Assignment 1

Analysis of Network Delivery

Feb 10th, 2024

Table of Contents

Executive Summary

In my network delivery task, I aimed to summarize the objectives, tools utilized and critical aspects of network security. My exploration began with Burp Suite, a web application security testing platform that acts as an intermediary to intercept, inspect, and modify web traffic. This allowed me to identify vulnerabilities such as SQL injection, XSS, brute force attacks, command injection, stored XSS, and open HTTP redirects in a controlled environment using the Damn Vulnerable Web Application (DVWA).

I tested for various web vulnerabilities by configuring traffic interception and utilizing Burp Suite's tools like Intruder and Repeater. These exercises emphasized the importance of input validation, secure coding practices, and encryption in protecting web applications against potential threats.

Simultaneously, I explored phishing attack simulations using Gophish for simulating phishing campaigns in my controlled environment. I gained insights into the effectiveness of phishing techniques and the importance of security awareness training to prevent breaches by setting up campaigns targeting Instagram and Spotify credentials.

The case study on the July 2020 Twitter spear-phishing attack provided a real-world example of the sophistication of social engineering tactics. It underscored the necessity of employee training and robust authentication mechanisms to safeguard against such threats.

Comparing the financial impact of phishing attacks to the investment in preventive cybersecurity measures highlighted the cost-effectiveness of implementing security awareness training, advanced security systems, and regular security audits. Despite the initial investment, these measures significantly reduce the risk and impact of cyberattacks, offering long-term savings and risk mitigation.

In summary, my exploration of Burp Suite and Gophish, the analysis of real-world attacks and the cost comparison of cybersecurity measures covered the critical need for comprehensive security strategies and the significance of robust security tools culture of security awareness to protect against diverse cyber threats.

Introduction

During this network delivery endeavour, I aimed to investigate web application vulnerabilities and simulate phishing attack campaigns. This exploration encompassed security vulnerabilities, including SQL injection, Cross-Site Scripting (XSS), brute force attacks, command injection, stored XSS, and open HTTP redirects. Furthermore, I conducted phishing campaign simulations to evaluate the resilience of email security against malicious attacks.

The significance of this attempt is an exploration of the various threats faced by web applications and an emphasis on the critical need for stringent security protocols. This assignment shows the role of human factors in cybersecurity and how enhanced awareness and comprehensive training can mitigate the risk of successful cyber breaches.

Moreover, the case study on the Twitter spear-phishing attack and a comparative analysis of the costs associated with mitigating cybersecurity threats has enhanced this endeavour. It has helped me understand cybersecurity challenges and the cost-effectiveness of proactive security measures.

In the end, all testing and simulations were conducted within my framework. This adherence to ethical standards was essential for maintaining the integrity of the Assignment and ensuring the security and privacy of any data or systems involved.

Body

Burp Suite Exploration: I used Burp Suite to intercept web traffic and test for vulnerabilities like SQL injection and XSS in a controlled environment. This involved setting up the environment, adjusting browser settings, and using features like Intruder and Repeater to simulate attacks.

Gophish Exercise: With Gophish, I created fake email campaigns mimicking Instagram and Spotify to test revealing credentials. I monitored interactions through Gophish's dashboard to measure the effectiveness of these simulations.

Case Study: Twitter Spear-Phishing Attack In 2020, Twitter was hit by a spear-phishing attack that compromised high-profile accounts for a Bitcoin scam. This highlighted social engineering risks and the importance of security awareness and multi-factor authentication.

Mitigation Cost Comparison The cost of data breaches and phishing attacks is increasing, emphasizing the need for preventive measures like security training and regular audits. Investing in cybersecurity is more cost-effective than managing the aftermath of an attack.

Burp Suite Exploration

Burp Suite is a comprehensive platform for web application security testing. It acts as an intermediary between a user's browser and the web server, allowing it to intercept, inspect, and modify the traffic passing through it. This is essential for identifying and exploiting web vulnerabilities.

I begin with configuring my browser to route traffic through Burp's proxy listener for capturing the application's traffic. I activate the proxy to start intercepting requests and responses. This lets me inspect headers, parameters, cookies, and payloads in detail.

I utilize the tools to test vulnerabilities. The Target tool helps organize and scope the testing area, identifying common vulnerabilities such as SQL injection, XSS, etc.

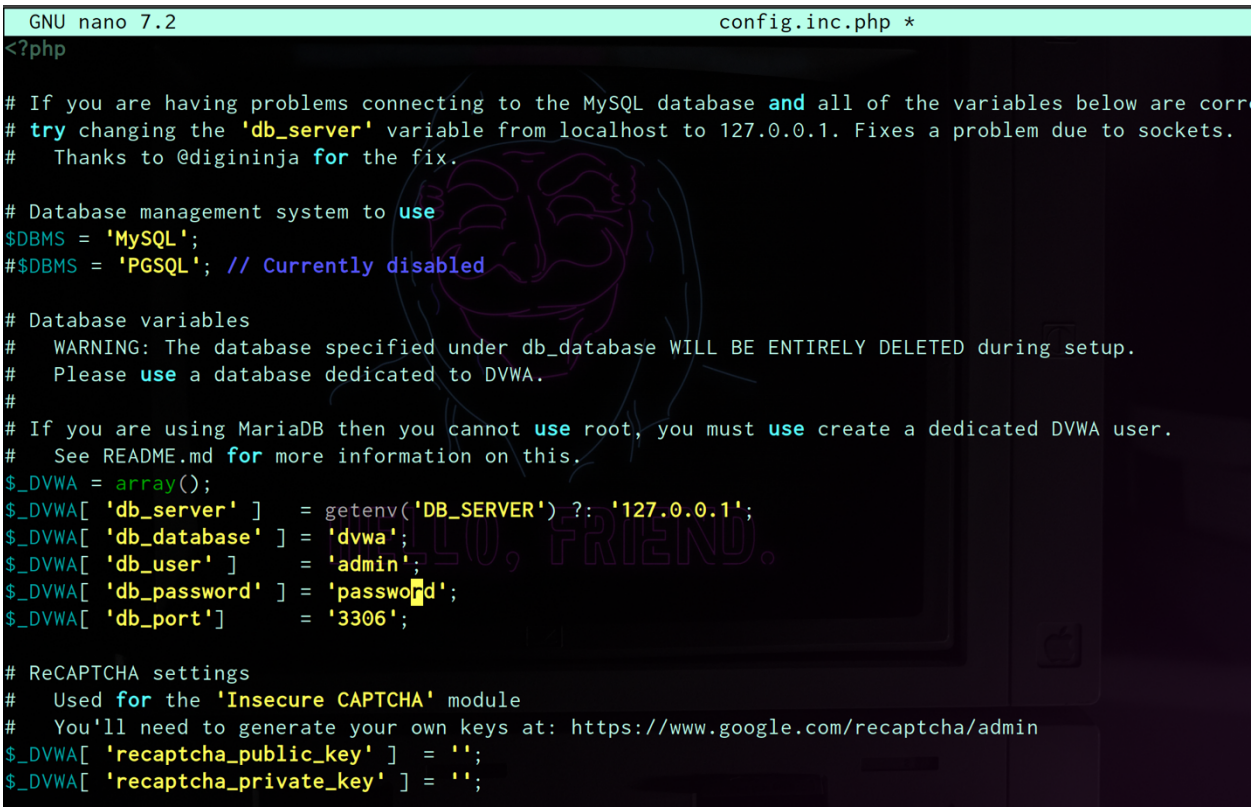
For exploitation and manual testing, I used the Intruder and Repeater tools. I perform attacks such as brute force or custom payload injections against identified vulnerabilities, configuring it to automate requests with varying inputs with Intruder. The Repeater tool allows me to manually modify and resend requests, enabling a detailed analysis of the application's response to manipulate inputs.

Initial Setup

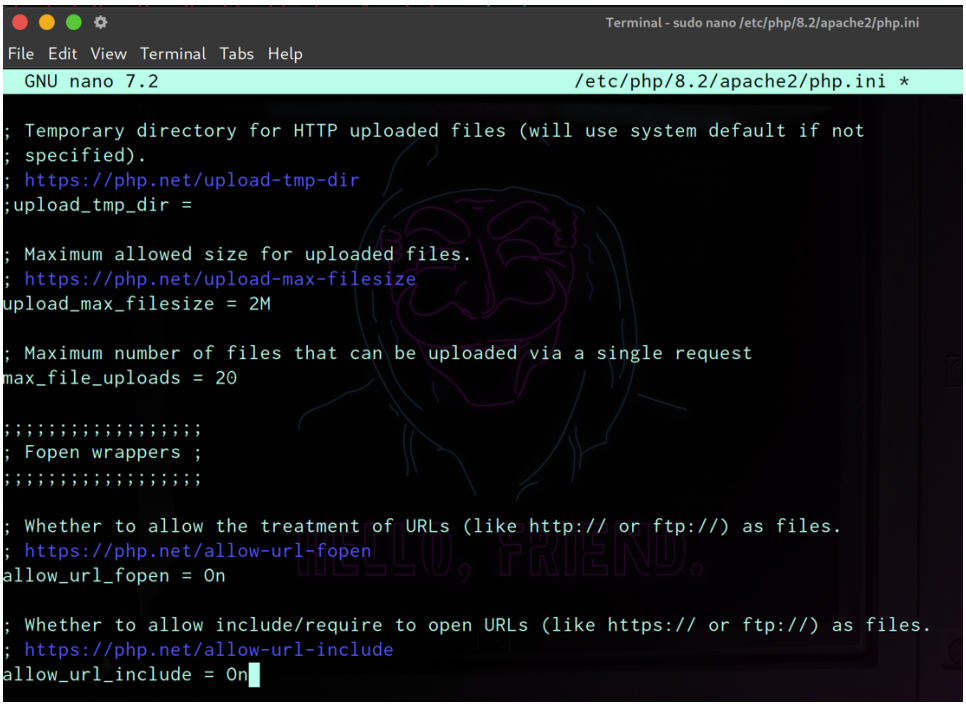
I began by installing and navigating the DVWA configuration directory using the command cd DVWA/config.

I navigated to the root of the web server's directory with cd /var/www/html. After, I moved into the DVWA configuration directory using cd DVWA/config.

After reaching the correct directory. The .dist indicates a distribution or template file, which is provided as an example or starting point. I copied this template to create the configuration file DVWA will use by executing cp config.inc.php.dist config.inc.php.

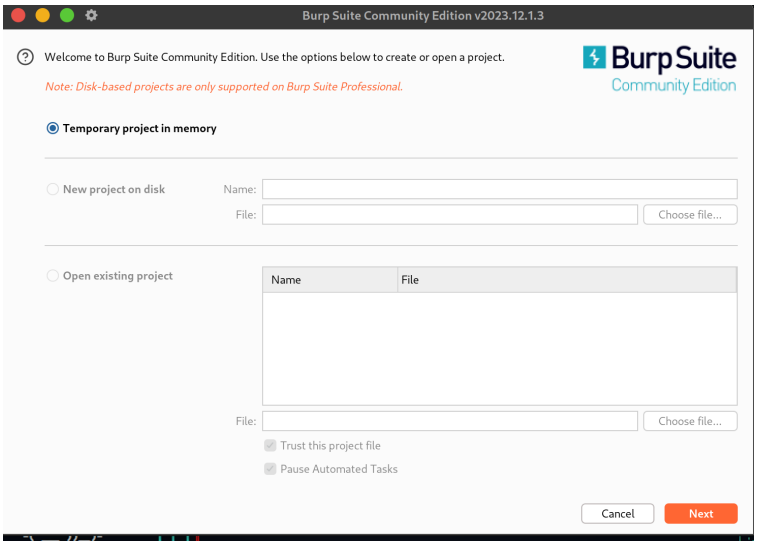


I opened the config.inc.php file in the nano text editor to edit the database connection settings. I modified the username 'admin', and the password 'password' because DVWA is designed to be vulnerable for educational purposes.

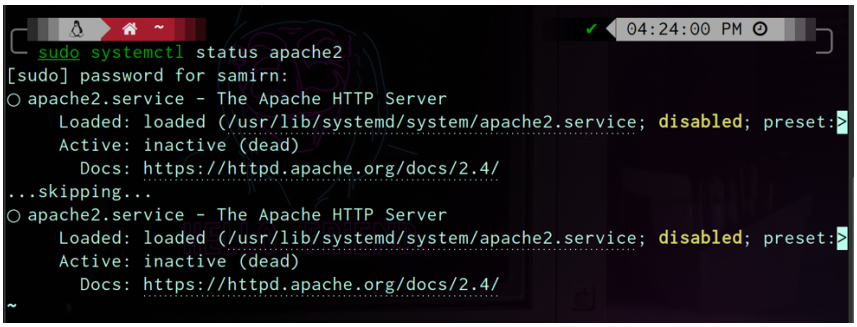


I edited the PHP configuration file php.ini for the Apache server to ensure that certain settings were enabled for DVWA to function correctly. I verified that allow_url_fopen and allow_url_include were set to 'On' for testing certain vulnerabilities where PHP might fetch or include code from remote locations.

Throughout this process, I documented each step and took screenshots as evidence of my actions to demonstrate a clear understanding of the setup process and its significance in preparing for vulnerability testing with Burp Suite.

