Contents

Assignment 1 WriteUp:

WarmUp:

Sanity check :

cs2107{246c00aeecddfe7aa00ffa1ac5adf948}

Hashlet:

By using md5sum command, we are able to find the checksum of the file existence.txt

```
nnythingy@DESKTOP:/mnt/d/Workable Shit/notes/School/School given/CS2107/Assignment/Assignment 1/hashlet$ md5sum existence.txt
7579ce22801161f6db141d075fb3191d  existence.txt
```

Flag : cs2107{7579ce22801161f6db141d075fb3191d}

Grep what you want:

Unzipping the compressed file reveals a folder full of binary files

```
nnythingy@DESKTOP:/mnt/d/Workable Shit/notes/School/School given/CS2107/Assignment/Assignment 1/grepWhatYouWant/data here$ grep cs2107 *
Binary file dump112.txt matches
nnythingy@DESKTOP:/mnt/d/Workable Shit/notes/School/School given/CS2107/Assignment/Assignment 1/grepWhatYouWant/data here$ strings dump112.txt | grep cs2107
cs2107{gr4pping_w4d_y0u_want_101}
```

Flag: cs2107{gr4pping_w4d_y0u_want_101}

Convert Convert:

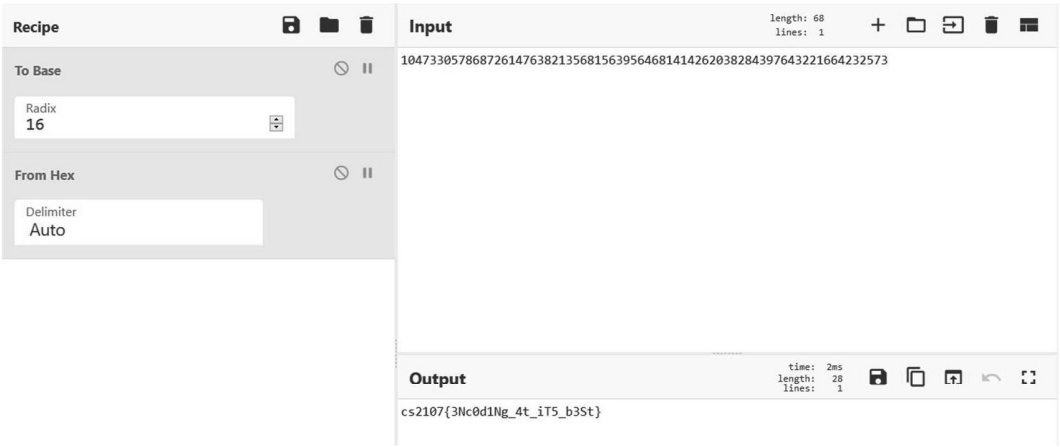


Figure 1: Changing the message from long to Hex, and then reading from Hex

Flag: cs2107{3Nc0d1Ng_4t_iT5_b3St}

Legit:

Password Cracking:

Creating a wordlist of information from her portfolio (file: passwordCracking_dict)
From the file (file:passwordCracking_cipher.py), encrypting takes in a password, pads it to be length of 32 by adding 0, using the function pad_key then uses that as the key along with the IV "helloworldcs2107".
To use the function decrypt(), the ciphertext (file: ciphertext), there would be a need to pad all password attempts if they are not length 32.
Therefore after editing and creating for loops to iterate through every possibility, the flag is found after appending 3 words from the dictionary together creating the password "accountantblue1976"

Flag: cs2107{pl5_us3_secur3_p4ssw0rd5}

Dear Husband:

Given: $p, g, g^a mod\ p\ , g^b mod\ p$, Diffie-Hellman key exchange tells us that to find the shared secret key would require $g^{ab} mod\ p = (g^a)^b\ mod\ p = \ (g^a\ mod\ p)^b\ mod\ p$. This would thus give us a discrete logarithm problem.

Using a discrete logarithm solver, we come to
b = 276498253302657425992663427422165482 0
a = 34991665369016937662864669287514076 46
Thus their secret key:
$80333195172482319605472656234018317339 1^{2764982533026574259926634274221654820}\ mod\ 84552981618332883228882682797894409243 3$
= 65592628605351277863618058470992509921 2

Flag: cs2107{655926286053512778636180584709925099212} (file: dearHusband_secret.txt)

Custom Protocol:

The encryption scheme takes the flag, and using the seed (2107), will shift each letter by a "random" amount from the list of character in alphabet. This is done by adding the value of shift to the index of the current letter of the flag.

