HACKATHON 2025

Comunicació amb PLCs industrials

Membres: Mireia Terricabras Vilardell, Amaru Alviña Valencia, Pol Lázaro Roca, Alex Marín Liébana

Escola Politècnica Superior

Universitat de Lleida

Treball presentat a: bonÀrea

ÍNDEX

INTRODUCCIÓ

Aquest informe descriu el procés d'anàlisi, desenvolupament i resolució del repte proposat per bonÀrea en el marc de la HackEPS 2025, relacionat amb la comunicació amb PLCs industrials i el control d'indicadors mitjançant diferents protocols i tecnologies.
L'objectiu principal del repte era establir comunicacions fiables i segures amb el PLC, identificar i manipular els blocs de dades corresponents als senyals NS1, NS2 i NS3, i finalment reproduir la seqüència de temps necessària per activar els indicadors a la HMI.

Al llarg del treball s'han aplicat eines com nmap, UAExpert, Wireshark, i llibreries de Python com python-snap7 i opcua, amb l'objectiu de comprendre l'estructura interna del PLC, manipular dades en temps real i deduir la lògica de funcionament dels temporitzadors.
Aquest document recull els passos seguits, els obstacles trobats i les solucions aplicades fins aconseguir completar amb èxit el repte.

PROCEDIMENT

Identificació del PLC i primera connexió

Per iniciar la prova vam utilitzar nmap per identificar els dispositius connectats a la xarxa preparada per al repte. Un cop localitzades les diferents IPs, vam comprovar que podíem accedir al PLC mitjançant el navegador web.

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sn 10.72.101.0/24
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-22 06:01 EST
Nmap scan report for 10.72.101.1
Host is up (0.042s latency).
MAC Address: 7C:8B:CA:F6:38:68 (TP-Link Technologies)
Nmap scan report for 10.72.101.16
Host is up (0.00064s latency).
MAC Address: D8:B3:2F:11:3D:D1 (Cloud Network Technology Singapore PTE.)
Nmap scan report for 10.72.101.22
Host is up (0.27s latency).
MAC Address: CA:6D:EA:C5:7D:E9 (Unknown)
Nmap scan report for 10.72.101.23
Host is up (0.27s latency).
MAC Address: C0:35:32:D1:61:BF (Liteon Technology)
Nmap scan report for 10.72.101.68
Host is up (0.023s latency).
MAC Address: 28:63:36:48:D4:D8 (Siemens AG)
Nmap scan report for 10.72.101.72
Host is up (0.024s latency).
MAC Address: 8C:F3:19:77:91:DE (Siemens Industrial Automation Products, Chengdu)
Nmap scan report for 10.72.101.86
Host is up (0.12s latency).
MAC Address: AC:64:17:B6:A0:E0 (Siemens AG)
Nmap scan report for 10.72.101.87
Host is up (0.12s latency).
MAC Address: AC:64:17:36:33:A9 (Siemens AG)
Nmap scan report for 10.72.101.90
Host is up (0.063s latency).
MAC Address: 30:13:89:45:EF:0D (Siemens AG, Automations & Drives,)
Nmap scan report for 10.72.101.226
Host is up (0.055s latency).
MAC Address: AC:64:17:3A:BA:16 (Siemens AG)
Nmap scan report for 10.72.101.21
Host is up.
Nmap done: 256 IP addresses (11 hosts up) scanned in 28.89 seconds
```

Selecció de protocols i eines de desenvolupament

Després de consultar la documentació proporcionada, vam confirmar que havíem de comunicar-nos amb el PLC utilitzant els protocols **S7** i **OPC UA**. Per aquest motiu vam estudiar diferents llibreries i opcions de programació que ens permetessin treballar amb aquests protocols. Finalment, vam optar per utilitzar **Python**, amb les llibreries **python-snap7** per S7 i **opcua** per OPC UA.

Control inicial del NS1 (protocol S7)

Seguint els exemples inicials de la llibreria python-snap7, vam aconseguir establir comunicació amb el PLC i activar el primer indicador (NS1) des del nostre codi.

Tot i que aquest primer script era bàsic i no resolia tot el repte, ens va permetre validar:

- Connexió S7
- Lectura/escriptura de blocs de dades
- Control directe d'una variable dins del PLC

