CS2107 ASSIGNMENT 2 REPORT

Nicholas Sun Jun Yang, A0217609B

E.1 Can you GDB? (Reverse Engineering)

Description

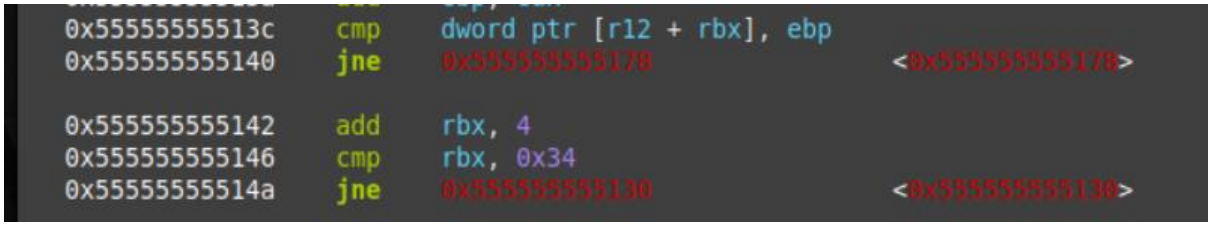
Given a binary file, the flag is to be extracted from it using dynamic analysis through gdb.

Analysis

Running the file through gdb using the command “gdb -q path/to/binaryfile” and then executing “r” creates a prompt to enter a password. The binary file contains an entry point for the user to enter the password and there is an output that tells the user whether the password entered was valid. Upon interrupting the control flow by inputing Ctrl+C and then stepping out using “finish”, a random password can be entered first then the code is stepped through until the fgets function is reached. This would be where the main function is since fgets is used to get the user input, and the assembly instructions and registers can then be analysed to determine the contents of the flag.

Solution

There are two registers($ebp and $r12) and one instruction (the first cmp instruction of the image below)that are important. This is because the instruction compares the content in ebp with the contents located at the address $r12+ $rbx where the value of $rbx starts from 0 and is incremented by 4 every loop and determines whether the loop should exit and return a “wrong password” or “correct password” message.



During the first loop, the contents of $ebp reads as an integer 825381699, which converts to 0x31325343. Using an ASCII table to convert the value into ascii we get 12SC which we can reverse to get CS21. This gives a big clue of

what to do to get the remaining parts of the flag. Since the input password is read 4 bytes at a time, the value of the password can be dynamically modified by using a “set $r12 “CS21” and then the next instruction of the code can be ran which will not cause the jne instruction to run since the values of the first 4 bytes of $r12 and $ebp have been made to be equal. This can then be repeated for the next series of loops where the other values of $ebp is obtained and then reverse engineered into next 4 bytes of the password and the newly obtained part of the flag can be appended to the value in $r12. After 13 loops, the parts of the flag obtained is retrieved and combined to get the flag.

Flag

CS2107{y4y_y0u_c4n_us3_4_d3bugger!GDB_my_best13}

E.2 keylogger (Forensics)

Description

Given a wireshark packet capture file, information about key logs can be extracted to determine the flag.

Analysis

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	2.10.1	host	USB	35	URB_INTERRUPT in
2	0.001020	2.10.1	host	USB	35	URB_INTERRUPT in
3	0.002992	2.10.1	host	USB	35	URB_INTERRUPT in
4	0.004994	2.10.1	host	USB	35	URB_INTERRUPT in
5	0.007043	2.10.1	host	USB	35	URB_INTERRUPT in
6	0.008004	2.10.1	host	USB	35	URB_INTERRUPT in
7	0.009036	2.10.1	host	USB	35	URB_INTERRUPT in
8	0.011035	2.10.1	host	USB	35	URB_INTERRUPT in
9	0.013031	2.10.1	host	USB	35	URB_INTERRUPT in
10	0.014035	2.10.1	host	USB	35	URB_INTERRUPT in
11	0.015028	2.10.1	host	USB	35	URB_INTERRUPT in
12	0.016036	2.10.1	host	USB	35	URB_INTERRUPT in
13	0.016986	2.10.1	host	USB	35	URB_INTERRUPT in
14	0.018036	2.10.1	host	USB	35	URB_INTERRUPT in

Opening the file gives us a lot of the same type of information as shown above. The leftover capture data provides a way to extract the key logs information.

