CS375-Project

OWASP Top 10 Web Security Vulnerabilities 2017

Syed Roshan Ali (2021648)

Muhammad Tahir Zia (2021465)

➔ Injection

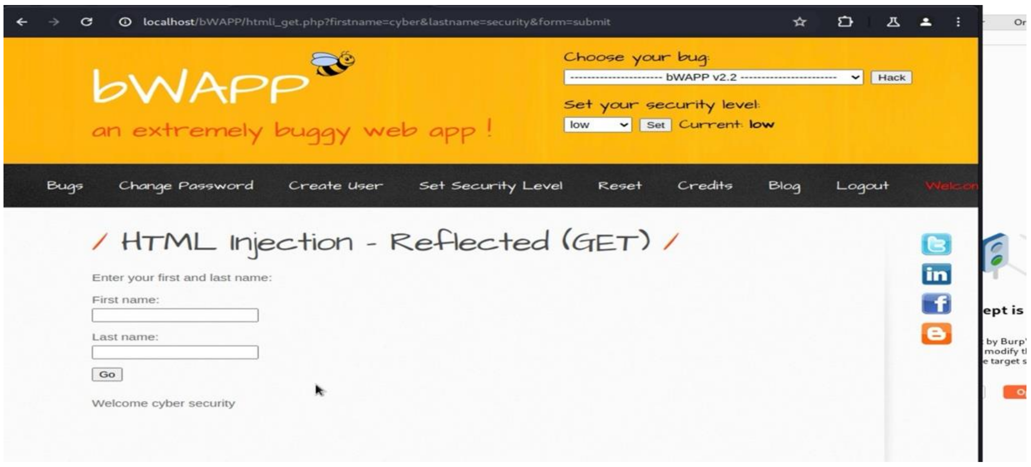
Imagine a website asks you to type in your username. Instead of just typing your name, a hacker could try to trick the website into running special instructions. These instructions might steal your information, mess with the website, or even give hacker control! This sneaky way of injecting instructions is called an injection attack

HTML Injection (Reflected GET)

http://localhost/bWAPP/htmli_get.php

1. Find the bug selection menu in the top right corner of the website. It might be a dropdown list or a button.
2. Clicking that bug menu will take you to a new page with two boxes for entering text. These are likely labeled "First Name" and "Last Name".
3. The interesting part is, you can try typing special codes in these boxes, not just your actual name.

- Screenshot Before Action



localhost/bWAPP/htmli_get.php?firstname=cyber&lastname=security&form=submit

Choose your bug: bWAPP v2.2 Hack

Set your security level: low Set Current: low

/ HTML Injection - Reflected (GET) /

Enter your first and last name:

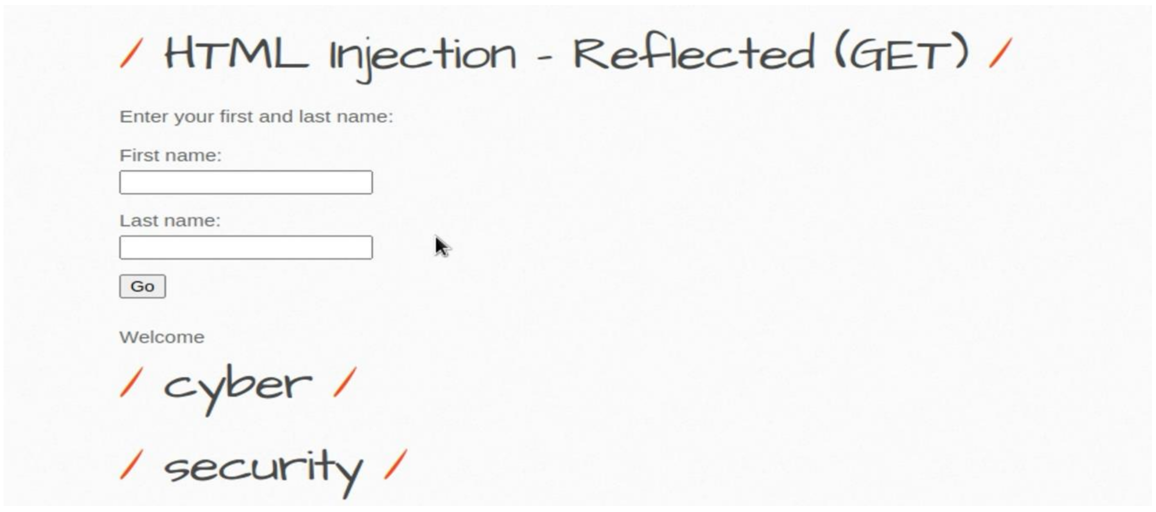
First name:

Last name:

Go

Welcome cyber security

- Screenshot After Action


/ HTML Injection - Reflected (GET) /

Enter your first and last name:

First name:

Last name:

Go

Welcome

/ cyber /

/ security /

Observation: From the images above, we see that we can inject HTML code snippets into the fields of the webpage and when executed they are processed as HTML tags rather than regular text. This works for low level security level and not for medium and high level. In this example we have injected an h1 heading html tag which is successfully injected into the body of the web page and can be viewed. This vulnerability can be exploited by an attacker to inject malicious scripts which can run on a user's system when he views a certain webpage and compromise his system.

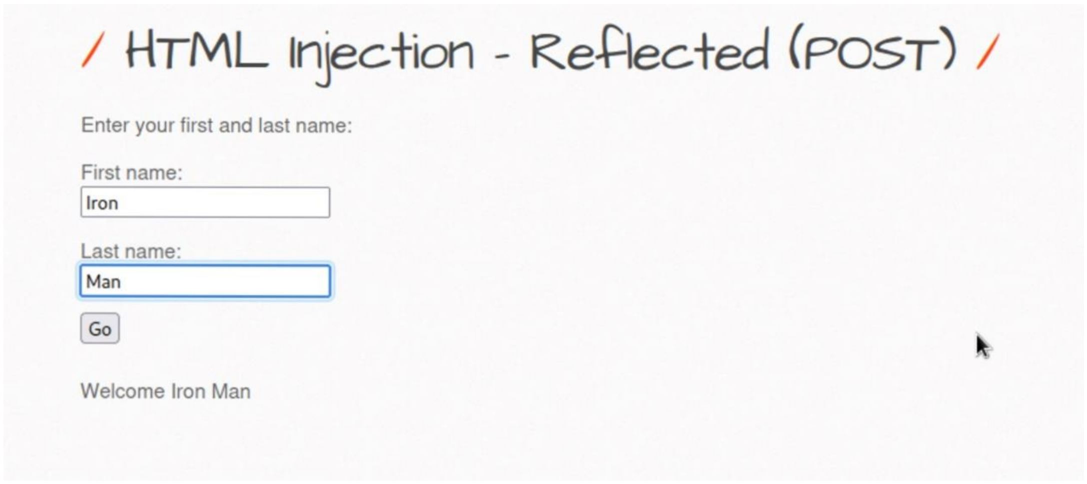
Preventions:

1. Validate all input data which is entered in an input field to ensure structure integrity.
2. Any output that is sent back to the user should be properly encoded so that an injected script or html snippet is unable to run on the user's system.
3. Save the users input to the database in a text only format so that when it is re-rendered it is done so as a text rather than a html tag.

HTML Injection (Reflected POST)

http://localhost/bWAPP/htmli_post.php

- Screenshot Before Action


/ HTML Injection - Reflected (POST) /

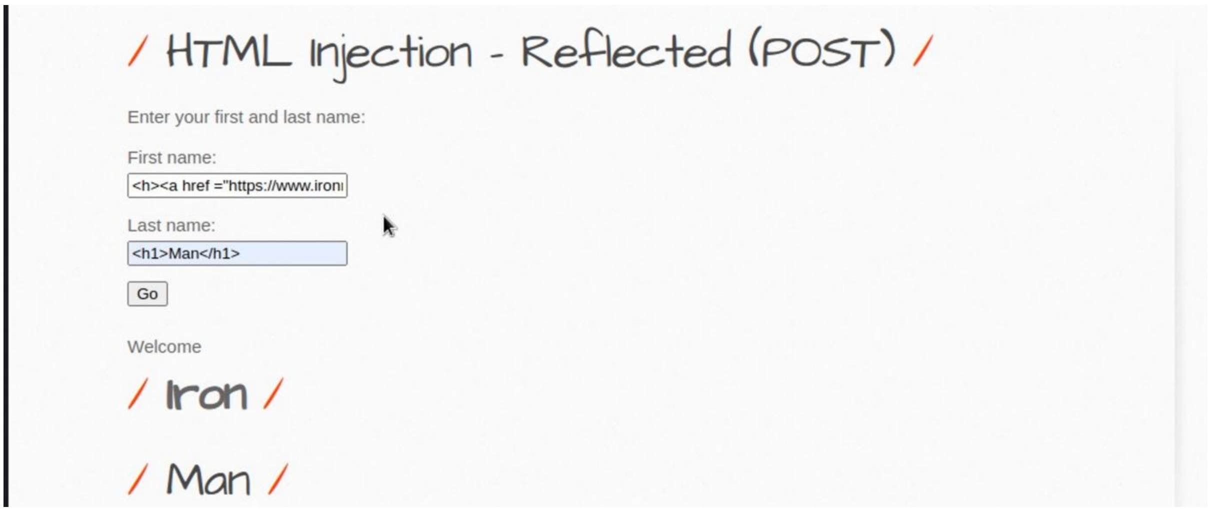
Enter your first and last name:

First name: Iron

Last name: Man

Welcome Iron Man

- Screenshot After Action


/ HTML Injection - Reflected (POST) /

Enter your first and last name:

First name: <h><a href ="https://www.ironı

Last name: <h1>Man</h1>

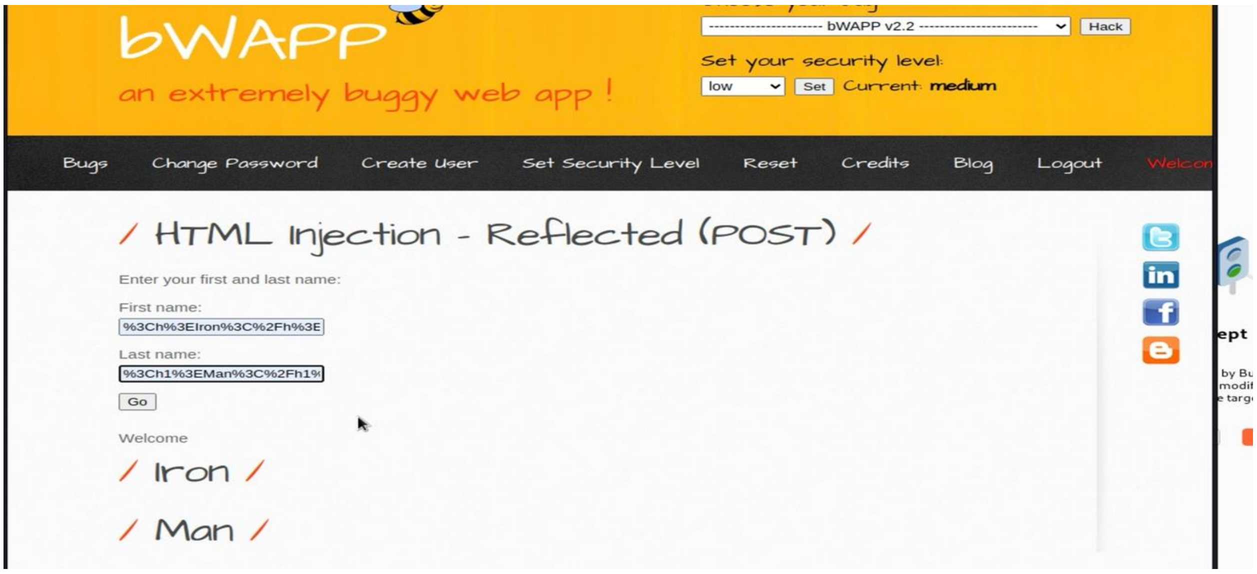
Welcome

/ Iron /

/ Man /

This didn’t worked when I set the difficulty level to medium. This indicated that there must be some encoding going on the input field values. Then I used the url encoder to encode the input and demonstrate reflection using input fields. Here is the output for <h1>Iron</h1>:Man

The first name is encoded as follows: “<h1>Iron</h1>”to “%3Ch%3EIron%3C%2Fh%3E” and last name is encoded as follows: “<h1>Man</h1>”to “%3Ch1%3EMan%3C%2Fh1%3E”. After using encoded url in the first name field, we get the following output:



/ HTML Injection - Reflected (POST) /

Enter your first and last name:

First name: %3Ch%3EIron%3C%2Fh%3E

Last name: %3Ch1%3EMan%3C%2Fh1%

Go

Welcome

/ Iron /

/ Man /

Preventions:

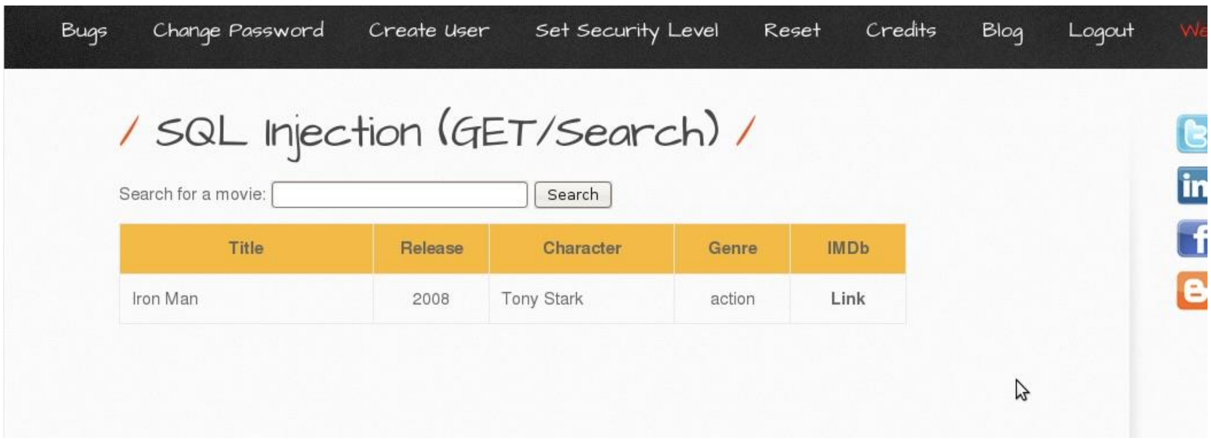
1. Validate all input data which is entered in an input field to ensure structure integrity.
2. Any output that is sent back to the user should be properly encoded so that an injected script or html snippet is unable to run on the user’s system.
3. Use security tokens to ensure that each POST request is a valid request before it is processed so that a malicious script cannot be injected.

SQL Injection (GET / Search)

http://localhost/bWAPP/sqli_1.php

The web page with an input field for a movie is displayed. In the input field we see that we can inject SQL queries which when run gives us access to data in the database which we should not have.

