Shellcodes sous Linux dans les processeurs de 32 bits x86

بسم الله الرحمان الرحيم و الصلاة و السلام على سيدنا محمد و آله و صحبه

paper par:
opt!x hacker(aidi youssef)
optix@9.cn

sommaire:

0x01:présentation:

un shellcode est un élément très important dans tous les exploits ,pendant l'attaque il est injecté à un autre programme opérationnel et lui fait éxécuter les opérations demandées et pour ecrire un shellcode on'est besoin d'un peu de connaisances en C et en assembleur et une experience dans LINUX.

0x02:les registres dans les processeurs 32 bits x86:

les registres sont des petites cellules de mémoire qui se trouvent dans le processeur est qui ont le rôle de stocker des valeurs numériques ils sont utilisées par le processeur pendant l'exécution de chaque programme ou tache .
Ils se divisent en deux catégories:
_rergistres de données(EAX,EBX,ECX,EDX)
explication::::

Nom de registre	Son role
EAX	L'éxecution des opérations arithmétiques et les opérations entrée/ sortie, et il contient la défintion de l'appel systeme à exécuter(sys call) et la valeur retournée par l'appel systeme.
EBX	Il stocke le premier argument de l'appel systeme.
ECX	Il stocke le deuxieme argument de l'appel systeme.
EDX	Utilisé pour stocker les adresses des variables, stocke le troisième argument de l'appel système.

_registres d'addresse(ESI,EDI,ESP,EBP,EIP)
Je vais donner leurs explications dans un prochain cour

0x03:appel systeme(Linux System Call):

Il faut premièrement savoir que chaque appel systeme(write , close , kill ...) est fournit par le noyau KERNEL.
Pour connaître la version du kernel on tape la commande suivante:
uname -r ou uname -a

```
File  Edit  View  Terminal  Tabs  Help
/ You're working under a slight handicap. \
\ You happen to be human.                  /
 ------------------------------------------
  \
   \
      ___
     {~._.~}
      ( Y )
     ()~*~()
     (_)-(_)
aidi@aidi-laptop ~ $ uname -r
2.6.27-7-generic
aidi@aidi-laptop ~ $
```

Chaque appel systeme a un nombre précis mais comment peut on défénir ce nombre . c'est très facile:

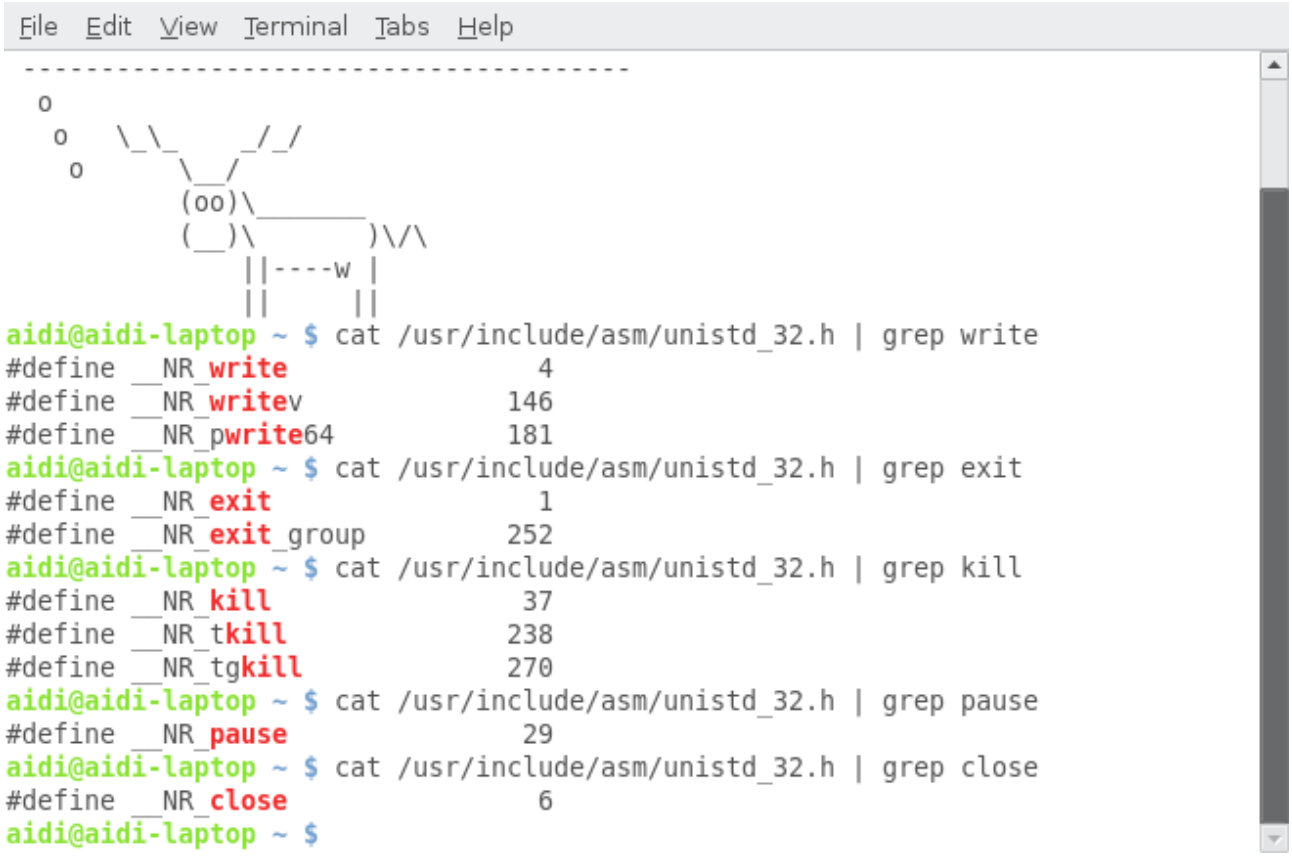
```
File  Edit  View  Terminal  Tabs  Help
/ Fame is a vapor; popularity an           \
| accident; the only earthly certainty is |
| oblivion.                                |
|                                          |
\ -- Mark Twain                            /
 ------------------------------------------
        \   ,__,
         \  (oo)____
            (__)    )\
               ||--|| *
aidi@aidi-laptop ~ $ cd /usr/include/asm
aidi@aidi-laptop /usr/include/asm $ dir
a.out.h       ipcbuf.h     poll.h              shmbuf.h         types.h
auxvec.h      ist.h        posix_types_32.h    sigcontext32.h   ucontext.h
boot.h        kvm.h        posix_types_64.h    sigcontext.h     unistd_32.h
bootparam.h   ldt.h        posix_types.h       siginfo.h        unistd_64.h
byteorder.h   mce.h        prctl.h             signal.h         unistd.h
debugreg.h    mman.h       processor-flags.h   socket.h         vm86.h
e820.h        msgbuf.h     ptrace-abi.h        sockios.h        vsyscall.h
errno.h       msr.h        ptrace.h            statfs.h
fcntl.h       msr-index.h  resource.h          stat.h
ioctl.h       mtrr.h       sembuf.h            termbits.h
ioctls.h      param.h      setup.h             termios.h
aidi@aidi-laptop /usr/include/asm $
```

```
File  Edit  View  Terminal  Tabs  Help
aidi@aidi-laptop /usr/include/asm $ cat unistd_32.h
#ifndef _ASM_I386_UNISTD_H_
#define _ASM_I386_UNISTD_H_

/*
 * This file contains the system call numbers.
 */

#define __NR_restart_syscall      0
#define __NR_exit                 1
#define __NR_fork                 2
#define __NR_read                 3
#define __NR_write                4
#define __NR_open                 5
#define __NR_close                6
#define __NR_waitpid              7
#define __NR_creat                8
#define __NR_link                 9
#define __NR_unlink              10
#define __NR_execve              11
#define __NR_chdir               12
#define __NR_time                13
#define __NR_mknod               14
#define __NR_chmod               15
#define __NR_lchown              16
#define __NR_break               17
#define __NR_oldstat             18
#define __NR_lseek               19
#define __NR_getpid              20
#define __NR_mount               21
#define __NR_umount              22
#define __NR_setuid              23
#define __NR_getuid              24
#define __NR_stime               25
#define __NR_ptrace              26
#define __NR_alarm               27
#define __NR_oldfstat            28
#define __NR_pause               29
#define __NR_utime               30
#define __NR_stty                31
#define __NR_gtty                32
```