Solution:

The solution was obtained using this tools. Running this command `tshark -r ./chall.pcap -Y 'usb.capdata && usb.data_len == 8' -T fields -e usb.capdata | sed 's/../:&/g2' > keylogger.txt` extracts the USB traffic data and selects only packets where the USB data length is 8 bytes. The

contents are then copied into a text file for later analysis. Using the usbkeyboard python script downloaded from the github page which can be seen below, the contents of the payload were then read against the HID tables and the comparisons were automated. The flag is then output through the command “python3 usbkeyboard.py keylogger.txt”.

```
nickysun@nickysun-VirtualBox:~/Downloads$ python3 usbkeyboard.py keys.txt
aaaaaaaeaacaaaaadaaaaaa


CS2107{K3YL0GGER_1S_@CTIVATED}aaaadaaaaaaeaabaaaabbaacbabbfbbbabbbbcbbbababbdaab
bbaacaaaaaaaaaaaaabanickysun@nickysun-VirtualBox:~/Downloads$
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys

#More symbols in
https://www.fileformat.info/search/google.htm?q=capslock+symbo
l&domains=www.fileformat.info&sitesearch=www.fileformat.info&c
lient=pub-6975096118196151&forid=1&channel=1657057343&ie=UTF-
8&oe=UTF-
8&cof=GALT%3A%23008000%3BGL%3A1%3BDIV%3A%23336699%3BVLC%3A6633
99%3BAH%3Acenter%3BBGC%3AFFFFFF%3BLBGC%3A336699%3BALC%3A0000FF
%3BLC%3A0000FF%3BT%3A000000%3BGFNT%3A0000FF%3BGIMP%3A0000FF%3B
FORID%3A11&hl=en
KEY_CODES = {
    0x04:['a', 'A'],
    0x05:['b', 'B'],
    0x06:['c', 'C'],
    0x07:['d', 'D'],
    0x08:['e', 'E'],
    0x09:['f', 'F'],
    0x0A:['g', 'G'],
    0x0B:['h', 'H'],
    0x0C:['i', 'I'],
    0x0D:['j', 'J'],
    0x0E:['k', 'K'],
    0x0F:['l', 'L'],
    0x10:['m', 'M'],
    0x11:['n', 'N'],
    0x12:['o', 'O'],
    0x13:['p', 'P'],
    0x14:['q', 'Q'],
    0x15:['r', 'R'],
    0x16:['s', 'S'],
    0x17:['t', 'T'],
    0x18:['u', 'U'],
    0x19:['v', 'V'],
    0x1A:['w', 'W'],
```

```
    0x1B:['x', 'X'],
    0x1C:['y', 'Y'],
    0x1D:['z', 'Z'],
    0x1E:['1', '!'],
    0x1F:['2', '@'],
    0x20:['3', '#'],
    0x21:['4', '$'],
    0x22:['5', '%'],
    0x23:['6', '^'],
    0x24:['7', '&'],
    0x25:['8', '*'],
    0x26:['9', '('],
    0x27:['0', ')'],
    0x28:['\n','\n'],
    0x29:['ᄐ','ᄐ'],
    0x2a:['□', '□'],
    0x2b:['\t','\t'],
    0x2C:[' ', ' '],
    0x2D:['-', '_'],
    0x2E:['=', '+'],
    0x2F:['[', '{'],
    0x30:[']', '}'],
    0x32:['#','~'],
    0x33:[';', ':'],
    0x34:['\'', '"'],
    0x36:[',', '<'],
    0x37:['.', '>'],
    0x38:['/', '?'],
    0x39:['ᄒ','ᄒ'],
    0x4f:[u'→',u'→'],
    0x50:[u'←',u'←'],
    0x51:[u'↓',u'↓'],
    0x52:[u'↑',u'↑']
}


#tshark -r ./usb.pcap -Y 'usb.capdata && usb.data_len == 8' -T
fields -e usb.capdata | sed 's/../:&/g2' > keyboards.txt
def read_use(file):
    with open(file, 'r') as f:
        datas = f.readlines()

    datas = [d.strip() for d in datas if d]
    cursor_x = 0
    cursor_y = 0
    lines = []
    output = ''
    skip_next = False
    lines.append("")

    for data in datas:
```

```
        shift = int(data.split(':')[0], 16) # 0x2 is left
shift 0x20 is right shift
        key = int(data.split(':')[2], 16)

        if skip_next:
            skip_next = False
            continue

        if key == 0 or int(data.split(':')[3], 16) > 0:
            continue

        #If you don't like output get a more verbose output
here (maybe you need to map new rekeys or remap some of them)
        if not key in KEY_CODES:
            #print("Not found: "+str(key))
            continue

        if shift != 0:
            shift=1
            skip_next = True

        if KEY_CODES[key][shift] == u'↑':
            lines[cursor_y] += output
            output = ''
            cursor_y -= 1

        elif KEY_CODES[key][shift] == u'↓':
            lines[cursor_y] += output
            output = ''
            cursor_y += 1

        elif KEY_CODES[key][shift] == u'→':
            cursor_x += 1

        elif KEY_CODES[key][shift] == u'←':
            cursor_x -= 1

        elif KEY_CODES[key][shift] == '\n':
            lines.append("")
            lines[cursor_y] += output
            cursor_x = 0
            cursor_y += 1
            output = ''

        elif KEY_CODES[key][shift] == '[BACKSPACE]':
            output = output[:cursor_x-1] + output[cursor_x:]
            cursor_x -= 1

        else:
            output = output[:cursor_x] + KEY_CODES[key][shift]
+ output[cursor_x:]
```

```
            cursor_x += 1

    if lines == [""]:
        lines[0] = output

    if output != '' and output not in lines:
        lines[cursor_y] += output

    return '\n'.join(lines)

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print('Missing file to read...')
        exit(-1)
    sys.stdout.write(read_use(sys.argv[1]))
```