- Screenshot Before Action

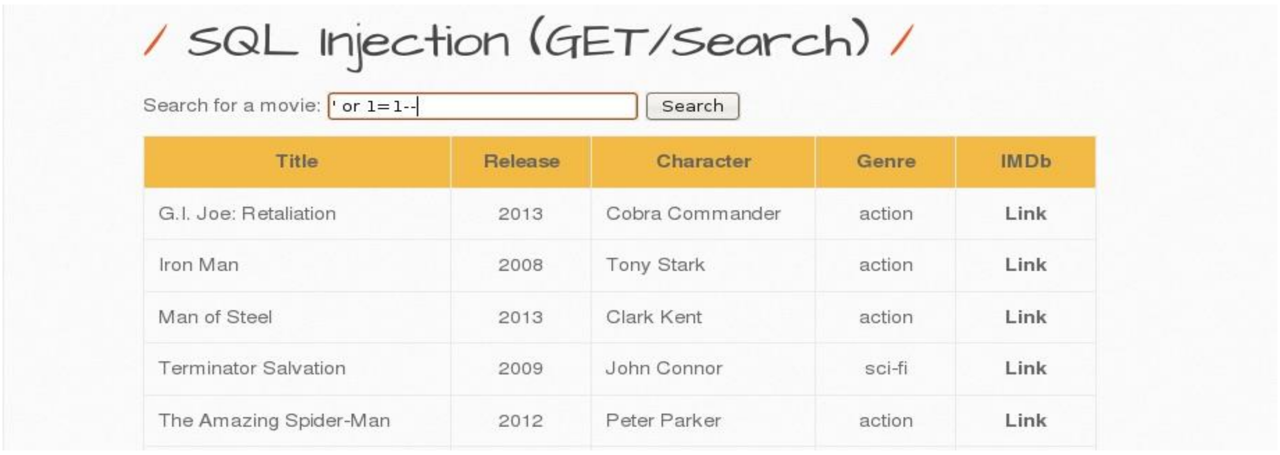


/ SQL Injection (GET/Search) /

Search for a movie: [ ] Search

Title	Release	Character	Genre	IMDb
Iron Man	2008	Tony Stark	action	Link

- Screenshot After Action



/ SQL Injection (GET/Search) /

Search for a movie: ' or 1=1-- Search

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

Observation: Typing special codes lets us see secret website stuff we shouldn't. Hackers could steal or change important information, or even take over the whole website!

➔ Broken Authentication

When an application's authentication procedures are not appropriately implemented, it creates a sort of security vulnerability known as broken authentication that enables an attacker to get around authentication and access restricted sections of the application or system. Insecure password reset features, weak passwords, session management difficulties, credential stuffing, and other techniques can all be used to take advantage of this vulnerability. A successful attack can have catastrophic effects, including breach of data, system compromise, and damage to the application’s reputation.

Forgotten Function

http://localhost/bWAPP/ba_forgotten.php

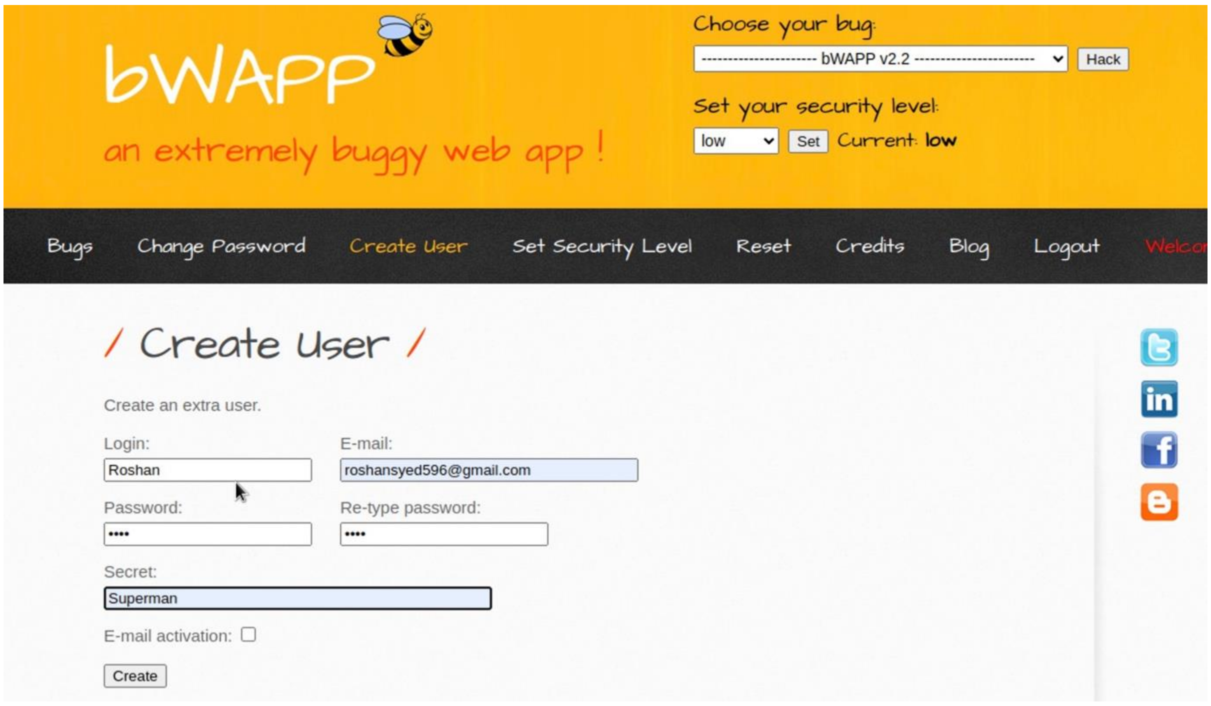
In our exploration of a simulated vulnerable web application known as Beebox, we decided to test its "Broken Authentication (Forgotten Function)" feature. We entered our email address into the designated input field. Alarmingly, the system provided us with what appeared to be the account's secret information.

Understanding the Simulated Attack:

This scenario highlights a classic web application security vulnerability: broken authentication through a forgotten password function. In a real-world scenario, an attacker could exploit this weakness by:

Targeting a large number of email addresses: They might use automated tools to try various email addresses in the hope of compromising an account.

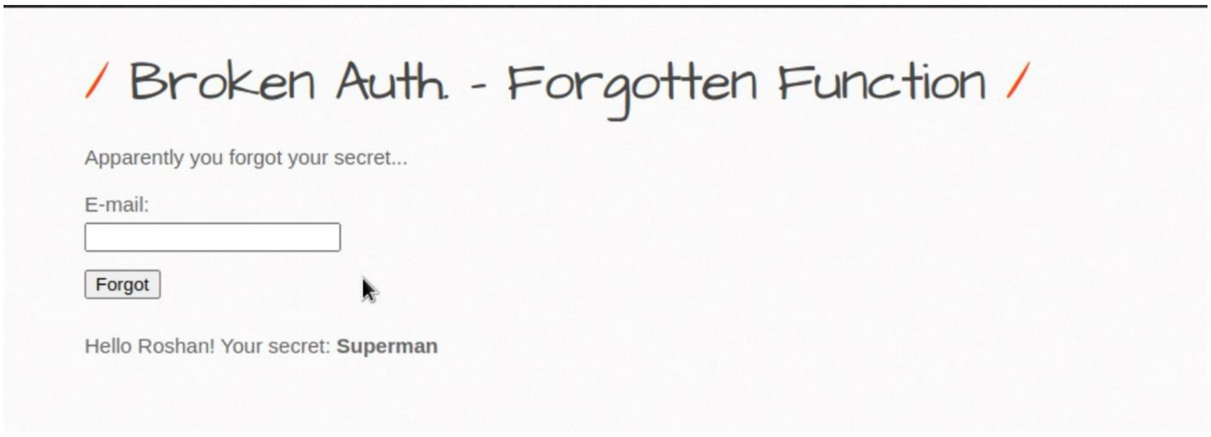
Leveraging leaked email lists: If email addresses and passwords have been leaked from another website or service, attackers could try those combinations on Beebox.



/ Create User /

Create an extra user.

Login: Roshan
E-mail: roshansyed596@gmail.com
Password: ••••
Re-type password: ••••
Secret: Superman
E-mail activation: ☐



/ Broken Auth. - Forgotten Function /

Apparently you forgot your secret...

E-mail:

Hello Roshan! Your secret: **Superman**

Here are some crucial steps to prevent such security breaches in real-world web applications:

Implement strong password policies: Enforce minimum password lengths, complexity requirements (including a mix of uppercase, lowercase, numbers, and symbols), and regular password changes.

Multi-factor authentication (MFA): This adds an extra layer of security by requiring a second verification step, such as a code sent to the user's phone, in addition to a password.

Secure password reset processes: Don't reveal sensitive information like passwords directly in the password reset process. Instead, send a secure link to the user's email where they can create a new password.

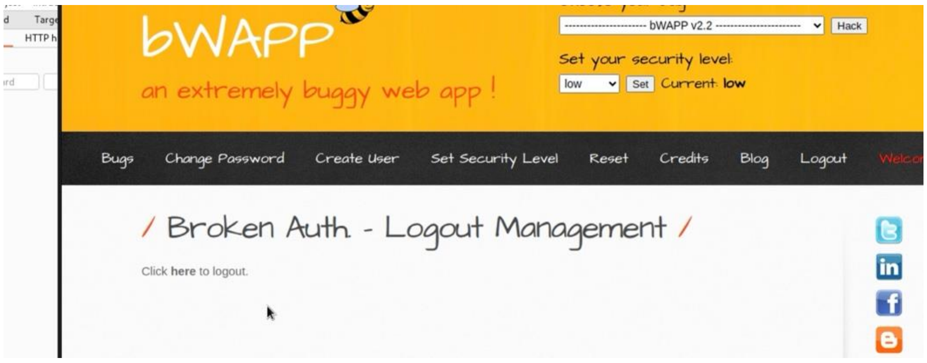
Regular security audits: Conduct regular penetration testing and vulnerability assessments to identify and address security weaknesses before attackers can exploit them.

Logout Management

http://localhost/bWAPP/ba_logout.php

A popup is displayed which asks us to confirm our logout and upon clicking OK it takes us back to the login page. However, when we click on the button of the client browser that allows us to go back to the previous web page, we observe that we can in fact again go back to the logged in session and navigate around the web application which we should not be able to do and hence the login functionality of this web application is broken. This only works for the low security level. On the medium and high-level security when we click the back button, we are still prompted to the login page and cannot go back to the session.

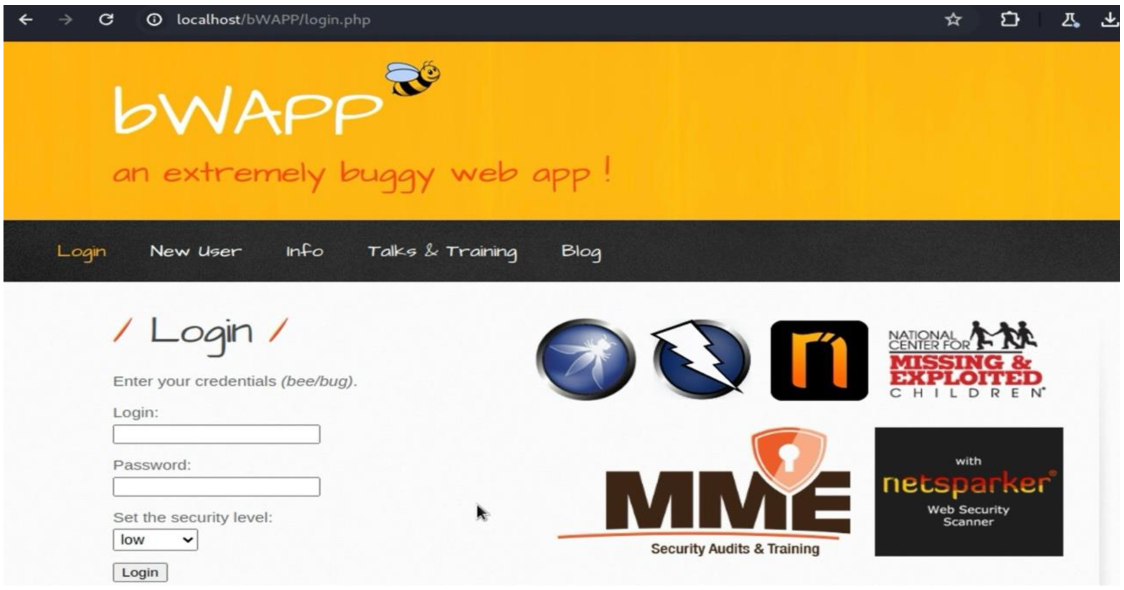
- Screenshot Before Action



/ Broken Auth. - Logout Management /

Click **here** to logout.

- Screenshot After Action


localhost/bWAPP/login.php
bWAPP
an extremely buggy web app !
Login
New User
Info
Talks & Training
Blog
NATIONAL CENTER FOR MISSING & EXPLOITED CHILDREN
MME
Security Audits & Training
with netsparker Web Security Scanner

/ Login /

Enter your credentials *(bee/bug)*.

Login:

Password:

Set the security level: low

Login

Observation:

The logout button is broken!

Clicking "back" in your browser lets you stay logged in even after clicking "logout." This is like the website forgetting you ever logged out!

You can also be logged in on two separate tabs at the same time. Logging out of one shouldn't keep you logged in on the other. Imagine two keys to the same door - only needing one to get in is a problem!

These problems are dangerous because attackers could steal your information or mess with your account if they trick the website into thinking they're you.

Preventions:

1. Implement a brief timeout period so that inactive users will be automatically logged out after a predetermined amount of time.
2. Make sure you delete any session tokens or cookies linked with a user's session after they log out.
3. To prevent client-side scripts from accessing or intercepting session data, use HTTP-only and secure cookies to store session information.

➔ Sensitive Data Exposure

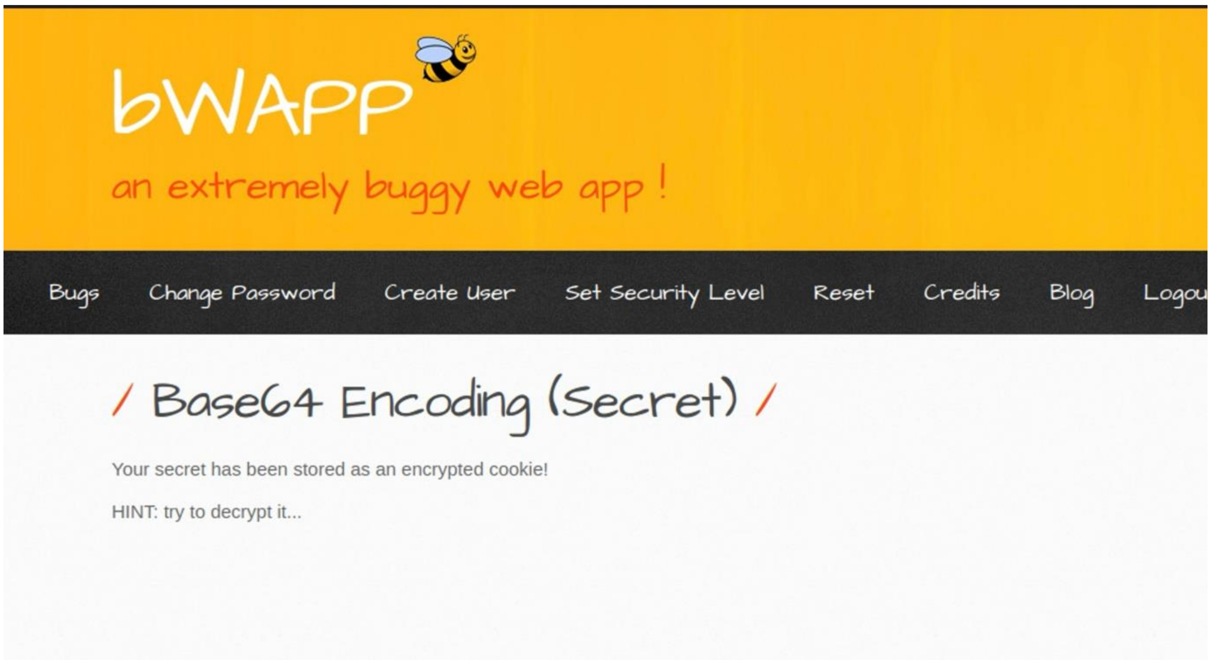
Websites can have vulnerabilities that expose your sensitive data like passwords, bank info, and personal details. This can happen due to poor data storage, weak encryption, or how the website handles what you type in. Hackers can exploit these weaknesses to steal your information, leading to identity theft, financial loss, ruined reputation, and even legal issues.

