o PC's hasta maquinaria utilizada en procesos industriales, a través de lo que se conoce como *Internet de las Cosas* o *IoT (Internet of Things)*[4]. Esta conexión de dispositivos mediante Internet ha traído consigo grandes ventajas a nivel industrial, como la automatización de procesos o la optimización del análisis de datos.

Sin embargo, este aumento de la popularidad de la *Internet de las Cosas* también ha dado lugar a grandes desventajas, como es la aparición de vulnerabilidades en la seguridad de estos sistemas, que muchas veces cuentan con mecanismos de protección insuficientes para hacer frente a todos los ciberataques que pueden afectarles. Esta falta de protección ante los ataques cibernéticos supone un importante peligro para la industria, ya que estos ataques pueden producir averías en la cadena de control, teniendo como consecuencia una inesperada interrupción del servicio, que puede ser de gran gravedad.

Por esta razón, es necesario disponer de mecanismos eficientes de protección frente a estos ataques. Aquí es donde entran en juego los Sistemas de Detección de Intrusiones o IDS (del inglés, *Intrusion Detection System*). En este Trabajo Fin de Grado (TFG) se ha llevado a cabo el desarrollo de un IDS para el protocolo Modbus TCP, el cual se usa en redes industriales en las que puede llegar a existir un gran número de dispositivos, lo que tiene como consecuencia una elevada cantidad de tráfico de red. Por este motivo, es necesario que el IDS sea realizado en un lenguaje eficiente, capaz de procesar la información en tiempo real.

Para la realización de este proyecto se ha tomado como referencia otro TFG [5] desarrollado en Python, razón por la cual no procesaba el tráfico a una velocidad suficientemente rápida para considerarse eficiente. Por este motivo, el lenguaje elegido para el desarrollo de este TFG ha sido C, ya que posee una velocidad de ejecución mucho más rápida y con un menor consumo de recursos.

1.2. Objetivos

El objetivo de este TFG es la realización de un Sistema de Detección de Intrusiones eficiente para redes SCADA, Supervisión, Control y Adquisición de Datos (*Supervisory Control And Data Acquisition*), basadas en el protocolo Modbus TCP.

Para ello, se ha implementado un sistema que analiza el tráfico recibido y enviado por el puerto TCP 502, que es el puerto que utiliza el protocolo Modbus TCP. Se estudian los diferentes campos que componen cada paquete para así detectar si presentan alguna anomalía, avisando al operador de la red SCADA en caso de que así fuera.

El sistema debe ser capaz también de reconocer el número de nodos maestros que se encuentran en una misma red. Además de esto, se debe analizar el número de paquetes enviados por un maestro en un determinado intervalo de tiempo, controlando así que no se produzca una sobrecarga de paquetes en el lado de los nodos remotos. Todo este análisis debe asimismo realizarse en un tiempo de ejecución razonable, es decir, el IDS debe ser capaz de procesar la información a una tasa de al menos 1 Gbps.

1.3. Fases de realización

A continuación, se detallan las fases en las que se ha llevado a cabo la realización de este TFG, según se muestra en la Figura 1.1:

- Estudio del Estado del Arte: Esta fase ha sido la más duradera, ya que se ha necesitado conocer en profundidad el funcionamiento de un IDS, las redes SCADA y el protocolo Modbus TCP, además de los ataques que pueden sufrir dichas redes. Se pueden distinguir dos partes, en la primera de ellas se ha recopilado información sobre el protocolo y el entorno en el que es utilizado, y en la segunda, sobre cómo llevar a cabo el proceso de validación mediante el estudio de diversas herramientas y programas.
- Diseño y Desarrollo: Durante esta fase se realizó un programa en C capaz de analizar el tráfico de red, centrándose en aquellos paquetes en los que está presente el protocolo Modbus TCP, y detectar anomalías si las hubiera.
- Validación: En esta etapa se llevó a cabo la comprobación del correcto funcionamiento del código, y la resolución de los errores encontrados. También se comprobó qué rendimiento ofrece el programa, para validar su eficiencia.
- Redacción de la Memoria: La última fase ha consistido en la redacción de este documento, en el que queda plasmado todo lo aprendido durante la elaboración de este TFG.

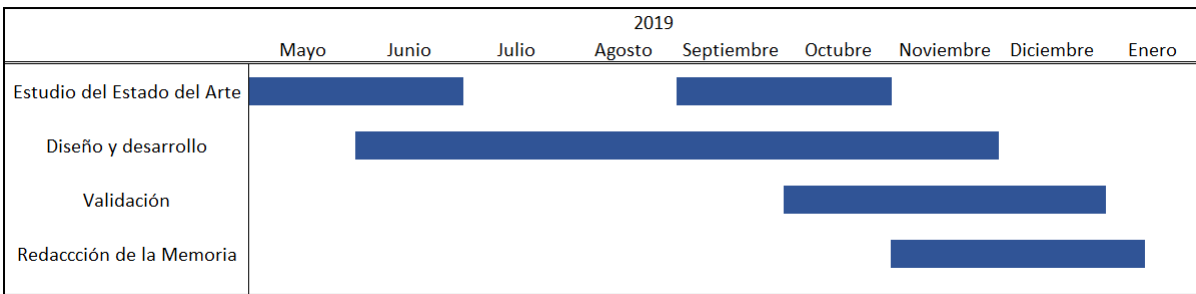


Figura 1.1: Diagrama de Gantt.

1.4. Estructura de la memoria

El resto de esta memoria se desglosa en los siguientes capítulos:

- Estado del arte: Se corresponde con el Capítulo 2 de la memoria. En él se realizará una explicación detallada sobre las redes SCADA, su arquitectura y las vulnerabilidades que afectan a este tipo de redes. También se profundizará en el protocolo Modbus TCP y los campos que lo componen. Por último, se llevará a cabo la descripción de la funcionalidad de un IDS y se explicarán los distintos tipos que existen.
- Desarrollo: Se trata del Capítulo 3 de este documento y en él se describe el entorno de desarrollo utilizado para la implementación del detector, también se describen las características del mismo y qué características se han tenido en cuenta a la hora de discernir entre tráfico anómalo y legítimo. Por último, se lleva a cabo una explicación de los resultados que ofrece el programa.
- Validación: Es el Capítulo 4 de la memoria. En él se detallan todas las pruebas realizadas para verificar que el IDS cumple con los requisitos necesarios para considerarse funcional. Adicionalmente, se detallan las características y el funcionamiento de todas las herramientas utilizadas en este proceso.
- Conclusiones y trabajo futuro: Se trata del último apartado de la memoria, correspondiente al Capítulo 4, en el cual se describen las conclusiones extraídas a partir de la realización de este proyecto, y se plantean varias mejoras y alternativas para la posible continuidad del desarrollo de este TFG.

2

Estado del arte

2.1. Introducción

Este capítulo tiene como objetivo realizar una recapitulación sobre los conocimientos previos que ha sido necesario adquirir para la realización de este TFG.

Se hará un repaso del estado del arte en relación a las redes SCADA y los ataques que reciben este tipo de sistemas. En concreto, se profundizará en aquellos sistemas que utilicen el protocolo Modbus TCP.

Adicionalmente, se realizará una descripción de los sistemas de detección de intrusiones y los distintos tipos que existen.

2.2. SCADA

Los sistemas SCADA, *Supervisory Control and Data Acquisition*, o en español Supervisión, Control y Adquisición de Datos, son aquellos ampliamente utilizados en infraestructuras industriales consideradas críticas. Su función es la de controlar y supervisar procesos industriales a través de la obtención y el procesamiento de datos para su posterior análisis. [6]

Algunas infraestructuras críticas que son controladas por redes SCADA pueden ser sistemas de suministro de agua, plantas de procesamiento químico, sistemas de transporte y centrales eléctricas y de distribución de gas.

2.2.1. Arquitectura

La arquitectura de los sistemas SCADA, mostrada en la Figura 2.1, se compone de los siguientes elementos:

MTU (*Master Terminal Unit* o *Unidad Terminal Maestra*)

El MTU es el elemento central del sistema. Su función es la de reunir la información proporcionada por las demás subestaciones, almacenando y procesando así los datos relevantes de manera que un operador humano sea capaz de entenderlos (en forma de gráficas, tablas, imágenes, etc), ya que en ellos se encuentra el software HMI, que será explicado más adelante.

La comunicación entre MTU y subestaciones es bidireccional, con la diferencia de que estas últimas no pueden iniciar la comunicación, actuando, de esta manera, el MTU como cliente y las subestaciones como servidores. La mayoría del tráfico en redes SCADA se basa en peticiones de lectura de datos que solicita el MTU.

RTU (*Remote Terminal Unit* o *Unidad Terminal Remota*)

Se trata de dispositivos instalados en localizaciones remotas, que se encargan de recolectar y enviar la información que solicita la estación central o MTU, donde posteriormente será procesada. Estos elementos tienen el papel de esclavos dentro de la red.

PLC (*Controlador Lógico Programable* o *Programmable Logic Controller*)

Se trata de instrumentos que controlan y programan procesos en tiempo real. Utilizan memoria programable para guardar instrucciones que deben implementar, como puede ser abrir o cerrar una válvula, o medir una determinada temperatura. Una de las ventajas de estos dispositivos es que están diseñados para resistir condiciones desfavorables en cuanto a temperatura, impactos, ruido eléctrico o humedad. [7]

Además, se trata de instrumentos fácilmente programables. Todo esto hace que los PLC sean ampliamente utilizados a nivel industrial.

PAC (*Controlador de Automatización Programable* o *Programmable Automation Controller*)

Estos dispositivos combinan las funcionalidades de los PLC junto con la capacidad de monitorización y cálculo de un PC. Se utilizan en procesos como adquisición de datos de precisión, análisis matemático, control de movimiento y robótica, seguridad controlada, etc.[8]

HMI (*Interfaz hombre-máquina* o *Human Machine Interface*)

Es el software que permite la comunicación entre un operador humano y los distintos sistemas remotos de una red SCADA. Presenta la información recogida de manera gráfica,

para que sea entendible por el operador y permitiéndole así manejar datos y controlar diferentes procesos en tiempo real.

Todos estos dispositivos se interconectan entre sí a través de la red de comunicaciones, que permite el intercambio de información dentro de un sistema SCADA. Puede estar diseñada para permitir una comunicación por cable o inalámbrica. Aquí es donde entran en juego los protocolos de comunicación industriales. Este TFG se centra en Modbus TCP, pero otros protocolos también utilizados en redes SCADA son DNP3 o IEC 60870-5. [9]

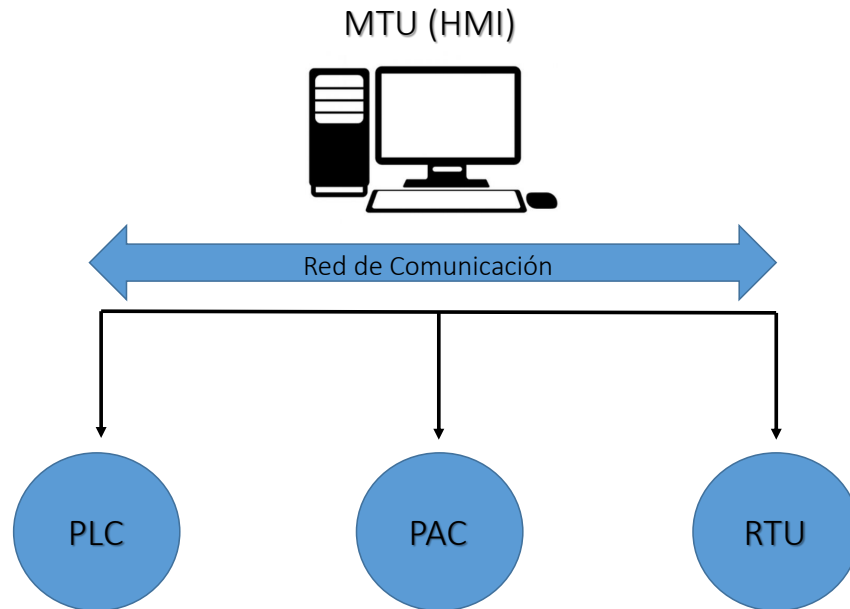


Figura 2.1: Arquitectura de una red SCADA.