Flag

CS2107{K3YLOGGER_1S_@CTIVATED}

E.3 Babypwn (Application Security: Binary Exploitation/Pwn)

Description:

Given a binary file and the logic of the code written in C(below), the flag is to be extracted from a server using buffer overflow exploitation.

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>


int setup() {

  setbuf(stdin, 0);

  setbuf(stdout, 0);

}


int func() {
```

```
    char user[0x10] = "user";
    char buf[0x10];

    printf("Hello, what's your name?\n");
    fgets(buf, 0x1C, stdin);
    printf("Hello %s\n", buf);
    if (!strncmp(user, "sudo", 4)) {
        printf("Good job! Here is your flag:\n");
        system("cat flag.txt");
    } else {
        printf("You do not have permission!\n");
    }
    return 0;
}

int main() {
    setup();
    func();
}
```

Analysis

`char user[0x10] = "user";`

`if (!strncmp(user, "sudo", 4)) {`

If the first four characters of user is sudo, the server outputs the flag.

`char buf[0x10];`

`fgets(buf, 0x1C, stdin);`

The user input is read into buf but the size specified for the input buffer for fgets is larger than the size of buf. This will lead to buffer overflow should an

input larger than 0x10 is entered and the contents of buf may spill over and overwrite the contents of user. This can be exploited to change the contents of user to “sudo”.

Solution

A simple solution is to enter as many sudos into the input as possible etc; “sudosudosudosudosudosudosudo”.

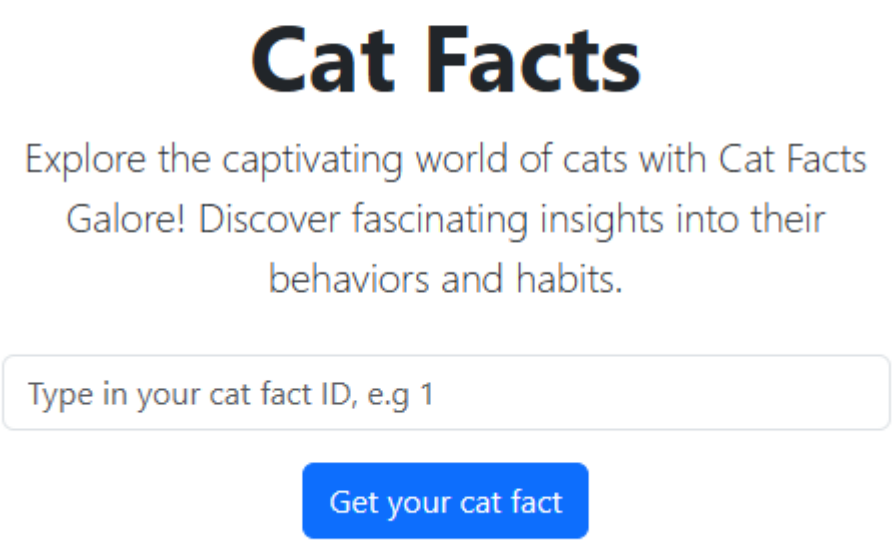
Flag

CS2107{ov3rfl0w_t0_unl1m1t3d_5ud0_4cc355}

E.4 Cat Facts (Web Security)

Description

Given a website and a bunch of files that gives away the logic behind the website, the flag is to be extracted using SQL injection exploitation.



Analysis

App.py:

```
@app.route('/catfact', methods=['POST'])
def get_cat_fact():
    cat_fact_id = request.form.get('id', '')

    try:
        query = f"SELECT id, fact FROM facts WHERE id = '{cat_fact_id}'"
        fact = query_database(query)
    except Exception:
        fact = [(0, "Oops, an error occured!")]

    if fact:
        return render_template('index.html', fact=fact[0][1])
    else:
        return render_template('index.html', fact="No cat facts for this ID :(")
```

```
cat_fact_id = request.form.get('id', '')
query = f"SELECT id, fact FROM facts WHERE id = '{cat_fact_id}'"
```

The contents of the user input is read into a variable cat_fact_id and is then passed into a formatted string.

```
fact = query_database(query)
if fact:
    return render_template('index.html', fact=fact[0][1])
```

If the sql query returns a non empty table, the second column of the table which is the fact is returned as the output.

Thus, a specifically designed user input can be sent to the variable in order to carry out an SQL injection attack.

Solution

The “master key” to be used is ' union select id,(select id from facts) as x from facts where id = 2 order by id desc limit 1 --'. The apostrophe in the front closes the where clause etc; where id = ‘’ and allows additional SQL commands to be padded without creating invalid queries. The SQL code following the apostrophe after was designed to test whether the query could be tampered with by returning the result of the query after “union”.

Next, ' union select id,(SELECT group_concat(tbl_name) FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%') as x from facts where id = 2 order by id desc limit 1 --' was used to get the name of the table containing the flag, which happens to be named “flags”.

Afterwards, ' union select id,(SELECT sql FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name ='flags') as x from facts where id = 2 order by id desc limit 1 --' is used to get the name of the column of the flags table.

Finally the query to obtain the flag could be constructed as ' union select id,(select flag from flags) as x from facts where id = 2 order by id desc limit 1 --'.

Flag

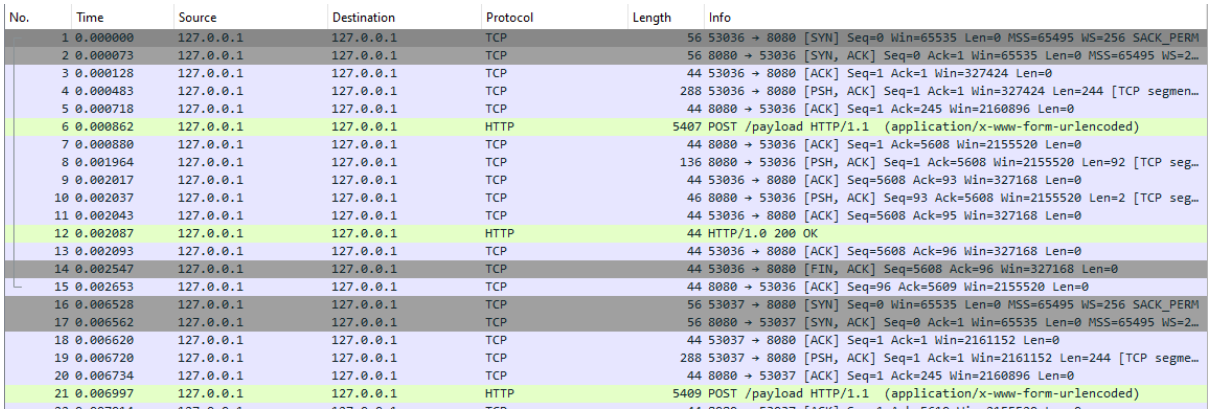
CS2107{SqL_iNj3cT10N_1s_qU1t3_e4sY!}

M.1 exfiltrator64 (Forensics)

Description

Given a wireshark packet capture file, the flag is to be extracted using data exfiltration techniques.

Analysis



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	53036 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000073	127.0.0.1	127.0.0.1	TCP	56	8080 → 53036 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=2…
3	0.000128	127.0.0.1	127.0.0.1	TCP	44	53036 → 8080 [ACK] Seq=1 Ack=1 Win=327424 Len=0