Base64 Encoding (Secret)

http://localhost/bWAPP/insecure_crypt_storage_3.php

The web page is loaded which states *"**Your secret has been stored as an encrypted cookie!**"*. We will now open burp suite to capture the request that is sent so that we can figure out a way to access the encrypted cookie. We see from the request packet we have a cookie in the header which contains an encrypted secret. When we copy the secret, we go to the decoder of the burp suite application. We will then use the base64 decoding functionality of the burp suite application to decode the secret that was present inside the cookie header.

- Screenshot Before Action

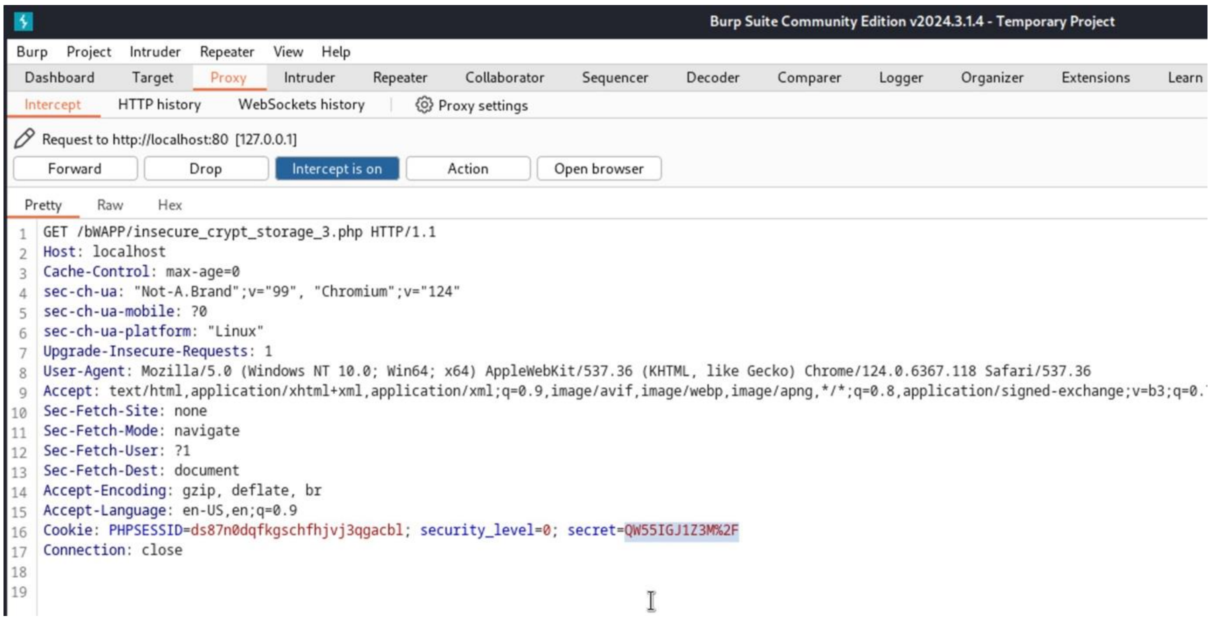


/ Base64 Encoding (Secret) /

Your secret has been stored as an encrypted cookie!

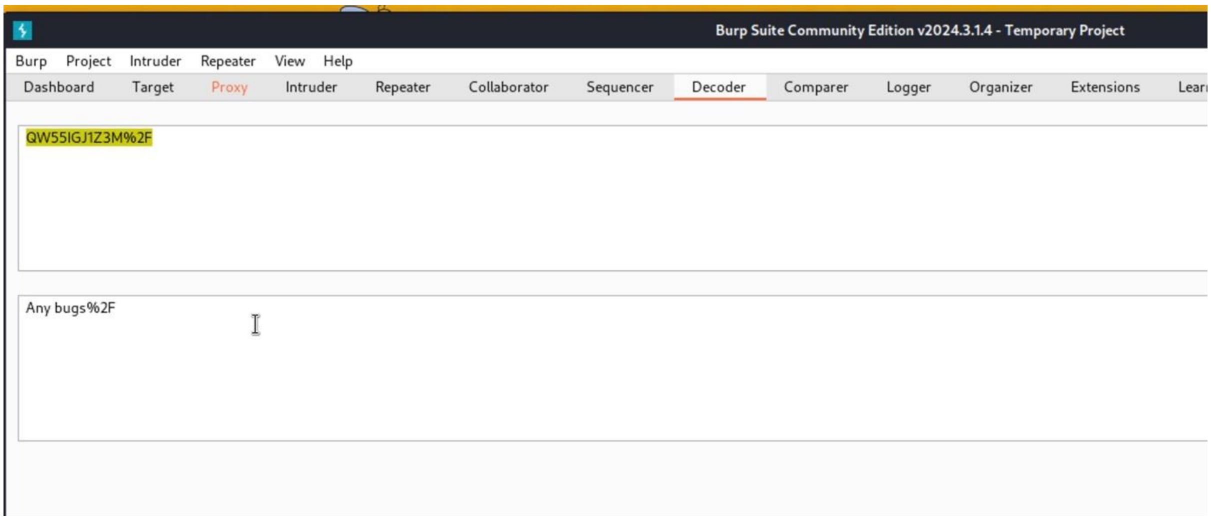
HINT: try to decrypt it...

- Screenshot After Action



Request to http://localhost:80 [127.0.0.1]

```
GET /bWAPP/insecure_crypt_storage_3.php HTTP/1.1
Host: localhost
Cache-Control: max-age=0
sec-ch-ua: "Not-A.Brand";v="99", "Chromium";v="124"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.6367.118 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=ds87n0dqfkgschfhjvj3qgacbl; security_level=0; secret=QW55IGJ1Z3M%2F
Connection: close
```


Burp Suite Community Edition v2024.3.1.4 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions

```
QW55IGJ1Z3M%2F
```

```
Any bugs%2F
```

Observation: We see that we can successfully decode the secret by using the base64 decode functionality of the burp suite application by copying the secret from the cookie header of the request packet and upon successfully decoding it prints “Any bugs?” which shows that the encryption used for the cookie secret was weak. This shows that anyone with access to the encrypted data may quickly and easily decode it to get the original data. Furthermore, Attackers can utilize Base64 encoding to conceal harmful data, sneak around input validation checks, and carry out various kinds of attacks on application users.

Preventions:

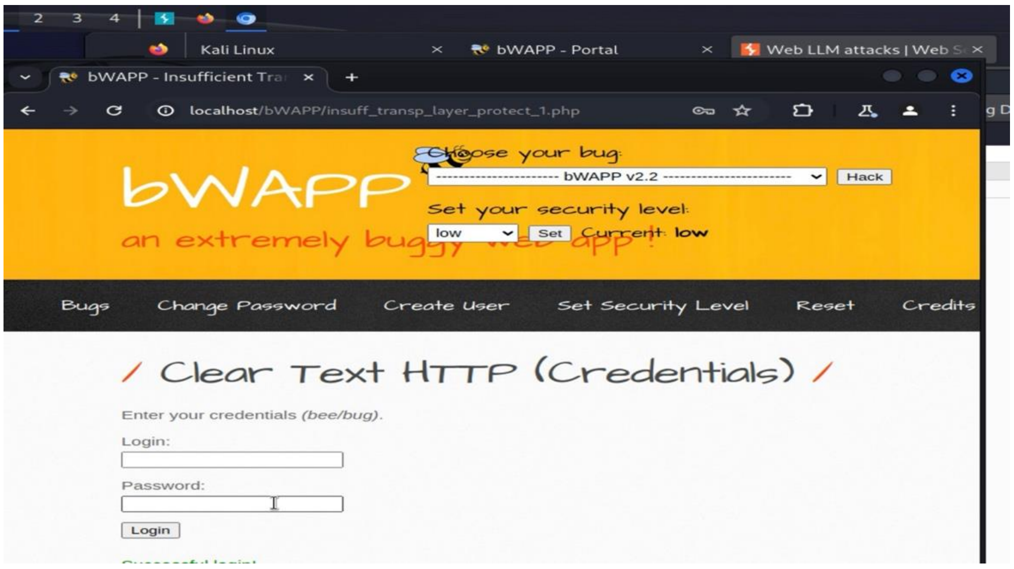
1. Use a strong encryption algorithm other than Base64 to make sure that the values cannot be decrypted easily.
2. Input validation should be done on both the encoded and decoded data.
3. When sending sensitive data over the internet, use a secure transport layer.
4. To guarantee that only authorized people have access to sensitive data, implement access control.

Clear Text HTTP (Credentials)

http://localhost/bWAPP/insuff_transp_layer_protect_1.php

The web page is loaded which displays two input fields, one for login and one for password. Now we will use a packet tracer application like burp suite to check if the login and password data that is sent back to the sever for the authentication process is properly encrypted or not.

- Screenshot Before Action



/ Clear Text HTTP (Credentials) /

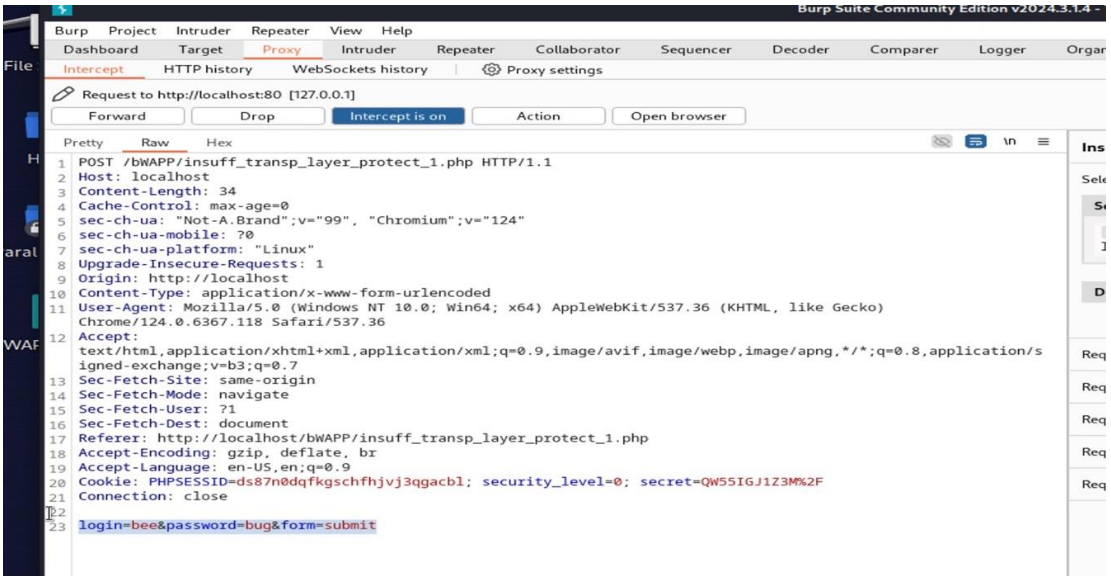
Enter your credentials (bee/bug).

Login:

Password:

Login

- Screenshot After Action



Burp Suite Community Edition v2024.3.1.4 -
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organ
Intercept HTTP history WebSockets history Proxy settings
Forward
Drop
Intercept is on
Action
Open browser
Pretty Raw Hex

