Shell-Storm.org

Création des shellcodes sous architecture Linux x86 32 bits.

Jonathan Salwan #
http://www.shell-storm.org #
04/07/2009 #

Bon! on va tout d'abord commencer par les bases de l'asm x86.
Il faut savoir que chaque syscall (appel système) est fourni par le noyau du système d'exploitation.

Exemple d'appel système fréquemment utilisé:
open, write, read, close, chmod, chown ...

Chaque appel système est défini par un nombre bien précis.
Exemple: open => 5
write => 4
chmod => 15

Comment faire pour connaître ce numero ?
Démonstration:

```
root@laptop:/root# cat /usr/include/asm/unistd_32.h | grep write
#define __NR_write              4
#define __NR_writev             146
#define __NR_pwrite64           181
root@laptop:/root# cat /usr/include/asm/unistd_32.h | grep chmod
#define __NR_chmod               15
#define __NR_fchmod              94
#define __NR_fchmodat            306
```

A quoi va servir notre numero ?

Lors de la programmation en asm, on va appeler notre syscall via son numero défini .

Premier petit programme en asm pour débuter doucement.

Réferons nous au site suivant:
http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html

On va programmer en premier lieu un code qui va tout simplement effectuer une pause.
sur le site que je vous est "linker" en regardant le syscall pause (sys_pause) on voit bien qu'on n'a pas besoin de remplir les registres (ebx,ecx,edx...) du coup notre code va être très court!

(Attention dans ce tutorial tous les codes asm seront compilés avec NASM version 2.03.01)

<code>

```
------------------------------
root@laptop:/home/jonathan/all/prog/asm/pause# cat main.s
section .text
        global _main
_main:
        mov eax,29
        int 0x80
------------------------------
```

Tout d'abord on attribue le syscall 29 au registre eax (l'appel système de pause)
L'instruction int 0x80 nous permet d'exécuter le tout

compilons et exécutons notre programme.

```
root@laptop:/home/jonathan/all/prog/asm/pause# nasm -f elf main.s
root@laptop:/home/jonathan/all/prog/asm/pause# ld -o main main.o
ld: warning: cannot find entry symbol _start; defaulting to
0000000008048060
root@laptop:/home/jonathan/all/prog/asm/pause# ./main
^C
root@laptop:/home/jonathan/all/prog/asm/pause#
```

Notre programme fonctionne parfaitement =)(Ctrl+C permet de quitter la pause ;))

Maintenant nous allons étudier l'appel système _exit.
Regardons quel est le numero du syscall exit

sys_exit   = 1
%ebx   = int

ici _exit aura besoin du registre ebx pour y contenir un entier
Nous allons donc essayer de faire l'équivalent de exit(0);

<code>

```
------------------
root@laptop:/home/jonathan/all/prog/asm/exit# cat main.s
section .text
       global _main
_main:
       mov eax,1 ; <====== on place 1 dans eax soit le syscall _exit
       xor ebx,ebx <====== on fait un ou exclusif sur
                          ebx pour avoir 0
                          (je vous expliquerais plus bas)
       int 0x80
------------------
```

compilons et exécutons

```
root@laptop:/home/jonathan/all/prog/asm/exit# nasm -f elf main.s
root@laptop:/home/jonathan/all/prog/asm/exit# ld -o main main.o
ld: warning: cannot find entry symbol _start; defaulting to
00000000008048060
root@laptop:/home/jonathan/all/prog/asm/exit# ./main
root@laptop:/home/jonathan/all/prog/asm/exit#
```

