

12/12/2024

CIBERSEGURIDAD UTFSM

PWN 101

Introducción a la Explotación de Bugs de
Corrupción de Memoria

Pablo Aravena - litneet64





CONTENIDOS

Agenda

1

2

3

4

5

6

7

8

Promesa

¿pwn? ¿codigo
vulnerable?

ELF / Process
Memory

Call
Conventions

Tools
(Debugging y
Dissassembly)

Demo

Mas Tecnicas

Pwn en el
Mercado



LA PROMESA

- ¿Qué es Pwn?
- Buffer Overflow
 - Why's and How's
- Más Técnicas
- ¿Hobby o Trabajo?



¿PWN? ¿CÓDIGO VULNERABLE?

- Ejemplo simple
- gets

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char buf[32]; // holds user input
6     char passwd[] = "1337pass"; // hardcoded password (very secure)
7
8     printf("\n Enter the password: \n");
9     gets(buf); ← NOT USED NOWADAYS
10
11    /* password validation */
12    if (strcmp(passwd, buf) == 0) {
13        printf("successfully logged in\n");
14    } else {
15        printf("wrong pass!\n");
16    }
17
18    return 0;
19 }
```



¿PWN? ¿CÓDIGO VULNERABLE?

- # • gets

NAME
gets - get a string from standard input (DEPRECATED)

LIBRARY
Standard C library (`libc`, `-lc`)

SYNOPSIS

```
#include <stdio.h>

[[deprecated]] char *gets(char *s);
```

DESCRIPTION

Never use this function.