4	0.000483	127.0.0.1	127.0.0.1	TCP	288	53036 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=244 [TCP segmen…
5	0.000718	127.0.0.1	127.0.0.1	TCP	44	8080 → 53036 [ACK] Seq=1 Ack=245 Win=2160896 Len=0
6	0.000862	127.0.0.1	127.0.0.1	HTTP	5407	POST /payload HTTP/1.1 (application/x-www-form-urlencoded)
7	0.000880	127.0.0.1	127.0.0.1	TCP	44	8080 → 53036 [ACK] Seq=1 Ack=5608 Win=2155520 Len=0
8	0.001964	127.0.0.1	127.0.0.1	TCP	136	8080 → 53036 [PSH, ACK] Seq=1 Ack=5608 Win=2155520 Len=92 [TCP seg…
9	0.002017	127.0.0.1	127.0.0.1	TCP	44	53036 → 8080 [ACK] Seq=5608 Ack=93 Win=327168 Len=0
10	0.002037	127.0.0.1	127.0.0.1	TCP	46	8080 → 53036 [PSH, ACK] Seq=93 Ack=5608 Win=2155520 Len=2 [TCP seg…
11	0.002043	127.0.0.1	127.0.0.1	TCP	44	53036 → 8080 [ACK] Seq=5608 Ack=95 Win=327168 Len=0
12	0.002087	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.0 200 OK
13	0.002093	127.0.0.1	127.0.0.1	TCP	44	53036 → 8080 [ACK] Seq=5608 Ack=96 Win=327168 Len=0
14	0.002547	127.0.0.1	127.0.0.1	TCP	44	53036 → 8080 [FIN, ACK] Seq=5608 Ack=96 Win=327168 Len=0
15	0.002653	127.0.0.1	127.0.0.1	TCP	44	8080 → 53036 [ACK] Seq=96 Ack=5609 Win=2155520 Len=0
16	0.006528	127.0.0.1	127.0.0.1	TCP	56	53037 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
17	0.006562	127.0.0.1	127.0.0.1	TCP	56	8080 → 53037 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=2…
18	0.006620	127.0.0.1	127.0.0.1	TCP	44	53037 → 8080 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
19	0.006720	127.0.0.1	127.0.0.1	TCP	288	53037 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=244 [TCP segme…
20	0.006734	127.0.0.1	127.0.0.1	TCP	44	8080 → 53037 [ACK] Seq=1 Ack=245 Win=2160896 Len=0
21	0.006997	127.0.0.1	127.0.0.1	HTTP	5409	POST /payload HTTP/1.1 (application/x-www-form-urlencoded)

A series of consecutive TCP connections were made to send and receive packets using HTTP. The HTTP packets could be exported out and analysed. Opening one such pair of packet shows that first packet is of the form req=* where * is a long stretch of ascii characters and the second packet is just the HTTP OK response message.

Solution

To combine the contents of the packet's payloads, the script below was used.

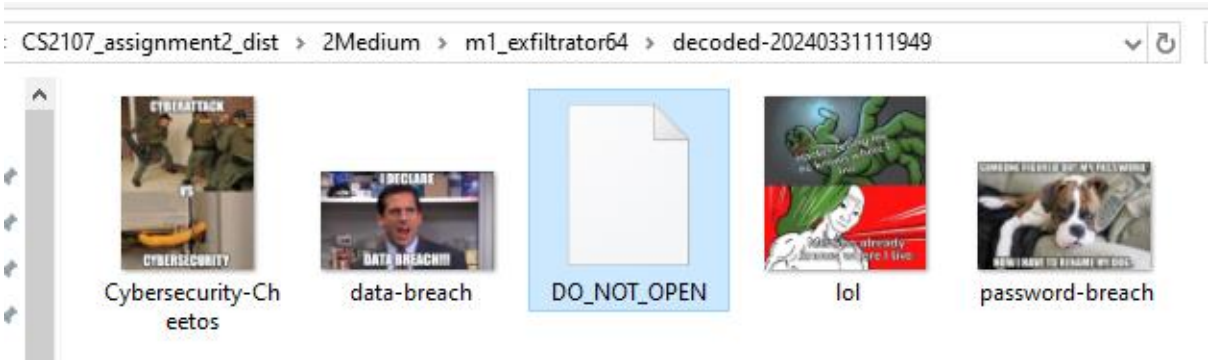
```
import os
import re
# Set the directory where the files are located
directory = #my directory
def extract_number(filename):
    match = re.search(r'\((\d+)\)$', filename)
    if match:
        return int(match.group(1))
    else:
        return 0
# Initialize an empty list to store the combined content
payload_content = []
filenames = sorted(os.listdir(directory), key=extract_number)
```