Thus to solve we would need to take the ciphertext and shift it back by the same "random" index using the same seed (2107), obtaining the letter in the flag.

```
nnythingy@DESKTOP:/mnt/d/Workable Shit/notes/School/School given/CS2107/Assignment/Assignment 1/customProtocol$ python3 test.py
cs2107{i_am_a_hackerman}
```

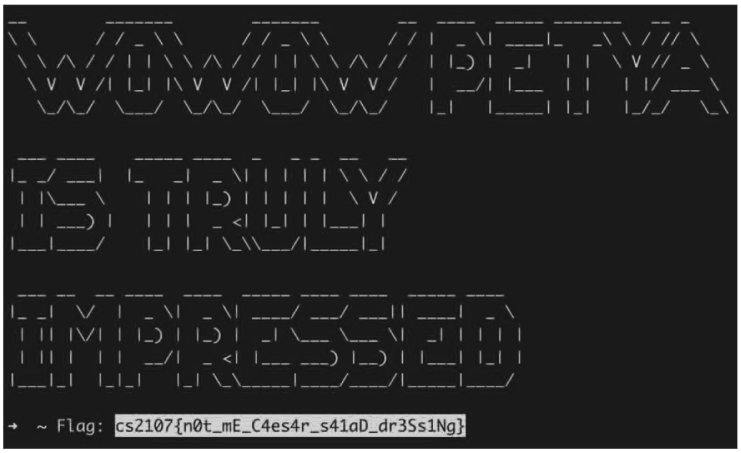
Flag: cs2107{i_am_a_hackerman} (file: customProtocol_attack.py)

Now XOR Never:

Since knowing the file is supposed to be a .png file, the first 8 bytes of the image is supposed to be `89504E470D0A1A0A`. After using the command: hexdump encrypted_file.png > hexdump it shows that the 1st 8 bytes of encrypted_file.png is `60ED223E3A696F6A`. Thus being able to XOR the 2 values (plaintext and ciphertext) will output the key.

However, doing the same process on a local .png file shows that the hex encoding is `5089474E0A0D0A1A` indicating that there is a need to change the endianness to obtain the key which is d0ped0pe.

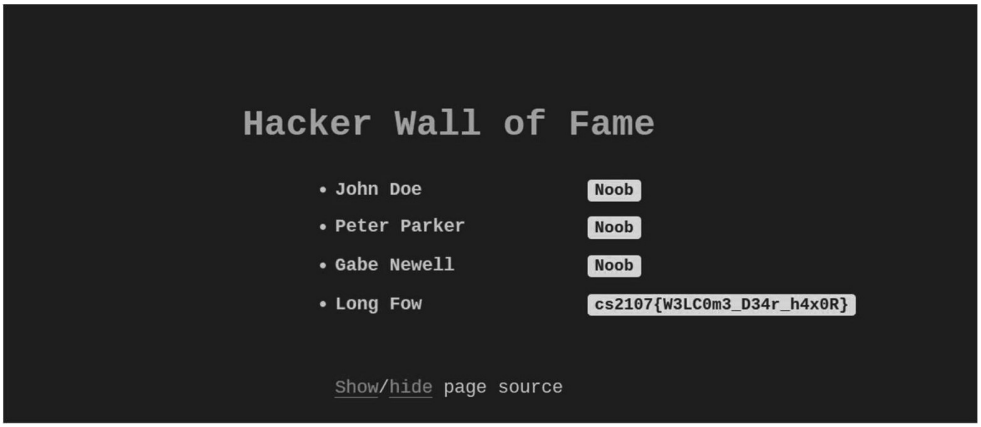
Thus, placing the file into CyberChef and using the key to perform XOR operation, saving and opening the file would give the decrypted image.



Flag: cs2107{n0t_mE_C4es4r_s41aD_dr3Ss1Ng}

Hacker Wall of Fame:

The challenge requires a Hash Extension attack, where we append our own data to the cookie: "users" and generate our own hash signature to place in the cookie: "signature". Thus using hash extender to generate a hash that is valid after appending a user with the "level" tag of 999 to obtain the flag.



Flag: cs2107{W3LC0m3_D34r_h4x0R} (files: hackerWallOfFame_hashExtentionCommand, hackerWallOfFame_results)

Password Manager:

The password manager is using AES-ECB, this would mean that plaintexts which are similar will have the same ciphertext.

1st Since all passwords are alphabetically sorted, this means a plaintext of 'A' will go to the top.

2nd Then after finding out that the block size is 16 bytes for plaintext, which would then produce a 32 bytes ciphertext

3rd Then realizing each input will have a newline character appended to them.

4th Thus when decrypting, just need to ensure we add a newline before the character we want to decrypt

5th For each iteration reduce the number of 'A's added in front to slowly leak out the flag

Flag: cs2107{pl5_p15_p1s_d0n7_u5e_ec8} (file: passwordV1_attack.py)

Password Manager v2:

1st Since we know that this is AES-CBC encrypted, we know the ciphertext is first decrypted then Xor with the "IV" block / previous ciphertext block to create the plaintext, we need to we need to find the block size – 16 bytes

2nd Then since the padding is in the form of 0x01 0x02 … 0x10. Means that every new block to find will be in the 0x01 padding slot.

3rd To find the plaintext (P), $P = P' \oplus C \oplus C'$. And where P' is the padding 0x01, C is the byte of ciphertext in the previous block and C' is the new byte of ciphertext to create the correct padding.

4th After finding each of the previous P, to find the next one, we have to adjust their padding accordingly by changing the corresponding ciphertext in the previous block where $C' = P \oplus C \oplus P'$

Flag: cs2107{k33p_Y0uR_3rr0r_m3Ss4G3S_70_y0Urs3LF} (file: passwordv2Attack.py)

Bonus:

Who is Petya:

The original image seems to contain a text. After inverting the image colour, the text is much clearer.

The text D4hLiAs is then used to unzip the password protect compressed file and, in the file, evilplans.txt we find the flag.

Flag: cs2107{sTuxXxXY_w000000000RlD_D0m1Ni0N}