Travaux Pratiques Systèmes d'Exploitation
Processus (Mémoire)

2014/2015

1 variables initialisées, variables non-initialisées et variables automatiques

1.1 Etape 1

Consultez le programme variables.c et remplissez le tableau suivant pour indiquer le segment qui va contenir les différentes variables de ce programme.

Variables	.text	données initialisées	données non initialisées	pile (stack)	tas (heap)
tableauGlobal					
tableau2					
f0_var0					
f0_var1					
f1_var0					
f1_var1					
main_v0					
main_v1					
main_v2					

1.2 Etape 2

Compilez le programme "variables.c" et notez la taille de l'exécutable avec la commande *ls -al*

Remplacez maintenant la ligne :

```
char tableauGlobal[65536] ;
```

par:

```
char tableauGlobal[65536] = {1};
```

Recompilez le programme et notez à nouveau sa taille.

- Que remarquez-vous?
- Donnez des explications.

```c
#include <stdio.h>
#include <stdlib.h>

char tableauGlobal[65536] ;
int tableau2[] = { 2, 3, 5, 7 };

static int f0(int f0_var0)
{
int f0_var1;
f0_var1 = f0_var0 * f0_var0;
return f0_var1;
}

static void f1(int f1_v0)
{
    printf("Le carre de %d est %d\n", f1_v0, f0(f1_v0));
    if (f1_v0 < 1000)
        {
            int f1_v1;
            f1_v1 = f1_v0 * f1_v0 * f1_v0;
            printf("Le cube de %d est %d\n", f1_v0, f1_v1);
        }
}

int main(int argc, char *argv[])
{
    static int main_v0 = 9973;
    static char main_v1[10240000];
    char *main_v2;
    main_v2 = malloc(1024);
    f1(main_v0);
    exit(0);
}
```

Figure 1: Programme variables.c

2 Gestion de la pile

2.1 Stack overflow (dépassement de la limite de la pile)

Le programme présenté dans Figure 2 permet de calculer une suite récursive en utilisant deux fonctions u et v. Ces deux fonctions sont "en théorie" équivalentes.

- Compilez et exécutez le programme.
- Que remarquez-vous ?
- Expliquez pourquoi les comportements de ces fonctions sont différents?

2.2 Du code dans la pile !

Figure 3 présente un programme C assez curieux. En effet, ce programme va définir une fonction mystérieuse en donnant directement son code binaire.

- Compilez le programme avec la commande gcc classique et essayez de l'exécuter.
- Que remarquez-vous ?
- Compilez maintenant le programme avec la commande gcc et l'option -z execstack
- Que remarquez-vous ?
- Que fait la fonction mystérieuse ?
- Que pensez-vous des programmes/librairies qui permettent l'exécution du code à partir de la pile ?

```
#include <stdio.h>
#include <stdlib.h>
#define taillemax 500000
long u(long n){
  char buffer[taillemax];long r;
  switch (n){
    case 0:
    case 1:
    case 2:
    case 3:
      r =1;
      break;
    default:
      r = u(n-1) + u(n-2) - u(n-3) + u(n-4) ;
      buffer[n%taillemax]=r;
  }
  return r;
}
long v(long n){
  int buffer[taillemax];long r;
  if (n<4){return(1);}
  else{
    long v_n3 = 1;long v_n2 = 1;long v_n1 = 1;
    long v_n0 = 1;
    long i ;
    for( i =3 ; i < n;i++){
      r = v_n3 + v_n2 - v_n1 + v_n0;
      buffer[n%taillemax]=r;
      v_n0 = v_n1;v_n1 = v_n2;
      v_n2 = v_n3;v_n3 = r;
    }
    return r;
  }
}
int main(int argc, char *argv[]){
  int n ; unsigned long i ;
  for( n = 1 , i = 1; n < 20 ; n++){
      i = i*2; printf( "v %ld = %lu\n", i, v(i));
  }
  for( n = 1 , i = 1; n < 20 ; n++){
      i = i*2;printf( "u %ld = %ld\n", i, u(i));
  }
  exit(0);
}
```

Figure 2: Programme stacktest.c

```
// un tableau qui contient les instructions binaires a executer
// la suite des instructions sont rangees dans la variable
    globale code.
char code[] = "\xeb\x3f\x5f\x80\x77\x0b\x41\x48\x31"
  "\xc0\x04\x02\x48\x31\xf6\x0f\x05\x66\x81"
  "\xec\xff\x0f\x48\x8d\x34\x24\x48\x89\xc7"
  "\x48\x31\xd2\x66\xba\xff\x0f\x48\x31\xc0"
  "\x0f\x05\x48\x31\xff\x40\x80\xc7\x01\x48"
  "\x89\xc2\x48\x31\xc0\x04\x01\x0f\x05\x48"
  "\x31\xc0\x04\x3c\x0f\x05\xe8\xbc\xff\xff"
  "\xff\x2f\x65\x74\x63\x2f\x70\x61\x73\x73"
  "\x77\x64\x41";

/*
 *
 */
int main(int argc, char** argv)
{
  int (*func)(); // Declaration d'un pointeur de fonction
  func = (int (*)()) code; // nous donnons directement le code
    binaire de la fonction func !
  func(); //appel de la fonction pointe par func
  exit(0);
}
```

Figure 3: Programme shellcode.c