2.3. Modbus TCP

Modicon Communication Bus, más conocido como Modbus es uno de los protocolos más usados en control de sistemas industriales. Fue desarrollado en 1979 por la empresa *Modicon* con el objetivo de dar soporte a la comunicación de sus PLCs. En el año 2004 fue liberado, pasando a ser un protocolo abierto y disponible sin necesidad de ningún tipo de licencia.

Si bien Modbus comenzó a utilizarse principalmente en comunicación por cable serie, actualmente existe una adaptación para comunicaciones sobre redes TCP/IP, pudiendo utilizarse otros medios de comunicación (Ethernet, WiFi, etc.) distintos al cable serie. [10]

A esta versión de Modbus sobre TCP/IP se le conoce de manera abreviada como Modbus TCP, y queda situado en el nivel 7 o nivel de aplicación en el Modelo OSI, que constituye un modelo descriptivo de la arquitectura de interconexión en los sistemas de comunicación.[10] En la Figura 2.2 se muestran los diferentes niveles que componen este modelo, y dónde se encuentra cada protocolo sobre los que se sitúa Modbus TCP.

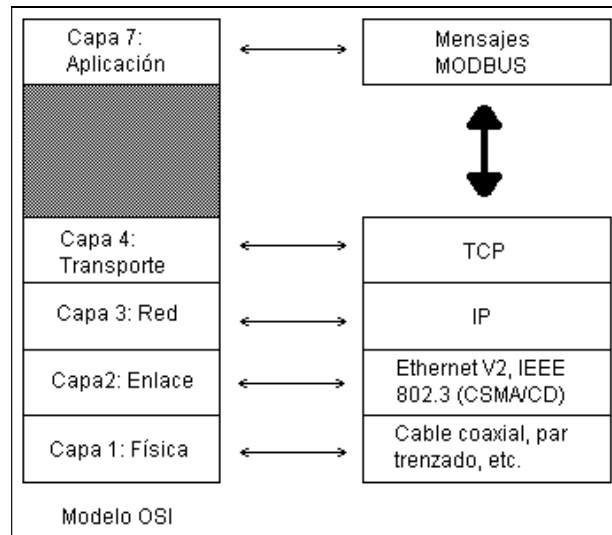


Figura 2.2: Capas del modelo OSI en Modbus TCP.

2.3.1. Modelo cliente-servidor

El modelo de comunicación cliente/servidor se trata de una arquitectura en la cual varios dispositivos informáticos se vinculan a través de una red. El cliente es el demandante, es decir, el encargado de iniciar siempre la comunicación, realizando peticiones periódicamente. El servidor no tiene la capacidad de iniciar la comunicación, sino que su función es la de esperar a recibir peticiones del cliente y responder a ellas proporcionando la información requerida.

Aunque no es lo más común en Modbus TCP, el servidor también puede designarse como esclavo, mientras que el cliente recibe también el nombre de maestro. El máximo número de servidores que puede haber en una red Modbus está limitado a 254. Modbus TCP permite que un cliente tenga múltiples transacciones y que un servidor participe en comunicaciones con múltiples clientes.

La comunicación entre dispositivos (esquemática en la Figura 2.3) se da sobre una red TCP/IP habitualmente en el puerto 502 y se basa en cuatro tipos de mensajes [11]:

- *Request* (Solicitud o petición): es el mensaje que el cliente envía a la red para iniciar una transacción.
- *Indication* (Notificación): es el mensaje de solicitud recibido en el lado del servidor.
- *Response* (Respuesta): es el mensaje de respuesta enviado por el servidor.
- *Confirmation* (Confirmación): es el mensaje de respuesta recibido en el lado del cliente.

El cliente también tiene la capacidad de enviar mensajes de difusión (*broadcast*), en cuyo caso, los servidores no envían respuesta.



Figura 2.3: Comunicación en Modbus TCP [1]

2.3.2. Estructura de la trama Modbus TCP

La trama Modbus TCP puede verse en la Figura 2.4. Tiene la misma estructura tanto en el caso de peticiones como de respuestas. Está compuesta por una cabecera llamada MBAP header (*Modbus Application Protocol header*) seguida de una unidad de datos de protocolo o PDU (*Protocol Data Unit*). Cada uno de estos bloques se divide en varios campos que veremos a continuación. [12]

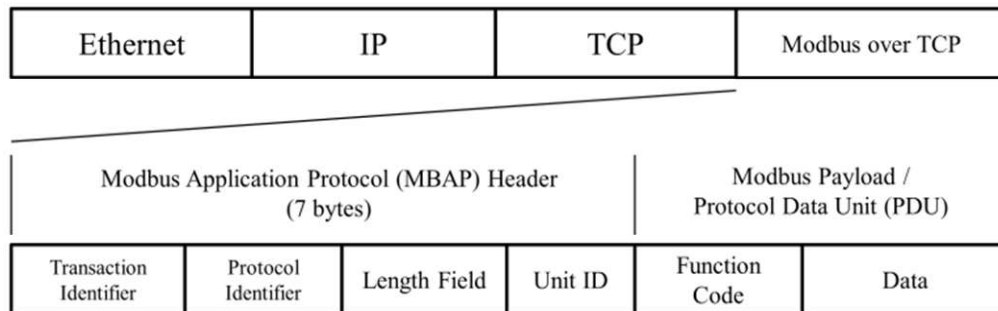


Figura 2.4: Estructura de la trama Modbus TCP [2]

La cabecera *MBAP* tiene un tamaño de 7 Bytes y está compuesta por cuatro campos:

- *Transaction Identifier* o Identificador de la transacción (2 Bytes): Es el primer campo de la cabecera. Permite a los dispositivos emparejar de manera correcta solicitudes y respuestas durante la comunicación. Este campo debe tener el mismo valor tanto en el cliente como en el servidor.
- *Protocol Identifier* o Identificador del protocolo (2 Bytes): Se trata del segundo campo de la cabecera. Indica el protocolo que está encapsulado en esta, siendo 0 el valor asignado para el protocolo Modbus TCP.
- *Length* o Longitud (2 Bytes): El tercer campo de la cabecera indica la longitud en Bytes de los campos restantes (identificador de unidad y PDU).
- *Unit ID* o Identificador de unidad (1 Byte): Es el último campo de la cabecera e indica cuál es el esclavo asociado a la transacción. Este campo toma valor en la petición del cliente y debe volver con el mismo valor en la respuesta del servidor.

La unidad de datos de protocolo (PDU) tiene longitud variable, aunque su longitud máxima son 253 Bytes. Está compuesta por dos campos:

- *Function Code* o Código de función (1 Byte): indica al servidor qué tipo de acción realizar. Los códigos de función válidos se encuentran en los rangos discontinuos [1, 64], [73, 99] y [111, 127]. Se trata de códigos públicamente documentados y validados por la comunidad Modbus.org. Dentro de estos rangos hay algunos valores que se encuentran reservados y no están disponibles para uso público, ya que son utilizados por algunas compañías para productos heredados.

Los códigos en los rangos [65, 72] y [100, 110] están definidos por cada usuario y no se consideran en el estándar Modbus. Los valores que se encuentren en el rango [128, 255] indican un error. En la Figura 2.5 se presenta una tabla con los códigos de función más utilizados en la comunicación Modbus.

| | | | | Function Codes | | |
|----------------------------|----------------------------------|---|-------------------------------|----------------|----------|-------|
| | | | | code | Sub code | (hex) |
| Data Access | Bit access | Physical Discrete Inputs | Read Discrete Inputs | 02 | | 02 |
| | | Internal Bits Or Physical coils | Read Coils | 01 | | 01 |
| | | | Write Single Coil | 05 | | 05 |
| | | | Write Multiple Coils | 15 | | 0F |
| | 16 bits access | Physical Input Registers | Read Input Register | 04 | | 04 |
| | | Internal Registers Or Physical Output Registers | Read Holding Registers | 03 | | 03 |
| | | | Write Single Register | 06 | | 06 |
| | | | Write Multiple Registers | 16 | | 10 |
| | | | Read/Write Multiple Registers | 23 | | 17 |
| | | | Mask Write Register | 22 | | 16 |
| | | | Read FIFO queue | 24 | | 18 |
| | File record access | Read File record | | 20 | | 14 |
| | | Write File record | | 21 | | 15 |
| | Diagnostics | Read Exception status | | 07 | | 07 |
| | | Diagnostic | | 08 | 00-18,20 | 08 |
| Get Com event counter | | 11 | | 0B | | |
| Get Com Event Log | | 12 | | 0C | | |
| Report Server ID | | 17 | | 11 | | |
| Read device Identification | | 43 | 14 | 2B | | |
| Other | Encapsulated Interface Transport | | 43 | 13,14 | 2B | |
| | CANopen General Reference | | 43 | 13 | 2B | |

Figura 2.5: Tabla con los códigos de función públicos más comunes. [3]

En la respuesta del servidor, se usa el campo del código de función para indicar una respuesta positiva o si ha ocurrido un error. En caso de respuesta positiva, el servidor repite el código de función original. En caso de error, al código de función de la petición se le suma el valor decimal 128, de ahí que recibir una respuesta con un código entre 128 y 255 indique que se ha producido un error.

- Datos (longitud variable): contiene la información adicional que el servidor utiliza para ejecutar la acción definida por el código de función. Puede no existir. En este caso el servidor no requiere información adicional ya que el código de función por sí solo especifica la acción.

Dado que la cabecera tiene una longitud de 7 Bytes, y la PDU tiene un tamaño máximo de 253 Bytes, el tamaño máximo que podrá tener una trama Modbus TCP es de 260 Bytes. [3]

2.3.3. Vulnerabilidades

Los protocolos industriales fueron inicialmente diseñados para sistemas que se encontraban aislados de la red, interconectados entre sí a través de cableado eléctrico, por lo que los escenarios de ataques cibernéticos se consideraban improbables, siendo así que la seguridad en los sistemas SCADA no suponía apenas una preocupación. Con el paso de los años, con la apertura de sus conexiones hacia redes externas como Internet y la liberación de los estándares haciéndolos abiertos y accesibles por cualquier persona, ha traído como consecuencia la aparición de numerosas vulnerabilidades en redes SCADA, ante las cuales se encuentran desprotegidas. Hoy en día, una gran proporción de infraestructuras críticas está bajo el control de sistemas SCADA. Las implicaciones de un ataque cibernético en estas infraestructuras pueden suponer un importante impacto sobre una región o país, por lo que es importante desarrollar mecanismos que protejan a estas redes de posibles ataques.

Centrándonos en el caso del protocolo Modbus, la versión Modbus TCP surgió a partir de la necesidad de conectar los diferentes dispositivos dentro de una red SCADA usando tecnologías posteriores a los cables serie. A la trama original de Modbus se le añadió una cabecera con los campos necesarios para su funcionamiento en redes abiertas, aunque como su predecesor, no cuenta con mecanismos de defensa ante posibles ataques.