Request to http://localhost:80 [127.0.0.1]

```
POST /bWAPP/insuff_transp_layer_protect_1.php HTTP/1.1
Host: localhost
Content-Length: 34
Cache-Control: max-age=0
sec-ch-ua: "Not-A.Brand";v="99", "Chromium";v="124"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/124.0.6367.118 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/bWAPP/insuff_transp_layer_protect_1.php
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=ds87n0dqfkgschfhjvj3qgacbl; security_level=0; secret=QW55IGJ1Z3M%2F
Connection: close

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

Observation: The login and password inputs transmitted in the request packet are in plain text and susceptible to interception by intruders performing packet sniffing. This poses a significant security concern for web applications, as sensitive data could be accessed by attackers during transmission, potentially leading to severe consequences such as account compromise and identity theft.

Preventions:

1. To encrypt all data sent between the client and the server, always utilize HTTPS encryption.
2. Protect user credentials from exposure by using secure password storage techniques like hashing and salting.
3. Using two-factor authentication to increase the security of user accounts.

➔ Cross-Site Scripting (XSS)

Cross-site scripting (XSS) is a security vulnerability enabling attackers to inject harmful code into a webpage viewed by others. This inserted malware can be utilized for stealing sensitive information like cookies, session tokens, and login credentials. There are two types of cross site scripting attacks:

Reflected XSS: In this type of attack, the server sends back harmful code within the URL or content to the user. When the user accesses the compromised webpage, the browser executes this malicious code, potentially granting the attacker access to confidential data or enabling them to take actions on behalf of the user.

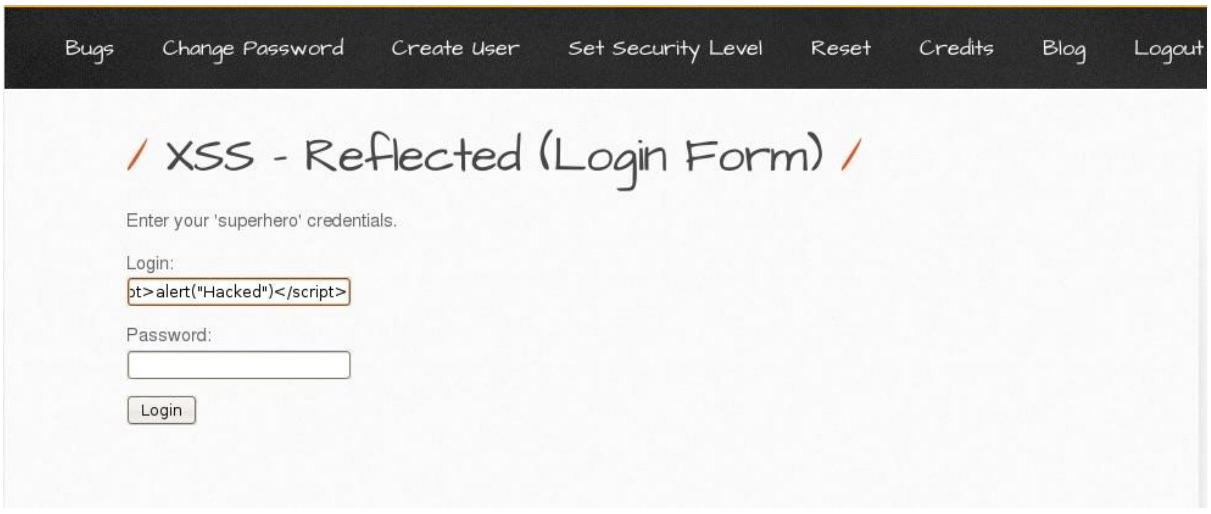
Stored XSS: Contrastingly, in a stored XSS attack, the malicious code resides on the server, typically within a database or other storage. Whenever a user accesses the compromised webpage, the code executes, posing a persistent threat as it can impact multiple users and continue functioning even after the attacker's access is removed. This type of attack is particularly damaging once successfully carried out by the attacker.

XSS – Reflected (Login Form)

http://localhost/bWAPP/xss_login.php

The web page is loaded which displays two input fields, one for login and one for password. Here we are prompted to enter your superhero credentials. If we enter invalid credentials the login fails and prints invalid credentials. However, if we input a special character, we are prompted with a message regarding an SQL syntax error which means we can perform SQL injection in the input fields which show that there is a vulnerability in the input fields. We can use the SQL error to carry out an XSS attack.

- Screenshot Before Attack



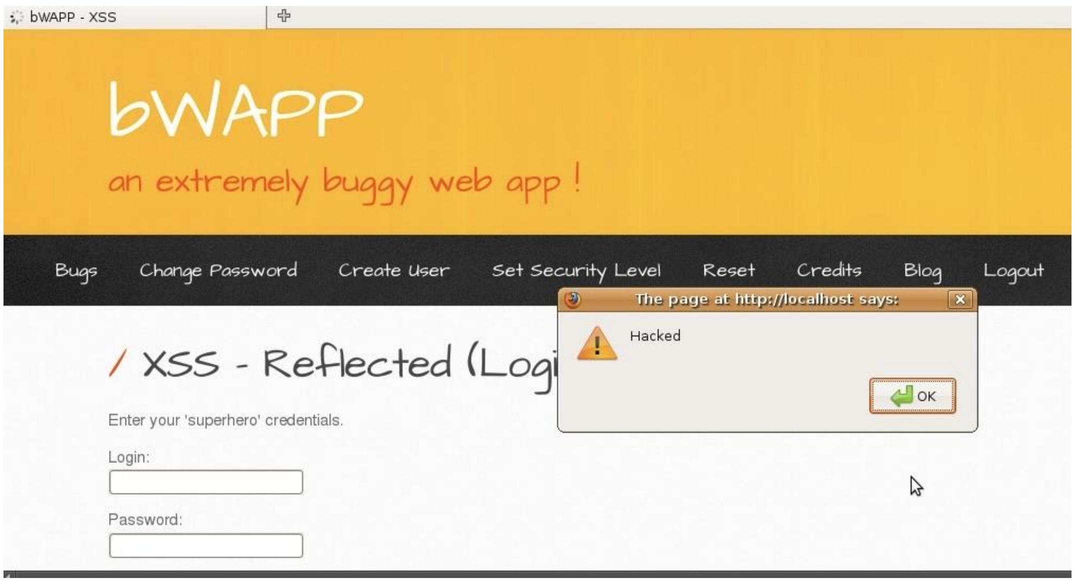
/ XSS - Reflected (Login Form) /

Enter your 'superhero' credentials.

Login: pt>alert("Hacked")</script>

Password:

- Screenshot After Attack



Hacked

Enter your 'superhero' credentials.

Login:

Password:

Observation: We see that we can bypass login form by ending the SQL query with a statement that will always be true, for example '1=1; once we enter this in the input field, we can bypass the login form and then inject the malicious code from there. In this demonstration I have executed the following piece of code in the login input field **'1=1; <script>alert("Hacked") </script>** which has executed successfully on the client's browser, and this means we can use the SQL injection vulnerability to carry out cross site scripting attack and gain access to a user's confidential information.

Preventions:

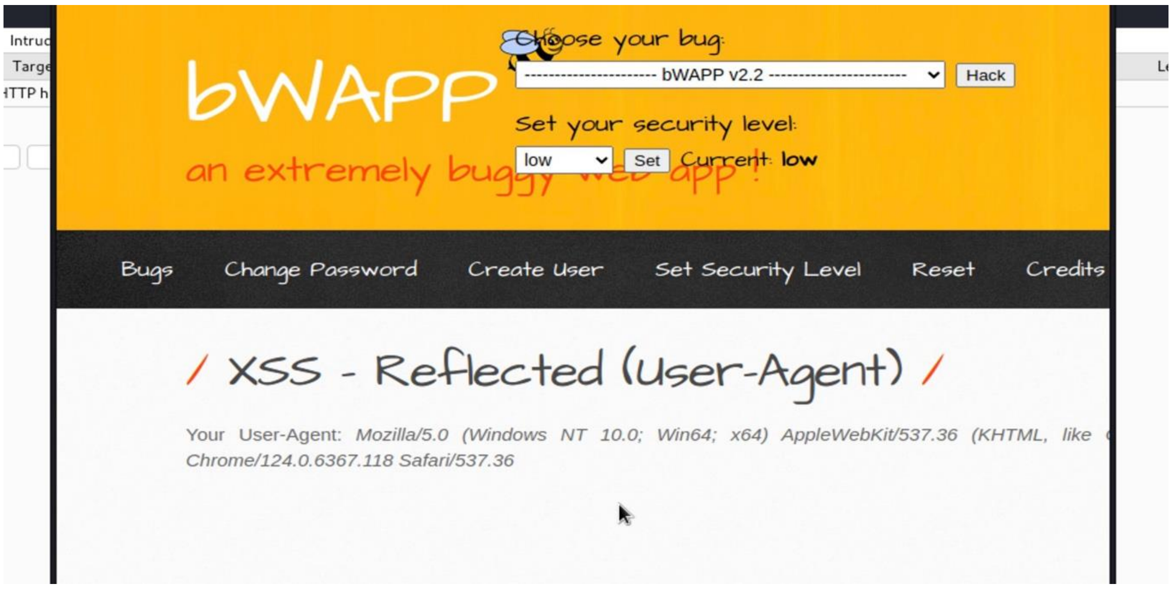
1. Using input validation techniques such as input filtering, data type validation, and length and range checking.
2. Prior to the data being shown to the user make sure to cleanse it to make sure it is free of any harmful code that could compromise the system.

XSS – Reflected (User – Agent)

http://localhost/bWAPP/xss_user_agent.php

The web page is loaded which displays the user agent header that is sent in the request packet when we load the page. It contains data about the client system like the OS version and the browser type and version. If we use a packet tracer like burp suite to read the contents of the packet, we see that we can change the values of the user agent header in the request packet, we also observe that we can edit the header to add a script and send it back to the server which is processed and is a XSS reflected vulnerability.

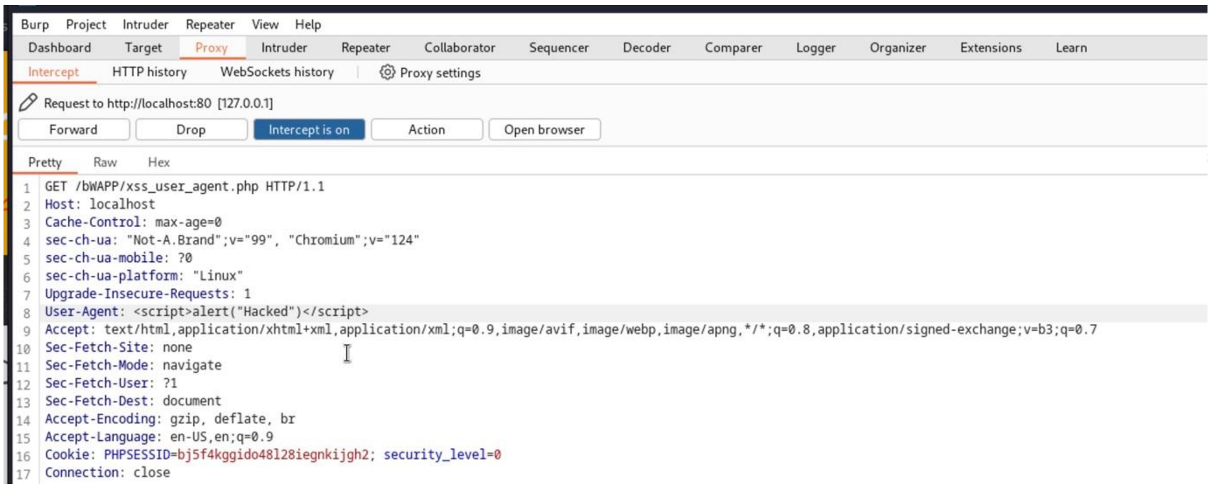
- Screenshot Before Action



/ XSS - Reflected (User-Agent) /

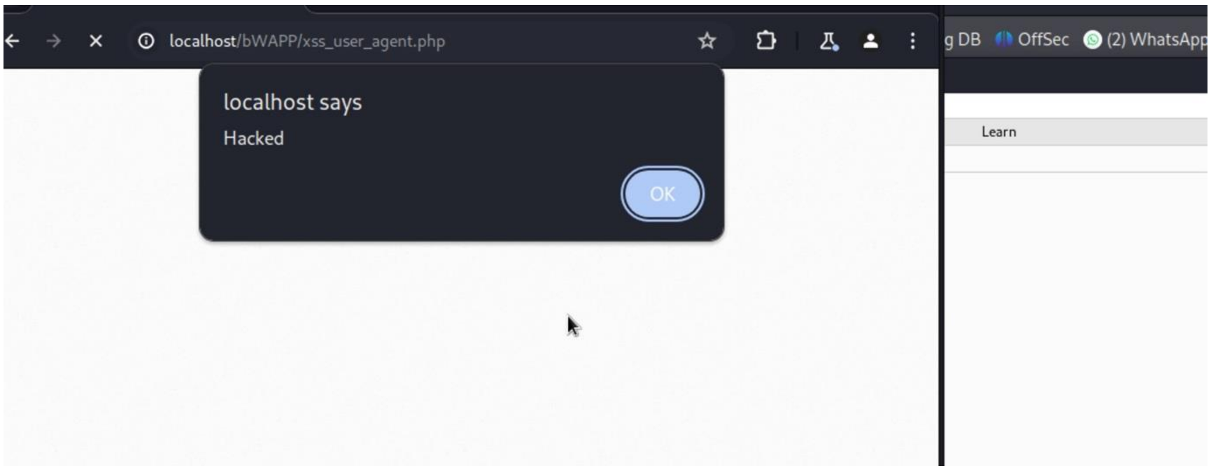
Your User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Chrome/124.0.6367.118 Safari/537.36

- Screenshot After Action


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Proxy settings
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex

Request to http://localhost:80 [127.0.0.1]

```
GET /bWAPP/xss_user_agent.php HTTP/1.1
Host: localhost
Cache-Control: max-age=0
sec-ch-ua: "Not-A.Brand";v="99", "Chromium";v="124"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: <script>alert("Hacked")</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=bj5f4kggido48l28iegnkijgh2; security_level=0
Connection: close
```



localhost says

Hacked

Observation: We've observed that injecting malicious code into the request packet header can lead to its execution in the client's browser, as confirmed by examining the response packet. This demonstrates the potential for XSS attacks by attackers who intercept network packets, thereby compromising systems and accessing sensitive data such as credit card numbers and account information.

Preventions:

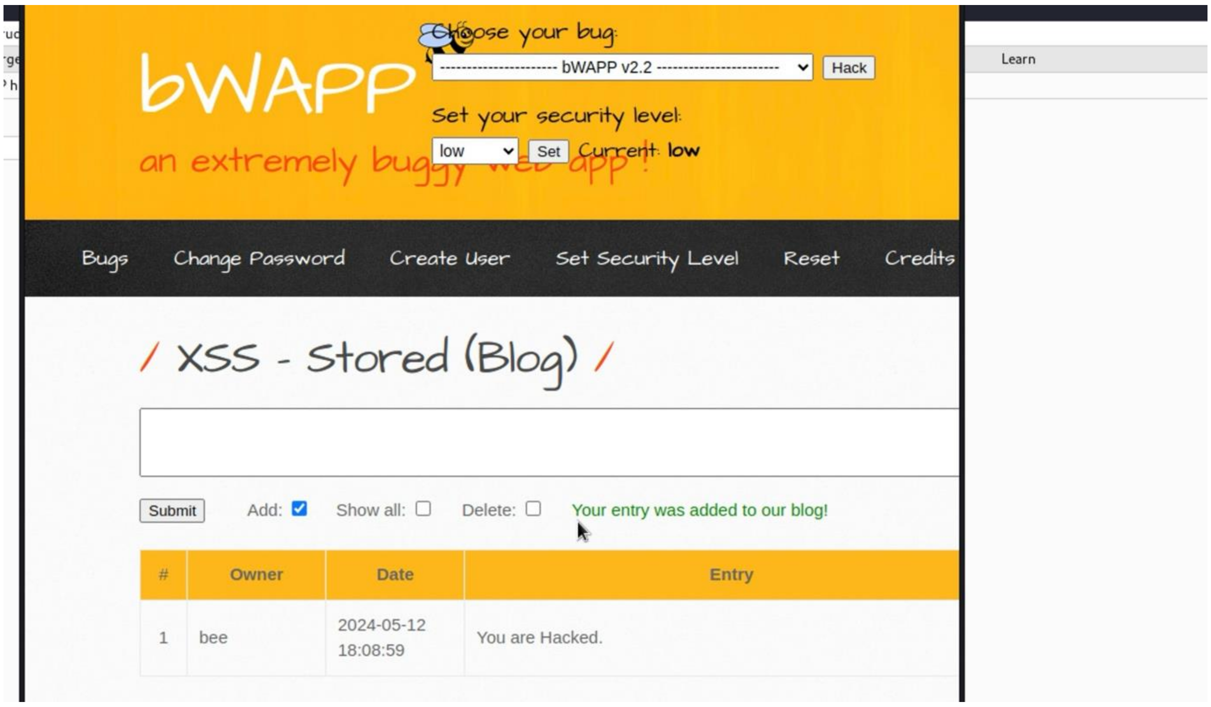
1. Encrypt all data sent between the client and server using HTTPS. Data is transferred safely and cannot be intercepted by or read by intruders thanks to HTTPS encryption.
2. To stop XSS attacks, use anti-XSS frameworks or libraries that automatically encrypt or sanitize input and output data.
3. Put in place a Content Security Policy (CSP) that enables web engineers to stop malicious code from being executed on a website.

XSS – Stored (Blog)

http://localhost/bWAPP/xss_stored_1.php

The web page is loaded which displays an input field where we can enter text which is stored in the database as a blog and can be used by other users. Here we can check if we can store a malicious script in the input field and when is displayed to a user the script is executed which results in the system being compromised.

- Screenshot Before Action

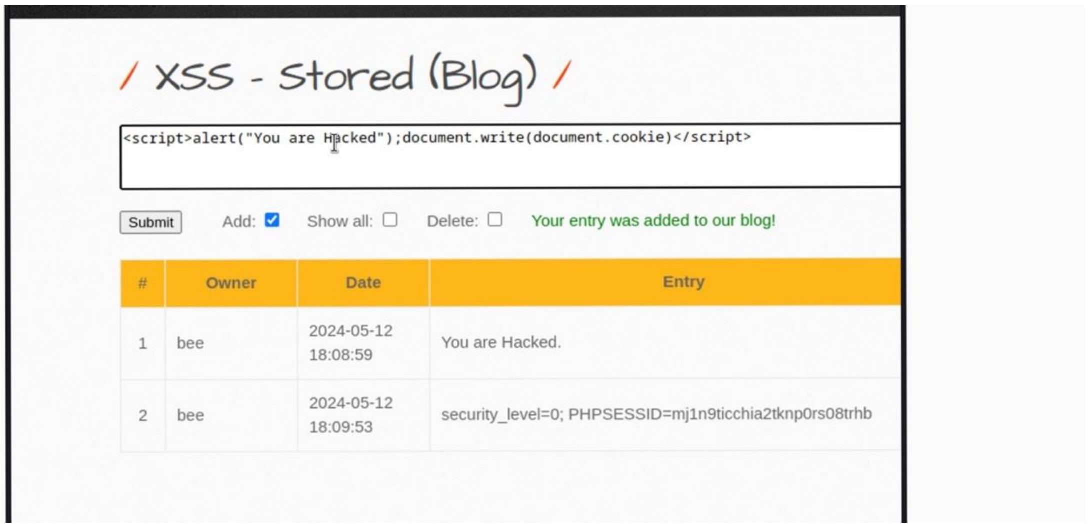


/ XSS - Stored (Blog) /

Submit Add: ✓ Show all: ☐ Delete: ☐ Your entry was added to our blog!

#	Owner	Date	Entry
1	bee	2024-05-12 18:08:59	You are Hacked.

- Screenshot After Action

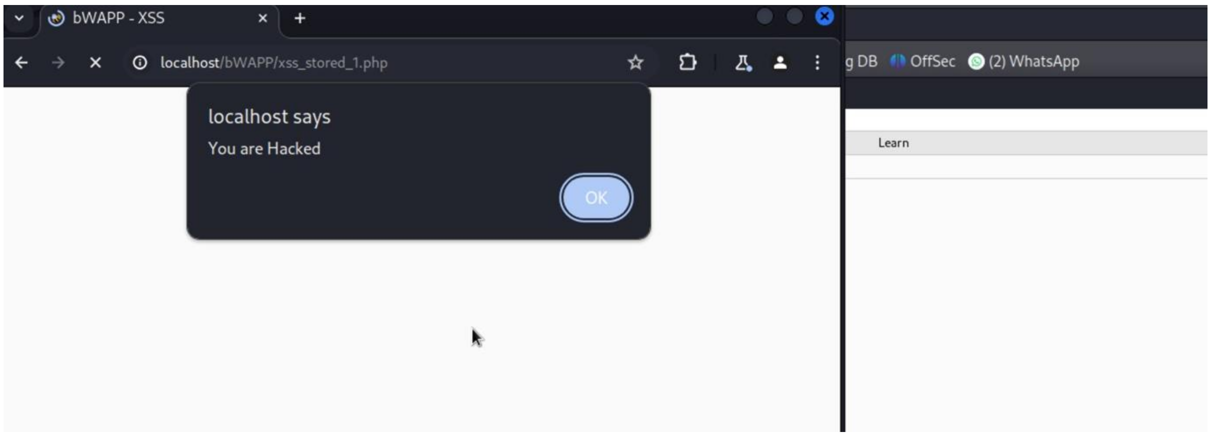


/ XSS - Stored (Blog) /

