



# GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Diseño e implementación de un laboratorio de  
ciberseguridad industrial

Autor: Miguel Oleo Blanco

Director: Antonio Pérez Sánchez

Madrid



Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Diseño e implementación de un laboratorio de ciberseguridad industrial

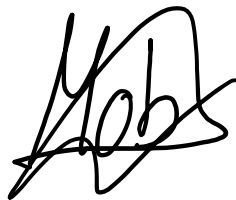
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2020/21 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Miguel Oleo Blanco

Fecha: 10/07/2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Antonio Pérez Sánchez

Fecha: 10/07/2021





# GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Diseño e implementación de un laboratorio de  
ciberseguridad industrial

Autor: Miguel Oleo Blanco

Director: Antonio Pérez Sánchez

Madrid



# Agradecimientos

En primer lugar, gracias a Gregorio, Jarauta, Mondéjar y a Antonio por haberme ofrecido esta oportunidad única y por acompañarme a lo largo de este proyecto. A este último agradecerle especialmente, que me haya guiado y ayudado siempre desde la cercanía. También darle gracias a mis compañeros y amigos, que a pesar de que he sido muy pesado con este proyecto, me han apoyado incondicionalmente. Por último, gracias a la Cátedra de Industria Conectada para haberse fiado de mí para desarrollar este proyecto y confiar plenamente en mi como para ofrecerme una beca y ofrecer apoyo económico en caso de ser necesario.





# **DISEÑO E IMPLEMENTACIÓN DE UN LABORATORIO DE CIBERSEGURIDAD INDUSTRIAL**

**Autor: Oleo Blanco, Miguel.**

Director: Pérez Sánchez, Antonio.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas y la Cátedra de la Industria Conectada (CIC) de ICAI

## **RESUMEN DEL PROYECTO**

En este trabajo se diseñará e implementará un laboratorio de ciberseguridad industrial en ICAI. Para ello, se investigará sobre distintos protocolos usados en la industria y se llevarán a cabo una serie de ataques para que los alumnos aprendan sobre estos protocolos y sus vulnerabilidades.

**Palabras clave:** Ciberseguridad, laboratorio, protocolos, vulnerabilidades

### **1. Introducción**

El campo de la ciberseguridad es, desde hace unos años, uno de los que más se está invirtiendo. También, los usuarios finales se han dado cuenta de su importancia, que no solo afecta a fabricantes o desarrolladores. La ciberseguridad también ha cogido fuerza en las industrias, por motivos obvios, como proteger plantas y procesos de producción, etc. Este crecimiento en la seguridad de los sistemas se debe a que, cada vez hay información más sensible en ellos y debido al crecimiento de los ataques. En este proyecto se centrará más en el ámbito industrial de la ciberseguridad. Se investigarán vulnerabilidades de varios protocolos industriales y se explotarán de forma controlada. Con este proyecto se pretende crear una red segura para poder realizar prácticas de ciberseguridad. Para realizarlo, hay una gran parte de investigación y testeo.

### **2. Definición del proyecto**

Este proyecto tiene varios objetivos. Uno de ellos es realizar una investigación profunda sobre distintos protocolos industriales, sus vulnerabilidades y cómo explotarlas. Para ello se usará la infraestructura que encontramos en el laboratorio de ICAI, del cual haremos uso de los puestos de trabajo, ordenadores, PLCs, HMIs y de la red interna. Este trabajo de investigación es muy importante, ya que algunos de los protocolos con los que nos enfrentaremos, son propietarios y hay poca información sobre ellos. Se pretende que al final del proyecto, se haya realizado una investigación detallada sobre los ataques que se pueden realizar paso a paso, desde la fase de investigación del protocolo, hasta la fase de testeo del ataque. También se busca poder crear una infraestructura segura para realizar prácticas de ciberseguridad para estudiantes de ICAI. Con este proyecto, se investigará que tipos de ataques se pueden llevar a cabo y cómo se realizan, para que los estudiantes en un futuro los puedan replicar, sin tener que realizar la parte de preparación del laboratorio. Esto conlleva una gran parte de testing para ver qué elementos pueden fallar y para saber que equipos son necesarios para llevar a cabo estas prácticas. El laboratorio tiene que quedar preparado de tal forma que también cumple su función actual y no se altera su funcionamiento al implementar esta nueva sección de ciberseguridad. En concreto se llevaría esto a cabo de una forma relativamente sencilla. Con un ordenador virtualizado con distintas herramientas de las que vienen citadas a continuación. A este

ordenador se conectarían los alumnos y podrán realizar las prácticas de ciberseguridad correspondientes. Esto también nos permitiría limitar las IPs que ve cada usuario, de esta forma no tendríamos que crear VLANs para segmentar la red. Con esto conseguiríamos modificar lo menos posible la red del laboratorio. Con esto, los alumnos podrán realizar distintas prácticas sobre distintos protocolos y distintos ataques. También se busca que el laboratorio esté preparado para alumnos o académicos que quieran realizar estudios de ciberseguridad en un futuro. Algunas de estas prácticas se realizarán con equipos físicos y otros con equipos virtualizados. Este concepto está muy desarrollado por un TFM de un alumno de la Universidad Oberta de Cataluña [1]. Otro de los objetivos principales de este proyecto es que la Universidad Pontificia de Comillas (ICAI) entre en la Red de Excelencia Nacional de Investigación en Ciberseguridad (RENIC). Esta red se creó de las manos de INCIBE y del ecosistema investigador en ciberseguridad español. Esta red está formada por centro que destacan por su excelente campo de I+D+i en ciberseguridad, tratando temas actuales, coordinados por un futuro plan director de la Unión Europea y contemplando las necesidades reales de los usuarios finales y de la industria. [2].

### **3. Descripción del modelo/sistema/herramienta**

La metodología que hemos empleado a lo largo del desarrollo de este trabajo es muy simple. Nos hemos basado en ataques parecidos y en vulnerabilidades para poder reproducirlas en el entorno seguro que es el laboratorio de ICAI. Al principio siempre se requiere una fase de análisis de tráfico y de descubrimientos de posibles fallos o necesidades que podrían llegar a surgir durante el proyecto. Por ejemplo, en nuestro caso, nos dimos cuenta de que hacían falta más equipos (switches) en el laboratorio y que el tráfico S7-comm plus no es reconocido por defecto en Wireshark. Esta fase es clave para poder continuar sin grandes imprevistos durante el desarrollo del proyecto. Una vez concluido lo anterior, pasamos a la investigación de cada protocolo. Todo lo que se explica a continuación se debe repetir para cada protocolo, ya que los pasos son básicos para el desarrollo. Lo primero es la fase de investigación sobre dicho protocolo. Esta fase es crítica, ya que al igual que hay protocolos muy sencillos, como puede ser Modbus, en general requieren conocer bien las capas del protocolo, cómo funciona, entre qué equipos se emplea y, sobre todo, lo más importante, es saber qué medidas de seguridad emplea (como puede ser criptografía asimétrica, SHA, etc) y las vulnerabilidades de la versión del protocolo a emplear. Una vez visto esto, también hay que realizar una fase de investigación sobre cómo saltarnos las medidas de seguridad implementadas. Una vez conseguido esto, pasaremos a una fase de investigación sobre los ataques que se pueden realizar sobre estos protocolos. Esto va de la mano de la fase de investigación realizada sobre cómo saltarse los métodos de seguridad. En esta fase también se requiere buscar ejemplos realizados por otras personas y realizar una lectura detallada de los CVEs (código de vulnerabilidades) encontrados. Una vez terminado esto, hay que pasar a una fase de pruebas, donde se testeará todo lo encontrado anteriormente y se evaluarán los resultados. Llegados a este punto, se puede retroceder al principio en caso de que los resultados no sean satisfactorios. Como ya he citado antes, todo lo anterior se deberá realizar para cada protocolo que investigaremos a lo largo del proyecto. Al final de este, se dedicará un tiempo exclusivo sobre la inyección de código malicioso a los PLCs. Toda esta metodología y planificación queda reflejada en el diagrama Gantt adjuntado en el Anexo B.

A continuación, se listan y explican las herramientas empleadas para llevar a cabo este proyecto. Las citadas, son las usadas de forma general, las herramientas específicas para cada protocolo se citarán y explicarán durante su respectivo capítulo.

- Kali Linux: esta es una distribución de Linux centrada en temas de ciberseguridad. Este sistema operativo viene con muchas herramientas que emplearemos a lo largo del proyecto, las cuales se citarán a lo largo del documento [3]. Este sistema operativo es montado sobre una Raspberry pi 4 de 4 GB de RAM, la cual nos permite fácilmente usar estas herramientas y poder conectarnos a las distintas redes con un formato de equipo reducido.
- Wireshark: esta es una herramienta ampliamente conocida para analizar los paquetes presentes en una red. Este software nos permite leer la información intercambiada en las interfaces seleccionadas al igual que nos permite ver con detalle la información de cada paquete. Esta herramienta es básica para este proyecto y será muy utilizada en la parte de investigación de los protocolos para extraer información de cómo funcionan. También lo usaremos para comprobar que los ataques llevados a cabo son correctos o para detectar fallos. [4]
- MSFConsole: Esta es una framework de metasploit. Ofrece una consola donde se centralizan muchos paquetes para distintos tipos de ataques. Estos paquetes van desde escáneres de redes o tráfico hasta herramientas preparadas para realizar ataques específicos. Esta es una de las herramientas que vienen preinstaladas en Kali Linux. [5]
- ICSsploit: es una herramienta muy similar a MSFConsole pero que nos aporta paquetes extra. Esta herramienta no se encuentra en Kali, por lo que debemos de descargarla y la podemos ejecutar desde cualquier equipo con Python. [6]
- Scapy: esta herramienta nos facilita la creación de paquetes y la transmisión de estos. Este software basado en Python lo emplearemos sobre todo para ataques de tipo Packet Replay (se crafean paquetes idénticos o similares a los originales del sistema y los enviamos para engañarlo). [7]
- GitHub: es un sistema que nos facilita la gestión y el control de las versiones del código. Es una plataforma ampliamente utilizada hoy en día. No solo sirve para gestionar nuestros proyectos si no que, desde GitHub desktop, podemos ver proyectos publicados por otra gente. [8]

#### **4. Resultados**

Las pruebas realizadas sobre los protocolos propuestos para este proyecto en general han sido satisfactorias. Por otro lado, en los que hemos encontrado dificultades (ya sea por su seguridad o complicaciones para implementarlos), hemos realizado una labor de auditoría. En ella definimos los problemas encontrados y como proceder en caso de querer realizar prácticas de estas partes. Entrando en detalle sobre los resultados obtenidos, en cuanto al protocolo S7Comm, hemos demostrado que, si se pueden realizar ataques y de forma muy sencilla, ya que es un protocolo que no incorpora suficiente seguridad. En cuanto a la versión más segura, el S7Comm Plus lo encontramos en comunicaciones entre equipos Siemens más modernos, como los presentes en el laboratorio. Sobre este protocolo, no se ha realizado ningún ataque, aunque se ha entrado a detalle sobre su seguridad y cómo sería realizar un ataque de inyección de código malicioso sobre un autómatas. En cuanto a Modbus, al ser un protocolo industrial antiguo, no contempla ningún tipo de seguridad y es el más fácil de atacar. En este apartado se han realizado varios ataques, como puede ser un Man in the Middle, un packet replay o

usar frameworks de explotis. Este protocolo es interesante, aunque probablemente el menos extendido hoy en día, para aprender los conceptos y como primer contacto a la ciberseguridad industrial. En Cuanto a Profinet, al emplear sobre todo equipos Siemenes, es complejo generar tráfico de este protocolo de tal forma que nos permita realizar ataques. Debido a esto, se han propuesto una serie de soluciones para que este protocolo pueda introducirse en el laboratorio de ciberseguridad. Por último, en cuanto a los sistemas SCADA, se basa en protocolos industriales como pueden ser S7, Profinet, Modbus... por lo que su seguridad y los ataques que se pueden realizar son los vistos en cada apartado de estos protocolos.

```
=====
Valor Q0.0: True
Valor Q0.1: False
Valor Q0.2: True
Valor Q0.3: False
Valor Q0.4: False
Valor Q0.5: False
=====
¿Desea Continuar? Pulse n/N para salir o cualquier otra para continuar:
Introduzca la dirección donde quiere escribir (0-5): 1
Introduzca 0 si quiere ponerlo a false o 1 para true: 1
VALOR INTRODUCIDO CORRECTAMENTE.
=====
Valor Q0.0: True
Valor Q0.1: True
Valor Q0.2: True
Valor Q0.3: False
Valor Q0.4: False
Valor Q0.5: False
=====
¿Desea Continuar? Pulse n/N para salir o cualquier otra para continuar: |
```

*Figura 1: Ejemplo de ataque realizado en el proyecto*

## 5. Conclusiones

La principal conclusión del proyecto es que se puede implementar un laboratorio de ciberseguridad en ICAI de manera inmediata y sin muchos costes. Partimos de un laboratorio con muchos equipos industriales que se emplearían también para este laboratorio, ya que uno de los objetivos principales han sido aprovechar esta infraestructura al máximo. Los distintos protocolos y los distintos ataques proporcionan un conocimiento amplio sobre la ciberseguridad en entornos industriales para los estudiantes que realicen estas prácticas.

Otra conclusión importante es que, el protocolo S7Comm Plus y los ataques principales sujetos a este protocolo, nos han parecido demasiado complejos para realizar en este laboratorio. Esto se debe al gran nivel de seguridad que aporta esta versión moderna del S7Comm. Por otro lado, en cuanto a Profinet, se pueden realizar ataques, pero hemos demostrado que, hacen falta equipos compatibles y que nos permitan escribir datos a los autómatas. También se podría plantear softwares de simulación de Profinet, pero también conllevan un coste. En este protocolo se ha realizado una pequeña auditoría para plantear los problemas y las opciones a priori para poder implementa

## 6. Referencias

- [1] Elder Pérez, I., 2019. *Estudio y desarrollo de un enfoque de Pentesting para Sstemas de Control Industrial (ICS)*. Disponible en: <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/96949/6/eperezignTFM0619memoria.pdf> [Consulta: 10 de octubre 2020].

- [2] INCIBE. 2021. *Red de Excelencia Nacional de Investigación en Ciberseguridad*. Disponible en: <https://www.incibe.es/red-excelencia-idi-ciberseguridad> [Consulta: 10 de octubre 2020].
- [3] n.d. *Página de descarga de Kali Linux*. Disponible en: <https://www.kali.org/downloads> [Consulta: 10 de octubre 2020].
- [4] Wireshark.org. n.d. *Wireshark · Go Deep*. Disponible en: <https://www.wireshark.org> [Consulta: 10 de octubre 2020].
- [5] Offensive-security.com. n.d. Disponible en: <https://www.offensive-security.com/metasploit-unleashed/msfconsole/> [Consulta: 17 de octubre 2020].
- [6] GitHub. 2017. *tijldeneut/icssexploit*. Disponible en: <https://github.com/tijldeneut/icssexploit> [Consulta: 20 de octubre 2020].
- [7] community., P., n.d. *Scapy*. Scapy. Disponible en: <https://scapy.net/> [Consulta: 22 de noviembre 2020].
- [8] Oleo Blanco, M., 2020. *Repositorio del proyecto*. Github. Disponible en: <https://github.com/miguelob/TFG-Ciberseguridad> [Consulta: 8 de julio 2021].

# **DESING AND IMPLEMENTATION OF AN INDUSTRIAL CYBERSECURITY LABORATORY**

**Author: Oleo Blanco, Miguel**

Supervisor: Pérez Sánchez, Antonio

Collaborating Entity: ICAI – Comillas Pontifical University & Chair of Connected Industry (CIC) of ICAI.

## **ABSTRACT**

In this final degree project, an industrial cybersecurity laboratory will be designed and implemented at ICAI. Most common protocols used at the industry will be investigated and a series of attacks will be carried out so that students learn about these protocols and their vulnerabilities.

**Keywords:** Cybersecurity, laboratory, protocols, Vulnerabilities

## **1. Introduction**

The field of cybersecurity has been one of the most fields where huge amounts of money was put into. Also, end users have realized its importance, which does not only affect manufacturers or developers. Cybersecurity has also gained strength in industries, for obvious reasons, such as protecting plants and production processes, etc. This growth in the security of the systems is due to the fact that there is more and more sensitive information in them and due to the growth of the number of attacks. In this project, we will focus on the industrial field of cybersecurity. Vulnerabilities of various industrial protocols will be investigated and exploited in a controlled manner. The aim of this project is to create a secure environment to be able to carry out cybersecurity practices. To achieve this, research and testing will be the key to success.

## **2. Project definition**

This project has several objectives. One of them is to conduct in-depth research on different industrial protocols, their vulnerabilities and how to exploit them. For this purpose, the infrastructure found in the ICAI laboratory will be used, from which we will make use of the workstations, computers, PLCs, HMIs and the internal network. This research work is very important, since some of the protocols that we will face, are proprietary and there is little information about them. It is intended that at the end of the project, a detailed investigation has been carried out on the attacks that can be carried out step by step, from the investigation phase of the protocol to the testing phase of the attack. It also seeks to create a secure infrastructure to perform cybersecurity internships for ICAI students. With this project, we will investigate what types of attacks can be carried out and how they are carried out, so that students in the future can replicate them, without having to perform the preparation part of the laboratory. This involves a lot of testing to see what elements can fail and to know what equipment is needed to carry out these practices. The laboratory must be prepared in such a way that it also fulfills its current function, and its operation is not altered by implementing this new cybersecurity section. This would be done in a relatively simple way. With a virtualized computer with different tools of which are mentioned below. To this computer students would connect and will be able to carry out the corresponding cybersecurity practices. This would also allow us to limit the IPs that each user sees, so we would not have to create VLANs to

segment the network. This would make it possible to modify the laboratory network as little as possible. With this, students will be able to perform different practices on different protocols and different attacks. It is also sought that the laboratory is prepared for students or academics who want to carry out cybersecurity studies in the future. Some of these practices will be done with physical equipment and others with virtualized equipment. This concept is very much developed by a TFM of a student of the Open University of Catalonia [1]. Another of the main objectives of this project is that the Pontifical University of Comillas (ICAI) enters the Network of National Excellence for Cybersecurity Research (RENIC). This network was created by the hands of INCIBE and the Spanish cybersecurity research ecosystem. This network is formed by a center that stands out for its excellent field of R&D&I in cybersecurity, dealing with current issues, coordinated by a future master plan of the European Union, and considering the real needs of end users and the industry. [2].

### **3. Description of the model/system/tool**

The methodology we have used throughout the development of this work is very simple. We have relied on similar attacks and vulnerabilities to be able to reproduce them in the secure environment that is the ICAI laboratory. At the beginning, a phase of traffic analysis and discoveries of possible failures or needs that could arise during the project is always required. For example, in our case, we realized that more equipment (switches) was needed in the lab and that S7-comm plus traffic is not recognized by default in Wireshark. This phase is key to be able to continue without great unforeseen events during the development of the project. Once the above is concluded, we move on to the investigation of each protocol. Everything explained below must be repeated for each protocol, as the steps are basic to development. The first thing is the research phase on that protocol. This phase is critical, since just as there are very simple protocols, such as Modbus, in general they require to know well the layers of the protocol, how it works, between which computers it is used and above all, the most important thing, is to know what security measures it uses (such as asymmetric cryptography, SHA, etc.) and the vulnerabilities of the version of the protocol to be used. Once seen this, there are also 6 to perform a phase of research on how to bypass the security measures implemented. Once this is achieved, we will move on to an investigation phase on the attacks that can be made on these protocols. This goes hand in hand with the research phase carried out on how to bypass security methods. In this phase it is also required to look for examples made by other people and make a detailed reading of the CVEs (vulnerability code) found. Once this is finished, you must move to a testing phase, where everything previously found will be tested and the results will be evaluated. At this point, we can go back to the beginning in case we are not satisfied with satisfactory results. As I mentioned before, all the above should be done for each protocol that we will investigate throughout the project. At the end of this, an exclusive time will be devoted on injecting malicious code into the PLCs. All this methodology and planning is reflected in the Gantt chart attached in Annex B.

Then, ready and explain the tools used to carry out this project. The aforementioned, are those used in a general way, the specific tools for each protocol will be cited and explained during their respective chapter.

- Kali Linux: This is a Linux distribution focused on cybersecurity issues. This operating system comes with many tools that we will use throughout the project,

which will be cited throughout the document [3]. This operating system is mounted on a Raspberry pi 4 of 4 GB of RAM, which allows us to easily use these tools and be able to connect to the different networks with a reduced computer format.

- Wireshark: This is a widely known tool for analyzing the packets present in a network. This software allows us to read the information exchanged in the selected interfaces as well as allows us to see in detail the information of each package. This tool is basic to this project and will be widely used in the research part of the protocols to extract information from how they work. We will also use it to verify that the attacks carried out are correct or to detect failures. [4]
- MSFConsole: This is a metasploit framework. It offers a console where many packages are centralized for different types of attacks. These packets range from network or traffic scanners to tools prepared to perform specific attacks. This is one of the tools that come preinstalled on Kali Linux. [5]
- ICSsploit: it is a tool very similar to MSFConsole but that gives us extra packages. This tool is not in Kali, so we must download it and we can run it from any computer with Python. [6]
- Scapy: this tool makes it easy for us to create packets and transmit them. This Python-based software will be used mainly for Packet Replay type attacks (identical or similar packets are crafted to the original system, and we send them to deceive you). [7]
- GitHub: it is a system that makes it easier for us to manage and control the versions of the code. It is a widely used platform today. Not only does it serve to manage our projects but, from GitHub desktop, we can see projects published by other people. [8]

#### **4. Results**

The tests carried out on the proposed protocols for this project have generally been satisfactory. On the other hand, in which we have encountered difficulties (either due to their security or complications to implement them), we have carried out an audit work. In it we define the problems encountered and how to proceed in case of wanting to carry out practices of these parts. Going into detail about the results obtained, regarding the S7Comm protocol, we have shown that, if attacks can be carried out and in a very simple way, since it is a protocol that does not incorporate enough security. As for the most secure version, the S7Comm Plus is found in communications between more modern Siemens equipment, such as those present in the laboratory. On this protocol, no attack has been made, although it has been entered into detail about its security and what it would be like to perform a malicious code injection attack on an automaton. As for Modbus, being an old industrial protocol, it does not contemplate any type of security and is the easiest to attack. In this section several attacks have been carried out, such as a Man in the Middle, a packet replay or using exploit frameworks. This protocol is interesting, although probably the least widespread today, to learn the concepts and as a first contact to industrial cybersecurity. As for Profinet, when using mainly Siemens equipment, it is complex to generate traffic of this protocol in such a way that allows us to carry out attacks. Because of this, a series of solutions have been proposed so that this protocol can be introduced into the cybersecurity laboratory. Finally, as for SCADA



systems, it is based on industrial protocols such as S7, Profinet, Modbus... so its security and the attacks that can be carried out are those seen in each section of these protocols.

```
=====
Valor Q0.0: True
Valor Q0.1: False
Valor Q0.2: True
Valor Q0.3: False
Valor Q0.4: False
Valor Q0.5: False
=====
¿Desea Continuar? Pulse n/N para salir o cualquier otra para continuar:
Introduzca la dirección donde quiere escribir (0-5): 1
Introduzca 0 si quiere ponerlo a false o 1 para true: 1
VALOR INTRODUCIDO CORRECTAMENTE.
=====
Valor Q0.0: True
Valor Q0.1: True
Valor Q0.2: True
Valor Q0.3: False
Valor Q0.4: False
Valor Q0.5: False
=====
¿Desea Continuar? Pulse n/N para salir o cualquier otra para continuar: |
```

Figure 1: Example of an attack developed in this project

## 5. Conclusions

The main conclusion of the project is that a cybersecurity laboratory can be implemented at ICAI imminently and without much cost. We started from a laboratory with a lot of industrial equipment that would also be used for this laboratory, since one of the main objectives has been to make the most of this infrastructure. The different protocols and the different attacks provide a broad knowledge about cybersecurity in industrial environments for the students who carry out these practices.

Another important conclusion is that the S7Comm Plus protocol and the main attacks subject to this protocol, have seemed too complex to perform in this laboratory. This is due to the high level of security provided by this modern version of the S7Comm. On the other hand, as for Profinet, attacks can be carried out, but we have shown that, compatible equipment is needed and that allows us to write data to automata. Profinet simulation software could also be considered, but they also come at a cost. In this protocol a small audit has been carried out to raise the problems and the options and how to be able to implement.

