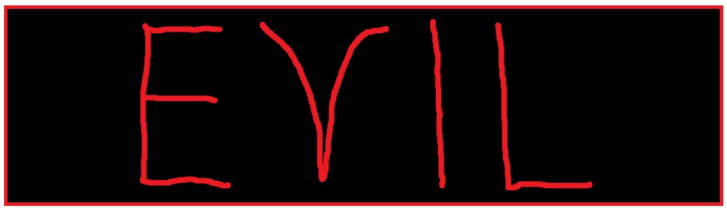

Evil Corp. Penetration Test

by Justus Uurtimo

Start of testing: May 1, 2022

End of testing: June 6, 2022

Contents

1 Executive Summary

In this penetration test the Evil Company was examined for security-relevant weaknesses. The testing was conducted as black-box testing, this is the kind where no information about the internals of the system is given. The scope of the assessment was as follows:

Table 1.1 contains the overview of examined systems during the penetration test.

Web Site	**Hostname**
Domain 1	https://csb-capture-the-flag.cs.helsinki.fi/
Path 1	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/pics/tireddog.jpg
Path 2	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/sites/lockhart/index.html
Path 3	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/keeper
Path 4	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/text/lawney_logs.zip
Path 5	https://csb-capture-the-flag.cs.helsinki.fi/bank
Path 6	https://csb-capture-the-flag.cs.helsinki.fi/challenges/sisu
Path 7	https://csb-capture-the-flag.cs.helsinki.fi/challenges/patronus
Path 8	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/sites/lockhart-revenge/index.html
Path 9	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/stash
Path 10	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/box
Path 11	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/machinek
Path 12	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/response_obf.py
Path 13	https://csb-capture-the-flag.cs.helsinki.fi/challenges/lounge
Path 14	https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/pics/ftuna.jpg

Table 1.1: Web sites examined during the penetration test

There were also some tasks where no domain / site address was given. As a result, several vulnerabilities have been found among the assets of the organization, some of them pose a significant risk For example there were multiaple SQL injection risks, where one was serious enough that it poses a threat to user password security. Solutions to remedy the discovered vulnerabilities are provided together with detailed descriptions and reproduction steps.

2 Vulnerability overview

Figure 2.1 shows the overview of vulnerabilities that could be measured in a meaningful way with CVSSv3 scoring. They are grouped by target and in descending order of risk. They are categorized by their risk and potential and are differentiated in the categories low, medium, high and critical. These categories essentially mean how likely it is that the vulnerability is taken advantage of and how much impact it would have on the business operations. In this table I am following the NVD severity ratings. They are based on the CVSSv3 scoring and are categorized:

score of 0.0 = None
score 0.1 - 3.9 = Low
score 4.0 - 6.9 = Medium
score 7.0 - 8.9 = High
score 9.0 - 10.0 = Critical

Risk	Asset	Vulnerability	Section	Page
Critical	Path 6	SQL injection vulnerability	3.9	22
Critical	Path 7	SQL injection vulnerability	3.10	25
Critical	Path 2	Exposure of Sensitive Information to an Unauthorized Actor	3.2	6
Critical	Path 8	Exposure of Sensitive Information to an Unauthorized Actor	3.11	28
Critical	Path 13	Reliance on Cookies without Validation and Integrity Checking	3.18	45
High	Path 3	Storing Passwords in a Recoverable Format	3.3	8
Medium	Path 4	Cleartext Transmission of Sensitive Information	3.7	17
Medium	Path 5	Exposure of Sensitive Information to an Unauthorized Actor	3.8	19

Table 2.1: Vulnerability overview

3 Results

In this chapter, the vulnerabilities found during the penetration test are presented. All of the issues are in the order that I solved their respective exercises. Each contain the following information.

- Brief description of the vulnerability
- CVSSv3 score and explanations for each scoring, when they are applicable.
- References to related CWE
- Steps to reproduce.

Also the remediation recommendations are given for each issue found during the penetration test.

3.1 Steganography I

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/pics/tireddog.jpg

3.1.1 General information

The challenge states that:
"You see a large sleepy guard dog. Can you figure out the password so that you can pass him? "
As the name of this challenge states, this task is essentially finding the secret from the image.