mais si on veut avoir le numéro d'un appel système précis on tape la commande suivante :



c'est très facile

0x04:Les instructions les plus importantes de l'assembleur:

mov:Copie le contenu d'un fragment de la mémoire vers un autre ou une valeur .par ex: *MOV EBP,ESP* **ou** *mov eax,4* **(valeur syscall write)**
push :**Copie sur la pile le contenu du fragment sélectionné de la mémoire:***PUSH EDI****(source)***
pop:Transfère le contenu de la pile vers le fragment voulu de la mémoire: pop EDI **(source)**
xor:Calcule la différence symétrique des fragments sélectionnés de la mémoire:(xor ecx,acx ou xor ecx,ebx)

remarque: xor a,b si a=b => xor a,b=0
xor ecx,ecx=0 (pour éviter les NULL BYTES)
jmp:c'est un saut vers une addresse définie et il permet de changer l'addresse EIP par une autre choisie : jmp +adresse
call:c'est un appel comme l'nstruction jmp mais celle-ci empile l'adresse de l'instruction suivante avant de changer EIP:call adr
int:elle donne un signale au noyau linux(kernel)en appelant l'interruption portant le numéro:int 0x80
et il'ya aussi des instructions qui effectuent des opérations arithmétigues et logiques(recherche in google)

0x05:tools:

1)un assembleur logiciel moi je préfère NASM
vous pouvez le telecharger:
http://nasm.sourceforge.net/
lorsque vous le téléchargez vous allez trouver un livre PDF d'aide:

To assemble a file, you issue a command of the form

nasm −f <format> <filename> [−o <output>]

For example,

nasm −f elf myfile.asm

will assemble myfile.asm into an ELF object file myfile.o. And

nasm −f bin myfile.asm −o myfile.com

will assemble myfile.asm into a raw binary file myfile.com.

To produce a listing file, with the hex codes output from NASM displayed on the left of the original sources,

use the −l option to give a listing file name, for example:

nasm −f coff myfile.asm −l myfile.lst

To get further usage instructions from NASM, try typing

nasm −h

As −hf, this will also list the available output file formats, and what they are.

If you use Linux but aren’t sure whether your system is a.out or ELF, type

file nasm

2)*un logiciel* Disassembler
comme:objdump
rechercher le dans google ou sourceforge
3)connaisances de C et Assembly et LINUX (ligne de commandes)
4)un vers de thé pour se concenter

0x06:notre premier shellcode

notre premier shellcode va etre basé sur l'appel systeme __write et __exit
nous devrons chercher le numéro de write dans syscalls