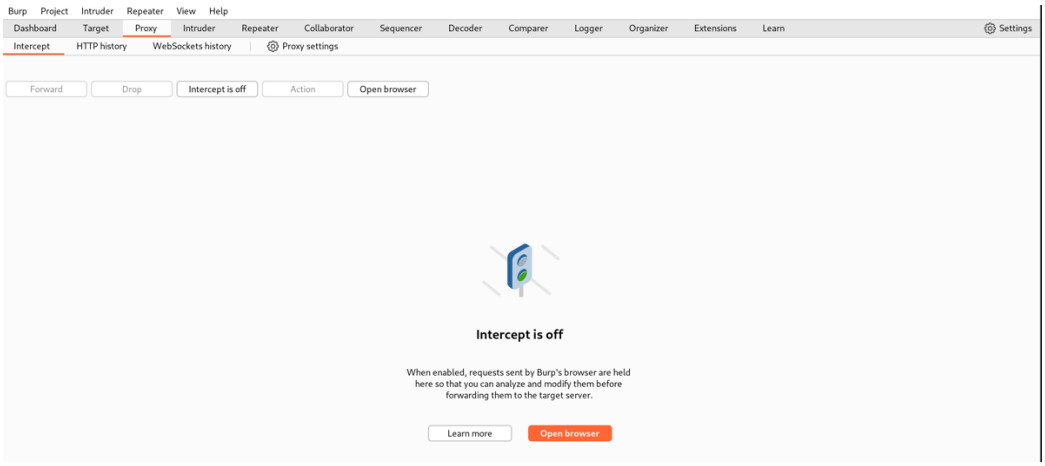


Now, we are ready to run Burpsuite. After running it, I was prompted with the project setup and for the community edition, the only available option is “Temporary project in memory,” which suits the assignment.

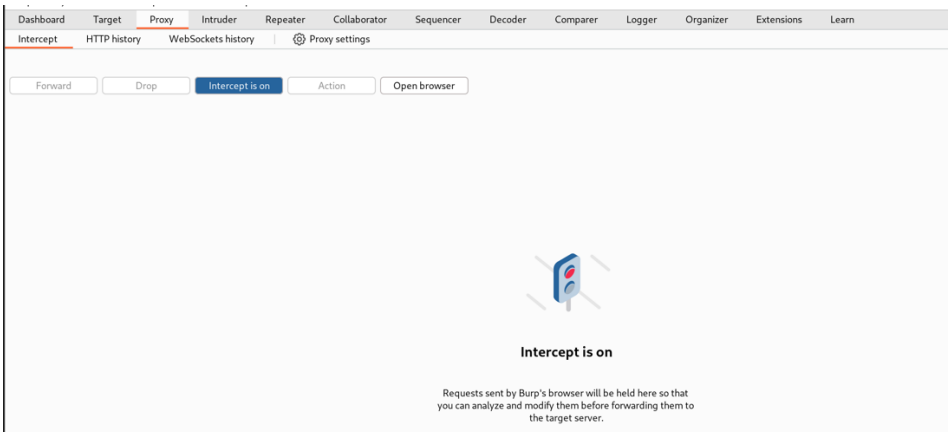


Before moving forward, I checked the status of the apache2 and realized it is not running.

I ran the “sudo systemctl start apache2” to run it and be able to run the DVWA on my local host.

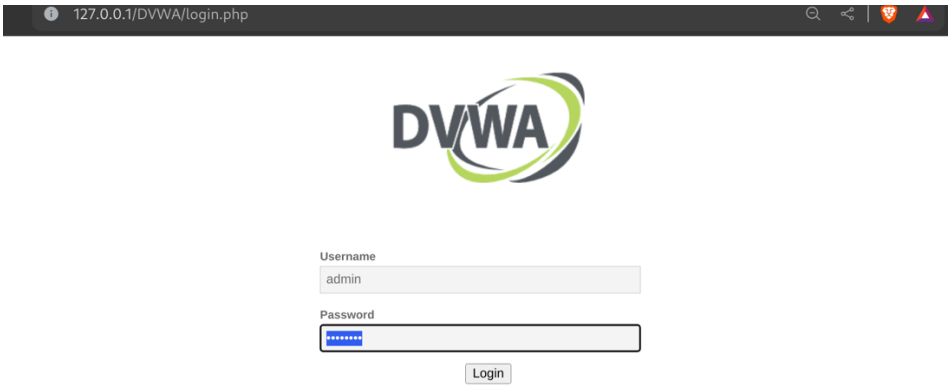


Going back to burpsuite, I navigated to the proxy tab and realized the interception mode is off.

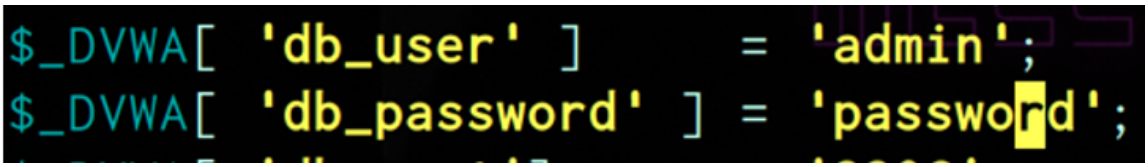


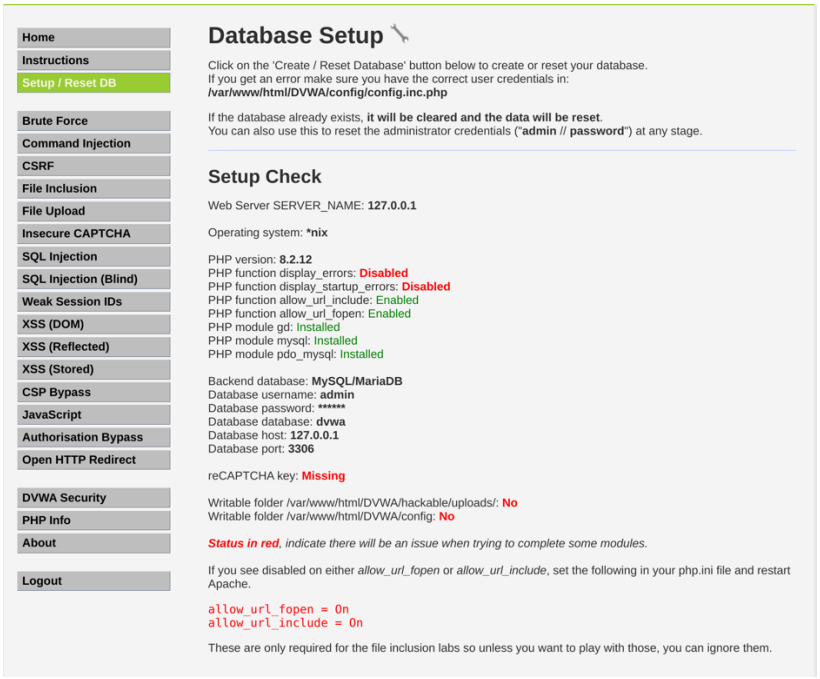
I turned it on by clicking on the “Intercept is off” button.

After, I clicked on the open browser option to access the Chromium browser.

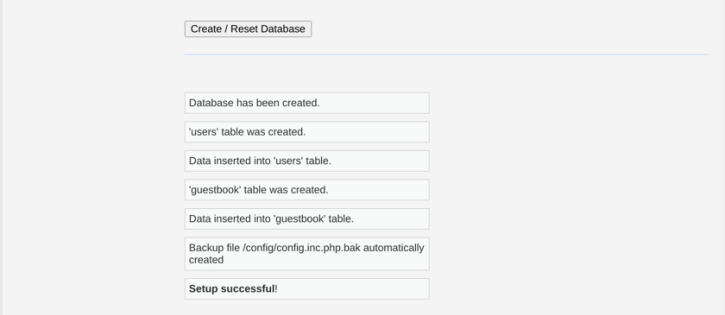


I browsed to where my apache runs DVWA , “127.0.0.1/DVWA/login.php”. I logged in with the username and password that I set during the configuration of DVWA.





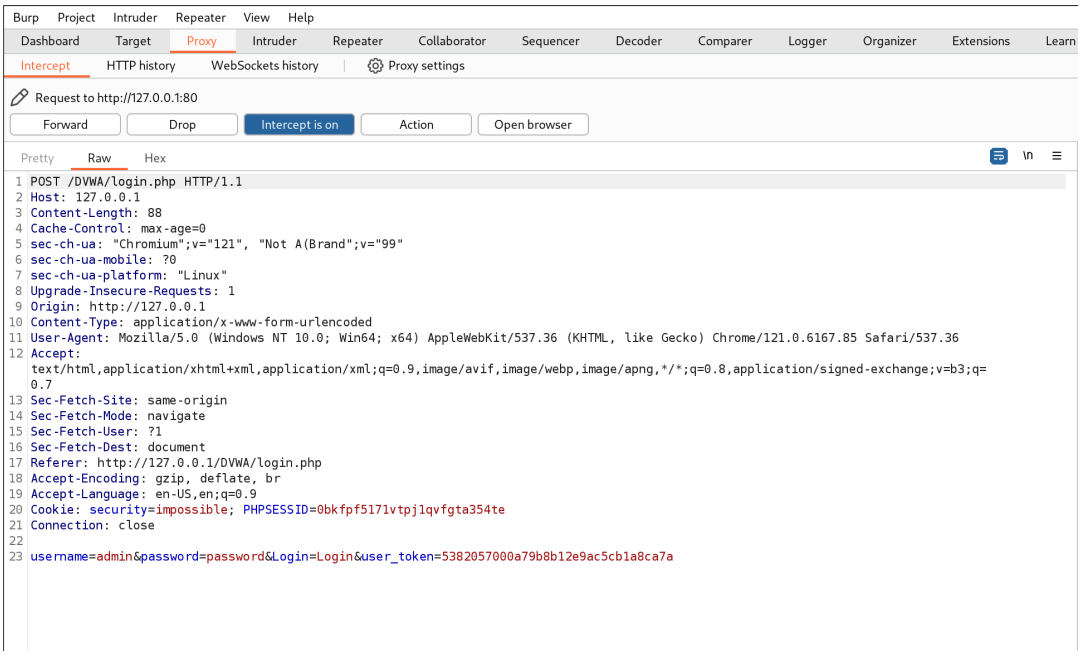
I navigated to the Setup/Reset DB tab and to check my setup,



Then click on “Create / Reset Database to ensure that the database is set up successfully. After this window, we will get back to the login page and has to login again.

Brute Force Attack

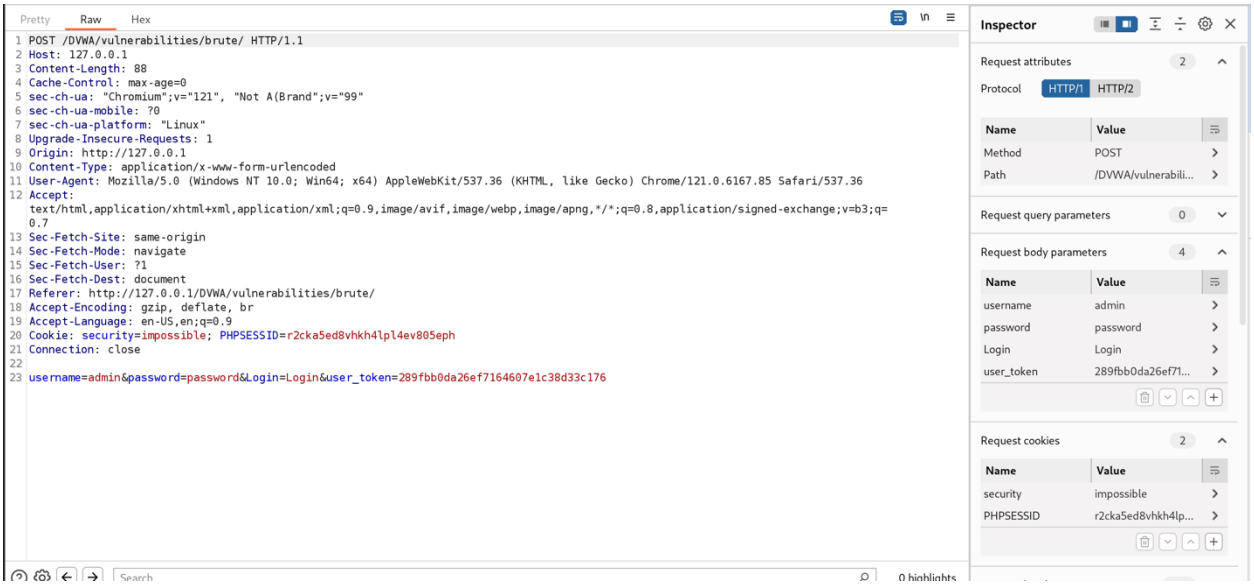
A Brute Force Attack is a trial-and-error method I used to find information such as a password or Personal Identification Number (PIN). By systematically checking all possible combinations, I attempted to guess the correct input to gain unauthorized access to a system. This type of attack exploits the weakness of simple passwords and emphasizes the necessity of implementing strong, complex passwords and account lockout policies.



Furthermore, I started by targeting the login mechanism of DVWA. This allowed me to capture a POST request when logging in to the application.

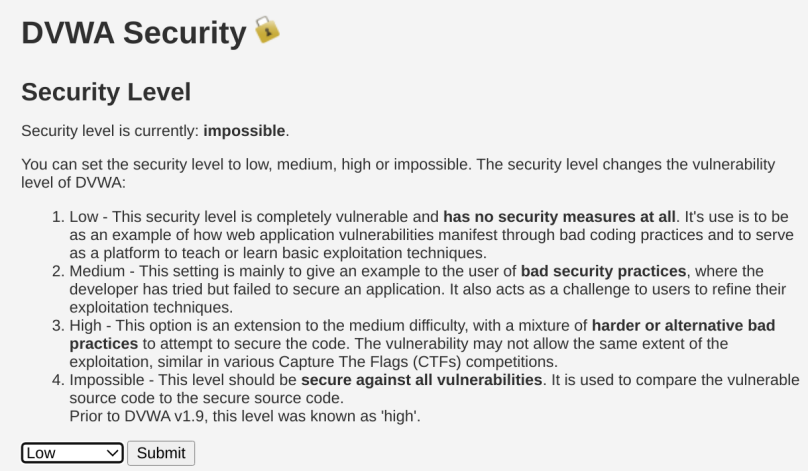
The captured request shows the raw HTTP request generated when submitting the login form with the username 'admin' and password 'password', the configured credentials for DVWA. The request includes HTTP headers such as User-Agent, specifying the browser's details, and Cookie containing session data. It also includes the form parameters in the body of the request, clearly displaying the credentials.