3.1.2 Proof of concept

I first start this challenge by checking the metadata of the image as sometimes there is glues hidden there. I used a commandline tool; *"exiftool* for this purpose. I ran the tool in the terminal with a command:
exiftool tireddog.jgp
This gives me information of the image. From that I can immediately see something that seems out of place, the artist is listed as *"RubeusHagrid"*. This name of the artist was the correct answer for this challenge.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ exiftool tireddog.jpg
ExifTool Version Number         : 12.39
File Name                       : tireddog.jpg
Directory                       : .
File Size                       : 35 KiB
File Modification Date/Time     : 2022:04:24 01:36:40+03:00
File Access Date/Time           : 2022:04:24 01:36:40+03:00
File Inode Change Date/Time     : 2022:04:24 01:36:40+03:00
File Permissions                : -rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 96
Y Resolution                    : 96
Resolution Unit                 : inches
Artist                          : RubeusHagrid
Y Cb Cr Positioning             : Centered
Image Width                     : 498
Image Height                    : 371
Encoding Process                : Progressive DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 498×371
Megapixels                      : 0.185
```

Figure 3.1: flag of Steganography 1

3.1.3 Proposed solutions

You should not try to hide sensitive information in images in a plain text as that is not secure and is easily retrievable.

3.2 Lockhart

CTF system
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/sites/lockhart/index.html

3.2.1 General information

The challenge states that :
"You have lost your key to Gilderoy Lockhart software and he is refusing to help you since he believes that those who are foolish enough to lose their keys don't deserve to use his products. Lockhart says that you should try to figure out what your key was. You can use his key validation service. "
CVSS calculation of this vulnerability is presented in Table 3.7.

The CWE categorization of this vulnerability could be CWE-200: Exposure of Sensitive Information to an Unauthorized Actor as the password is accessible just by viewing the parameter of the source code.

Description	score	reason
Attack Vector (AV)	Network	Attack can be done remotely
Attack Complexity (AC)	Low	No special conditions needed
Privileges Required (PR)	None	Attacker is unauthorized prior the attack
User Interaction (UI)	None	No user interaction required
Scope (S)	Unchanged	Impacted component is the same as the vulnerable component
Confidentiality Impact (C)	High	With the master password all of the can be compromised
Integrity Impact (I)	High	With the password the information on the software can be modified at will.
Availability Impact (A)	None	Attacker is not able to affect availability
Overall	9.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

Table 3.1: CVSSv3 score of the issue

3.2.2 Proof of concept

I open the site and quickly discover that there is no submit button. I then open the developer tools and the site inspector. It seems that the site actually checks the password after each letter is submitted via event. From this event the password can be seen in plain text.

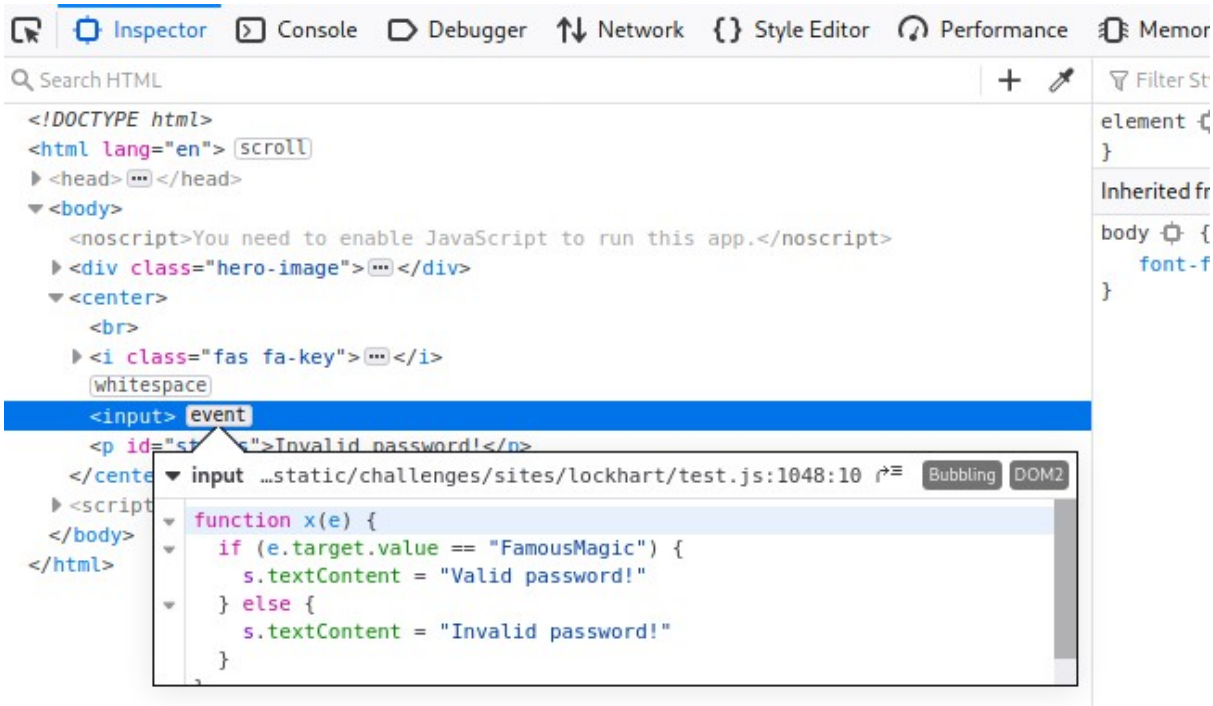

Figure 3.2: flag of Lockhart

3.2.3 Proposed solutions

The main issue here is that the comparisson of the correct password is done in plain text. As such the correct password can be retrieved by simply examining the source code functions.

The comparison should be conducted in the server side and never against plain text values. The input should be salted and hased and after that compared against a a salted and hased value of the password. Hashing should always be used in password storage as: *"Hashing is appropriate for password validation. Even if an attacker obtains the hashed password, they cannot enter it into an application's password field and log in as the victim"* (OWASP 2022)

3.3 Password checker

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/keeper

3.3.1 General information

The challenge states that:
"Dobby has lost password for his proprietary software. Can you help him out? https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/keeper"

The CWE categoriztion for this vulnerability could be: CWE-257: Storing Passwords in a Recoverable Format

CVSS calculation of this vulnerability is presented in Table 3.2.

Description	score	reason
Attack Vector (AV)	Local	Attackers need the program in their own system
Attack Complexity (AC)	Low	No special conditions needed
Privileges Required (PR)	None	Attacker is unauthorized prior the attack
User Interaction (UI)	None	No user interaction required
Scope (S)	Unchanged	Impacted component is the same as the vulnerable component
Confidentiality Impact (C)	High	Attack is be able to retrieve and read the users data stored in the software.
Integrity Impact (I)	High	Attacker would be able to modify the users infomration in that software
Availability Impact (A)	None	Availability can not be affected
Overall	7.7	AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

Table 3.2: CVSSv3 score of the issue

3.3.2 Proof of concept

After downloading the file I try to run it but this leads me nowhere since it obviously needs the password. Next I tought that maybe the password is again stored in plain text inside the file and used command:
"strings keeper"
This prints any human readable string inside the file. As can be seen from the image below this indeed revealed the secret.

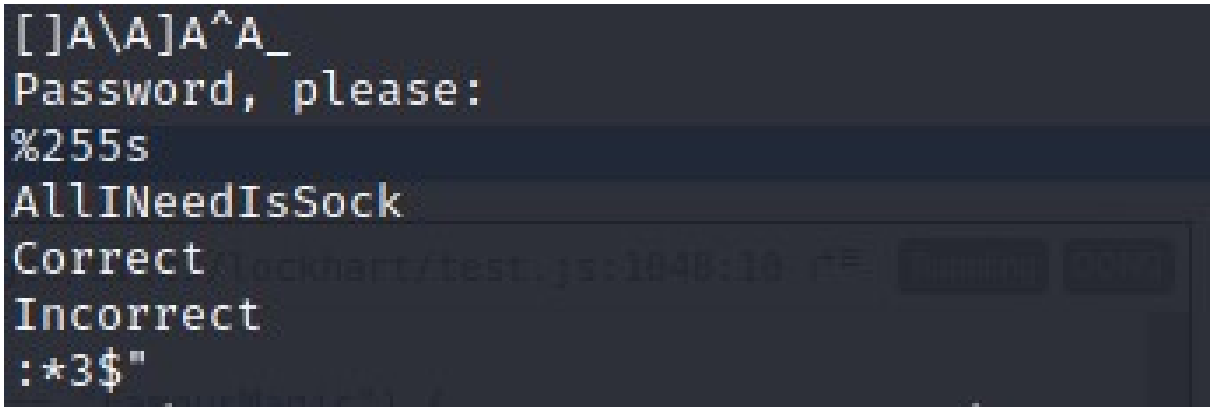

Figure 3.3: flag of Password checker

3.3.3 Proposed solutions

It seems that the password was stored in as plain text and thus I was able to retrieve it so easily. The solution for this vulnerability would be to at least use some hashed form of the password. This will not stop determined attacker, since:
"After an attacker has acquired stored password hashes, they are always able to brute force hashes offline" (OWASP 2022)
Which means that a strong hash should be used so that the cracking is as hard as possible to deter away most attackers.

3.4 Two time pad

3.4.1 General information