/*code1
aidi@aidi-laptop ~ $ cat /usr/include/asm/unistd_32.h | grep write

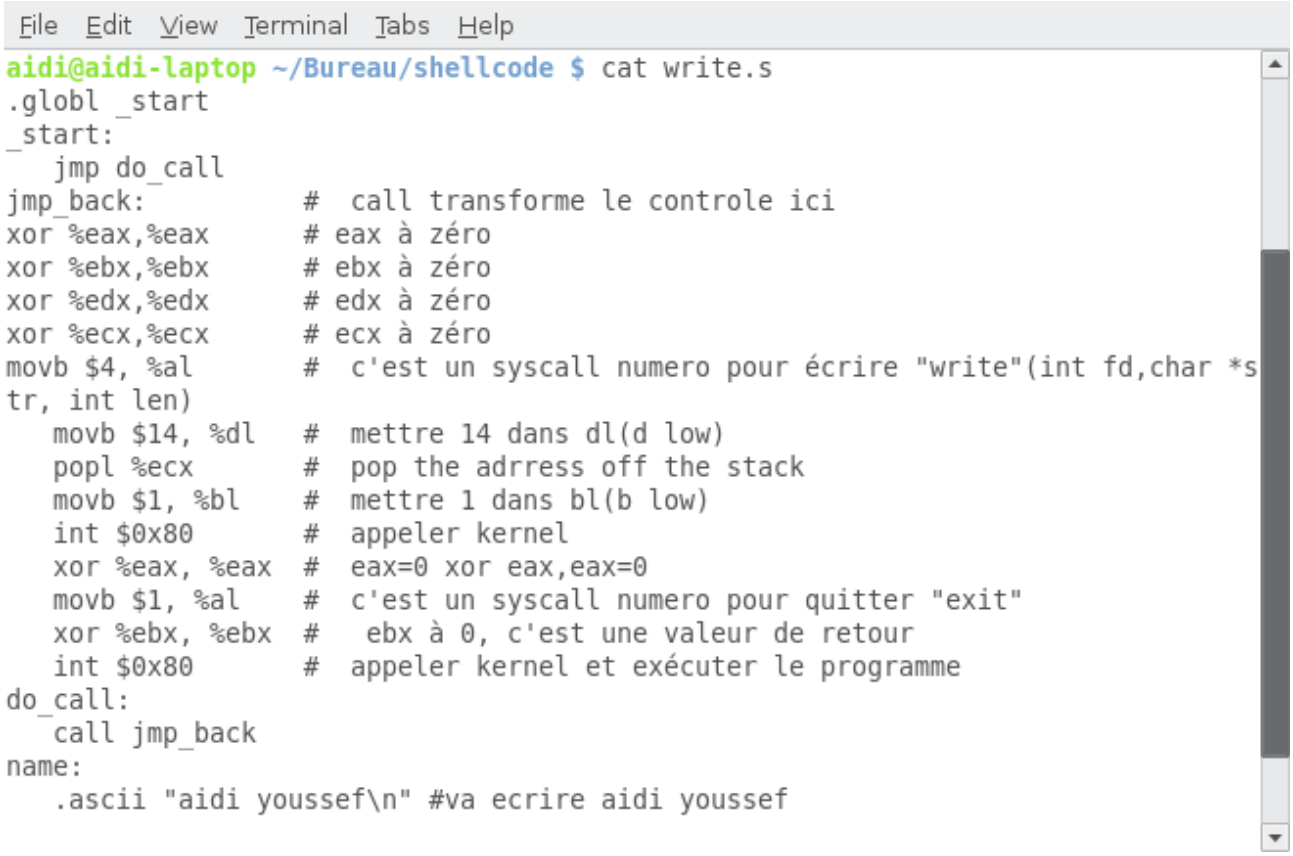
#define __NR_write 4

#define __NR_writev 146

/*code2
#define __NR_exit 1

chaque registre va avoir son propre travail:
on va les remettre à zero pour eviter les zéros dans notre shellcode par l'instruction xor comme on'a vu (xor a,a=0)

on va commencer avec un tres simple code qui va ecrire mon nom'aidi youssef' . Voilà notre code:



```
File  Edit  View  Terminal  Tabs  Help
aidi@aidi-laptop ~/Bureau/shellcode $ cat write.s
.globl _start
_start:
   jmp do_call
jmp_back:          #  call transforme le controle ici
xor %eax,%eax      # eax à zéro
xor %ebx,%ebx      # ebx à zéro
xor %edx,%edx      # edx à zéro
xor %ecx,%ecx      # ecx à zéro
movb $4, %al       #  c'est un syscall numero pour écrire "write"(int fd,char *s
tr, int len)
   movb $14, %dl   #  mettre 14 dans dl(d low)
   popl %ecx       #  pop the adrress off the stack
   movb $1, %bl    #  mettre 1 dans bl(b low)
   int $0x80       #  appeler kernel
   xor %eax, %eax  #  eax=0 xor eax,eax=0
   movb $1, %al    #  c'est un syscall numero pour quitter "exit"
   xor %ebx, %ebx  #   ebx à 0, c'est une valeur de retour
   int $0x80       #  appeler kernel et exécuter le programme
do_call:
   call jmp_back
name:
   .ascii "aidi youssef\n" #va ecrire aidi youssef
```

Assemble & link aprés Disassemble:

on va utiliser pour ce travail les outils que j'ai cité dans la paragraphe tools :)))))))))))))))))))))))))))))))))))):

je vais vous expliquer:

pour assembler:

```
as nameoffile.s -o nomdefichiervoulu.o
```

pour linker:

```
ld nomdefichiervoulu.o name
```

pour disassembler on doit telecharger l'outils objdump et taper la commande:

```
objdump -d fichierfinale
```