Los ataques que pueden afectar a Modbus TCP se dividen en cuatro categorías [13]:

- Reconocimiento. Consiste recopilar diferentes tipos de información sobre la red. Algunos ataques conocidos de este tipo son el escaneo de direcciones y códigos de función para construir una lista de aquellos que están implementados dentro de una red. Otro tipo de ataque trata de identificar las características de cada dispositivo, como el fabricante, el modelo o los protocolos de red admitidos.
- Inyección de respuestas. Intentan presentar información maliciosa través de la inyección de tráfico anómalo en la red. Estos ataques pueden producirse de 3 formas distintas: a partir del control de un PLC o RTU, capturando paquetes durante la transmisión y alterando su contenido y, por último, mediante la inyección de paquetes por un dispositivo que se encuentre fuera de la red.
- Inyección de comando. En estos ataques se inyectan comandos inválidos que causan acciones de control anormales para llevar al sistema de un estado seguro a un estado crítico.
- Denegación de servicio. Los ataques de denegación de servicio intentan interrumpir o romper los enlaces de comunicación debido a la inyección de grandes volúmenes de tráfico en la red para acabar con el control y la supervisión de un sistema.

2.4. Sistema de Detección de Intrusiones

Un Sistema de Detección de Intrusiones o IDS (*Intrusion Detection System*) consiste en una herramienta de seguridad que identifica intentos no autorizados de manipulación o acceso a un sistema mediante el análisis de diversas áreas dentro de un dispositivo o una red.

En función de donde se encuentre situado el detector de intrusiones, podemos distinguir principalmente dos tipos de IDS [14]:

- NIDS (*Network-Intrusion Detection System*): Su función es la de capturar el tráfico dentro de una red y analizar los campos de cada paquete capturado, para, en base a unas reglas y patrones establecidos, decidir si se trata de tráfico anómalo o tráfico legítimo, informando al usuario en caso de detectar alguna anomalía. Por este motivo necesitan estar instalados en máquinas que tengan acceso completo a la red que vayan a monitorizar.
- HIDS (*Host-Intrusion Detection System*): Se encuentran integrados en un *host* (como puede ser un servidor o un PC). Operan sobre la información almacenada en el equipo, realizando acciones como monitorizar el tráfico que pasa por este o analizar si han sido alterados distintos archivos que no deben ser modificados, para así ver qué procesos están involucrados en un ataque.

Por otro lado, también podemos clasificarlos según las técnicas que utilizan para analizar el tráfico [15]:

- Detección por firma (*Signature-based IDS*): El IDS guarda información sobre ataques pasados en forma de patrones, conocidos también como firmas (*signatures*). Se trata de encontrar la presencia de estos patrones o firmas en el tráfico de la red.

Los detectores por firma son muy efectivos en la detección de ataques sin generar un número elevado de falsas alarmas. Por contra, tienen como desventaja que solo son capaces de detectar aquellos ataques que conocen, por lo que deben ser actualizados continuamente con firmas sobre nuevos ataques.

- Detección por anomalía (*Anomaly-based IDS*): Se basa en la identificación de comportamientos inesperados dentro de la red, asumiendo como ataque cualquier actividad diferente a lo usual. Los detectores por anomalía recogen información durante un largo periodo de tiempo del comportamiento normal de usuarios, *host*, conexiones de red, etc.

En base a esta información, se construye un perfil que representa actividad de tráfico normal, de manera que cualquier comportamiento que no forme parte de este perfil será considerado como un ataque.

Estos IDS tienen como ventaja principal la capacidad de reconocer ataques sin necesidad de tener un conocimiento específico sobre ellos. Sin embargo, producen un gran número de falsas alarmas debido a que en ocasiones tienen lugar comportamientos no predecibles en el tráfico.

El tipo de IDS que se ha desarrollado en este TFG es un NIDS con detección por anomalía, que, además de analizar los distintos campos de los paquetes capturados, también informa sobre el número de maestros presentes en la red, y la tasa de transmisión de paquetes, alertando en todo momento si se traspasa un umbral de paquetes transmitidos en un determinado periodo de tiempo.

2.5. Conclusiones

En este capítulo se ha realizado un análisis de las redes SCADA, su arquitectura y su importancia en el mundo actual. Se ha detallado la estructura de la cabecera Modbus, así como todos los campos que la componen y el modelo de comunicación que utiliza el protocolo. También se ha realizado un análisis de la situación actual en cuanto a las vulnerabilidades existentes en los protocolos industriales, haciendo especial hincapié en aquellas que afectan a Modbus TCP. Por último, se ha hecho un resumen sobre los distintos IDS que existen en función de dónde se encuentren ubicados y las diferentes técnicas de análisis que aplican, concretando cuál será el tipo de IDS desarrollado en este Trabajo de Fin de Grado.

3

Desarrollo

3.1. Introducción

Una vez conocido el funcionamiento tanto del protocolo Modbus como el de un detector de intrusiones, en este capítulo se procede a la explicación de las distintas fases llevadas a cabo en la implementación del IDS para este protocolo, comenzando primero por el entorno de desarrollo creado y la configuración necesaria para un correcto funcionamiento del mismo, y continuando con un resumen del procedimiento utilizado para la disección y análisis del tráfico de red. Por último, se muestran cuáles son los resultados que ofrece el IDS tras su ejecución.

3.2. Entorno de desarrollo

El desarrollo del código de este TFG se ha realizado en una máquina virtual que posee Ubuntu como sistema operativo. Para la posterior validación ha sido necesaria la instalación de dos máquinas virtuales más, que cuentan con Kali Linux como distribución, debido a que están equipadas con diversas herramientas orientadas al análisis de seguridad en redes.

A las tres máquinas virtuales se les han asignado 2 GB de memoria para su correcto funcionamiento y se encuentran alojadas en la plataforma de virtualización de escritorios VMware Workstation 11. Esta aplicación permite ejecutar e interconectar entre sí varias máquinas virtuales. Para dicha interconexión, es necesario configurar en todas las máquinas la opción de 'Adaptador de Red' con el modo NAT (*Network Address Translation*), tal y como se muestra en la Figura 3.1.

El lenguaje de programación elegido para el desarrollo del IDS ha sido *C*, debido a que se trata de un lenguaje que posee ventajas con respecto a otros en cuanto a velocidad de ejecución y consumo de recursos, ya que el objetivo es que el programa procese el tráfico de red a alta velocidad.

Para la captura y el procesamiento de paquetes, ha sido necesaria la instalación de la herramienta libpcap. Se trata de una biblioteca en C que ofrece al programador la posibilidad de capturar tráfico de la red. El comando para su instalación es `apt-get install libpcap`.

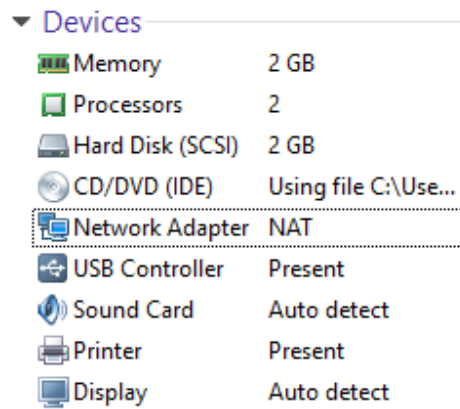


Figura 3.1: Configuración modo NAT.

3.3. Análisis del tráfico

El sistema que se ha seguido para el análisis de paquetes es la aplicación de filtros, es decir, si un paquete cumple con las características requeridas irá pasando por los distintos filtros mientras los diferentes campos que lo componen son extraídos para su análisis. De lo contrario, el paquete será descartado.

Los paquetes pueden descartarse dos motivos:

- Por no cumplir con las características requeridas para ser analizados, es decir, no se corresponden con la estructura de cabeceras Ethernet, IPv4, TCP, Modbus. Los paquetes que no incluyan cabecera Modbus pero tengan como destino u origen el puerto TCP 502 también serán contabilizados.
- Si a pesar de cumplir con las características de cabeceras requeridas, se detecta alguna anomalía en algún campo analizado del paquete. En este caso se dará aviso de que el paquete posee un campo anómalo y dejará de ser analizado.

La ejecución del programa tiene la posibilidad de comenzar de dos formas distintas en función de los parámetros introducidos por línea de comandos.

- La primera, (Figura 3.2) a través de la función `pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *errbuf)`, que permite la captura de tráfico en tiempo real. Posteriormente, el tráfico capturado se guarda en una traza de extensión `.pcap` con la función `pcap_open_dead(int linktype, int snaplen)`.


```
lubuntu@lubuntu:~/Desktop/TFG$ sudo ./ analisisModbus -i eth0
```

Figura 3.2: Ejecución del programa mediante captura de tráfico.

- La segunda posibilidad de ejecución (Figura 3.3) consiste en la lectura de una traza ya existente. Esto se consigue gracias a la función `pcap_open_offline(const char *fname, char *errbuf)`. Para la realización de pruebas y validación de este programa, este será el método de ejecución utilizado.

```
lubuntu@lubuntu:~/Desktop/TFG$ ./ analisisModbus -f traza_modbus.pcap
```

Figura 3.3: Ejecución del programa mediante la lectura de una traza.

Independientemente de cuál sea el modo de ejecución, el tráfico es analizado con la función `pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)`. Esta función invoca a una rutina (que se corresponde con el tercer parámetro) que indique el programador, y se ejecuta cada vez que se capture un paquete.

Para el almacenamiento de los datos de algunos campos de cada paquete, se ha decidido optar por la implementación de tablas *hash*. Se trata de estructuras de datos que permiten el almacenamiento de datos así como también su búsqueda a partir de una clave generada. El motivo de esta elección se debe a que tienen como principal ventaja un acceso muy rápido a los datos guardados. Esta característica es de vital importancia ya que queremos que el código sea capaz de procesar el mayor número de paquetes por unidad de tiempo.

Cuando se implementan tablas *hash*, es muy frecuente que se produzcan colisiones al tratar de almacenar datos en una posición ya ocupada. Por eso ha sido necesario implementar mecanismos para su resolución. El método elegido ha sido la utilización de listas enlazadas. En la Figura 3.4 se muestra un esquema sobre el modelo utilizado.

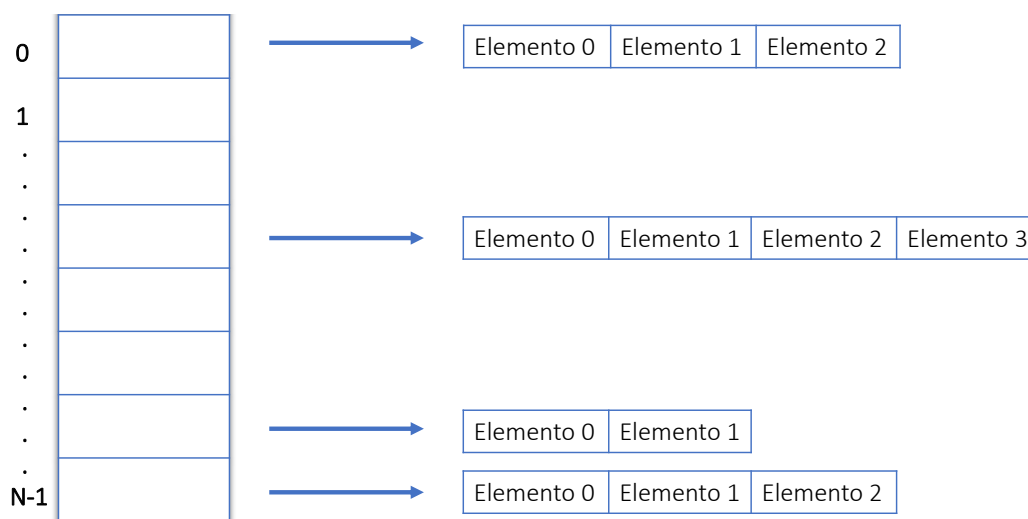


Figura 3.4: Tabla *hash* con listas enlazadas.

