CS2107 Assignment 1

Easy Challenges

Sanity Check

1. I locate mouse
2. I scroll down
3. I see flag
4. I copy
5. I paste

flag: `CS2107{l3t_7He_j0uRn3Y_b3g1N}`

Something's Off

1. The Characters `{`, `}`, `_` seems to match the usual CTF format, so we assume those are not characters.
2. based on characters shown we only have `a-zA-Z0-9`.
3. We get the ascii value with `ord()` in python, then use a char array to shift the characters.

flag: `CS2107{5h1ft_c1ph3rs_4r3_4_gR8_w4rMuP}`

Script:

```
num = "0123456789"
lower = "abcdefghijklmnopqrstuvwxyz"
upper = lower.upper()


lookup = list(num + upper + lower)
maxShift = len(lookup)

if __name__ == "__main__":
    prefix = r"QgGFEL{JvFt7_qF3vH56_I5H_I_ufM_AI5a8d}"
    res = []
    for offset in range(1, maxShift):
        temp = []
        for i in prefix:
            if i == '{' or i == '}' or i == '_':
                temp += i
            else:
                char = ord(i)
                index = 0
                if char < 65:
                    char -= 48
                elif char < 97:
                    char -= 55
                else:
                    char -= 61
                temp += lookup[(char + offset) % maxShift]
        res.append("".join(temp))
        print(res[-1])
```

HMAC

- From string content:

flag: `CS2107{28025dd41abceecad6056fd5b99587feac67089e6cc874679ab75e0c335a9a67}`

command:

```
openssl
sha256 -hmac CS21072022 text.txt
```

Secret penguin

flag: `CS2107{4851ed69abe9830dda4ecca87c4634aef98ef8c2f9d7060e8ec5aaedf787a262}`

```
openssl
aes-128-cbc -iv abcdef1234567890abcdef1234567890 -K 1234567890abcdef1234567890abcdef -in tux.png -out tux.out
sha256 tux.out
```

E5.

- idk

Medium

Insecure OTP

1. Known plaintext attack, xor first 20 bytes of `p` and `c` to obtain key.

flag: `CS2107{OTPOTP_0tp0tp_R3p3at_k3y_15_vuln3rable}`

```
if __name__ == "__main__":
        plaintext = "Hey Grandma Susan'oo, I have told you not to play with my Photoshop!".encode()
        bytes_ciphertext = "faa4a0ba8d435a2b2015c4625c80443e820c523a9ee190baa2504d20640cca2e6bd54e30990b533ac6e1adf5e
        ciphertext = bytes.fromhex(bytes_ciphertext)

        key = xor(plaintext[:20], ciphertext[:20])
        dec = encrypt(key, ciphertext)

        print(dec.decode())
```

John the ripper

- Use John the ripper

flag: `CS2107{abcd1234}`

Birthday hash.

- Birthday paradox: It is extremely difficult to find a person with a birthday equals to a specific date, but it is surprisingly common for two people to have the same birthdays in a small crowd.
- We compute various possible strings until we find 2 strings that cause a hash collision.

flag: `CS2107{No_h@sh_can_esc4pe_b1rthd@y_p@rad0x}`

script:

```
from hashlib import sha512

memory = {}

print("enter hex: ", end="")
event = bytes.fromhex(input())

def compute(q1):
        return sha512(event + q1).digest()[:6]

for i in range(6969696969):
        strVal = str(i)
        byteVal = strVal.encode()
        res = compute(byteVal)

        if i % 5000000 == 0:
                print("5 mil done")
                print("computing: {}".format(strVal))

        if res in memory:
                print("hash collision det")
                print(strVal)
                print(memory[res])
                break
        else:
                memory[res] = strVal
```

Perfect AES imperfect key

- There are only 3 bytes that are used in the sha512 key used for encryption
- Approx 16 million combinations

```
def generateCt(seed):
        key = sha512(seed).digest()[:16]
        return key

def decryptCt(iv, key, cipherText):
        cipher = AES.new(key, AES.MODE_CBC, iv)
        try:
                res = unpad(cipher.decrypt(cipherText), AES.block_size)
                print(res.decode("utf-8"))
        except ValueError as e:
                pass

if __name__ == "__main__":
        cipherText = bytes.fromhex("c089a2553fdcbb0bbdbd7655fc34c75eb7f2ccd28fc801480c5a15b7f366f8737a30aa3e845d79e50
        iv = bytes.fromhex("4b0fb9a4dfbabe6810b2fb01d2012b84")
        for i in range(pow(2, 8 * 3)):
                decryptCt(iv, generateCt(i.to_bytes(3, "big")), cipherText)
```

Substitution cipher

`UT2107{` : "C" = "U", "S" = "T".

`(H), (HH), (HHH)...` : "I" = "H", "V" = "M",

`B., Q., U.` : "A" = "B", "B" = "Q", "C" = "U", "D" = "P", "E" = "Y", "F" = "O", "G" = "N", "H" = "F", "I" = "H"

- Run decoder once

`WHIS AGXEEKEDW DESCXIBES` : `THIS AGREEMENT DESCRIBES` , "T" = "W", "R" = "X", "M" = "K", "N" = "D", "T" = "W"

`DISCJAIKEX` , "L" = "J"

- Run decoder again

`RARRANTG` : "W" = "R", "Y" = "G"

`LIMITATILN` : "O" = "L"

`DEMICE MANAFACTARER` : "V" = "M", "U" = "A"

`ADOBE FLASH SLAYER LICENSE TERMS` : "P" = "S"

flag: `CS2107{SUBSITUTION_CIPHER_IS_OFTEN_SHOWN_IN_MOVIE_FOR_SOME_REASON}`

HARD

COPPER RSA

Reference

- Since we have `[c1,..., c5], [n1, ..., n5]` , we can use Chinese Remainder Theorem to find M using Hastad's Broadcast Attack:

```
def chinese_remainder(n, a):
    sum = 0
    prod = reduce(lambda a, b: a*b, n)
    for n_i, a_i in zip(n, a):
        p = prod // n_i
        sum += a_i * mul_inv(p, n_i) * p
    return sum % prod

def mul_inv(a, b):
    b0 = b
    x0, x1 = 0, 1
    if b == 1: return 1
    while a > 1:
        q = a // b
        a, b = b, a%b
        x0, x1 = x1 - q * x0, x0
    if x1 < 0:
```

- Using mathematica, we compute cube root of M, m. We then find the roots from the quadratic equation.

```
b = a^(1/3)
Solve[4*x^2 + 521  * x + 47829 == b, x]
```

- We get 1 negative root and 1 positive root. We can then use `long_to_bytes()` and `decode()` to get the text:

```
THIS FISH IS SO RAW CS2107{c0pP3r_br@s5_Br0nz3_m3tAl_s73el_1r0n_Go1d} HE'S STILL FINDING NEMO
```