I used this intercepted request to confirm the application is vulnerability to simple credential interception. At this security risk, an attacker could capture sensitive information if the traffic is not encrypted using HTTPS.



I captured another POST request, this time to the URL endpoint responsible for brute force vulnerability within DVWA (/vulnerabilities/brute/). The request format was similar to the login attempt but targeted a different application functionality designed to demonstrate brute force attacks.

I set up a brute force attack by running the intercepted request against a list of common passwords, systematically changing the password parameter while keeping the username constant. The aim was to find the correct password for the 'admin' user by automating the submission of many login attempts.



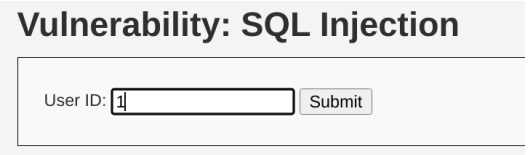
I interacted with the DVWA Security page, which allowed me to change the application's security level. The security levels available are low, medium, high, and impossible. Each level configures the application to be susceptible to different degrees of vulnerabilities:

- Low: No security measures are implemented, making the application fully vulnerable.
- Medium: Some security measures are in place but are not strong enough.
- High: More advanced security measures require more sophisticated methods to exploit.
- Impossible: The application is secured against all known vulnerabilities.

For testing and learning, I started with setting the security level to easy.

SQL Injection

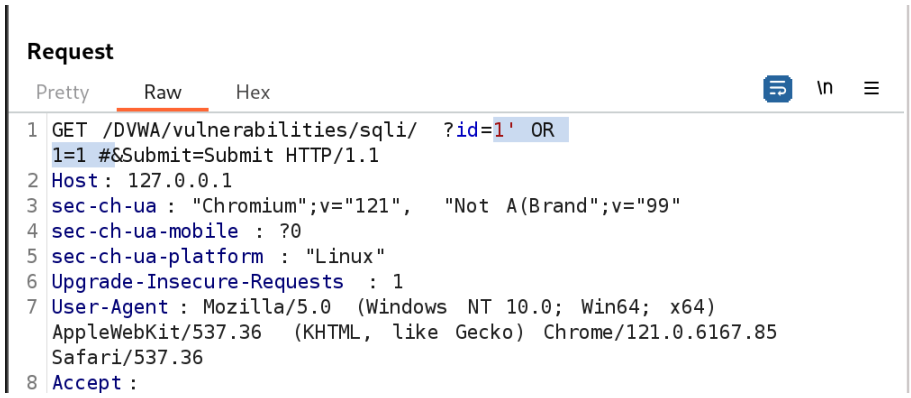
An SQL Injection vulnerability allows insertion or tampering with database queries by manipulating the application's input fields. By inputting special SQL commands, I could alter the query to retrieve additional data or perform unauthorized actions. This can compromise database integrity and confidentiality, highlighting the need for robust input validation and prepared statements to prevent SQL Injection attacks.



I focused on the SQL Injection vulnerability page of DVWA. The input field "User ID" accepts a numeric value used in a backend SQL query to retrieve user details.

```
GET /DVWA/vulnerabilities/sqli/?id=1&Submit=Submit&user_token=e53a5ae94674f083c67969d0812161f9 HTTP/1.1
```

Right clicked on the Get HTTP request and sent it to repeater therefore the result will not show up on the web page.



I crafted a SQL injection payload by entering 1' OR '1'='1 in the "User ID" input field. This payload is designed to alter the SQL query logic always to return true, thereby bypassing the intended control of selecting a single user and instead retrieving all users.

Right Click -> Convert Selection -> URL -> URL-encode Key characters.

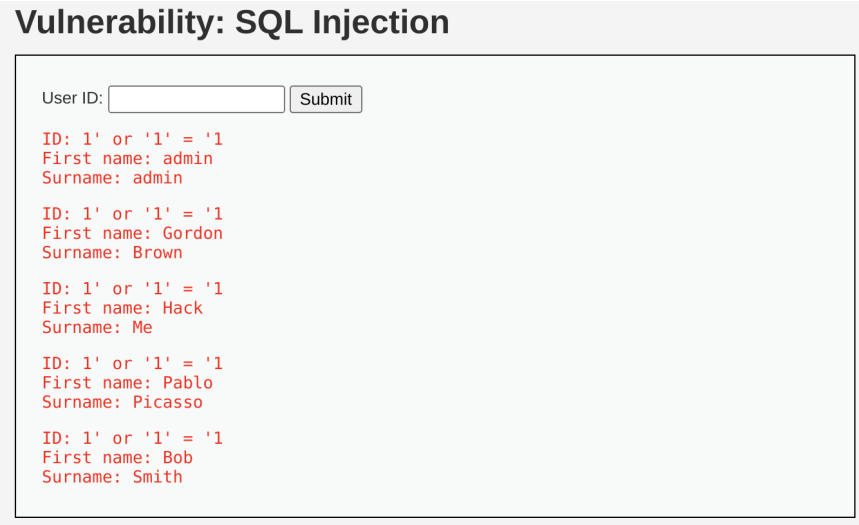
```
GET /DVWA/vulnerabilities/sqli/ ?id=1'+OR+1%3d1+%23 &Submit=Submit&
user_token=e53a5ae94674f083c67969d0812161f9 HTTP/1.1
```

I captured the HTTP request generated by the form submission using Burp Suite. The request URL included my injection in the id parameter, confirming that the payload was sent to the server.

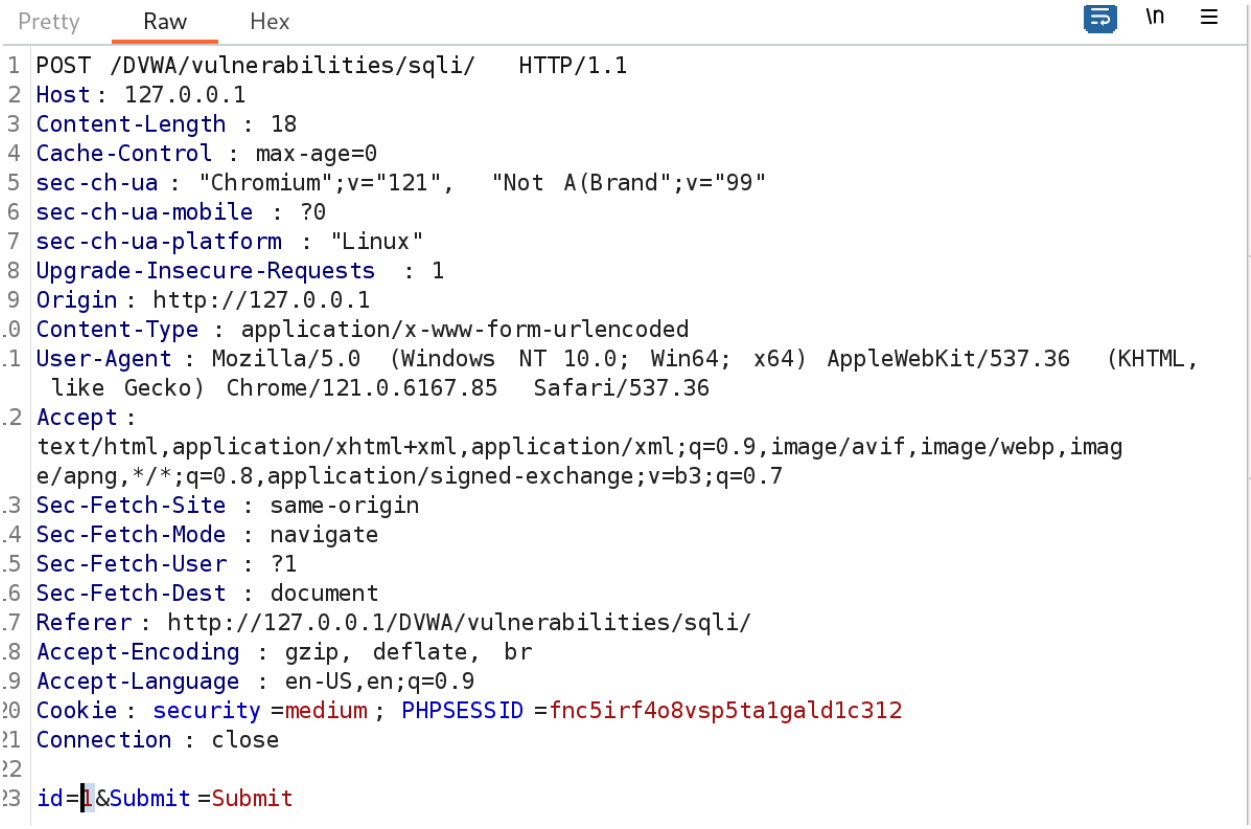
```
</form>
<pre>
  ID: 1' OR 1=1 #<br />
  First name: admin<br />
  Surname: admin
</pre>
<pre>
  ID: 1' OR 1=1 #<br />
  First name: Gordon<br />
  Surname: Brown
</pre>
<pre>
  ID: 1' OR 1=1 #<br />
  First name: Hack<br />
  Surname: Me
</pre>
<pre>
  ID: 1' OR 1=1 #<br />
  First name: Pablo<br />
  Surname: Picasso
</pre>
<pre>
  ID: 1' OR 1=1 #<br />
  First name: Bob<br />
  Surname: Smith
</pre>
</div>
```

the result of the SQL injection is displayed on the web page. The payload 1' OR '1'='1 modified the SQL query to return all records from the users' table, as evidenced by the list of all user IDs,

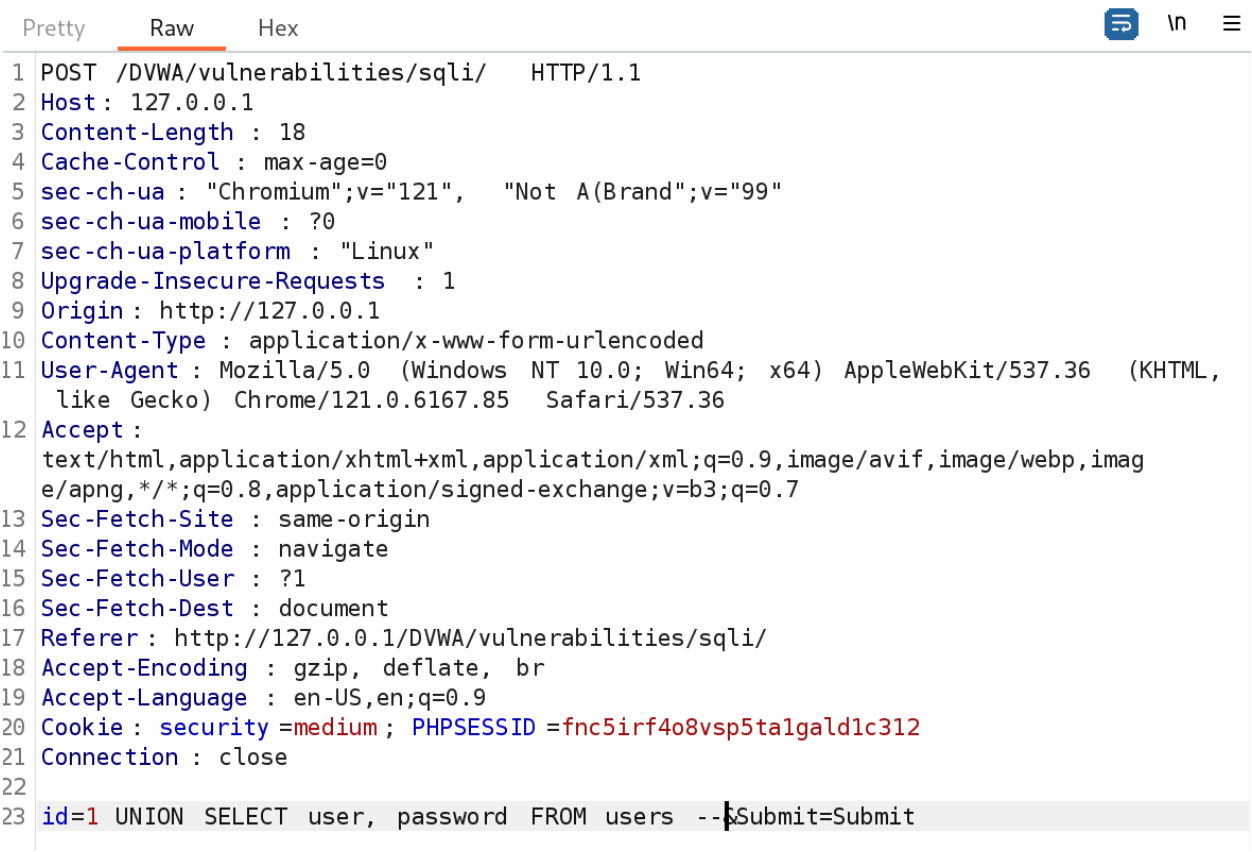
first names, and surnames. This confirmed the application's vulnerability to SQL injection at the low-security level, as it did not properly sanitize user input to prevent such attacks.



Moving on, I submitted the payload 1' or '1' = '1 inside the webpage and manipulated the SQL query executed by the application. This payload is a common test to check if an application is vulnerable to SQL injection, as it always evaluates to true and could potentially return more data than intended.