Cada índice de la tabla apunta al comienzo de una lista enlazada. Así, cada elemento que va llegando a una misma posición avanza por la lista hasta llegar al final, creándose una nueva posición y almacenándose en ese lugar, evitando de esta manera, que se produzcan colisiones.

En este desarrollo se han utilizado tablas *hash* para la implementación de diferentes procesos. El primero de ellos consiste en una tabla en la que se muestra el número de peticiones o respuestas que ha enviado cada dispositivo de la red, sea cliente o servidor, haciendo una distinción según el código de función.

Este número de peticiones o respuestas no tiene por qué coincidir necesariamente con el número de mensajes enviados, ya que un mismo paquete puede contener varias tramas Modbus con diferente código de función en cada una de ellas. Es decir, no se contabiliza el número de paquetes intercambiados, sino la cantidad de tramas Modbus enviadas por cada dispositivo, agrupadas por código de función, ya que esto ofrece mayor utilidad. Esto se consigue con una estructura en la que se almacenan los datos mostrados en la Figura 3.5.

```
typedef struct DataNode {
    uint8_t ipsrc[IP_ALEN];
    u_char function_code;
    int counter;
    struct DataNode* next;
}DataNode;
```

Figura 3.5: Estructura para la muestra de tramas por dirección IP.

Otra razón para la utilización de tablas *hash* es llevar a cabo un seguimiento del número de clientes o maestros presentes en la red (Figura 3.6). Para esto, se almacena la dirección IP perteneciente a cada dispositivo y el puerto TCP al que envían cada paquete. Se considerará que se trata de un dispositivo cliente cuando el puerto TCP destino sea el 502.

```
typedef struct IPnode {
    uint8_t ipsrc[IP_ALEN];
    uint8_t ipdst[IP_ALEN];
    struct IPnode* next;
}IPnode;
```

Figura 3.6: Estructura para la muestra del número de clientes o maestros.

También se han utilizado tablas *hash* para almacenar el identificador de la transacción o *transaction identifier* de cada paquete, lanzando un mensaje por pantalla en caso de que aparezca en más de dos paquetes (petición y respuesta). La estructura implementada para esta funcionalidad aparece en la Figura 3.7.

```
typedef struct table_id {
    u_short id;
    int counter;
    struct table_id* next;
}table_id;
```

Figura 3.7: Estructura para la muestra de IDs repetidos.

Por último, otra de las funcionalidades clave en este TFG consiste en llevar un control del número de peticiones enviadas por cada cliente en un intervalo de tiempo determinado. El objetivo es detectar si se produce una saturación de peticiones en la red, lo cual podría identificarse como un ataque de denegación de servicio (*Denial of Service*), o por el contrario, se producen menos envíos de los habituales, lo cual también puede considerarse una amenaza para el sistema. En la Figura 3.8 se observa la estructura utilizada para el desarrollo de esta funcionalidad:

```
typedef struct peticion {
    uint8_t ipsrc[IP_ALEN];
    int num_pet;
    double media;
    double media_anterior;
    double desviacion;
    double desviacion_anterior;
    struct peticion* next;
}peticion;
```

Figura 3.8: Estructura para el cálculo de peticiones por unidad de tiempo.

El método utilizado para llevar a cabo este seguimiento se trata del modelo de suavizado exponencial *Holt-Winters* [16]. Se basa en el cálculo de una media y desviación típica para establecer unos umbrales máximo y mínimo, entre los cuales tiene que encontrarse el número de peticiones enviadas por cada cliente. En caso de que el número de peticiones quede fuera de este umbral, se notificará mediante un aviso por pantalla.

Esta técnica tiene como principal ventaja que tanto la media como la desviación típica son dinámicas, es decir, estos valores son continuamente recalculados para ajustarse a las fluctuaciones que pueda tener el tráfico. Cabe destacar que el tráfico en redes SCADA se mantiene estable a lo largo del tiempo, presentando fluctuaciones pequeñas, por lo que esta medida es perfecta para detectar desviaciones que se salgan de esta estabilidad.

En este cálculo se distinguen dos términos. El primero de ellos se corresponde al valor actual y el segundo, a la media o desviación acumuladas. A cada uno de ellos se le asigna un peso, a elección en función de la prioridad que se le quiera dar a cada término, aunque normalmente se le asigna más peso al valor acumulado que al valor actual.

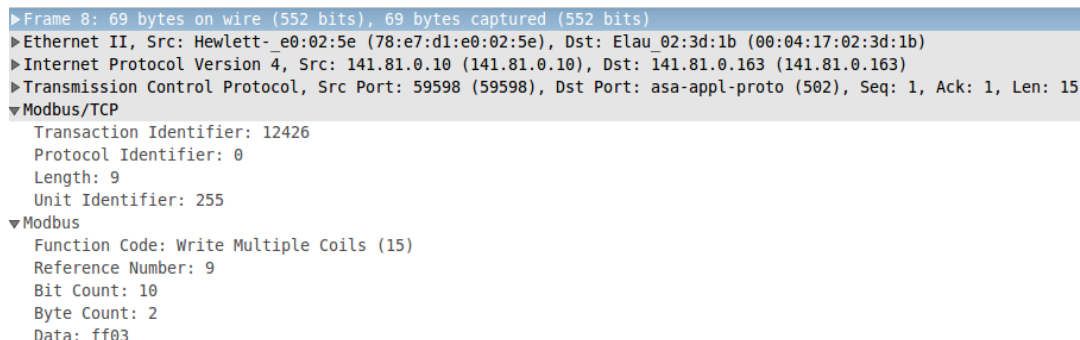
```
media = alpha*(n->num_pet) + (1-alpha)*media;
desviacion = beta * abs(media - (n->num_pet)) + (1 - beta) * desviacion;
```

Figura 3.9: Fórmulas aplicadas para el suavizado exponencial.

3.4. Mecanismos de comprobación de anomalías

Como se ha comentado anteriormente, la rutina a la que se llama con cada paquete capturado se basa en una serie de filtros, según los cuales los paquetes son o no descartados, o detectados como anómalos. Esta rutina consiste en diseccionar los diferentes campos de cada paquete, decidiendo cuáles de ellos son relevantes para el análisis, para así analizar si el paquete presenta alguna anomalía o se trata de tráfico legítimo.

La Figura 3.10 muestra la estructura típica de un paquete Modbus TCP, junto con los campos que componen la cabecera Modbus.



```
▶ Frame 8: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface 0
▶ Ethernet II, Src: Hewlett-e0:02:5e (78:e7:d1:e0:02:5e), Dst: Elau_02:3d:1b (00:04:17:02:3d:1b)
▶ Internet Protocol Version 4, Src: 141.81.0.10 (141.81.0.10), Dst: 141.81.0.163 (141.81.0.163)
▶ Transmission Control Protocol, Src Port: 59598 (59598), Dst Port: asa-appl-prot (502), Seq: 1, Ack: 1, Len: 15
▼ Modbus/TCP
  Transaction Identifier: 12426
  Protocol Identifier: 0
  Length: 9
  Unit Identifier: 255
▼ Modbus
  Function Code: Write Multiple Coils (15)
  Reference Number: 9
  Bit Count: 10
  Byte Count: 2
  Data: ff03
```

Figura 3.10: Captura de Wireshark con la estructura de un paquete.

Cabecera Ethernet

La función comienza con el análisis de la cabecera Ethernet, compuesta por las direcciones MAC origen y destino y el tipo. A continuación se hace una comprobación de si el paquete posee la etiqueta VLAN, que se trata de un campo opcional, para después comenzar con el análisis de la cabecera IP.

Cabecera IP

Los requisitos que debe cumplir la cabecera IP para que el paquete no sea descartado son las siguientes:

- La versión del protocolo debe ser IPv4.
- El tamaño de la cabecera debe tener un tamaño mayor o igual a 20 Bytes.
- El *offset* o desplazamiento debe de ser 0, de lo contrario el paquete es descartado y se pasa a analizar el siguiente.
- El tiempo de vida (*TTL* o *time to live*) indica el máximo número de nodos que un paquete puede atravesar. Este campo debe ser mayor a 1.

Si la cabecera cumple estas características, se extraen los campos correspondientes a las direcciones IP origen y destino, que serán almacenados en diferentes estructuras para su utilización en tablas *hash*, al ser de utilidad a la hora de presentar información

relevante para el usuario, como el número de mensajes (tanto peticiones como respuestas) enviados por cada dispositivo dentro de la red, el número de maestros que se encuentran en dicha red y, por último, la comprobación de que el envío de paquetes se produce a una tasa más o menos constante.

Cabecera TCP

A continuación, se procede a comprobar que la estructura del paquete continúa con la cabecera TCP. Esto se comprueba analizando que el campo protocolo de la cabecera IP sea igual a 6. De ser así, el requisito que debe cumplirse para que el paquete no sea descartado es que el tamaño de la cabecera TCP no sea menor a 20 Bytes.

En esta cabecera se encuentran los flags SYN y FIN. El flag SYN se activa en caso de querer establecer una nueva conexión, mientras que el flag FIN se activa para el cierre de una conexión. El programa muestra por pantalla si hay algún paquete que tenga alguno de estos flags activados.

Cabecera Modbus

Después de extraer la cabecera TCP, se filtra el tráfico para distinguir aquel que se trate de Modbus TCP. Este protocolo se corresponde con el puerto TCP origen o destino 502 (el cliente lo utiliza como puerto destino y el servidor como puerto origen).

Una vez analizada la cabecera TCP, queda la estructura de la trama Modbus, cuyos campos han quedado definidos en el Estado del Arte.

Un mismo paquete puede contener varias tramas Modbus con diferente código de función en cada una de ellas. Por este motivo, es necesario que el programa cuente con un bucle que lea todos los bytes de la trama Modbus hasta llegar a la última cabecera.

```
► Ethernet II, Src: Elau_02:3d:1b (00:04:17:02:3d:1b), Dst: Hewlett-e0:02:5e (78:e7:d1:e0:02:5e)
► Internet Protocol Version 4, Src: 141.81.0.163 (141.81.0.163), Dst: 141.81.0.10 (141.81.0.10)
► Transmission Control Protocol, Src Port: asa-appl-proto (502), Dst Port: 59598 (59598), Seq: 13, Ack: 56, Len: 36
► Modbus/TCP
► Modbus
► Modbus/TCP
▼ Modbus
  Function Code: Write Multiple Coils (15)
  Reference Number: 6
  Bit Count: 1
► Modbus/TCP
▼ Modbus
  Function Code: Read Coils (1)
  Byte Count: 3
  Data: 01ff07
```

Figura 3.11: Ejemplo de paquete con varias tramas Modbus TCP.

Como se puede observar en la Figura 3.11, se muestra la estructura de un paquete que contiene 3 tramas Modbus TCP, con sus correspondientes cabeceras y PDUs.

El requisito indispensable que debe cumplir esta cabecera, es que el campo *Protocol Identifier* o Identificador del protocolo sea 0, el valor asignado al protocolo Modbus TCP. De lo contrario, será detectado como anomalía y se descartará el paquete.