```
# Loop through the files in the directory
for filename in filenames:
    if filename.startswith('payload'):
        # Open the file and read its contents
        with open(os.path.join(directory, filename), 'r') as file:
            content = file.read().strip()
        # Check if the content contains "req="
        if 'req=' in content:
            # Extract the text after "req="
            req_content = content.split('req=')[1]
            req_content = req_content.split('&id=')[0]
            payload_content.append(req_content)

# Join the combined content into a single string
final_content = ''.join(payload_content).strip()

with open(directory + '\\combined.txt', 'w') as file:
    file.write(final_content)
```

After obtaining the whole payload, since the payload was uncoded in url,on top of being encoded in base64, some websites came in handy to help me decode it. This returns a zip file which unzips into 5 files but only 1 file is of importance.



The main mechanism of the file is given as

```
def controller():

    cat = r'''
        /\_/\
        ( o.o )
        > ^ <
==CS2107 Secret C2 Server==
    '''
    print(cat)

    input_str = """
    input_str = input("Give me the secret key: ")

    is_valid, content = check_one(input_str)

    loading(50)
    if not is_valid:
        print("incorrect check 1")
        exit()
```

```
    print("Correct 1")

    correct = check_two(content[:20])

    loading(75)
    if not correct:
        print("incorrect check 2")
        exit()

    print("Correct 2")

    correct = check_three(content[20:])

    loading(100)
    if not correct:
        print("incorrect check 3")
        exit()

    print("Correct 3")

controller()

def check_one(input_str):

    if len(input_str) != 48:
        return False, "wrong len"
```

```
    if not input_str.startswith("CS2107{") or not input_str.endswith("}"):
        return False, "no fL@g header"

    content = input_str[len("CS2107{"):-1]

    return True, content
```

The check one function simply returns the contents of the string after the "CS2107{".

```
correct = check_two(content[:20])
def check_two(input_two):

    wwww = "771010022c177a5c0c772a154b34625f54005200"
    bbb = list(bytes.fromhex(wwww))
    for i in range(len(input_two)):
        if ord(input_two[i]) ^ bbb[i] != ord(kkk[i]):
            return False

    return True
```

To pass the check two function, the following function was used to determine the first 20 characters of "content".

```
ascii_table = [chr(i) for i in range(128)]
wwww = "771010022c177a5c0c772a154b34625f54005200"
bbb = list(bytes.fromhex(wwww))
pw = "CS2107{"
#starting index = 7
```

```
for i in range(0,20):
    for j in range(len(ascii_table)):
        if ord(ascii_table[j]) ^ bbb[i] == ord(kkk[i]):
            pw += ascii_table[j]
            break
print(pw) #first half, so now we have CS2107{SuspIc1ou$_exF1l7rat, 27 CHARACTERS
correct = check_three(content[20:])
```

The check three function is simply a super nested one direction if statement tree, with each if statement comparing the values of one or more characters in content[20:] to another character or a literal number. For example, "if (ord(user_input[18]) == 84):" and "if (ord(user_input[7]) * 6 * ord(user_input[1]) == 56610):". Passing of all the if statements is needed to pass the function and each if statement relates to a unique character. Thus, through observation, creating mathetmatical equations and solving them, the remaining part of the flag is obtained as "sec_half = [105,111,110,95,48,98,102,85,115,99,64,116,105,48,110,95,104,84,84,112]".

Flag

CS2107{SuspIc1ou$_exF1l7ration_0bfUsc@ti0n_hTTp}

M.3 Cat Breeds (Web Security)

Description

Given a website and the website logic, extract the flag using a blind SQL injection attack.

Analysis

App.py

```
@app.route('/catbreed', methods=['POST'])
def get_cat_fact():
    cat_breed = request.form.get('breed', '')
```

```
    if "sleep" in cat_breed.lower():
        return render_template('index.html', message="Obviously there's no
Sleep cat breed, duh!")

    try:
        query = f"SELECT id, breed FROM cat_breeds WHERE breed =
'{cat_breed}'"
        query_result = query_database(query)
    except Exception:
        query_result = []

    if query_result:
        return render_template('index.html', message="Cat breed exists!")
    else:
        return render_template('index.html', message="Cat breed does not exist
:(")
```

From the app logic, the cat breed variable can be injected with a specially designed input to override the original SQL query. Given that the only information provided by the website output reveals whether the query result is a non-empty table, it is not going to be as straightforward as to repeat the query from the challenge Cat Facts. The way to get the flag would be to first get the length of the flag and then to check the value of each character against a range of ascii values and narrowly filter out the wrong ascii values to get the correct character of the flag.

Solution