`gets()` reads a line from `stdin` into the buffer pointed to by `s` until either a terminating newline or `EOF`, which it replaces with a null byte ('\0'). No check for buffer overrun is performed (see BUGS below).

RETURN VALUE

`gets()` returns `s` on success, and `NULL` on error or when end of file occurs while no characters have been read. However, given the lack of buffer overrun checking, there can be no guarantees that the function will even return.



¿PWN? ¿CÓDIGO VULNERABLE?

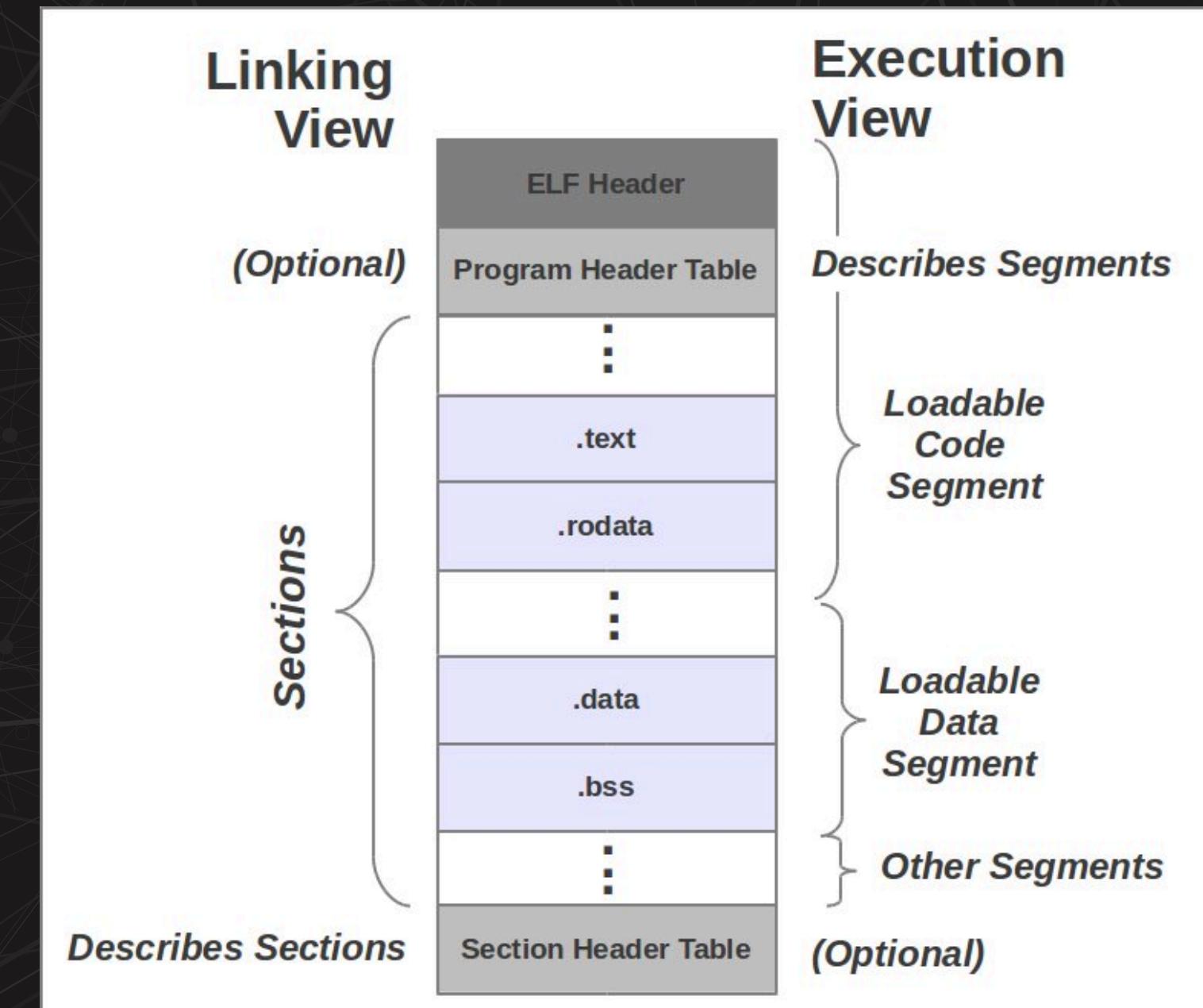
- Ejemplo un poco mas elaborado
- BOF no tan obvio

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define BUF_SIZE 64
5
6 int main() {
7     char c;
8     char buf[BUF_SIZE]; // intermediate buffer for dealing with user input
9     char userpass[16]; // user password for comparation
10    char passwd[] = "1337pass"; // hardcoded password (very secure)
11
12    printf("Enter the password: ");
13
14    for (char i = 0; i < BUF_SIZE; i++) {
15        c = getchar();
16
17        if (c == '\n') {
18            buf[i] = '\0';
19            break;
20        }
21
22        buf[i] = c; // very secure buffer write
23    }
24
25    strcpy(userpass, buf);
26
27    /* password validation */
28    if (strcmp(passwd, userpass) == 0) {
29        printf("successfully logged in\n");
30    } else {
31        printf("wrong pass!\n");
32    }
33
34    return 0;
35}
```



