Rapport ISI – Labo 5

Manipulations mémoire

Départements : TIN

Unité d'enseignement ISI

Auteurs : **Anatole Roy** et **Timothée Van Hove**

Professeur : **Abraham Rubistein**

Assistant : **Nathan Séville**

Classe : **B**

Date : **jeudi, 2 juin 2022**

Table des matières

1 Introduction

Ce travail s'inscrit dans le cadre de la formation Informatique et systèmes de communication de la HEIG-VD à Yverdon-les-Bains. Durant le cours « Introduction à la sécurité informatique » du 2e semestre, les élèves doivent réaliser un travail pratique portant sur le la manipulation mémoire dans un logiciel codé en C++ sur l'IDE NetBeans.

L'objectif de ce travail est d'appliquer les connaissances de l'étudiant vues en cours en matière de sécurité bas niveau, principalement en ce qui concerne les attaques par manipulation de mémoire (plus précisément « stack overflow »).

De plus, appliquer les connaissances de l'étudiant en programmation C / C++ et les familiariser à l'utilisation d'un debugger (code assembleur, visualisation de la mémoire).

Pour ce laboratoire, nous avons choisi de compiler notre programme pour une architecture 32 bits.

2 Programme à analyser

```
6  void evalue_fcc(int id, int in, int* res){
7    int vote[3]={0, 0, 0}; // vote[0]-feuille, vote[1]-caillou, vote[2]-ciseaux
8    int ordi[3]={0, 0, 0}; // ordi[0]-feuille, ordi[1]-caillou, ordi[2]-ciseaux
9    int i, alea;
10
11   vote[in]+=id;
12
13   alea = time(0) % 3;
14   ordi[alea]+=id;
15   cout << "L'ordi a mise sur ";
16   if ( ordi[0] )
17     cout << "Feuille";
18   else if ( ordi[1] )
19     cout << "Caillou";
20   else
21     cout << "Ciseaux";
22   cout << ". ";
23
24   *res = 0; // égal
25   for (i=0; i<3; i++){
26      if ( vote[i] )
27      {
28         switch (i)
29         {
30            case 0: // vote feuille
31               if (ordi[2]) // ordi vote ciseaux
32                  *res = 1; // perdu
33               else
34                  *res = 0; // gagne
35               break;
36            case 1: // vote caillou
37               if (ordi[0]) // ordi vote ciseaux
38                  *res = 1; // perdu
39               else
40                  *res = 0; // gagne
41               break;
42            case 2: // vote ciseaux
43               if (ordi[1]) // ordi vote ciseaux
44                  *res = 1; // perdu
45               else
46                  *res = 0; // gagne
47         }
48      }
49   }
50
51   if (*res != 0){
52     cout << "Perdu :-(" << endl;
53   }
54   else
55     cout << "Gagne :-)" << endl;
56 }
```

```
58 int main(){
59         int score=0;
60         int vote, id, i, res;
61 
62         // message de bienvenue
63         cout << "Bienvenue dans le jeux Feuille-Caillou-Ciseaux" << endl
64              << "-----------------------------------------------" << endl
65              << endl
66              << "Vous devez gagner un maximum de fois." << endl
67              << "Chaque fois que vous perdez, votre score augmente d'un point." <<
   ↪  endl
68              << "L'objectif est de terminer avec un score de 0." << endl
69              << endl;
70 
71         // choix de l'identifiant
72         cout << "Entrez votre identifiant entre 1 et 1000 : ";
73         while ( ! ( cin >> id ) )
74           {
75             cerr << "Erreur de saisie.\n";
76           }
77         cout << endl;
78 
79         // deroulement des 10 rounds de jeu
80         for (i=0; i<10; i++){
81           cout << "--- Round " << i+1 << " ---" << endl;
82           cout << "Choisissez "
83                << "[0] Feuille, "
84                << "[1] Caillou, "
85                << "[2] Ciseaux : ";
86           while ( ! ( cin >> vote ) )
87             {
88               cerr << "Erreur de saisie.\n";
89             }
90           cout << endl;
91 
92           evalue_fcc(id,vote,&res);      // evaluation du resultat du round
93 
94           if (res != 0 || vote < 0 || vote > 2)
95             score++;
96 
97           cout << "Score actuel est : " << score << endl;
98         }
99 
00         // affichage du score final
01         cout << endl;
02         cout << "Votre score final est :" << score << endl;
03 
04         return 0;
05 }
```

3 Réponse aux questions

Question 4.1 : Indiquer comment utiliser la faille fonctionnelle permettant de gagner à tous les coups. Illustrer la réussite par une capture d'écran du programme.