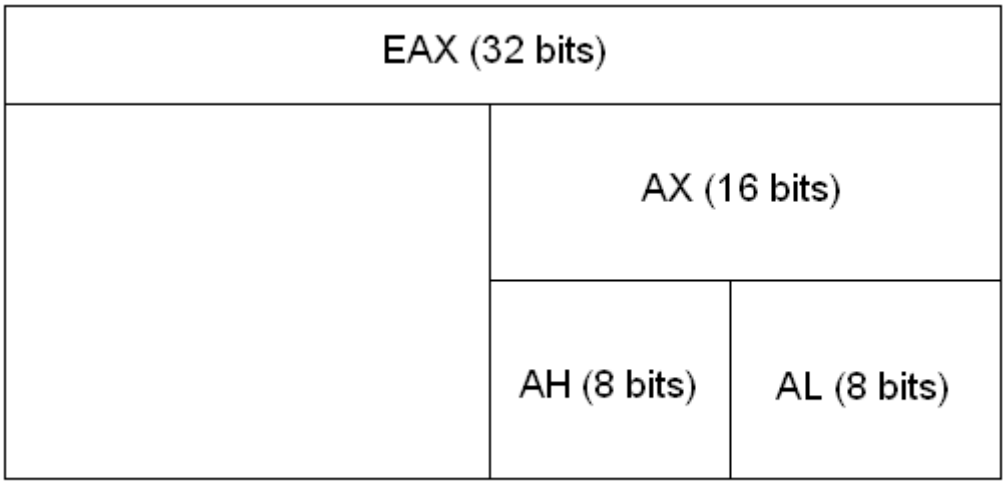
Voila tout a bien fonctionné. Maintenant utilisons le programme objdump, objdump va nous permettre d'avoir les valeurs des instructions asm en hexadecimal ce qui va nous servir pour coder notre futur shellcode.

```
root@laptop:/home/jonathan/all/prog/asm/exit# objdump -d main
main:      file format elf32-i386
Disassembly of section .text:
08048060 <_main>:
 8048060:          b8 01 00 00 00      mov $0x1,%eax
 8048065:          31 db               xor %ebx,%ebx
 8048067:          cd 80               int $0x80
```

Tout à droite nous retrouvons nos instructions en asm, au milieu nous avons leur équivalence en hexadecimal et à gauche nous avons l'adresse des instructions dans notre programme.

Pour effectuer un shellcode nous devons avoir aucun 00 sinon notre shellcode marquerait la fin de la chaîne de caractère représentée par celui-ci.

Info registre eax:



Nous devons donc modifier notre code asm, et remplacer le registre eax par al pour que 8 bits soit simplement utilisé au lieu de 32.

```
<code>
----------------------
root@laptop:/home/jonathan/all/prog/asm/exit# cat main.s
section .text
       global _main
_main:
       mov al,1    <== ici le changement al au lieu de eax
       xor ebx,ebx <== ou exclusif pour eviter les zéros
       int 0x80
----------------------
```

Vérifions le changement.

```
root@laptop:/home/jonathan/all/prog/asm/exit# objdump -d main
main:    file format elf32-i386
Disassembly of section .text:
08048060 <_main>:
 8048060:       b0 01          mov $0x1,%al
 8048062:       31 db          xor %ebx,%ebx
 8048064:       cd 80          int $0x80
```

et voilà notre programme ne contient désormais aucun zéro.

Etudions la fonction write.

Par contre cette fois nous allons compiler notre programme avec as et non nasm par conséquence la syntaxe de notre programme va légèrement changer...

Info write:
```
-----------
sys_write  = 4
%ebx       = unsigned int
%ecx       = const char *
%edx       = size
```

Nous allons écrire jona than :p, tout d'abord nous devons couper le mot tous les 32 bits donc 4 caractères, ensuite nous devons écrire tout à l'envers car dans la pile on empile ;) vous devez vous dire que c'est galère mais c'est pas si compliquer que ça regardez le schéma suivant.

```
jonathan => | jona | than => | than => | naht => | 6e 61 68 74
                          => | jona => | anoj => | 61 6e 6f 6a
```

