# Création d'un binaire multiplateforme

Depuis quelques temps, nous voyons apparaître ça et là quelques virus multiplateformes. Nous citerons par exemple Winux, virus qui a pour cible les OS Linux et Windows...
Bien entendu, les virus ne sont pas les seuls binaires (car on parle ici de binaires, et non de scripts) à pouvoir profiter de cette avantageuse programmation. Bien qu'ils soient peu répandus, il peut s'avérer intéressant de proposer de tels binaires dans le cas par exemple d'un programme d'installation ou autre. En effet, les utilisateurs n'ont alors plus à se préoccuper de savoir à quel OS est destiné le programme, le binaire détectant automatiquement son environnement d'installation. Ceci n'est qu'un exemple parmi tant d'autres et les possibilités d'utilisation sont multiples. Les virus multiplateformes semblent toutefois être un des " débouchés " les plus évidents.
Nous verrons ainsi au cours de l'article comment créer un tel binaire. Toute l'astuce repose sur une manipulation des en-têtes du binaire et sur les aspects de relocation du segment de code.

Cette étude s'appuie sur un travail préalable de Kuno Woudt (warp-tmt@dds.nl) et Rafal Skoczylas (nils@secprog.org).

# I. Linux

Il existe plusieurs formats d'exécutables sous Linux. Nous nous intéresserons plus particulièrement au format ELF, le plus répandu sous cet OS.
Nous ne reviendrons pas en détail sur ce format, toute la documentation étant largement accessible par Internet. Rappelons juste que tout binaire ELF présente un en-tête qui renseigne sur le fonctionnement du binaire (format, type de binaire, point d'entrée, taille des sections et autres).
Nous commençons par créer un programme en assembleur tout simple qui affichera un message type Hello Strange World... pour ne pas faillir à la tradition. Nous implémenterons nous-mêmes le header ELF, de manière à pouvoir le changer par la suite (pas de linker).
Voici donc le corps tout simple d'un programme de ce type :

```
 ; Hello Strange World    4 Linux.
;
; Propriétés : Binaire exécutable sur Linux.
; Assemblage : nasm -f bin hello_linux.asm -o hello_linux

BITS 32

%define ELF_RELOC 0x08048000    ; Adresse de relocation traditionnelle pour un
ELF
%define __NR_write      4
%define __NR_read       3

; 1ere étape : l'en-tête ELF

ehdr                                    ; ELF header
               db       0x7F            ; EI_MAG0
               db       'E'             ; EI_MAG1
               db       'L'             ; EI_MAG2
               db       'F'             ; EI_MAG3
               db       1               ; EI_CLASS: 32-bit objects
               db       1               ; EI_DATA: ELFDATA2LSB
               db       1               ; EI_VERSION: EV_CURRENT
               db       0               ; EI_PAD

       times 8 db       0               ; EI_NIDENT
```

```
              dw       2                       ; e_type
              dw       3                       ; e_machine
              dd       1                       ; e_version
              dd       _start_unx + ELF_RELOC  ; e_entry
              dd       phdr                    ; e_phoff
              dd       0                       ; e_shoff
              dd       0                       ; e_flags (unused on intel)
              dw       ehdrsize                ; e_ehsize
              dw       phdrsize                ; e_phentsize

; Ensuite, nous créons un en-tête de programme...
; Il convient de noter que les 8 premiers octets de phdr cohabitent avec la
; fin de l'en-tête ELF, les valeurs étant identiques...

phdr                                          ; Elf32_Phdr
              dw         1                    ; e_phnum       ; p_type
              dw         0                    ; e_shentsize
              dw         0                    ; e_shnum       ; p_offset
              dw         0                    ; e_shstrndx

ehdrsize      equ        $ - ehdr             ; Fin de l'entête ELF

              dd         ELF_RELOC            ; p_vaddr
              dd         ELF_RELOC            ; p_paddr (ignored)
              dd         filesize             ; p_filesz

              dd         filesize             ; p_memsz
              dd         5                    ; p_flags
              dd         0x1000               ; p_align

phdrsize       equ     $ - phdr               ; Fin de l'en-tête de programme

_start_unx:
        mov     ecx,    hello_unx_msg + ELF_RELOC
        mov     edx,    hello_unx_msg_length
        call    print
        call    exit

;-----------------------------------------------
; Définitions de quelques procédures générales |
;-----------------------------------------------

; Print (ecx = string pointer, edx = length)
; ------------------------------------------
print:
        xor     eax, eax
        mov     ebx, eax
        mov     al, __NR_write
        int     80h
        ret

; Exit function
; -------------

exit:
        xor     eax, eax
        mov     ebx, eax
        inc     al
        int     80h
        ret
```

```
;----------
; Données |
;----------

hello_unx_msg         db        "Hello Strange Linux World...", 10, 0
hello_unx_msg_length equ        $ - hello_unx_msg

filesize        equ     $ - $$
```

Après assemblage, le programme fonctionne de la manière escomptée et nous affiche le message voulu, avant de quitter.