## 6. References

- [1] Elder Pérez, I., 2019. *Estudio y desarrollo de un enfoque de Pentesting para Sistemas de Control Industrial (ICS)*. Available at: <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/96949/6/eperezignTFM0619memoria.pdf> [Accessed 10 October 2020].
- [2] INCIBE. 2021. *Red de Excelencia Nacional de Investigación en Ciberseguridad*. Available at: <https://www.incibe.es/red-excelencia-idi-ciberseguridad> [Accessed 10 October 2021].
- [3] n.d. *Download page for Kali Linux*. Available at: <https://www.kali.org/downloads/> [Accessed 10 October 2020].
- [4] Wireshark.org. n.d. *Wireshark · Go Deep*. Available at: <https://www.wireshark.org> [Accessed 10 October 2020].
- [5] Offensive-security.com. n.d. Available at: <https://www.offensive-security.com/metasploit-unleashed/msfconsole/> [Accessed 17 October 2020].

- [6] GitHub. 2017. *tijldeneut/icssexploit*. Available at: <https://github.com/tijldeneut/icssexploit> [Accessed 20 October 2020].
- [7] community., P., n.d. *Scapy*. Scapy. Available at: <https://scapy.net/> [Accessed 22 November 2020].
- [8] Oleo Blanco, M., 2020. *Project repository*. Github. Available at: <https://github.com/miguelob/TFG-Ciberseguridad> [Accessed 8 July 2021].

## *Índice de la memoria*

<b>Capítulo 1. Introducción .....</b>	<b>6</b>
Motivación del proyecto .....	6
<b>Capítulo 2. Descripción de las Tecnologías.....</b>	<b>7</b>
<b>Capítulo 3. Estado de la Cuestión .....</b>	<b>9</b>
<b>Capítulo 4. Definición del Trabajo .....</b>	<b>11</b>
4.1 Justificación.....	11
4.2 Objetivos .....	11
4.3 Metodología.....	12
4.4 Planificación y Estimación Económica .....	12
<b>Capítulo 5. Sistema/Modelo Desarrollado .....</b>	<b>14</b>
5.1 Análisis de la topología y tráfico de red.....	14
5.2 S7comm.....	20
5.3 Modbus.....	30
5.3.1 Impersonating del Máster.....	37
5.3.2 Man in the Middle .....	40
5.3.3 Packet replay.....	45
5.4 Profinet .....	50
5.5 Sistemas SCADA .....	56
5.6 Inyección de código sobre PLCs .....	60
<b>Capítulo 6. Análisis de Resultados.....</b>	<b>61</b>
<b>Capítulo 7. Conclusiones y Trabajos Futuros.....</b>	<b>64</b>
<b>Capítulo 8. Bibliografía.....</b>	<b>67</b>
<b>ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS .....</b>	<b>71</b>
<b>ANEXO II: S7Comm plus.....</b>	<b>72</b>

## *Índice de figuras*

Figura 1: Ejemplo de ataque realizado en el proyecto.....	12
Figura 2: Diagrama Gantt del proyecto .....	13
Figura 3: Esquema de red del laboratorio.....	15
Figura 4: Salida del discovery de equipos de la red.....	17
Figura 5: Interfaz web de los switches Scalance .....	18
Figura 6: Configuración del port mirroring Scalance X208.....	18
Figura 7: Menú de configuración del Procurve 2626 .....	19
Figura 8: Configuración del Switch Procurve 2626 .....	20
Figura 9: Módulos S7 para node-red .....	22
Figura 10: Variables en el proyecto [8] .....	23
Figura 11: HMI del proyecto [8] .....	23
Figura 12: Conexión con el PLC y node-red.....	24
Figura 13: Configuración de las variables en node-red.....	24
Figura 14: Flow de escritura [9] .....	25
Figura 15: Output de la ejecución del ProyectoCyber.py [13] .....	26
Figura 16: Conexión ProcessSimulator y PLC [17] .....	27
Figura 17: Introducción IP y Slot [17].....	27
Figura 18: Petición de lectura S7Comm [19] .....	28
Figura 19: Petición de escritura S7Comm [19] .....	28
Figura 20: Respuesta de lectura S7Comm [19] .....	29
Figura 21: ModbusPal [21].....	31
Figura 22: QModMaster [22] .....	32
Figura 23: Acceder a los registros .....	32
Figura 24: Acceder a coils y crear tabla .....	33
Figura 25: Tabla de coils y valores.....	33
Figura 26: Conexión QModMaster .....	34

Figura 27: Configuración lectura de coils .....	34
Figura 28: Lectura de los coils .....	34
Figura 29: Tráfico Modbus .....	34
Figura 30: Datos de Modbus en query de escritura .....	35
Figura 31: Topología de red .....	36
Figura 32: Búsqueda Modbus en MSFconsole .....	37
Figura 33: show options de msfconsole .....	38
Figura 34: Establecimiento de parámetros y ejecución .....	38
Figura 35: Tabla de coils .....	38
Figura 36: Establecimiento de parámetros y escritura .....	38
Figura 37: Resultado de escritura .....	38
Figura 38: Tráfico del atacante .....	39
Figura 39: Flujo de tráfico normal .....	40
Figura 40: Flujo de tráfico con MiTM .....	40
Figura 41: Añadir targets .....	42
Figura 42: Activar ARP poisoning (1) .....	42
Figura 43: Activar ARP poisoning (2) .....	42
Figura 44: Filtro Ettercap .....	42
Figura 45: Cargar el filtro (1) .....	42
Figura 46: Cargar el filtro (1) .....	42
Figura 47: Petición de escritura interceptada .....	43
Figura 48: Petición de escritura interceptada y modificada .....	44
Figura 49: Mac en hexadecimal .....	45
Figura 50: Paquete completamente descompuesto .....	46
Figura 51: Paquete reconstruido .....	47
Figura 52: Paquete original .....	47
Figura 53: Profinet en Wireshark [29] .....	50
Figura 54: Profinet Commander [28] .....	51
Figura 55: Resultado del envío del paquete DCP [31] .....	52
Figura 56: Tráfico profinet entre Cognex y PLC [33] .....	54

---

Figura 57: Esquema de un sistema SCADA [34] .....	55
Figura 58: Sistema SCADA creado en Python [37] .....	57
Figura 59: Tráfico generado por el sistema SCADA [38] .....	58
Figura 60: Ejemplo de sistema SCADA con WinCC .....	58

## *Índice de tablas*

Tabla 1: Tabla con las IPs de cada puesto .....	16
Tabla 2: Capas del protocolo S7Comm [5] .....	21
Tabla 3: Tipos de registros Modbus [20] .....	29
Tabla 4: Campos del protocolo Modbus [20] .....	29
Tabla 3: Direcciones de las estaciones .....	54

# Capítulo 1. INTRODUCCIÓN

## *MOTIVACIÓN DEL PROYECTO*

Este proyecto nace conjuntamente de ICAI y de la Cátedra de Industria Conectada de ICAI. La universidad desde hace un par de años ofrece un máster de ciberseguridad, el cual cuenta con mucho prestigio gracias a los profesores y al contenido de este. Para mejorar la docencia y el conocimiento de los alumnos, se ha planteado la creación de unas prácticas de ciberseguridad.

Este trabajo de fin de grado trata de estudiar la creación, el desarrollo y la implementación de estas prácticas. En concreto se centra en la ciberseguridad en entornos industriales, ya que es un ámbito que está sometido a una gran revolución tecnológica. La importancia de la ciberseguridad es algo creciente, pero más todavía en estos ámbitos que todavía están en transformación digital.

El sector de la industria es algo más complejo en cuanto a ciberseguridad se trata. Muchos otros sectores se basan en proteger los datos, otros las comunicaciones, etc. La industria tiene muchos requerimientos que la seguridad de este sector sea crítica. Estos equipos presentes en las fábricas pueden ser explotados y causar graves daños, tanto económicos como humanos. Normalmente estas redes no suelen estar conectadas con el exterior por motivos obvios, pero esto no impide que se realicen ataques.

Otro de los motivos principales de este proyecto es que, al crear el laboratorio de ciberseguridad, ICAI pueda entrar en la Red Nacional de Laboratorios Industriales (RNLI). Esta red es un conjunto de laboratorios creados por distintas organizaciones y universidades. Esta red es una iniciativa del Instituto Nacional de Ciberseguridad (INCIBE) y apoyado por el Centro Nacional para la Protección de las Infraestructuras Críticas (CNPIC). El objetivo es que haya más investigación en este tema debido a su importancia y la necesidad de proteger nuestras industrias. [1]



## Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este proyecto se hace hincapié en varios protocolos de ámbito industrial. Como el trabajo se divide en estos protocolos, se explican cada uno en su apartado, ya que es necesario entender cómo funcionan antes de realizar los ataques.

En cuanto a herramientas, se emplean varios equipos físicos presentes en el laboratorio, así como distintas aplicaciones de software, las cuales también citaremos a continuación. Como se va a utilizar el laboratorio de la minifábrica de ICAI como base, muchos de los dispositivos que tiene se usarán. En concreto, se hará uso de los switches Scalance y de los PLCs S7-1500 y los HMI. En cuanto a los primeros, son unos switches de la marca de Siemens, los cuales están preparados para ser colocados en entornos industriales y trabajar con estos protocolos.

En cuanto a los PLC S7-1500, estos son en español Controladores Lógicos Programables, o también conocidos como autómatas programables. Estos equipos nos permiten la automatización en entornos industriales o de procesos fácilmente. En este caso se trata de la serie 1500 de la marca Siemens, los cuales son líderes en el sector. Es importante saber el modelo del que se dispone, ya que la seguridad de estos equipos depende en gran manera de la serie a la que pertenecen.

Por otro lado, para poder interactuar con los PLCs, tenemos los equipos, también de Siemens, conocidos como HMI. Estos equipos aportan una interfaz entre la máquina y el humano para poder interactuar con los equipos de forma siempre. Estos equipos están conectados en red con los autómatas e incorporan una pantalla para poder realizar ciertas gestiones. Al igual que los PLCs, son programables. Para gestionar y programar los equipos de Siemens, empleamos su software propietario, en nuestro caso la versión 16 de TIA Portal.

Otras herramientas importantes para el desarrollo de este trabajo son Wireshark y Scapy. Wireshark es una aplicación de captura y análisis de tráfico. Nos permite escoger la interfaz de la cual queremos obtener información y nos muestra todos los paquetes que circulan por

ella. Esto es importante para obtener información sobre el funcionamiento de los protocolos y también para realizar ataques como el *Packet Replay* (mandar un paquete capturado y modificarlo levemente para conseguir un comportamiento específico). En concreto para este ataque vamos a emplear la herramienta Scapy, la cual está basada en Python y nos permite fácilmente capturar, modificar, crear y mandar paquetes.

También empleamos la herramienta de GitHub para llevar un control de las versiones de los archivos empleados en el proyecto y para tener un repositorio de consulta para cualquier trabajo futuro basado en este proyecto.

El resto de las herramientas y equipos usados, serán citados en su capítulo correspondiente ya que no son de uso general como los citados anteriormente.

## Capítulo 3. ESTADO DE LA CUESTIÓN

El sector de la ciberseguridad es de los que más está creciendo desde hace varios años. Esto se debe a que la revolución digital en la que vivimos está teniendo un crecimiento exponencial. También se debe a que puede aplicar a un gran número de sectores muy distintos. Los consumidores de cualquier producto tecnológico también son más conscientes de la importancia de la ciberseguridad para sus vidas. Ya hoy en día todos somos conscientes de las pautas básicas para proteger nuestra información, datos, dinero, etc.

La importancia de la ciberseguridad, entre otras causas, ha crecido debido a que se están digitalizando cada vez más industrias y sectores con datos críticos. Uno de estos sectores es la industria, donde los datos y los procesos ahora se controlan desde máquinas conectadas y, a pesar de que no suelen tener conexión con el exterior, es importante que estén protegidas.

A priori puede parecer que estas industrias no necesitan tomar medidas de seguridad ya que como muchas no están conectadas con el exterior, los ataques son irrealizables. Pues a pesar de que esto haga que ser atacado sea mucho más complejo, sí que se pueden realizar ataques y como ejemplo está el ataque que se realizó hacia una planta secreta de enriquecimiento de uranio para armas nucleares. Este ataque es un buen ejemplo de todas las fases en las que se divide un ataque. Se realizó una fase muy importante de ingeniería social para descubrir información sobre los equipos de la planta (autómatas programables o también conocidos como PLCs) e información sobre el punto de recogida de los trabajadores de dicha planta. Con todo este proceso, se consiguió desarrollar un ataque con el que se ralentizaba la centrifugadora de uranio sin que saltara ninguna alarma y mostrando datos normales a los ingenieros de la planta. Con se consiguió hacer que este productor perdiese las características necesarias para realizar armas y tardaron en darse cuenta varios años [2].

Estos ataques se realizaron hacia los PLCs de Siemens, los mismos que tenemos presentes en el laboratorio de la minifábrica de ICAI y los cuales usaremos para llevar a cabo ataques y pruebas de seguridad. La importancia que implica proteger estas industrias es muy alta, por razones obvias. Hay muchos protocolos que se usan en comunicaciones entre máquinas

enfocadas a la industria y en ellos reside gran parte del peso de la seguridad de las infraestructuras. Los protocolos más importantes serán estudiados en este proyecto con el fin de crear un laboratorio puntero en el estudio de la ciberseguridad en el ámbito industrial. Los objetivos de este proyecto están explicados al detalle en ese apartado.

## **Capítulo 4. DEFINICIÓN DEL TRABAJO**

### **4.1 JUSTIFICACIÓN**

Una vez listadas las motivaciones del proyecto, este apartado es muy similar. La principal justificación del proyecto que se va a desarrollar es la poca presencia de laboratorios de ciberseguridad industrial. Gracias al INCIBE cada vez se desarrollan más, pero al ser algo crítico, pensamos que son insuficientes y podemos aportar a ello.

También no podemos dejar de lado que, este laboratorio, será usado principalmente por estudiantes y profesores de la escuela de distintos ámbitos para desarrollar sus conocimientos en esta área. Esto es importante para la universidad ya que es una oportunidad muy importante para mejorar más todavía la calidad de su enseñanza y seguir apostando por la parte práctica de la misma. En muy pocos centros se realizan tantas prácticas y menos todavía en ámbitos de ciberseguridad con equipos de verdad y no simulados.

Muchos ingenieros, ya sean de telecomunicaciones o industriales, acaban trabajando en temas relacionadas con la industria del automóvil, farmacéuticas, etc. Debido a esto, son conocimientos que son considerados importantes para nuestro desarrollo intelectual de cara al mundo laboral.

### **4.2 OBJETIVOS**

Aprovechando la infraestructura y los equipos presentes en el laboratorio de la minifábrica de ICAI, se va a hacer un estudio de los principales protocolos presentes y los ataques que se pueden llevar a cabo. Haciendo un estudio de estos protocolos presentes en el laboratorio y de los principales protocolos de la industria, decidimos investigar los siguientes: S7Comm, Profinet, Modbus y unos apartados extra para inyecciones de código sobre PLCs y ataques sobre sistemas SCADA.

Con esto, se busca poder tener una batería de prácticas que los alumnos, principalmente del máster de ciberseguridad, puedan desarrollar la teoría vista en clase y ganar conocimiento sobre el ámbito de la ciberseguridad en la industria.

Otro de los objetivos principales, como ya se ha citado en el apartado de justificación, es conseguir que ICAI entre dentro de los centros con un laboratorio de ciberseguridad en la red creada por INCIBE.

### **4.3 METODOLOGÍA**

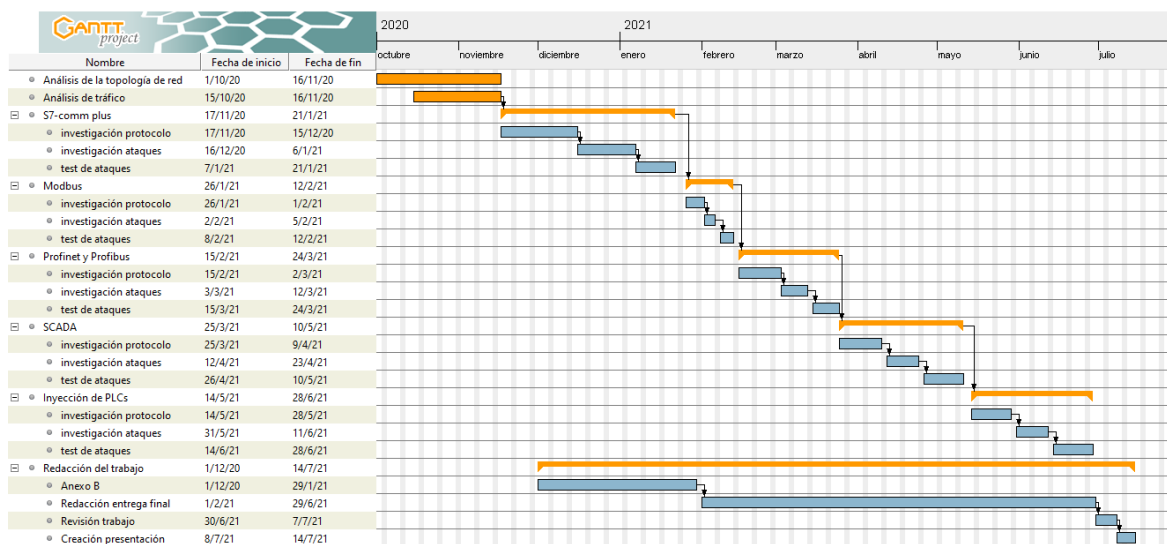
La metodología es muy simple y hemos seguido la que indicamos al principio del proyecto en el Anexo B. Antes de estudiar los protocolos, se debe hacer un estudio precioso de la topología de red presente y del tráfico, ya que es preciso solucionar estos problemas lo antes posible y poder continuar con los protocolos sin mucha complicación.

En cuanto a los protocolos que vamos a tratar (citados en el apartado anterior), en todos se va a emplear la misma metodología. Primero, hay una fase de investigación en la que se conoce el protocolo, sus funcionalidades, como se compone y, sobre todo, los mecanismos de seguridad que este conlleva. Esto va seguido de otra fase de investigación, pero esta vez de ataques. Es importante ver si ya hay ataques realizados sobre estos protocolos, ya que nos facilita mucho el trabajo y nos puede servir de apoyo para otros tipos de ataques que no se hayan realizado. También es importante, ya que no queremos realizar ataques que inhabiliten los equipos, por lo que es importante conocerlos antes. Por último, se realizan los ataques y se sacan ciertas conclusiones de ellos, como puede ser el nivel de seguridad, la dificultad y, sobre todo y lo más importante, como poder protegerse ante estos ataques.

### **4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA**

Para mostrar la planificación del proyecto, muestro el diagrama Gantt que se realizó para el Anexo B. En este se muestra las partes en las que se divide este proyecto y su duración en el tiempo.

En el diagrama, se puede observar las principales partes del proyecto, así como las partes en las que se divide, explicadas en el apartado anterior. En cuanto al reparto, la duración y las fechas inicio, en algunos de los apartados no se han podido seguir tal y como se estableció, ya que este diagrama fue anterior al inicio del proyecto.



*Figura 2: Diagrama Gantt del proyecto*

En cuanto a la planificación económica, no es un proyecto en el que tenga mucho peso. Esto se debe a que partimos de la infraestructura y los equipos que ya hay presentes en el laboratorio e intentamos que todos los ataques que se realizan aprovechen estos equipos. En los protocolos en los que no se pueda, se busca como alternativa simuladores de código abierto.

Aun así, como se explica a continuación, hay equipos que son interesantes y otros necesarios. Por ejemplo, para el desarrollo del proyecto se ha necesitado un Switch Procurve 2626, el cual no ha costado nada ya que nos lo han cedido por Parte de STIC. A pesar de esto, si se quisiera realizar este proyecto de cero, se necesitaría como mínimo un PC, un PLC y un Swtich o Hub, estimando el precio como mínimo en unos 3000 €. Como se citará en las conclusiones, también se podría plantear para un futuro comprar otros equipos para que las pruebas realizadas en simulador se hagan con equipos reales. También se puede plantear obtener PLCs más antiguos y aprovechar su poca seguridad para ver otros ataques.

## Capítulo 5. SISTEMA/MODELO DESARROLLADO

En este capítulo se desarrollará con detalle el trabajo realizado. Para ello, se dividirá en varios sub-apartados de acuerdo con lo especificado en el diagrama de Gantt del Anexo B:

- 5.1 Análisis de la topología y tráfico de red.
- 5.2 S7comm.
- 5.3 Modbus.
- 5.4 Profinet.
- 5.5 Sistemas SCADA.
- 5.6 Inyección de código en PLCs.

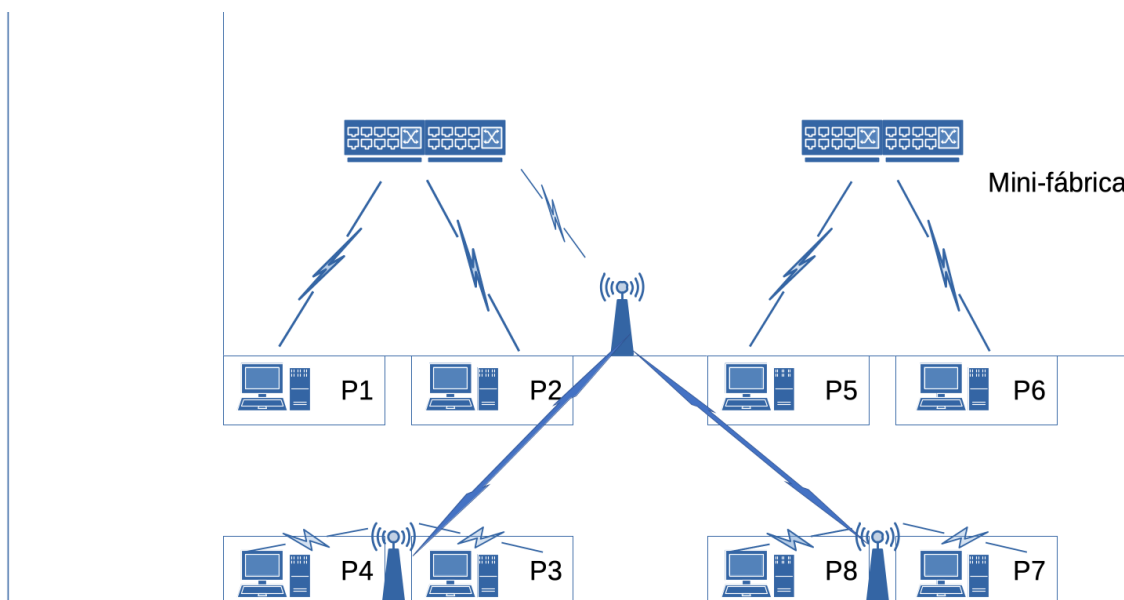
### ***5.1 ANÁLISIS DE LA TOPOLOGÍA Y TRÁFICO DE RED***

Esta fase es clave para el desarrollo del proyecto. Esto se debe a que, como vamos a implementar una serie de ataques sobre la red existente del laboratorio de la “minifábrica” de ICAI, debemos de conocer los equipos que hay presentes. Es importante no solo saber que equipos hay, sino que también es importante saber la seguridad que tienen por defecto y el tráfico que encontramos en la red.