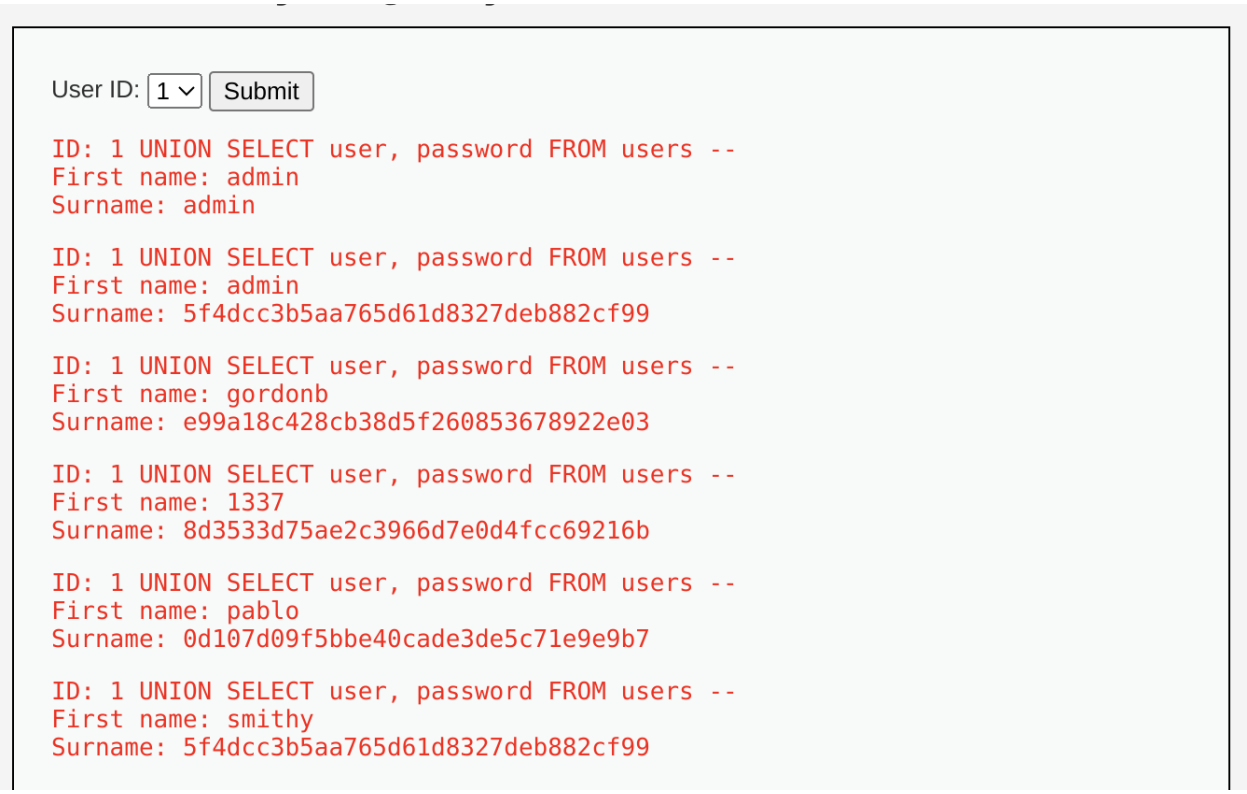


Again, I escalated the Security to medium and intercepted the request made by the form submission, which is visible in the raw HTTP request in Burp Suite. The request shows that the payload was included in the id parameter. This confirmed that the application's input is not correctly sanitized, allowing SQL injection.



Therefore, I executed a more advanced SQL injection attack using the UNION SELECT statement. This allowed me to retrieve information from other tables within the database, effectively bypassing the intended functionality of the form.

By submitting 1 UNION SELECT user, password FROM users --, I instructed the SQL database to combine the results of two SELECT statements. The first one is the original query that the application performs, and the second is my injection, which retrieves usernames and passwords from the 'users' table.



I could see usernames and hashed passwords listed, indicating a successful SQL injection attack. These exercises gave me hands-on experience in understanding and executing different SQL injection web application attacks.

Reflected XSS Attack

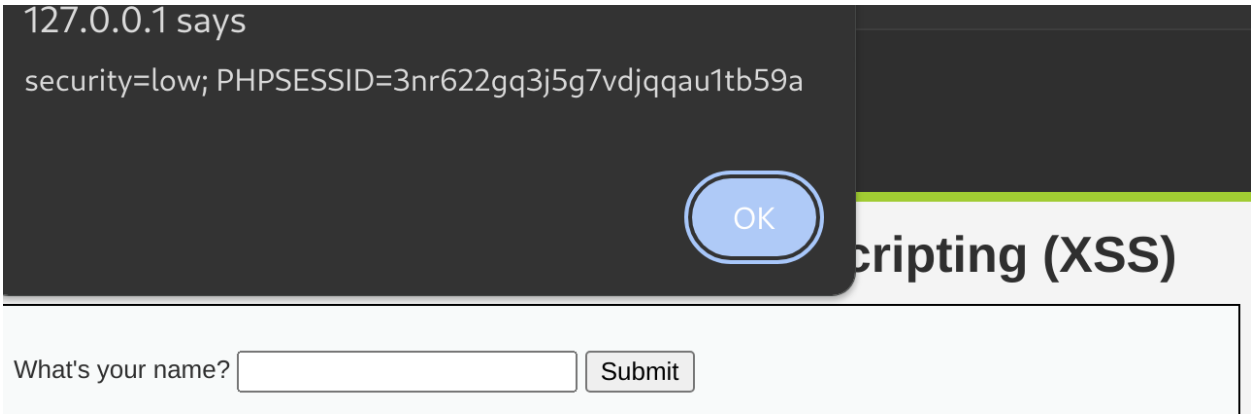
A Reflected XSS attack occurs when an application includes user-supplied data in its immediate response without proper validation or escaping. I noticed that by submitting crafted input with malicious JavaScript, the script executed in my browser reflected the attack vector. It can lead to data theft or malicious actions if the user interacts with a crafted link or form. Ensuring input sanitization is key to defending against such attacks.

<img src=”#” onclick=alert(document.cookie) >

I crafted a malicious HTML snippet: <img src="#" onclick=alert(document.cookie)>. This code triggers a JavaScript alert displaying the document's cookies when the user clicks the image. The src="#" attribute is a placeholder, creating an image that does nothing until clicked.

```
GET /DVWA/vulnerabilities/xss_r/?name=
%3Cimg+src%3D%E2%80%9D%23%E2%80%9D+onclick%3Dalert%28document.cookie%29+%3E
HTTP/1.1
Host: 127.0.0.1
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="121", "Not A(Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/121.0.6167.85 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
http://127.0.0.1/DVWA/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%E2%80%9CY
ou%E2%80%99ve+been+XSSed%21%E2%80%9D%29%3C%2Fscript%3E
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: security=low; PHPSESSID=mcjda3miu10ttg4sf6ijph8e4d
Connection: close
```

I set the security to low and injected this snippet into a vulnerable parameter in a GET request to the DVWA application. The request URL was encoded to pass the malicious script through the 'name' parameter: /vulnerabilities/xss_r/?name=%3Cimg+src%3D%22%23%22+onclick%3Dalert%28document.cookie%29%3E. The server processed this request and reflected my script in the response, which is the behavior exploited in a reflected XSS attack.



The result of this attack illustrates that when the vulnerable page loaded, the browser executed my script because the input was not properly sanitized. As expected, a JavaScript alert box popped up, displaying the cookie information for the session, confirming the vulnerability to reflected XSS.

Command Injection Attack

A Command Injection attack is a security vulnerability that I identified where an application executes arbitrary commands on the server due to insufficient input validation. I manipulated the application to run unintended system commands by crafting special inputs. This can lead to unauthorized access or control over the server, making it crucial to sanitize user inputs to prevent such attacks properly.

Pretty Raw Hex \n

```
1  POST /DVWA/vulnerabilities/exec/ HTTP/1.1
2  Host: 127.0.0.1
3  Content-Length: 26
4  Cache-Control: max-age=0
5  sec-ch-ua: "Chromium";v="121", "Not A(Brand";v="99"
6  sec-ch-ua-mobile: ?0
7  sec-ch-ua-platform: "Linux"
8  Upgrade-Insecure-Requests: 1
9  Origin: http://127.0.0.1
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
   like Gecko) Chrome/121.0.6167.85 Safari/537.36
12 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
   e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://127.0.0.1/DVWA/vulnerabilities/exec/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: security=low; PHPSESSID=mcjda3miu10ttg4sf6ijph8e4d
21 Connection: close
22
23 ip=127.0.0.1 &Submit =Submit
```

Moreover, I targeted a command injection. The form on the web page was designed to take an IP address and ping that address from the server. I entered the input 127.0.0.1 and intercepted the corresponding POST request with Burp Suite.

Pretty Raw Hex

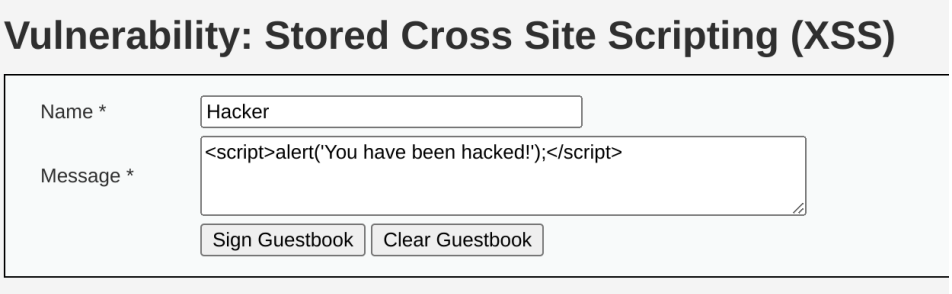
```
POST /DVWA/vulnerabilities/exec/ HTTP/1.1
Host: 127.0.0.1
Content-Length : 26
Cache-Control : max-age=0
sec-ch-ua : "Chromium";v="121", "Not A(Brand";v="99"
sec-ch-ua-mobile : ?0
sec-ch-ua-platform : "Linux"
Upgrade-Insecure-Requests : 1
Origin : http://127.0.0.1
Content-Type : application/x-www-form-urlencoded
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/121.0.6167.85 Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : navigate
Sec-Fetch-User : ?1
Sec-Fetch-Dest : document
Referer : http://127.0.0.1/DVWA/vulnerabilities/exec/
Accept-Encoding : gzip, deflate, br
Accept-Language : en-US,en;q=0.9
Cookie : security =low; PHPSESSID =mcjda3miu10ttg4sf6ijph8e4d
Connection : close

ip=|ls&Submit=Submit
```

I modified the intercepted request by the ls command: ip= |ls. This would instruct the server to list the directory contents after executing the ping command, assuming the input was directly inserted into a shell command.

The result of this command injection shows the success of the attack. The server executes the ls command and reveals sensitive information about the server's file system.

Stored XSS Attack

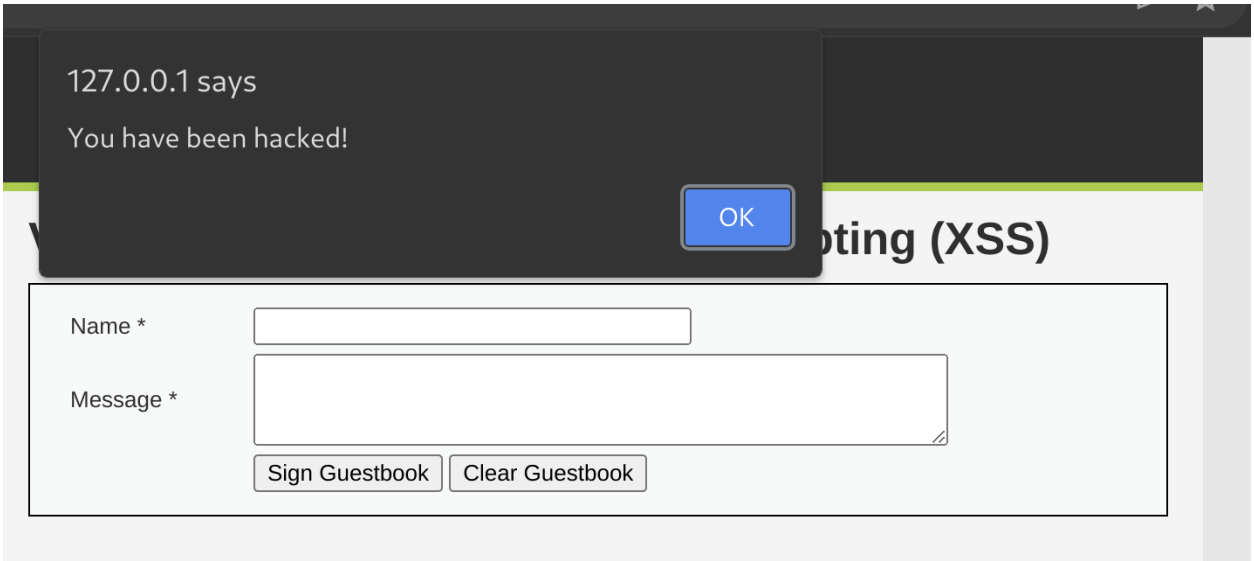
Stored Cross-Site Scripting (XSS) is a vulnerability where an application incorrectly sanitizes user input that gets saved and displayed to other users. I found that by injecting malicious scripts into the application, these scripts are executed in other users' browsers without their knowledge. This poses a significant security risk as it can lead to unauthorized access to user sessions or data. It's essential to sanitize and validate user inputs to prevent Stored XSS attacks.