<code>
```
----------------------
root@laptop:/home/jonathan/all/prog/asm/write/2# cat main.s
xor %eax,%eax          ; eax à zéro
xor %ebx,%ebx          ; ebx à zéro
xor %edx,%edx          ; edx à zéro
xor %ecx,%ecx          ; ecx à zéro
                       ; on fait tout ça pour éviter
                       ; l'erreur de segment après compilation
movb $0x4,%al          ; syscall _write
movb $0x1,%bl          ; unisgned int
pushl $0x0a            ; line feed (retour Ã la ligne)
push $0x6e616874       ; naht
push $0x616e6f6a       ; anoj
movl %esp,%ecx         ; on place esp dans ecx
movb $0x9,%dl          ; size (ici la taille du mot jonathan + \n
                       ; soit 8 + 1 = 9)
int $0x80

movb $0x1,%al          ; syscall _exit
xor %ebx,%ebx          ; ou exclusif pour éviter les zéros
int $0x80
------------------------
```

Compilons et exécutons tout ça.

```
root@laptop:/home/jonathan/all/prog/asm/write/2# as -o main.o main.s
root@laptop:/home/jonathan/all/prog/asm/write/2# ld -o main main.o
ld: warning: cannot find entry symbol _start; defaulting to
0000000008048054
root@laptop:/home/jonathan/all/prog/asm/write/2# ./main
jonathan
root@laptop:/home/jonathan/all/prog/asm/write/2#
```

Maintenant on va voir ce que ça donne pour créer un shellcode en C.
Utilisons objdump pour commencer .

```
root@laptop:/home/jonathan/all/prog/asm/write/2# objdump -d main
main:   file format elf32-i386
Disassembly of section .text:
08048054 <.text>:
 8048054:       31 c0           xor   %eax,%eax
 8048056:       31 db           xor   %ebx,%ebx
 8048058:       31 d2           xor   %edx,%edx
 804805a:       31 c9           xor   %ecx,%ecx
 804805c:       b0 04           mov   $0x4,%al
 804805e:       b3 01           mov   $0x1,%bl
 8048060:       6a 0a           push  $0xa
 8048062:       68 74 68 61 6e  push  $0x6e616874
 8048067:       68 6a 6f 6e 61  push  $0x616e6f6a
 804806c:       89 e1           mov   %esp,%ecx
 804806e:       b2 09           mov   $0x9,%dl
 8048070:       cd 80           int   $0x80
 8048072:       b0 01           mov   $0x1,%al
 8048074:       31 db           xor   %ebx,%ebx
 8048076:       cd 80           int   $0x80
```

Cool nous n'avons pas de zéro =)
Voyons ce que cela donne en C.

```
<code>:
-----------------------------------
root@laptop:/home/oxosyscall/all/prog/asm/write/2/c# cat main.c

#include "stdio.h"
int main()
{
     char shellcode[] =   "\x31\xc0"
                          "\x31\xdb"
                          "\x31\xd2"
                          "\x31\xc9"
                          "\xb0\x04"
                          "\xb3\x01"
                          "\x6a\x0a"
                          "\x68\x74\x68\x61\x6e"
                          "\x68\x6a\x6f\x6e\x61"
                          "\x89\xe1"
                          "\xb2\x09"
                          "\xcd\x80"
```

```
                        "\xb0\x01"
                        "\x31\xdb"
                        "\xcd\x80";

     printf("Length: %d\n",strlen(shellcode)); //pour afficher la taille
                                               //du shellcode

     (*(void(*)()) shellcode)();
     //asm("jmp %esp"); (autre façon d'exécuter notre shellcode)

return 0;
}
-----------------------------------
```

Aller hOp! compilons notre first shellcode =)

```
root@laptop:/home/jonathan/all/prog/asm/write/2/c# gcc -o main main.c
main.c: In function "main":
main.c:22: attention : incompatible implicit declaration of built-in
function "strlen"
root@laptop:/home/jonathan/all/prog/asm/write/2/c# ./main
Length: 36
jonathan
```

Super!!! notre premier shellcode à fonctionné nickel sans aucun problème =) avec une taille de 36 bytes.

Si vous voulez d'autres exemples de shellcodes ou documents, vous pouvez vous connecter au site **http://www.shell-storm.org**, vous pouvez également nous envoyer vos sources à l'adresse **submit[AT]shell-storm.org**

Jonathan Salwan #
http://www.shell-storm.org #