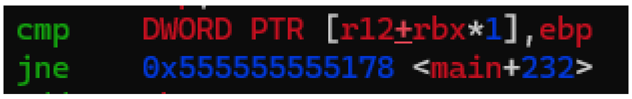
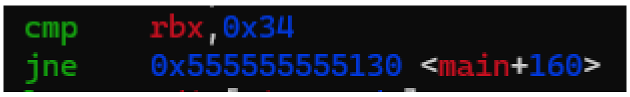
E.1 Can you GDB? (Reverse Engineering)

With the given chal-unstripped, we will use gdb to see the corresponding assembly code within the binary file. With additional hints given by the teaching team, we can see that there are two portions in which the program compares the registers:

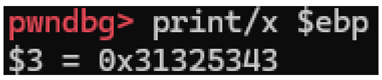


```
cmp    DWORD PTR [r12+rbx*1],ebp
jne    0x555555555178 <main+232>
```



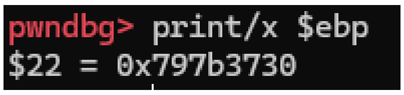
```
cmp    rbx,0x34
jne    0x555555555130 <main+160>
```

Since the second comparison function is comparing with a constant, we can focus on the first comparison function which compares the value stored in $r12 + $rbx with $ebp. This function compares the user's input with the flag, as such, we can observe the value stored in $ebp for each loop. Examples are shown below:



```
pwndbg> print/x $ebp
$3 = 0x31325343
```

Which corresponds to 12SC.



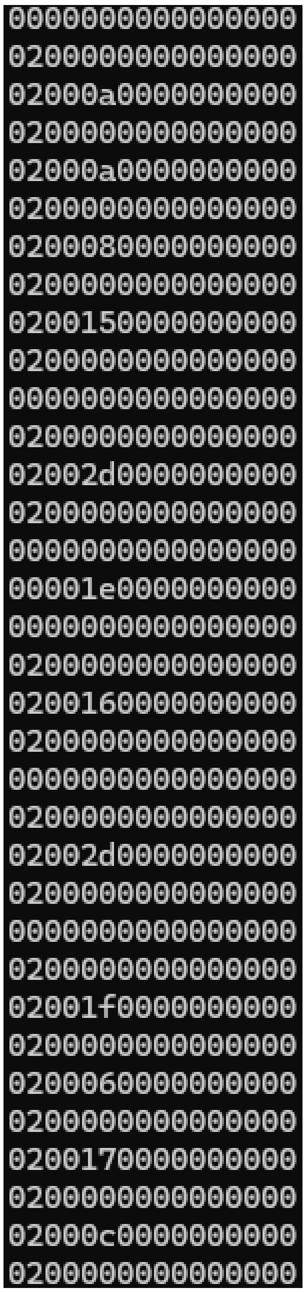
```
pwndbg> print/x $ebp
$22 = 0x797b3730
```

Which corresponds to y{70.

We can then repeatedly run the program and input the next 4 characters found in $ebp. Thus, the flag is CS2107{y4y_y0u_c4n_us3_4_d3bugger!GDB_my_best13}.

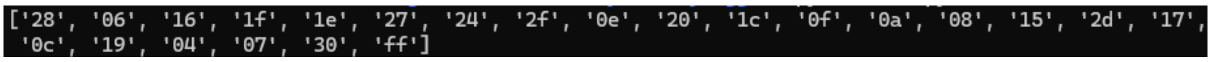
E.2 keylogger (Forensics)

By visual inspection of the given pcap file, we can see that there are two sources, one from 2.10.1 and the other from 2.17.1. Majority of the data is coming from 2.10.1, by guessing that the flag would be hidden within the lesser source, 2.17.1, we can therefore narrow down the number of rows. By further observation from wireshark, we can see that the last 16 bytes, Leftover Capture Data, changes at two distinct positions, bytes 0-1 and 4-5. Image below shows the last 16 bytes and the changes.



```
0000000000000000
0200000000000000
02000a0000000000
0200000000000000
02000a0000000000
0200000000000000
0200080000000000
0200000000000000
0200150000000000
0200000000000000
0000000000000000
0200000000000000
02002d0000000000
0200000000000000
0000000000000000
00001e0000000000
0000000000000000
0200000000000000
0200160000000000
0200000000000000
0000000000000000
0200000000000000
02002d0000000000
0200000000000000
0000000000000000
0200000000000000
02001f0000000000
0200000000000000
0200060000000000
0200000000000000
0200170000000000
0200000000000000
02000c0000000000
0200000000000000
```

From https://wiki.osdev.org/USB_Human_Interface_Devices, we can see that the first two bytes corresponds to whether the left shift being pressed. Furthermore, from https://github.com/IntergatedCircuits/hid-usage-tables/blob/master/pages/0007-keyboard-keypad.txt, we can do a simple mapping of the data in the pcap file to their corresponding keystrokes on the keyboard. As "00" and "ff" data captured by pcap is not found in the mappings, we can ignore them. Image below shows the hex values of the keys logged by the pcap.



```
['28', '06', '16', '1f', '1e', '27', '24', '2f', '0e', '20', '1c', '0f', '0a', '08', '15', '2d', '17',
 '0c', '19', '04', '07', '30', 'ff']
```

Therefore, by simply doing a mapping, we can obtain the flag, CS2107{K3YL0GGER_1S_@CTIVATED}.