La seguridad de los equipos nos interesa, especialmente de los equipos como los switches Scalance X208 que hay en el lab, ya que necesitamos acceder a su configuración y ver si los podemos configurar algunos puertos con port mirroring (reenvío de paquetes de un puerto a otro). Para ello, es importante conocer primero la topología de red. El laboratorio está dividido en dos secciones: los puestos de trabajo y la minifábrica. Los puestos de trabajo son 8 y cada uno de ellos tiene un ordenador, un PLC S7-1500 y un HMI de Siemens. Los puestos que están pegados a la minifabrica están conectados directamente a su red y el resto, se conectan a través de unos módulos wifi. Lo importante es saber que la red del laboratorio está dividida en dos, la parte izquierda y la derecha. Cada uno de los lados tiene dos switches Scalance donde se conectan 4 puestos con todos sus equipos y el resto de los elementos del



laboratorio (cámaras Cognex, brazos robóticos ABB...). A continuación, se muestra este esquema de red:



*Figura 3: Esquema de red del laboratorio*

También es importante conocer las IPs presentes en los equipos de cada puesto, ya que cuando vayamos a realizar algún tipo de análisis de tráfico u ataque nos harán falta. A continuación, se muestra una tabla con las direcciones lógicas de los ordenadores, PLCs y HMIs de cada puesto:

Puesto	IP PC	IP PLC	IP HMI
<b>P1</b>	192.168.56.5	192.168.56.15	192.168.56.25
<b>P2</b>	192.168.56.6	192.168.56.16	192.168.56.26
<b>P3</b>	192.168.56.2	192.168.56.12	192.168.56.22
<b>P4</b>	192.168.56.1	192.168.56.11	192.168.56.21
<b>P5</b>	192.168.56.7	192.168.56.17	192.168.56.27

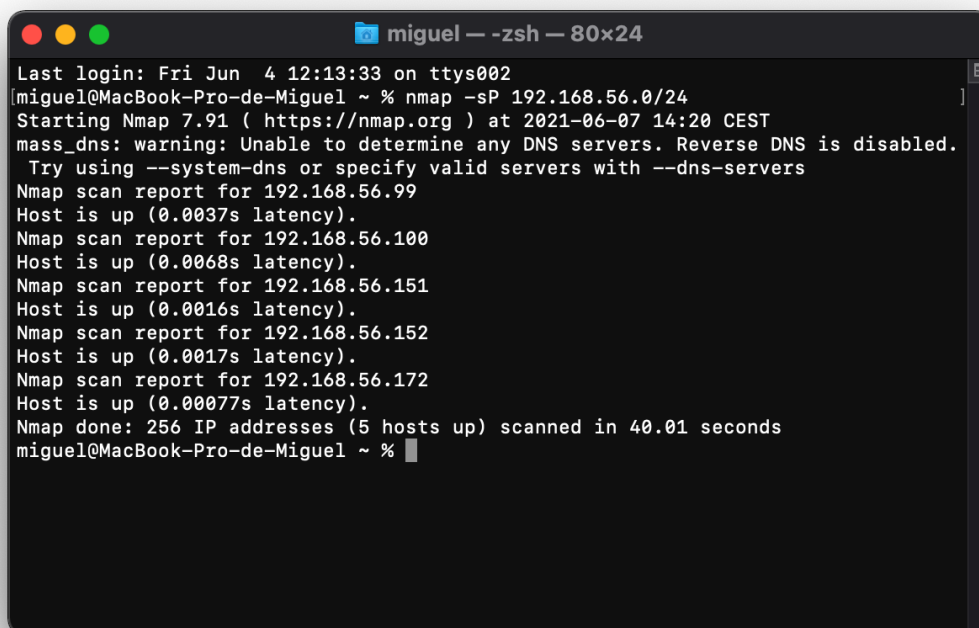
<b>P6</b>	192.168.56.8	192.168.56.18	192.168.56.28
<b>P7</b>	192.168.56.4	192.168.56.14	192.168.56.24
<b>P8</b>	192.168.56.3	192.168.56.13	192.168.56.23

*Tabla 1: Tabla con las IPs de cada puesto*

Una vez visto esto, sabemos que la red del laboratorio es la 192.168.56.0/24. Si queremos ver una lista de los dispositivos que hay accesibles en la red, podemos utilizar la herramienta Nmap [3] para escanearlos. Para ello, debemos instalar esta herramienta e introducir el comando correspondiente por la consola. Para instalarlo, yo voy a indicar el procedimiento para MacOS, que es el sistema donde realizo el proyecto. En este sistema es recomendable usar HomeBrew [4] para poder instalar paquetes (equivalente de apt install de Linux). Para instalar este gestor de aplicaciones, es tan fácil como introducir el siguiente comando en el terminal:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

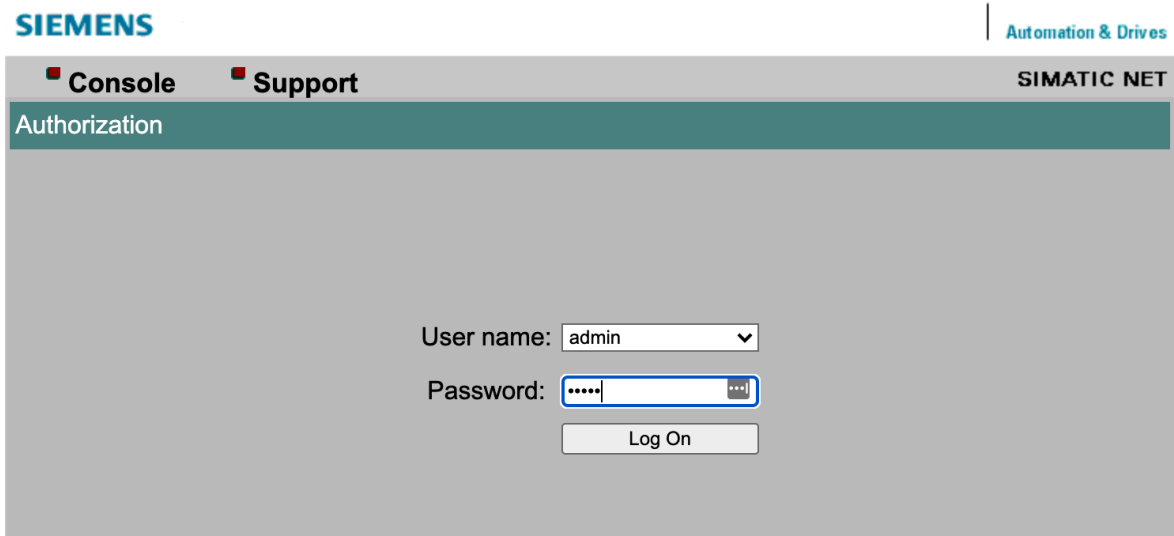
Una vez tenemos instalado este gestor, podemos instalar las aplicaciones con brew install [aplicación]. Por lo que para nmap basta con ejecutar brew install nmap. Para ejecutar nmap, basta con poner nmap -flags/opciones en el terminal. Para saber que opciones y flags son necesarias para lo que se busca, recomiendo mirar la documentación en la referencia [3] o usar la ayuda con -help. Para hacer un discovery de los dispositivos de la red, empleamos el comando nmap -sP 192.168.56.0/24. Obtenemos la siguiente lista de dispositivos (importante destacar que el comando ha sido ejecutado desde el puesto 1 y por lo tanto, los dispositivos accesibles desde la otra sección del laboratorio no se muestran):



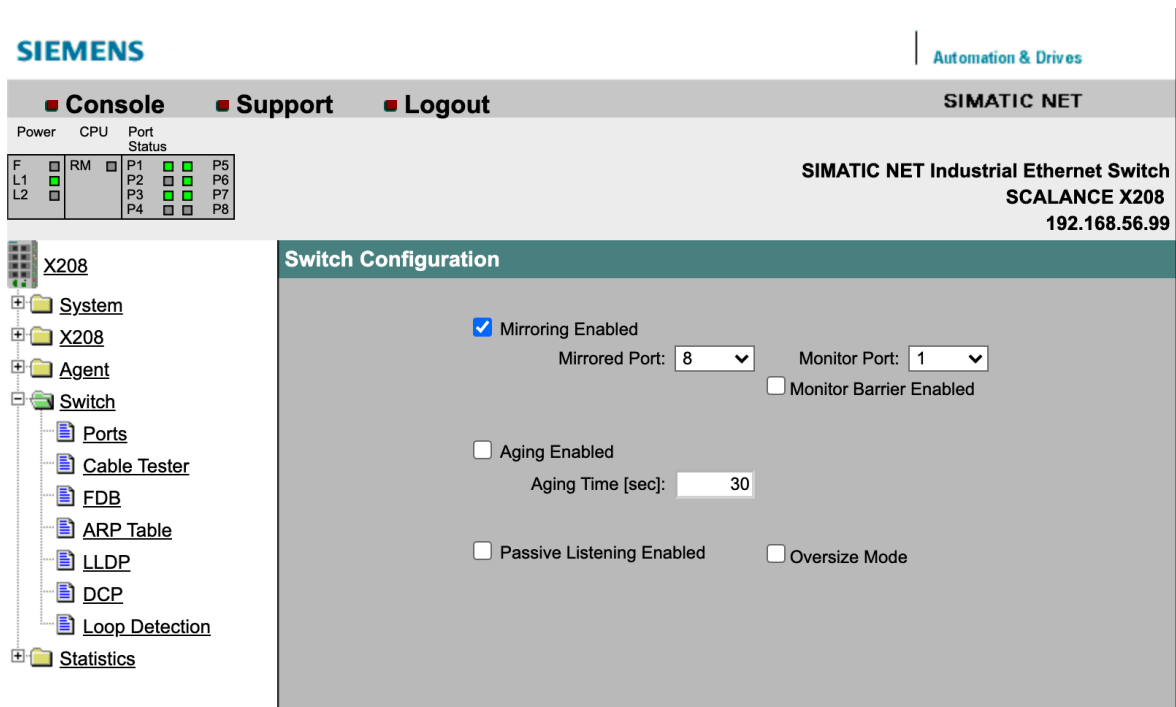
```
miguel — -zsh — 80x24
Last login: Fri Jun  4 12:13:33 on ttys002
miguel@MacBook-Pro-de-Miguel ~ % nmap -sP 192.168.56.0/24
Starting Nmap 7.91 ( https://nmap.org ) at 2021-06-07 14:20 CEST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled.
Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.56.99
Host is up (0.0037s latency).
Nmap scan report for 192.168.56.100
Host is up (0.0068s latency).
Nmap scan report for 192.168.56.151
Host is up (0.0016s latency).
Nmap scan report for 192.168.56.152
Host is up (0.0017s latency).
Nmap scan report for 192.168.56.172
Host is up (0.00077s latency).
Nmap done: 256 IP addresses (5 hosts up) scanned in 40.01 seconds
miguel@MacBook-Pro-de-Miguel ~ %
```

*Figura 4: Salida del discovery de equipos de la red*

Una vez que tenemos las IPs devueltas por el nmap, podemos probar a acceder a estas IPs desde el navegador, ya que los switches suelen tener interfaz web para configurar. Aquí nos encontramos uno de los errores más típicos de seguridad y de los más fáciles de salvaguardar. Las credenciales para acceder a la configuración son las por defecto (usuario: admin contraseña: admin). Desde esta web podríamos cambiar este usuario y contraseña y tomar el control de estos equipos y bloquear temporalmente el acceso a los propios ingenieros de la planta. Aun así, lo que más nos interesa es ir a la configuración de los puertos para hacer mirroring de algún puerto para poder obtener los paquetes que hay en esa interfaz.



*Figura 5: Interfaz web de los switches Scalance*



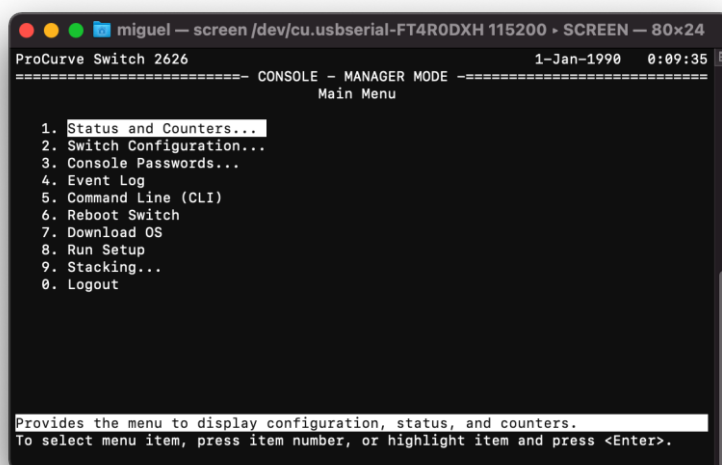
*Figura 6: Configuración del port mirroring Scalance X208*

Es importante saber que este switch existe y es el presente en la red del laboratorio, pero al ser un switch industrial que suele trabajar con protocolos industriales, hay ciertas cosas que nos limita. Por ejemplo, cuando se intenta hacer port mirroring con s7comm o profinet, el switch filtra estos paquetes para que no se puedan ver desde fuera. Para solucionar esto,

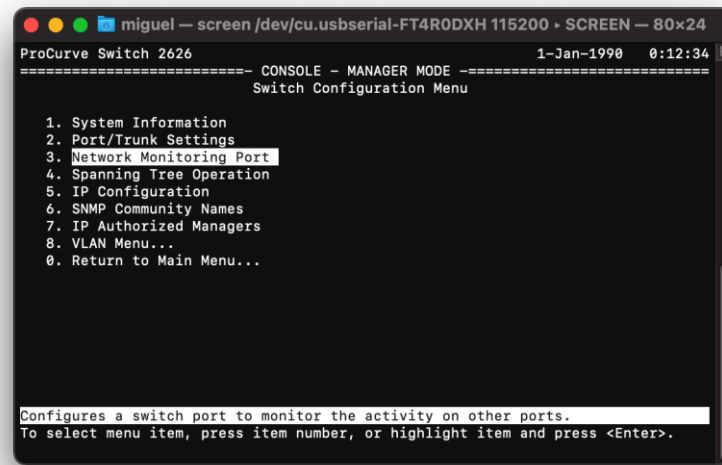
hemos colocado un switch HP Procurve 2626 el cual configuraremos para poder obtener estos paquetes de protocolos industriales.

Este switch se configura por interfaz serial, por lo que podemos utilizar varios programas para acceder a este terminal, en mi caso, voy a emplear la propia consola de MacOS para ello. Para poder conectarnos, primero debemos obtener la ruta donde se monta el adaptador usb-serial, para ello si ponemos en el terminal `ls /dev/cu.*` se nos devuelve una lista de los dispositivos de comunicación externos. De esta lista sacamos la ruta de nuestro adaptador, que en mi caso es `/dev/cu.usbserial-FT4R0DXH`. Una vez tenemos esta información solo queda saber los baudios a los que transmite el switch, en este caso 115200. Por último, nos conectamos con `screen /dev/cu.usbserial-FT4R0DXH 115200` y se nos abre la consola del switch. Una vez dentro, vamos a acceder a la configuración básica escribiendo `menu` sobre la línea de comandos.

Para configurar el port mirroring debemos primero acceder a *Switch Configuration* y luego a *Network monitoring port*. En mi caso, he establecido como monitor ports el 1 y 2 y el monitoring el 3. Esto quiere decir que cualquier paquete que entre o salga por el puerto 1 y 2 será reenviado por el 3.



*Figura 7: Menú de configuración del Procurve 2626*



*Figura 8: Configuración del Switch Procurve 2626*

## 5.2 S7COMM

S7Comm es un protocolo propietario de Siemens para las comunicaciones entre sus equipos (PLCs, HMIs ...). Este protocolo fue introducido con la familia de PLCs S-300 y S-400 y desde entonces sigue en desarrollo constante. Al ser un protocolo propietario, la cantidad de información que tenemos disponible es muy limitada. También hay que indicar que como estos equipos trabajan en sectores donde los ataques pueden llegar a ser catastróficos, el secretismo de este es muy importante. Un ejemplo del impacto que puede tener un ataque sobre estos sistemas fue el ataque que se lanzó contra una planta de enriquecimiento de uranio en Irán. Se consiguió inyectar un programa malicioso sobre unos PLCs, consiguiendo modificar unos parámetros de la centrifugadora a la vez que se mandaban datos correctos a las estaciones de ingeniería [2].

Este protocolo manda los datos en el payload de COTP (Connection Oriented Transport Protocol). Las capas del protocolo son las siguientes:

	<b>OSI layer</b>	<b>Protocol</b>
7	Application Layer	S7 communication
6	Presentation Layer	S7 communication
5	Session Layer	S7 communication
4	Transport Layer	ISO-on-TCP (RFC 1006)
3	Network Layer	IP
2	Data Link Layer	Ethernet
1	Physical Layer	Ethernet

*Tabla 2: Capas del protocolo S7Comm [5]*

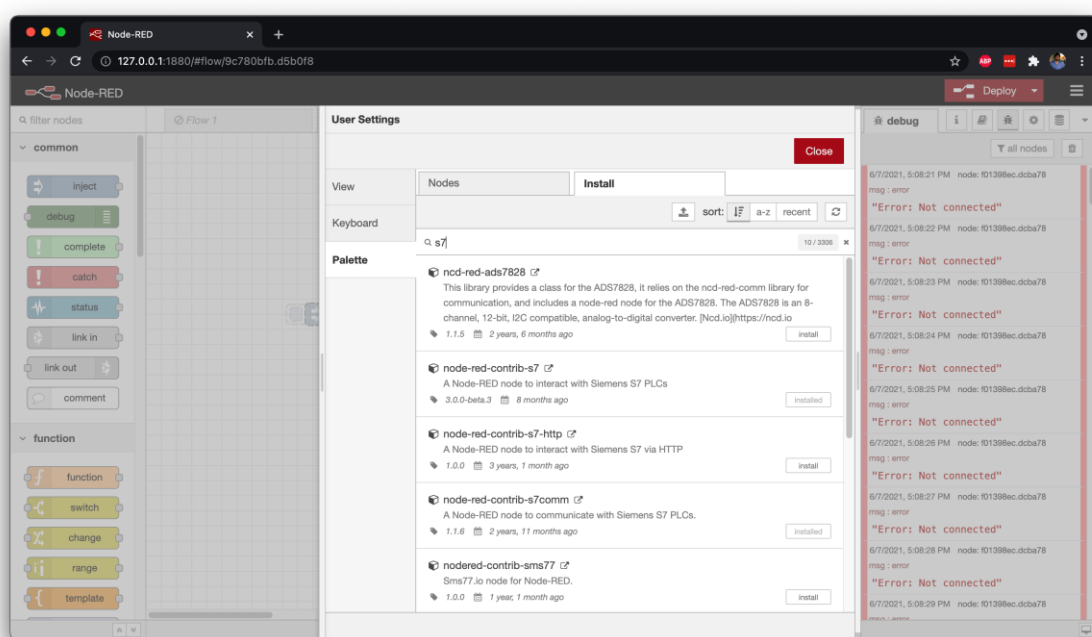
Lo primero que hacemos siempre es analizar el tráfico para ver en concreto que versión del protocolo se encuentra en la red. Con cualquier proyecto del TIA Portal V16 (Software de Siemens para programar sus PLCs y HMIs), al cargar el software en los equipos o la comunicación entre ellos, detectamos que este tráfico se trata de S7Comm plus. Esta es una versión más moderna del conocido S7Comm y que integra muchos mecanismos de seguridad. Debido a esto, vamos a centrarnos en el S7comm que, aunque no lo usen por defecto, se pueden llevar a cabo algunos ataques. También al tratarse de equipos modernos (S7-1500) los ataques son más complejos y menos efectivos (ya que implementan más métodos de seguridad). Aun así, al final de este apartado citaré un poco en que consiste la seguridad del protocolo S7Comm plus.

En cuanto al funcionamiento de este protocolo, es importante fijarse en la Tabla 3, que nos aporta mucha información sobre las capas del protocolo. Lo más importante es saber que crea un socket TPC y está a la escucha del puerto 102. Esta información la podemos obtener de varias formas: leyendo la información del protocolo o analizando los paquetes en Wireshark (se verá adelante en los ataques).

Es importante destacar que los ataques que se van a desarrollar en este apartado todos tienen como objetivo las variables de la memoria del PLC, ya que las entradas input/output analógicas requieren acceso físico al equipo. Aun así, es interesante ya que podemos leerlas

remotamente y en las memorias digitales se puede escribir, afectando o modificando el comportamiento que los ingenieros programaron al PLC.

El primer ataque que vamos a documentar es el más sencillo de llevar a cabo, ya que se realiza a través de una interfaz. El programa que se utilizará para ello es Node-Red [6]. Esta herramienta nos facilita la comunicación entre dispositivos de hardware y apis. Para instalarlo en MacOS se necesitan varios paquetes, los cuales se muestran en la referencia [7]. Una vez instalada y arrancada la aplicación, debemos instalar varios módulos para comunicarnos a través de S7 con los PLCs. En el apartado de configuraciones debemos buscar en palette los siguientes dos módulos:



*Figura 9: Módulos S7 para node-red*

Ahora vamos a configurar un flow para poder leer las entradas de un proyecto del PLC. Para este proyecto, voy a emplear uno de prueba con la pequeña maqueta de la cinta transportadora presente en el laboratorio. El proyecto está subido en el repositorio de GitHub [8].



Este proyecto del TIA Portal tiene una serie de variables, con las cuales a través del HMI podemos controlar varios parámetros de la maqueta de la cinta transportadora. También podemos ver señales que la cinta nos manda (sensores de proximidad y un sensor que se activa y desactiva con el movimiento de esta). El control de estas variables y de la cinta, en un entorno realista se realizaría desde un sistema SCADA o desde un HMI, como sería en este caso.

1	[-]	Clock_Byte	Byte	%MB0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	[-]	Clock_10Hz	Bool	%MO.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	[-]	Clock_5Hz	Bool	%MO.1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
4	[-]	Clock_2.5Hz	Bool	%MO.2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
5	[-]	Clock_2Hz	Bool	%MO.3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
6	[-]	Clock_1.25Hz	Bool	%MO.4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
7	[-]	Clock_1Hz	Bool	%MO.5	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
8	[-]	Clock_0.625Hz	Bool	%MO.6	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
9	[-]	Clock_0.5Hz	Bool	%MO.7	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
10	[-]	poGiro	Bool	%Q2.6	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
11	[-]	poSentido	Bool	%Q2.7	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
12	[-]	piTaco	Bool	%I3.5	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
13	[-]	mClock	Bool	%M10.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
14	[-]	mClockPermiso	Bool	%M10.1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figura 100: Variables en el proyecto [8]

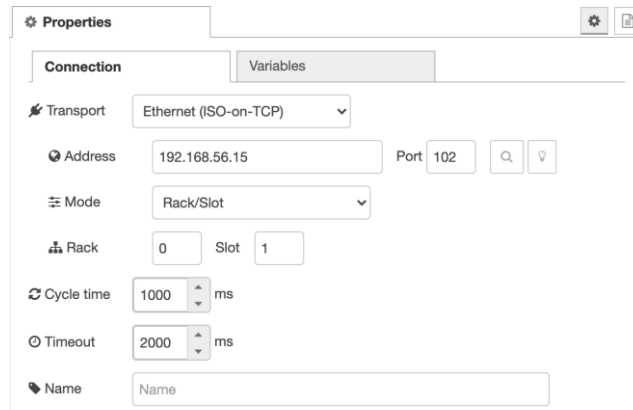


Figura 11: HMI del proyecto [8]

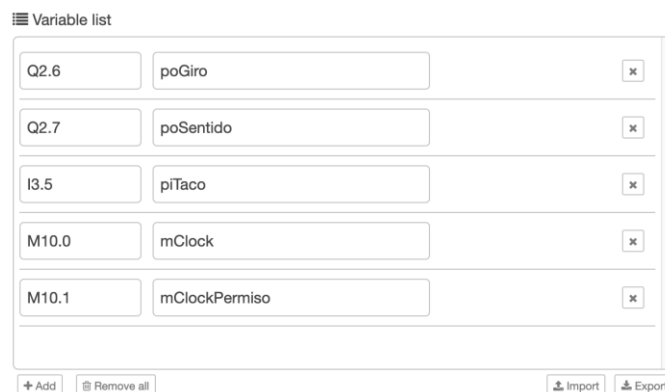
En un ataque realista, el atacante no sabría que variables son las que necesita para cambiar el funcionamiento del programa. En este caso, analizando el tráfico podríamos ver que variables son las que se están utilizando (en este caso es complicado ya que se comunican a través del protocolo S7Comm-plus) y cambiar sus valores. Otra opción más sencilla, es leer todas las variables posibles, y deducir cuales son las que se están utilizando en el proyecto.

Para este trabajo, vamos a partir de que se saben las variables son conocidas. Lo primero que vamos a ver es la lectura de estas variables desde Node-Red. Para ello, voy a utilizar los bloques S7in (del paquete PLC instalado anteriormente) y el debug para que nos muestre por

la consola del framework las salidas. Para ello, primero debemos establecer la conexión con el PLC en el bloque de S7 in. En este punto se nos pedirá la IP del PLC y lo que es muy importante, el rack y el slot donde se encuentra la CPU del PLC. Por defecto el slot está a 2, pero como el PLC del lab es un S7-1500 debemos ponerlo a 1. Por otro lado, en la pestaña de variables, hay que añadir todas las variables que queramos.