ELF BASICS

- Executable Linux File
- Metadata para Linker
 - Id
 - libc (GLIBC, uLibc, varios)

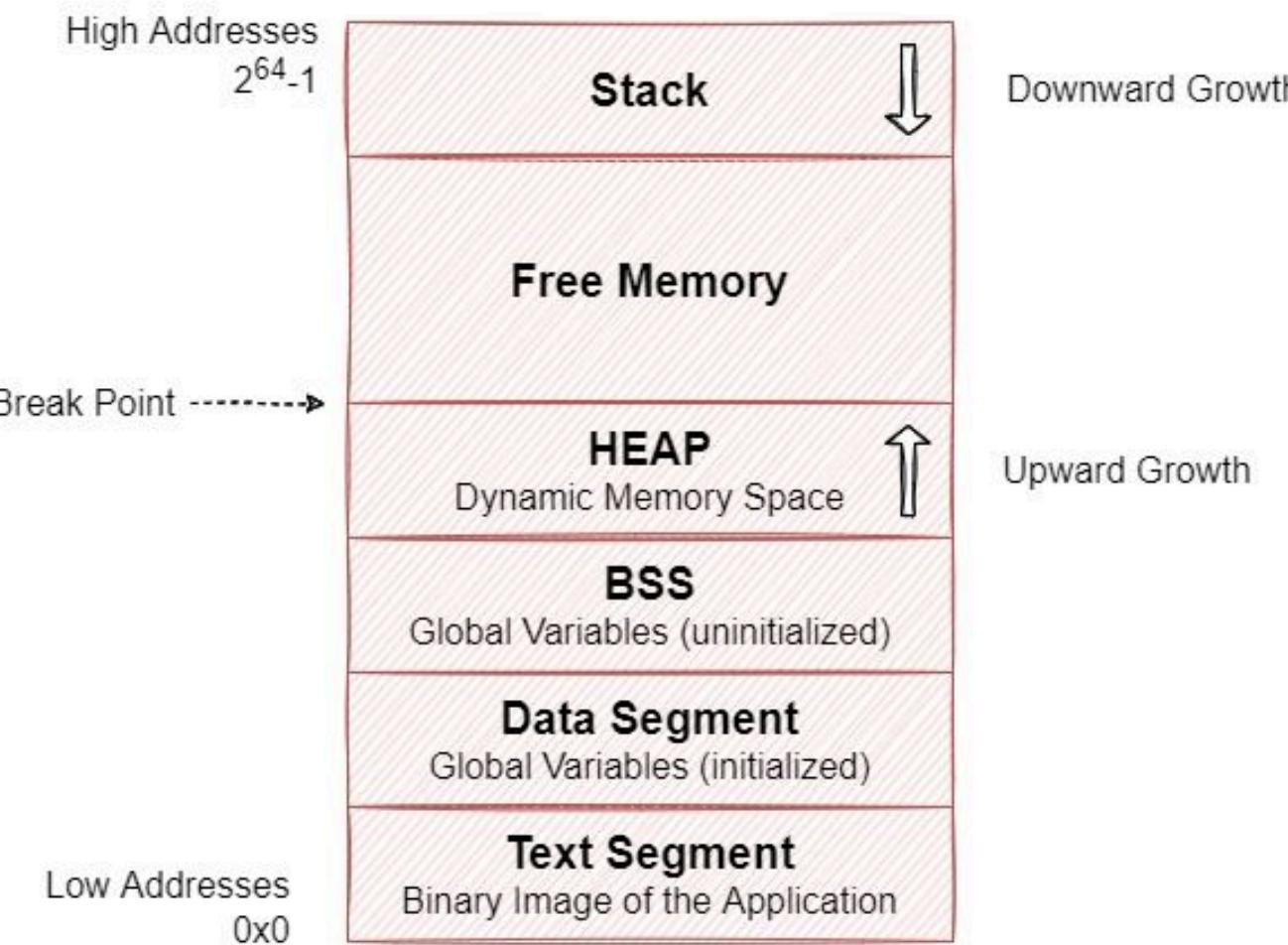




PROCESS MEMORY BASICS

- Programa cargado == Proceso

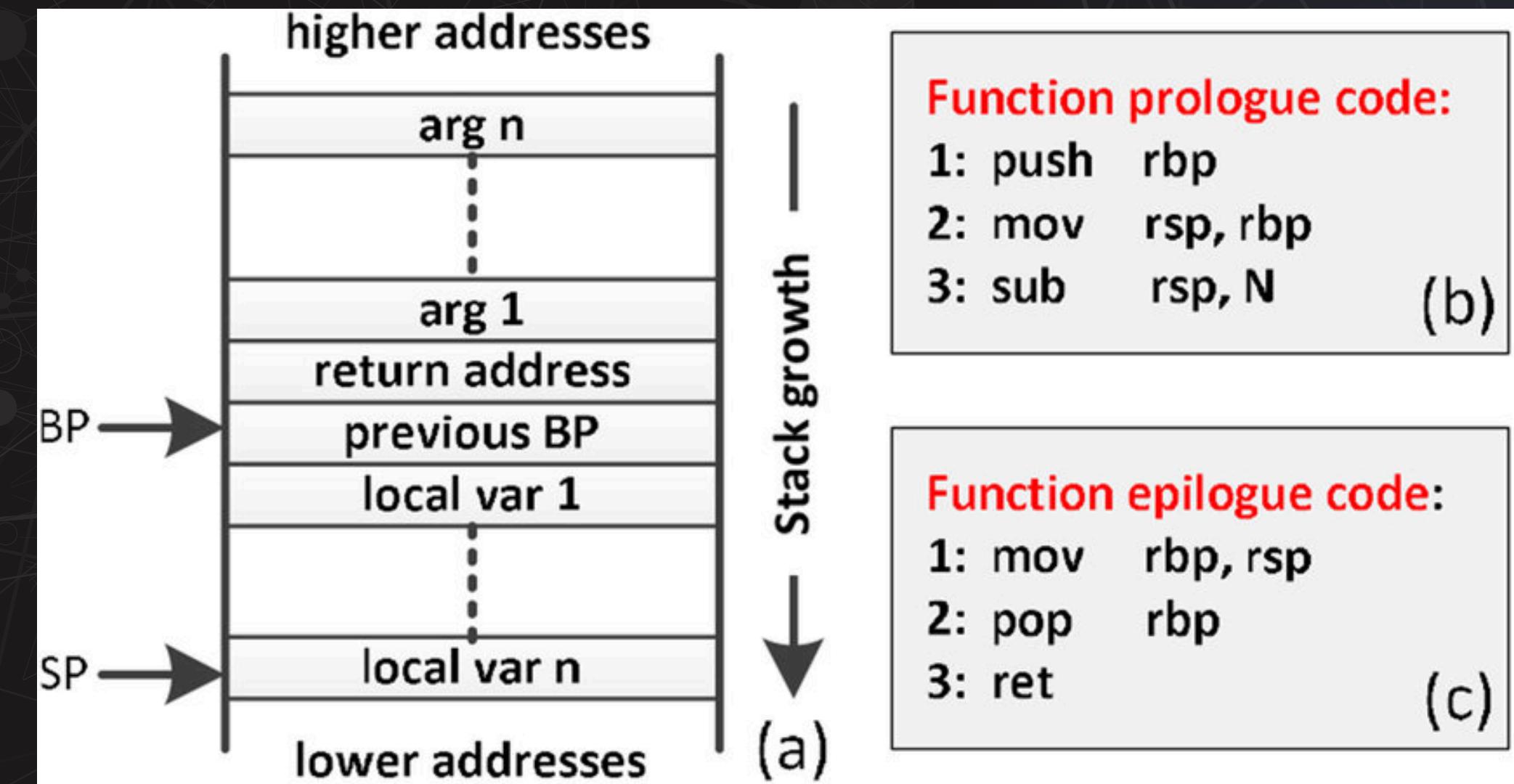
Process Memory Allocation (simplified)





PROCESS MEMORY BASICS

- Function Call
 - Stack Frame
- x86_32 args
 - Todo por stack
- x86_64
 - rdi
 - rsi
 - rcx
 - rdx
 - r8
 - r9
 - Resto por stack





¿PWN? ¿CÓDIGO VULNERABLE?

- Ejemplo simple
- fgets

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char passwd[] = "1337pass"; // hardcoded password (very secure)
6     char buf[32]; // holds user input
7
8     printf("Enter the password: \n");
9
10    fgets(buf, 64, stdin);
11
12    /* password validation */
13    if (strcmp(passwd, buf) == 0) {
14        printf("successfully logged in\n");
15    } else {
16        printf("wrong pass!\n");
17    }
18
19    return 0;
20 }
```



TOOLS - DEBUGGING

- **gdb**
- **pwntools**
 - \$ pip install pwntools
- **pwndbg**
 - \$ git clone https://github.com/pwndbg/pwndbg
 - \$ cd pwndbg
 - \$./setup.sh
- **seer (frontend de gdb)**
 - <https://github.com/epasveer/seer>