```python
import time
import snap7
from snap7.util import set_bool
from snap7.type import Areas

def connect_to_plc(ip="192.168.0.1", rack=0, slot=1):
    client = snap7.client.Client()
    client.connect(ip, rack, slot)

    if client.get_connected():
        print("Connected to PLC!")
    else:
        print("Connection failed")

    return client

def blink_light(client, db_number, byte_index, bit_index, duration=1):
    """
    Enciende y apaga un bit en un DB concreto (parpadeo visible).
    """

    # Leer 1 byte del DB donde está la luz
    data = client.read_area(Areas.DB, db_number, byte_index, 1)

    print(f"[+] Luz en DB{db_number}.BYTE{byte_index}.BIT{bit_index} → ON")
    set_bool(data, 0, bit_index, True)
    client.write_area(Areas.DB, db_number, byte_index, data)

    time.sleep(duration)

    print(f"[-] Luz en DB{db_number}.BYTE{byte_index}.BIT{bit_index} → OFF")
    set_bool(data, 0, bit_index, False)
    client.write_area(Areas.DB, db_number, byte_index, data)


plc = connect_to_plc("10.72.101.68", rack=0, slot=1)
blink_light(plc, db_number=1, byte_index=0, bit_index=0, duration=10)
```

Exploració i control del NS2 (protocol OPC UA)

Un cop validat el parcial funcionament del NS1, vam començar a investigar com interactuar amb el NS2, que segons documentació i deducció, vam arribar a la conclusió de que utilitzava OPC UA. Per comprendre millor l'estructura del servidor OPC UA del PLC vam utilitzar UAExpert, amb el qual ens vam poder autenticar i navegar pels nodes disponibles.

A més, vam desenvolupar un petit script en Python que registrava de forma automàtica tots els nodes i subnodes en un fitxer de text, per ajudar-nos a identificar variables rellevants o valors ocults que poguessin formar part del repte.

```python
from opcua import Client

def browse_node_to_file(node, file, depth=0):
    indent = "  " * (depth * 2)
    browse_name = node.get_browse_name().Name
    node_str = f"{indent}{node}: {browse_name}\n"
    file.write(node_str)
    for child in node.get_children():
        browse_node_to_file(child, file, depth + 1)

url = "opc.tcp://10.72.101.68:4840"
client = Client(url)
client.connect()
try:
    root = client.get_root_node()
    with open("opcua_browse_log.txt", "w", encoding="utf-8") as logfile:
        logfile.write("Browsing OPC UA address space:\n")
        browse_node_to_file(root, logfile)
finally:
    client.disconnect()

print("Browse results logged to opcua_browse_log.txt")
```

Gràcies a aquestes exploracions vam localitzar els nodes relacionats amb el NS2:

- DATA_HACK_NS2.ON_OFF
- DATA_HACK_NS2.TEMPS_ON
- DATA_HACK_NS2.TEMPS_OFF
- DATA_HACK_NS2.NS1_HELP

opcua_browse_log.txt

```
  2   i=84: Root
  3     i=85: Objects
346       ns=2;i=5001: DeviceSet
347         ns=3;s=PLC: PLC_HACK_01
358           ns=3;s=SoftwareRevision: SoftwareRevision
359           ns=3;i=5201: Icon
360           ns=3;s=Counters: Counters
361             ns=3;i=5204: Icon
362           ns=3;s=DataBlocksGlobal: DataBlocksGlobal
363             ns=3;i=5202: Icon
364             ns=3;s="DATA_HACK_NS2": DATA_HACK_NS2
365               ns=3;s="DATA_HACK_NS2"."ON_OFF": ON_OFF
366               ns=3;s="DATA_HACK_NS2"."TEMPS_ON": TEMPS_ON
367               ns=3;s="DATA_HACK_NS2"."TEMPS_OFF": TEMPS_OFF
368               ns=3;s="DATA_HACK_NS2"."NS1_HELP": NS1_HELP
369           ns=3;s=DataBlocksInstance: DataBlocksInstance
370             ns=3;i=5203: Icon
```

A partir d'aquesta informació vam poder implementar fàcilment una funció en Python per controlar l'NS2, incloent-hi l'ajust del seu temporitzador.

