**Memory Loss – Find your way back (Workflow/WriteUps for all CTFs)**

The order corresponds to the order of the CTFs in the story and in the CTF overview. The same CTFs will appear in the story.

7) CTF - Web Exploitation (2):

* The user must find the flag in the html source.
* Open the right click on the challenge page --> open inspector view --> Element tab.
* To find the flag you can either look manually for the source of the html code or: CTRL+F and search for "puzzlePiece"
* The first part is located in the code right after the five shelfs in a hidden <div>
* Flag: puzzlePiece{og9XEhgLE2}

4) XSS (1):

* In this challenge, the player should inject an “alert(“XSS”)” function using a script tag.
* This script will be executed from the local client. One of the correct script tags, can be this “<script> alert("XSS");</script>”
* It is also possible to use <script alert("XSS")/>, but keep in mind, that is a html one line tag.
* In this CTF no flag is necessary.
* The execution of this XSS is simulated because the browser will not execute it automatically.

1) - CTF - Web Exploitation (1):

* The goal here is for the user to find his login data in the source code since they are hardcoded in their, base64 encoded.
* Open inspector --> search for ("Incorrect Username or Password")
* In this function you should find the login data base64 decoded. The user can you an online base64 decoder like this. <https://www.base64decode.org/> to decode the user and password.
* Username (deocded): www-admin
* Password: puzzlePiece{DhFpMXHyX5}
* The password is also the flag.

11) CTF - Cryptography (3):

* Copy the text
* Go to an encryption identifier to identify the encryption method or look at the text. You can see that there are words with some rotated characters.
* The text is Vigenère decrypted, the user can you this online tool <https://gc.de/gc/vigenere/> to decrypt the text using the key written in the CTF.
* The flag is written in the decrypted text.
* The user must wrap the flag into puzzlePiece{}
* puzzlePiece{Amj8u4y2Ka}

5) - CTF - XSS (2):

* Here the goal is to teach the player that other HTML tags can be used in XSS. The image in the browser should be a hint for using the image tag.
* For that the player must define an arbitrary location for the image and use the attribute onerror to call the alert()-function.
* To complete the ctf the use must create an Image tag like this:

“<img src='#' onerror=alert(1) />”

* This image tag, will execute the javascript alert(1), when the image could not be loaded. In this case, the img “#” could not be loaded and this cause the JavaScript to be executed.

 3) CTF - Forensics (2):

* For this challenge you should copy the text. And then go to an online Base64 converter. (Example: [Base64 to Image Decoder / Converter (codebeautify.org)](https://codebeautify.org/base64-to-image-converter))
* The flag is on the image you generate form converting the string.
* puzzlePiece{TJeqNhQ5C9}

10) CTF - Cryptography (2):

* This challenge can be done manually using pen and paper or using online tools or by writing a small program.
* The user must map very given number to character
* A=1, B=2, C=3, Z=26
* <https://www.rentfort.de/>
* puzzlePiece{FHZJDFGETR}

2) - CTF - Forensics (1):

* To pass this challenge, you will need to download the image.
* Right click on the file --> properties --> Details. The flag is in the title field.
* The flag is in the details of the file.
* Flag: puzzlePiece{gx6wQudvZb}

 8) CTF - Cryptography (1):

To complete this challenge the user has to write a small program that does the following:

* Takes each number from the list an calculates it to mod 42
* Takes the results and maps it to scheme defined in the challenge
* The output should be wrapped into puzzlePiece{FLAG\_HERE}
* You can use python-online compiler: <https://www.online-python.com/>
* Example code:

#!/bin/pyhton

list = [181, 152, 103, 335, 147, 256, 101, 192, 125, 396, 172, 170, 104, 269, 214, 377, 288, 324, 401, 142, 163]

char = ['#', '!', '"', '{', '}']

for i in list:

x = i % 42

if 0 <= x <= 25:

print(chr(65+x), end='')

if 26 <= x <= 35:

print(chr(48+x-26), end='')

if 36 <= x <= 40:

print(char[x-36], end='')

if x == 41:

print("\_", end='')

* Flag: puzzlePiece{N0T\_VERY\_SECURE\_#4XQ!}

9) CTF - Reverse Engineering:

* For this challenge you can either manually sort the characters of the arrays from 0 to 9
* Or the faster way is to copy the code to your programming environment of choice and change the method encryptSecret to create a string from the char. The main method should call the function encryptSecret()
* Flag: puzzlePiece{7tG8eJ58eY}
* Example code:

public class ctf9 {

static char [] theSecret;

public static String encryptSecret() {

theSecret = new char [10];

theSecret [2] = 'G';

theSecret [5] = 'J';

theSecret [0] = '7';

theSecret [6] = '5';

theSecret [8] = 'e';

theSecret [3] = '8';

theSecret [9] = 'Y';

theSecret [1] = 't';

theSecret [4] = 'e';

theSecret [7] = '8';

return new String(theSecret);

}

public static String conclusion() {

String conc = "This case is unsolvable. Give u p!";

return conc;

}

public static void main(String[] args) {

System.out.println(encryptSecret());

}

}

6) CTF - XSS (3):

* Here the player should notice that he/she can edit the URL. Another thing we point the players attention to is the code that can be accessed through the light blue icon.
* The player should investigate the code how the result HTML is generated. He/She will notice that user input is used without any validation. Here it is possible to inject a image-tag with an on-error field.
* A solution for the url would be:

“https://twine.ctf/ctf6.html#3' onerror='alert();//”

* With this solution, the old image will try to old the image but will fail and execute the onerror method. The JavaScript alert() will be executed.
* No flag required