Réponse 4.1 :

Le vote de l'utilisateur est incrémenté par la valeur de la variable « id ». Si « id » vaut 0, alors le tableau de vote utilisateur ne contiendra que des 0 à tous ses indices.

De ce fait dans la boucle for, le branchement if retournera toujours false. En sortant de la boucle le résultat *res sera toujours égal à 0, car il a été initialisé à 0 au début de la fonction.

```
L'ordi a mise sur Ciseaux. Gagne :-)
Score actuel est : 0
--- Round 4 ---
Choisissez [0] Feuille, [1] Caillou, [2] Ciseaux : 1

L'ordi a mise sur Ciseaux. Gagne :-)
Score actuel est : 0
--- Round 5 ---
Choisissez [0] Feuille, [1] Caillou, [2] Ciseaux : 1

L'ordi a mise sur Ciseaux. Gagne :-)
Score actuel est : 0
--- Round 6 ---
Choisissez [0] Feuille, [1] Caillou, [2] Ciseaux : 1

L'ordi a mise sur Ciseaux. Gagne :-)
Score actuel est : 0
--- Round 7 ---
Choisissez [0] Feuille, [1] Caillou, [2] Ciseaux : 1

L'ordi a mise sur Ciseaux. Gagne :-)
Score actuel est : 0
--- Round 8 ---
Choisissez [0] Feuille, [1] Caillou, [2] Ciseaux : 1

L'ordi a mise sur Ciseaux. Gagne :-)
Score actuel est : 0
--- Round 9 ---
Choisissez [0] Feuille, [1] Caillou, [2] Ciseaux : 1

L'ordi a mise sur Ciseaux. Gagne :-)
Score actuel est : 0
--- Round 10 ---
Choisissez [0] Feuille, [1] Caillou, [2] Ciseaux : 1

L'ordi a mise sur Ciseaux. Gagne :-)
Score actuel est : 0

Votre score final est :0
```

```
void evalue_fcc(int id, int in, int* res){
  int vote[3]={0, 0, 0}; // vote[0]-feuille
  int ordi[3]={0, 0, 0}; // ordi[0]-feuille
  int i, alea;

  vote[in]+=id;

  alea = time(0) % 3;
  ordi[alea]+=id;
  cout << "L'ordi a mise sur ";
  if ( ordi[0] )
    cout << "Feuille";
  else if ( ordi[1] )
    cout << "Caillou";
  else
    cout << "Ciseaux";
  cout << ". ";

  *res = 0; // égal
  for (i=0; i<3; i++){
    if ( vote[i] )
    {
      switch (i)
      {
        case 0: // vote feuille
          if (ordi[2]) // ordi vote cis
            *res = 1; // perdu
          else
            *res = 0; // gagne
          break;
```

Question 4.2 : Expliquer, au niveau du code, comment cette attaque fonctionne. Décrire les modifications à apporter au programme pour fixer cette faille.

Réponse 4.2 :

L'explication de cette attaque est fournie dans la réponse 4.1

Pour réparer la faille nous proposons 3 approches :

1ère approche : Contrôler les entrées utilisateur. Il suffit d'interdire à l'utilisateur d'entrer un ID qui est égal à 0.

```
// choix de l'identifiant
cout << "Entrez votre identifiant entre 1 et 1000 : ";
while ( ! ( cin >> id ) or id >1000 or id <1)
  {
    cerr << "Erreur de saisie.\n";
  }
cout << endl;
```

2e approche : Dans la fonction, à la place d'incrémenter le tableau Vote et Ordi avec l'id, nous pouvons juste l'incrémenter par 1.

```
  vote[in]+=1;

  alea = time(0) % 3;
  ordi[alea]+=1;
```