The challenge states:
"Fred sends a message to his twin using XOR and a one-time pad. You find out that the plain text message 'pranks' encrypted is (124, 71, 111, 59, 15, 103). Fred being not the shapest tool in the shed sends a new message (96, 92, 125, 33, 1, 122) and reuses the pad. Decrypt the message. "

3.4.2 Proof of concept

The first message, "pranks", encrypted is (124, 71, 111, 59, 15, 103) As such I set all letter for the corresponding ASCII value
p r a n k s
112 114 97 110 107 115

Because XOR works in both ways I can retrieve the original key from the first encrypted word.
112 114 97 110 107 115 = Decimal values of the original characters
124 71 111 59 15 103 = Decimal values of the encrypted characters

This results: 12, 53, 14, 85, 100, 20 which is the decimal values of the key of the original encryption.

Since the key was reused I can use the same key to crack the second encrypted message.
96, 92, 125, 33, 1, 122 = decimal values of the second encrypted word
12, 53, 14, 85, 100, 20 = decimal values of the key.

This results in: 108, 105, 115, 116, 101, 110. Which can be changed to letters using ASCII alphabet. It reads: listen

For XOR comparisons I used a web-based tool: https://xor.pw/
For ASCII dictionary I used https://www.ascii-code.com/

Flag found: listen

3.4.3 Proposed solutions

Fred should have used One-time pad, in which a randomly generated private key is used only once for encryption. The advantage of one-time pads is that there is no real way of braking the code, just by analyzing the succession of messages. In this challenge however Fred re-used the pad and encryption, which allowed the attacker to decipher the message quite easily.

Solution for this would be to deter from re-using the encryption keys. Also utilizing a better encryption is suggested. While one-time pad is truly unbreakable if utilized correctly it has some flaws, such as the fact that the key must be the same size as the message sent and the key must be securely shared between the two parties.

3.5 Luna's Secret

3.5.1 General information

The challenge states: *"Luna has chosen two primes p=17 and q=23 for her RSA modulus n, and a public key e=9. She receives a cipher 263 that has been encrypted with her public key. What is the plaintext?"*

The CWE category of this vulnerability could be CWE-326: Inadequate Encryption Strength, as RSA is a valid encryption scheme, but the values used here are so weak they can be cracked by hand.

3.5.2 Proof of concept

For this challenge I used the lecture materials and *"https://www.di-mgt.com.au/rsa_alg.html"* as a help on how to calculate this. Calculations:
p = 17
q = 23
e = 9
c = 263
n = $p * q = 391$
O(n) = (17-1)(23-1) = 352
gcd = (9, 352) = 1

d = $9^{-1} mod 352 = 313$

m = $263^{313} mod 391$ = 42

flag is 42 CORRECT!

3.5.3 Proposed solutions

For this vulnerability, Luna should choose stronger values for the key creation. The best way to approach this would be to advice Luna to use some existing methods for RSA key creation and using strong key strengths that have not been cracked, such as 2048.

3.6 Unbreakable

3.6.1 General information

The challenge states that:
"Neville reads about ancient Rome history, and decides to encrypt his letters to his grandmother. Unfortunately, he forgets the code. Can you help him out: Wxtk Zktgwfhmaxk. B ahix rhn tkx ybgx. B uhnzam t gxp ltyx yhk fr uxehgzbgzl tgw max itllphkw bl AtggtaTuuhmm. "

The CWE categoriation of this vulnerability is:
CWE-261: Weak Encoding for Password

3.6.2 Proof of concept

The way that the code looks and the "Rome history" part of the description is a dead give away and this is most likely some sort of Caesar's cipher. For this challenge I used a web version of Caesar decryption that I found from: https://www.dcode.fr/caesar-cipher

With that tool the original text is found almost immediately.

↑↓	↑↓
→19 (←7)	I hope you are fine. I bought a new safe for my belongings and the password is HannahAbbott.

Figure 3.4: flag of Unbreakable

3.6.3 Proposed solutions

Neville should utilize a encryption / hash method that is more secure and at least force the attacker to use a lot of computational power for the password crack.

3.7 Logs Logs Logs

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/text/lawney_logs.zip

3.7.1 General information

The challenge states:
"Prof. Trelawney asked you to find any leaks in her web site. You noticed that the developer wrote ...< form > < input type=text name=passwd >... So to prove a point you started searching the logs."

The CWE categorization of this vulnerability is:

CWE-319: Cleartext Transmission of Sensitive Information

CVSS calculation of this vulnerability is presented in Table 3.8

Description	score	reason
Attack Vector (AV)	Adjacent Network	Attacker need to be in the same network as the target
Attack Complexity (AC)	Low	No special conditions needed
Privileges Required (PR)	None	Attacker does not need any privileges
User Interaction (UI)	None	The attack requires no interaction from users
Scope (S)	Unchanged	Impacted component is the same as the vulnerable component
Confidentiality Impact (C)	High	Attacker is able to see all traffic from that web site
Integrity Impact (I)	None	The attacker can not modify any data with this attack alone.
Availability Impact (A)	None	The attacker can not impact the availability of the resources with this attack alone.
Overall	6.5	AV:A/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Table 3.3: CVSSv3 score of the issue

3.7.2 Proof of concept

The protocols used here are all HTTP and as such are not secured. Also since the developer wrote the field as name=passwd, I can just search the logs for that field name. And as expected the flag was quickly found in plain text.

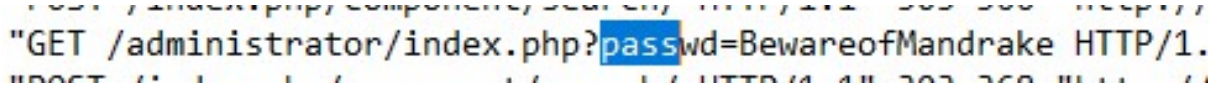

Figure 3.5: flag of Logs logs logs

3.7.3 Proposed solutions

A more safe protocol should be used, such as HTTPS, or even VPN to hide the content of the requests. Also since the input type is text, the password is visible when it is written, which is very insecure and could compromise a password, should someone shoulder surf at the right moment.

3.8 DiamondHands bank

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/bank

3.8.1 General information