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

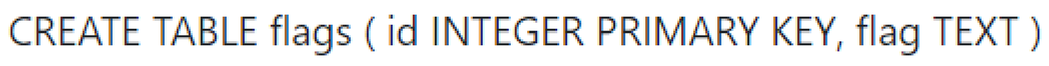
From the code given, we can see that it compares user with “sudo” and returns the flag when it is the same. We can also see that the buf and user variables each hold 16 characters. As such we can use the buffer overflow method to override the user variable by inputting any 16 random characters followed by “sudo”. Example, ‘aaaaaaaaaaaaaaaasudo’ and the flag is CS2107{ov3rfl0w_t0_unl1m1t3d_5ud0_4cc355}.

E.4 Cat Facts (Web Security)

With reference to https://portswigger.net/web-security/sql-injection, we can use the sql command sqlite_schema to obtain the tables in the database. Thus by using,

' UNION SELECT sql,sql FROM sqlite_schema limit 2 offset 1--

we can get the flag’s table schema.



CREATE TABLE flags (id INTEGER PRIMARY KEY, flag TEXT)

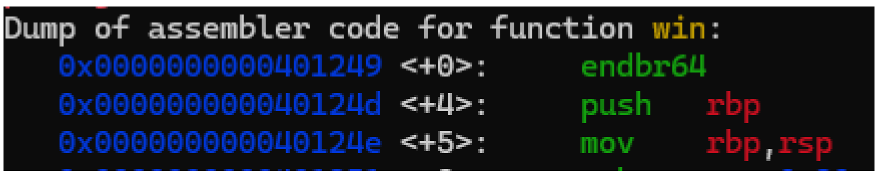
Thereafter, using the same logic, we can get the flag using,

' UNION SELECT id, flag FROM flags –

giving us the flag CS2107{SqL_iNj3cT10N_1s_qU1t3_e4sY!}.

M.2 Stacksweeper (Application Security: Binary Exploitation/Pwn)

This is another buffer overflow task; thus, we can see that the application compares the input string to a set of different variables. As such, we need to overflow the return address to win() function. We can use gdb to find the starting address of the win() function, which is 0x401249.



```
Dump of assembler code for function win:
   0x0000000000401249 <+0>:     endbr64
   0x000000000040124d <+4>:     push   rbp
   0x000000000040124e <+5>:     mov    rbp,rsp
```

We can then craft the payload to execute all the way until the supposed return address, then override the return address with 0x401249. Upon successful attack, we can then obtain the flag, CS2107{p4thf1nd3r_tr41lbl4z3r_h0w_d1d_y0u_g3t_p4st_r3d4ct3d_v4r14bl3s???}.

Reference to crafting the python script is from, https://febinj.medium.com/decode-e-cyber-ctf-2023-pwn-binary-exploitation-writeup-1-3e4c93a01dd.

M.3 Cat Breeds (Web Security)

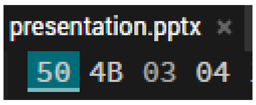
This is another sql injection attack but is however a blind one since there is no output given this time. We can inspect the code governing the webpage, and see that when the database returns a value, regardless of the input, we get "Cat breed exists!" and if there is no corresponding output from the sql query, we get "Cat breed does not exist :(". As such we can craft the query to return us a "Cat breed exists!" when we match a character of the flag. As such we can use this query in a loop to return us the flag.

' UNION SELECT id, flag FROM flags WHERE CASE WHEN SUBSTR(flag, {i}, 1) = '{char}' THEN (1=1) ELSE (1=2) END—

Full code is attached in the corresponding python file.

H.4 Presentation (Forensics)

Given the hint that might not be a powerpoint slide, we can use the magic numbers of files to determine the type of file it is supposed to be. Using online hexeditor, https://hexed.it/, we can see the magic number of the presentation.pptx to be



```
presentation.pptx ×
50 4B 03 04
```

We can then check the type of files it can be using https://en.wikipedia.org/wiki/List_of_file_signatures, and one of it was a .zip extension. By changing it to a .zip file, we can extract the files within. Among it, there is a file in ppt/media named Doc.zip. Using 7-zip to unzip this file, we get a .pdf file with a password. By scanning thru the original powerpoint slides, the password is hidden in one of the comment slides, B3stP@$$w0rd1sH3lloWorld. After entering the password, we got a missing page 2 from the pdf file. From online resources and hints, https://www.youtube.com/watch?v=Y7WV-nfGK3w and https://ctftime.org/writeup/7488, we can make use of pdftotext to see potential redacted text within the pdf. Using online tool, https://www.pdf2go.com/result#j=e45b8ab4-89cf-4f2f-983d-1f76e55a397a, and we got the flag CS2107{Ju$7_a_Pr3sen7ation_pptx_pDf}.

track for the scheduled release date.
3. Discussion on Employee Engagement Initiatives:

○

Jim Halpert led a discussion on implementing new initiatives to enhance employee engagement and morale within the company.
○ Ideas discussed included organizing team-building activities, wellness programs, and recognition schemes.
○ Action Item: A subcommittee consisting of Dwight Schrute, Kevin Malone, and Andy Bernard was formed to further explore and implement these initiatives.
4. Any Other Business:
○ Michael Scott opened the floor for any additional topics or concerns.
○ David Wilson raised a concern about communication channels between departments and suggested regular cross-departmental meetings to improve collaboration.
○ Action Item: Oscar Martinez agreed to schedule quarterly cross-departmental meetings starting next month. Jim Halpert will discuss about
CS2107{Ju$7_a_Pr3sen7ation_pptx_pDf}
5. Next Meeting Date and Time:
○ The next meeting was scheduled for April 12, 2024, at 9:00 AM in Conference Room B.
Adjournment: The meeting was adjourned at 10:00 AM by Michael Scott, thanking all attendees for their participation and contributions.

WHERE IS PAGE 2?