3e approche : Le paramètre ID n'est pas nécessaire pour réaliser cette fonction. Il suffit de l'enlever et d'incrémenter les tableaux par 1.

```
void evalue_fcc(int in, int* res){
  int vote[3]={0, 0, 0}; // vote[0]-f
  int ordi[3]={0, 0, 0}; // ordi[0]-f
  int i, alea;

  vote[in]+=1;

  alea = time(0) % 3;
  ordi[alea]+=1;
```

Selon nous, l'approche n°3 est la plus efficace, car elle évite une entrée utilisateur inutile, donc cela évite un contrôle d'entrée utilisateur et cela supprime le bug, en simplifiant le programme.

Note : Dans la fonction, il est très dangereux d'utiliser un paramètre comme index d'un tableau. Si ce comportement est nécessaire, il faut impérativement contrôler que l'index est dans les bornes de la taille du tableau.

Question 4.3 : Chercher un moyen de modifier la pile. Étant donné que le joueur n'a que peu d'entrées possibles (identifiant, choix feuille-caillou-ciseau), cela ne doit pas être compliqué.

Réponse 4.3 :

Le paramètre « in » contrôle l'indice du tableau vote. Si ce paramètre à une valeur supérieure à la taille du tableau (buffer overflow), nous pouvons accéder aux éléments de la pile. L'élément qui nous intéresse est « saved EIP », car c'est l'emplacement dans la pile qui contient la valeur de retour de la fonction.

Dans notre cas, il serait intéressant de changer l'adresse de retour de la fonction dans le but de « sauter » l'incrément du score final.

Pour cela, nous avons plusieurs solutions :

1. Passer à la ligne juste après l'incrément (Faire toutes les boucles for et ne jamais incrémenter) à la ligne 97
2. Passer directement à l'affichage du score (directement sortir de la boucle for et aller à l'affichage) à la ligne 101

Nous avons choisi la solution n°2 car cela évite d'exécuter les 10 boucles for.

Question 4.4 : Grâce à la question précédente, il est donc possible de modifier la pile à l'endroit désiré. Quel paramètre est important pour modifier EIP/RIP ?

Réponse 4.4 :

Le paramètre « in » permet d'accéder à un indice du tableau vote. Ce paramètre prend la valeur du choix feuille, caillou, ciseau lors de l'entrée utilisateur. Nous devons donc utiliser ce paramètre pour faire le buffer overflow dans le but de modifier EIP.

En premier, nous devons savoir l'adresse du début de notre fonction, qui est représentée par le registre EBP.

```
eax                                    0x1
ecx                                    0xffffd67c
edx                                    0x1
ebx                                    0x0
esp                                    0xffffd630
ebp                                    0xffffd658
```

Ensuite, nous savons que l'adresse de la pile où est stockée l'adresse de retour de la fonction est dans le registre EIP. Ce registre se trouve toujours à EBP + 0x4 dans une architecture 32 bits.

Ensuite, nous devons savoir à quelle adresse de la pile se situe notre tableau vote. En analysant le code assembleur de notre fonction, nous avons trouvé que le tableau commence (vote[0]) à l'adresse EBP-0x1C et finit (vote[2]) à l'adresse EBP-0x14.

```
7     !  int vote[3]={0, 0, 0}; // vote[0]-feuille, vote[1]-caillou
8     _Z10evalue_fcciiPi+7: mov    DWORD PTR [ebp-0x1c],0x0
9     _Z10evalue_fcciiPi+14: mov    DWORD PTR [ebp-0x18],0x0
.0    _Z10evalue_fcciiPi+21: mov    DWORD PTR [ebp-0x14],0x0
```

Il suffit maintenant de représenter notre pile à l'aide d'un tableau :

Référence à EBP	Ordonnancement	Variable - Description	Adresse pile absolue
EBP+0x4	Saved EIP	Adresse de retour de la fonction	0xffffd65c
EBP	Saved EBP	Adresse de début de la fonction	0xffffd658
EBP-0x04	Variables locales		0xffffd654
EBP-0x08			0xffffd650
EBP-0x0c			0xffffd64c
EBP-0x10			0xffffd648
EBP-0x14		vote[2]	0xffffd644
EBP-0x18		vote[1]	0xffffd640
EBP-0x1c		vote[0]	0xffffd63c

Grâce à ces information, nous savons que pour accéder à saved EIP, il faut accéder à vote[8].