The challenge states that:
"Gringotts Bank has a new competitor. You decide to check it's security. Rumour has it that there is a secret admin panel containing top secret information. The panel supposedly leaked previously when a popular search engine accidentally indexed it. However, security has been tightened since then. "

The CWE categorie for this challenge could be: CWE-200: Exposure of Sensitive Information to an Unauthorized Actor as the admin panel is available for anyone that know its location.

CVSS calculation of this vulnerability is presented in Table 3.8

Description	score	reason
Attack Vector (AV)	Network	Attack can be done remotely
Attack Complexity (AC)	Low	No special conditions needed
Privileges Required (PR)	None	Attacker does not need any privileges
User Interaction (UI)	None	The attack requires no interaction from users
Scope (S)	Unchanged	Impacted component is the same as the vulnerable component
Confidentiality Impact (C)	Low	Access to some restricted inforamtion is gained, but the attacker can not control what is obtained
Integrity Impact (I)	None	The attacker can not modify any data with this attack alone.
Availability Impact (A)	None	The attacker can not impact the availability of the resources with this attack alone.
Overall	5.3	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Table 3.4: CVSSv3 score of the issue

3.8.2 Proof of concept

The hint here is that the panel was leaked by a popular search engine. They way the search engines collect sites is by crawling them. For this reason many sites utilize robots.txt that tell the crawlers where they are not allowed to go. So I set out to look for the robots.txt.

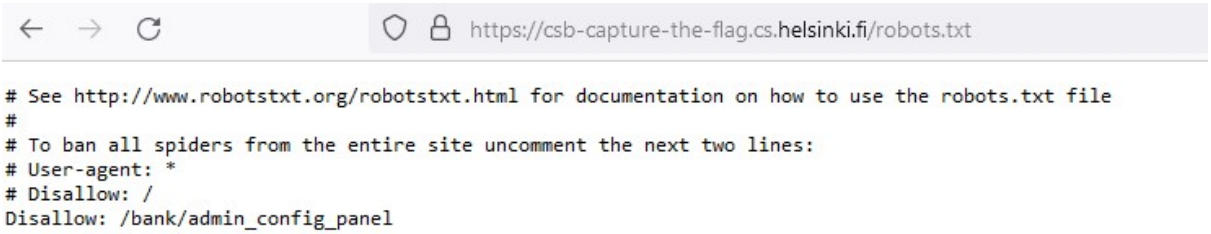

Figure 3.6: Robots.txt

The site disallows /bank/admin_config_panel from crawlers, so I set out to get there. The panel is accessable by anyone and thus the flag is found.

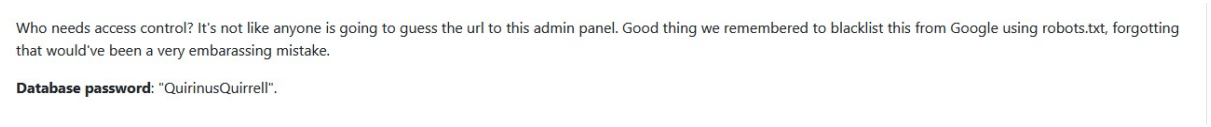

Figure 3.7: Diamondhands bank flag

3.8.3 Proposed solutions

The solution for this vulnerability would be restrict any access to the config panel. Also storing a database password in plain text in any pages is a huge vulnerability. The password should not be stored anywhere where it is accessable online. The solution here would be to restrict access to the configurations panel and to remove the password from there.

3.9 Where's Hermione

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/challenges/sisu

3.9.1 General information

The challenge states that:
"You need to find Hermione quickly. She's attending a class but you don't know which one. Luckily, the school has a new course system"

The CWE of this vulnerability is: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

CVSS calculation of this vulnerability is presented in Table 3.6

Description	score	reason
Attack Vector (AV)	Network	Attack can be done remotely
Attack Complexity (AC)	Low	No special conditions needed
Privileges Required (PR)	None	Attacker is unauthorized prior the attack
User Interaction (UI)	None	No user interaction required
Scope (S)	Changed	The vulnerable component is the web interface and the affected component is the database.
Confidentiality Impact (C)	High	Attack is be able to retrieve and read contents of files and expose sensitive data.
Integrity Impact (I)	High	It is possible for the attacker to modify the files in the server with a specialized SQL Injection.
Availability Impact (A)	High	Availability can be affected if the attacker shuts the database manager down.
Overall	10.0	AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Table 3.5: CVSSv3 score of the issue

3.9.2 Proof of concept

This is without a doubt a SQL injection exercise. I start by simply inputting ' in the seach bar and the site gives me an error message:

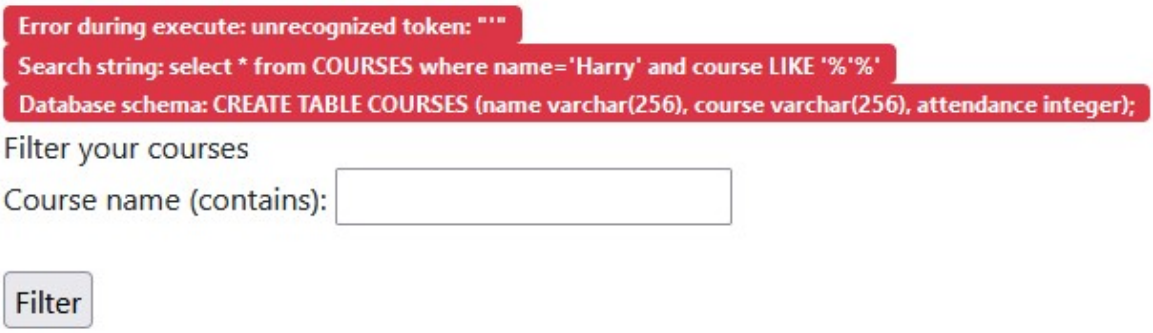

Figure 3.8: Error message

This error message gives me the whole query made and basicly tells me there are no sanitasion in place. The whole query is Search string:
*select * from COURSES where name='Harry' and course LIKE '%'%'*

I make union injection and try to print all of the Hermiones course. I do this with the following injection:
*jaa' union select * from COURSES where name = 'Hermione' –*

this gave me the list of Hermiones courses and the information of where she currently is.

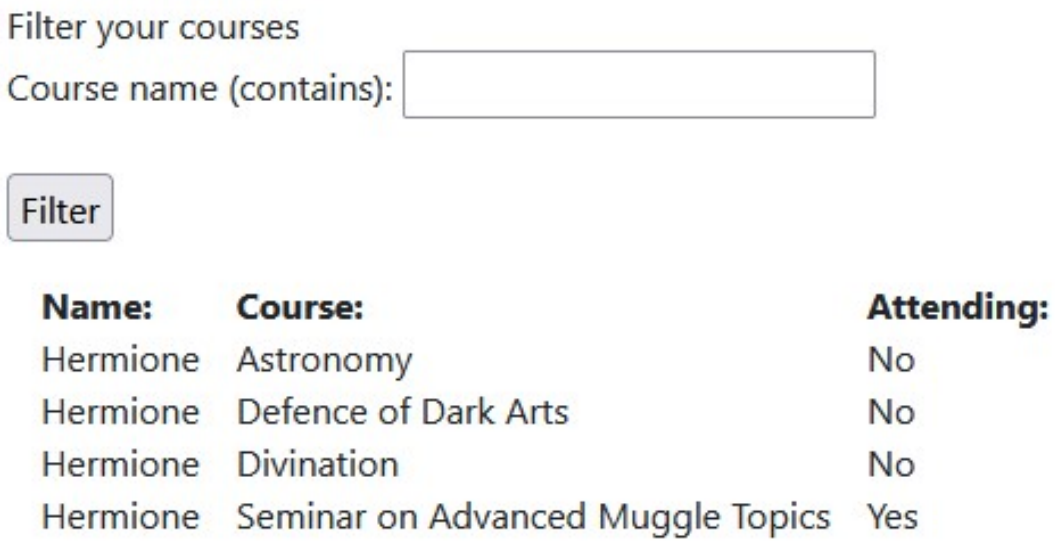

Figure 3.9: Hermione found

3.9.3 Proposed solutions

Since SQl Injection can occur when dynamic database queries are constructed with string concatenation, which has user supplied input, it can be avoided by simply not writing dynamic queries with string concatenation. Other way is to simply prevent user input from affecting the logic of the query. One way of making sure that the logic is not interfered is by using prepared statements that force the developer to define the SQL code and pass the parameters to the query later. (OWASP, n.d.)

3.10 Patronus

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/challenges/patronus

3.10.1 General information

The challenge states:
"Dolores Umbridge requires that every student will enter their Patronus in the database. Can you find out Albus' password? "

The CWE of this vulnerability is: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

CVSS calculation of this vulnerability is presented in Table 3.6

Description	score	reason
Attack Vector (AV)	Network	Attack can be done remotely
Attack Complexity (AC)	Low	No special conditions needed
Privileges Required (PR)	None	Attacker is unauthorized prior the attack
User Interaction (UI)	None	No user interaction required
Scope (S)	Changed	The vulnerable component is the web interface and the affected component is the database.
Confidentiality Impact (C)	High	Attack is be able to retrieve and read contents of files and expose sensitive data.
Integrity Impact (I)	High	It is possible for the attacker to modify the files in the server with a specialized SQL Injection.
Availability Impact (A)	High	Availability can be affected if the attacker shuts the database manager down.
Overall	10.0	AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Table 3.6: CVSSv3 score of the issue

3.10.2 Proof of concept

This is again a SQL injection challenge.