```
pwndbg> start
Temporary breakpoint 1 at 0x115d

Temporary breakpoint 1, 0x00005555555515d in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
*RAX 0x5555555555159 (main) ← push rbp
*RBX 0x7fffffff918 → 0x7fffffffdd01 ← '/home/litneet64/HTB/meetup/2024/dec/12/examples/simple_bof'
*RCX 0x555555557dd8 (_do_global_dtors_aux_fini_array_entry) → 0x55555555110 (_do_global_dtors_aux) ← endbr64
*RDX 0x7fffffff928 → 0x7fffffffdd3c ← 'SHELL=/bin/bash'
*RDI 0x1
*RSI 0x7fffffff918 → 0x7fffffffdd01 ← '/home/litneet64/HTB/meetup/2024/dec/12/examples/simple_bof'
R8 0x0
*R9 0x7ffff7fcbf40 (_dl_fini) ← push rbp
*R10 0x7fffffff540 ← 0x800000
*R11 0x206
R12 0x0
*R13 0x7fffffff928 → 0x7fffffffdd3c ← 'SHELL=/bin/bash'
*R14 0x7ffff7fd000 (_rtld_global) → 0x7ffff7ffe2e0 → 0x555555554000 ← 0x10102464c457f
*R15 0x555555557dd8 (_do_global_dtors_aux_fini_array_entry) → 0x55555555110 (_do_global_dtors_aux) ← endbr64
*RBP 0x7fffffff800 ← 0x1
*RSP 0x7fffffff800 ← 0x1
*RIP 0x5555555515d (main+4) ← sub rsp, 0x30
[ DISASM / x86-64 / set emulate on ]
▶ 0x5555555515d <main+4>    sub   rsp, 0x30
0x55555555161 <main+8>    movabs rax, 0x7373617037333331
0x5555555516b <main+18>   mov    qword ptr [rbp - 9], rax
0x5555555516f <main+22>   mov    byte ptr [rbp - 1], 0
0x55555555173 <main+26>   lea    rax, [rip + 0xe8a]
0x5555555517a <main+33>   mov    rdi, rax
0x5555555517d <main+36>   call   puts@plt             <puts@plt>
0x55555555182 <main+41>   mov    rdx, qword ptr [rip + 0x2ea7] <stdin@GLIBC_2.2.5>
0x55555555189 <main+48>   lea    rax, [rbp - 0x30]
0x5555555518d <main+52>   mov    esi, 0x40
0x55555555192 <main+57>   mov    rdi, rax
[ STACK ]
00:0000| rbp rsp 0x7fffffff800 ← 0x1
01:0008+008 0x7fffffff808 → 0x7ffff7dcce68 (_libc_start_call_main+120) ← mov edi, eax
02:0010+010 0x7fffffff810 → 0x7fffffff900 → 0x7fffffff908 ← 0x38 /* '8' */
03:0018+018 0x7fffffff818 → 0x55555555159 (main) ← push rbp
04:0020+020 0x7fffffff820 ← 0x155554040
05:0028+028 0x7fffffff828 → 0x7ffff7fcbf40 (_dl_fini) ← push rbp
06:0030+030 0x7fffffff830 → 0x7fffffff918 → 0x7fffffffdd01 ← '/home/litneet64/HTB/meetup/2024/dec/12/examples/simple_bof'
07:0038+038 0x7fffffff838 ← 0xb616bc040b05a2
[ BACKTRACE ]
▶ 0 0x5555555515d main+4
1 0x7ffff7dcce68 _libc_start_call_main+120
2 0x7ffff7dcce25 _libc_start_main+133
3 0x55555555091 _start+33
```



TOOLS - DISASSEMBLY

- IDAFree / IDAPro
- Ghidra
- BinaryNinja
- ImHex

The screenshot shows the IDA Pro interface with several windows open:

- IDA View-RIP**: Shows assembly code for the `main` function. The code reads a password from standard input, compares it with a hardcoded value, and prints "successfully logged in" or "wrong pass!".
- Pseudocode-A**: Shows the pseudocode corresponding to the assembly code.
- Stack of main**: Shows the stack contents of the current thread.
- Local Types**: Shows declarations for registers (RAX, RBX, RCX, RSI, RDI) and modules (libc.so.6, ld-linux-x86-64.so.2).
- Hex View-1**: Shows the raw memory dump of the program's memory space.