Otros motivos por los que un mensaje puede considerarse anómalo son los detallados a continuación:

- Si el campo Longitud o *Length* es menor que 2 (el tamaño mínimo de la PDU es 1 Byte y a este hay que sumarle otro Byte del Identificador e Unidad o *Unit ID*) o mayor que 254 (tamaño máximo de la PDU más el Identificador de Unidad).
- Si el código de función toma el valor 0, ya que este valor no se considera válido.
- Si el código de función toma un valor entre 128 y 255, ya que indican que se ha producido un error.
- Si el código de función toma un valor reservado y no disponible para uso público.

Si el paquete consigue superar todas estas comprobaciones, podemos afirmar que no presenta ninguna anomalía, y en caso de haberlos, se procede a analizar el resto de los datos que componen el paquete.

3.5. Visualización de los resultados

En la visualización de los resultados de la ejecución, además de notificar al usuario si se ha detectado alguna anomalía en paquetes o en la tasa de tráfico, pueden distinguirse distintos apartados que se explicarán a continuación.

Se comienza con una serie de líneas en las que se realiza un resumen del contenido de la traza, detallando el número de paquetes que contiene, qué tipo de protocolos contienen, y en el caso de aquellos paquetes con cabecera Modbus TCP, se proporciona información sobre el número de peticiones de lectura y escritura, y si ha habido algún paquete anómalo o que utilice un código de función no válido (reservado). También se muestra el número de maestros presentes en la red.

A continuación, se muestra una tabla en la que aparecen tres columnas. La primera de ellas se corresponde la dirección IP origen del dispositivo, ya sea maestro o esclavo; la segunda columna es el código de función, y la tercera se trata de un contador sobre el número de veces que cada código de función ha sido enviado por la IP correspondiente.

Después de la tabla, se muestran los IDs que aparecen en la traza más de 2 veces, indicando el número de veces que aparecen.

Para terminar, la última información que proporciona el programa consiste en varias medidas de rendimiento del código: tiempo de ejecución, paquetes por segundo (pps) analizados y throughput.

3.6. Conclusiones

Mediante los pasos y requisitos detallados en los apartados anteriores, se ha llevado a cabo el diseño y desarrollo un sistema de detección de intrusiones, tratando de que sea lo más eficiente posible. Se han extraído los datos de cada paquete capturado, haciendo uso de estructuras para tablas *hash* con listas enlazadas para lograr un mejor tiempo de ejecución del código. Como resultado, se obtiene un IDS capaz de distinguir distintos tipos de anomalías tanto en la estructura de los paquetes como en la tasa de envío y recepción de los mismos. En el siguiente capítulo se realizarán las pruebas necesarias para la comprobación de su efectividad.

4

Validación

4.1. Introducción

Una vez explicado el funcionamiento del detector de intrusiones desarrollado, en este capítulo se validará su funcionalidad y efectividad. Estas pruebas son fundamentales en la implementación del IDS para verificar su correcto funcionamiento y analizar mejoras que puedan introducirse como trabajo futuro. Con el fin de poder analizar resultados en tráfico de una red real, se ha facilitado el acceso a una traza por parte de la Cátedra UAM-Naudit.

4.2. Validación de funcionalidad

Para validar la correcta funcionalidad del IDS, se han llevado a cabo varias pruebas, las cuales serán detalladas en los siguientes apartados.

4.2.1. Pruebas iniciales

Las primeras pruebas de verificación han sido llevadas a cabo mediante la obtención de una traza encontrada en la plataforma Github.¹

Esta traza se ha utilizado durante el desarrollo del IDS para tener una referencia del correcto funcionamiento del mismo, comprobando que la extracción y lectura de los diferentes campos de cada paquete se estaba realizando correctamente y poder tener también una primera idea sobre el tiempo de ejecución del programa.

A continuación se muestran los resultados que ofrece la ejecución del código con esta traza. Debido a que la salida por pantalla de esta ejecución presenta demasiadas líneas

¹Link a la traza: <https://github.com/ITI/ICS-Security-Tools/tree/master/pcaps/ModbusTCP>

para ser plasmada en esta memoria, se ha optado por mostrar los resultados de una manera simplificada en la Tabla 4.1.

Tabla 4.1: Resultado de la ejecución de la traza de Github.

```
Se procesaron 2727 paquetes, de los cuales hay:
2727 paquetes IPv4.
2727 paquetes TCP.
2722 paquetes Modbus TCP, de los cuales:
    1310 peticiones de lectura.
    498 peticiones de escritura.
    0 paquetes con datos erróneos.
    0 respuestas con excepción.
    0 paquetes con código de función no válido (reservado).
Número de maestros en la red: 1

Flag SYN activado en 2 paquetes.
Flag FIN activado en 2 paquetes.
```

| IP | Function Code | Cantidad |
|--------------|---------------|----------|
| 141.81.0.10 | 1 | 345 |
| 141.81.0.10 | 2 | 362 |
| 141.81.0.10 | 4 | 603 |
| ... | | |
| 141.81.0.163 | 15 | 70 |
| 141.81.0.164 | 15 | 28 |

```
El transaction id 2956 aparece 3 veces.
El transaction id 3057 aparece 3 veces.
...
El transaction id 22508 aparece 6 veces.
El transaction id 22509 aparece 6 veces.

El programa ha tardado 0.006075 segundos en ejecutar.
Se procesaron 448888.888889 pps
El throughput es de 38761152 Bytes/segundo.
```

Las comprobaciones realizadas con esta traza han sido de gran utilidad, sin embargo, resultan ser insuficientes, pues se trata de una traza con un número muy reducido de paquetes, que nada tiene que ver con el volumen de tráfico que aparece en una red real que utilice este protocolo.

Se puede apreciar que el tiempo de ejecución es muy bajo, pero, por la razón citada anteriormente, se necesita contar con una traza que posea un número mucho mayor de paquetes, para verificar que tenga un correcto rendimiento.

Herramienta Tshark

Tshark se trata de una herramienta capaz de analizar el tráfico de red que permite obtener datos de paquetes en tiempo real o a partir de capturas previamente guardadas [17]. Es capaz de analizar los mismos archivos que son compatibles con Wireshark y además utilizan la misma sintaxis, con la diferencia de que Tshark funciona por línea de comandos. Para instalarlo sólo se necesita ejecutar el siguiente comando en la terminal: `sudo apt-get install tshark`.

Este software ha sido utilizado con el propósito de poder comparar que los resultados obtenidos con el IDS son los mismos que los resultados que arroja Tshark a partir del análisis de la traza obtenida en Github.

A continuación, en las Figuras 4.1, 4.2 y 4.3 se muestran algunos ejemplos de pruebas realizadas con Tshark, que corroboran que los resultados ofrecidos por el detector son correctos. Pueden verse los resultados de la ejecución del IDS en la Tabla 4.1.

```
lubuntu@lubuntu:~/Desktop/TFG$ tshark -r traza_modbus.pcap -Y "modbus.func_code=1 && ip.src == 141.81.0.10" -T fields -e modbus.func_code | awk '{split($1,a,",");for (i in a) print a[i];}' | sort | uniq -c
   345 1
     1 2
```

Figura 4.1: Resultado de Tshark para la IP 141.81.0.10 y código de función 1.

```
lubuntu@lubuntu:~/Desktop/TFG$ tshark -r traza_modbus.pcap -Y "modbus.func_code=2 && ip.src == 141.81.0.10" -T fields -e modbus.func_code | awk '{split($1,a,",");for (i in a) print a[i];}' | sort | uniq -c
     1 1
    13 15
   362 2
    223 4
```

Figura 4.2: Resultado de Tshark para la IP 141.81.0.10 y código de función 2.

```
lubuntu@lubuntu:~/Desktop/TFG$ tshark -r traza_modbus.pcap -Y "modbus.func_code=15 && ip.src == 141.81.0.164" -T fields -e modbus.func_code | awk '{split($1,a,",");for (i in a) print a[i];}' | sort | uniq -c
    28 15
```

Figura 4.3: Resultado de Tshark para la IP 141.81.0.164 y código de función 15.

La columna de la derecha muestra el código de función y la columna de la izquierda, el número de veces que aparece enviado por la IP introducida en el comando. Como se ha explicado anteriormente, el número de veces que aparece no tiene por qué coincidir con el número de paquetes.

De esta manera, tomando como ejemplo la Figura 4.2, buscamos conocer el número de veces que la IP 141.81.0.10 ha enviado tramas Modbus con el código de función 2. Podemos observar que el resultado es que este código aparece 362 veces. El resto de valores que se muestran se debe a que los paquetes enviados por esa IP que contienen el código de función 2, poseen más tramas Modbus con códigos de función diferentes.

4.2.2. Pruebas con Modpoll, Metasploit y ModbusPal

Antes de comenzar con los resultados de las pruebas realizadas con estas herramientas, se procede a la definición de las mismas.

- ModbusPal: se trata de un simulador de servidor o esclavo Modbus escrito en Java, gratuito y de código abierto. Cuenta con una interfaz sencilla (Figura 4.4) y posee el modo *Learn* en el que a medida que recibe solicitudes Modbus del maestro, crea dinámicamente los recursos que faltan: esclavos, registros, etc.

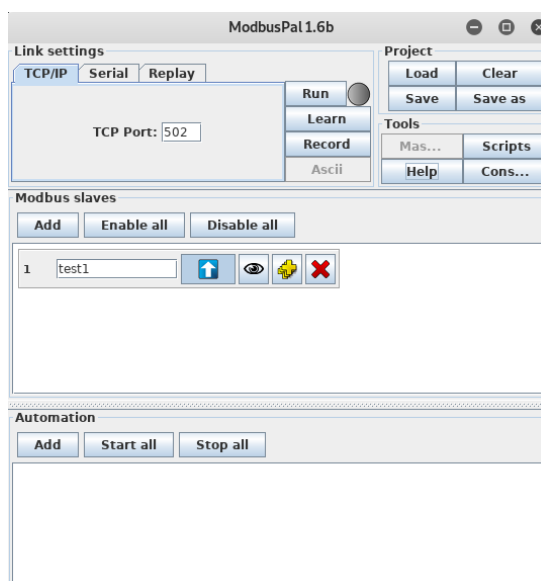


Figura 4.4: Interfaz del simulador de servidor ModbusPal.

El comando para la ejecución de ModbusPal es el siguiente:

```
root@kali:~/Desktop# sudo java -jar ModbusPal.jar
```

Figura 4.5: Comando de ejecución de ModbusPal.

- Modpoll: es un simulador de cliente o maestro que funciona por línea de comandos. Al igual que ModbusPal, es un software gratuito y está disponible para varios sistemas operativos.
- Metasploit Framework: se trata de una herramienta que permite llevar a cabo diferentes pruebas para encontrar vulnerabilidades en un sistema remoto a través de la ejecución de *exploits*.

Para el uso de estas tres herramientas, ha sido necesaria la creación de dos máquinas virtuales que contaran con la distribución Kali Linux, además de la máquina virtual inicial con Ubuntu en la que se desarrolló el código.

Una vez configuradas correctamente y conectadas entre sí las tres máquinas virtuales, se procede a la ejecución de un programa distinto en cada una de ellas.

Las dos máquinas con Kali Linux actuarán como clientes, en una ejecutándose Metasploit y en otra Modpoll, mandando peticiones a la tercera máquina virtual, en la cual se estará ejecutando el servidor ModbusPal.

La manera de capturar el tráfico intercambiado entre las diferentes máquinas virtuales, es a través del uso de Wireshark en el sistema anfitrión. En este programa se debe seleccionar la interfaz de red Vmnet8, como se observa en la Figura 4.6, que es exclusiva de VMware y se crea automáticamente con la instalación de VMware Workstation.