Again I input ' and I am presented with an error message telling me the query done.

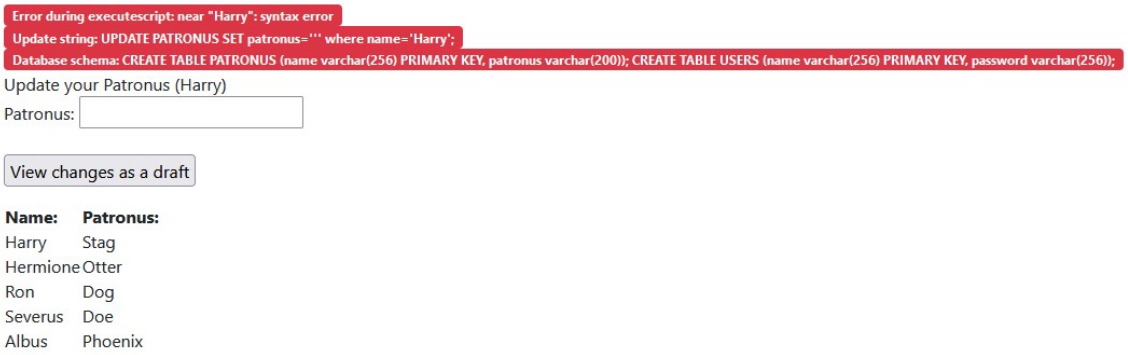

Figure 3.10: Error message

First I managed to change both Harrys and Dumbledores Patronuses. I did this with injection:

jaa' where name='Harry'; UPDATE PATRONUS SET patronus='asd' where name='Albus' –

This is not what is wanted in thies exercise but now I was able to run two queries.

I modified this query into this:

jaa' where name='Harry';UPDATE PATRONUS SET patronus=(select password from users where name='Albus') where name='Albus' –

Now this changes Harry's patronus to "jaa" and Dumbledore's patronus to his password.

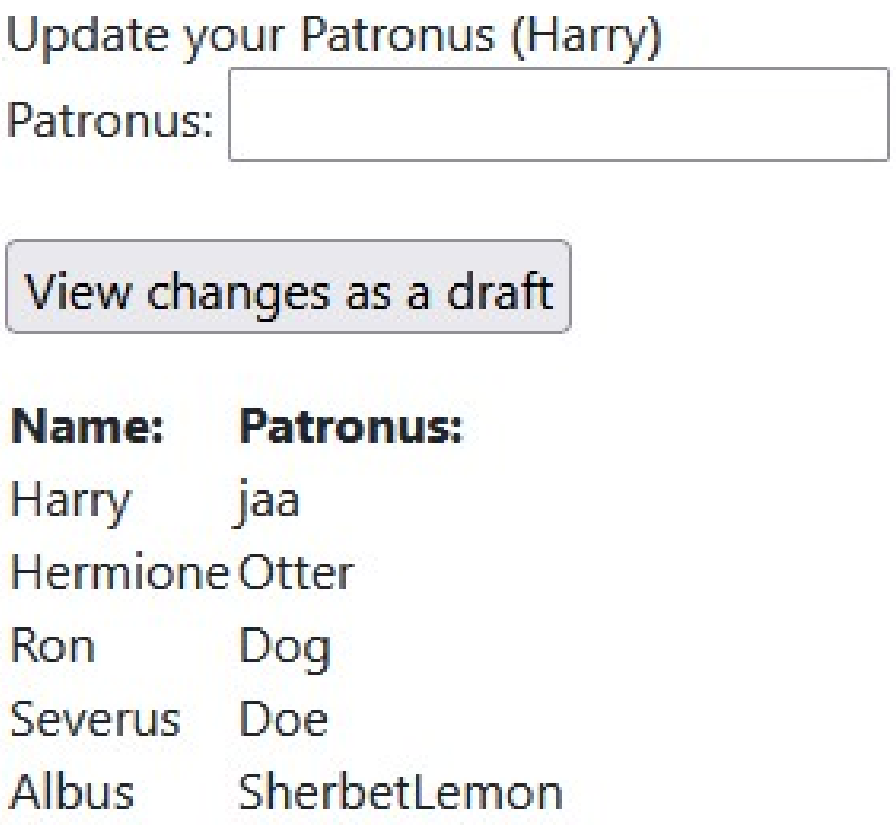

Figure 3.11: Patronus flag found

3.10.3 Proposed solutions

Since SQl Injection can occur when dynamic database queries are constructed with string concatenation, which has user supplied input, it can be avoided by simply not writing dynamic queries with string concatenation. Other way is to simply prevent user input from affecting the logic of the query. One way of making sure that the logic is not interfered is by using prepared statements that force the developer to define the SQL code and pass the parameters to the query later. (OWASP, n.d.)

3.11 Lockhart Mk II

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/sites/lockhart-revenge/index.html

3.11.1 General information

The challenge states that:
"Lockhart is furious that you have cracked his key service! He has changed all of his keys and made a new validation service. Regain access."

The CWE categorization of this vulnerability could be CWE-200: Exposure of Sensitive Information to an Unauthorized Actor as the password is accesseble just by viewing the parameter of the source code.

CVSS calculation of this vulnerability is presented in Table 3.8

Description	score	reason
Attack Vector (AV)	Network	Attack can be done remotely
Attack Complexity (AC)	Low	No special conditions needed
Privileges Required (PR)	None	Attacker is unauthorized prior the attack
User Interaction (UI)	None	No user interaction required
Scope (S)	Unchanged	Impacted component is the same as the vulnerable component
Confidentiality Impact (C)	High	With the master password all of the can be compromised
Integrity Impact (I)	High	With the password the information on the software can be modified at will.
Availability Impact (A)	None	Attacker is not able to affect availability
Overall	9.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

Table 3.7: CVSSv3 score of the issue

3.11.2 Proof of concept

Again the site is using an event to check the password, but this time the password is not in plain sight. With looking at the event in the developer tools I can see that it is a function called "decoder". This function is checking the targets value against a parameter "t". With this information I can simply put in a break point into the function and keep watch on this parameter "t".

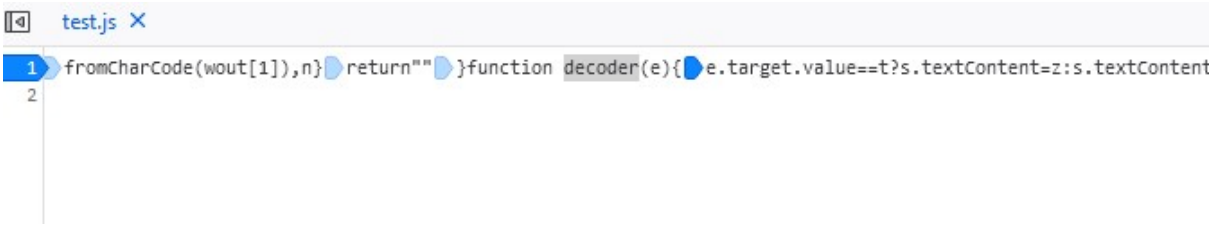

Figure 3.12: Break points

After this I just simply refresh the site and insert anything on the input field to trigger the break point. After that I simply scrolled down on the Scopes and Window: Global and quickly found the parameter "t" which held the password

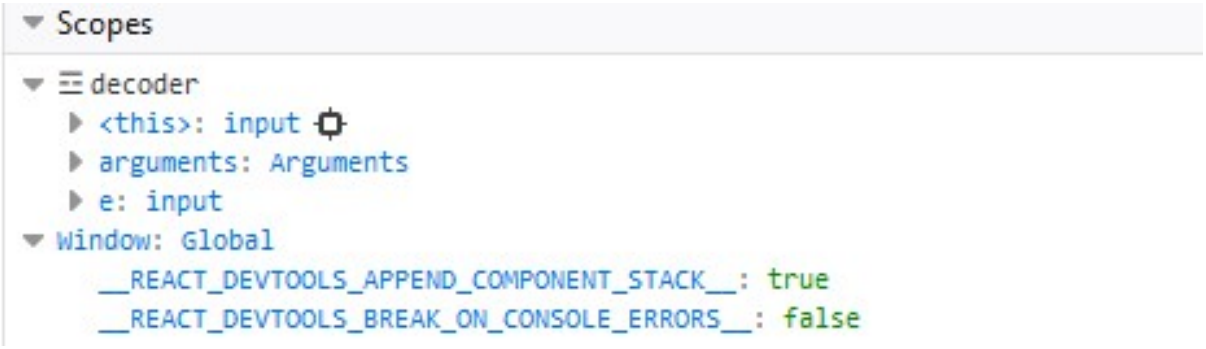

Figure 3.13: Scopes

Figure 3.14: Lockhart MK II flag

3.11.3 Proposed solutions

Again the main issue here is that the comparison of the correct password is done in plain text. As such the correct password can be retrieved by simply examining the source code functions.

The comparison should be conducted in the server side and never against plain text values. The input should be salted and hashed and after that compared against a a salted and hashed value of the password. Hashing should always be used in password storage as: *"Hashing is appropriate for password validation. Even if an attacker obtains the hashed password, they cannot enter it into an application's password field and log in as the victim"* OWASP 2022

3.12 Lazy passwords

3.12.1 General information

The challenge states that:
"Draco's password is 'Lucius2022' with a secure hash of 'b88ef4990be398d48ad0588b0ab582d18e9b0fe1653a4d2f345571372 d5dec6c62f64fba6617725746525502e71989f9'.
Draco decided to change his password but only changed the number. The new hash is '9675570bdf6a99 892a67dc4b7ad0b7a7fa36d3ea837360a051c42b56fdb f215fc639b36d1d16eb76da31a9f67d8cc915'. Can you figure out the new password?"

The CWE categoriztion of this vulnerability could be CWE-521: Weak Password Requirements as the service used has apprerently accepted that password, which is a very weak one.

3.12.2 Proof of concept

The exercise hints that the passwords are hashed. I used this courses information and since I know that Draco only changed the number I can create a password file that only contains the word: Lucius After this I create new rule for john the ripper that will append four numbers from 0-9 at the end of Lucius.

The rule was: $[0-9]$[0-9]$[0-9]$[0-9] and I replace the existing Wordlist rule with this rule by simply commenting out the existing header and inserting my rule at the start of the file.

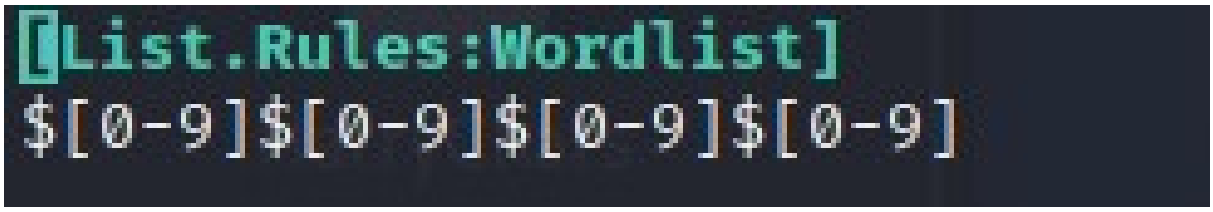

Figure 3.15: New rule for the ripper

After this I ran the command:
john –wordlist=CTF_PASS -stdout –rules > CTF_APPEND.txt
that will append the numbers and create a new file with all the possible passwords.

After this I simply ran command:
john –wordlist=CTF_APPEND.txt CTF_Crack
This quickly found the password.

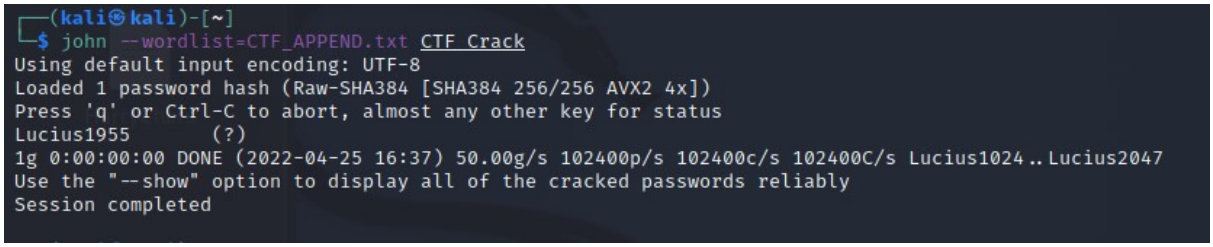

Figure 3.16: Flag for lazy passwords

3.12.3 Proposed solutions

The first solution for this vulnerability is that Draco should use much stronger passwords. Using name + year is a very weak combination, for a password. Also when changing a password one should change it more than just a number, because as we can see from this challenge it is quite easy to crack if you know what the previous one was.

A strong password is a password with a alot of entropy and as such using a term "passphrase" would be more correct. It should be also noted that replacing characters with easy to quess numbers, such as i to 1, is known to attackers and as such does not increase security.

Relevant xkcd: https://xkcd.com/936/