I moved on to a stored XSS attack, which is more insidious than reflected XSS because the injected script is stored on the server and executed for every user who accesses the affected page. I filled out a guestbook form with a name 'Hacker' and a message containing the script: <script>alert('You have been hacked!');</script>.

```
Pretty  Raw  Hex                                                    \n  ≡
1  POST /DVWA/vulnerabilities/xss_s/  HTTP/1.1
2  Host: 127.0.0.1
3  Content-Length : 119
4  Cache-Control : max-age=0
5  sec-ch-ua : "Chromium";v="121",  "Not A(Brand";v="99"
6  sec-ch-ua-mobile : ?0
7  sec-ch-ua-platform : "Linux"
8  Upgrade-Insecure-Requests : 1
9  Origin : http://127.0.0.1
10 Content-Type : application/x-www-form-urlencoded
11 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
   like Gecko) Chrome/121.0.6167.85 Safari/537.36
12 Accept :
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
   e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
13 Sec-Fetch-Site : same-origin
14 Sec-Fetch-Mode : navigate
15 Sec-Fetch-User : ?1
16 Sec-Fetch-Dest : document
17 Referer : http://127.0.0.1/DVWA/vulnerabilities/xss_s/
18 Accept-Encoding : gzip, deflate, br
19 Accept-Language : en-US,en;q=0.9
20 Cookie : security =low; PHPSESSID =mcjda3miu10ttg4sf6ijph8e4d
21 Connection : close
22
23 txtName =Hacker &mtxMessage =
   %3Cscript%3Ealert%28%27You+have+been+hacked%21%27%29%3B%3C%2Fscript%3E      &btnSign =
   Sign+Guestbook
```

I intercepted the form submission with Burp Suite. The raw HTTP request showed the full POST request that included my malicious script in the 'txtMessage' parameter: txtName=Hacker&txtMessage=%3Cscript%3Ealert%28%27You+have+been+hacked%21%27%29%3B%3C%2Fscript%3E.

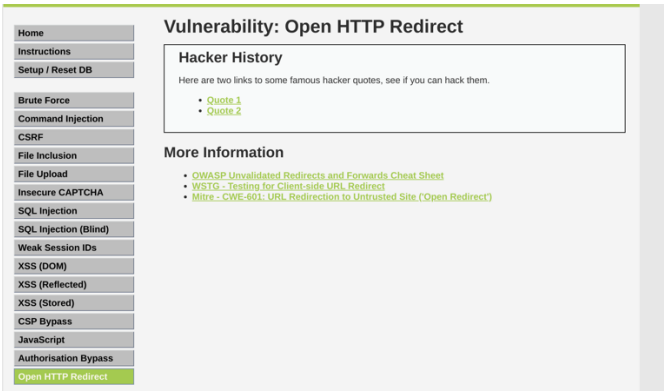


the browser executed the stored script, and the alert box was displayed with the message "You have been hacked!". This confirmed that the application was vulnerable to stored XSS, where the malicious script is saved in the server's database and rendered in the HTML page for future visitors.

These attacks demonstrate the importance of proper input validation and output encoding to prevent malicious scripts from being executed in the context of the web page, which can lead to cookie theft, session hijacking, and other malicious activities.

Open HTTP Redirect

The Open HTTP Redirect vulnerability is a security flaw where an application takes a user-supplied input and uses it to redirect the user to a specified URL without properly validating that URL. Attackers can exploit this to redirect users to phishing or malicious websites, thereby compromising their security.

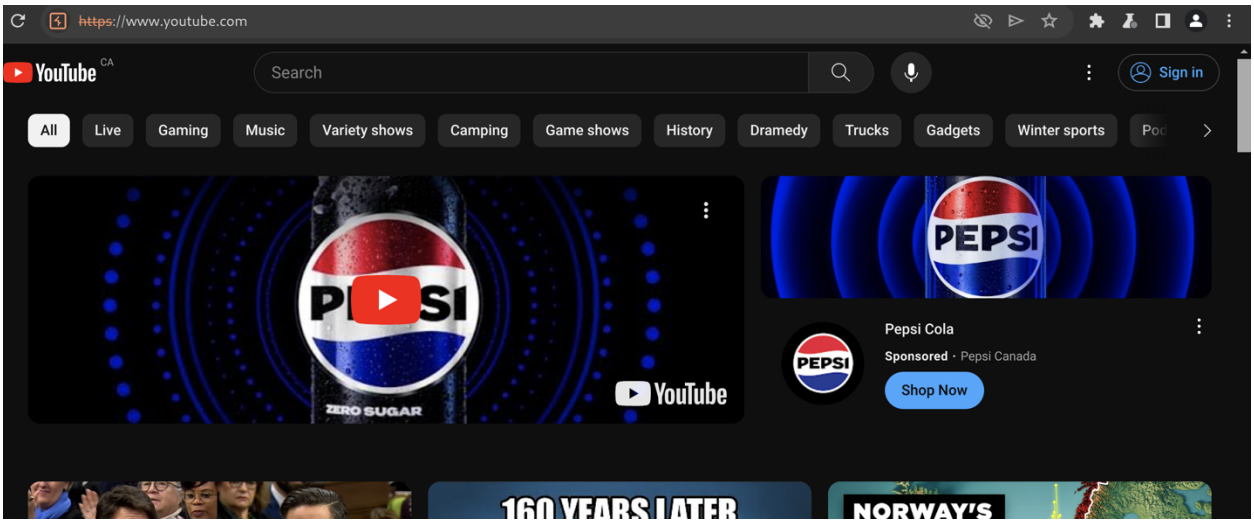


DVWA presents an example that redirects users to quotes from famous hackers. This indicates a feature that you might find on a real website, where redirection to external resources is provided for user convenience. However, without proper security measures, this feature can be

manipulated.

I used Burp Suite to intercept the network request triggered by clicking one of the provided links. This interception allowed me to examine and modify the details of the HTTP request sent by the browser to the server. I noticed that the **redirect** parameter within the URL is where the application expects the redirect's destination.

Moreover, I altered the value of the **redirect** parameter to **http://youtube.com** instead of the intended destination. This modification was a test to see if DVWA would redirect the user to any URL provided in the **redirect** parameter, which should not happen in a secure application.



The browser was redirected to **youtube.com**, an external website not originally intended by the application's functionality. This result demonstrates that DVWA did not validate the URL in the **redirect** parameter, making it susceptible to an Open Redirect vulnerability.
The success of this manipulation implies that an attacker could craft malicious URLs that appear legitimate and redirect the user to an attacker-controlled website. This can lead to various security threats, such as credential theft, installation of malware, or other phishing-related exploits.
This attack illustrates the danger of Open Redirect vulnerabilities and the ease with which they can be exploited. We must Implement robust input validation and URL sanitization measures in web applications to prevent such vulnerabilities from being exploited in real-world scenarios.

Overall, the exploration with Burp Suite provided deep insights into identifying and exploiting vulnerabilities. By configuring traffic interception and utilizing tools like Intruder and Repeater, I tested for SQL injection, XSS, brute force, command injection, stored XSS, and open HTTP redirects within DVWA. This part of the assignment highlighted the importance of input validation, secure coding practices, and the necessity of using comprehensive security tools like Burp Suite to safeguard web applications against potential threats.

Gophish Exercise

Gophish is an open-source phishing toolkit designed for penetration testers to test the effectiveness of their email security measures. It allows me to simulate phishing attacks in a controlled environment. By setting up mock campaigns, I can gain insights into how users respond to phishing attempts, which helps in developing better security protocols and training programs to prevent actual breaches. Gophish provides real-time results, enabling me to track the success rate of these simulated attacks and analyze user behavior. It's a practical approach to improving email security awareness within an organization.

Initial Setup

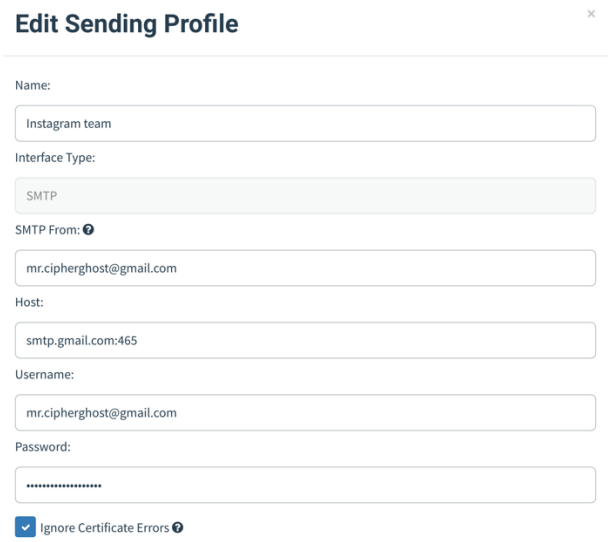
For the setup, I cloned the https://github.com/gophish/gophish.git navigated to the directory and ran the command “go build” and after “gophish”.
After running the command Gophish will run on the localhost, port 3333.

```
time="2024-02-10T14:25:40-08:00" level=info msg="Please login with the username admin and the password 314
22c6414996451"
```

I started by logging into the Gophish administrative panel I utilized the provided login credentials—username "admin" and an alphanumeric password—as instructed by the message displayed on the terminal.

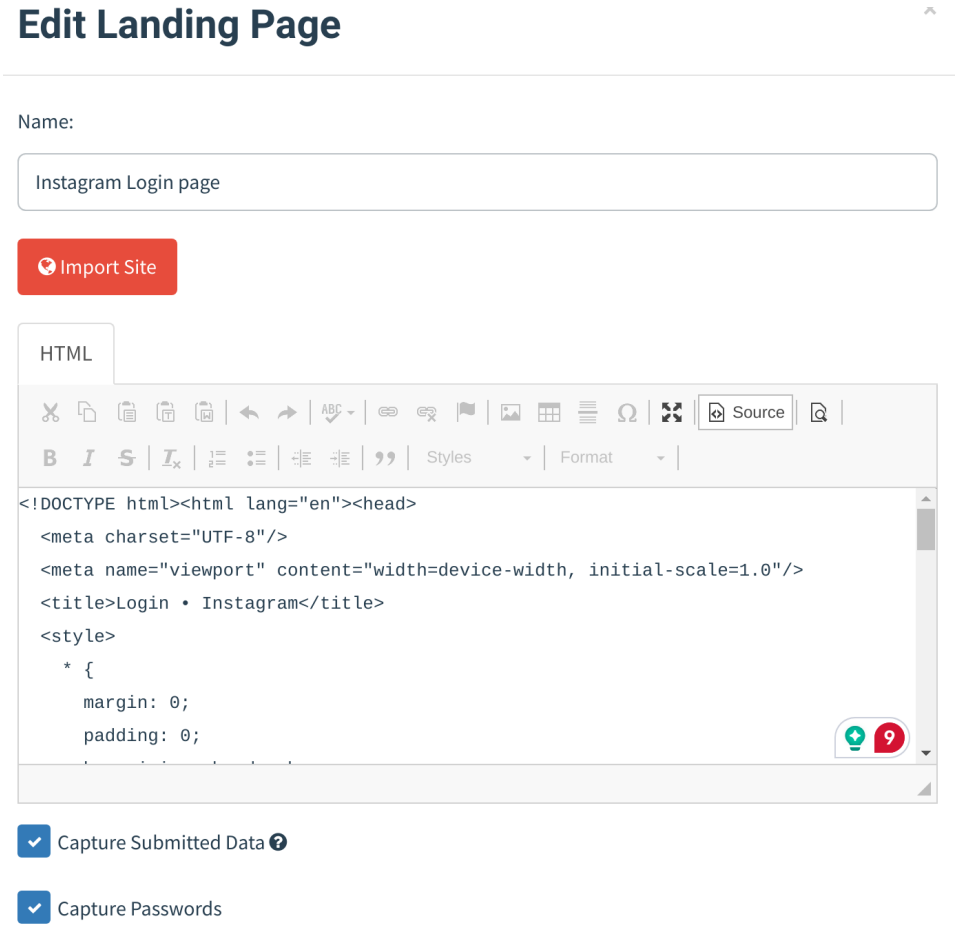
After inputting these credentials into the Gophish login page, I was granted access to the Gophish dashboard.

Instagram



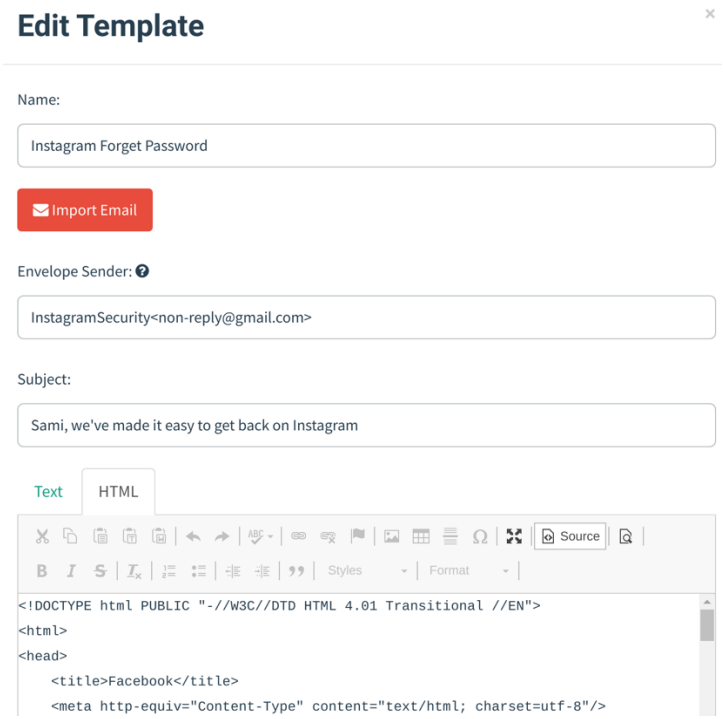
Once logged in, I set up the sending profile for the phishing campaign. This required configuring the SMTP settings to enable email dispatch. I created a profile named "Instagram team" and set the "SMTP From" field to an email address that would appear legitimate to the recipients, in this case, "mr.cipherghost@gmail.com." The SMTP server was set to "smtp.gmail.com" with the appropriate port number, and the same email address was used for the SMTP authentication along with a corresponding password. Ensuring "Ignore Certificate Errors" was checked allowed

me to bypass any potential SSL certificate validation issues, which could hinder email delivery.