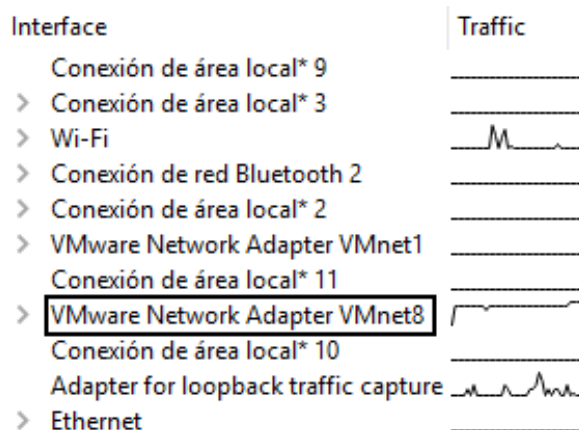


Figura 4.6: Interfaz de captura de tráfico Vmnet8.

Pruebas con Modpoll

El comando de ejecución de Modpoll se compone de varios parámetros y tiene la siguiente estructura:

```
lubuntu@lubuntu:~/Desktop/modpoll/linux_i386$ ./modpoll 192.168.244.133 -m tcp -a1 -c3 -t4
```

Figura 4.7: Comando de Modpoll para petición de lectura.

El primer parámetro a introducir es la dirección IP de la máquina virtual en la que se encuentra el simulador de esclavo ModbusPal. El siguiente parámetro **-m tcp** indica la versión del protocolo que se desea utilizar, en este caso Modbus TCP.

Los parámetros **-a#**, **-c#** y **-t#** se corresponden respectivamente con el número de esclavo al que va dirigido el paquete, el número de registros sobre los que se va a leer (entre 1 y 255) y la acción a realizar.

Modpoll también permite la opción de escribir en múltiples registros (código de función 16) mediante el uso del comando mostrado en la Figura 4.8, en la que se indica la orden de escribir en tres registros los valores que aparecen en el comando.

```
lubuntu@lubuntu:~/Desktop/modpoll/linux_i386$ ./modpoll 192.168.244.133 -m tcp 1 0 1
```

Figura 4.8: Comando de Modpoll para petición de escritura.

Tras ejecutar varios comandos cambiando los parámetros para abarcar el máximo de funciones y variables posibles, se obtiene una traza, la cual, tras ser analizada por el IDS muestra los resultados de la Tabla 4.2:

Tabla 4.2: Resultados tras la ejecución con Modpoll.

```
Se procesaron 1076 paquetes, de los cuales hay:
1076 paquetes IPv4.
1076 paquetes TCP.
356 paquetes Modbus TCP, de los cuales:
    145 peticiones de lectura.
    33 peticiones de escritura.
    0 paquetes con datos erróneos.
    0 respuestas con excepción.
    0 paquetes con código de función no válido (reservado).
Número de maestros en la red: 1

Flag SYN activado en 114 paquetes.
Flag FIN activado en 116 paquetes.
```

| IP | Function Code | Cantidad |
|-----------------|---------------|----------|
| 192.168.244.133 | 1 | 41 |
| 192.168.244.133 | 3 | 104 |
| ... | | |
| 192.168.244.142 | 3 | 104 |
| 192.168.244.133 | 16 | 33 |
| 192.168.244.142 | 16 | 33 |

```
El transaction id 1 aparece 104 veces.
El transaction id 2 aparece 36 veces.
...
El transaction id 10 aparece 10 veces.
El transaction id 11 aparece 6 veces.

El programa ha tardado 0.007992 segundos en ejecutar.
Se procesaron 134634.634635 pps
El throughput es de 11471096 Bytes/segundo.
```

Como en las pruebas iniciales se comprobó que fuera correcto el número de tramas Modbus TCP enviadas por cada IP, en este apartado se va a proceder a comprobar que es correcto el número de paquetes de cada tipo que indica el programa. Se han realizado varias comprobaciones de nuevo con la herramienta Tshark, y los resultados se pueden observar en la Figura 4.9 y Figura 4.10.

```
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "tcp" -T fields -e tcp | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
  1076 Transmission
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "mbtcp" -T fields -e mbtcp | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
   356 Modbus/TCP
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "tcp.dstport == 502 && mbtcp" -T fields -e tcp.dstport | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
   178 502
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "tcp.flags.syn==1" -T fields -e tcp.flags.syn | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
   114 1
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "tcp.flags.fin==1" -T fields -e tcp.flags.fin | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
   116 1
```

Figura 4.9: Comprobación de tipos de paquete con la herramienta Tshark.

```
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "mbtcp.trans_id==1" -T fields -e mbtcp.trans_id | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
   104 1
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "mbtcp.trans_id==2" -T fields -e mbtcp.trans_id | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
   36 2
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "mbtcp.trans_id==10" -T fields -e mbtcp.trans_id | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
   10 10
lubuntu@lubuntu:~/Desktop/TFG_2$ tshark -r modpoll.pcap -Y "mbtcp.trans_id==11" -T fields -e mbtcp.trans_id | awk '{split($1,a,"");for (i in a) print a[i];}' | sort | uniq -c
    6 11
```

Figura 4.10: Comprobación de IDs con la herramienta Tshark.

Con Tshark no es posible diferenciar si una petición es de lectura o de escritura, ya que esta diferenciación la realiza el IDS, por tanto se ha optado por mostrar el número total de peticiones en la traza (178), que comparándolo con la suma de peticiones de la Tabla 4.2, se puede comprobar que coincide el resultado.

La última comprobación que se ha realizado con Modpoll puede visualizarse en la Figura 4.11, y ha consistido en originar una traza en la que no se envíen peticiones durante unas decenas de segundos. El programa, al ejecutar la traza lanza un aviso ya que detecta que se ha producido una anomalía en el flujo del tráfico.

```
ALERTA. Flujo de paquetes anormal
Traza leída

Se procesaron 126 paquetes, de los cuales hay:
  126 paquetes IPv4.
  126 paquetes TCP.
  126 paquetes Modbus TCP, de los cuales:
    63 peticiones de lectura.
    0 peticiones de escritura.
    0 paquetes con datos erróneos.
    0 respuestas con excepción.
    0 paquetes con código de función no válido (reservado).
Numero de maestros en la red: 1
```

Figura 4.11: Resultado tras generar anomalías en la tasa de paquetes.

Pruebas con Metasploit

Este programa viene instalado por defecto en Kali Linux. Una vez ejecutado, haciendo una búsqueda mediante el comando `msf >search modbus` aparecen los módulos mostrados en la Figura 4.12.

Para cada uno de estos módulos, antes de ejecutar es necesario configurar la IP contra la que se quieren lanzar con el parámetro RHOSTS. El puerto 502 ya viene configurado por defecto en el parámetro RPORT.

```
Matching Modules
=====
#  Name                               Disclosure Date  Rank  Check
Description
-----
0  auxiliary/admin/scada/modicon_command 2012-04-05      normal No
Schneider Modicon Remote START/STOP Command
1  auxiliary/admin/scada/modicon_stux_transfer 2012-04-05      normal No
Schneider Modicon Ladder Logic Upload/Download
2  auxiliary/analyze/modbus_zip           normal No
Extract zip from Modbus communication
3  auxiliary/scanner/scada/modbus_findunitid 2012-10-28      normal No
Modbus Unit ID and Station ID Enumerator
4  auxiliary/scanner/scada/modbusclient    normal No
Modbus Client Utility
5  auxiliary/scanner/scada/modbusdetect    2011-11-01      normal Yes
Modbus Version Scanner
```

Figura 4.12: Módulos de Metasploit para Modbus TCP.

- El exploit *modbusdetect* sirve como prueba para determinar que el esclavo está utilizando Modbus TCP. En la Figura 4.13 se muestran todas las opciones que contiene este módulo. En este caso, únicamente ha sido necesario introducir la dirección IP del esclavo contra el que queremos lanzar el exploit.

```
Module options (auxiliary/scanner/scada/modbusdetect):
Name      Current Setting  Required  Description
-----
RHOSTS    192.168.244.133 yes        The target address range or CIDR identifier
RPORT     502              yes        The target port (TCP)
THREADS   1                yes        The number of concurrent threads
TIMEOUT   10               yes        Timeout for the network probe
UNIT_ID   1                yes        Modbus Unit Identifier, 1..255, most often
1
```

Figura 4.13: Opciones del módulo *modbusdetect*.

Como cabía esperar, en la ejecución de este exploit, que se corresponde con la Figura 4.14, se muestra por pantalla que efectivamente nos encontramos ante un esclavo Modbus.

```
[+] 192.168.244.133:502 - 192.168.244.133:502 - MODBUS - received correct MODBUS/TCP header (unit-ID: 1)
[*] 192.168.244.133:502 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figura 4.14: Respuesta ante el exploit *modbusdetect*.

Tras capturar el tráfico generado con esta acción, se procede a ejecutar el IDS con la traza guardada, obteniendo por pantalla la información de la Figura 4.15.

Como se puede observar, Metasploit envía una única petición, en este caso de lectura. El servidor responde a la petición mandando una excepción, con un código de función igual a 132.

```

El código de función 132 del paquete 25 indica error
Traza leída

Se procesaron 47 paquetes, de los cuales hay:
 10 paquetes IPv4.
 10 paquetes TCP.
 1 paquetes Modbus TCP, de los cuales:
   1 peticiones de lectura.
   0 peticiones de escritura.
   0 paquetes con datos errneos.
   1 respuestas con excepción.
   0 paquetes con código de función no válido (reservado).
Numero de maestros en la red: 1

Flag SYN activado en 2 paquetes

Flag FIN activado en 2 paquetes

IP      Function Code  Cantidad
192.168.244.135    4          1
    
```

Figura 4.15: Resultado de la ejecución del tráfico generado con *modbusdetect*.

El valor 132 se debe a que el servidor responde con un código de función que es igual a la suma de 128 más 4, que es el código enviado por el cliente. El tipo de excepción *Illegal function* (Figura 4.16) puede deberse a que la orden no esté implementada para el servidor, o que no tenga la configuración necesaria para llevarla a cabo. En cualquier caso, se comprueba que el IDS es capaz de detectarlo y lanzar un mensaje de aviso.

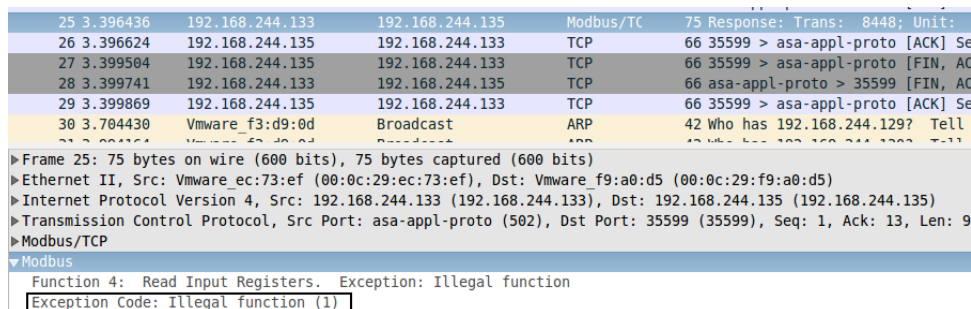


Figura 4.16: Captura de Wireshark con la excepción *Illegal Function*.

- Los esclavos en una red Modbus se identifican mediante el *Unit ID*. El módulo *modbus_findunitid* hace un sondeo de este parámetro sobre los 254 posibles esclavos en una red Modbus (Figura 4.17), para enumerar aquellos que se encuentran disponibles en la red.

A partir de los resultados del detector en la Figura 4.18, se observa que Metasploit envía un mensaje a cada posible servidor, obteniendo respuesta en aquellos casos en los que existe un servidor disponible.

Todas las respuestas de los esclavos devuelven una excepción, que se debe a la misma razón que la explicada en el caso del módulo *modbus_detect*. A pesar de que el cliente recibe excepciones, tiene información sobre los servidores que se encuentran funcionando en la red contra la que ha lanzado el ataque. En este caso, según el número de respuestas observamos que en la red hay 247 servidores disponibles.