3.13 Neville the cipherer

3.13.1 General information

The challenge states that:
"Neville tries to be clever and invented a better scheme for encryption. Or did he? Can you prove him wrong? Ciphertext: 'LB5HO2TRMQQHS3LOPAQG46BAOJ2HO2RAPBVGQ6TXNI======'"

The CWE for this vulnerability could be: CWE-261: Weak Encoding for Password

3.13.2 Proof of concept

At first sight this seems to be a base64 and thus **I** try to decode it with a online decoder **I** found :
https://emn178.github.io/online-tools/base64_decode.html

This gives me an error that this contains invalid character for base64. **I** then try to decode it as base32 as it looks alot like it as well.

This gives me a result: Xzwjqd ymnx nx rtwj xjhzwj Now this looks like it is again a caesar cipher or Rot encoded. **I** first try the same caesar cipher as before: https://www.dcode.fr/caesar-cipher

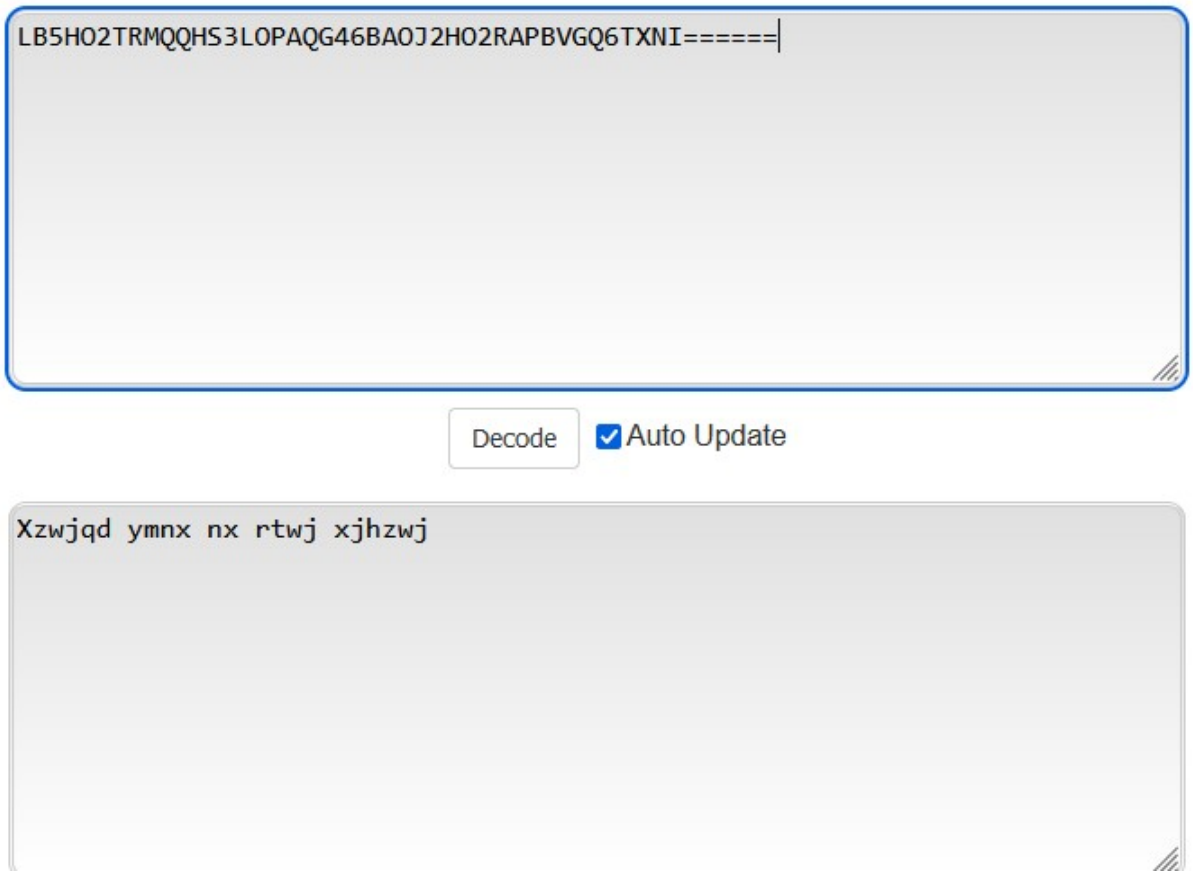

Figure 3.17: Base34 decoded

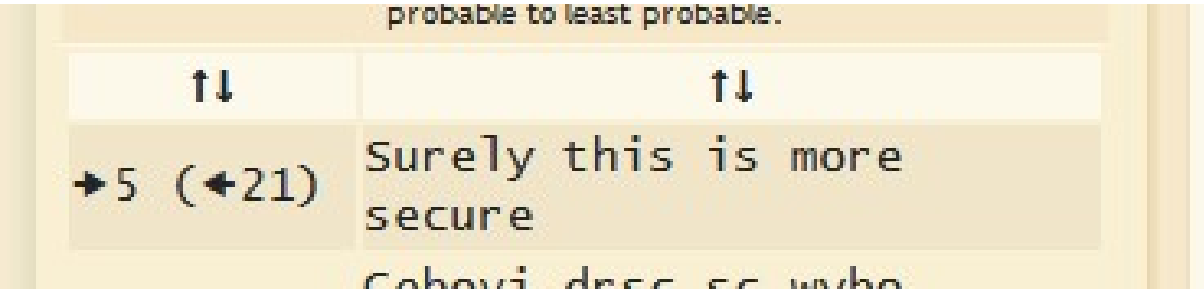

Figure 3.18: Neville The Cipherer Flag

3.13.3 Proposed solutions

Again Neville is trying to secure his messages with scheme that can be cracked with on hand tools and with quite ease. It can be recommend for Neville that the usage of stronger encryption is advised.

3.14 Passwords II

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/stash

3.14.1 General information

The challenge states that:
"Dobby has a friend that also needs help. Can you help him?"

This challenge is quite similiar to the challenge *"Passwords"* as in this the goal is to get the password from the binary file.

3.14.2 Proof of concept

I download the file and again first run the command:
strings stash

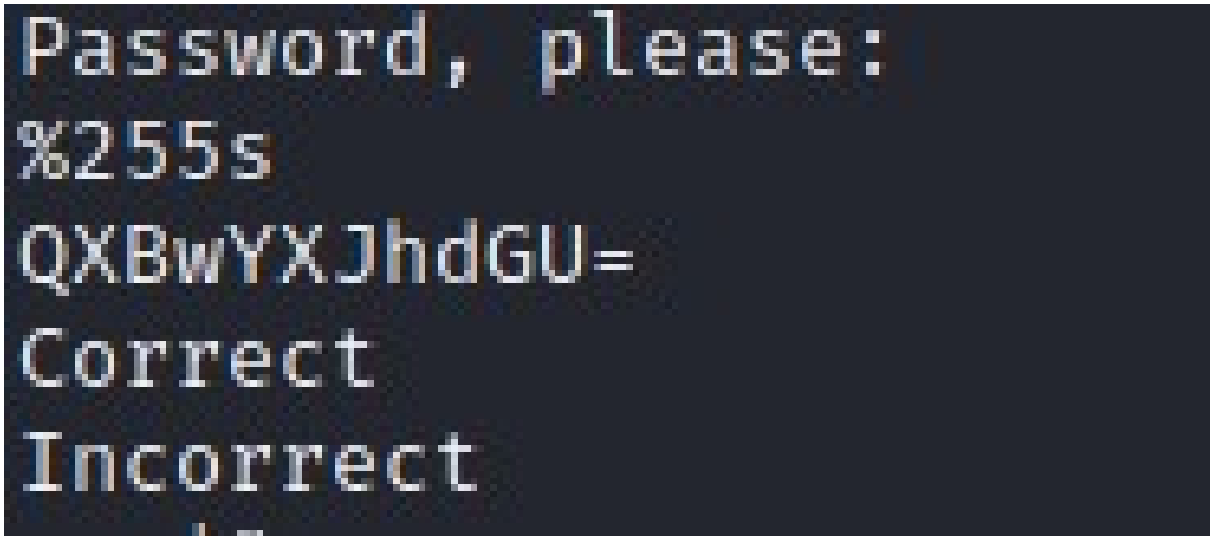

Figure 3.19: Password II Strings

This time the file did not have the password in plain text but there is a curious encrypted string in there. *QXBwYXJhdGU=*

This seems to be a base64 or base32 encoded so I try to decode it again using the web decoder. The string is base64 and I could decode it with the web decoder.

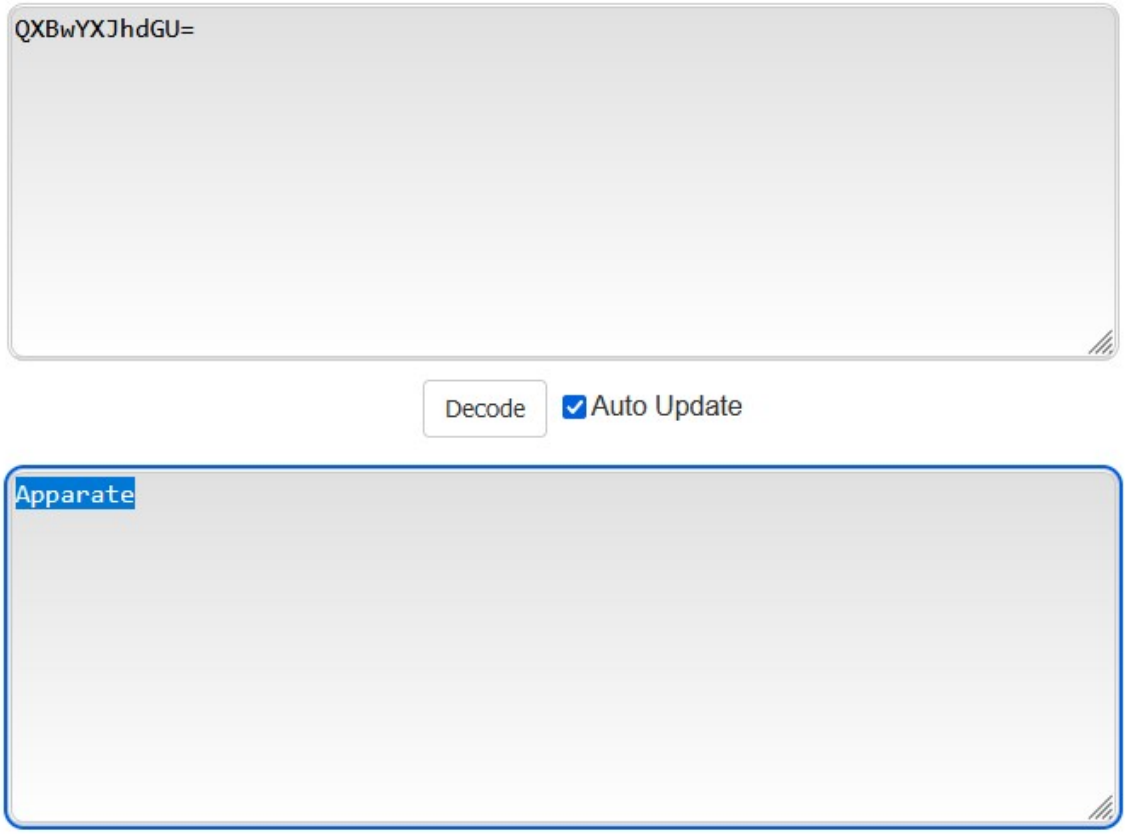

Figure 3.20: Password II flag found

3.14.3 Proposed solutions

It seems that the password was stored in base64 encrypted text and thus I was able to retrieve it easily. The solution for this vulnerability would be to at least use some stronger hashing system for hashing the password. Although this will not stop determined attacker, since:
"After an attacker has acquired stored password hashes, they are always able to brute force hashes offline" (OWASP 2022)
Which means that a strong hash should be used so that the cracking is as hard as possible to deter away most attackers.

3.15 Passwords III

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/box

3.15.1 General information

The challenge states that:
"A program written in C is given to you by the local wandmaker. It uses a known library with a type declaration 'struct state char input[2048]; int debug; ;'. Can you find the secret?"

From the description it can be seen that this challenge is based on Buffer overflow -vulnerability.

The CWE categorization of this challenge could be: CWE-121: Stack-based Buffer Overflow

3.15.2 Proof of concept

The exercise gives me the hint that the key to this exercise can be gained by inserting some value to the input. I did some research on how C char [] works and found this:
http://www.sis.pitt.edu/jjoshi/courses/IS2620/Spring07/Lecture3.pdf
What was most useful was the information from page 17-18 which was about how to trigger the buffer overflow. In a nutshell it seemed that I could simply input a value greater than 2048 bits to trigger the debug.

I found an online string length byte calculator:
https://mothereff.in/byte-counter
With that tool I made a string with length of 3456 bytes and inputted it to the program, as can be seen from image below. This indeed triggered the debug mode and gave me the secret.

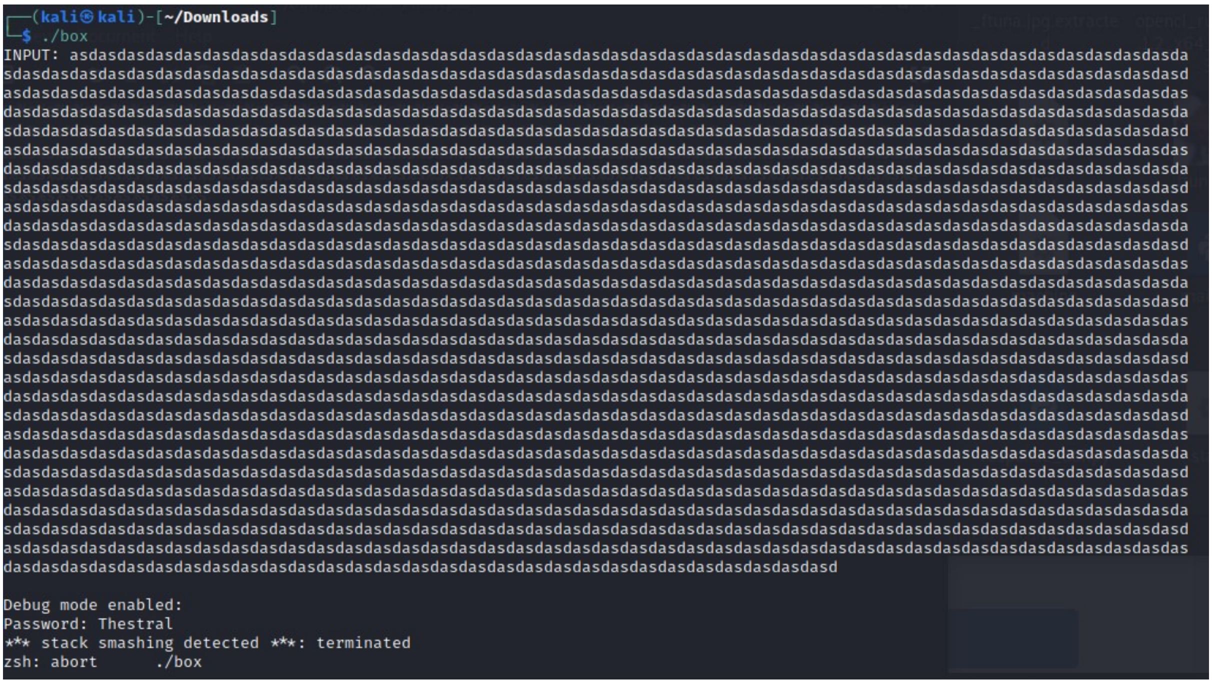

Figure 3.21: Password III flag found

3.15.3 Proposed solutions

The overflow could be mitigated by avoiding functions that do not do buffer checks. Another way would be to start using Canaries, that are inserted before the return address and are checked before the retrun address is accessed. Should the progarm detect any change in canary value, the program will abort the process and thus preventing any damage being done. (Kaczanowski 2021)

3.16 Nevilles message

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/machine

3.16.1 General information

The challenge states:
"Neville overhears a quiet conversation, and sends it to you: 'TNIJCRMFZNTCUPULTBSHPFFHANHIEQJGVOEOJ'. Unfortunately, he forgets the decryption key. Luckily, you got access to his encrypter. Can you figure out the message? "

3.16.2 Proof of concept

After downloading the encryper I try to run it with arbitary input "abc" this prints "GAA"

Figure 3.22: First try

After this I try to inpout the encrypted text to see if that would come out normally. I used command:
./machine TNIJCRMFZNTCUPULTBSHPFFHANHIEQJGVOEOJ
and the flag was found: *DIDYAPUTYOURNAMEINDAGOBLETOFFIREHARRY*

Figure 3.23: Nevilles message flag found

3.17 Death eaters

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/binaries/response_obf.py

3.17.1 General information

The challenge states that:
"You find out Death Eaters use proprietary client that connects to csb-capture-the-flag.cs.helsinki.fi:8787 to find the current secret key. See whether you can find it out. The client doesn't print it but you are sure that the server tells it. "

3.17.2 Proof of concept

After first downloading the file, I try to open it and instantly see that it is really weird and does not look like a python file at all. So, I run the command:
file response_obf.py to find out what it actually is.

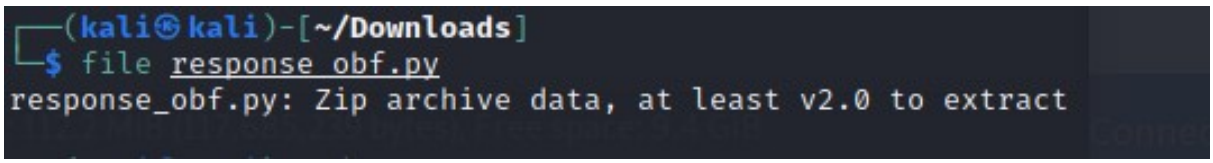

Figure 3.24: First file check

As can be seen from the image above the file is in fact a ZIP file! So, I will use the unzip to extract the ZIP file into a folder I have created. the command I used was:
unzip response_obf.py -d extract

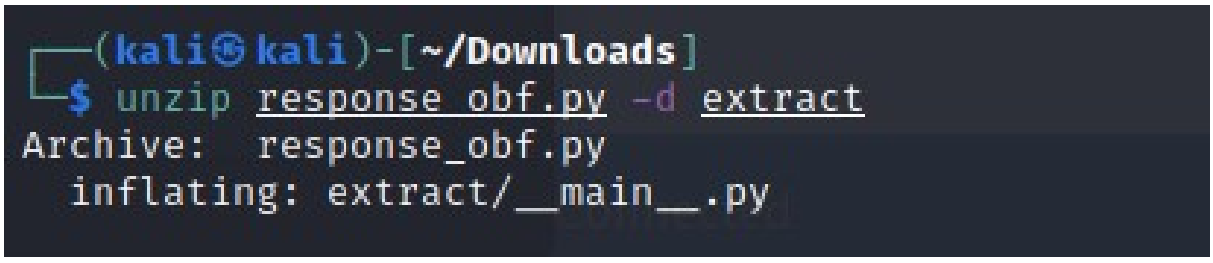

Figure 3.25: File extract

Next I run the file command on the newly found python file to make sure it is infact a python file.