```
<script>alert("You are Hacked");document.write(document.cookie)</script>
```

Submit Add: ✓ Show all: ☐ Delete: ☐ Your entry was added to our blog!

#	Owner	Date	Entry
1	bee	2024-05-12 18:08:59	You are Hacked.
2	bee	2024-05-12 18:09:53	security_level=0; PHPSESSID=mj1n9ticchia2tknp0rs08trhb



localhost says
You are Hacked

Observation: We see that we can successfully inject a script into the input field which is stored in the blog database. When a user requests the blog webpage the script is successfully ran and the user’s system is compromised. An attacker can use this bug to steam sensitive information from a user like his bank details or session cookies, it can also be used to inject malware via the script.

Preventions:

1. To stop attackers from executing lengthy scripts, user input fields should be kept to a reasonable length so that huge scripts are not ran due to buffer overflow.
2. To guarantee that only authorized users may access and alter stored data, put strong authorization, and access control measures in place.
3. Use input validation techniques as well as sanitize the input and output when it is being stored or sent back from the database.

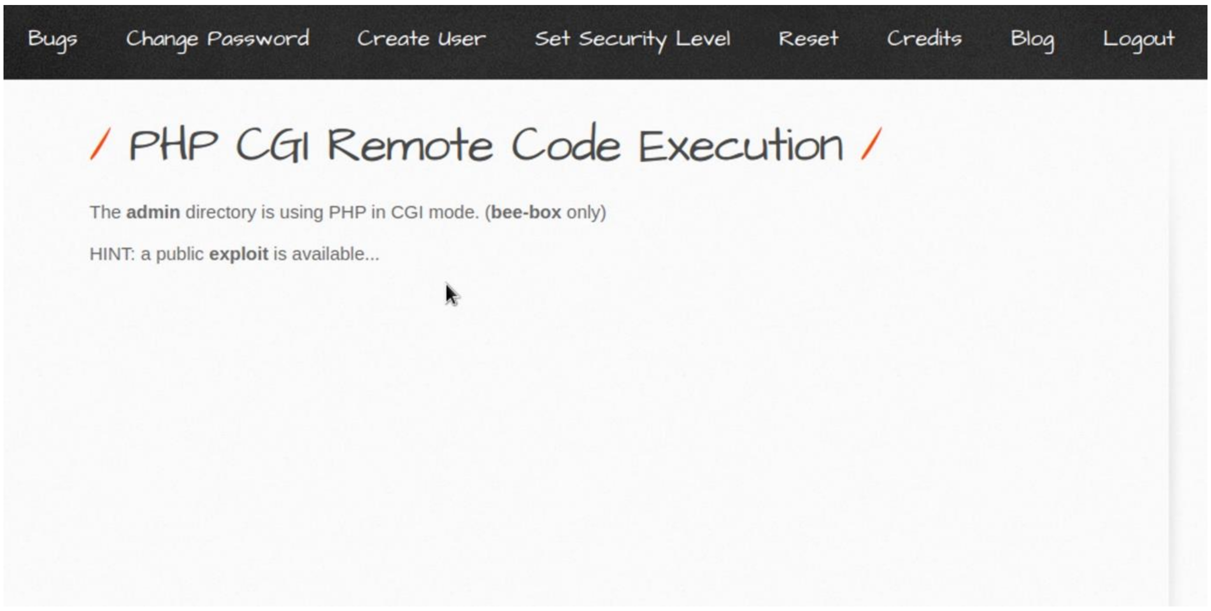
➔ Using Components with Known Vulnerabilities

An attack exploiting flaws in software components utilized within an application is termed as "using components with known vulnerabilities." This scenario often arises when a software developer integrates outdated or unpatched software libraries or components into their program. By exploiting these known weaknesses in the components, attackers can gain access to the application or its data.

PHP CGI Remote Code Execution:

http://localhost/bWAPP/php_cgi.php

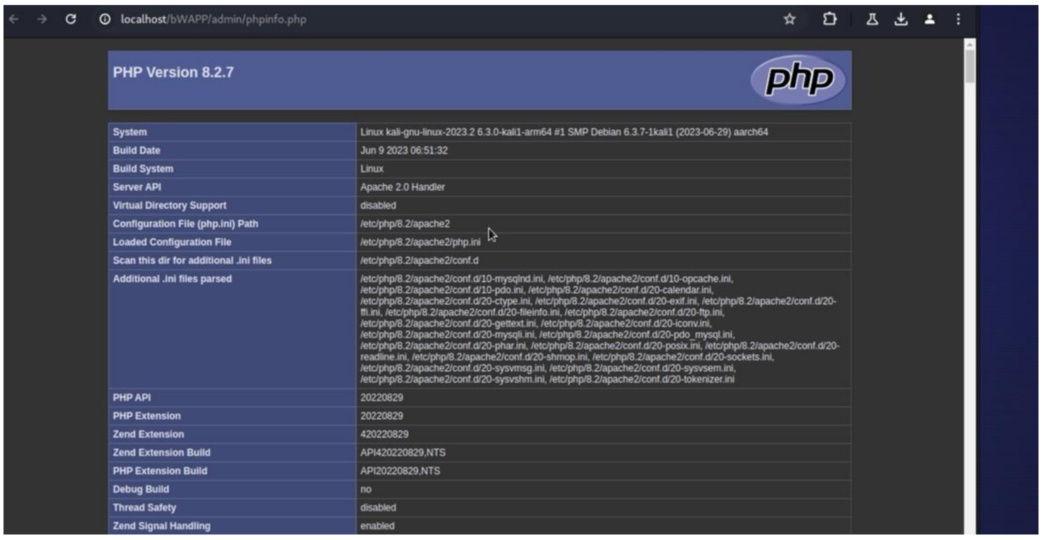
Screenshot after normal user action: By clicking on the admin, we are redirected to a php file which is present in the directory of admin/



/ PHP CGI Remote Code Execution /

The **admin** directory is using PHP in CGI mode. (**bee-box** only)

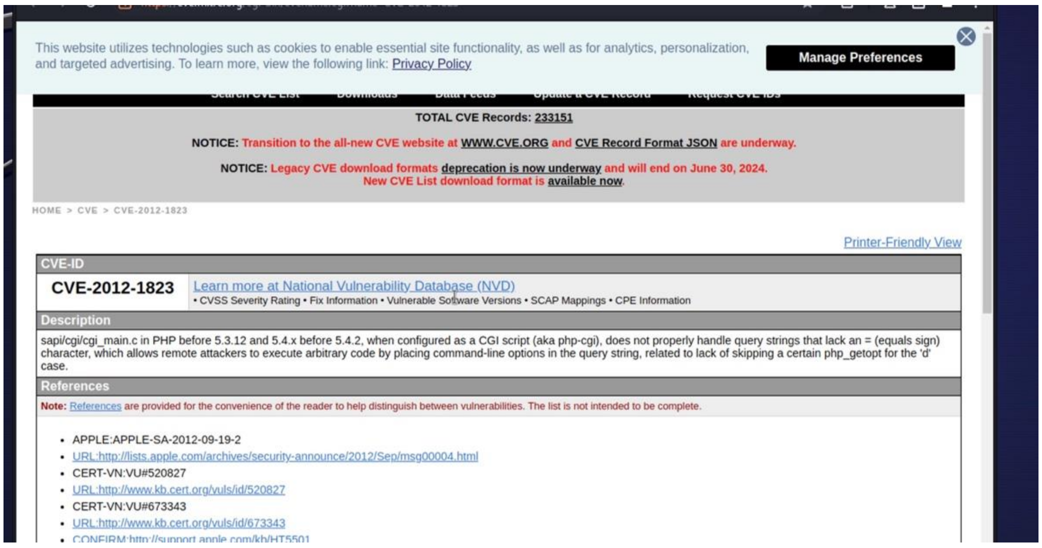
HINT: a public **exploit** is available...


localhost/bWAPP/admin/phpinfo.php

PHP Version 8.2.7

System	Linux kali-gnu-linux-2023.2 6.3.0-kali1-arm64 #1 SMP Debian 6.3.7-1kali1 (2023-06-29) aarch64
Build Date	Jun 9 2023 06:51:32
Build System	Linux
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/8.2/apache2
Loaded Configuration File	/etc/php/8.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/8.2/apache2/conf.d
Additional .ini files parsed	/etc/php/8.2/apache2/conf.d/10-mysqlnd.ini, /etc/php/8.2/apache2/conf.d/10-opcache.ini, /etc/php/8.2/apache2/conf.d/10-pdo.ini, /etc/php/8.2/apache2/conf.d/20-calendar.ini, /etc/php/8.2/apache2/conf.d/20-ctype.ini, /etc/php/8.2/apache2/conf.d/20-exif.ini, /etc/php/8.2/apache2/conf.d/20-ffi.ini, /etc/php/8.2/apache2/conf.d/20-fileinfo.ini, /etc/php/8.2/apache2/conf.d/20-ftp.ini, /etc/php/8.2/apache2/conf.d/20-gettext.ini, /etc/php/8.2/apache2/conf.d/20-iconv.ini, /etc/php/8.2/apache2/conf.d/20-mysqli.ini, /etc/php/8.2/apache2/conf.d/20-pdo_mysql.ini, /etc/php/8.2/apache2/conf.d/20-phar.ini, /etc/php/8.2/apache2/conf.d/20-posix.ini, /etc/php/8.2/apache2/conf.d/20-readline.ini, /etc/php/8.2/apache2/conf.d/20-shmop.ini, /etc/php/8.2/apache2/conf.d/20-sockets.ini, /etc/php/8.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/8.2/apache2/conf.d/20-sysvsem.ini, /etc/php/8.2/apache2/conf.d/20-sysvshm.ini, /etc/php/8.2/apache2/conf.d/20-tokenizer.ini
PHP API	20220829
PHP Extension	20220829
Zend Extension	420220829
Zend Extension Build	API420220829,NTS
PHP Extension Build	API20220829,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled

Screenshot after taking action: By clicking on the exploit, we came to know how this version of PHP does not properly handle query strings that lack an = (equals sign) character, which allows remote attackers to execute arbitrary code by placing command-line options in the query string


This website utilizes technologies such as cookies to enable essential site functionality, as well as for analytics, personalization, and targeted advertising. To learn more, view the following link: Privacy Policy

TOTAL CVE Records: 233151

NOTICE: Transition to the all-new CVE website at WWW.CVE.ORG and CVE Record Format JSON are underway.

NOTICE: Legacy CVE download formats deprecation is now underway and will end on June 30, 2024.
New CVE List download format is available now.

HOME > CVE > CVE-2012-1823

Printer-Friendly View

CVE-ID	
CVE-2012-1823	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

sapi/cgi/cgi_main.c in PHP before 5.3.12 and 5.4.x before 5.4.2, when configured as a CGI script (aka php-cgi), does not properly handle query strings that lack an = (equals sign) character, which allows remote attackers to execute arbitrary code by placing command-line options in the query string, related to lack of skipping a certain php_getopt for the 'd' case.

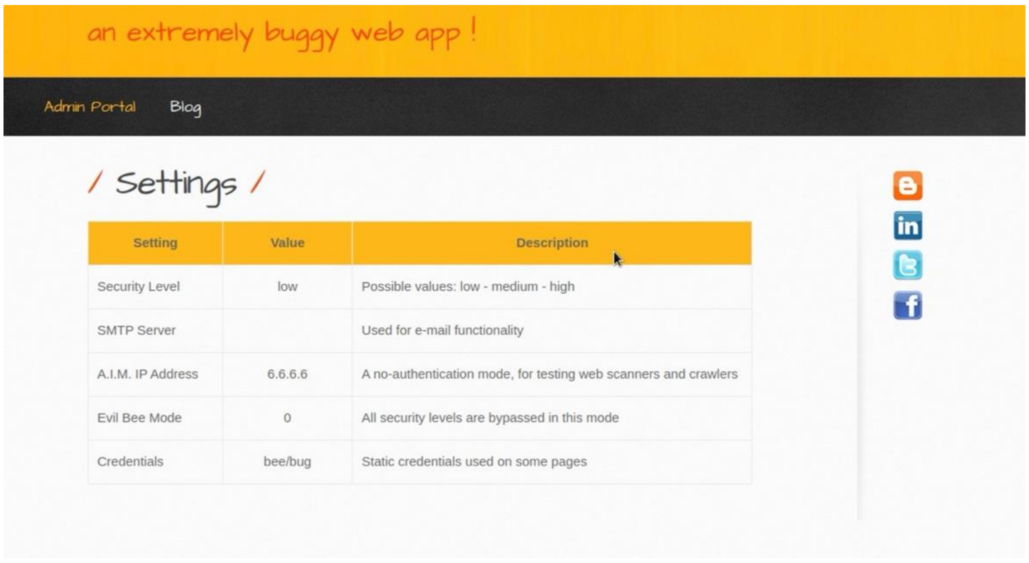
References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- APPLE:APPLE-SA-2012-09-19-2
- URL:http://lists.apple.com/archives/security-announce/2012/Sep/msg00004.html
- CERT-VN:VU#520827
- URL:http://www.kb.cert.org/vuls/id/520827
- CERT-VN:VU#673343
- URL:http://www.kb.cert.org/vuls/id/673343
- CONFIRM:http://support.apple.com/kb/HT5501