*Figura 12: Conexión con el PLC y node-red*

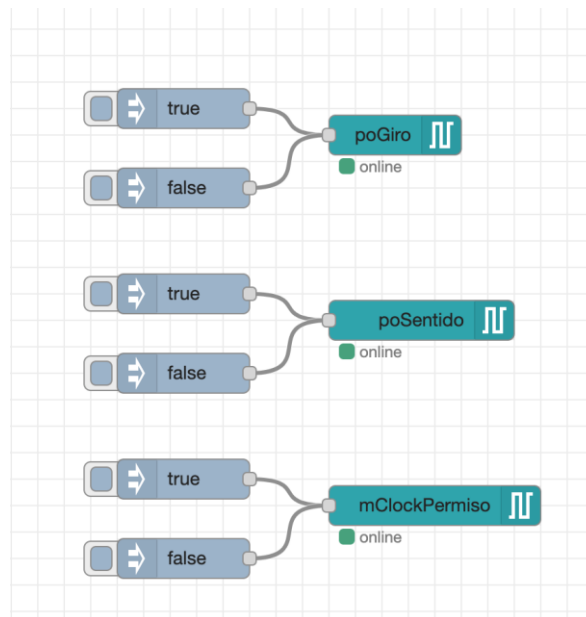


*Figura 13: Configuración de las variables en node-red*

Una vez hecho todo esto, establecemos en el S7 el modo “todas las variables” y al darle a deploy se carga el flow. También se puede establecer que solo muestre los valores cuando ha habido algún cambio en alguno de ellos (esta es la opción recomendable para poder ver con detalle lo que sucede). Para ponerlo a prueba, desde el HMI controlamos a mano la maqueta de la cinta, encendiendo el motor, cambiando sentidos y activando la señal del reloj. Si todo es correcto, en la pestaña debug, deberíamos ver los cambios en estas variables de la siguiente forma:

```
{"poGiro":true,"poSentido":true,"piTaco":false,"mClock":false,"mClockPermiso":false}
```

Una vez visto esto, procedemos a la parte de escritura. Hay que destacar que hay ciertas variables sobre las que no podemos escribir en el PLC, como pueden ser las tipo I. Aun así, como las de tipo Q las podemos modificar, conseguimos controlar esta maqueta con nuestro PC. Esto llevado a escala real y a una fábrica de verdad, las consecuencias podrían ser catastróficas. A continuación, se muestra el flow creado para este apartado, el cual nos da el control sobre todas las variables a las que tendría acceso el ingeniero desde el HMI:



*Figura 11: Flow de escritura [9]*

Para cambiar los valores de estas variables, basta con hacer click sobre true o false de la que queramos cambiar. Tanto el flow de escritura como el de lectura se encuentran en el repositorio de GitHub [9]. También en la referencia [10] se puede ver un breve vídeo del resultado de este ataque.

Como hemos visto, este ataque realiza las funciones que hace el HMI, por lo que este tipo de ataque es conocido como *impersonating*, en concreto un impersonating del HMI. De este tipo de ataque, voy a explicar dos formas extra. La primera es de forma programática (Python) y la última es otra herramienta parecida a node-red pero centrada en PLCs (se explicará menos extenso). Para estos dos últimos, voy a implementar los ataques usando el PLCSIM, que es la herramienta de TIA Portal que nos permite simular en red un PLC. Esto es interesante ya que, a la hora de realizar prácticas de ciberseguridad, siempre es más seguro realizarlo sobre un equipo virtualizado. También es interesante ver este método debido a la

situación actual en la que nos encontramos. La pandemia nos ha forzado a una transformación digital de todos los ámbitos, entre ellos la enseñanza. Con esta herramienta, se podrían impartir las prácticas de laboratorio de forma remota sin perder detalle sobre el funcionamiento de los PLCs.

Para el impersonating con Python, voy a emplear una herramienta extra, que me permite acceder a este PLC virtual desde cualquier equipo de la red (no solo desde el equipo donde se está virtualizando). Esta herramienta es NetToPLCSim [11]. El equipo donde está el PLCSIM tiene IP 192.168.1.10, por lo que, en Python, cuando nos intentemos conectar al PLC, pondremos esta IP. Este software se encarga de mandar luego los paquetes al socket correspondiente de PLCSIM.

La librería de Python necesaria para poder comunicarnos a través de S7comm es Snap7. Esta librería la instalamos fácilmente con el comando `pip install snap7`. Una vez instalado este paquete, hay que instalar archivos extra para poder establecer la comunicación. En caso de Windows necesitamos los archivos `snap7.dll` y `snap7.lib` de la página oficial de snap7 [12] y añadirlos al path. En MacOS, podemos instalar estos paquetes con `brew install snap7`.

Una vez hecho esto, ya podemos proceder a escribir código. Este código lo podemos encontrar en el repositorio de GitHub en la siguiente referencia [13]. El funcionamiento de este código es muy simple y la interfaz es por consola. Al ejecutarlo se muestra las 5 variables que usamos en el proyecto del TIA y sus respectivos valores. El programa pide que se introduzca por teclado cual queremos modificar y el nuevo valor. Una vez introducido, nos confirma la actualización y nos vuelve a mostrar las variables.

```
=====
Valor Q0.0: True
Valor Q0.1: False
Valor Q0.2: True
Valor Q0.3: False
Valor Q0.4: False
Valor Q0.5: False
=====
¿Desea Continuar? Pulse n/N para salir o cualquier otra para continuar:
Introduzca la dirección donde quiere escribir (0-5): 1
Introduzca 0 si quiere ponerlo a false o 1 para true: 1
VALOR INTRODUCIDO CORRECTAMENTE.
=====
Valor Q0.0: True
Valor Q0.1: True
Valor Q0.2: True
Valor Q0.3: False
Valor Q0.4: False
Valor Q0.5: False
=====
```

*Figura 12: Output de la ejecución del ProyectoCyber.py [13]*

También se puede comprobar que las escrituras y lecturas son correctas en las tablas de observación y activando la visión en tiempo real. Este código se puede fácilmente adaptar para crear un sistema SCADA como se verá en ese apartado del documento. Añadiendo una interfaz gráfica y unos threads para leer y escribir a la vez, tendríamos nuestro propio sistema estado muy sencillito.

Usando la librería Snap7 también podemos escribir un código que nos permita controlar la maqueta de la cinta transportadora (modificamos los valores de las variables Q2.6 y Q2.7). Este código se encuentra en GitHub en la siguiente referencia [14] y en la referencia [15] hay un vídeo demostrando su funcionamiento. El código es prácticamente idéntico al explicado anteriormente [16] con algunas modificaciones para modificar estos valores.

Otra herramienta con la que podemos hacer esto mismo es ProcessSimulator [17] que solo está disponible en Windows. Para conectarlo con el PLCSIM (o un PLC real), en la pestaña de conexión tenemos que añadir el PLC:

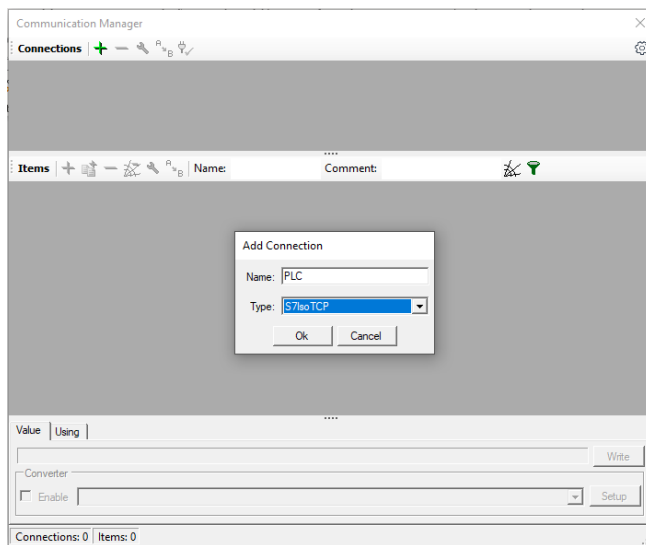


Figura 13: Conexión ProcessSimulator y PLC [17]

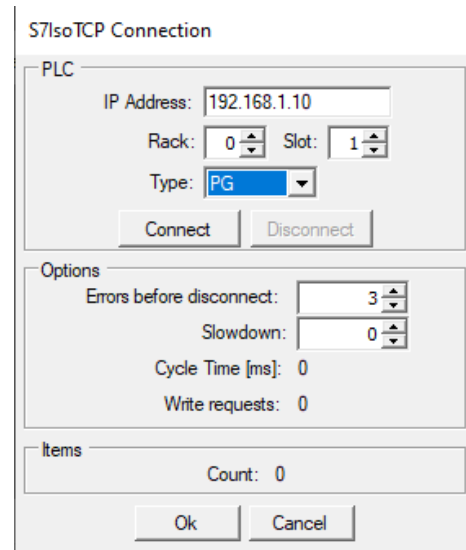


Figura 14: Introducción IP y Slot [17]

Una vez establecida la conexión, tenemos que añadir un *item* en este caso vamos a añadir de Q0.0 a Q0.5. Una vez agregadas las variables, en la parte inferior nos sale el valor actual de la variable seleccionada y en write podemos cambiar el valor por el que queramos. Para comprobar que esto funciona correctamente, podemos ver en tiempo real la tabla de variables en TIA Portal y asegurarnos de que está funcionando correctamente. Este comportamiento y configuración del programa se puede ver en el siguiente vídeo [18]. Para controlar la

maqueta, ocurre lo mismo que con Snap7, tendríamos que añadir las variables Q2.6 y Q2.7 y cambiar sus valores.

También es importante ver el contenido de los paquetes S7comm. Para ello, vamos a analizar el tráfico capturado con wireshark durante la prueba con node-red [19]. Observando la información de los paquetes, podemos ver que está basado en TCP, COTP y luego ya el *payload* de S7comm. También se observa, y es muy importante, que los datos del paquete no están cifrados, ya que corresponde la información que se envía con su correspondiente valor en hexadecimal (lo cual no ocurre en S7Comm plus ya que la información está cifrada). Si vemos un paquete de escritura y de lectura, se puede entender fácilmente S7Comm. También se puede apreciar cómo, en un paquete, es capaz de realizar varias lecturas o escrituras (nunca las dos a la vez) de varias variables. El protocolo siempre responde con ACKs y en el caso de la petición de lectura, en esta confirmación es donde se encuentran los valores devueltos por el PLC.

```

S7 Communication
  Header: (Job)
    Protocol Id: 0x32
    ROSCTR: Job (1)
    Redundancy Identification (Reserved): 0x0000
    Protocol Data Unit Reference: 2278
    Parameter length: 38
    Data length: 0
  Parameter: (Read Var)
    Function: Read Var (0x04)
    Item count: 3
    Item [1]: (I 3.0 BYTE 1)
      Variable specification: 0x12
      Length of following address specification: 10
      Syntax Id: S7ANY (0x10)
      Transport size: BYTE (2)
      Length: 1
      DB number: 0
      Area: Inputs (I) (0x81)
      Address: 0x000018
      .... 0000 0000 0000 0001 1... = Byte Address: 3
      .... 0000 0000 0000 0000 = Bit Address: 0
    > Item [2]: (Q 2.0 BYTE 1)
    > Item [3]: (M 10.0 BYTE 1)

0000 28 63 36 9a 21 04 00 e0 4c 68 06 28 08 00 45 00 (c6!... Lh (.E
0010 00 5f 00 00 40 00 40 06 48 8d c0 a8 38 ac c0 a8  H... 8...
0020 38 0f cf 00 00 00 00 7f d8 b5 02 41 1a 5a 50 18  8...f...A-P
0030 ff ff a0 34 00 00 03 00 00 37 02 f0 30 32 01 00 4... 7...2...
0040 00 00 e5 00 26 00 00 04 03 12 0a 10 02 00 01 00  &...
0050 00 01 00 00 18 12 0a 10 02 00 01 00 00 02 00 00  .....
0060 10 12 0a 10 02 00 01 00 00 03 00 00 50 00 00 50  ...P

```

Figura 15: Petición de lectura S7Comm [19]

```

Header: (Job)
  Protocol Id: 0x32
  ROSCTR: Job (1)
  Redundancy Identification (Reserved): 0x0000
  Protocol Data Unit Reference: 2300
  Parameter length: 14
  Data length: 6
Parameter: (Write Var)
  Function: Write Var (0x05)
  Item count: 1
  Item [1]: (Q 2.6 BIT 1)
    Variable specification: 0x12
    Length of following address specification: 10
    Syntax Id: S7ANY (0x10)
    Transport size: BIT (1)
    Length: 1
    DB number: 0
    Area: Outputs (Q) (0x82)
    Address: 0x000016
Data
  Item [1]: (Reserved)
    Return code: Reserved (0x00)
    Transport size: BIT (0x03)
    Length: 1
    Data: 01

```

Figura 16: Petición de escritura S7Comm [19]

```

S7 Communication
  Header: (Ack_Data)
    Protocol Id: 0x32
    ROSTR: Ack_Data (3)
    Redundancy Identification (Reserved): 0x0000
    Protocol Data Unit Reference: 2278
    Parameter length: 2
    Data length: 17
    Error class: No error (0x00)
    Error code: 0x00
  Parameter: (Read Var)
    Function: Read Var (0x04)
    Item count: 3
  Data
    Item [1]: (Success)
      Return code: Success (0xff)
      Transport size: BYTE/WORD/DWORD (0x04)
      Length: 1
      Data: 00
      Fill byte: 0x00
    Item [2]: (Success)
    Item [3]: (Success)

```

0000	00 e0 4c 68 06 28 28 63 38 9a 21 04 08 00 45 00	..Lh-((c 6 1...E-
0010	00 4e ff 19 40 00 40 06 49 84 c0 a8 38 0f c0 a8	-N-@ I...8...
0020	38 ac 00 66 cf 9a 82 41 1a 8a a0 7f d8 ec 50 18	8-f-A .....P-
0030	20 00 0b 0f 00 00 03 00 00 26 02 f0 80 32 03 00	.....&...2...
0040	00 08 e6 00 02 00 11 00 00 04 03 ff 04 00 08 00	.....
0050	00 ff 04 00 08 00 00 ff 04 00 08 00	.....

Figura 17: Respuesta de lectura S7Comm [19]

S7Comm es un protocolo relativamente sencillo y en caso de hacer un ataque de *packet replay* (responder con paquetes a partir de los capturados por el atacante). Para ello, hay que saber cómo se construye el paquete y como son las direcciones de las variables.

En los PLCs del laboratorio que son la serie S7-1500, por defecto no usan este tipo de tráfico. La comunicación entre PLC, TIA Portal y HMI se realiza a través de S7comm plus, que es una mejora sobre el S7comm explicado anteriormente. La seguridad de este protocolo se muestra en el Anexo 2: S7Comm Plus.

Estos programas vistos también nos permiten escribir y leer datos de un *Data Block*, los cuales, en los sistemas reales, también se suelen almacenar muchos datos y con los cuales se pueden crear funciones. No entro en detalle de cómo se realiza, ya que es casi igual que los anteriores. Un ejemplo de estos códigos en Python para ello lo podemos encontrar en la siguiente referencia [20]

### 5.3 MODBUS

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

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

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

El funcionamiento del protocolo es idéntico independientemente de la versión empleada. Modbus se basa en registros, en los cuales los esclavos y maestros escriben y leen. En el caso de los esclavos, como ya he citado anteriormente, algunas de las acciones solo las podrá llevar a cabo cuando el maestro correspondiente se lo indique. Cada esclavo tiene sus registros y su indicador (ID) con el cual el maestro le mandará peticiones, junto al número o números de registros a los que quiere acceder. Los registros son de varios tipos, con distintas políticas de acceso y tamaño. A continuación, se muestran estos datos en la tabla 3 y el formato de la trama TCP en la tabla 4:

Tipo de objeto	Acceso	Tamaño
Discrete input	Solo leer	1 bit
Coil	Leer/escribir	1 bit
Input register	Solo leer	16 bits
Holding register	Leer/escribir	16 bits

Tabla 3: Tipos de registros Modbus [20]

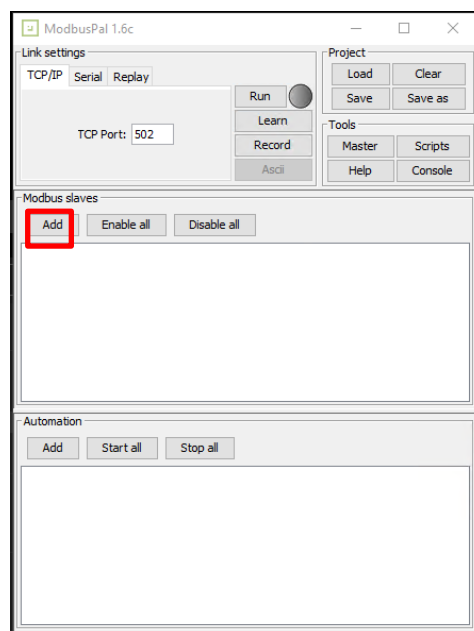
Nombre	Longitud (Bytes)	Función
Identificador de la transacción	2	Para la sincronización entre mensajes de servidor y cliente
Identificador del protocolo	2	0 para Modbus/TCP
Campo de longitud	2	Número de bytes en esta trama
Identificador de unidad	1	Dirección del esclavo (255 si no se usa)
Código de función	1	Códigos de función como en otras variantes
Bytes de datos	<i>n</i>	Datos como respuesta o comandos

Tabla 4: Campos del protocolo Modbus [20]

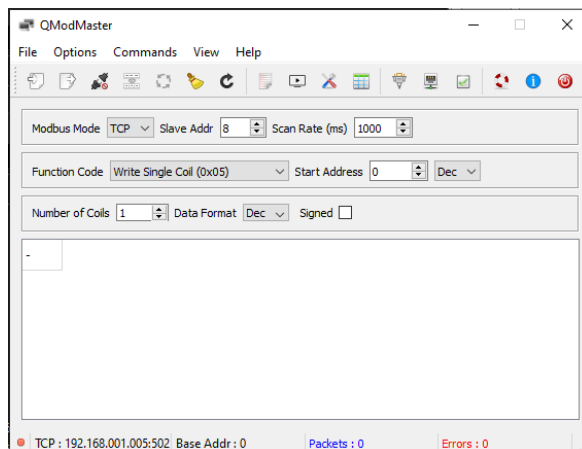


Antes de realizar un ataque sobre Modbus, hay que investigar qué medidas de seguridad implementa el protocolo. En este caso, Modbus es muy simple ya que no incorpora ningún tipo de cifrado (manda los datos en texto plano) y no incorpora grandes medidas de seguridad. En concreto Modbus lo único que aporta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

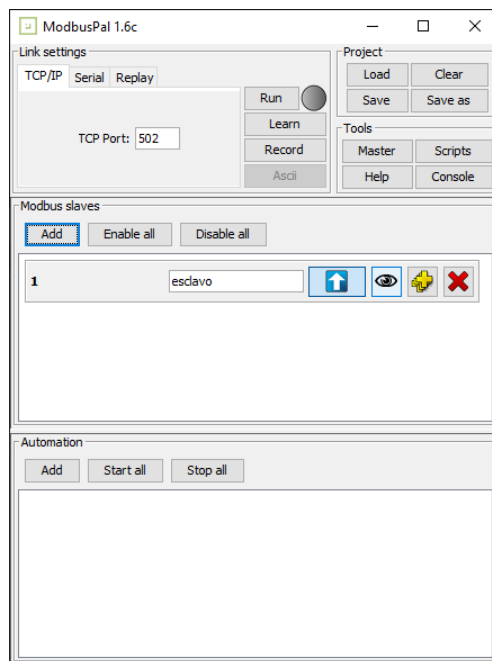
Para realizar los ataques vamos a emplear un simulador de Modbus. En concreto para el cliente voy a emplear ModbusPal que es un cliente basado en java, lo cual nos aporta la posibilidad de ejecutarlo en distintos sistemas operativos indistintamente. Para descargarlo en la referencia [21]. Con esta aplicación podemos crear esclavos y sus registros. También nos permite modificar manualmente los registros, lo cual puede venir bien para llevar a cabo pruebas. Este programa se muestra en la figura 1. Para simular el master, utilizaremos QModMaster [22] un programa que nos permite conectarnos al ModbusPal para leer y escribir datos. Este programa se muestra en la figura 2. A continuación se muestra el proceso de configuración.



*Figura 18: ModbusPal [21]*

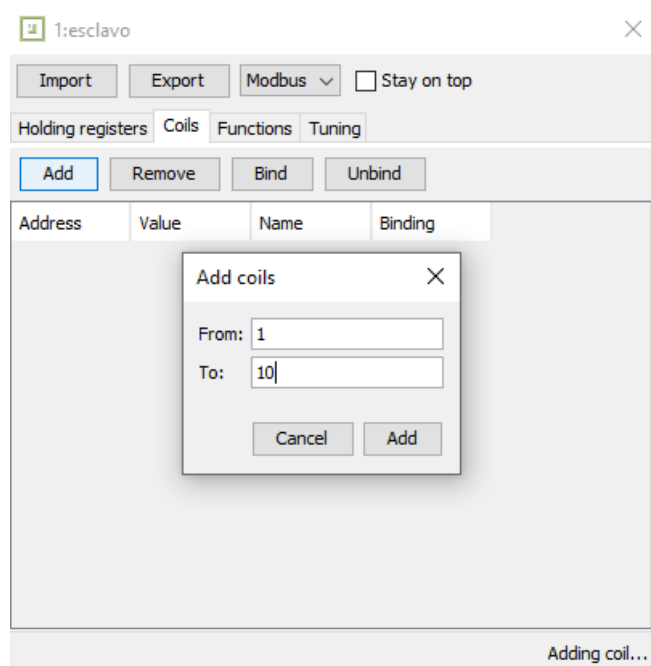


*Figura 20: QModMaster [22]*

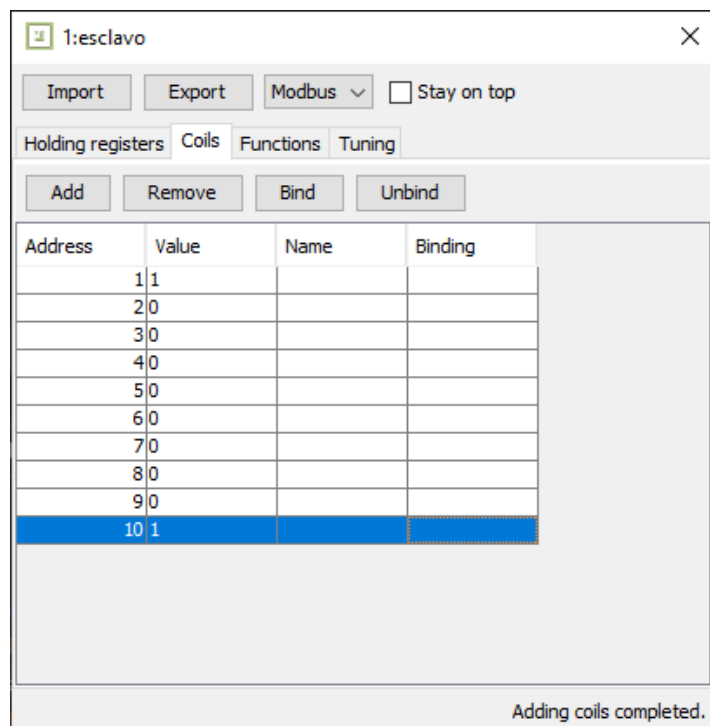


*Figura 19: Acceder a los registros*

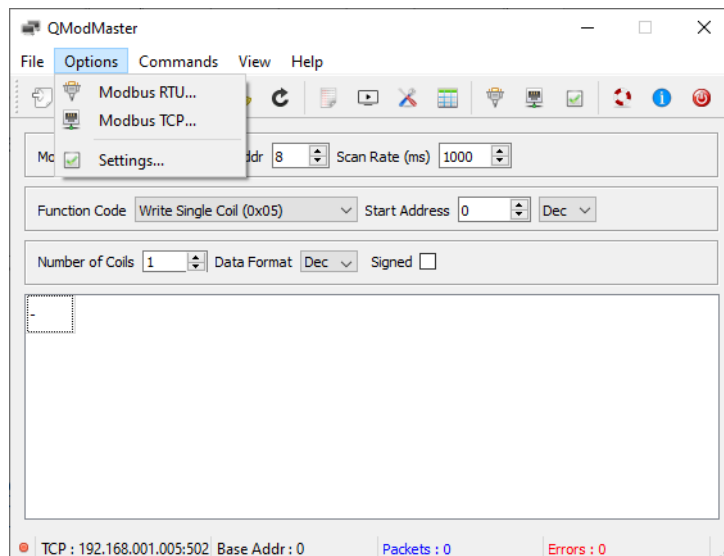
El proceso de conexión es muy simple. Primero crearemos un esclavo en el ModbusPal en el botón Add del apartado Modbus slaves. Le asignaremos un ID y un nombre. Una vez hecho esto, podemos crear un registro y poner valores. Para este ejemplo, en los registros 1 y 10, he puesto el valor 1 para que veamos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