```python
NODE_ON_OFF    = 'ns=3;s="DATA_HACK_NS2"."ON_OFF"'
NODE_TEMPS_ON  = 'ns=3;s="DATA_HACK_NS2"."TEMPS_ON"'
NODE_TEMPS_OFF = 'ns=3;s="DATA_HACK_NS2"."TEMPS_OFF"'
HELP_VALUE = 'ns=3;s="DATA_HACK_NS2"."NS1_HELP"'

def opc_write_any(client, nodeid, value):
    node = client.get_node(nodeid)
    t = node.get_data_value().Value.VariantType
    dv = ua.DataValue(ua.Variant(value, t))
    node.set_value(dv)
    print(f"[NS2] {nodeid} = {value} (type OK)")

def ns2_set_timer_and_start(ms_on_time, ms_off_time):
    c = Client(OPC_URL)
    c.connect()
    opc_write_any(c, NODE_TEMPS_ON, ms_on_time)
    opc_write_any(c, NODE_TEMPS_OFF, ms_off_time)
    opc_write_any(c, NODE_ON_OFF, True)
    opc_write_any(c, HELP_VALUE, "1")
    c.disconnect()
    print("[NS2] Timer set + ON_OFF TRUE")
```

Investigació sobre NS3 i pistes inicials errònies

A mesura que avançàvem en el repte ens vam adonar que el comportament de l'NS1 no era del tot correcte. Tot i poder activar-lo, faltava algun paràmetre addicional, concretament el timer, que també havíem de modificar per completar la funcionalitat que el PLC esperava.

Durant aquesta investigació, els logs d'OPC UA també ens van revelar l'existència d'un node anomenat ns=3;s=ActivacionCerraduraNS3, que inicialment vam interpretar com una possible pista per controlar el tercer NS. Vam dedicar-hi un temps provant de modificar aquest valor, però tots els intents acabaven en errors i no produïen cap canvi observable al PLC. Això ens va indicar que aquest node no formava part de la lògica de control real del repte.

Anàlisi de blocs de dades (DB1 i DB2)

Com que aquesta línia d'investigació no donava resultats, vam decidir analitzar més a fons els blocs de dades (DB) disponibles. Mitjançant un script de lectura sistemàtica, vam confirmar que només teníem accés a DB1 i DB2, i que per tant tota la informació rellevant per controlar NS1 i NS2 havia de trobar-se dins d'aquests dos blocs.

```python
import snap7
from snap7.type import Areas

PLC_IP = "10.72.101.68"

c = snap7.client.Client()
c.connect(PLC_IP, 0, 1)

for db in range(1, 20):
    try:
        data = c.read_area(Areas.DB, db, 0, 32)
        print(f"DB{db} successfully read (size 32)")
    except:
        pass

c.disconnect()
```

Posteriorment, vam analitzar el contingut de les dues DB's disponibles amb el següent codi:

```python
import snap7
from snap7.type import Areas

PLC_IP = "10.72.101.68"

c = snap7.client.Client()
c.connect(PLC_IP, 0, 1)

for db in [1, 2]:
    try:
        data = c.read_area(Areas.DB, db, 0, 32)
        print(f"\n=== DB{db} (32 bytes) ===")
        print(data.hex())
    except Exception as e:
        print(f"DB{db} error:", e)