```
iguana' union select id,(select flag from flags) as x from cat_breeds where
length(x) = 57 order by id desc limit 1  --'
```

This query was used to test out a possible format for the attack. The where clause is closed using an intentionally chosen animal that is not a cat breed. This is to make the original SQL query return an empty table so as to only make the results of the padded select statement matter. The length of the flag was obtained through trial and error but one way to do it is to start off with a random number say 100 and check whether the length is less than 100. Then if valid, binary search can be used to narrow down the length.

Similarly the characters of the flag can be obtained through testing whether they fall within one of the three ascii ranges: 32-64,65-95 and 96-122, then slowly narrow them down using binary search. An example input to do so is iguana' union select id,(select flag from flags) as x from cat_breeds where substr(x,8,1) BETWEEN 'a' and 'z' order by id desc limit 1 --'.

Flag

CS2107{ bL1nd_1s_n0t_4_Pr0BL3m_f0R_ThE_m1gHtY_Sqli_mASTeR}

H.4 Presentation (Forensics)

Description

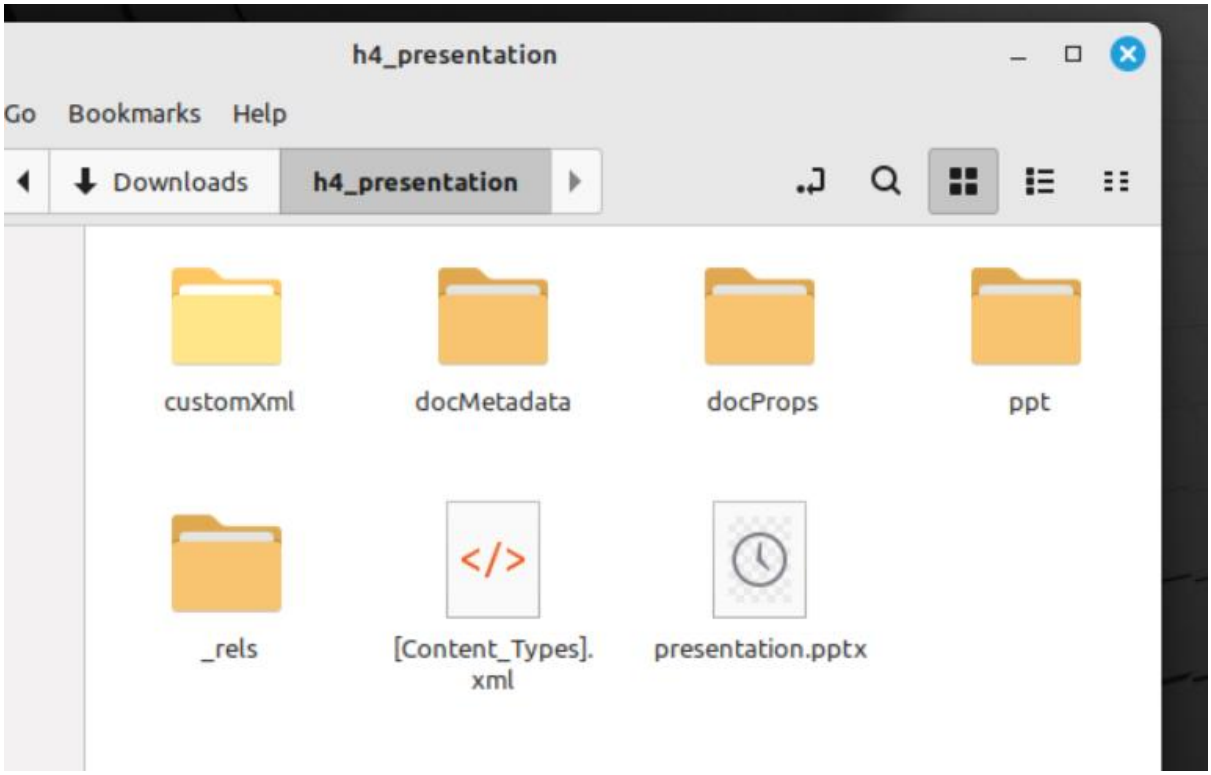
Given a powerpoint file that is corrupted, extract out the flag that is hidden inside a file embedded in the powerpoint file.

Analysis

Opening the powerpoint file gives an error on the file content. The file could still be opened but with a message stating that certain content was removed. Scrolling through the file, the only interesting piece of information was a password hidden in the slide notes: B3stP@$$w0rd1sH3lloWorld.

Solution