pour exécuter ce dernier :

```
./lenomdufichier
```

pour avoir de l'aide: par exemple:

```
objdump --help
```

```
File  Edit  View  Terminal  Tabs  Help
aidi@aidi-laptop ~/Bureau/shellcode $ as write.s -o write.o
aidi@aidi-laptop ~/Bureau/shellcode $ ld write.o -o write
aidi@aidi-laptop ~/Bureau/shellcode $ objdump -d write

write:     file format elf32-i386


Disassembly of section .text:

08048054 <_start>:
 8048054:       eb 19                   jmp    804806f <do_call>

08048056 <jmp_back>:
 8048056:       31 c0                   xor    %eax,%eax
 8048058:       31 db                   xor    %ebx,%ebx
 804805a:       31 d2                   xor    %edx,%edx
 804805c:       31 c9                   xor    %ecx,%ecx
 804805e:       b0 04                   mov    $0x4,%al
 8048060:       b2 0e                   mov    $0xe,%dl
 8048062:       59                      pop    %ecx
 8048063:       b3 01                   mov    $0x1,%bl
 8048065:       cd 80                   int    $0x80
 8048067:       31 c0                   xor    %eax,%eax
 8048069:       b0 01                   mov    $0x1,%al
 804806b:       31 db                   xor    %ebx,%ebx
 804806d:       cd 80                   int    $0x80

0804806f <do_call>:
 804806f:       e8 e2 ff ff ff          call   8048056 <jmp_back>

08048074 <name>:
 8048074:       61                      popa
 8048075:       69 64 69 20 79 6f 75    imul   $0x73756f79,0x20(%ecx,%ebp,2),%es
p
 804807c:       73
 804807d:       73 65                   jae    80480e4 <name+0x70>
 804807f:       66                      data16
 8048080:       0a                      .byte 0xa
aidi@aidi-laptop ~/Bureau/shellcode $
```

comme vous voyez c'est très bien on'a pas de zéro
maintenant nous pouvons écrire notre shellcode avec C
'a i d i y o u s s e f \n' en ASCII
61 69 64 69 20 79 6f 75 73 73 65 66 0a
espace=20 en ASCII

```
File  Edit  View  Terminal  Tabs  Help
     (_)-(_)
aidi@aidi-laptop ~/Bureau/shellcode $ ./write
aidi youssef
aidi@aidi-laptop ~/Bureau/shellcode $
```

et maintenat aprés on'a compilé notre shellcode on doit l'écrire avec C et l'éxecuter:
pour ce travail on doit ajouter \x aux valeurs hex

```
File  Edit  View  Terminal  Tabs  Help
 ______________________________________
< You have taken yourself too seriously. >
 --------------------------------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *
aidi@aidi-laptop ~/Bureau/optix@9.cn $ cat write.c
#include "stdio.h"

int main(int argc, char *argv[])
{
char shellcode[] =
         "\xeb\x19\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x04"
         "\xb2\x0e\x59\xb3\x01\xcd\x80\x31\xc0\xb0\x01\x31"
         "\xdb\xcd\x80\xe8\xe2\xff\xff\xff\x61\x69\x64\x69\x20\x79\x6f\x75\x73\x73\x65\x66\x0a";

printf("Length: %d\n",strlen(shellcode));
        (*(void(*)()) shellcode)();

        return 0;
}
aidi@aidi-laptop ~/Bureau/optix@9.cn $ this is our shellcode let's try it
```

je vais compiler le fichier write.c et l'éxecuter :

```
File  Edit  View  Terminal  Tabs  Help
 ______________________________________
( Something's rotten in the state of )
( Denmark.                            )
(                                     )
( -- Shakespeare                      )
 --------------------------------------
  o
   o

     ___
    {~._.~}
     ( Y )
    ()~*~()
    (_)-(_)
aidi@aidi-laptop ~/Bureau/optix@9.cn $ gcc write.c -o write
write.c: In function ‘main’:
write.c:10: warning: incompatible implicit declaration of built-in function ‘str
len’
aidi@aidi-laptop ~/Bureau/optix@9.cn $ ./write
Length: 45
aidi youssef
aidi@aidi-laptop ~/Bureau/optix@9.cn $
```

:) notre premier shellcode fonctionne parfaitement avec une taille de 45 bytes

0x07:Greetz

je veux remercier:

Jonathan Salwan

Michał Piotrowski

his0k4

stack

and all muslims crackers

ce paper est ecrit sous la distrubition linux mint 6 pour les débutants.