```
┌──(kali㉿kali)-[~/Downloads/extract]
└─$ file __main__.py
__main__.py: Python script, ASCII text executable
```

Figure 3.26: Second file check

The exercise states that "The client doesn't print it but you are sure that the server tells it. " So I will just modify the client to show the secret.

```
1 import sys
2 import base64
3 import asyncio
4
5 async def connect(host, port):
6         reader, writer = await asyncio.open_connection(host, port)
7         line = await reader.readline()
8         t = ''.join([chr(x) for x in line if x < 128])
9         t = t.strip()
10        if len(t) != 32:
11                return
12        resp = base64.b64encode(bytes([ord(x) ^ ord(y) for x, y in zip(t[:16], t[16:])]))
13        writer.write(resp)
14        writer.write(b'\n')
15        await writer.drain()
16        line = await reader.readline()
17        line = line.decode()
18        if line.startswith('OK'):
19                print('Success')
20        else:
21                print('FAIL')
22
23
24 if __name__ == "__main__":
25        host = sys.argv[1]
26        port = int(sys.argv[2])
27        asyncio.run(connect(host, port))
28
```

Figure 3.27: Original python

After reading the original python file I simply just modified the line 19. to print(line). This makes it so that if connection is successful the python will print the servers response instead of the "SUCCESS" text