From a website from its references **“https://www.kb.cert.org/vuls/id/520827”**, I got some more hints to how to exploit this vulnerability. An example of the -s command, allowing an attacker to view the source code of index.php is:

http://localhost/index.php?-s. We implemented this to our bWAPP local host and I was redirected to admin portal.



Setting	Value	Description
Security Level	low	Possible values: low - medium - high
SMTP Server		Used for e-mail functionality
A.I.M. IP Address	6.6.6.6	A no-authentication mode, for testing web scanners and crawlers
Evil Bee Mode	0	All security levels are bypassed in this mode
Credentials	bee/bug	Static credentials used on some pages

Observation: The screenshot shows we're in the admin portal settings, where confidential info is visible. A remote attacker without authentication could access sensitive data, cause a service denial, or run code with web server privileges. Adding "-s" to any vulnerable PHP file reveals its source code, exploited to run arbitrary code on the server.

Prevention: PHP has released version 5.4.3 and 5.3.13 to address this vulnerability. PHP is recommending that users upgrade to the latest version of PHP. PHP has stated an alternative is to configure your web server to not let these types of requests with query strings starting with a "-" and not containing a "=" through.

➔ Security Misconfiguration

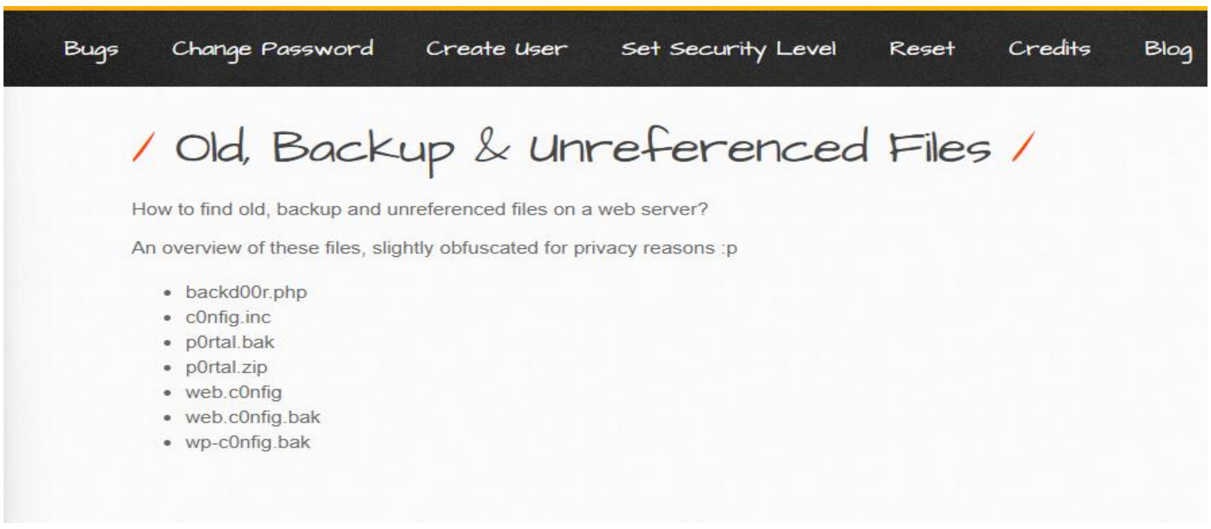
A "security misconfiguration attack" exploits system flaws caused by incorrect setup of software and hardware components. Cybercriminals often exploit default settings, unpatched software, open ports, and weak passwords to carry out these attacks.

Old, Backup & Unreferenced Files

http://localhost/bWAPP/sm_obu_files.php

The web page is loaded in which we must file old, backup, or unreferenced files in the web server. If we can do this then this means, there is a security vulnerability in the code which can be exploited by hackers.

- Screenshot Before Action



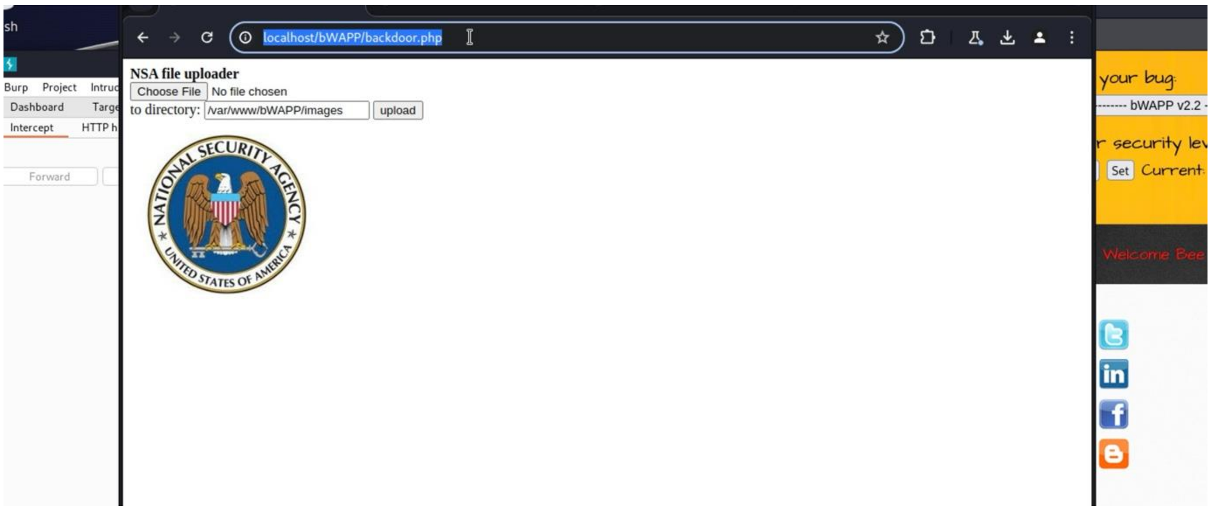
/ Old, Backup & Unreferenced Files /

How to find old, backup and unreferenced files on a web server?

An overview of these files, slightly obfuscated for privacy reasons :p

- backd00r.php
- c0nfig.inc
- p0rtal.bak
- p0rtal.zip
- web.c0nfig
- web.c0nfig.bak
- wp-c0nfig.bak

- Screenshot After Action


localhost/bWAPP/backdoor.php

NSA file uploader

Choose File No file chosen

to directory: /var/www/bWAPP/images upload

Observation: The server admin has altered characters in file names to hide them, but with an overview, one can modify names to access unreferenced files. Attackers use automated tools to find these files, exploiting them to gain unauthorized access. For instance, they might find unprotected backup files or configuration files containing sensitive information like passwords.

Preventions:

1. Review your systems frequently to spot and remove any files that are no longer necessary.
2. To make sure that files containing sensitive information cannot be recovered, delete them using secure deletion techniques.

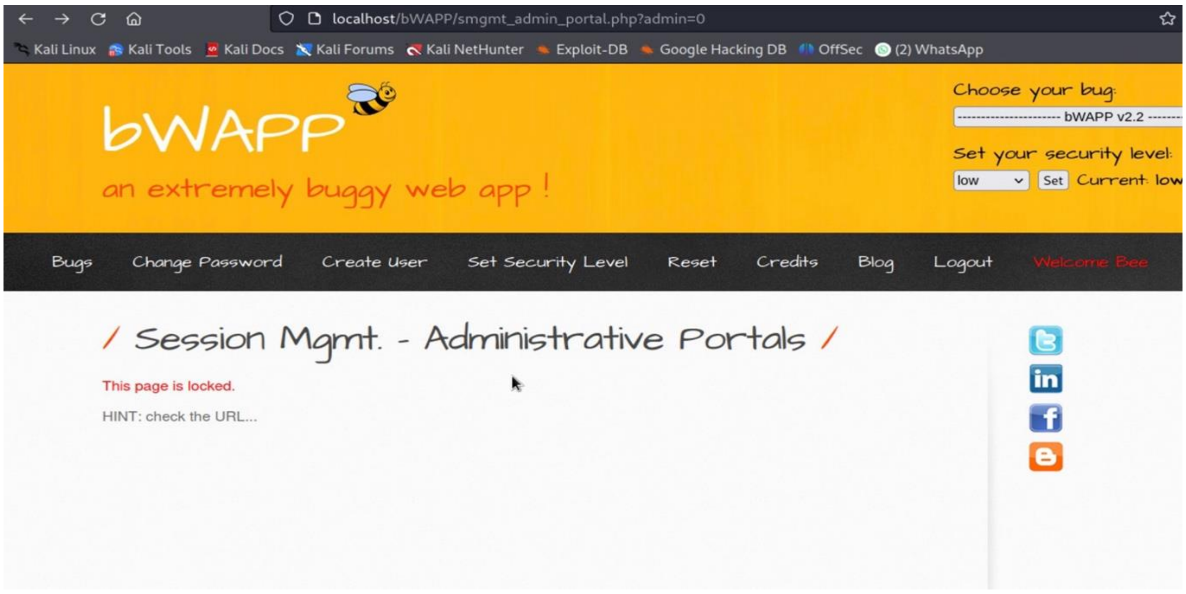
➔ Broken Access Control:

http://localhost/bWAPP/smgmt_admin_portal.php?admin=0

Administrative Portals

Broken access control (BAC) happens when a system fails to properly regulate how users access data and functionalities.

- Screenshot Before action


localhost/bWAPP/smgmt_admin_portal.php?admin=0

Set your security level: low Set Current: low

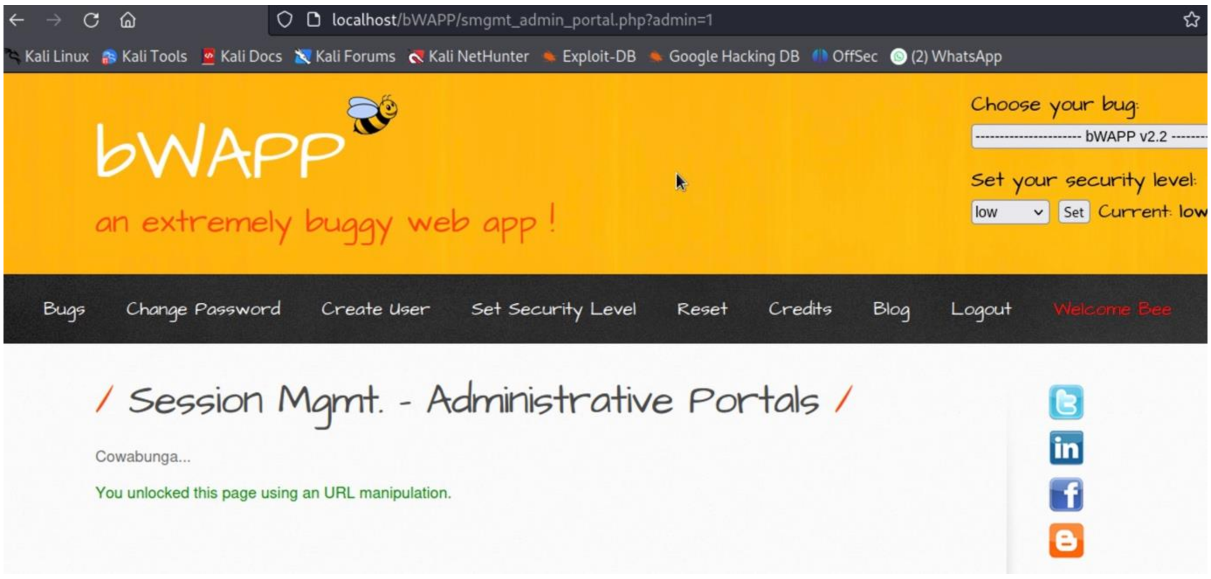
/ Session Mgmt. - Administrative Portals /

This page is locked.

HINT: check the URL...

Here we could not access the Administrative Portals.

- Screenshot after action


localhost/bWAPP/smgmt_admin_portal.php?admin=1

Set your security level: low Set Current: low

/ Session Mgmt. - Administrative Portals /

Cowabunga...

You unlocked this page using an URL manipulation.

Observation: But here after manipulating the URL we unlocked the administrative portals by gaining access to it. Changed the end part from admin=0 to admin=1.

Preventions:

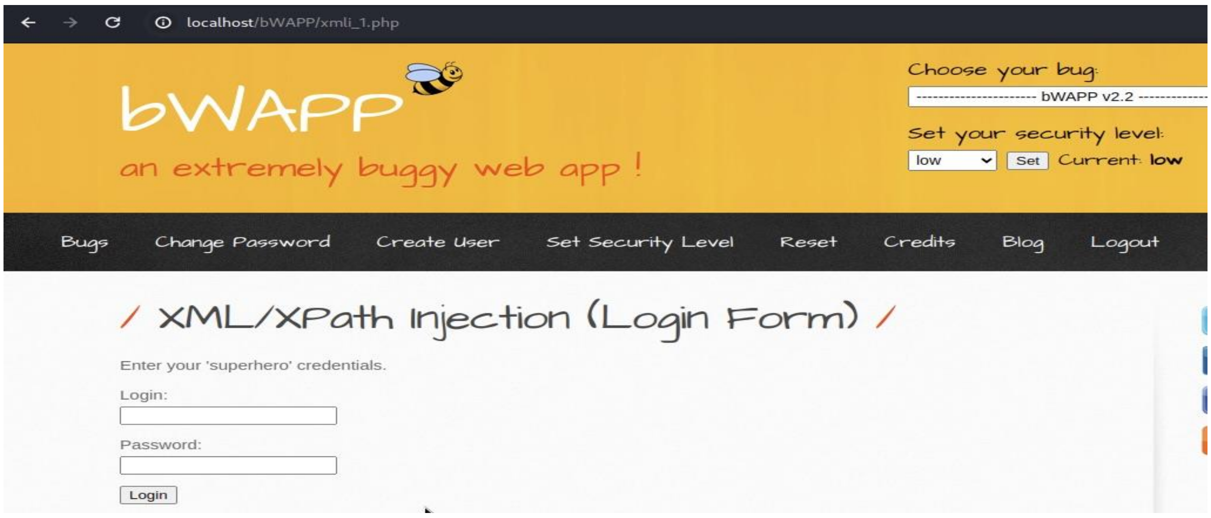
- Use Strong Authentication: Implement multi-factor authentication (MFA) to add an extra layer of security beyond just usernames and passwords.
- Session Timeouts: Set session timeouts for user inactivity to prevent unauthorized access if someone leaves their session open.
- Invalidate Sessions: Implement mechanisms to invalidate sessions when users log out or after a certain period of inactivity.
- Secure Session IDs: Use strong random numbers for session IDs and store them securely (e.g., with encryption).

➔ XML/XPath injection (login Form)

http://localhost/bWAPP/xmli_1.php

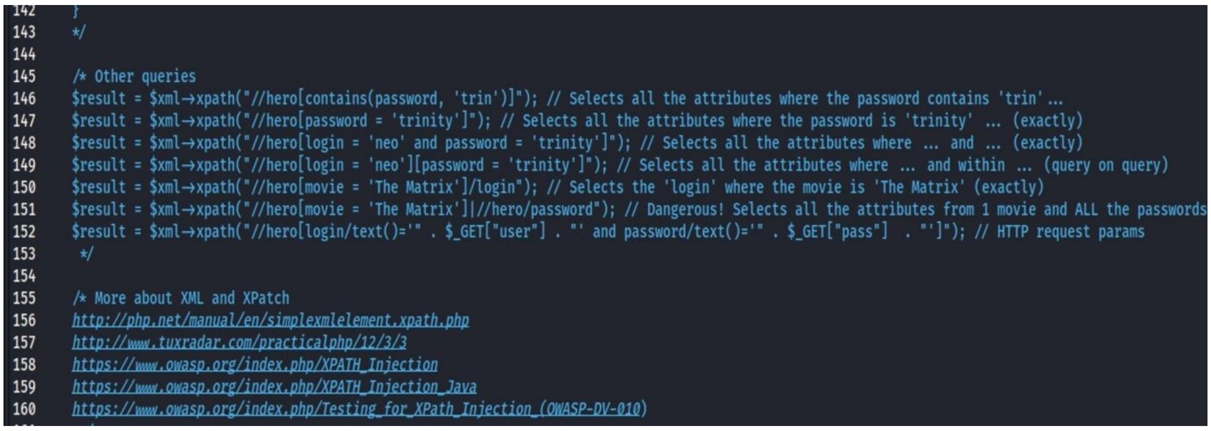
XML XPath injection is a type of security vulnerability that occurs when an application uses user-supplied data to construct an XPath query for processing XML data. If the application doesn't properly sanitize this user input, an attacker can inject malicious code into the XPath query.

- Screenshot before action



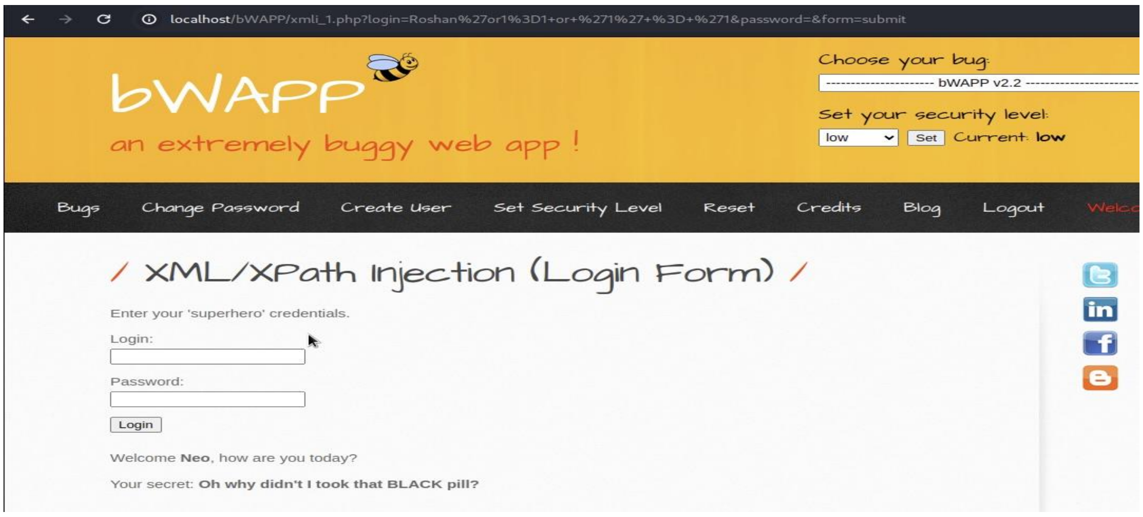
- Screenshot after action

Here we inspected the page to find out the XML code and the XPath query pointing towards it. The XML code is human readable, and it contained the username and password for the superhero credentials required for the login page.