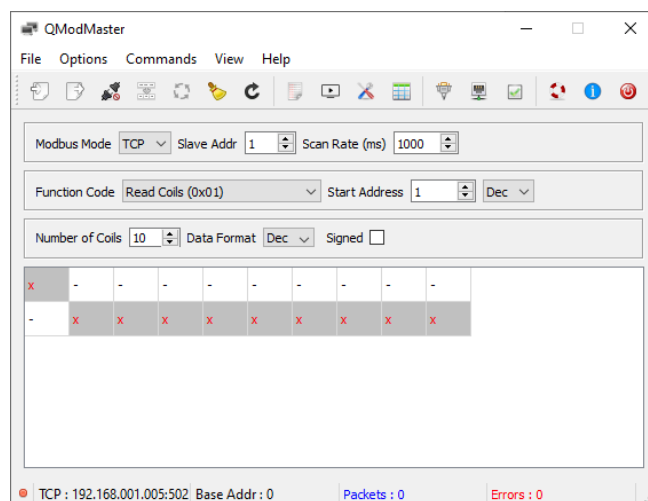
*Figura 22: Acceder a coils y crear tabla*



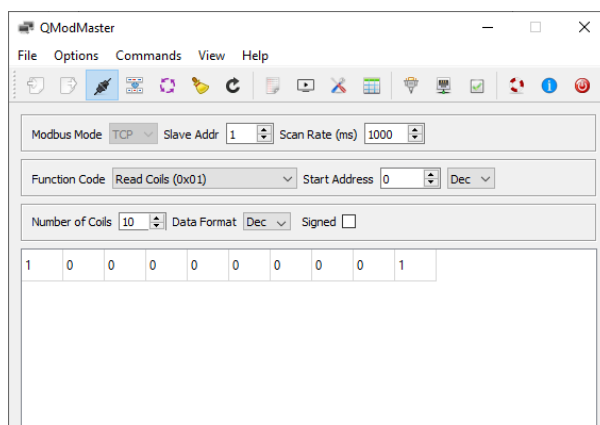
*Figura 21: Tabla de coils y valores*



*Figura 25: Conexión QModMaster*



*Figura 24: Configuración lectura de coils*



*Figura 23: Lectura de los coils*

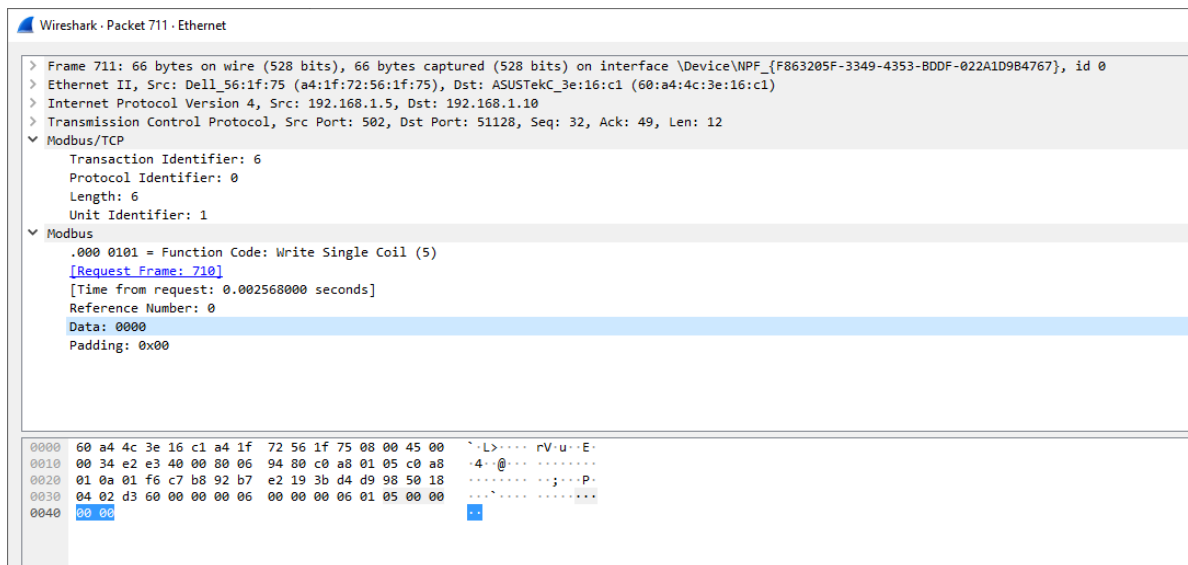
Una vez hecho este proceso, solo hace falta ir a la pantalla principal de ModbusPal y darle a

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

Al leer o escribir datos, se comunican los dos programas utilizando Modbus. Este tráfico lo podemos analizar usando Wireshark y veremos que es un protocolo bastante simple y confirmamos que no está cifrado. Un descubrimiento importante que se aprecia en este punto es que el checksum no se valida en ningún momento. En este punto también vemos los distintos mecanismos de seguridad que implementa. Al estar basado en TCP, se ve que cada petición se valida por un ACK, por lo que hay unas secuencias TCP que sigue. También se ve que sigue una secuencia de transacción. En la figura 29 se puede ver tráfico de lectura y de escritura y en la figura 30 se ve los datos que añade Modbus sobre TCP en una query de escritura de un valor 0 en el primer registro, los cuales se aprecia que se transmiten en texto plano.

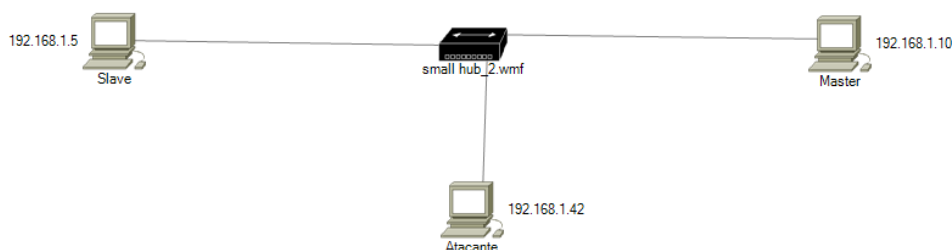
207	8.653265	192.168.1.10	192.168.1.5	Modbus...	66	Query: Trans:	3; Unit:	1, Func:	1: Read Coils
208	8.655607	192.168.1.5	192.168.1.10	Modbus...	65	Response: Trans:	3; Unit:	1, Func:	1: Read Coils
598	34.864890	192.168.1.10	192.168.1.5	Modbus...	66	Query: Trans:	4; Unit:	1, Func:	1: Read Coils
599	34.873827	192.168.1.5	192.168.1.10	Modbus...	64	Response: Trans:	4; Unit:	1, Func:	1: Read Coils
600	34.874919	192.168.1.10	192.168.1.5	Modbus...	66	Query: Trans:	5; Unit:	1, Func:	1: Read Coils
601	34.878917	192.168.1.5	192.168.1.10	Modbus...	64	Response: Trans:	5; Unit:	1, Func:	1: Read Coils
710	38.625362	192.168.1.10	192.168.1.5	Modbus...	66	Query: Trans:	6; Unit:	1, Func:	5: Write Single Coil
711	38.627930	192.168.1.5	192.168.1.10	Modbus...	66	Response: Trans:	6; Unit:	1, Func:	5: Write Single Coil

*Figura 29: Tráfico Modbus*



*Figura 30: Datos de Modbus en query de escritura*

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



*Figura 31: Topología de red*

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

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

*Tabla 5: Direcciones de las estaciones*

### 5.3.1 IMPERSONATING DEL MÁSTER

Los ataques de Impersonating (imitar en español) son los que, con un software específico, podemos hacernos pasar por el sistema que imita. En este caso, utilizaremos la herramienta de Metasploit que está incluido en Kali Linux para utilizar unos paquetes que nos proporciona listos para realizar este ataque. Para arrancar este framework basta con escribir el siguiente comando en el terminal de Linux: `sudo msfconsole`. Esperamos a que se abra la interfaz de terminal y hay que continuar buscando los módulos que vamos a emplear. Para buscarlos escribimos `search modbus` y se nos muestran los distintos paquetes preparados relacionados con modbus. Como nosotros partimos del hecho de que conocemos la topología

de red (quien es el esclavo y quien el máster) no empleamos ningún paquete de scanner. Esta fase es fácil de hacer con Wireshark una vez que el atacante se conecta a la red y arranca el programa. Solo hay que filtrar por protocolo modbus y vemos quien es el que escucha en el puerto 502 que es el esclavo y el que escribe a ese puerto es el máster. Por ello, vamos a seleccionar el paquete de modbusclient. Este paquete se puede seleccionar escribiendo *use 5* o con *use auxiliary/scanner/scada/modbusclient*. Este proceso se muestra en la figura 13. Una vez seleccionado, hay que ver que parámetros debemos establecer antes de ejecutar el paquete. Para ver estos parámetros, usamos el comando *show options*, como se muestra en la figura 14. Por último, hay que establecer todos los parámetros y ejecutarlo. Este último paso se muestra en la figura 15, que está establecido para leer el dato 2 de la coil del esclavo 1. Para establecer los valores se hace poniendo *set [opción] [valor]* y se ejecuta con *run*.

```
msf6 > search modbus

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_banner_grabbing		normal	No	Modbus Banner Grabbing
4	auxiliary/scanner/scada/modbus_findunitid	2012-10-28	normal	No	Modbus Unit ID and Station ID Enumerator
5	auxiliary/scanner/scada/modbusclient		normal	No	Modbus Client Utility
6	auxiliary/scanner/scada/modbusdetect	2011-11-01	normal	No	Modbus Version Scanner

```
Interact with a module by name or index. For example info 6, use 6 or use auxiliary/scanner/scada/modbusdetect
msf6 > use 5
msf6 auxiliary(scanner/scada/modbusclient) > 
```

Figura 32: Búsqueda modbus en msfconsole



```
msf6 auxiliary(scanner/scada/modbusclient) > show options
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   |                 | 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         | 1               | no       | Number of coils/registers to read (READ_COILS, READ_DISCRETE_INPUTS, READ_HOLDING_REGISTERS, READ_INPUT_REGISTERS modes only) |
| RHOSTS         |                 | yes      | The target host(s), range CIDR identifier, or hosts file with syntax 'file:paths'                                             |
| RPORT          | 502             | yes      | The target port (TCP)                                                                                                         |
| UNIT_NUMBER    | 1               | no       | Modbus unit number                                                                                                            |


Auxiliary action:

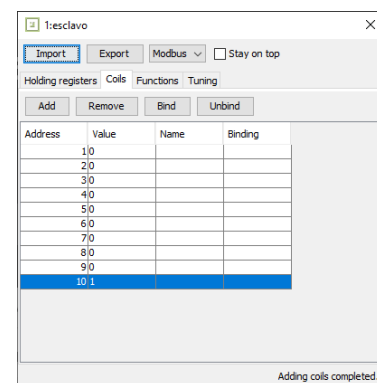

| Name                   | Description                               |
|------------------------|-------------------------------------------|
| READ_HOLDING_REGISTERS | Read words from several HOLDING registers |


```

Figura 33: show options de msfconsole

```
msf6 auxiliary(scanner/scada/modbusclient) > SET DATA_ADDRESS 1
[-] Unknown command: SET.
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_ADDRESS 1
DATA_ADDRESS => 1
msf6 auxiliary(scanner/scada/modbusclient) > set RHOSTS 192.168.1.5
RHOSTS => 192.168.1.5
msf6 auxiliary(scanner/scada/modbusclient) > set ACTION READ_COILS
ACTION => READ_COILS
msf6 auxiliary(scanner/scada/modbusclient) > run
[*] Running module against 192.168.1.5

[*] 192.168.1.5:502 - Sending READ COILS ...
[+] 192.168.1.5:502 - 1 coil values from address 1 :
[+] 192.168.1.5:502 - [0]
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) >
```



Address	Value	Name	Binding
10			
20			
30			
40			
50			
60			
70			
80			
90			
101			

Figura 35: Tabla de coils

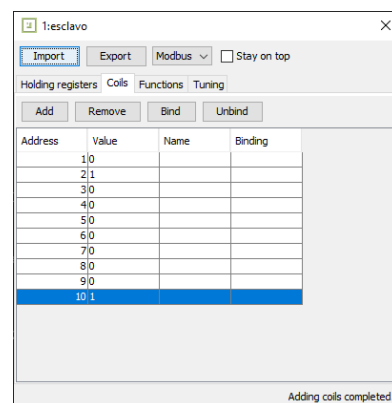
Figura 34: Establecimiento de parámetros y ejecución

Como se puede observar en la figura 34, el programa nos devuelve el valor del coil 2 del slave número 1. El valor que nos devuelve es correcto como se puede comprobar en la figura 35. Ahora vamos a emplear el mismo paquete, pero para escribir un valor. En la figura 36 se muestra la configuración y el resultado del run.

```
msf6 auxiliary(scanner/scada/modbusclient) > set DATA 1
DATA => 1
msf6 auxiliary(scanner/scada/modbusclient) > set ACTION WRITE_COIL
ACTION => WRITE_COIL
msf6 auxiliary(scanner/scada/modbusclient) > run
[*] Running module against 192.168.1.5

[*] 192.168.1.5:502 - Sending WRITE COIL ...
[+] 192.168.1.5:502 - Value 1 successfully written at coil address 1
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) >
```

Figura 36: Establecimiento de parámetros y escritura

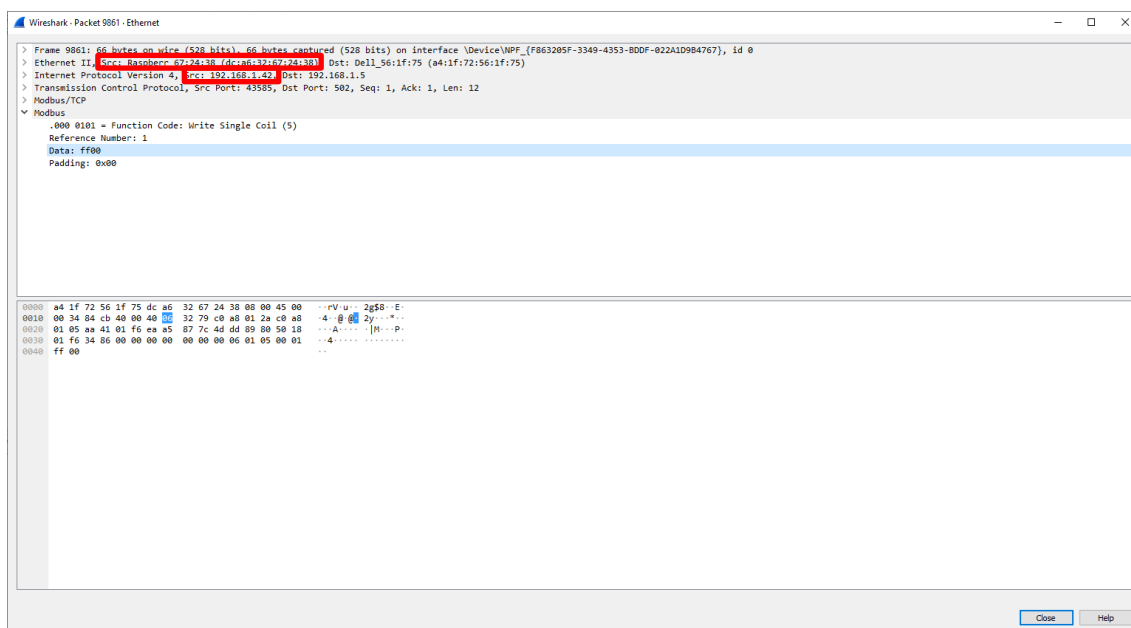


Address	Value	Name	Binding
10			
21			
30			
40			
50			
60			
70			
80			
90			
101			

Figura 37: Resultado de escritura

Como se puede ver en la figura 36 y 37, la escritura del valor 1 en el registro 2 se ha realizado con éxito. Con esto queda demostrado lo fácil que es imitar al máster. Si miramos el tráfico que hemos generado (me centro en este último de escritura), podemos confirmar con el wireshark que

todo es correcto, ya que la dirección de origen es la de la máquina del atacante, como se aprecia en la figura 38.



*Figura 38: Tráfico del atacante*

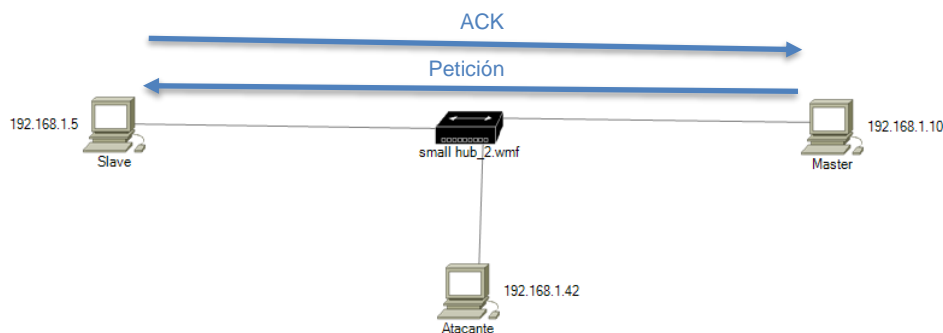
Este programa es muy fácil de usar, pero si queremos realizar ataques más complejos o automatizarlos, podemos también emplear una librería de Python y picar nuestro código. Esta librería se llama PyModbus y es muy fácil de usar. En la referencia [5] dejo la documentación sobre este paquete.

### 5.3.2 MAN IN THE MIDDLE

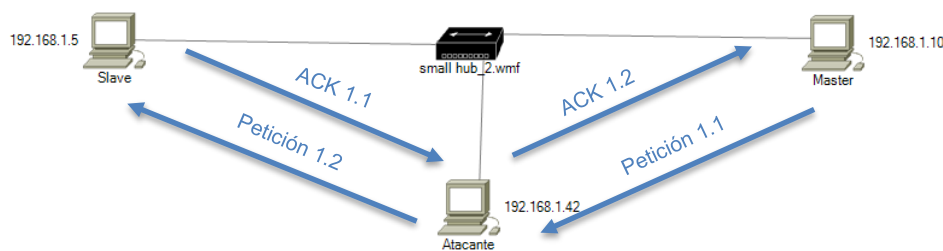
Este tipo de ataques son de los más conocidos, ya que se puede aplicar a un motón de ámbitos. Este ataque se basa en el concepto de que hay una comunicación entre dos partes (en nuestro caso, el máster y el esclavo) y una tercera persona (el atacante) se pone entre medias de los dos. El atacante es capaz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

aprovecharemos esta alteración para implementar un filtro que nos modifique automáticamente los datos que se envían.

El flujo normal de la información es la que genera el Máster, va directamente al esclavo y este le responde directamente al máster con un ACK confirmando la petición. Al implementar este ataque, el máster pensará que el slave es el ordenador del atacante, por lo que le envía al equipo del atacante la petición. El equipo del atacante puede o bien, no modificar nada y mandárselo al slave real (con esto podría analizar el tráfico) o aprovechar y modificar los datos que se envían. Este último es el que se implementará en este proyecto. Una vez modificado el paquete, se lo envía al esclavo. El ack que genera el esclavo, seguirá también este mismo camino. A continuación, en la figura 39 y 40, se muestra cómo es el tráfico normal y el tráfico con el MiTM implementado.



*Figura 39: Flujo de tráfico normal*



*Figura 40: Flujo de tráfico con MiTM*

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

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

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

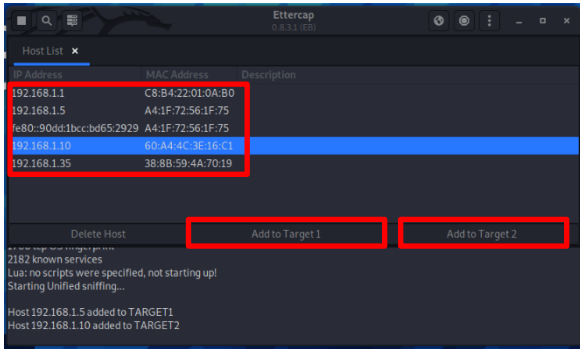


Figura 41: Añadir targets

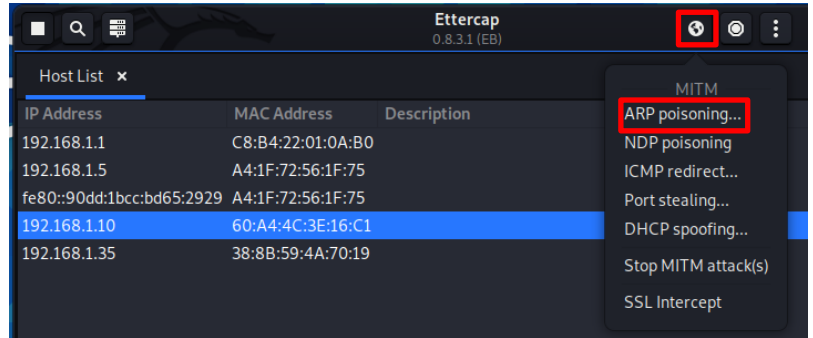


Figura 42: Activar ARP poisoning (1)

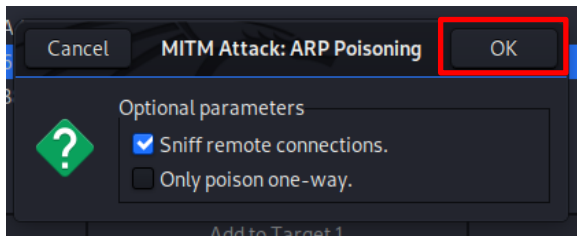


Figura 43: Activar ARP poisoning (2)

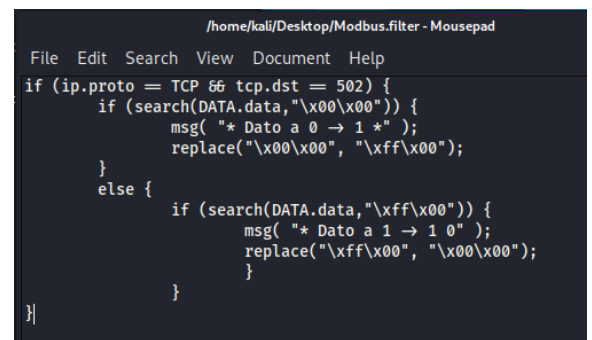


Figura 44: Filtro ettercap

Una vez creado y compilado el filtro, solo debemos añadirlo a ettercap y generar tráfico de escritura en QModMaster mientras analizamos el tráfico con Wireshark. El paso de analizar el tráfico es muy importante, ya que podremos ver si el proceso de MiTM se está realizando correctamente. Si todo funciona correctamente, deberíamos de ver en la parte inferior de

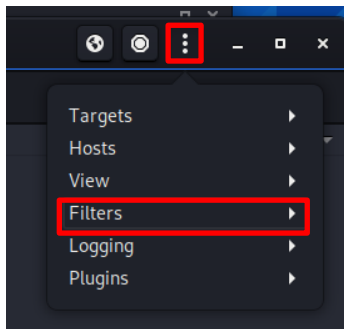


Figura 45: Cargar el filtro (1)

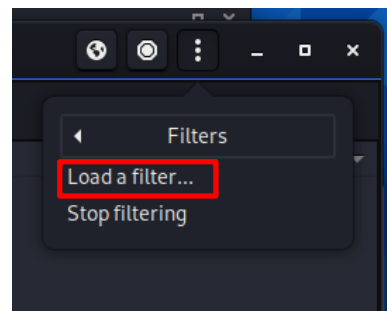


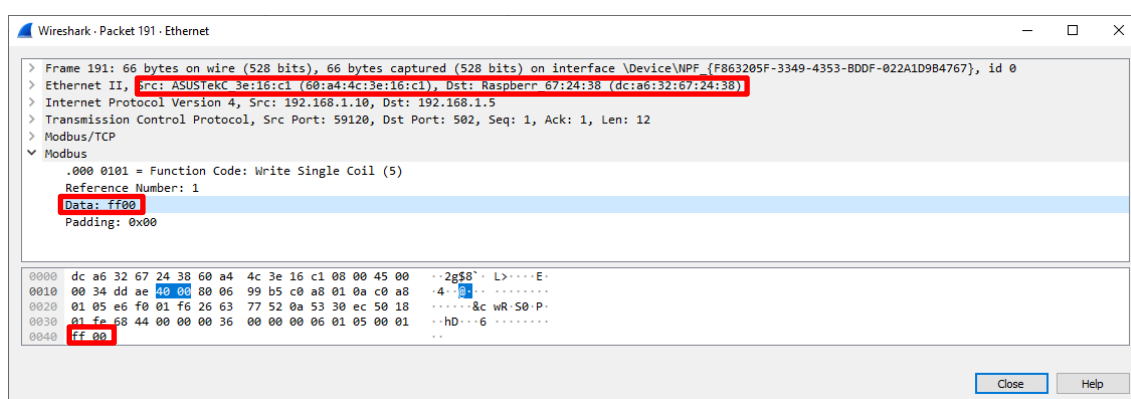
Figura 46: Cargar el filtro (2)

ettercap mensajes cada vez que escribimos algún valor desde el máster. En este ejemplo se va a emplear escritura del coil 2 del esclavo 1. A continuación, se muestran capturas de wireshark del proceso del MiTM. Se puede observar que la petición de escritura del valor 1 se manda del máster al equipo del atacante y del atacante se manda la misma petición al esclavo, pero con el valor modificado (pasa a ser 0). Es importante destacar que la

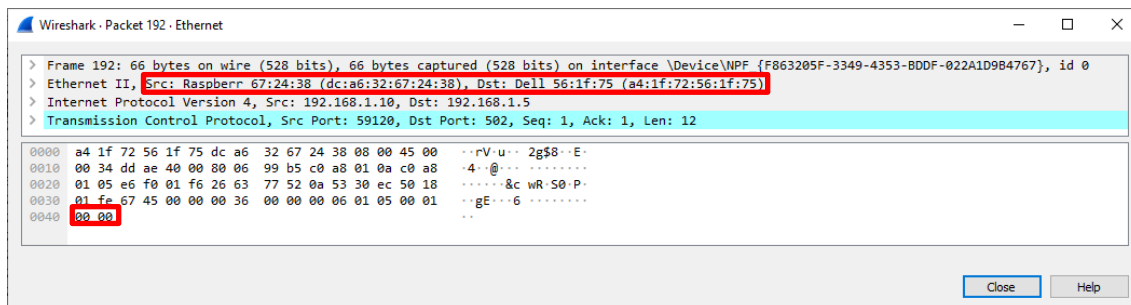
comunicación de escritura entre el atacante y el esclavo se realiza en TCP, no en modbus (pero el contenido del paquete es el mismo). En la figura 47 y 48 se aprecia como se han cambiado las direcciones para que el tráfico entre maestro y esclavo pase por el equipo del atacante. También se observa cómo se ha modificado el valor a escribir. Para el ACK que nos devuelve el esclavo, se sigue el mismo proceso.

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

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



*Figura 47: Petición de escritura interceptada*



*Figura 48: Petición de escritura interceptada y modificada*

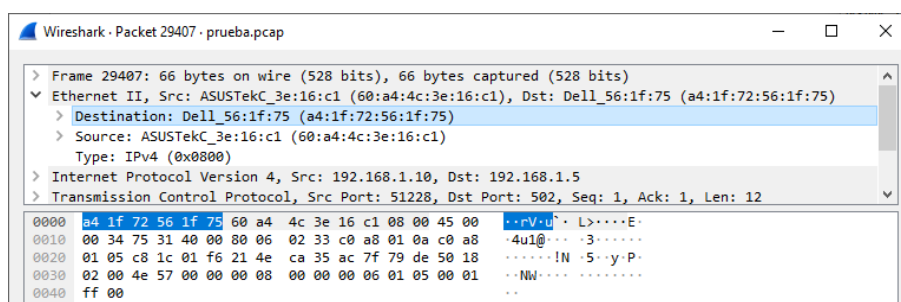
### 5.3.3 PACKET REPLAY

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

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

*a41f72561f7560a44c3e16c10800450000347531400080060233c0a8010ac0a80105c81c01f  
6214eca35ac7f79de501802004e57000000080000000601050001ff00*

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



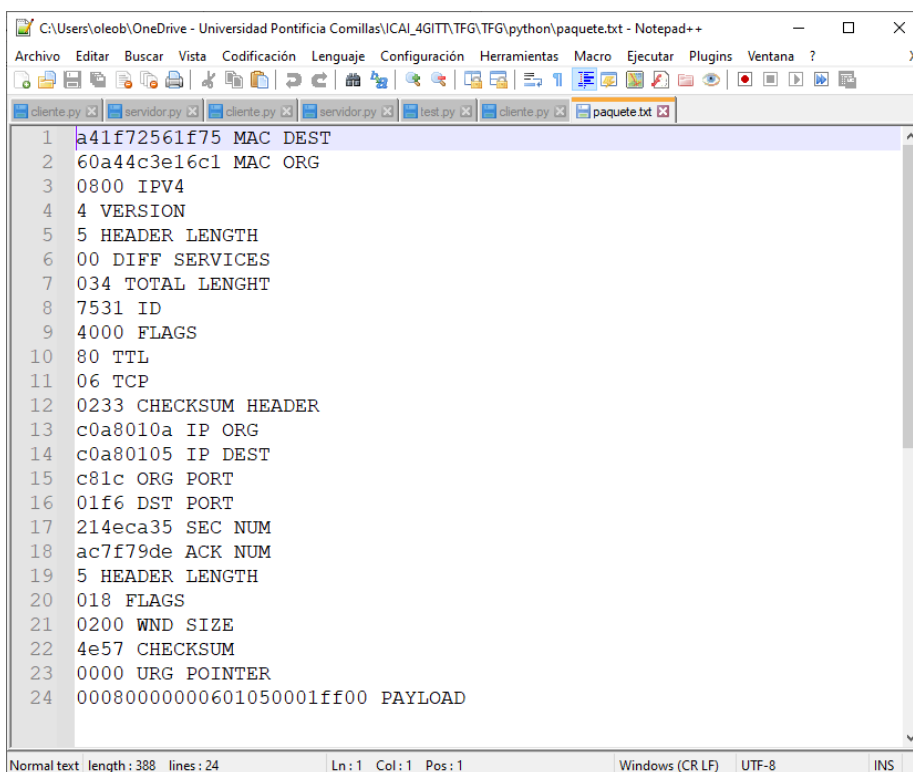
*Figura 49: Mac en hexadecimal*

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

*a41f72561f7560a44c3e16c10800450000347531400080060233c0a8010ac0a80105c81c01f6214eca35ac7f79de501802004e57000000080000000601050001ff00*

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