Opening terminal and applying the unzip command on the powerpoint file extracts out the contents.



Going into ppt/media, there is a Doc.zip file which can be extracted and then the password retrieved earlier can be used to open the pdf file inside. However, the second page of the pdf file is missing.

Examining the raw contents of the pdf file reveals that the pdf formatting has been messed around.

Header

%PDF-1.4

%Óëéá

The header contains a redundant value below which can be removed.

Body

1 0 obj

<</Title (Minutes)

/Producer (Skia/PDF m124 Google Docs Renderer)>>

endobj

3 0 obj

<</ca 1

/BM /Normal>>

Endobj

There are many objects that are not arranged in the order of their index.

Xref table

xref

0 25

0000000000 65535 f

0000000015 00000 n

0000207919 00000 n

0000000098 00000 n

0000227254 00000 n

0000258051 00000 n

0000000135 00000 n

0000208138 00000 n

0000004061 00000 n

0000003690 00000 n

0000092578 00000 f

0000208381 00000 n

0000095509 00000 n

0000207554 00000 n

0000208617 00000 n

0000208686 00000 n

0000208790 00000 n

0000208923 00000 n

0000226066 00000 n

0000226307 00000 n

0000226877 00000 n

0000227398 00000 n

0000256658 00000 n

0000256894 00000 n

0000257646 00000 n

There is an extra object that has the free object label and the objects are not arranged in the order of the byte offset.

Trailer

trailer

<</Size 25

/Root 16 0 R

/Info 1 0 R>>

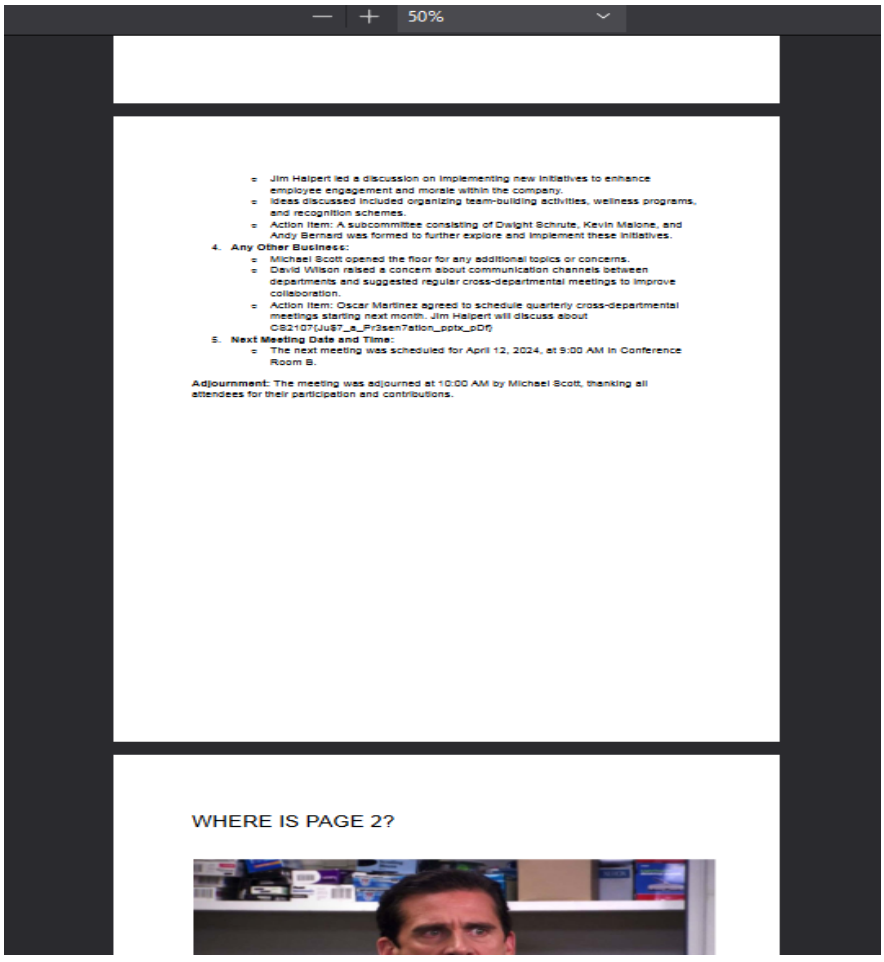
startxref

258190

%%EOF

The root data should go before the size data.

After correcting for the pdf format, the second page is recovered and the flag is observed.



Flag

CS2107{Ju$7_a_Pr3sen7ation_pptx_pDf}