```
    }
    */

    /* Other queries
    $result = $xml->xpath("//hero[contains(password, 'trin')]"); // Selects all the attributes where the password contains 'trin'...
    $result = $xml->xpath("//hero[password = 'trinity']"); // Selects all the attributes where the password is 'trinity' ... (exactly)
    $result = $xml->xpath("//hero[login = 'neo' and password = 'trinity']"); // Selects all the attributes where ... and ... (exactly)
    $result = $xml->xpath("//hero[login = 'neo'][password = 'trinity']"); // Selects all the attributes where ... and within ... (query on query)
    $result = $xml->xpath("//hero[movie = 'The Matrix']/login"); // Selects the 'login' where the movie is 'The Matrix' (exactly)
    $result = $xml->xpath("//hero[movie = 'The Matrix']|//hero/password"); // Dangerous! Selects all the attributes from 1 movie and ALL the passwords
    $result = $xml->xpath("//hero[login/text()='" . $_GET["user"] . "' and password/text()='" . $_GET["pass"]  . "']"); // HTTP request params
     */

    /* More about XML and XPatch
    http://php.net/manual/en/simplexmlelement.xpath.php
    http://www.tuxradar.com/practicalphp/12/3/3
    https://www.owasp.org/index.php/XPATH_Injection
    https://www.owasp.org/index.php/XPATH_Injection_Java
    https://www.owasp.org/index.php/Testing_for_XPath_Injection_(OWASP-DV-010)
```

Now after getting the username: “neo” AND password: “trinity”. We logged in the account with the superhero credentials.



localhost/bWAPP/xmli_1.php?login=Roshan%27or1%3D1+or+%271%27+%3D+%271&password=&form=submit

/ XML/XPath Injection (Login Form) /

Enter your 'superhero' credentials.

Login:

Password:

Welcome **Neo**, how are you today?

Your secret: **Oh why didn't I took that BLACK pill?**

Preventions:

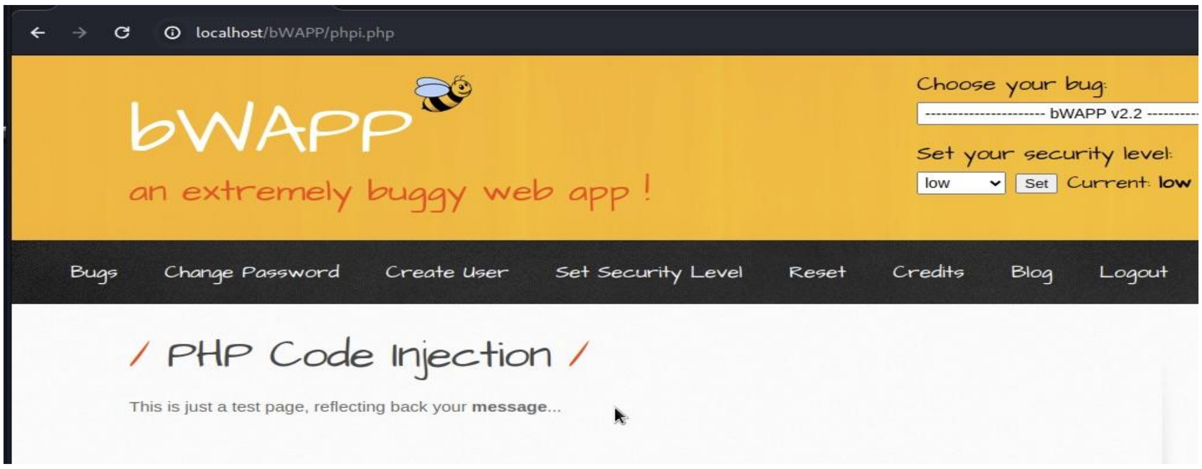
- Input Validation and Sanitization: Filter out harmful characters (', ", <, >). Encode special characters (&lt;, &amp;). Consider whitelisting allowed characters.
- Prepared Statements or Parameterized Queries: Separate data from the XPath query. Use placeholders for user input.
- Least Privilege Principle: Minimize user access to XML data manipulation.
- Regular Security Testing: Conduct penetration testing to identify vulnerabilities.

➔ Php code injection:

http://localhost/bWAPP/phpi.php

PHP code injection is a web security vulnerability that arises when an application incorporates untrusted user input into dynamically evaluated code. This allows attackers to inject malicious code that gets executed by the server on the user's behalf.

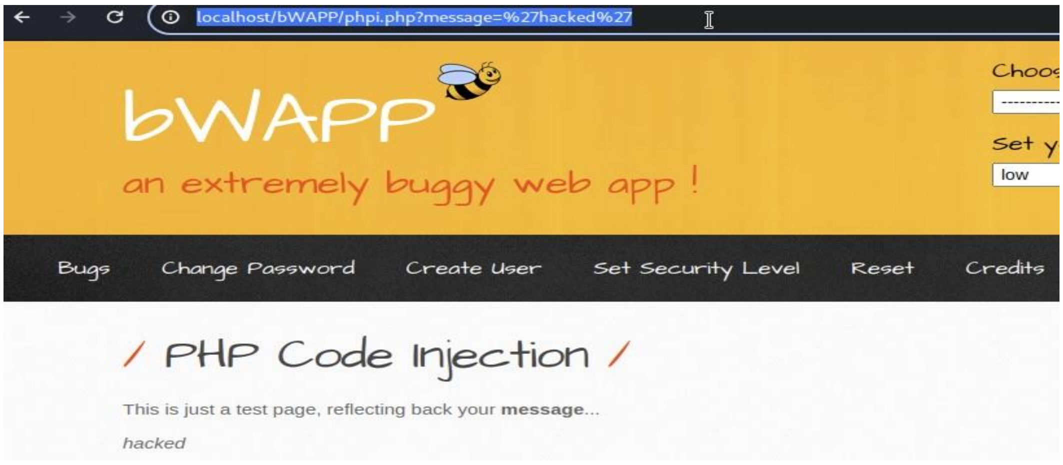
- Screenshot before action



/ PHP Code Injection /

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

- Screenshot after action



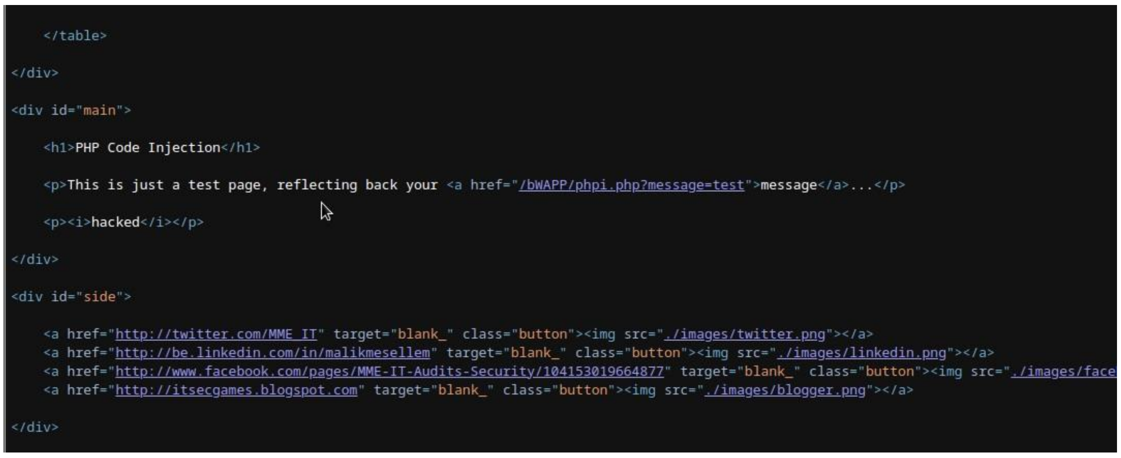
localhost/bWAPP/phpi.php?message=%27hacked%27
bWAPP
an extremely buggy web app !
low
Bugs
Change Password
Create User
Set Security Level
Reset
Credits

/ PHP Code Injection /

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

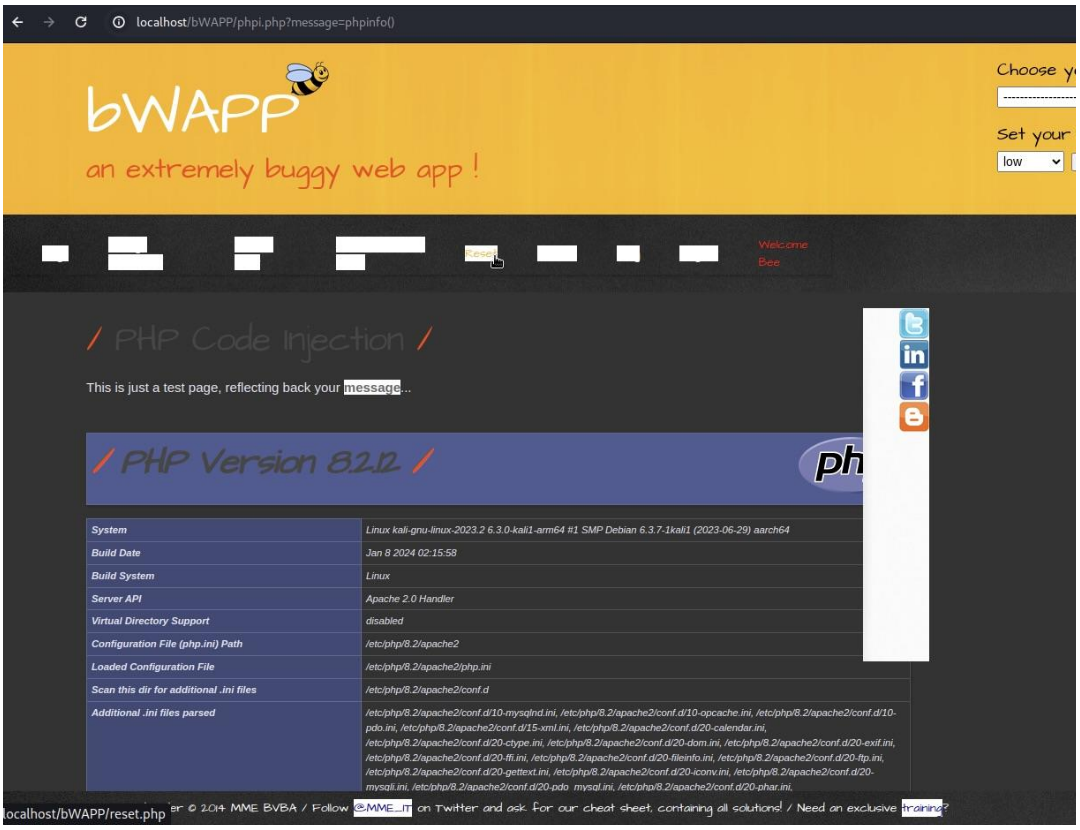
hacked

After clicking the message on the page. There is no message shown. We have to manipulate the URL in order to show a message. The URL was manipulated at the end → message = hacked. Now by clinking on the message again we can see that our message “hacked” has been shown on the page.



```
    </table>

</div>

<div id="main">

    <h1>PHP Code Injection</h1>

    <p>This is just a test page, reflecting back your <a href="/bWAPP/phpi.php?message=test">message</a>...</p>

    <p><i>hacked</i></p>

</div>

<div id="side">

    <a href="http://twitter.com/MME_IT" target="blank_" class="button"><img src="./images/twitter.png"></a>
    <a href="http://be.linkedin.com/in/malikmesellem" target="blank_" class="button"><img src="./images/linkedin.png"></a>
    <a href="http://www.facebook.com/pages/MME-IT-Audits-Security/104153019664877" target="blank_" class="button"><img src="./images/face
    <a href="http://itsecgames.blogspot.com" target="blank_" class="button"><img src="./images/blogger.png"></a>