```
11                return
12        resp = base64.b64encode(bytes([ord(x) ^ ord(y) for x, y in zip(t[:16], t[16:])]))
13        writer.write(resp)
14        writer.write(b'\n')
15        await writer.drain()
16        line = await reader.readline()
17        line = line.decode()
18        if line.startswith('OK'):
19                print(line)
20        else:
21                print('FAIL')
22
23
24 if __name__ == "__main__":
25        host = sys.argv[1]
26        port = int(sys.argv[2])
27        asyncio.run(connect(host, port))
28
```

Figure 3.28: New python

After this I run the python with the command:
python3 __main__.py csb-capture-the-flag.cs.helsinki.fi 8787 and this gives me the flag.

```
┌──(kali㉿kali)-[~/Downloads/extract]
└─$ python3 __main__.py csb-capture-the-flag.cs.helsinki.fi 8787
OK key=AvaraK3davra
```

Figure 3.29: Death Eaters Flag found

3.18 Teacher Lounge

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/challenges/lounge

3.18.1 General information

The challenge states that:
"You find out that there is a teacher's lounge, and you want to snoop around. You heard that Severus Snape has the access to the key. "

This web service uses cookies on determining who is trying to access the site. By manipulating these cookies the attacker can gain access to the site.

CWE categorization of this vulnerability could be: CWE-565: Reliance on Cookies without Validation and Integrity Checking

CVSS calculation of this vulnerability is presented in Table 3.8

Description	score	reason
Attack Vector (AV)	Network	Attack can be done remotely
Attack Complexity (AC)	Low	No special conditions needed
Privileges Required (PR)	None	Attacker does not need any privileges
User Interaction (UI)	None	The attack requires no interaction from users
Scope (S)	Unchanged	Impacted component is the same as the vulnerable component
Confidentiality Impact (C)	High	The attacker is able to retrieve the password and thus access to the whole service and its files
Integrity Impact (I)	High	With the access to the service the attacker would be able to modify any data in there
Availability Impact (A)	None	The attacker can not impact the availability of the resources with this attack alone.
Overall	9.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

Table 3.8: CVSSv3 score of the issue

3.18.2 Proof of concept

I am first greeted with message:
Name: Harry
Key: Access not allowed

My initial quess is that the site is using cookies to track who is visiting it. I confirmed this by using the developers tools Network page and selecting the first GET with file lounge. From the headers I can see that there is a field: member_name=Harry

Figure 3.30: Teacher lounge cookie

I am using fire fox in this exercise and from the network page I can right click on the packet and select "edit and resend". From there I can edit the cookie value to say: *member_name=Severus Snape*.

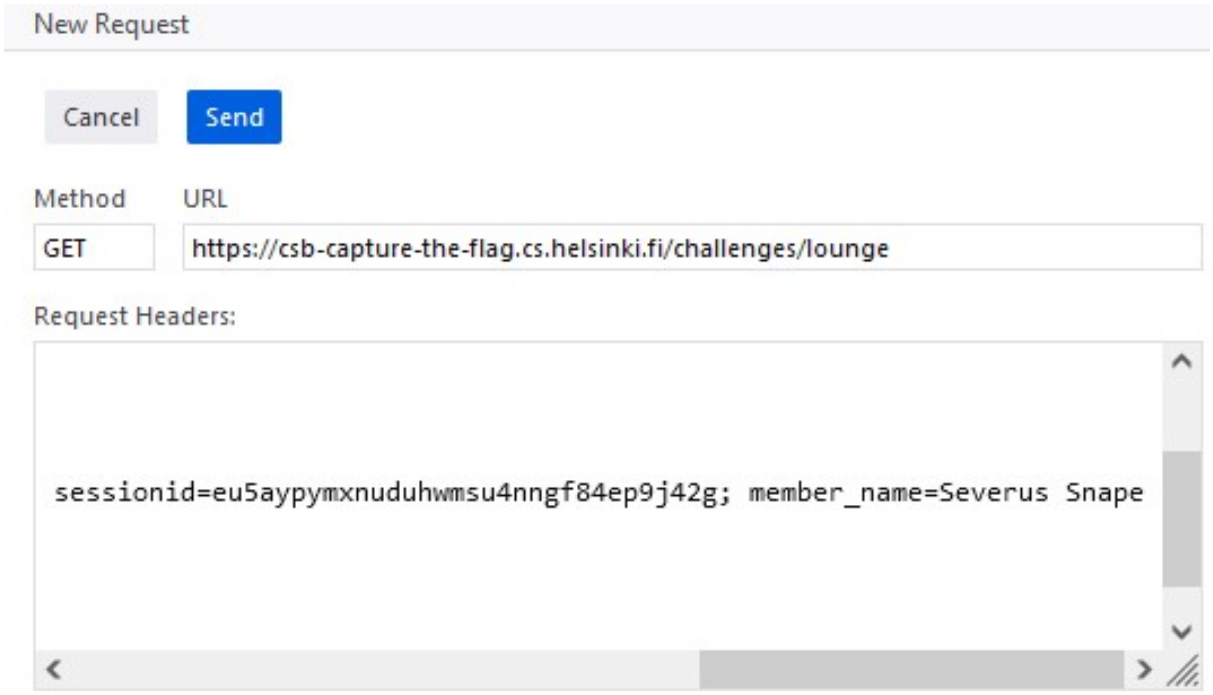

Figure 3.31: New reques

After sending that packet, the site does not change but from that new packets response I can get the key for this exercise.

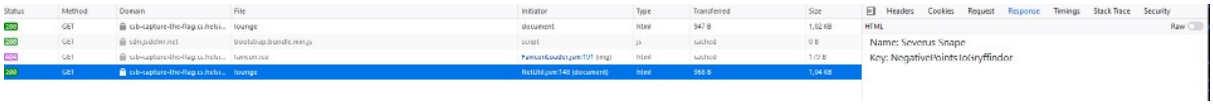

Figure 3.32: Teacher lounge flag found

3.18.3 Proposed solutions

There are few possible ways to mitigate this vulnerability. For example it is advisable to avoid using cookie data for a security-related decision. Also the service shoud perform thorough input validation (i.e.: server side validation) on the cookie data if you're going to use it for a security related decision. Lastly there should be integrity checks to detect tampering. (Common Weakness Enumeration, n.d.)

3.19 Steganography II

CTF system.
Hostname: https://csb-capture-the-flag.cs.helsinki.fi/static/challenges/pics/ftuna.jpg

3.19.1 General information

The only information that is given in this exercise is that "Something is hidden in the image. Can you find it? (Image was link to a .jpg image)

From the title and from the text it can be quessed that this is another Steganography challenge.

3.19.2 Proof of concept

I saved that image to my kali linux VM and started searching for some information on how to proceed.

First I installed steghide and tried to extract information on the image. Unfortunately the file is locked with passphrase that is unknown.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ steghide extract -sf ftuna.jpg
Enter passphrase:
steghide: could not extract any data with that passphrase!
```

Figure 3.33: steghide fail

One way to proceed would be to utilize stegcracker and try to bruteforce the password, but since that is quite time consuming I will first try something else.

So next I install exiftool, since maybe the password can be again found from the metadata.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ exiftool ftuna.jpg
ExifTool Version Number         : 12.39
File Name                       : ftuna.jpg
Directory                       : .
File Size                       : 96 KiB
File Modification Date/Time     : 2022:04:23 13:45:11+03:00
File Access Date/Time           : 2022:04:23 13:51:02+03:00
File Inode Change Date/Time     : 2022:04:23 13:45:11+03:00
File Permissions                : -rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 1
Y Resolution                    : 1
Profile CMM Type                : Apple Computer Inc.
Profile Version                 : 2.1.0
Profile Class                   : Display Device Profile
Color Space Data                : RGB
Profile Connection Space        : XYZ
Profile Date Time               : 2012:10:18 15:31:23
Profile File Signature          : acsp
Primary Platform                : Apple Computer Inc.
CMM Flags                       : Not Embedded, Independent
Device Manufacturer             :
Device Model                    :
Device Attributes               : Reflective, Glossy, Positive, Color
Rendering Intent                : Perceptual
```

Figure 3.34: Exfil Fail

Unfortunately this did not find anything usefull. So next I try to see if there are some human readable strings inside the file. For this I use the command strings that prints strings of printable characters from files.

The command was simply: string ftuna.jpg

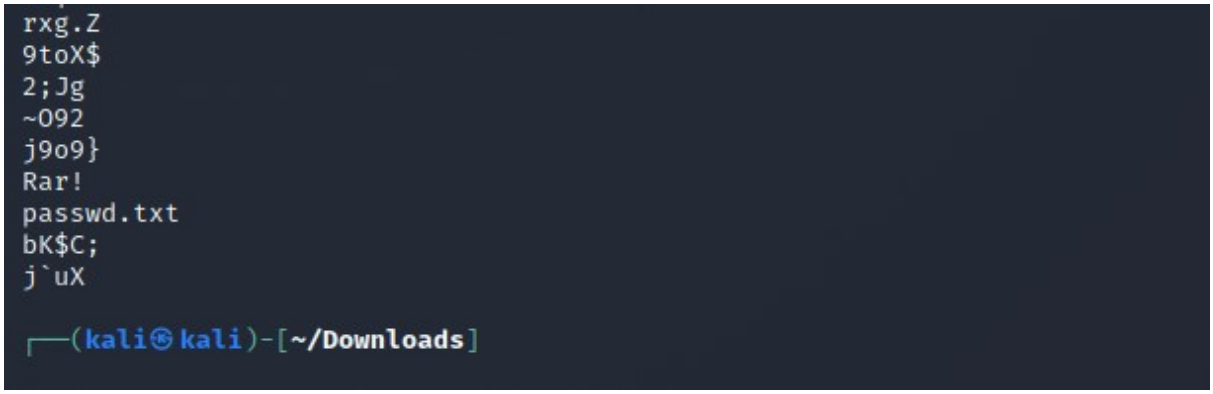

Figure 3.35: Strings Found

And I found something. There seems to be somthing called passwd.txt inside the file! This means that this "image" is hiding another file inside it.

So next I install binwalk and run command: binwalk -e ftuna.jpg

-e will extact the files.

Figure 3.36: BinWalk commandline

this produces a folder "_ftuna.jpg.extracted" that contains the passwd.txt file and from that file the password can be retrieved.

```
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
The password is LordVoldermortSucks
```

Figure 3.37: Password found

Bibliography

Common Weakness Enumeration, a. n.d. “CWE-565: Reliance on Cookies without Validation and Integrity Checking,” https://cwe.mitre.org/data/definitions/565.html.

Kaczanowski, Megan. 2021. “What is a Buffer Overflow Attack – and How to Stop it,” https://www.freecodecamp.org/news/buffer-overflow-attacks/.

OWASP. 2022. “Password Storage Cheat Sheet,” https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.

———. n.d. “SQL Injection Prevention Cheat Sheet,” https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.