Intéressons-nous maintenant aux programmes destinés aux environnements tels que Windows et DOS.

# II. DOS

Dans le cas précis d'un binaire DOS, nous codons un .COM. Les fichiers .COM n'ont aucun header et exécutent tout simplement les instructions depuis les premiers octets du fichier. Nous programmons, toujours en assembleur, un mini binaire qui affiche un texte sous DOS pour commencer.

```
;
; Hello Strange World    4 DOS.
;
; Propriétés : Binaire exécutable sur DOS
; Assemblage : nasm -f bin hello_dos.asm -o hello_dos.com

org 0

%define COM_RELOC 0x100

   mov        dx, hello_dos_msg + COM_RELOC
   mov        ah, 0x09
   int        0x21

   mov        ax, 0x4C00
   int        0x21

hello_dos_msg           db      "Hello strange dos world...", 13, 10, "$"
```

Il est possible de faire coexister deux formats dans un même exécutable en exploitant les propriétés de ces formats. Voyons cela...

# III. Faire coexister ces formats...

À la différence d'un .COM, un binaire ELF commence toujours par un header qui détermine les paramètres propres au fichier :

```
 yanisto@kaya:~/Multi-Platform$ readelf -h hello_linux
En-tête ELF:
  Magique:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Classe:                            ELF32
  Données:                           complément à 2, système little endian
```

```
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ...
```

Ce header comporte certaines données indispensables au bon fonctionnement du binaire ELF qu'il nous faut impérativement conserver dans notre binaire généraliste. D'autres données pourront en revanche être écrasées (version ABI notamment, fin du champ e_ident de l'en-tête ELF cf. elf(5)).

À ce point, nous pourrions nous dire que cela risque d'empêcher la coexistence avec un format .COM, qui, lui, exécute les instructions en commençant par le début de l'exécutable, c'est-à-dire notre en-tête ELF. Il n'en est rien, car les octets du header se traduisent en instructions tout à fait valides :

```
yanisto@kaya:~/Multi-Platform$ ndisasm -a hello_linux|head   (16 bits)

00000000  7F45              jg 0x47
00000002  4C                dec sp
00000003  46                inc si
00000004  0101              add [bx+di],ax
00000006  0100              add [bx+si],ax
00000008  0000              add [bx+si],al
0000000A  0000              add [bx+si],al
0000000C  0000              add [bx+si],al
0000000E  0000              add [bx+si],al
00000010  0200              add al,[bx+si]
```

La première instruction est un saut conditionnel, pour lequel nous ne contrôlons pas le résultat. Il faut donc prendre en compte les 2 possibilités :

- Le saut est exécuté et il faut qu'à 0x47 octets de là se retrouvent nos instructions DOS valides pour qu'elles soient immédiatement exécutées.
- Le saut n'est pas exécuté et les octets suivants vont être lus. Nous allons donc devoir compenser (annuler) toutes les instructions induites par l'existence du header. Par exemple, la séquence dec sp; inc si sera annulée par dec si; inc sp. Nous profitons du fait que tout l'en-tête n'est pas indispensable (version ABI...) au bon fonctionnement pour insérer un saut dans l'en-tête (qui ne sera pas exécuté dans le ELF, mais le sera dans le .COM.