</div>
```

After inspecting the page, we found out that the html code also has changed the message to “hacked”. Similarly, it can be exploited in many ways. One such way is by php.info(). Which gives the whole php information.


localhost/bWAPP/phpi.php?message=phpinfo()

/ PHP Code Injection /

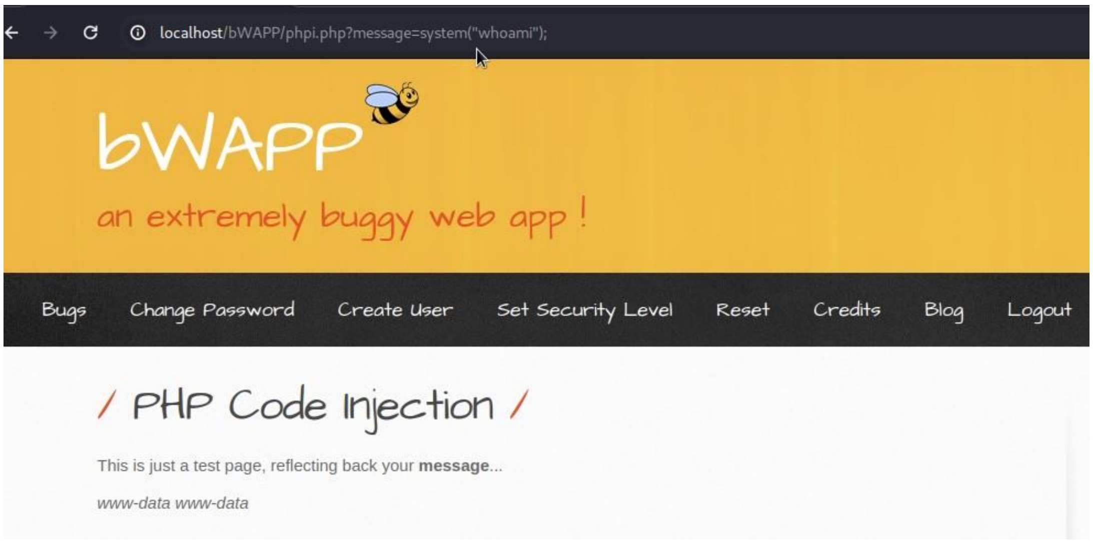
This is just a test page, reflecting back your message...

/ PHP Version 8.2.12 /

System	Linux kali-gnu-linux-2023.2 6.3.0-kali1-arm64 #1 SMP Debian 6.3.7-1kali1 (2023-06-29) aarch64
Build Date	Jan 8 2024 02:15:58
Build System	Linux
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/8.2/apache2
Loaded Configuration File	/etc/php/8.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/8.2/apache2/conf.d
Additional .ini files parsed	/etc/php/8.2/apache2/conf.d/10-mysqlnd.ini, /etc/php/8.2/apache2/conf.d/10-opcache.ini, /etc/php/8.2/apache2/conf.d/10-pdo.ini, /etc/php/8.2/apache2/conf.d/15-xml.ini, /etc/php/8.2/apache2/conf.d/20-calendar.ini, /etc/php/8.2/apache2/conf.d/20-ctype.ini, /etc/php/8.2/apache2/conf.d/20-dom.ini, /etc/php/8.2/apache2/conf.d/20-exif.ini, /etc/php/8.2/apache2/conf.d/20-ffi.ini, /etc/php/8.2/apache2/conf.d/20-fileinfo.ini, /etc/php/8.2/apache2/conf.d/20-ftp.ini, /etc/php/8.2/apache2/conf.d/20-gettext.ini, /etc/php/8.2/apache2/conf.d/20-iconv.ini, /etc/php/8.2/apache2/conf.d/20-mysqli.ini, /etc/php/8.2/apache2/conf.d/20-pdo_mysql.ini, /etc/php/8.2/apache2/conf.d/20-phar.ini,

localhost/bWAPP/reset.php er © 2014 MME BVBA / Follow @MME_IT on Twitter and ask for our cheat sheet, containing all solutions! / Need an exclusive training?

Similarly, it can be exploited in many ways. One such way is by using the system clause. Message = system(“whoami”); It provides the information about the user running the application.

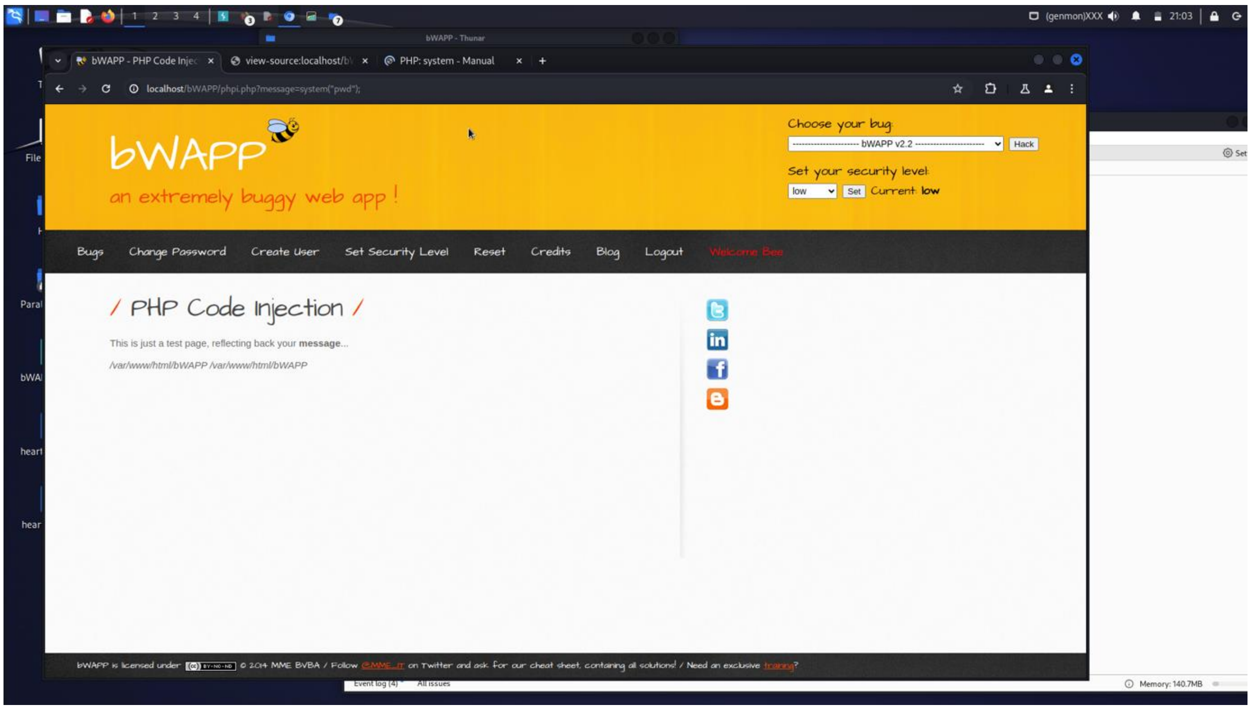

localhost/bWAPP/phpi.php?message=system("whoami");

/ PHP Code Injection /

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

www-data www-data

Similarly, it can be exploited in many ways. One such way is by using the system clause. Message = system("pwd"); It means print working directory. It outputs the current working directory.

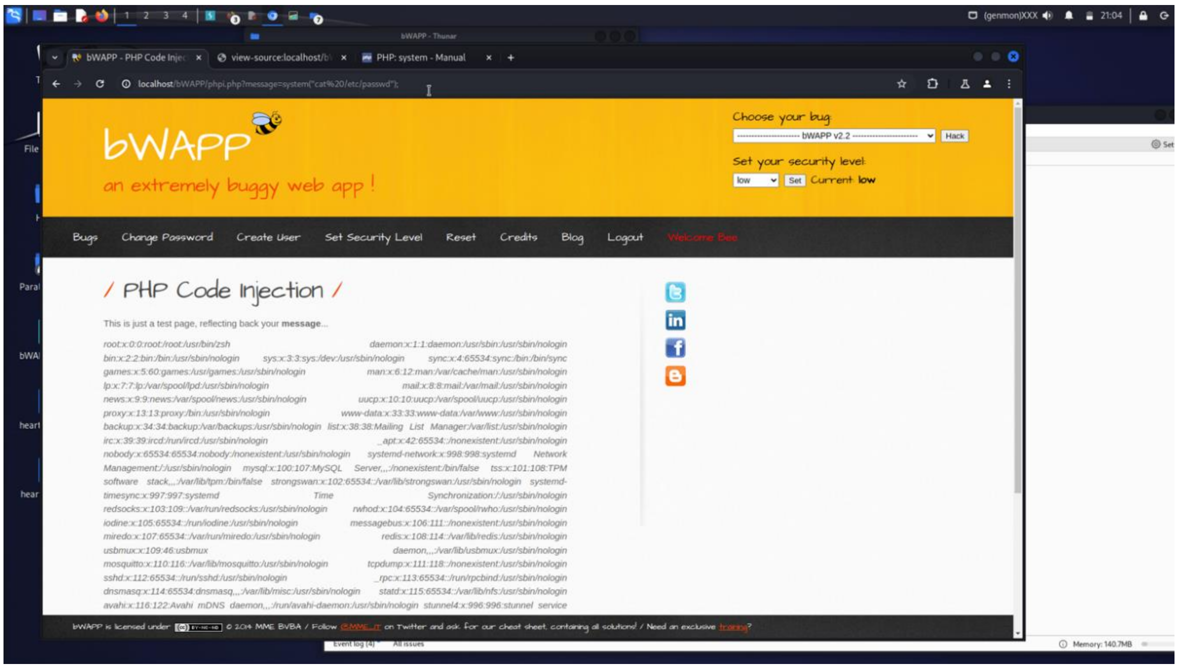

localhost/bWAPP/phpi.php?message=system("pwd");

/ PHP Code Injection /

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

```
/var/www/html/bWAPP /var/www/html/bWAPP
```

Similarly, it can be exploited in many ways. One such way is by using the system clause. Message = system("cat 20/etc/passwd"); To reveal the contents of the /etc/passwd file on a Linux or Unix system.


localhost/bWAPP/phpi.php?message=system("cat%20/etc/passwd");

/ PHP Code Injection /

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

```
root:x:0:0:root:/root:/usr/bin/zsh daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:998:998:systemd Network
Management:/:/usr/sbin/nologin mysql:x:100:107:MySQL Server,,,:/nonexistent:/bin/false tss:x:101:108:TPM
software stack,,,:/var/lib/tpm:/bin/false strongswan:x:102:65534::/var/lib/strongswan:/usr/sbin/nologin systemd-
timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin
redsocks:x:103:109::/var/run/redsocks:/usr/sbin/nologin rwhod:x:104:65534::/var/spool/rwho:/usr/sbin/nologin
iodine:x:105:65534::/run/iodine:/usr/sbin/nologin messagebus:x:106:111::/nonexistent:/usr/sbin/nologin
miredo:x:107:65534::/var/run/miredo:/usr/sbin/nologin redis:x:108:114::/var/lib/redis:/usr/sbin/nologin
usbmux:x:109:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
mosquitto:x:110:116::/var/lib/mosquitto:/usr/sbin/nologin tcpdump:x:111:118::/nonexistent:/usr/sbin/nologin
sshd:x:112:65534::/run/sshd:/usr/sbin/nologin _rpc:x:113:65534::/run/rpcbind:/usr/sbin/nologin
dnsmasq:x:114:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin statd:x:115:65534::/var/lib/nfs:/usr/sbin/nologin
avahi:x:116:122:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin stunnel4:x:996:996:stunnel service
```

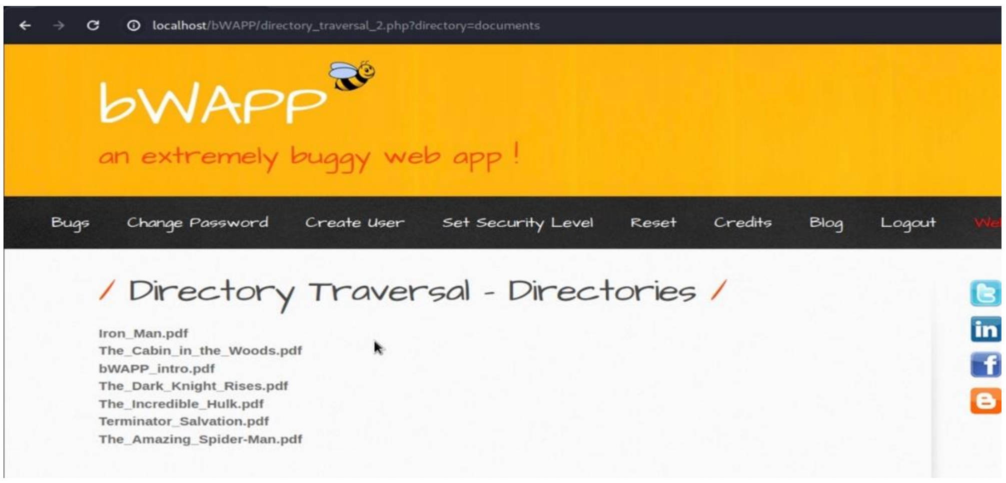
Preventions:

- **Filtering**: Remove potentially harmful characters that could be used for code injection, such as single quotes ('), double quotes ("), backslashes (), angle brackets (<, >), and semicolons (;).
- **Encoding:** Encode special characters like '<' and '&' as their corresponding entity references (&lt; and &amp;) to prevent them from being interpreted as code within the context you're using the data (e.g., HTML output, database queries).
- **Whitelisting:** Consider a whitelist approach where you define a limited set of allowed characters that can be used in user input. This can be stricter than just filtering and helps prevent unexpected vulnerabilities.

➔ Missing Function Level Access Control

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

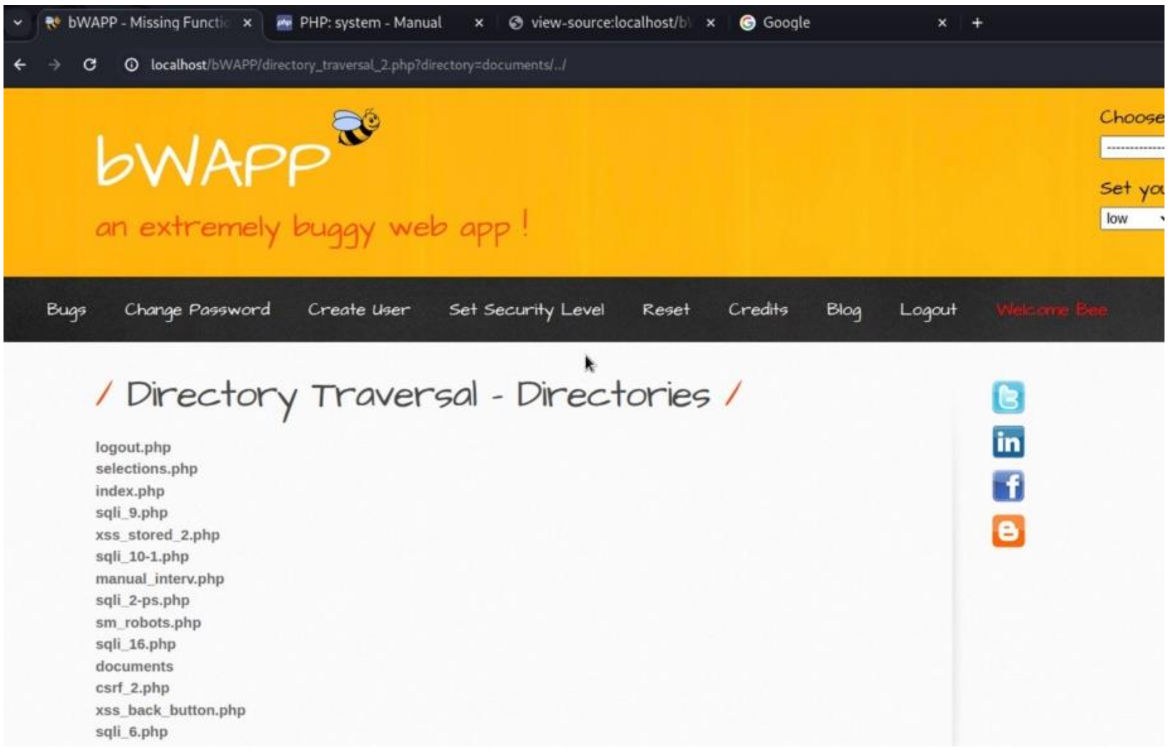
Screenshot after normal user action: A normal user would probably click on any pdf and view its contents.


localhost/bWAPP/directory_traversal_2.php?directory=documents
bWAPP
an extremely buggy web app !
Bugs
Change Password
Create User
Set Security Level
Reset
Credits
Blog
Logout

/ Directory Traversal - Directories /

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

Screenshot after taking action as ethical hacker: We checked the URL of the webpage and we saw their directory=documents. "*http://localhost/bWAPP/directory_traversal_2.php?directory=documents*". Then I edited the URL to make it go a directory backwards by using ".." and it went back to previous directory.



/ Directory Traversal - Directories /

logout.php
selections.php
index.php
sqli_9.php
xss_stored_2.php
sqli_10-1.php
manual_interv.php
sqli_2-ps.php
sm_robots.php
sqli_16.php
documents
csrf_2.php
xss_back_button.php
sqli_6.php

Observation: In the screenshot, we navigated through directories using the web page URL. Directory traversal, also called file path traversal, is a web security flaw enabling attackers to read any files on the server, like application code, credentials, and system files. They may even write to files, altering app data and gaining full server control. The severity depends on the data's importance, making it a serious concern.

Prevention: To prevent file path traversal vulnerabilities, it's best to avoid using user-supplied input directly in filesystem APIs. Instead, the application should validate the input before processing it. Preferably, validation should compare against a whitelist of allowed values. If not feasible, ensure the input contains only permitted content, like alphanumeric characters. After validation, append the input to the base directory and use a platform filesystem API to standardize the path. Confirm that the standardized path begins with the expected base directory.