- Stack view**: Shows the stack history and current state.
- Output**: Shows the terminal output of the debugger, including library loading and process start information.



HANDS ON!

- Look at my screen!



MAS TECNICAS DE PWN

- “Classic” Pwn
 - Executable Shellcode (in RWX stack or mem map)
 - Format string vulnerabilities
 - Canary bypasses through different write-primitives
 - Jump to RSP / “Stack Pivot”
 - Return-Oriented-Programming / ROP-Chaining
 - Leaking Libc address through GOT (ret2libc)
 - Seccomp Filters
 - **one_gadget**
 - **malloc_hooks** (older Libc versions)



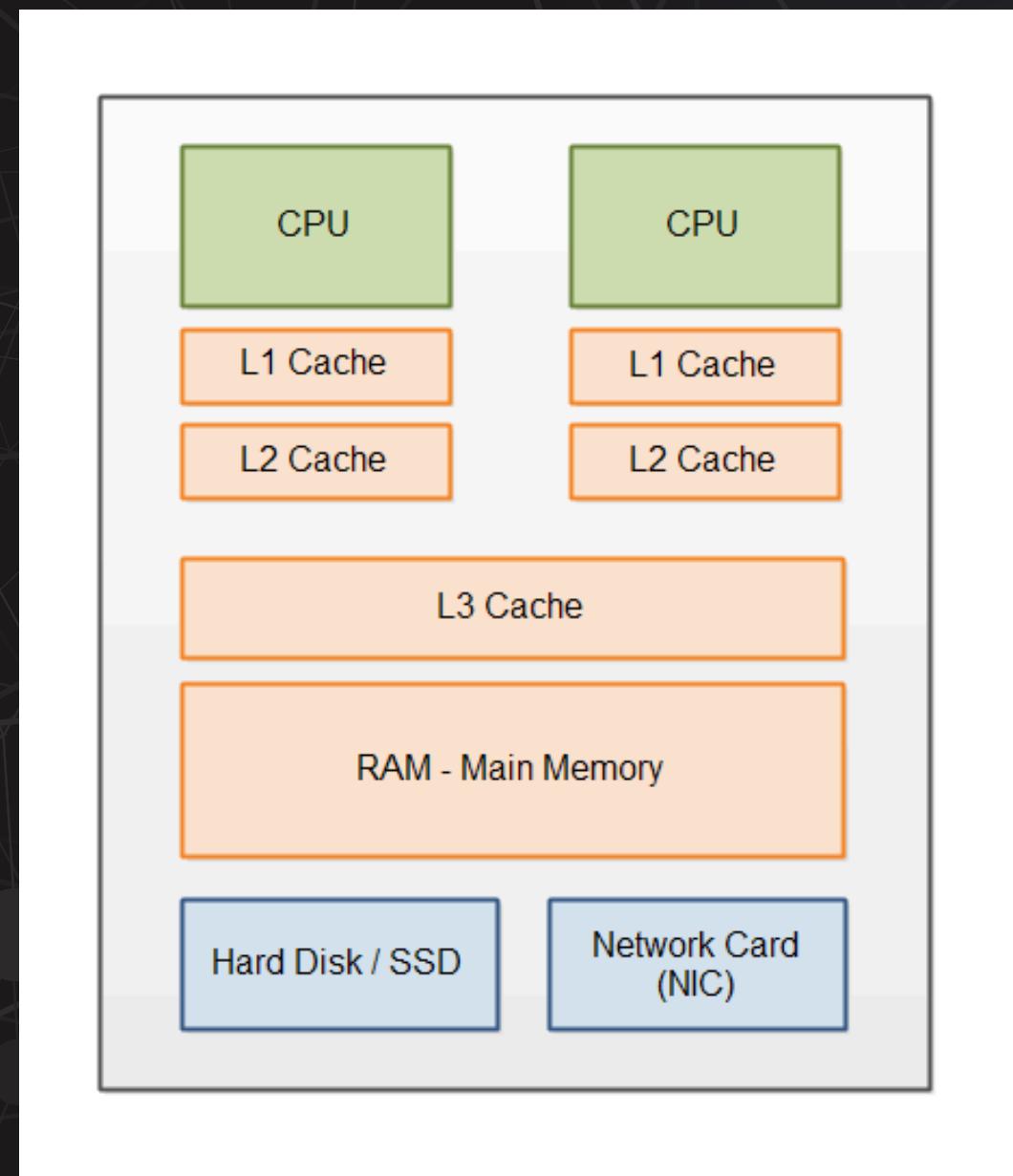
MAS TECNICAS DE PWN

- Heap-Based and “House of ...”
 - **tcache-duping (double free, UAF)**
 - Leak main arena address with unsorted-bin
 - Read/Write from/to arbitrary places through file descriptor structs (File-Oriented-Programming)
 - “off-by-one” vulns
 - Heap Fengshui



MAS TECNICAS DE PWN

- Microarquitectura de Procesador
 - Cache Timing attack
 - Speculative Execution (Spectre)
 - Address Dereferencing / Prefetching





MAS TECNICAS DE PWN

- Otros
 - Race Conditions (multithreading)
 - Kernel Drivers
 - Kernel vulns (syscalls, slab allocator)
 - Otras arquitecturas
 - ARM (mobile, IoT/embedded)
 - Xtensa (ESP32)
 - MIPS (IoT/embedded)
 - AVR (Arduino)
 - RISC-V (The Future!)
 - Windows
 - userland
 - kernel (where the money is)



“LOW-LEVEL” EN EL MERCADO

The screenshot shows a blog post from the website gynvael.coldwind//vx.log. The header features a white cat logo and the site's name. Below the header are two buttons: "Available for Consulting and Projects" and "Upcoming Talks". A dark banner at the top of the main content area displays the title "2024-08-03: FAQ: The tragedy of low-level exploitation" and the word "faq". The main content starts with a "Obligatory FAQ note" about frequently asked questions. It then addresses a question about low-level exploitation in cybersecurity, noting its rarity and often being a small part of another role. A disclaimer follows, stating the goal is to provide information rather than discourage. The post concludes with a "Background" section and a note about learning hacking/cybersecurity.

Obligatory FAQ note: Sometimes I get asked questions, e.g. on my Discord/IRC, via e-mail, or during my livestreams. And sometimes I get asked the same question repeatedly. To save myself some time and be able to give the same answer instead of conflicting ones, I decided to write up selected answers in separate blog posts. Please remember that these answers aren't necessarily authoritative - they are limited by my experience, my knowledge, and my opinions on things. Do look in the comment section as well - a lot of smart people read my blog and might have a different, and likely better, answer to the same question. If you disagree or just have something to add - by all means, please do comment.

Q: I love low-level exploitation and exploit development! How can I make this my whole career?