```
[+] 192.168.244.133:502 - Received: correct MODBUS/TCP from stationID 245
[+] 192.168.244.133:502 - Received: correct MODBUS/TCP from stationID 246
[+] 192.168.244.133:502 - Received: correct MODBUS/TCP from stationID 247
[*] 192.168.244.133:502 - Received: incorrect/none data from stationID 248 (probably not in use)
[*] 192.168.244.133:502 - Received: incorrect/none data from stationID 249 (probably not in use)
[*] 192.168.244.133:502 - Received: incorrect/none data from stationID 250 (probably not in use)
[*] 192.168.244.133:502 - Received: incorrect/none data from stationID 251 (probably not in use)
[*] 192.168.244.133:502 - Received: incorrect/none data from stationID 252 (probably not in use)
[*] 192.168.244.133:502 - Received: incorrect/none data from stationID 253 (probably not in use)
[*] 192.168.244.133:502 - Received: incorrect/none data from stationID 254 (probably not in use)
[*] Auxiliary module execution completed
```

Figura 4.17: Respuesta ante el exploit *modbus_findunitid*.

```
Se procesaron 4842 paquetes, de los cuales hay:
  2546 paquetes IPv4.
  2530 paquetes TCP.
  501 paquetes Modbus TCP, de los cuales:
    254 peticiones de lectura.
    0 peticiones de escritura.
    0 paquetes con datos erroneos.
    247 respuestas con excepción.
    0 paquetes con código de funcion no válido (reservado).
Numero de maestros en la red: 1

Flag SYN activado en 508 paquetes

Flag FIN activado en 506 paquetes
```

Figura 4.18: Resultado de la ejecución del tráfico generado con *modbus_findunitid*.

- EL siguiente módulo a ejecutar es *modbusclient*. Este exploit permite tanto leer como escribir en los registros y bobinas (*coils*) del sistema. Tiene varias posibles ejecuciones, que se muestran a continuación en la Figura 4.19.

```
Module options (auxiliary/scanner/scada/modbusclient):
```

| Name | Current Setting | Required | Description |
|----------------|-----------------|----------|---|
| DATA | | no | Data to write (WRITE_COIL and WRITE REGISTER modes only) |
| DATA_ADDRESS | 1 | yes | Modbus data address |
| DATA_COILS | | no | Data in binary to write (WRITE_COILS mode only) e.g. 0110 |
| DATA_REGISTERS | | no | Words to write to each register separated with a comma (WRITE_REGISTERS mode only) e.g. 1,2,3,4 |
| NUMBER | 100 | no | Number of coils/registers to read (READ_COILS and READ_REGISTERS modes only) |
| RHOSTS | 192.168.244.133 | yes | The target address range or CIDR identifier |
| RPORT | 502 | yes | The target port (TCP) |
| UNIT_NUMBER | 1 | no | Modbus unit number |

Figura 4.19: Opciones del exploit *modbusclient*.

Para cambiar entre las diferentes opciones disponibles, únicamente es necesario utilizar el comando *set ACTION* seguido de la opción que se desea ejecutar.

Se han realizado varias peticiones utilizando las posibilidades que ofrece este módulo, haciendo que algunas de estas no sean válidas, como solicitar leer más registros de los que existen, con el fin de comprobar que el IDS detecta estos paquetes como anómalos. En la Figura 4.20 se enseña un ejemplo de petición de escritura.

```
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/scada/modbusclient) > set DATA 1
DATA => 1
msf5 auxiliary(scanner/scada/modbusclient) > run
[*] Running module against 192.168.244.133

[*] 192.168.244.133:502 - Sending WRITE COIL...
[+] 192.168.244.133:502 - Value 1 successfully written at coil address 1
[*] Auxiliary module execution completed
```

Figura 4.20: Ejemplo de escritura en bobina con *modbusclient*.

Al pasar la traza resultante por el IDS (Figura 4.21), obtiene que la traza contiene un paquete con contenido erróneo y 4 respuestas con excepción.

```
El código de función 131 del paquete 287 indica error
El código de función 129 del paquete 7674 indica error
El código de función 131 del paquete 7837 indica error
Error en el id del protocolo modbus 10352
El código de función 143 del paquete 10354 indica error
Traza leída

Se procesaron 10987 paquetes, de los cuales hay:
  468 paquetes IPv4.
  358 paquetes TCP.
  54 paquetes Modbus TCP, de los cuales:
    9 peticiones de lectura.
    21 peticiones de escritura.
    1 paquetes con datos erroneos.
    4 respuestas con excepción.
    0 paquetes con código de funcion no válido (reservado)
Numero de maestros en la red: 1

Flag SYN activado en 80 paquetes

Flag FIN activado en 79 paquetes
```

Figura 4.21: Resultado de la ejecución del tráfico generado con *modbusclient*.

Al filtrar la captura en Wireshark, se comprueba que los datos ofrecidos por el programa son correctos, tal y como se muestra en la Figura 4.22 y Figura 4.23.

```
Filter: modbus.exception_code
Expression... Clear Apply Guardar
```

| No. | Time | Source | Destination | Protocol | Length | Info |
|-------|-------------|-----------------|-----------------|------------|--------|---------------------------|
| 287 | 43.360366 | 192.168.244.133 | 192.168.244.135 | Modbus/TCP | 75 | Response: Trans: 0; Unit: |
| 7674 | 1198.401476 | 192.168.244.133 | 192.168.244.135 | Modbus/TCP | 75 | Response: Trans: 0; Unit: |
| 7837 | 1220.874771 | 192.168.244.133 | 192.168.244.135 | Modbus/TCP | 75 | Response: Trans: 0; Unit: |
| 10354 | 1605.402225 | 192.168.244.133 | 192.168.244.135 | Modbus/TCP | 75 | Response: Trans: 0; Unit: |

```
▶Frame 287: 75 bytes on wire (600 bits), 75 bytes captured (600 bits)
▶Ethernet II, Src: Vmware_ec:73:ef (00:0c:29:ec:73:ef), Dst: Vmware_f9:a0:d5 (00:0c:29:f9:a0:d5)
▶Internet Protocol Version 4, Src: 192.168.244.133 (192.168.244.133), Dst: 192.168.244.135 (192.168.244.135)
▶Transmission Control Protocol, Src Port: asa-appl-prot (502), Dst Port: 45863 (45863), Seq: 1, Ack: 13, Len: 9
▶Modbus/TCP
▼Modbus
  Function 3: Read Holding Registers. Exception: Illegal data value
  Exception Code: Illegal data value (3)
```

Figura 4.22: Captura de Wireshark de las respuestas con excepción.

El aviso de información errónea en el paquete 10352 de la traza se debe a que el paquete tiene una longitud mayor a la permitida.

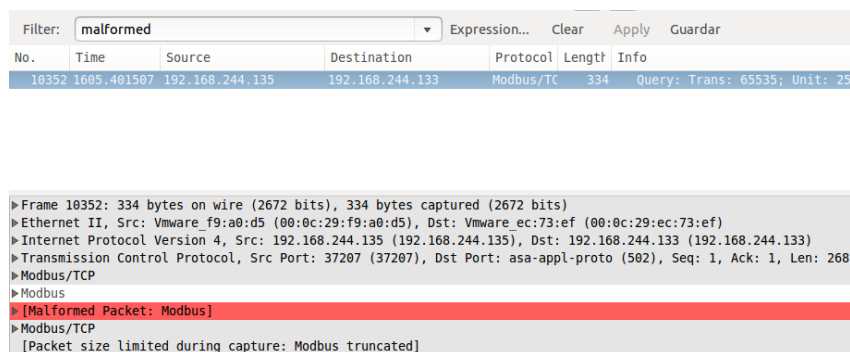


Figura 4.23: Captura de Wireshark de un paquete con anomalía.

- El exploit *modicon_command* está diseñado para dispositivos pertenecientes a *Schneider Electric*, los cuales utilizan un protocolo propio basado en Modbus TCP. Este módulo permite a un usuario remoto cambiar el estado del PLC, de manera que un atacante puede hacerse con el control de un PLC haciendo que inicie o finalice un proceso.

Esto se consigue mediante el envío de peticiones con el código de función 90, que es uno de los valores que se encuentran reservados.

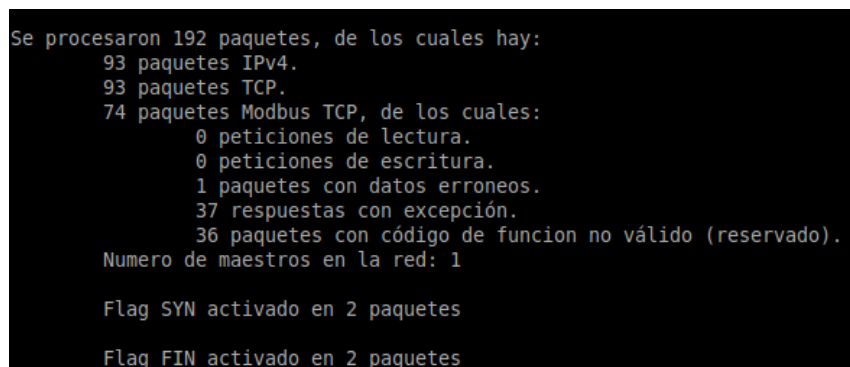


Figura 4.24: Resultado de la ejecución del tráfico generado con *modicon_command*.

En la Figura 4.24 se observa que el programa no detecta las solicitudes enviadas por Metasploit como peticiones de lectura ni de escritura. La razón se debe a que el programa es capaz de reconocer aquellos códigos de función que son públicos, y están definidos en la Figura 2.5. Los códigos reservados, al no encontrarse definidos públicamente, no ha sido posible implementarlos en el IDS. Sin embargo, sí es capaz de reconocer cuando se trata de uno de ellos.

Adicionalmente, el programa detecta 37 respuestas con excepción, ya que el simulador de servidor no es capaz de llevar a cabo la acción que solicita el cliente, por lo que responde a cada petición con una excepción.

Por último, se detecta una petición errónea, como se ve en la Figura 4.25 y 4.26 debido a que la longitud del campo *Longitud* es superior a la permitida para Modbus TCP, que es 254 Bytes (PDU más identificador de unidad).