```
1 a41f72561f75 MAC DEST
2 60a44c3e16c1 MAC ORG
3 0800 IPV4
4 4 VERSION
5 5 HEADER LENGTH
6 00 DIFF SERVICES
7 034 TOTAL LENGHT
8 7531 ID
9 4000 FLAGS
10 80 TTL
11 06 TCP
12 0233 CHECKSUM HEADER
13 c0a8010a IP ORG
14 c0a80105 IP DEST
15 c81c ORG PORT
16 01f6 DST PORT
17 214eca35 SEC NUM
18 ac7f79de ACK NUM
19 5 HEADER LENGTH
20 018 FLAGS
21 0200 WND SIZE
22 4e57 CHECKSUM
23 0000 URG POINTER
24 000800000000601050001fff00 PAYLOAD
```

*Figura 50: Paquete completamente descompuesto*

Una vez ya hemos deshecho el paquete, es relativamente sencillo implementarlo en Scapy. Para editar el paquete, no vamos a hacerlo a través de la consola de Scapy, ya que requiere que pongamos muchos comandos. Para hacerlo de forma más clara y sencilla, se hará el mismo proceso que en la consola, pero en un script de Python implementando las librerías correspondientes. Para este proceso me baso en el documento de la referencia [23]. En este documento se trata sobre los elementos de los que se compone este protocolo y la base sobre la cual me baso en este proyecto para el script de Python. El script que se nos presenta en dicho documento no tiene todos los campos necesarios para este tipo de query que estamos intentando replicar (escritura de coil). Para adaptar este código a nuestro caso, podemos consultar la documentación de Modbus en Scapy [24]. En esta documentación se nos indica los tipos de datos y los campos que son siempre necesarios más aquellos que debemos de incluir para determinadas queries. Aplicando todo esto, nos queda un código que se muestra en github por su extensión [25]

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

```
0000 A4 1F 72 56 1F 75 60 A4 4C 3E 16 C1 08 00 45 00 ..rV.u`.L>...E.
0010 00 34 75 31 40 00 80 06 02 33 C0 A8 01 0A C0 A8 .4u1@...3.....
0020 01 05 C8 1C 01 F6 21 4E CA 35 AC 7F 79 DE 50 18 .....!N.5..y.P.
0030 02 00 4E 57 00 00 00 08 00 00 00 06 01 05 00 01 ..NW.....
0040 FF 00 ..

Sent 1 packets.
PS C:\Users\oleob\OneDrive - Universidad Pontificia Comillas\ICAI_4GITT\TFG\TFG\python>
```

*Figura 51: Paquete reconstruido*

0000	a4 1f 72 56 1f 75 60 a4 4c 3e 16 c1 08 00 45 00	..rV.u`.L>...E.
0010	00 34 75 31 40 00 80 06 02 33 c0 a8 01 0a c0 a8	.4u1@...3.....
0020	01 05 c8 1c 01 f6 21 4e ca 35 ac 7f 79 de 50 18	.....!N.5..y.P.
0030	02 00 4e 57 00 00 00 08 00 00 00 06 01 05 00 01	..NW.....
0040	ff 00	..

*Figura 52: Paquete original*

Esta es la forma típica de realizar un packet replay, pero dependiendo del programa de destino, nos podemos encontrar con algunos contratiempos. Por ejemplo, con el ModbusPal, no podemos enviar un paquete al destino y que este lo lea, sin previamente haber establecido una conexión con el socket de destino. Al igual que en el GitHub está presente el código para generar el paquete de la figura 51, también está un código similar estableciendo un socket (PacketReplay.py). Este código se queda con la parte de la formación de los datos Modbus del paquete y una vez establecido el socket TCP, los manda. También podemos recoger la respuesta, la cual se muestra por consola como stream en hexadecimal. Utilizando este método de envío de datos basado en un socket TCP conseguimos modificar datos de los coils del esclavo. La clave de este método se basa en sacar la información sobre la ip y puerto que el esclavo utiliza usando wireshark y en saber qué tipo de mensaje Modbus queremos crear. Desde la documentación de Modbus de Scapy podemos ver que campos y tipos de

datos son necesarios para distintos tipos de queries, ya que este método no solo sirve para escribir.

## **5.4 PROFINET**

Profinet y Profibus son dos protocolos de comunicación empleados en entornos industriales. Como otros protocolos tiene su versión basada en un bus de datos RS-485 (Profibus) y su estándar basado en Ethernet (Profinet). Para más detalles sobre las diferencias entre estos protocolos en la referencia [27]. Es importante saber que la tendencia en la industria se inclina por Profinet. La gran mayoría de equipos aceptan Profinet, mientras que Profibus pierde fuerza. En las redes existentes que emplean equipos Profibus y Profinet, se implementa un Proxi para traducir los mensajes entre los dos protocolos. Debido a esto, este trabajo se centrará en Profinet [28].

En la actualidad, Profinet es ampliamente usado en comunicaciones entre PLCs y equipos de visión artificial (cámaras) y lectores de RFID. Esto se debe al gran aumento de estos dos últimos dispositivos en la industria, ya que facilitan automatizar procesos y validar calidades y clasificar productos automáticamente. Este protocolo ha sido adaptado por muchos fabricantes, como puede ser Siemens y Phoenix Contact. Al ser un protocolo relativamente moderno, tiene las características de estos: es flexible, implementa seguridad, escalable, alta disponibilidad y desarrollo continuado.

En cuanto a los detalles técnicos de las topologías Profinet, como ya he citado, está basado en Ethernet y funciona sobre cobre, fibra óptica, PoE y en sistemas inalámbricos. Como en muchos otros protocolos industriales, en Profinet encontramos tres roles en la topología (tipos de equipos): controladores, dispositivos y supervisores. Los controladores son los equipos que contienen un programa que se ejecuta en ellos y se intercambian datos con los dispositivos (PLC). Los dispositivos son cualquier equipo de captación de datos o ejecución de instrucciones (sensores/actuadores). Un ejemplo actual de estos dispositivos son por ejemplo una cámara o un lector RFID, como ya he citado antes. Un equipo supervisor es aquel desde el cual podemos lanzar instrucciones, monitorear y llevar a cabo análisis de diagnósticos. Normalmente estos equipos suelen ser PCs o HMIs.

A la hora de emplear Wireshark para poder analizar este tráfico, nos toparemos con que aparecen varios “protocolos” asociados a Profinet dependiendo del objetivo de esa transmisión. En la siguiente figura se muestran los nombres de los protocolos que aparecen y el tráfico al que están asociados:

**Protocols**

- **PROFINET/CBA**: distributed automation.
- **PROFINET/DCP**: discovery and basic configuration.
- **PROFINET/IO**: decentralized periphery.
- **PROFINET/MRP**: media redundancy protocol.
- **PROFINET/MRRT**: media redundancy for **PROFINET/RT**.
- **PROFINET/PTCP**: precision time control protocol.
- **PROFINET/RT**: real time data transfer.
- **PROFINET/IRT**: isochronous real time data transfer

*Figura 53: Profinet en Wireshark [29]*

Ahora vamos a ir viendo los principales con algo más de detalle. Para empezar, voy a desarrollar el DCP. Este está basado en la capa de enlace y se dedica a configurar las IPs y nombres de los equipos. Este es necesario en redes en las que no haya presentes un servidor DHCP, ya que hace la misma función que este. Profinet IO es un protocolo relacionado con los periféricos descentralizados y que utiliza Profinet RT para mandar los datos. Este último es usado para mandar datos cíclicos en tiempo real a un PLC. Esta comunicación hace de bypass de la interfaz TCP/IP que proporciona Profinet y nos permite mayor velocidad de transmisión (12MBd). Como ya he citado antes, Profinet IO se basa en RT para mandar los datos y esto se debe a que RT se diseñó para poder mandar datos de input/output con tiempos de envío críticos. El resto de las etiquetas Profinet que aparecen en Wireshark, serán explicadas cuando aparezcan en el documento.

Para empezar, es importante conocer los equipos profinet que hay presentes en la red para empezar a testear ataques. Para descubrir estos equipos, empleamos del Profinet DCP para descubrirlos (es como el arp en profinet). Para ello, vamos a craftear este paquete y enviarlo y observamos en wireshark la respuesta de los equipos. Para craftear este paquete de Discovery podemos basarnos en dos métodos: mirar en internet cómo se compone y el payload que manda o mandar ese paquete desde otra aplicación dedicada a ello y luego realizar una especie de “paquet replay” cambiando los parámetros necesarios. Para implementarlo, me baso en el segundo método. Para crear este paquete de Discovery, empleo

**PROFINET Commander - unlicensed version**

File Edit Tools Help

Import Config
 Browse
 Read Diagnostic
 R/W Data Record(s)
 Configure Controller

NIC Selection: 1-Network adapter 'Microsoft' on local host

Select Mode: Run Stop / Offline

Device Diagnostic Device Offline Run Stop

Navigation

**Devices (double click output cell for changing outputs)**

Device Name	Input	Status	Output	Status

**Properties**

Property Name	Property Value

**Information and Alarms (double click below to clear)**

Mon Jun 21 19:16:03 2021 NICs have changed since last session. Ensure that the correct interface is selected  
Mon Jun 21 19:16:03 2021 NIC - 5 Network adapter 'Realtek PCIe GbE Family Controller' on local host  
Mon Jun 21 19:16:03 2021 NIC - 4 Network adapter 'Microsoft' on local host  
Mon Jun 21 19:16:03 2021 NIC - 3 Network adapter 'Microsoft' on local host  
Mon Jun 21 19:16:03 2021 NIC - 2 Network adapter 'Microsoft' on local host  
Mon Jun 21 19:16:03 2021 NIC - 1 Network adapter 'Microsoft' on local host  
Mon Jun 21 19:16:03 2021 Please select NIC to use from dropdown list  
Mon Jun 21 19:16:03 2021 There were 5 NICs Found

En la pestaña de buscar, podemos encontrar el botón que nos permite buscar los dispositivos Profinet de la red y nos muestra una lista. También nos permite restablecer sus IPs, nombres, etc. También podemos aprovechar para ver el aspecto de estas respuestas en Wireshark. Lo que más interesa es ver la MAC a la que se dirige el paquete de DCP, ya que para craftear el nuestro lo debemos conocer. También debemos conocer el payload que se manda. Toda esta información la podemos obtener en Wireshark. El paquete creado tiene el siguiente hex stream:

Si aplicamos el mismo método para dividir este stream en sus componentes, como vimos en Modbus, obtenemos fácilmente cada uno de los componentes. La parte roja se corresponde al MAC de destino, la cual es muy importante porque es la “MAC de broadcast” que emplea Profinet y que nunca varía. La otra parte que nunca varía es la azul, que se corresponde al

payload de Profinet junto a alguna etiqueta de Vlans. Por último, la parte verde es la que deberíamos cambiar, ya que es la MAC de origen. En el stream anterior se ha mandado el paquete desde el puesto con el PLC\_15. Para que este proceso sea más real, creamos y enviamos el paquete desde otro equipo, simulando el del atacante. En mi caso, el ordenador desde el que mando el paquete tiene como MAC la 00:e0:4c:68:06:28. Juntando esta información con la anterior podemos crear el paquete y enviarlo usando Scapy. Para la recepción de las respuestas, he escogido no recogerlas en Python y verlas directamente en Wireshark, ya que aportan mayor nivel de información. Este código está subido al GitHub en la referencia [31] y también se muestra a continuación.

1.	1281.818735	Cognex_1c:27:c2	RealtekS_68:0e:01	PN-DCP	120	Ident	Ok	Xid:0x1	Dev-Options(9)	DeviceVendorValue	NameOfStation:"camaraxbs1d40b"	Dev-ID	Dev-Role	IP
1.	1281.818736	Cognex_1c:0c:58	RealtekS_68:0e:01	PN-DCP	120	Ident	Ok	Xid:0x1	Dev-Options(9)	DeviceVendorValue	NameOfStation:"camaraxbs2d54b"	Dev-ID	Dev-Role	IP
1.	1281.818737	Siemens_9a:21:0c	RealtekS_68:0e:01	PN-DCP	132	Ident	Ok	Xid:0x1	Dev-Options(1)	DeviceVendorValue	NameOfStation:"plcxbis5.interfazaprofinetxib3d53"	Dev-ID	Dev-Role	Dev-Instance
1.	1281.819025	IntelCor_bc:c0:58	RealtekS_68:0e:01	PN-DCP	124	Ident	Ok	Xid:0x1	Dev-Options(11)	DeviceVendorValue	NameOfStation:"aa2d5ldip05"	Dev-ID	Dev-Role	IP
1.	1281.819537	Siemens_07:15:58	RealtekS_68:0e:01	PN-DCP	114	Ident	Ok	Xid:0x1	Dev-Options(7)	DeviceVendorValue	NameOfStation:"ioxbdd1719a"	Dev-ID	Dev-Role	IP
1.	1281.819578	Siemens_27:65:58	RealtekS_68:0e:01	PN-DCP	118	Ident	Ok	Xid:0x1	Dev-Options(8)	DeviceVendorValue	NameOfStation:"plcxbis1xb52aafb"	Dev-ID	Dev-Role	IP
1.	1281.819631	Siemens_07:10:58	RealtekS_68:0e:01	PN-DCP	112	Ident	Ok	Xid:0x1	Dev-Options(7)	DeviceVendorValue	NameOfStation:"ioxb52abcb"	Dev-ID	Dev-Role	IP
1.	1281.819783	Siemens_07:10:58	RealtekS_68:0e:01	PN-DCP	112	Ident	Ok	Xid:0x1	Dev-Options(7)	DeviceVendorValue	NameOfStation:"ioxb51aa80"	Dev-ID	Dev-Role	IP
1.	1281.819783	Siemens_03:b1:58	RealtekS_68:0e:01	PN-DCP	126	Ident	Ok	Xid:0x1	Dev-Options(2)	DeviceVendorValue	NameOfStation:"hubxbis1xb52xb0b60"	Dev-ID	Dev-Role	Dev-Instance
1.	1281.820351	Siemens_a8:6b:1a	RealtekS_68:0e:01	PN-DCP	146	Ident	Ok	Xid:0x1	Dev-Options(14)	DeviceVendorValue	NameOfStation:"hubxbis1xb52xb27fe1"	Dev-ID	Dev-Role	IP
1.	1281.821212	Siemens_d0:19:58	RealtekS_68:0e:01	PN-DCP	122	Ident	Ok	Xid:0x1	Dev-Options(1)	DeviceVendorValue	NameOfStation:"hubxbis1xb52xb17ea1"	Dev-ID	Dev-Role	Dev-Instance
1.	1281.822521	Siemens_53:f4:4a	RealtekS_68:0e:01	PN-DCP	158	Ident	Ok	Xid:0x1	Dev-Options(20)	DeviceVendorValue	NameOfStation:"variadorxb52d41c"	Dev-ID	Dev-Role	IP
1.	1281.824372	Siemens_04:84:4a	RealtekS_68:0e:01	PN-DCP	122	Ident	Ok	Xid:0x1	Dev-Options(9)	DeviceVendorValue	NameOfStation:"ffidxb5291e5"	Dev-ID	Dev-Role	IP
1.	1281.824677	Siemens_04:83:4a	RealtekS_68:0e:01	PN-DCP	122	Ident	Ok	Xid:0x1	Dev-Options(9)	DeviceVendorValue	NameOfStation:"ffidxb5190a5"	Dev-ID	Dev-Role	IP

Figura 55: Resultado del envío del paquete DCP [31]