c.disconnect()
```

```
0000  20 00 00 00 00 00 03 e8 75 73 65 72 3a 68 61 63   .......user:hac
0010  6b 20 61 6e 64 20 70 61 73 73 3a 48 61 63 6b 61   k and pass:Hacka
0020  74 68 6f 6e 32 30 32 35 00 00 00 00 00 00 00 00   thon2025........
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0060  00 00 00 00                                       ....
```

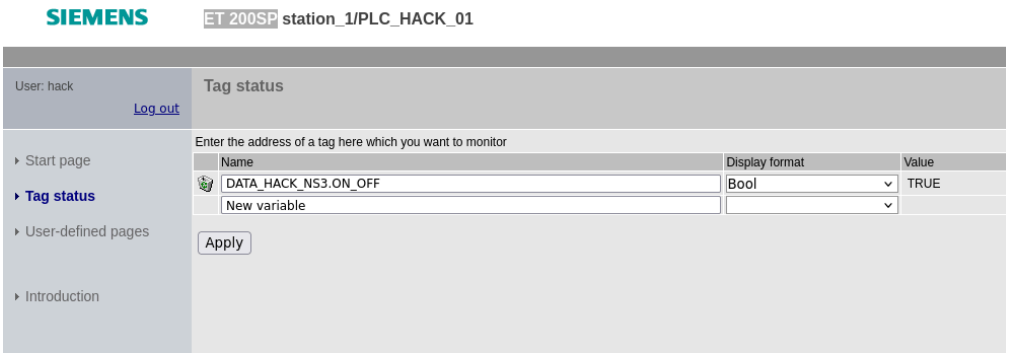
L'anàlisi del contingut en format hexadecimal dels blocs de dades ens va proporcionar una descoberta inesperada: en convertir alguns dels valors, vam identificar una cadena ASCII que contenia credencials d'accés al PLC. En concret:

- **Usuari**: hack
- **Contrasenya**: Hackathon2025

Tot i que en aquell moment no teníem clar quin ús podíem donar a aquestes credencials dins el context del repte, vam documentar aquest fet i vam continuar centrant-nos en la resolució del comportament del NS1, especialment en la configuració dels seus temporitzadors, que encara no sabíem com assignar correctament.

Deducció dels offsets i tipus de dades dels temporitzadors

En paral·lel, aquesta mateixa fase d'anàlisi també ens va donar la pista sobre la variable relacionada amb l'activació del NS3. A partir d'aquí vam començar a provar diferents variables i combinacions per validar si algun d'aquests valors podia correspondre al mecanisme necessari per controlar el tercer senyal. Tot i que inicialment cap de les proves era concloent, aquest procés ens va orientar cap a la direcció correcta per resoldre l'última part del repte.



Sabíem que havíem d'escriure certs valors dins el DB1, però encara no teníem clar quin era el tipus de dada esperat ni la seva posició exacta dins del bloc de memòria. Per intentar deduir-ho, vam començar fent diferents proves pràctiques basant-nos en els hex-dumps i comparant el contingut del DB1 amb el del DB2. Aquesta comparació ens va permetre observar que ambdós blocs compartien una estructura molt similar i que les variables principals seguien un patró d'offsets coincidents.

A partir d'aquesta observació, vam identificar que les tres comandes bàsiques (ON/OFF, temps d'encès i temps d'apagat) es trobaven a les primeres posicions de memòria. Tanmateix, encara quedaven dos reptes importants:

1. Determinar quin era l'offset exacte de cada variable
2. Descobrir la mida i el format (8 bits, 16 bits, etc.) que el PLC esperava per a cada valor.

Vam provar diferents combinacions de mides i offsets, però cap d'elles generava el comportament correcte de manera consistent.

Cerca de la seqüència correcta

Finalment, vam elaborar un petit script que escrivia valors distintius a cada offset per tal d'identificar, de forma sistemàtica, quines posicions de memòria modificaven realment els temporitzadors. Aquest procés va ser el que ens va permetre determinar els offsets correctes.