```
El código de función 218 del paquete 60 indica error
Código de función reservado (no válido) 61
El código de función 218 del paquete 63 indica error
Error en la longitud del paquete modbus 64
El código de función 218 del paquete 66 indica error
Código de función reservado (no válido) 67
El código de función 218 del paquete 69 indica error
Código de función reservado (no válido) 70
El código de función 218 del paquete 71 indica error
```

Figura 4.25: Alertas generadas tras la ejecución de *modicon_command*.

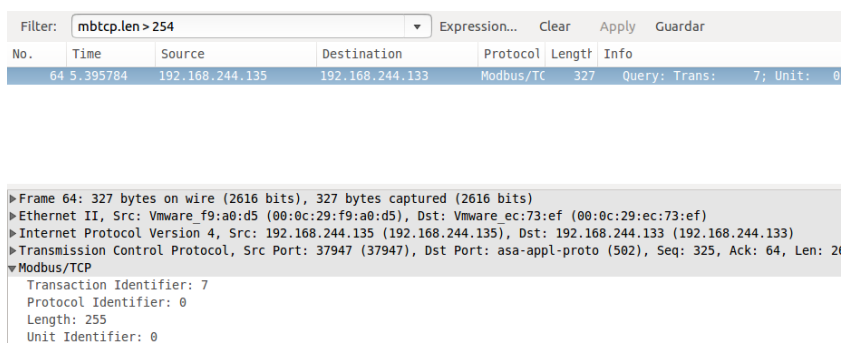


Figura 4.26: Captura de paquete con longitud superior a la permitida.

- Con el módulo *modicon_stux_transfer* se obtienen resultados muy similares a los mostrados con el uso de *modicon_command*, ya que este exploit está también diseñado para dispositivos de Schneider-Electric.

La única diferencia que se observa entre los resultados del uso de estos dos exploit, es que el módulo *modicon_stux_transfer* envía el doble de peticiones, por lo que se obtiene el doble de respuestas con excepción. Por este motivo, se han obviado las capturas en este apartado.

- Por último se encuentra el módulo *modbus_zip*, el cual no se utilizará ya que no resulta relevante para la validación del IDS. Su función es la de extraer un archivo zip a partir de una comunicación Modbus.

4.2.3. Pruebas con tráfico real

Como pruebas de funcionalidad también cabe destacar las que han sido realizadas con tráfico real. La traza, facilitada por la Cátedra UAM-Naudit, contiene el tráfico capturado en una red real en un intervalo de tiempo de 24 horas, y cuenta unos de 6 millones de paquetes.

Antes de proceder al análisis de esta traza, ha sido necesario filtrarla para eliminar las retransmisiones que contenía (alrededor del 18%, quedando reducida a unos 5 millones de paquetes), ya que el IDS no es capaz de detectarlas como tal, sino que las lee y analiza como si se trataran de otra petición más.

El uso de esta traza ha servido para añadir al código el análisis de la cabecera VLAN, en caso de haberla, ya que las trazas utilizadas hasta el momento no disponían de esta cabecera. Sin embargo, todos los paquetes de esta traza sí la contienen.

También sirvió para solucionar un problema en la detección del número de maestros, ya que el resto de trazas analizadas poseían un solo maestro, en cambio, esta traza cuenta con 5 maestros en la red.

En cuanto a la ejecución, no se detectó ninguna anomalía, lo cual es esperable puesto que se trata de tráfico legítimo. Además, las pruebas realizadas en esta traza han sido útiles para verificar que el programa desarrollado no muestra problemas para procesar una traza de gran tamaño.

4.3. Validación de rendimiento

La validación del rendimiento del IDS es una prueba clave a realizar, ya que aunque el detector cumpliera con su funcionalidad, no sería de gran utilidad si no consiguiese procesar los paquetes a una alta tasa de ejecución.

Como ya se ha mencionado, la traza que contiene tráfico real cuenta con unos 5 millones de paquetes una vez filtradas las retransmisiones, número más que suficiente para probar la eficiencia en cuanto a rendimiento que posee el detector de intrusiones.

Tabla 4.3: Resultados de las medidas de rendimiento.

| Medida de rendimiento | Resultado |
|-----------------------|-------------------|
| Tiempo de ejecución | 0.973701 segundos |
| Paquetes por segundo | 5.2 Mpps |
| Throughput | 2.7 Gbps |

En la Tabla 4.3 se exponen los resultados del rendimiento del IDS. El *Throughput* es una medida de rendimiento que se calcula con el cociente de los bytes analizados entre el tiempo total de ejecución. Se puede observar que esta medida supera con creces el valor que se había puesto como objetivo al inicio del TFG, que era 1 Gbps.

4.4. Conclusiones

A lo largo de este capítulo se han realizado diversos tipos de pruebas mediante el uso de diferentes programas y herramientas que han validado el correcto funcionamiento y rendimiento del IDS.

Ha quedado comprobado que el sistema es capaz de detectar distintos tipos de anomalías a una muy buena velocidad de procesamiento, el cual es un requisito indispensable para que el programa resulte de utilidad.

Tras este apartado, queda finalizado el IDS. En el siguiente capítulo se plasmarán las conclusiones extraídas de la elaboración de este Trabajo de Fin de Grado y las posibles líneas de trabajo futuro del mismo.

5

Conclusiones y trabajo futuro

5.1. Conclusiones

A partir de las pruebas realizadas, ha quedado demostrada la validez y seguridad que presenta el IDS frente a distintos tipos de anomalías y variaciones en el volumen del tráfico de la red. También han resultado satisfactorias las pruebas realizadas en cuanto a medidas del rendimiento, obteniendo como resultado un sistema eficiente capaz de analizar tráfico a una velocidad de 2.7 Gbps. Se ha tenido la oportunidad de probar el código con tráfico tomado de un entorno real, lo cual ha permitido obtener resultados más realistas.

Por lo tanto, queda cumplido el objetivo de este TFG, para el han resultado de gran utilidad asignaturas cursadas en el grado como *Programación II* y *Arquitectura de Redes I y II* y sobre las cuales se han ampliado conocimientos debido a la realización de este proyecto. Adicionalmente, se ha podido comprobar que la elección del lenguaje de programación ha sido acertada, ya que el uso de *C* junto con la implementación de tablas *hash* han resultado en la obtención de un código capaz de procesar los datos obtenidos a gran velocidad.

Gracias a este TFG también he aprendido sobre el manejo de máquinas virtuales y diversas herramientas como Metasploit, que ha servido para entrar en contacto con los sistemas de intrusión. Por último, también cabe destacar el aprendizaje adquirido sobre el funcionamiento de las redes SCADA y los protocolos que se utilizan en ellas, así como el estudio en profundidad de los protocolos IP y TCP, ya conocidos con anterioridad pero de una manera más superficial.

Si se desea consultar el código de este proyecto, se encuentra alojado en el repositorio de Github <https://github.com/ceIiapascual/modbustcp>.

5.2. Trabajo futuro

Para la validación del rendimiento sobre la traza con tráfico real, ha sido necesario antes filtrarla para eliminar las retransmisiones con las que contaba, que suponían aproximadamente un 18 % del tráfico total. Una trabajo futuro de este IDS podría consistir en realizar un código capaz de detectar cuando un paquete se trata de una retransmisión para así contabilizarlo, ya que un número elevado de retransmisiones indica problemas en la red, por lo que este dato puede resultar de utilidad.

Otra posibilidad sería que el IDS lance un aviso en tiempo real cuando detecte un nuevo maestro en la red. También podría ser una mejora que se muestre el número de esclavos que hay en la red por cada maestro, de la misma manera que ahora se muestra el número de maestros.

Además, se podría tratar de modificar el programa para convertirlo en un sistema de prevención de intrusiones o IPS (del inglés, *Intrusion Prevention System*), y que en lugar de únicamente detectar anomalías, fuese capaz de eliminarlas. Para llevarlo a cabo sería necesario que el IPS actuase como un cortafuegos, interceptando los paquetes que considerase anómalos para que no llegasen al destino.

Otra mejora del proyecto podría ser exportar ficheros con gráficas al ejecutar el programa, además de mostrar la información por pantalla, para así poder obtener también resultados visuales. Algunos datos que se podrían mostrar en las gráficas serían la evolución en el tiempo de la cantidad de peticiones enviadas, o de los códigos de función más utilizados en la red, entre otros.

Por último, también se podría llevar a cabo una optimización del código, por un lado mediante el uso de *callgrind* para ver qué funciones del código llevan un mayor tiempo de ejecución y centrarse en su optimización. Por otro lado, sería conveniente estudiar el algoritmo de generación de claves *hash* para encontrar una manera en la que se produzca el menor número de colisiones posibles.

Bibliografía

- [1] The Modbus Organization, “Modbus messaging on TCP/IP implementation guide V1.0b,” http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf, 2006, [Último acceso: 2 de enero de 2020].
- [2] L. Zhou, H. Guo, D. Li, J. Zhou, and J. Wong, “A scheme for lightweight SCADA packet authentication,” in *Proc. 23rd Asia-Pacific Conference on Communications (APCC). Singapore*. IEEE, 2017, pp. 1–6.
- [3] The Modbus Organization, “Modbus application protocol specification v1.1b3,” http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf, 2016, [Último acceso: 4 de enero de 2020].
- [4] L. D. Candia, A. S. Rodríguez, N. Castro, P. Bazán, V. M. Ambrosi, and F. J. Díaz, “Mejoras en maquinaria industrial con IoT: hacia la industria 4.0,” in *Actas XXIV Congreso Argentino de Ciencias de la Computación. La Plata, Argentina*, 2018.
- [5] A. Culebras Sánchez, “Desarrollo de un sistema de monitorización de redes SCADA para la detección de tráfico anómalo,” Trabajo Fin de Grado, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2016.
- [6] A. Daneels and W. Salter, “What is SCADA?” in *Proc. International Conference on Accelerator and Large Experimental Physics Control Systems. Trieste, Italy*, 1999, pp. 339–343.
- [7] J. M. Molina Martínez and M. Jiménez Buendía, *Programación gráfica para ingenieros*. Marcombo, 2010.
- [8] D. Sanches Gómez, “Desarrollo de un sistema eficiente de análisis del protocolo IEC 60870-5-104 para la detección de anomalías en redes SCADA,” Trabajo Fin de Grado, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2019.
- [9] G. Clarke, D. Reynders, and E. Wright, *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Newnes, 2004.
- [10] M. M. Alvarez Pichizaca, “Implementación de la arquitectura para automatización distribuida Modbus-IDA para el laboratorio de automatización industrial (EIS),” Tesis de Grado, Escuela Superior Politécnica de Chimborazo, Ecuador, 2010.
- [11] Q. Liu and Y. Li, “Modbus/tcp based network control system for water process in the firepower plant,” in *Proc. 6th World Congress on Intelligent Control and Automation. Dalian, China*, vol. 1. IEEE, 2006, pp. 432–435.

- [12] S. Bhatia, N. Kush, C. Djameludin, J. Akande, and E. Foo, “Practical modbus flooding attack and detection,” in *Proc. Twelfth Australasian Information Security Conference- Volume 149, Auckland, New Zealand*. Australian Computer Society, Inc., 2014, pp. 57–65.
- [13] W. Gao and T. H. Morris, “On cyber attacks and signature based intrusion detection for modbus based industrial control systems,” *Journal of Digital Forensics, Security and Law*, vol. 9, no. 1, 2014.
- [14] I. N. Fovino, A. Carcano, T. D. L. Murel, A. Trombetta, and M. Masera, “Modbus/DNP3 state-based intrusion detection system,” in *Proc. 24th IEEE International Conference on Advanced Information Networking and Applications. Perth, Australia*. IEEE, 2010, pp. 729–736.
- [15] E. J. Mira Alfaro, “Implantación de un Sistema de Detección de Intrusos en la Universidad de Valencia,” Proyecto Final de Carrera, Universidad de Valencia, España, 2003.
- [16] G. Trubetskoy, “Holt-Winters Forecasting for dummies (or developers),” <https://grisha.org/blog/2016/01/29/triple-exponential-smoothing-forecasting/>, 2016, [Último acceso: 19 de diciembre de 2019].
- [17] Oracle, “Análisis del tráfico de red con analizadores tshark y wireshark,” https://docs.oracle.com/cd/E56339_01/html/E53800/gncns.html, 2011, [Último acceso: 4 de enero de 2020].