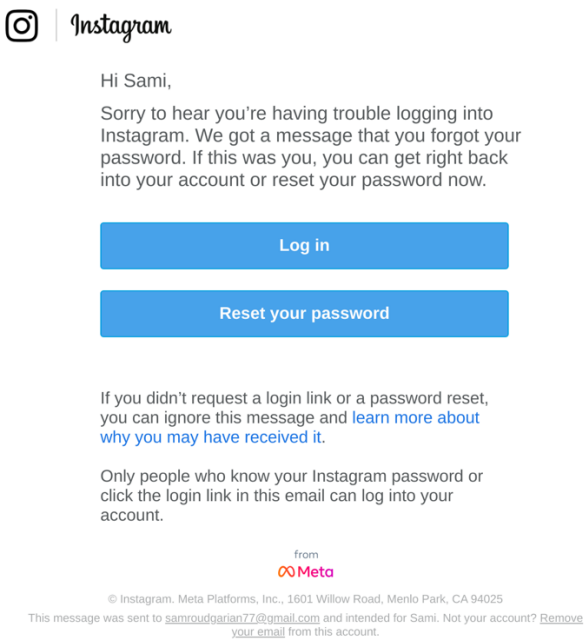


Next, I prepared the landing page, which is designed to mimic the login page of a well-known service, Instagram, to deceive targets into entering their credentials. I imported the HTML code for the fake Instagram login page and ensured the "Capture Submitted Data" and "Capture Passwords" options were enabled. These settings would allow me to collect the data entered by

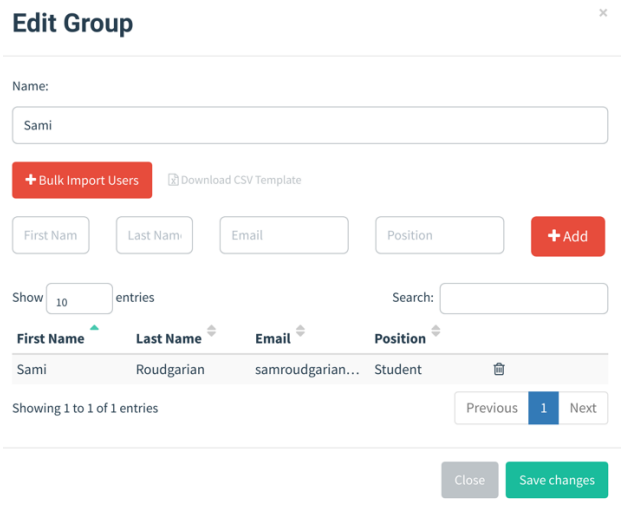
any user who fell for the phishing attempt, storing their credentials for analysis.



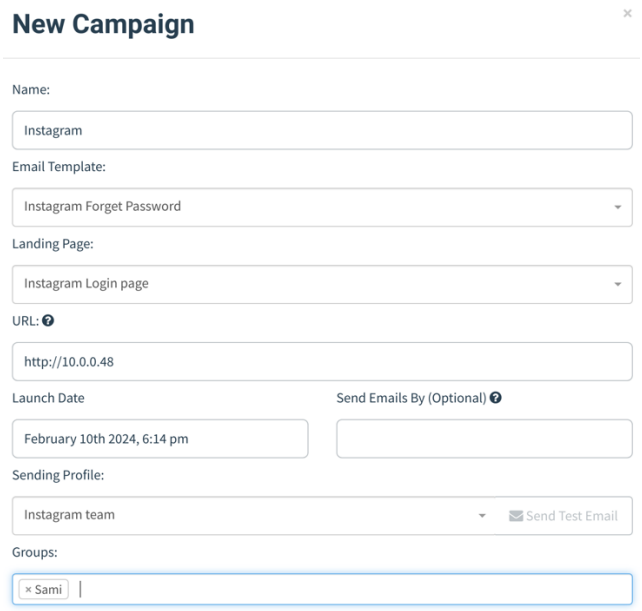
For the email template, I named it "Instagram Forget Password." I used an envelope sender that would appear official, and the email's subject was personalized to increase the chances of the recipient trusting the email. The HTML content was designed to look like an official Instagram communication, with options to log in or reset the password.



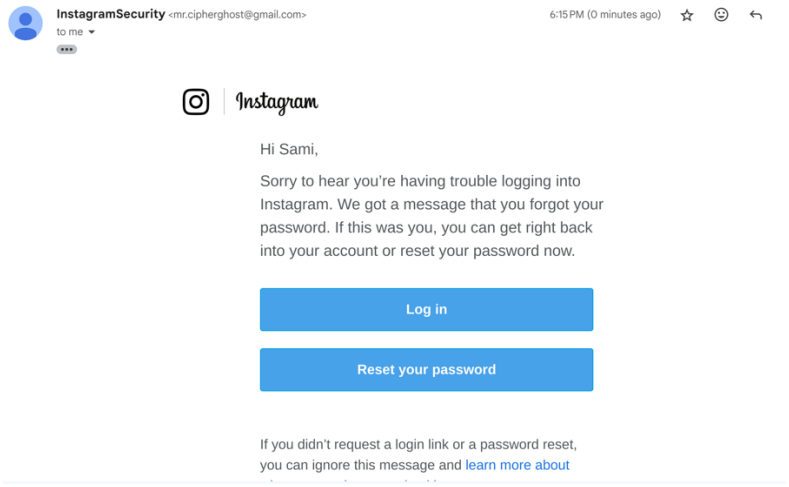
I used the actual email content that would be sent to the targets. It contained a message addressing the recipient by name, stating that they had requested a password reset, and provided buttons to either login or reset their password. The email design followed Instagram's branding to appear authentic, complete with a signature from Meta to lend credibility.
I designed the email and landing page carefully to avoid suspicion and encourage the recipients to click the links and enter their credentials, which Gophish would capture.



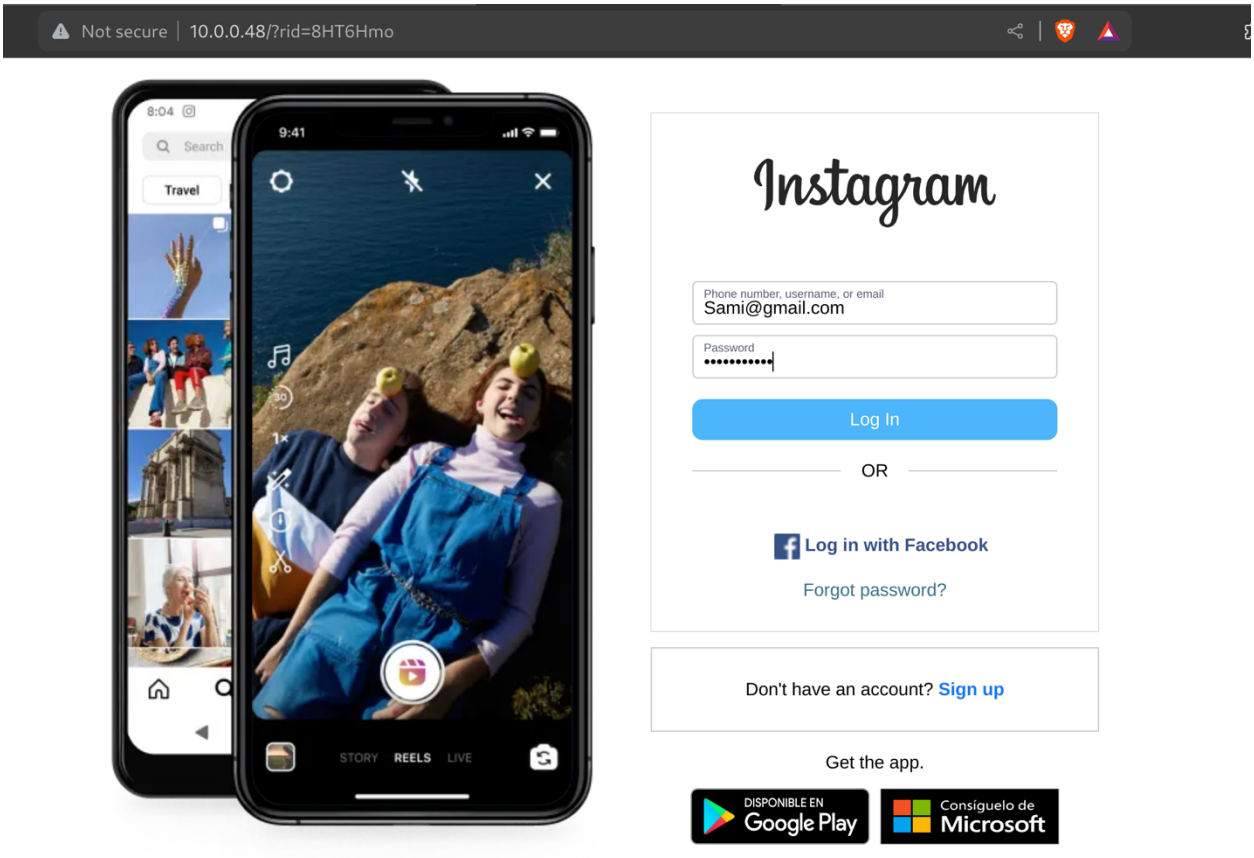
I set up a target group for this phishing exercise, naming it "Sami" after the intended recipient to add a personalized touch. Within this group, I included details such as the first name, last name, email, and position. These are only my personal information for educational purposes, and I am only targeting myself.



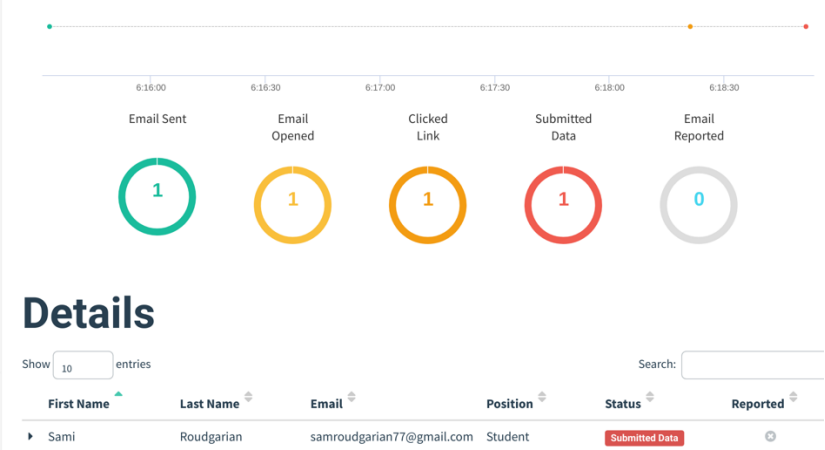
I set up a new campaign named "Instagram" and linked the email template and landing page I created. I defined the URL the phishing emails will use, where the Gophish server is listening to capture the data. I set the campaign to launch immediately and targeted a group containing your test recipient, 'Sami.'



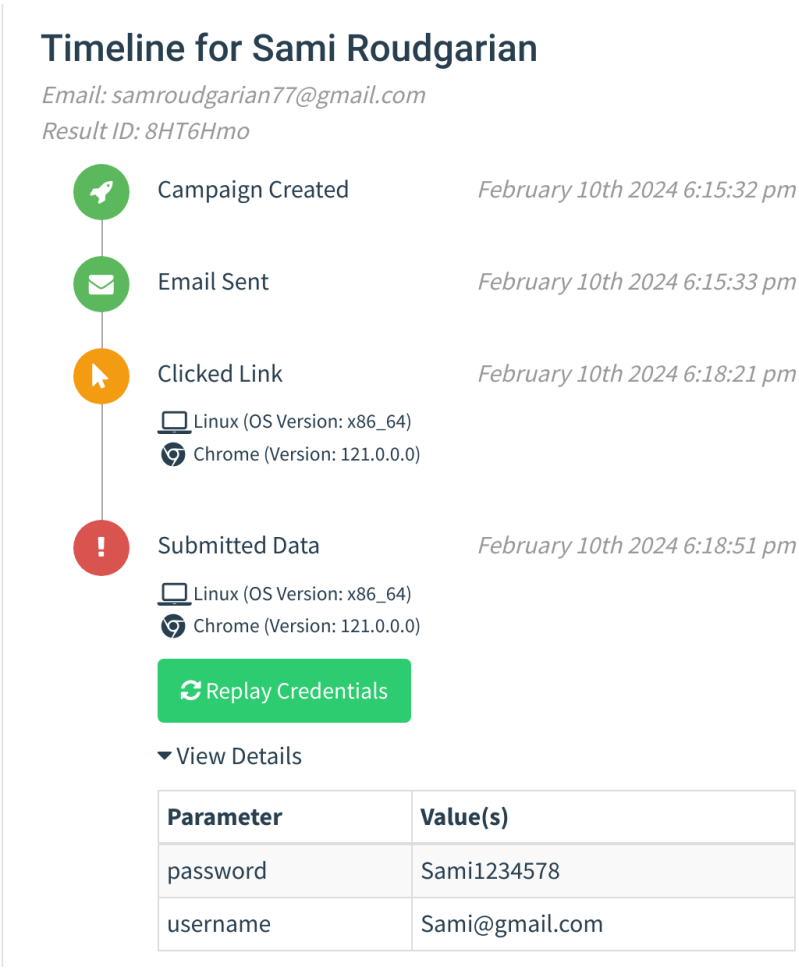
The targeted group has successfully received the email.



They have clicked on the Reset Password/Login button on the email template and navigated to our HTML page.