Traduisons cela en assembleur :

```
 ; Hello Strange World    4 Linux \& DOS.
;
; Propriétés : Binaire exécutable sur Linux \& DOS
; Assemblage : nasm -f bin hello_multi.asm -o hello_multi

BITS 32

%define ELF_RELOC 0x08048000    ; Adresse de relocation traditionnelle pour un
ELF
%define COM_RELOC 0x100         ; Adresse de relocation traditionnelle pour un
COM

%define __NR_write      4
%define __NR_read       3

; 1ere étape : l'en-tête ELF
```

```
ehdr                                    ; ELF header
              db       0x7F             ; EI_MAG0
              db       'E'              ; EI_MAG1
              db       'L'              ; EI_MAG2
              db       'F'              ; EI_MAG3
              db       1                ; EI_CLASS: 32-bit objects
              db       1                ; EI_DATA: ELFDATA2LSB
              db       1                ; EI_VERSION: EV_CURRENT
              db       0                ; EI_PAD

               jmp short _start_dos     ; saut "Tobogan" pour DOS qui
                                        ; compense les instructions
précédentes

       times 6 db       0               ; EI_NIDENT (8-2 = 6...)

              dw       2                       ; e_type
              dw       3                       ; e_machine
              dd       1                       ; e_version
              dd       _start_unx + ELF_RELOC  ; e_entry
              dd       phdr                    ; e_phoff
              dd       0                       ; e_shoff
              dd       0                       ; e_flags (unused on intel)
              dw       ehdrsize                ; e_ehsize
              dw       phdrsize                ; e_phentsize

phdr                                      ; Elf32_Phdr
              dw         1                ; e_phnum       ; p_type
              dw         0                ; e_shentsize
              dw         0                ; e_shnum       ; p_offset
              dw         0                ; e_shstrndx

ehdrsize      equ        $ - ehdr         ; Fin de l'entête ELF

              dd         ELF_RELOC        ; p_vaddr
              dd         ELF_RELOC        ; p_paddr (ignored)
              dd         filesize         ; p_filesz
              dd         filesize         ; p_memsz
              dd         5                ; p_flags
              dd         0x1000           ; p_align

phdrsize       equ     $ - phdr           ; Fin de l'en-tête de programme

; ---------------( Code DOS )-------------

BITS 16

_start_dos:
.compensation
;00000000  7F45              jg 0x47
;00000002  4C                dec sp
;00000003  46                inc si
;00000004  0101              add [bx+di],ax
;00000006  0100              add [bx+si],ax

   sub [bx+si],ax
   sub [bx+di],ax
   dec si
   inc sp

.affichage
   mov        dx, hello_dos_msg + COM_RELOC
```

```
   mov        ah, 0x09
   int        0x21

.exit
   mov        ax, 0x4C00
   int        0x21

; ---------------( Code Unix )-------------

BITS 32

_start_unx:
.affichage
        mov     ecx,    hello_unx_msg + ELF_RELOC
        mov     edx,    hello_unx_msg_length
        xor     eax, eax
        mov     ebx, eax
        mov     al, __NR_write
        int     80h

.exit
        xor     eax, eax
        mov     ebx, eax
        inc     al
        int     80h

;----------
; Données |
;----------

hello_unx_msg           db      "Hello strange linux World...", 10, 0
hello_unx_msg_length    equ     $ - hello_unx_msg

hello_dos_msg           db      "Hello strange dos World...", 13, 10, "$"

filesize                equ     $ - $$
Testons tout cela :
[ Sous Linux ]

yanisto@kaya:~/Multi-Platform$ ./hello_multi.com
Hello strange linux World...

[ Sous Ligne de commande Windows ]

C:\MultiP> hello_multi.com
Hello strange dos World...
```

# IV. Diversification OS

Une fois que cette coexistence des formats ELF et COM est rendue possible, nous pouvons nous pencher sur la diversification de l'exécutable vers d'autres OS utilisant ces mêmes formats.
Il est en effet possible d'identifier les versions d'OS (DOS, Win 3.x, Win9x, WinNT, Linux, BSD, OS/2...). Pour ce faire, il convient d'effectuer quelques tests.

# 1/ DOS/Windows

Une fois que le programme se situe dans la partie propre à DOS/Windows (c'est-à-dire _start_dos), il nous est possible de distinguer entre les différentes versions de Windows grâce à l'insertion dans la partie DOS/Windows du code ci-après :

```
mov ax, 0x1600
int 0x2F
```

Ensuite, il suffit d'examiner le registre AX pour déterminer l'OS plus précisément :

- Si AL = 0x80, alors l'OS n'est pas du type Windows.
- Si AL = 1 ou AL = 0xFF, alors l'OS est un Windows 2.x.
- Si AL = 0 ou AL = 0x16, alors l'OS est un Windows NT/XP.
- Sinon, si AL comporte une valeur différente, AL désigne le numéro de version (major) et ah, le minor de la version (ex : 3.1).

Le code complet pour distinguer les différents cas est donc :

```
 mov ax, 0x1600
int 0x2F

cmp        al, 0x80
je  short  _start_dos           ; OS = DOS

cmp        al, 0
je  short  _start_dosxp         ; OS = Windows NT/XP

cmp        al, 3
jle short  _start_w3x           ; OS = Windows 3.x/2.x...

jmp short  _start_w9x           ; OS = Windows 9x
```

Il nous suffirait donc à présent d'insérer de nouvelles fonctions plus spécifiques à chacun de ces cas dans le code, avant le bloc de données par exemple, pour les traiter séparément.

# 2/ Unix-Flavor

Enfin, dans la partie de code dédiée aux Unix-like (**_start_unx**), pour distinguer Linux et les *BSD, il nous suffit de vérifier le contenu des registres **fs** et **gs**. Si ces deux registres sont à 0, alors, nous sommes sur un Linux, sinon c'est BSD.

```
mov        eax, fs
cmp        eax, 0
jnz        _start_bsd

mov        eax, gs
cmp        eax, 0
jz         _start_lnx
```

# V. Conclusion

Il est bien entendu qu'une partie du code doit être réécrit, les appels système n'étant pas les mêmes selon les différents environnements. Toutefois, pour un programme relativement simple, il pourrait être intéressant de dresser un tableau de pointeurs sur nos principales fonctions selon l'environnement (du type '**print(LINUX)(stdout, "Hello")**' / '**print(DOS)(stdout, "Hello dos")**'...). Un registre contenant l'index tout au long du programme ferait en somme office d'une sorte de wrapper.
Il n'est donc pas impossible de faire un programme (binaire) assemblé et non scripté tournant sous différents OS sans besoin de modification. Après, les limites de votre programme sont celles de votre imagination...