Travaux Pratiques n°5

Architecture des ordinateurs.

—IR 1—

Ce sujet de travaux pratiques est facultatif. Vous pouvez envoyez votre rapport à votre enseignant:

Arnaud Carayol : `arnaud.carayol@univ-mlv.fr`
Guillaume Blin : `guillaume.blin@univ-mlv.fr`

Vous donnerez à votre mail le sujet suivant:

`[Archi IR1][TP5] Nom1--Nom2`

Le but de ce sujet est d'illustrer comment exploiter un *buffer overflow*. Le programme avec un *buffer overflow* est `mdp.c`.

Question 1 Que fait le programme `mdp.c` ? Où se trouve le *buffer overflow* ?

Question 2 En utilisant GDB, désassemblez la fonction `mdp` et en déduire la manière dont la pile est utilisée par cette fonction.

Question 3 Expliquez le comportement observé en exécutant:

```
./mdp  "AAAABBBBCCCCDDDDEEEE"
```

Vous pouvez utiliser la commande suivante dans GDB pour ajouter l'argument:

```
run AAAABBBBCCCCDDDDEEE
```

Nous allons exploiter la possibilité de changer l'adresse de retour pour faire exécuter au programme un code de notre choix. L'idée est de stocker dans une variable d'environnement le code à exécuter. Dans notre cas, nous allons exécuter un code qui exécute `/bin/sh`.

Il y a deux problèmes dans cette approche:

1. Le code que l'on veut exécuter ne doit contenir aucun octet de valeur 0. Par exemple, l'instruction `mov eax,0` qui va être assemblée en `B800000000` ne peut pas être utilisée. On utilisera l'instruction `xor eax,eax` qui est assemblée en `31C0`.
2. Il nous faut déterminer l'adresse de la variable d'environnement.

1. Dans notre cas, nous voulons que notre code exécute `/bin/sh`. Pour cela, nous utilisons l'appel système `execve` comme dans le TP 2.

Le code assembleur se trouve dans `code.asm`. Ce code se modifie lui-même : il ne peut donc pas être exécuté directement. En effet, le système interdit d'écrire dans le segment de code.

Le programme `code.c` utilise une petite astuce pour charger le code dans la pile. Le tableau `shellcode` contient le code assemblé. Le code de la fonction `toto` remplace l'adresse de retour de la fonction par l'adresse de `shellcode`.

Question 4 Utilisez GDB pour étudier le comportement du code.

```
gdb code
set disassembly-flavor intel
disassemble 0x8049540
break shellcode
run
```

2. Nous allons mettre le code dans une variable d'environnement appelée `TOTO`. Le programme `affcode.c` affiche le code.

```
EXPORT TOTO=`./affcode`
```

Question 5 Le programme `look.asm` affiche l'adresse de la variable d'environnement `TOTO` en mémoire. Quel argmument faut-il donner au programme pour qu'il exécute notre code ?