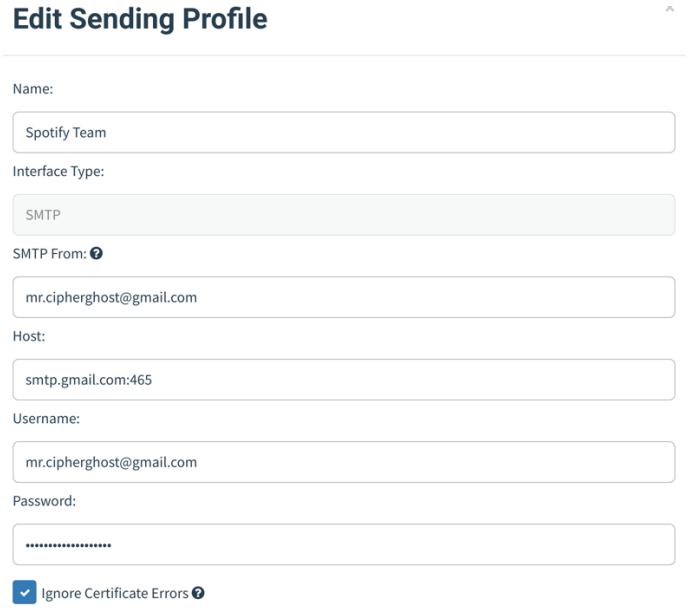
After launching the campaign, I monitored the progress using Gophish's dashboard to see metrics such as emails sent, opened, clicked, data submitted and reported. I observed that 'Sami' received the email, clicked the link, and submitted the data, indicating a successful phishing attempt.



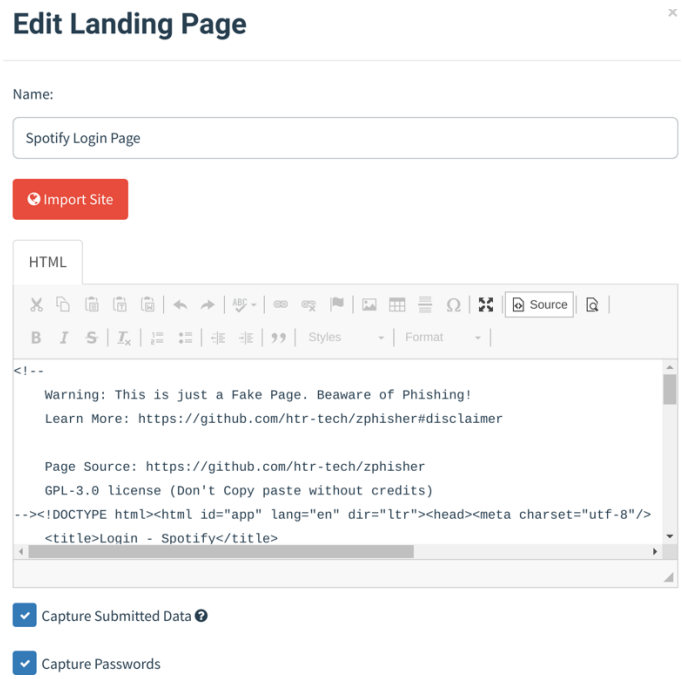
The campaign showed me the details of 'Sami's' interaction with the phishing email. I saw that ‘Sami’ clicked the link and entered the credentials. This detailed timeline helps us understand user behaviour and the effectiveness of our phishing campaign.
The detailed timeline illustrates when the phishing email was sent, when the victim clicked (contains information about the OS and browser), and their submitted credentials. In this campaign, I made the victim’s Instagram Username and Password field our goal to receive. Throughout this process, I would have ensured that all steps were legal, ethical, and within the boundaries of a controlled environment for educational or authorized testing purposes only.

Spotify

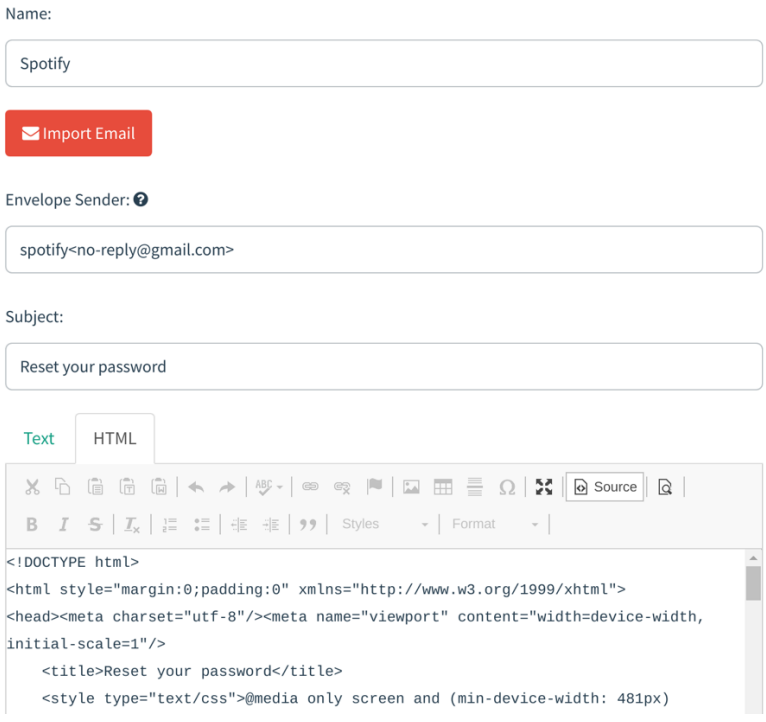
I launched another phishing campaign targeting Spotify credentials.



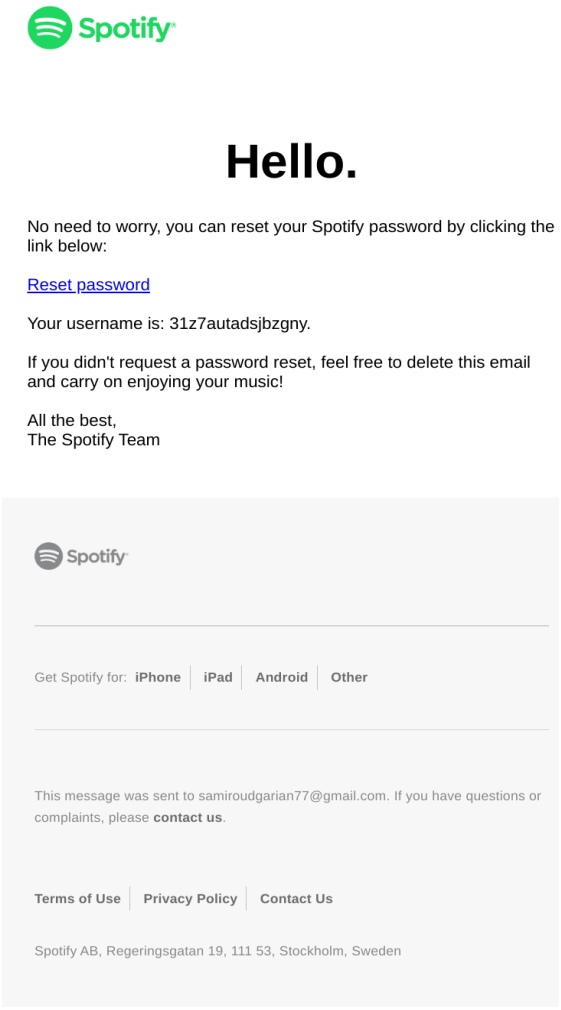
Firstly, I configured the sending profile, naming it "Spotify Team”. I set the SMTP server to smtp.gmail.com with SSL encryption on port 465, using "mr.cipherghost@gmail.com" as both the sender and username, along with the corresponding password (Same steps as the previous campaign). The option to "Ignore Certificate Errors" was enabled, ensuring the emails would be sent without any SSL validation issues that could impede their delivery.



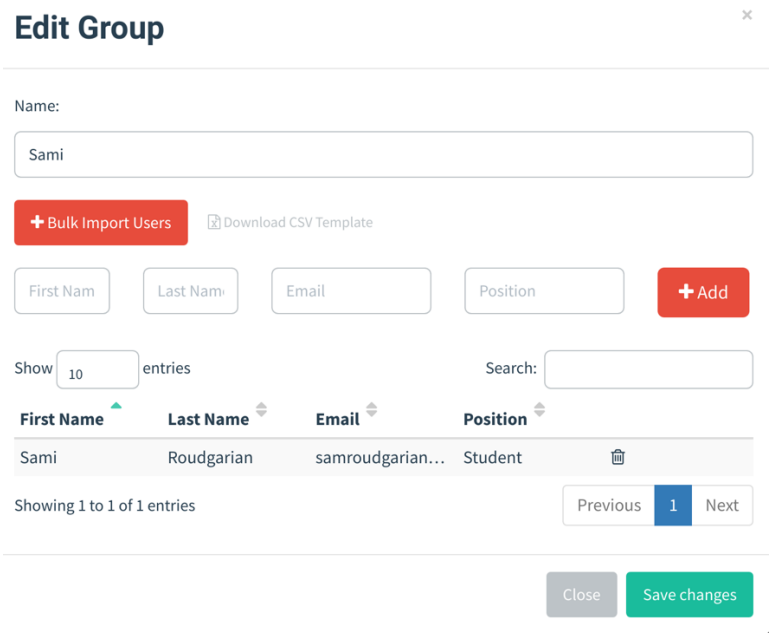
Next, I moved on to create the landing page. I crafted a "Spotify Login Page," designed to closely mimic the authentic Spotify login interface to effectively deceive the targets. I enabled options to "Capture Submitted Data" and "Capture Passwords," which were crucial for recording the credentials entered by anyone who fell for the phishing attempt.



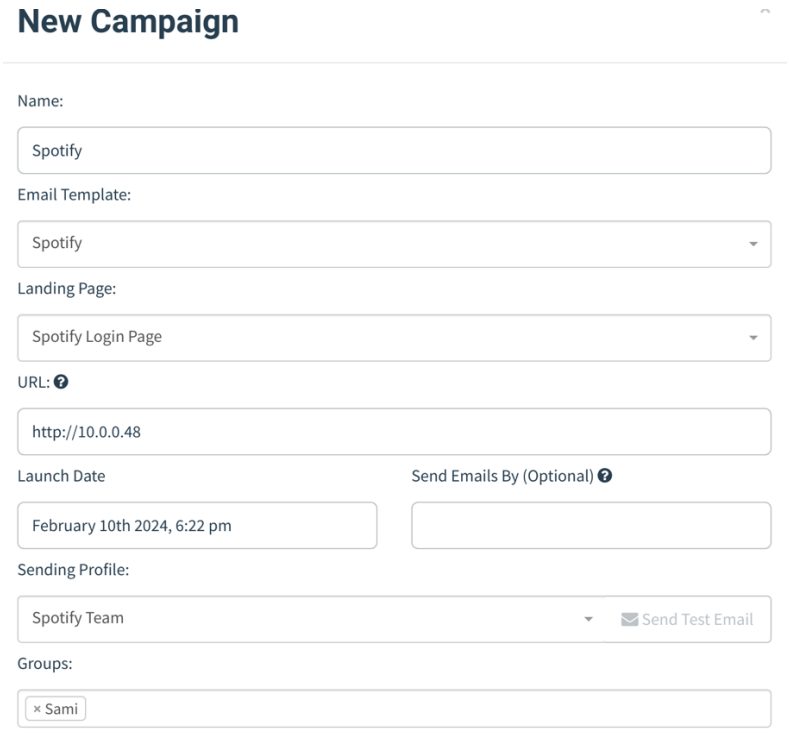
For the email template, I designed it under the name "Spotify" and chose "spotifyno-reply@gmail.com" as the envelope sender to reinforce the impression of an official Spotify communication.



The subject line "Reset your password" was carefully chosen to incite a sense of urgency and prompt the recipient to take immediate action. Using HTML, I copied a convincing body of the official spotify email, instructing recipients on resetting their password.

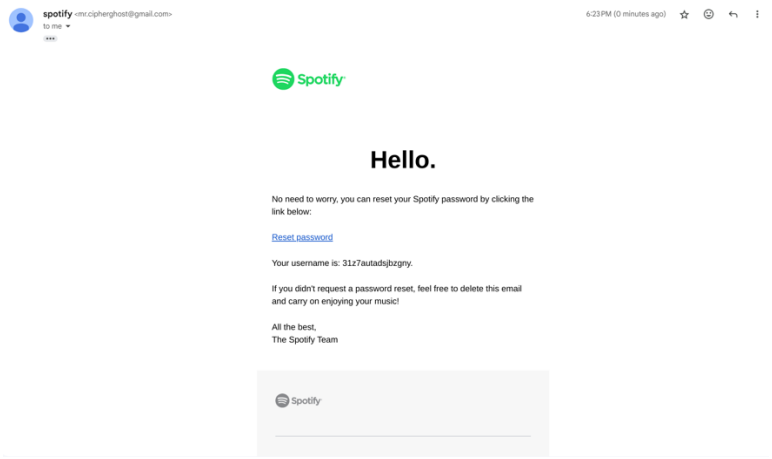


Again, I set up a target group for this phishing exercise, naming it "Sami" after the intended recipient to add a personalized touch. Within this group, I included details such as the first name, last name, email, and position. These are only my personal information for educational purposes, and I am only targeting myself.