```
from scapy.all import *
import binascii

#010ecf00000 mac dest
#6805cabcc38e mac origen (CAMBIAR POR LA DEL EQUIPO) --> ether 00:e0:4c:68:06:28
#810000008892fe05000000000100010004ffff000000000000000000000000000000000000000000000 Payload + resto

raw_pkt = binascii.unhexlify('010ecf00000000e04c680628810000008892fe05000000000100010004ffff000000000000000000000000000000000000000000000000000')
pn_dcp = Ether(raw_pkt)
pn_dcp.show()
sendp(pn_dcp,iface="en8")
```

Observando el tráfico podemos sacar la información respecto a los dispositivos Profinet que se encuentran en este segmento de la red. En concreto nos encontramos varios hubs, PLCs, cámaras, lectores RFID, etc. A nosotros, los equipos que nos interesan son los PLCs, ya que nos permiten realizar más pruebas y ataques.

Si seleccionamos cualquiera de estos paquetes, conseguimos información muy útil para realizar ataques. Los principales (aparte del nombre) son la IP y la MAC del dispositivo, las cuales necesitaremos para llevar a cabo los ataques. Del paquete correspondiente al PLC\_15,





imagen de ese instante en el software correspondiente. Debido a esto, cuando se acciona este disparador, se genera tráfico TCP para transportar dicha imagen.

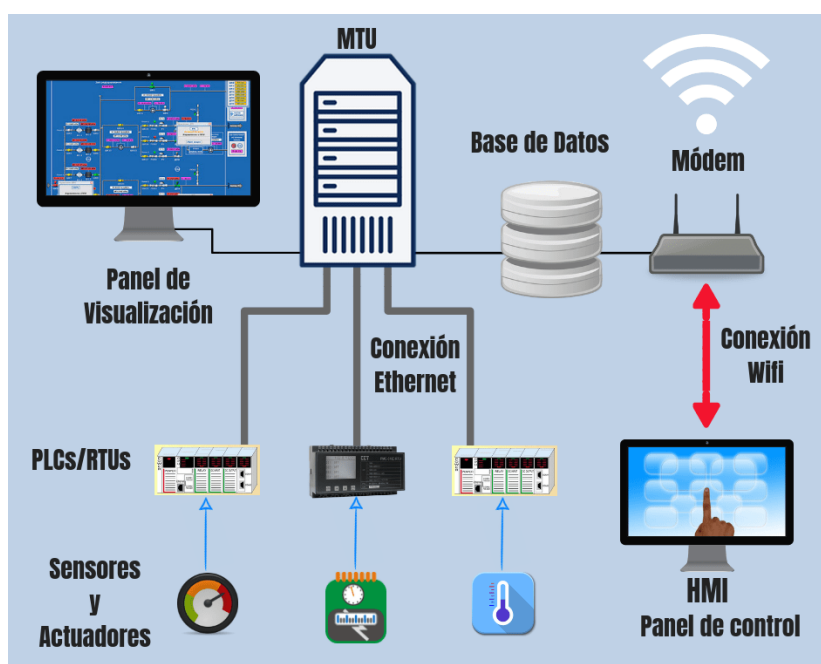
297	0.033265	Cognex_1c:27:c2	Siemens_9a:21:04	PNIO	106	RTCI, ID:0x8000, Len: 86, Cycle:39936 (Valid,Primary,Ok,Run)
298	0.033266	Siemens_9a:21:04	Cognex_1c:27:c2	PNIO	100	RTCI, ID:0x8010, Len: 80, Cycle:26048 (Valid,Primary,Ok,Run)
299	0.035234	Cognex_1c:27:c2	Siemens_9a:21:04	PNIO	106	RTCI, ID:0x8000, Len: 86, Cycle:40000 (Valid,Primary,Ok,Run)
300	0.035236	Siemens_9a:21:04	Cognex_1c:27:c2	PNIO	100	RTCI, ID:0x8010, Len: 80, Cycle:26112 (Valid,Primary,Ok,Run)
301	0.037310	Cognex_1c:27:c2	Siemens_9a:21:04	PNIO	106	RTCI, ID:0x8000, Len: 86, Cycle:40064 (Valid,Primary,Ok,Run)
302	0.037313	Siemens_9a:21:04	Cognex_1c:27:c2	PNIO	100	RTCI, ID:0x8010, Len: 80, Cycle:26176 (Valid,Primary,Ok,Run)
303	0.039249	Cognex_1c:27:c2	Siemens_9a:21:04	PNIO	106	RTCI, ID:0x8000, Len: 86, Cycle:40128 (Valid,Primary,Ok,Run)
304	0.039251	Siemens_9a:21:04	Cognex_1c:27:c2	PNIO	100	RTCI, ID:0x8010, Len: 80, Cycle:26240 (Valid,Primary,Ok,Run)

*Figura 56: Tráfico profinet entre Cognex Y PLC [33]*

Una vez llegados a este punto, urge la necesidad de encontrar un dispositivo o un software que nos permita seguir indagando en Profinet y realizar ataques sobre el PLC. En concreto, el objetivo es encontrar una herramienta o equipo que nos ofrezca la capacidad de escribir valores no síncronos, ya que para realizar ataques es lo óptimo. Debido a esto, en el apartado de conclusiones se listará una serie de opciones que se pueden tomar para poder realizar prácticas de un nivel algo más avanzado sobre este protocolo. Estas conclusiones serán del estilo de una consultoría, ya que nos parece interesante que los alumnos entren en detalle de este protocolo, ya que es de los más usados y estandarizados para comunicaciones industriales entre distintos equipos de distintas compañías.

## 5.5 SISTEMAS SCADA

Los sistemas SCADA son herramientas de automatización y de control industrial, cuyas principales funciones son: controlar, supervisar, recopilar datos, analizar datos y generar informes. Estos sistemas están compuestos por varios componentes de los cuales, los más importantes para este proyecto serán los PLCs y los sensores (suponiendo que es un sistema real). En la siguiente imagen se ve con detalle estos componentes:



*Figura 57: Esquema de un sistema SCADA [34]*

Para entenderlo de forma sencilla y aplicado a nuestro ejemplo del laboratorio de ICAI, es un sistema que nos proporciona la información de la fábrica y control sobre ciertos parámetros. Estos sistemas son importantes debido a la automatización que nos permite tener en nuestro entorno industrial. También están presentes en muchos ámbitos, como el sector energético, transporte, alimentación, etc.

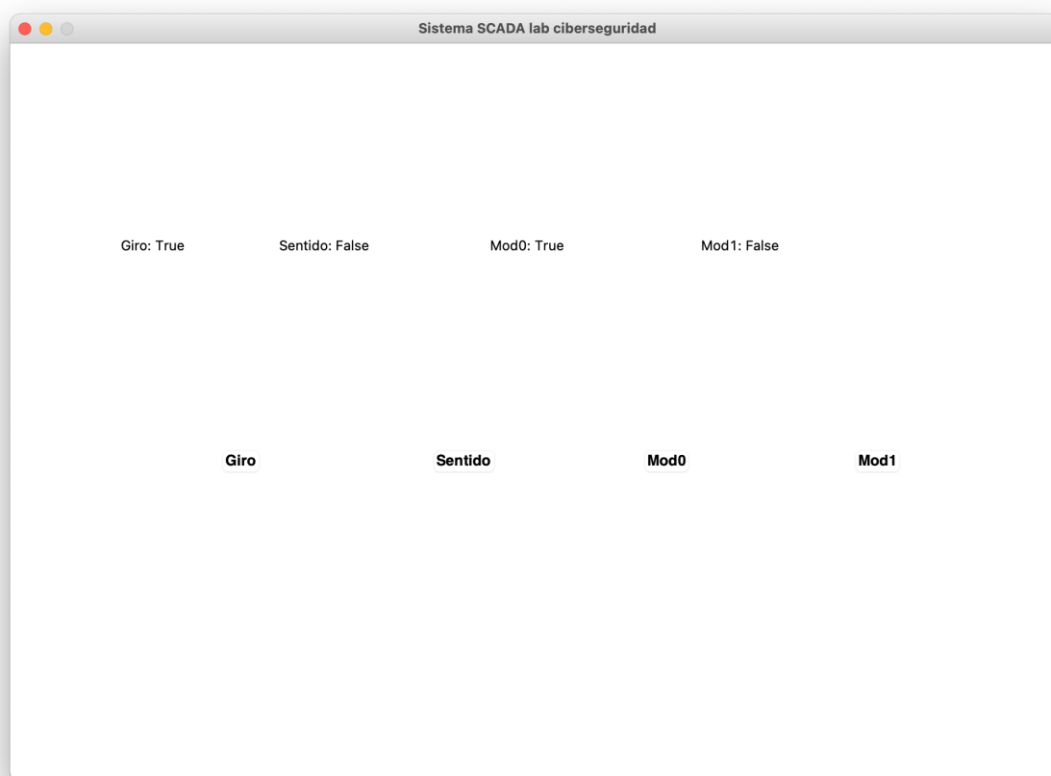
Para crear un sistema SCADA tenemos disponibles varios softwares. Como trabajamos con autómatas Siemens, el más integrado con estos equipos es WinCC [35]. También podemos reutilizar el código de Python del apartado de S7comm añadiendo una interfaz gráfica. El ataque por excelencia en este tipo de sistemas es modificar los valores mientras que se

muestren los originales en la pantalla, de este modo, las personas que tienen acceso a estos datos no son capaces de ver estos errores.

Para desarrollar esta parte del proyecto, voy a crear un sistema SCADA muy simple con Python. Esta opción es más interesante ya que puedo, dentro del mismo sistema SCADA, utilizar varios protocolos a la vez. Esto ocurre mucho en los sistemas reales, ya que no todos los sensores se comunican por el mismo protocolo. Este programa utiliza S7Comm para controlar la maqueta de la cinta transportadora (giro y sentido) a través de la librería Snap7. Como segundo protocolo he escogido Modbus, el cual usamos para visualizar y modificar dos coils. Para usar este protocolo, empleo la librería citada en el apartado de ModBus PyModbus [36].

La aplicación tiene una interfaz gráfica super sencilla, solo se compone de unos textos donde se muestran los estados de las variables y unos botones para modificarlos. Para que sea realista, los datos que se muestran son “en tiempo real”. Se ha creado un *Thread* aparte para la lectura. Cada medio segundo se leen las variables del PLC y los Coils de Modbus. La aplicación no está optimizada, por lo que el tiempo de respuesta puede variar, pero en cualquier caso sirve para hacer las pruebas pertinentes.

Para las lecturas y escrituras, se emplea el PLCSim ya visto anteriormente y para ModBus empleamos el ModBusPal. En este último es importante que, al crear el esclavo, tenga la ip de la máquina. Es decir, el equipo donde se va a ejecutar el ModBusPal es la que tiene IP 192.168.1.10, por lo que tengo que crear en ese equipo un esclavo que tenga como nombre esa IP. Esto se debe a que el PyModbus busca el esclavo con nombre igual a la IP del equipo que se indicar. Para que esto funcione correctamente, debemos tener arrancados y funcionando el PLCSim y el ModbusPal.



*Figura 58: Sistema SCADA creado en Python [37]*

En cuanto a los ataques que se pueden realizar sobre estos sistemas, se recurren a los ya vistos. Estos sistemas no tienen sus propios protocolos para comunicarse, si no que utilizan varios protocolos industriales para ello. Como es en este ejemplo, el SCADA de ejemplo se basa en S7Comm y Modbus. No voy a entrar en detalle de cómo realizar ataques sobre estos dos protocolos ya que han sido explicados anteriormente. Todos los ataques citados se pueden emplear para este SCADA y modificar las variables. Lo único interesante que se puede añadir, es el factor de que, al ser un sistema en tiempo real, es importante cuando se realiza un ataque, no ser detectado. Para ello, cuando escribamos un valor en una variable, es interesante mandar mensajes del estado previo de la variable de vuelta al SCADA para que el ingeniero que lo monitoriza no sospeche de que el sistema ha sido comprometido. En la siguiente imagen se puede apreciar el tráfico que genera este sistema por defecto (lectura y escritura de valores)

1	0.000000	192.168.1.10	192.168.1.38	Modbus/TCP	64	Response: Trans: 29; Unit: 0, Func: 1: Read Coils
2	0.000183	192.168.1.38	192.168.1.10	TCP	54	55273 → 502 [ACK] Seq=1 Ack=11 Win=4095 Len=0
6	0.503697	192.168.1.38	192.168.1.10	S7COMM	85	R0SCTR: [Job ] Function: [Read Var]
7	0.525865	192.168.1.10	192.168.1.38	S7COMM	86	R0SCTR: [Ack_Data] Function: [Read Var]
8	0.526056	192.168.1.38	192.168.1.10	TCP	54	55271 → 102 [ACK] Seq=32 Ack=33 Win=4095 Len=0
9	0.527733	192.168.1.38	192.168.1.10	Modbus/TCP	66	Query: Trans: 30; Unit: 0, Func: 1: Read Coils
13	0.621874	192.168.1.10	192.168.1.38	TCP	60	502 → 55273 [ACK] Seq=11 Ack=13 Win=511 Len=0
29	1.194671	192.168.1.10	192.168.1.38	Modbus/TCP	64	Response: Trans: 30; Unit: 0, Func: 1: Read Coils

*Figura 59: Tráfico generado por el sistema SCADA [38]*

En caso de realizar un sistema SCADA con WinCC, se puede realizar directamente sobre TIA Portal gracias a su integración. A continuación, se muestra un ejemplo de cómo sería un sistema SCADA muy simple para este proyecto. El sistema no es funcional, pero es para mostrar cómo sería visualmente esta herramienta. En TIA creamos esta aplicación como un HMI normal, tiene una interfaz gráfica donde se van añadiendo los distintos componentes que conforman al sistema y se vinculan las variables del PLC a estos.



*Figura 60: Ejemplo de sistema SCADA con WinCC*

## **5.6 INYECCIÓN DE CÓDIGO SOBRE PLCs**

Este tipo de ataques es uno de los que más puede comprometer el funcionamiento de una planta o entorno industrial donde se encuentren los equipos donde se ha inyectado código. El funcionamiento de este tipo de ataques se basa en interceptar la carga del programa entre el TIA Portal y el PLC (en caso de hablar de autómatas Siemens). Al interceptar esta comunicación, el atacante es capaz de crear su propio programa para el PLC e inyectarlo en los paquetes de dicha comunicación. Con esto, el atacante consigue poder subir el programa que él quiera sin que sea la persona que inicia y autoriza la carga sobre el autómata.

Para ello hay que entender el protocolo que se encarga de transportar estos datos entre el PC y el PLC. Este protocolo, en el caso de los equipos del laboratorio (S7-1500) se trata de la versión mejorada del S7Comm, conocida como S7Comm plus. El funcionamiento de este protocolo se explica con detalle en el Anexo 2.

Debido a que para llevar a cabo este tipo de ataques se basan sobre S7Comm plus, concluimos que es un ataque con una alta complejidad como para que se lleve a cabo por parte de los alumnos. Esto se debe a la seguridad que este protocolo aporta a la comunicación. Aun así, es un ataque que se puede realizar, pero requiere muchos conocimientos previos y saber de otros conceptos que no proceden.

Viendo este ataque desde un alto nivel, una vez interceptada la comunicación y comprometida la red y tenemos la posibilidad de cambiar el payload de los paquetes que se intercambian la estación del ingeniero (desde donde se sube el software) y el PLC. Para ello, también es muy importante conocer y descubrir cómo se forman los paquetes y ver que bytes son los que transportan el payload de los paquetes que cargan el código al PLC. Una vez conocido y detectado esto, se puede llevar a cabo el ataque con un packet replay (nos basamos en los paquetes de subida de código original para modificarlos) o con un Man in the Middle, interceptando la comunicación y variando los bytes de la carga del código.

Para más detalle sobre este tipo de ataques, recomiendo consultar la siguiente referencia [39].

## Capítulo 6. ANÁLISIS DE RESULTADOS

Con todo lo anterior, podemos analizar los resultados que hemos obtenido de los distintos ataques y de los distintos protocolos. Los ataques estudiados en su gran mayoría son muy simples de llevar a cabo y no requieren mucho estudio previo, por lo que es muy adecuado para el desarrollo de unas prácticas de ciberseguridad.

En cuanto al protocolo S7Comm, es interesante ya que los ataques los realizamos a través de varias herramientas, dos de las cuales se pueden emplear para otros ataques. No solo es importante saber los ataques y cómo se realizan, sino que también cómo funciona el protocolo. En este caso, como se ha visto, el protocolo no cifra la información y desde Wireshark podemos conseguir mucha información sobre las variables que se están empleando y cómo se usan. Esto también facilita otro tipo de ataques como el packet replay.

En cuanto al hermano mayor de este protocolo, el S7Comm Plus, podemos afirmar que el objetivo con el que se desarrolló este protocolo se cumple con creces. Con esta versión se complican los ataques mucho, aunque son posibles. A pesar de que concluimos que es un protocolo demasiado complejo como para realizar prácticas, es interesante ver cómo funciona y en que se basa su seguridad.

Otro de los ataques que se plantearon en un principio para el desarrollo de este proyecto, es la inyección de código sobre los PLCs. Como este ataque se basa en el protocolo S7Comm Plus, no se realiza debido a su complejidad. A pesar de ello, también se estudia levemente en qué consisten estos ataques, ya que es interesante para el desarrollo del conocimiento de los alumnos, ya que se podría aplicar este tipo de ataque a otros dispositivos con otros protocolos.

En cuanto a Modbus, es un buen protocolo para iniciarse en temas de ciberseguridad industrial. Es un protocolo muy simple que transmite la información en texto plano y eso facilita mucho los ataques y comprender como funcionan. Gracias a esto, podemos realizar varios tipos de ataques más complejos como pueden ser el packet replay y un man in the

middle sin que la complejidad sea demasiado alta. Este protocolo nos permite aprender el funcionamiento de estos ataques con detalle. Debido a esto, sería recomendable realizar las prácticas de este protocolo de los primeros, para asentar los conocimientos de los ataques antes de proceder con protocolos más complejos.

En cuanto a Profinet, también se demuestra que, a pesar de ser uno de los protocolos más usados y estandarizados de la industria, para ciertos tipos de tráfico se pueden realizar ataques fácilmente. En el laboratorio hay presentes varios equipos que se comunican a través de profinet pero o son Siemens (prioriza comunicación S7) o no son equipos que nos permitan realizar escrituras asíncronas sobre registros o valores discretos. A pesar de esto, se ha podido crear dos pequeñas pruebas en las que se crean paquetes de descubrimiento (para que los equipos en red nos avisen si son compatibles con profinet) y otro para parpadear el LED de encendido de un autómat. Para realizar estas dos prácticas, se ha partido de un software que nos permite realizar estas funciones de forma simple. Al mandar estos paquetes desde la aplicación, podemos capturar los paquetes desde Wireshark y realizar un ataque de packet replay. Para poder hacer estos “ataques”, debemos de descomponer el stream en hexadecimal del paquete original para modificar los campos necesarios, como en este caso las MACS. El payload del paquete original se debe mantener idéntico para que el paquete, al ser recibido y procesado por el destino, realice la acción deseada. El resto de las pruebas y ataques que se pensaron para este proyecto, no se han podido realizar debido a la falta de herramientas. A pesar de esto, en las conclusiones, se listan una serie de soluciones o directrices a seguir en caso de querer implementar este protocolo de forma completa en el laboratorio de ciberseguridad.

Por último, en cuanto a los sistemas SCADA, es interesante estudiarlos debido a que es algo muy extendido en la industria y cada vez más utilizado. Estos sistemas van en crecimiento debido al éxito que están teniendo los sistemas distribuidos en todas las industrias. Nos permite en un software simple, controlar y monitorizar los datos más importantes de una fábrica. En este proyecto se ha creado e implementado un sencillo sistema SCADA creado en Python y con comunicaciones sobre S7Comm y Modbus. Con este sistema, al ser “virtual”, aunque sea muy sencillo, nos ha permitido crear un sistema algo más real. Esto se debe a que, en la industria, tenemos muchos sensores, accionadores y otros equipos



conectados en red, que no se comunican todos con el mismo protocolo. Estos sistemas suelen trabajar con varios protocolos y marcas de fabricantes y nos son útiles ya que nos centralizan toda la información, ya sea de lectura o de escritura. Debido a esto, los ataques realizados sobre este sistema son los ya vistos en los apartados específicos de estos dos protocolos. En caso de tratarse de un SCADA con otros protocolos, los ataques serán los concretos de los protocolos con los que trabaja el sistema.

## Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Todos los objetivos que se propusieron al principio del proyecto han sido estudiados y tratados. Las conclusiones en algunos casos han sido más que satisfactorias y en otras nos hemos topado con inconvenientes.

La principal conclusión es que sí se puede desarrollar un laboratorio de ciberseguridad en ICAI. Para ello se pueden emplear los ataques citados en este trabajo como manual. Con lo explicado en este trabajo, los alumnos serán capaces de desarrollar su conocimiento con respecto a la ciberseguridad en entornos industriales. Esto es importante debido a la gran revolución en la que está involucrada este sector. Los protocolos que se han tratado en este trabajo son los principales que se manejan en la industria, por lo que las prácticas serían bastante realistas.

A pesar de todos esto, en este trabajo, también se ha realizado una pequeña “auditoría” para ver si es viable realizar este laboratorio y cómo se implementaría. Esta parte de auditoría es importante, ya que como se ha explicado en la primera parte de análisis de redes, se observa que hacen falta equipos para que el proyecto salga adelante. En este caso han sido unos switches. También se podría plantear el obtener equipos más antiguos para aprender cómo realizar algunos de los ataques vistos directamente en equipos reales en vez de simuladores. También se podrían realizar otro tipo de ataques como, por ejemplo, en el caso de tener un PLC Siemens de la serie 1200 o inferior, podríamos realizar ataques de *start/stop*. También en caso de tener PLCs de la gama 300 y 200, aparte de todo lo anterior, se pueden realizar los ataques de inyecciones de código ya citados anteriormente.

Otras conclusiones importantes son que, durante el desarrollo del proyecto, nos hemos dado cuenta de que ciertos protocolos son complejos. Por ejemplo, analizando el tráfico presente en las comunicaciones entre equipos de Siemens, nos esperábamos encontrar Profinet o S7Comm, no S7Comm Plus.

Después de haber visto los resultados obtenidos de Profinet, el paso más lógico es proponer soluciones tecnológicas para poder implementar de forma completa este protocolo. En esta fase se ha hecho un estudio de las soluciones disponibles en el mercado, evaluando hardware y software. Definitivamente, creemos que la mejor opción sería emplear alguno de los softwares que se citarán a continuación. Esto se debe a que los equipos de hardware Siemens que nos permiten realizar escrituras de datos discretos y no cíclicos a través de Profinet, son equipos antiguos y de un precio todavía algo elevado. Debido a esto, se ha optado por soluciones de software. En cuanto a las aplicaciones disponibles, tampoco hay tantas opciones y no son de libre uso.

La primera opción es optar por un simulador de un máster de Profinet [40], que nos proporciona la empresa Anybus (ofrecen este tipo de simuladores de muchos protocolos y equipos físicos). En cuanto a este programa, la implementación es completamente virtual, ya que no tiene buena comunicación con equipos reales (TIA Portal). Es importante destacar que no hemos podido probar esta solución, ya que el departamento de ventas no ofrece pruebas de esta herramienta y no nos han contestado de vuelta.

A pesar de esto, la segunda herramienta que proponemos es la que recomendamos para la implementación de Profinet en el laboratorio. Esta herramienta es la citada en el apartado de Profinet usada para sacar los payloads de los mensajes que se iban a crear. Esta herramienta es Profinet Commander. Esta herramienta tiene una versión gratis, la cual hemos usado en el apartado de este protocolo. Esta herramienta, en la versión de pago nos ofrece opciones muy interesantes para desarrollar prácticas y ataques sobre este protocolo. Nos ofrece leer variables y escribir sobre ellas, entre otras cosas, que es lo que nos interesa. La principal ventaja que nos ofrece este software es la posibilidad de integrarlo con TIA Portal a través de los archivos GSD y tener una implementación completa con los equipos físicos del laboratorio. Con este software se podría implementar ataques parecidos a los vistos en Modbus, con un packet replay y man in the middle. Esta herramienta tiene un precio de cara al cliente normal (particular) de unos 600 dolares, lo cual es bastante elevado y requiere una licencia por instalación.