A: So to not bury the lead, the problem is that **low-level exploitation is rarely needed in cybersecurity**, and jobs where one works mostly on low-level exploitation are few and far between. Furthermore, these jobs are even more rare if one wants to stay away from the gray area of hacking and away from the black market. It's more common for low-level exploitation to be a small occasional part of another role.

DISCLAIMER: The goal of this post is not to discourage anyone from pursuing a career in low-level hacking, nor do I think that it isn't an important area of cybersecurity. Rather than that, the goal is to give folks enough information to think things through and plan their approach instead of walking into this blindly.

Let's start with a bit of background...

Background

While the start of the path of learning hacking / cybersecurity changes from time to time, sooner or later it leads folks to try low-level security. This commonly starts with a bit of reverse code engineering – the one on binary / assembly level – paired up with getting to know how to use a debugger and learning the CPU architecture, and leads down the path to learning low-level vulnerability classes and



“LOW-LEVEL” EN EL MERCADO INTERNACIONAL



The image shows a screenshot of a blog post from gynvael.com. The post is titled "2024-08-03: FAQ: The tragedy of low-level exploitation". It features a cat logo and a purple header. The main content discusses the rarity of low-level exploitation jobs and the challenges of being a low-level exploit developer. A QR code is overlaid on the image.

Available for Consulting and Training

2024-08-03: FAQ: The tragedy of low-level exploitation

Obligatory FAQ note: Sometimes I get asked questions, e.g. on my Discourse forum, that are either very specific or very general and often conflict with each other. Instead of answering them all individually, I decided to write up selected answers in separate posts like this one. You can also look in the comment section as well - a lot of smart people read my blog and comment on it.

Q: I love low-level exploitation and exploit development!
A: So to not bury the lead, the problem is that low-level exploitation is a small field. Furthermore, these jobs are even more rare if one wants to make it their primary or occasional part of another role.

DISCLAIMER: The goal of this post is not to discourage anyone from learning low-level exploitation. Instead, it's to give folks enough information to think things through and plan accordingly.

Let's start with a bit of background...

Background

While the start of the path of learning hacking / cybersecurity changes from time to time, sooner or later it leads folks to try low-level security. This commonly starts with a bit of reverse code engineering – the one on binary / assembly level – paired up with getting to know how to use a debugger and learning the CPU architecture, and leads down the path to learning low-level vulnerability classes and



MEETUP HTB #1

**¡MUCHAS GRACIAS
POR SU ATENCIÓN!**

¿Preguntas?