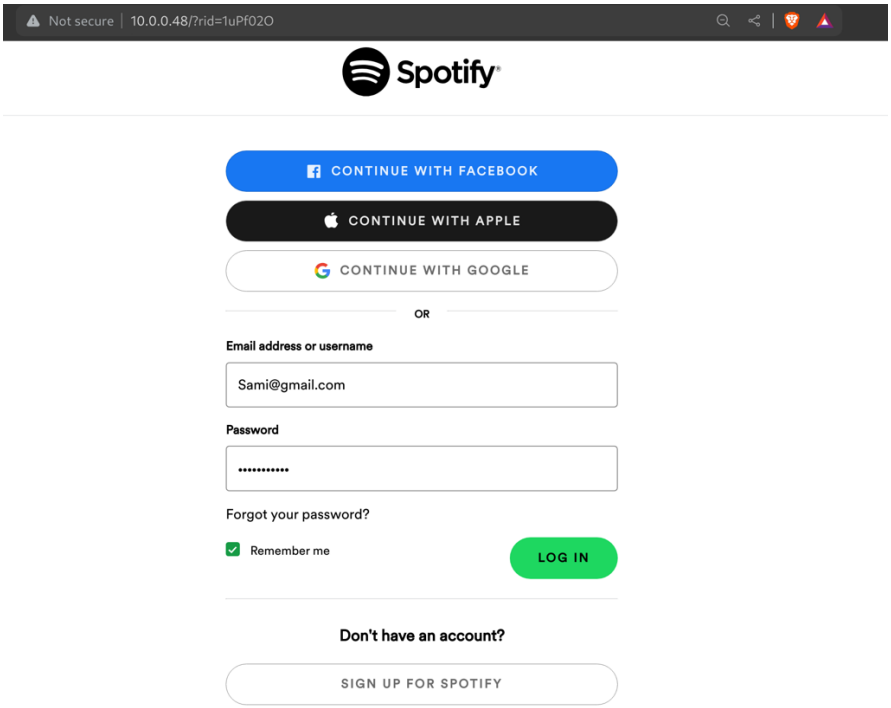


I proceeded to set up the new phishing campaign named "Spotify." I chose the email template and landing page to mimic a legitimate Spotify password reset process. I then specified the URL

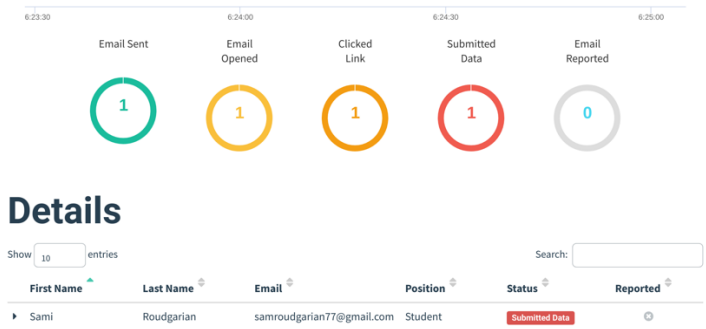
where the Gophish server was listening to capture any data entered by the recipient. I scheduled the campaign to launch immediately, targeting the group "Sami," which contained my information. This approach simulates when the attacker has prior knowledge of the potential victim.



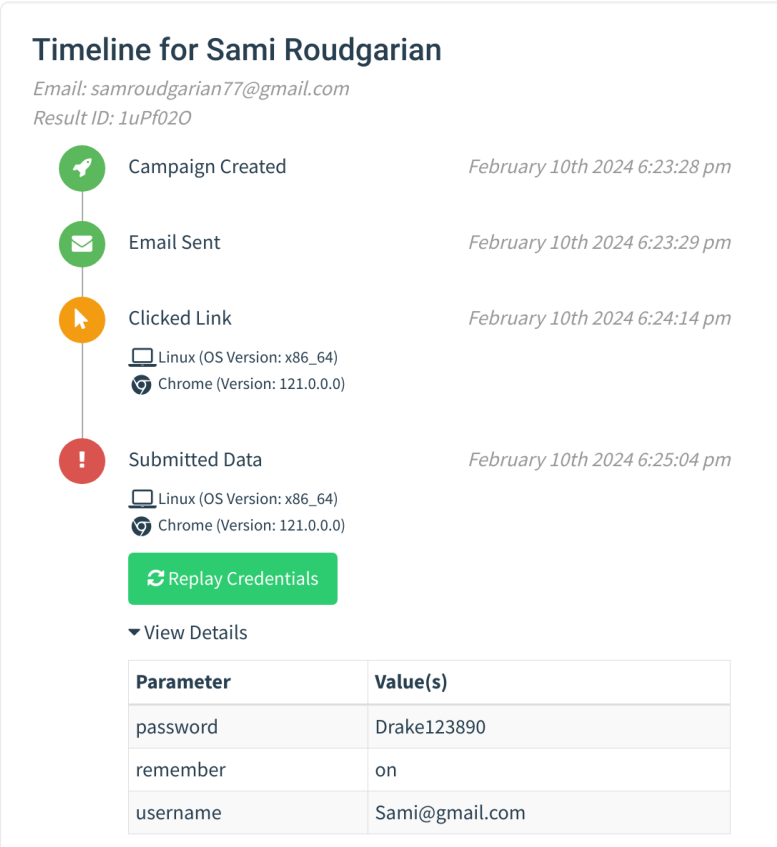
I (the victim) received the crafted email, which appeared genuine, encouraging me to reset my Spotify password. The email provided a sense of urgency and a direct call to action.



Upon clicking the 'Reset password' link, I was directed to the fake Spotify login page, where I was prompted to enter my credentials.



As expected, the Gophish dashboard showed that the email was sent, opened, the link was clicked, and the data was submitted without any reports of the email being flagged as phishing. This indicated a successful deception, with the recipient interacting with the email as intended by the attacker.



The timeline detailed when the email was sent, the link was clicked, and the data was submitted, including the specific username and password entered, which would represent the sensitive information the attacker sought to obtain in a real-world scenario.

Overall, Gophish has demonstrated its utility as a powerful tool for simulating phishing campaigns to test security awareness and defenses. Through meticulous setup and execution of

campaigns targeting both Instagram and Spotify credentials, I've evaluated the effectiveness of phishing techniques and the importance of attention to detail in crafting convincing emails and landing pages. The campaign's success was measured by the recipients' interactions, from opening emails to submitting data, which were tracked in real-time. This exercise reinforced the necessity of education in email security. Importantly, all activities were conducted ethically within my controlled environment for security training and respecting legal and ethical boundaries.

Case study

Introduction to the Twitter Spear-Phishing Attack

In July 2020, Twitter faced a sophisticated spear-phishing attack that targeted its employees. The Twitter spear-phishing attack was a multi-staged operation where attackers impersonated Twitter's IT department to trick employees into revealing their credentials. This allowed them to access Twitter's internal systems and take control of high-profile accounts for a Bitcoin scam.
https://en.wikipedia.org/wiki/2020_Twitter_account_hijacking

Methodology of the Attack

The attackers began their operation by gathering personal information about these employees to make their phishing attempts more convincing. They then made phone calls pretending to be from Twitter's IT department, claiming they were addressing VPN issues—a common problem for employees working remotely at the time. This initial contact was a ruse to lure employees to a phishing website that closely resembled Twitter's legitimate VPN login page. Once the employees entered their credentials on the phishing site, the attackers simultaneously entered them into the actual Twitter site, triggering an MFA (Multi-Factor Authentication) notification. Some employees, believing the communication was genuine, authenticated the access, which allowed the attackers to breach Twitter's systems. The first employee whose account was compromised did not have direct access to account management tools, but the attackers used the breach to conduct reconnaissance within Twitter's internal systems to identify and target additional employees.
https://www.lookout.com/threat-intelligence/article/twitter-phone-spear-phishing-attack

https://en.wikipedia.org/wiki/2020_Twitter_account_hijacking
https://teampassword.com/blog/what-happened-during-the-twitter-spear-phishing-attack

Financial Impact

The attack targeted 130 Twitter accounts, tweeted from 45, accessed the direct message inboxes of 36, and downloaded the Twitter data of 7 accounts. The financial impact of the attack was felt as the attackers used high-profile accounts, such as those of Elon Musk and Barack Obama, and other celebrities and public figures, using these accounts to tweet a Bitcoin scam to millions of followers. The fraudulent tweets promised to double any Bitcoin sent to a specified address. This resulted in at least $100,000 in Bitcoin being transferred to the attackers within a short timeframe.
https://en.wikipedia.org/wiki/2020_Twitter_account_hijacking
https://teampassword.com/blog/what-happened-during-the-twitter-spear-phishing-attack

Long-term Effects

The attack caused significant reputational damage to Twitter and led to increased security from regulatory organizations. It highlighted the potential risks to elections, financial markets, and national security due to the influential role of social media.

Resolution and Response

In response to the attack, Twitter has acknowledged the need for heightened security awareness across all areas of its organization. It was taking measures to limit access to internal tools and systems while it completed its investigation. The company has also accelerated pre-existing security workstreams and improved its methods for detecting and preventing inappropriate access to its internal systems. They have emphasized the importance of organizing ongoing phishing exercises throughout the year to bolster their defenses against such social engineering attacks.
https://www.lookout.com/threat-intelligence/article/twitter-phone-spear-phishing-attack

Analysis and Documentation

The Twitter incident underscores the importance of employee training and the challenges in detecting phishing attacks, especially when attackers use information obtained from previous

breaches to convince. Security experts have commented on the need for organizations to be vigilant about their employee data and to train employees to be ready for unexpected emails or phone calls. It's crucial to have policies to report such incidents promptly and investigate them appropriately.

Human vulnerabilities can often be the weakest link in security. Despite robust technical controls, attackers can exploit human nature to bypass even the most sophisticated defences. This case study highlights the risk of social engineering and the need of security awareness that can mitigate such risks.

https://www.dfs.ny.gov/Twitter_Report

https://www.lookout.com/threat-intelligence/article/twitter-phone-spear-phishing-attack

Mitigation Cost Comparison

Financial Impact of Phishing Attacks

Phishing attacks can lead to significant financial losses for organizations. These losses stem from the theft of funds or data and the costs associated with breach remediation, legal fees, reputation damage, and loss of customer trust. In July 2023, IBM released its annual Cost of a Data Breach Report based on data from 553 global organizations, showing a 15% increase over the last three years, with an average of $4.45 million. According to Zscaler's 2023 Phishing Report, there was a 47.2% surge in phishing attacks compared to the previous year. This highlights the escalating complexity of breach investigations and the need for advanced security measures.

https://www.zscaler.com/blogs/security-research/2023-phishing-report-reveals-47-2-surge-phishing-attacks-last-year#

https://newsroom.ibm.com/2023-07-24-IBM-Report-Half-of-Breached-Organizations-Unwilling-to-Increase-Security-Spend-Despite-Soaring-Breach-Costs

Cost and Efficacy

Investing in preventive cybersecurity measures is essential for organizations to protect against financial losses from phishing attacks. Costs for these measures can include:

- **Security Awareness Training:** Ranges from USD$0.42 to USD$4 monthly per employee, depending on the provider and the service type (self-service, semi or fully managed, or niche companies).

- **Advanced Security Systems:** Costs vary widely depending on the system's complexity, the organization's size, and the level of protection required.
- **Regular Security Audits:** Essential for identifying vulnerabilities, with costs dependent on the audit's scope and the organization's size.

The efficacy of these investments is demonstrated by their ability to significantly reduce the risk and impact of phishing attacks, which can lead to substantial long-term savings and risk mitigation.
https://caniphish.com/blog/how-much-does-security-awareness-training-cost

Mitigation vs. Investment

A comparative analysis of the costs associated with phishing attacks versus the investment required for preventive measures includes:

- **Financial Impact of Phishing Attacks:** The average data breach cost has risen to $4.45 million, highlighting the significant potential losses that organizations face.
- **Investment in Preventative Measures:** While upfront costs for training and security measures may seem substantial, they are generally lower than those associated with a successful phishing attack.
- **Cost-Benefit of Preventive Measures:**
    - Security awareness training can reduce the average breach cost by $232,867, demonstrating a direct return on investment.
    - The investment in employee training is validated by reducing breach likelihood, making it a financially sound decision.

Initial investments in cybersecurity training and measures may be significant, but they are justified when considering the high costs associated with phishing attacks and data breaches. To ensure a comprehensive and cost-effective cybersecurity strategy, organizations must weigh the costs of implementing these measures against the potential financial losses from cyber incidents.
https://www.livingsecurity.com/blog/how-much-security-awareness-training-cost

Conclusion

In concluding my project, I've gained significant insights into the vulnerabilities of web applications and the effectiveness of phishing as a cybersecurity threat. Using Burp Suite, I found various vulnerabilities such as SQL injection, XSS, brute force, command injection, stored XSS, and open HTTP redirects. This exploration emphasized the critical need for thorough input validation, the importance of secure coding practices, and the role of encryption in defending against potential threats. Tools like Intruder and Repeater highlighted the need for ongoing monitoring and testing to identify and mitigate vulnerabilities efficiently.

My simulations with Gophish underscored human errors in cybersecurity breaches. By simulating phishing campaigns targeting credentials for platforms like Instagram and Spotify, I demonstrated how individuals could be deceived into compromising their personal and sensitive information. This underlines the importance of comprehensive security awareness training to equip individuals with the knowledge to recognize and appropriately respond to phishing attempts.

The case study on the Twitter spear-phishing attack served as a real-world example of the sophistication of social engineering tactics and their associated risks. It reinforced the necessity of employee training and robust authentication mechanisms, such as multi-factor authentication, to prevent unauthorized access.

Furthermore, comparing the financial impact of phishing attacks with the investment in preventive cybersecurity measures revealed the cost-effectiveness of proactive security measures. Investments in security awareness training, advanced security systems, and regular security audits reduce the likelihood of successful attacks and mitigate potential financial losses and reputational damage.

Based on these findings, I recommend several measures to enhance network security. Implementing input validation and secure coding practices to prevent common vulnerabilities. Encryption should be used to protect data in transit and at rest. Regular security testing using tools like Burp Suite is crucial for the timely identification and remediation of vulnerabilities. Security awareness training must be enhanced to educate employees and users on the risks of phishing and other cybersecurity threats. Robust authentication mechanisms, such as multi-factor authentication, should be adopted to reduce the risk of compromised credentials. Finally,

investing in advanced security systems and conducting regular audits are necessary to identify and address security gaps, ensuring a comprehensive defence against cyber threats.
In summary, my project highlighted the multifaceted nature of cybersecurity, involving both technological defences and human factors. Adopting a holistic approach that combines technology with education and awareness is key to significantly enhancing an organization's security posture in the face of the constantly evolving landscape of cyber threats.

Appendix A (HTML Codes)

HTML files are available in the data folder.
I wrote Instagram.html.
Spotify.html is copied from the Zphisher template. (https://github.com/htr-tech/zphisher.git)

Appendix B (Input Data)

Data	Info	Value
Name	My first and last name which was used during the GoPhish task.	Sami Roudgarian
Email	My email as victim	samroudgarian77@gmail.com
Attacker Email	My Email for the Gophish taks	mr.cipherghost@gmail.com