De cara a continuar este proyecto en un futuro, se podrían implementar equipos de otras marcas o equipos Siemens que soporten más protocolos. También para que todo esto sea mucho más visual, se podría crear una maqueta en la que distintos sensores y actuadores funcionen con distintos protocolos y así tener un sistema centralizado sobre el que realizar distintos ataques.

## Capítulo 8. BIBLIOGRAFÍA

- [1] INCIBE. 2015. *Arranca la Red Nacional de Laboratorios Industriales, una iniciativa para elevar el nivel de seguridad de las infraestructuras industriales*. Disponible en: <https://www.incibe.es/sala-prensa/notas-prensa/rnli> [Consulta: 10 de octubre 2020].
- [2] En.wikipedia.org. 2021. *Stuxnet - Ataque PLCs Irán*. Disponible en: <https://en.wikipedia.org/wiki/Stuxnet> [Consulta 14 de noviembre 2020].
- [3] Nmap.org. n.d. *Nmap: the Network Mapper - Free Security Scanner*. Disponible en: <https://nmap.org/> [Consulta: 23 de noviembre 2020].
- [4] Homebrew. n.d. *Homebrew*. Disponible en: [https://brew.sh/index\\_es](https://brew.sh/index_es) [Consulta: 23 de noviembre 2020].
- [5] Wiki.wireshark.org. n.d. *S7comm - The Wireshark Wiki*. Disponible en: <https://wiki.wireshark.org/S7comm> [Consulta: 10 de noviembre 2020].
- [6] Nodered.org. n.d. *Node-RED*. Disponible en: <https://nodered.org/> [Consulta: 4 de junio 2021].
- [7] Nodered.org. n.d. *Running Node-RED locally: Node-RED*. Disponible en: <https://nodered.org/docs/getting-started/local#prerequisites> [Consulta: 4 de junio 2021].
- [8] Oleo Blanco, M., 2021. *Proyecto TIA Portal y Node-Red*. GitHub. Disponible en: <https://github.com/miguelob/TFG/blob/main/S7comm/PLCHMICyber.rar> [Consulta: 4 de junio 2021].
- [9] Oleo Blanco, M., 2021. *Flow de Node-Red para lectura y escritura en TIA Portal*. GitHub. Disponible en: <https://github.com/miguelob/TFG/blob/main/S7comm/flows.json> [Consulta: 4 de junio 2021].
- [10] Oleo Blanco, M., 2021. *Demostración del Impersonating del HMI*. Youtube. Disponible en: [https://youtu.be/nmDZZVDXV\\_I](https://youtu.be/nmDZZVDXV_I) [Consulta: 4 de junio 2021].
- [11] Nettoplcsim.sourceforge.net. n.d. *NetToPLCsim - Network extension for Plcsim*. Disponible en: <http://nettoplcsim.sourceforge.net/> [Consulta: 12 de junio 2021].

- [12] Nardella, D., n.d. *Snap7 Homepage*. Snap7.sourceforge.net. Disponible en: <http://snap7.sourceforge.net/> [Consulta: 24 de junio 2021].
- [13] Oleo Blanco, M., 2021. *Código Python escritura en variables S7Comm*. GitHub. Disponible en: <https://github.com/miguelob/TFG/blob/main/S7comm/ProyectoCyber.py> [Consulta: 24 de junio 2021].
- [14] Oleo Blanco, M., 2021. *Código Python impersonating HMI S7Comm*. GitHub. Disponible en: <https://github.com/miguelob/TFG/blob/main/S7comm/ProyectoCyberLab.py> [Consulta: 24 de junio 2021].
- [15] Oleo Blanco, M., 2021. *Demostración del Impersonating del HMI con Python*. Youtube. Disponible en: <https://youtu.be/ajvkUDptOJU> [Consulta: 24 de junio 2021].
- [16] Oleo Blanco, M., 2021. *Demostración Snap7 con variables Q y tabla de observación PLCSIM*. Youtube. Disponible en: <https://youtu.be/bcqXCAxFRT0> [Consulta: 24 de junio 2021].
- [17] Automation.ucoz.com. n.d. *Process Simulator - Idea*. Disponible en: <https://automation.ucoz.com/> [Consulta: 12 de junio 2021].
- [18] Oleo Blanco, M., 2021. *Configuración y funcionamiento del ataque ProcessSimulator*. Youtube. Disponible en: <https://youtu.be/OKSKDYc6Okg> [Consulta: 12 de junio 2021].
- [19] Oleo Blanco, M., 2021. *Pcapng del tráfico S7Comm con node-red*. GitHub. Disponible en: [https://github.com/miguelob/TFG/blob/main/S7comm/s7\\_w.pcap](https://github.com/miguelob/TFG/blob/main/S7comm/s7_w.pcap) [Consulta: 4 de junio 2021].
- [20] Oleo Blanco, M., 2021. *Ejemplo lectura y escritura de un DataBlock empleando Python*. GitHub. Disponible en: <https://github.com/miguelob/TFG/blob/main/S7comm/snap7.py> [Consulta: 13 de mayo 2021].
- [20] Es.wikipedia.org. n.d. *Protocolo Modbus*. Disponible en: <https://es.wikipedia.org/wiki/Modbus> [Consulta: 19 de enero 2021].
- [21] Sourceforge.net. n.d. *ModbusPal - a Java MODBUS simulator*. Disponible en: <https://sourceforge.net/projects/modbuspal/files/modbuspal/> [Consulta: 19 de enero 2021].
- [22] SourceForge. n.d. *QModMaster*. Disponible en: <https://sourceforge.net/projects/qmodmaster/> [Consulta: 19 de enero 2021].

- [23] Westoahu.hawaii.edu. n.d. *Crafting Modbus Packets with Scapy*. Disponible en: [https://westoahu.hawaii.edu/cyber/wp-content/uploads/2018/09/craft\\_modbus.docx](https://westoahu.hawaii.edu/cyber/wp-content/uploads/2018/09/craft_modbus.docx)  
[Consulta: 10 de enero 2021].
- [24] Scapy.readthedocs.io. n.d. *scapy.contrib.modbus — Scapy 2.4.5. documentation*.  
Disponible en: <https://scapy.readthedocs.io/en/latest/api/scapy.contrib.modbus.html>  
[Consulta: 13 de enero 2021].
- [25] Oleo Blanco, M., 2021. *Código Python para Packet Replay completo de Modbus*. GitHub.  
Disponible en: <https://github.com/miguelob/TFG/blob/main/Modbus/PacketReplay-Completo.py> [Consulta: 17 de enero 2021].
- [26] Oleo Blanco, M., 2021. *Código Python para Packet Replay de Modbus con socket TCP*. GitHub. Disponible en:  
<https://github.com/miguelob/TFG/blob/main/Modbus/PacketReplay.py>  
[Consulta: 17 de enero 2021].
- [27] Edcontrol.com. n.d. *PROFIBUS versus PROFINET: Estrategias de comparación y migración*.  
Disponible en:  
<http://www.edcontrol.com/index.php/instrumentacion/instrumentacion-189/item/116-profibus-versus-profinet-estrategias-de-comparacion-y-migracion#:~:text=PROFIBUS%20es%20un%20bus%20de,m%C3%A1s%20r%C3%A1pid%20y%20m%C3%A1s%20flexible>. [Consulta: 11 junio 2021].
- [28] aula21 | Formación para la Industria. n.d. *PROFINET: Qué es y cómo funciona | Comunicaciones Industriales*. Disponible en: [https://www.cursosaula21.com/profinet-que-es-y-como-funciona/#:~:text=PROFINET%20\(Process%20Field%20Network\)%20es,datos%20entre%20controladores%20y%20dispositivos](https://www.cursosaula21.com/profinet-que-es-y-como-funciona/#:~:text=PROFINET%20(Process%20Field%20Network)%20es,datos%20entre%20controladores%20y%20dispositivos). [Consulta: 11 de junio 2021].
- [29] Wiki.wireshark.org. n.d. *PROFINET - The Wireshark Wiki*. Disponible en:  
<https://wiki.wireshark.org/PROFINET> [Consulta: 11 de junio 2021].
- [30] Profinetcommander.com. n.d. *PROFINET Commander – Take Command of your PROFINET Network*. Disponible en: <https://profinetcommander.com/> [Consulta: 13 de junio 2021].
- [31] Oleo Blanco, M., 2021. *Código DCP profinet Discovery.py*. GitHub. Disponible en:  
<https://github.com/miguelob/TFG/blob/main/Profinet%20%26%20Profibus/Discovery.py>  
[Consulta: 14 de junio 2021].

- [32] Oleo Blanco, M., 2021. *Código led.py*. GitHub. Disponible en:  
<https://github.com/miguelob/TFG/blob/main/Profinet%20%26%20Profibus/FlashLED.py> [Consulta: 14 de junio 2021].
- [33] Oleo Blanco, M., 2021. *Tráfico profinet entre Cognex y PLC*. GitHub. Disponible en:  
<https://github.com/miguelob/TFG/blob/main/Profinet%20%26%20Profibus/camara.png> [Consulta: 24 de marzo 2021].
- [34] aula21 | Formación para la Industria. n.d. *Qué es un sistema SCADA, para qué sirve y cómo funciona* | Aula21. Disponible en: <https://www.cursosaula21.com/que-es-un-sistema-scada/> [Consulta: 23 de junio 2021].
- [35] siemens.com Global Website. n.d. *SIMATIC WinCC V7*. Disponible en:  
<https://new.siemens.com/global/en/products/automation/industry-software/automation-software/scada/simatic-wincc-v7.html>  
[Consulta: 17 de junio 2021].
- [36] Pymodbus.readthedocs.io. n.d. *Welcome to PyModbus's documentation! — PyModbus 2.5.0 documentation*. Disponible en: <https://pymodbus.readthedocs.io/en/latest/>  
[Consulta: 26 de junio 2021].
- [37] Oleo Blanco, M., 2021. *Aplicación SCADA en Python*. GitHub. Disponible en:  
<https://github.com/miguelob/TFG/blob/main/SCADA/SCADA LAB.py>  
[Consulta: 28 de junio 2021].
- [38] Oleo Blanco, M., 2021. *Tráfico Modbus y S7Comm generado por el sistema SCADA*. GitHub. Disponible en: <https://github.com/miguelob/TFG/tree/main/SCADA>  
[Consulta: 28 de junio 2021].
- [39] Ahmed, I. and Yoo, H., 2019. *Inyección de código sobre PLCs*. researchgate. Disponible en:  
[https://www.researchgate.net/publication/331371639\\_Control\\_Logic\\_Injection\\_Attacks\\_on\\_Industrial\\_Control\\_Systems](https://www.researchgate.net/publication/331371639_Control_Logic_Injection_Attacks_on_Industrial_Control_Systems) [Consulta: 13 de mayo 2021].
- [40] Anybus.com. 2021. *PROFINET Master Simulator*. Disponible en:  
<https://www.anybus.com/products/gateway-index/specific-gateways/master-simulators/detail/profinet-master-simulator?ordercode=024710>  
[Consulta: 8 de abril 2021].



# ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Los ODS (Objetivos de Desarrollo Sostenible) son 17 objetivos y áreas de intervención en la que los estados miembros de la ONU se han puesto como objetivo. Todo esto se establece para acabar con la desigualdad entre países y mejorar la calidad de vida de todas las personas por igual. También, nosotros como alumnos tenemos que aportar a estos objetivos, ya que al ser ingenieros (como muchos otros empleos), tenemos un compromiso e impacto sobre la sociedad y su desarrollo.

Mi proyecto a priori no tiene mucho que aportar a los ODS, pero si leemos con detalle, en alguno de ellos sí que se puede integrar. Como se trata de ciberseguridad en entornos industriales, todo lo que tenga que ver con realizar los procesos más eficientes, Otro punto de vista importante es aplicar esta ciberseguridad para evitar malfuncionamientos intencionados de los productos y evitar contaminaciones excesivas de los mismos y de las plantas de producción.

Debido a esto, el objetivo que más se parece a mi proyecto es el 9. Este se titula industria, innovación e infraestructuras. En concreto el 9.5 y 9.c. En el primero, se busca la innovación de las infraestructuras, sobre todo de los países en desarrollo. Para conseguir esto es clave realizarlo de forma segura, ya que muchas veces los países con pocos recursos no invierten en ciberseguridad y acaban siendo objetivo de muchos ataques y hace que la producción sea contraproducente. Para ello es muy importante la investigación para poder tener todos los fallos de seguridad bajo control y desarrollar comunicaciones más seguras.

En el apartado 9.C se centra en aumentar el acceso de la tecnología de la información y comunicación a las personas a las personas. Como ya he dicho antes, esto se debe de realizar de forma segura, ya que podríamos dar estos servicios, pero si no son seguros, se verán comprometidos y servirían de poco. En resumen, es un proyecto que se puede involucrar en varios ODS debido a las necesidades actuales de la tecnología, pero no hay ningún objetivo centrado en ciberseguridad como tema principal.

## ANEXO II: S7COMM PLUS

En este anexo se va a tratar sobre la versión mejorada del S7comm, conocida como S7comm plus. Este protocolo se basa en la versión anterior y se creó para incrementar la seguridad que aporta el protocolo. Como se explicará a lo largo del anexo, podemos confirmar que hace muy buen trabajo aportando confidencialidad, integridad y no repudio en las comunicaciones.

Este protocolo se observa en los autómatas Siemens de la serie 1200 y 1500, que fueron los primeros en incorporar el protocolo. Estos PLCs usan dicho protocolo para cualquier comunicación con cualquier otro equipo moderno de Siemens. También se emplea para la comunicación con TIA Portal, aportando seguridad en la carga, descarga y lecturas entre PC y PLC, uno de los puntos más débiles del S7Comm, donde se puede fácilmente interceptar la carga del programa de TIA al PLC e inyectar código malicioso sobre el autómata.

En concreto, el protocolo implementa sistemas de autenticación y de encriptación de punto a punto en la comunicación. Para ello, se basa en un proceso de handshake para negociar las claves y autenticar a los equipos que participan en la comunicación, como se explicará posteriormente. Debido a esto, es muy complicado que un atacante puede comprometer la red. Ataques del tipo del packet replay también son muy complejos de realizar, ya que habría que desencriptar el paquete original, modificar lo que se desee y por último, volver a aplicarle la encriptación correspondiente y recalcular varios parámetros.

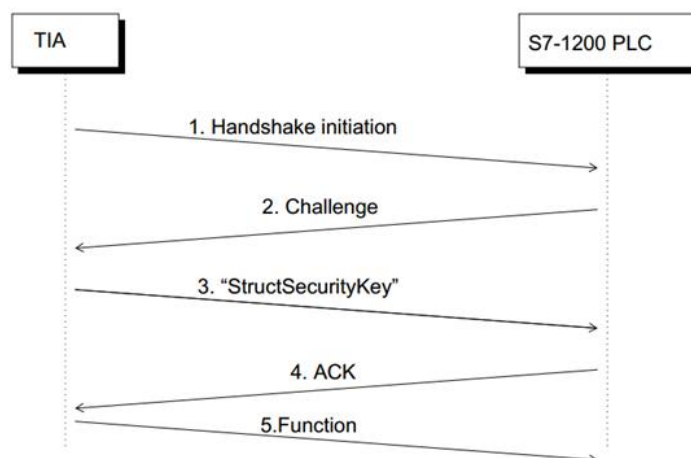
Otro detalle que hay que tener en cuenta es que no todos los equipos que usan S7Comm Plus tienen el mismo nivel de seguridad. Hay parte de la seguridad del protocolo que recae sobre las versiones del PLC que usemos y del TIA Portal. En la siguiente tabla se puede observar cómo cambia. En los equipos del laboraorio está presente la versión de TIA Portal 16, de la cual no tenemos información y complica la realización de ataques. Por ejemplo, para demostrar las diferencias dentro de las distintas versiones del S7Comm Plus, con un PLC de la serie S7-1200, se pueden crear paquetes para parar, arrancar y resetear el PLC, mientras que con uno de la serie S7-1500 no es posible.

	S7-1200	S7-1500 V1.1	S7-1500 V1.8	S7-1500 V2.1
TIA V12	P2 <sup>-</sup>	P2	P2	P2
TIA V14	P2 <sup>-</sup>	P2	P3	P3
TIA V15	P2 <sup>-</sup>	P2	P3	P3

*Tabla 1: Versiones del S7Comm Plus según PLC y TIA Portal [1]*

Antes de ver como se efectúa el proceso de handshake, es necesario saber que wireshark no nos muestra este tráfico por defecto, ya que tenemos que añadir el disector. Este lo podemos conseguir en la referencia [2] y según el sistema operativo se añade de formas distintas. En sistemas Windows basta con descargar el .dll y ponerlo en la carpeta de plugins de Wireshark. Por otro lado, para sistemas Linux, tenemos que descargarnos el código fuente del disector y el de Wireshark, juntar los dos códigos (el disector hay que meterlo con el resto) y compilar la aplicación. En la referencia [3] está el enlace al repositorio de GitHub donde se encuentra el código fuente de ambos empleados para esta sección.

Este proceso de handshake es muy parecido al de otros protocolos, en los que se negocia las claves que se van a emplear en la comunicación para asegurar la integridad de los datos y la seguridad del sistema. A continuación, se muestra como es este proceso de handshake en un diagrama secuencial (para un S7-1200):



*Figura 1: Proceso de handshake [1]*

En la siguiente imagen se muestra el tráfico generado en este proceso y se detallará el contenido de estos mensajes:

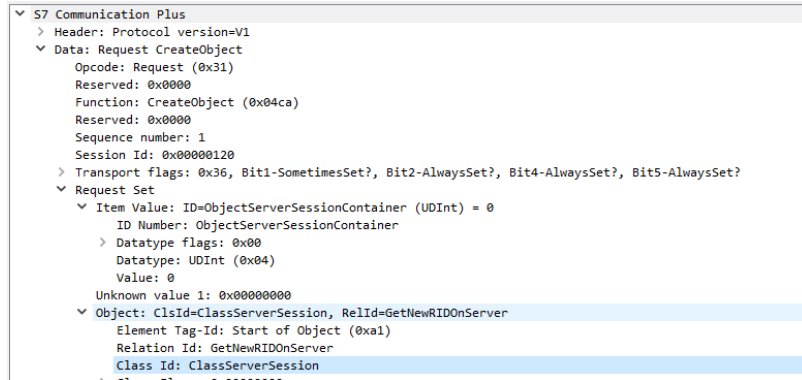
28	27.086742	IntelCor_bc...	Siemens_9f:...	ARP	60	192.168.56.6	is at 68:05:ca:bc:d3:05
29	27.086745	192.168.56.16	192.168.56.6	TCP	60	102 → 53684	[SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
30	27.087142	192.168.56.6	192.168.56.16	TCP	60	53684 → 102	[ACK] Seq=1 Ack=1 Win=64240 Len=0
31	27.087498	192.168.56.6	192.168.56.16	COTP	89	CR TPDU	src-ref: 0x0012 dst-ref: 0x0000
32	27.087764	192.168.56.16	192.168.56.6	COTP	89	CC TPDU	src-ref: 0x0010 dst-ref: 0x0012
33	27.088812	192.168.56.6	192.168.56.16	S7COM...	284	+53684 Ver:[V1]	Seq=1 [Req CreateObject] ObjectServerSessionContainer
34	27.093299	192.168.56.16	192.168.56.6	S7COM...	288	+53684 Ver:[V1]	Seq=1 [Res CreateObject] Retval=OK ObjId=DynObjX7.0.3
35	27.093302	192.168.56.6	192.168.56.16	COTP	61	DT TPDU (0)	[COTP fragment, 0 bytes]
36	27.097266	192.168.56.6	192.168.56.16	S7COM...	476	+53684 Ver:[V2]	Seq=2 [Req SetMultiVariables] ObjId=DynObjX7.0.3667
37	27.097269	192.168.56.16	192.168.56.6	TCP	60	102 → 53684	[ACK] Seq=270 Ack=695 Win=7770 Len=0
38	27.143714	192.168.56.16	192.168.56.6	S7COM...	86	+53684 Ver:[V2]	Seq=2 [Res SetMultiVariables] Retval=OK
39	27.143716	192.168.56.6	192.168.56.16	COTP	61	DT TPDU (0)	[COTP fragment, 0 bytes]
40	27.143971	192.168.56.6	192.168.56.16	S7COM...	155	+53684 Ver:[V3]	Seq=3 [Req GetVarSubStreamed]
41	27.144332	192.168.56.16	192.168.56.6	TCP	60	102 → 53684	[ACK] Seq=302 Ack=803 Win=8091 Len=0
42	27.146141	192.168.56.16	192.168.56.6	S7COM...	123	+53684 Ver:[V3]	Seq=3 [Res GetVarSubStreamed] Retval=OK

*Figura 2: Tráfico del proceso de handshake [4]*

Como se puede apreciar en la captura anterior, los paquetes S7Comm plus se basan al igual que el S7Comm en TCP y COTP. Es importante también observar que, según el paquete, se puede observar que tiene una versión distinta (V1, V2 o V3). Estas versiones no se refieren a las versiones del protocolo vistas anteriormente, si no a la fase de la comunicación a la que hacen referencia. Por ejemplo, el 1 y 2 son las que establecen la comunicación con el handshake y comparten sus claves y el 3 es el que indica que la comunicación ya se ha establecido y el tráfico es normal.

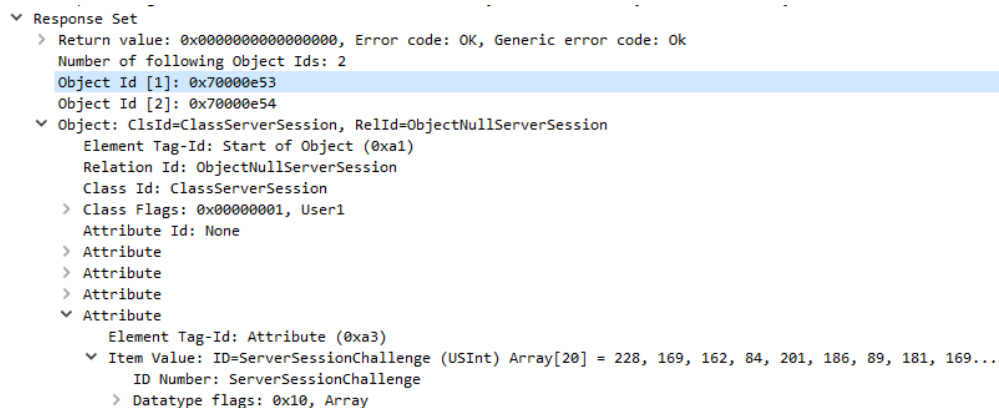
Ahora, vamos a entrar en detalle de todas las fases vistas en el esquema del handshake y como se reflejan en los paquetes capturados. La fase de iniciación del handshake son los paquetes 31 y 32. Estos paquetes contienen los campos Connect Request (CR) y Connect Confirm (CC) respectivamente. La fase del Challenge son los paquetes 33 y 34. En este punto, el plc genera 20 bytes aleatorios que se usarán para identificar la sesión y otros

parámetros. El paquete 33 es el que manda TIA al PLC y en este se solicita crear el objeto `ClassServerSession`:



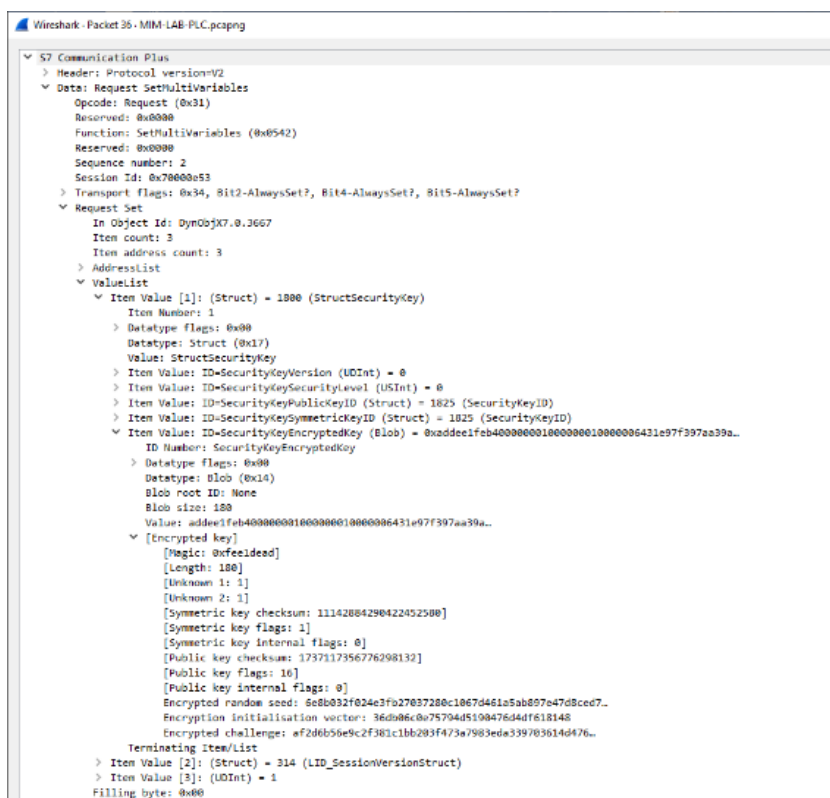
*Figura 3: TIA solicita al PLC crear el objeto `ClassServerSession` [4]*

A este mensaje le responde con la información sobre el firmware que el PLC esta corriendo y 20 bytes generados aleatoriamente citados anteriormente (`ServerSessionChallenge`). También se manda el `sessionID`, que se emplea para saber que mensajes son de cada sesión:



*Figura 4: `ServerSessionChallenge` y `SessionID` [4]*

La siguiente fase es la llamada StructSecurityKey y se corresponden a los paquetes 36 y 38. TIA al recibir los 20 bytes del PLC, emplea los 16 bytes centrales para cálculos de la clave publica de la sesión. Para ello utiliza complejos algoritmos criptográficos, desde más sencillos como XOR, pasando por SHA-156, HMAC-SHA-256, CBC-MAC, AES-ECB y ECC. Con esto genera datos de autenticación que enviará al PLC. Esta información también la podemos ver dentro del paquete:



*Figura 5: StructSecurityKey[4]*

El PLC al recibir esto, lo desencriptará con su clave privada. Si este proceso es correcto, se lo avisará al TIA. En nuestro ejemplo, esta respuesta es la del paquete 38. A partir de este punto, la comunicación está establecida y la comunicación empieza a transportar el tráfico generado por el programa específico.

Como se puede intuir con todo lo visto, es bastante complejo de realizar ataques sobre este protocolo. Esta versión cumple bastante bien con su principal propósito: la seguridad. Esto no hace que sea imposible realizar ataques, ya que se puede realizar por otras vías y por otros protocolos, pero una buena medida de seguridad sería solo emplear S7Comm plus en las

comunicaciones entre los equipos de una fábrica. De este modo se reduce mucho el riesgo, aunque nunca sería cero, sería un nivel muy aceptable.

## Bibliografía

- [1] Programmersought.com. n.d. *S7Comm Plus protocol research - Programmer Sought*. Disponible en: <https://www.programmersought.com/article/64975702507> [Consulta: 26 de octubre 2020].
- [2] SourceForge. n.d. *S7comm Wireshark dissector plugin*. Disponible en: <https://sourceforge.net/projects/s7commwireshark/> [Consulta: 29 de octubre 2020].
- [3] Oleo Blanco, M., 2020. *Wireshark + S7Comm Plus*. GitHub. Disponible en: [https://github.com/miguelob/TFG/tree/main/wireshark%20%2B%20s7comm\\_plus](https://github.com/miguelob/TFG/tree/main/wireshark%20%2B%20s7comm_plus) [Consulta: 29 de octubre 2020].
- [4] Oleo Blanco, M., 2020. *Tráfico S7Comm Plus*. GitHub. Disponible en: <https://github.com/miguelob/TFG/blob/main/S7comm-plus/MIM-hmi-plc.pcapng> [Consulta: 30 de octubre 2020].