```
print("Writing test values as 16-bit integers:")
for offset, value in test_values:
    data = struct.pack('>H', value)  # 16-bit big-endian
    c.write_area(Areas.DB, 1, offset, data)
    print(f"Offset {offset}: {value}")

# Read back entire block
read_back = c.read_area(Areas.DB, 1, 0, 30)
print("\nFirst 30 bytes after write:")
for i in range(0, 30, 2):
    word_value = struct.unpack('>H', read_back[i:i+2])[0]
    hex_str = ' '.join(f'{b:02X}' for b in read_back[i:i+2])
    print(f"Offset {i:2d}: {hex_str} = {word_value}")

c.disconnect()
```

Un dels principals problemes que ens vam trobar durant l'anàlisi dels blocs de dades va ser que el text del hexdump contenia un espai addicional al principi, fet que ens va portar a interpretar incorrectament l'estructura de memòria. Inicialment pensàvem que l'espai reservat per als temporitzadors era molt més gran del que realment era; estàvem assignant massa bytes a dues variables que, en realitat, només requerien dos bytes cadascuna.

A conseqüència d'aquesta mala interpretació, quan escrivíem un valor petit en un dels temporitzadors, per exemple un "1", el PLC el llegia com a 256 (0x0100), ja que estàvem desplaçant els bytes fora de la seva posició correcta. Aquest comportament ens va fer sospitar que els timers havien de ser enters de 16 bits i que els offsets reals eren molt més propers a l'inici del bloc.

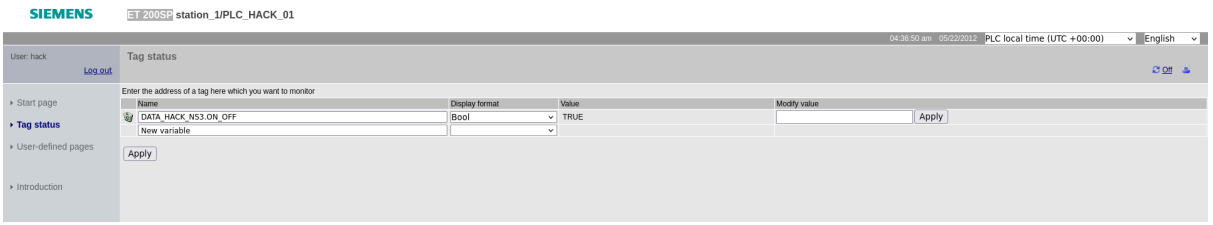
Finalment, i gràcies a les proves sistemàtiques escrivint valors distintius en cada posició, vam arribar a la conclusió que els temporitzadors estaven definits com a valors de 16 bits situats als offsets 2 i 4. Un cop identificades aquestes posicions, vam poder implementar un codi molt senzill que ens permetia escriure qualsevol valor als timers de manera fiable.

A partir d'aquí, vam crear diversos bucles que recorrien totes les possibles combinacions de temps per trobar la configuració correcta. Sabíem que els valors havien de correspondre a intervals inferiors a un minut, així que la cerca no era infinita. Tot i això, en un primer moment la cerca es va alentir perquè pensàvem que els timers no podien ser múltiples de 5 mil·lisegons i vam estar filtrant valors basant-nos en aquesta hipòtesi, que finalment es va demostrar incorrecta.

Com que el repte es resolia segons l'ordre i la rapidesa, vam prioritzar provar combinacions de temps molt curts. No sabíem si el missatge d'enhorabona de la HMI es mantindria o desapareixeria tan bon punt canviéssim la configuració, així que per assegurar-nos de no perdre cap pista vam començar a registrar contínuament l'estat de l'objecte de sortida ActivacionCerraduraNS3, per detectar qualsevol canvi rellevant de manera immediata.

Validació final del repte

Finalment, un altre equip va aconseguir identificar els valors correctes dels temporitzadors abans que nosaltres. A partir d'aquí, vam escriure manualment aquests valors al PLC mitjançant el nostre codi, fet que ens va permetre completar satisfactòriament la seqüència requerida i, en conseqüència, finalitzar el repte amb èxit.



Tots els membres de l'equip hem participat activament en el desenvolupament del codi. No obstant això, per evitar problemes derivats dels canvis constants de xarxa i les desconnexions freqüents, vam optar per realitzar els push al repositori des d'un mateix ordinador. Aquesta metodologia ens va permetre treballar amb més estabilitat i assegurar una gestió coherent del codi al llarg de tot el repte.

FOTO FINAL