Question 4.5 : Grâce à la question précédente, il est donc possible de modifier EIP. Quel paramètre contrôle l'incrément d'EIP/RIP ?

Réponse 4.5

Nous pouvons voir que le tableau vote est incrémenté par le paramètre « id », à la ligne 11 du programme. `vote[in]+=id`

Question 4.6 et 4.7 : Quel incrément d'EIP/RIP permettrait de gagner ? Donner la logique (par rapport au code C++), quelles instructions devraient être sautées. Quelle serait la valeur de l'incrément d'EIP/RIP correspondant à la question précédente ? Justifier la réponse.

Réponses 4.6 et 4.7 :

Pour connaître l'incrément, il suffit de connaitre l'adresse de retour de la fonction et l'adresse à laquelle on veut se rendre. L'incrément sera la différence de ces deux adresses.

Comme expliqué à la réponse 4.3, nous avons choisi de sauter directement à l'affichage de notre fonction main().

Dans notre cas, l'adresse de retour de la fonction est main() + 609 et l'adresse que nous voulons accéder avec notre buffer overflow est main() + 705.

L'incrément, donc la valeur de la variable « id » est : $705 - 609 = 96$.

Cela nous permet de sortir de la boucle, et de directement executer l'affichage du score à la ligne 101

```
main()+604: call   0x5655622d <_Z10evalue_fcciiPi
main()+609: add    esp,0x10
!
!         if (res != 0 || vote < 0 || vote > 2)
main()+612: mov    eax,DWORD PTR [ebp-0x1c]
main()+615: test   eax,eax
main()+617: jne    0x565566a4 <main()+634>
main()+619: mov    eax,DWORD PTR [ebp-0x14]
main()+622: test   eax,eax
main()+624: js     0x565566a4 <main()+634>
main()+626: mov    eax,DWORD PTR [ebp-0x14]
main()+629: cmp    eax,0x2
main()+632: jle    0x565566a8 <main()+638>
!       |    score++;
main()+634: add    DWORD PTR [ebp-0xc],0x1
!
!         cout << "Score actuel est : " << score
main()+638: sub    esp,0x8
main()+641: push   0x565571ca
main()+646: push   0x56559040
main()+651: call   0xf7ea46b0 <_ZStlsISt11char_tr
main()+656: add    esp,0x10
main()+659: sub    esp,0x8
main()+662: push   DWORD PTR [ebp-0xc]
main()+665: push   eax
main()+666: call   0xf7ea4b50 <_ZNSolsEi>
main()+671: add    esp,0x10
main()+674: sub    esp,0x8
main()+677: push   0xf7ea3fe0
main()+682: push   eax
main()+683: call   0xf7ea31f0 <_ZNSolsEPFRSoS_E>
main()+688: add    esp,0x10
!       }
!
!       // affichage du score final
!       cout << endl;
main()+705: sub    esp,0x8
```

Question 4.8 : L'attaque ayant été analysée, il reste à la réaliser. Fournir une capture d'écran prouvant « la triche » (en ayant le score de 0 à la fin). Le déroulement du programme ainsi que le message de bienvenue au score final doivent bien évidemment y figurer.

```
Bienvenue dans le jeux Feuille-Caillou-Ciseaux
----------------------------------------------

Vous devez gagner un maximum de fois.
Chaque fois que vous perdez, votre score augmente d'un point.
L'objectif est de terminer avec un score de 0.

Entrez votre identifiant entre 1 et 1000 : 96

--- Round 1 ---
Choisissez [0] Feuille, [1] Caillou, [2] Ciseaux : 8

L'ordi a mise sur Feuille. Gagne :-)

Votre score final est :0

RUN FINISHED; exit value 0; real time: 3s; user: 0ms; system: 0ms
```

Comme prévu, en saisissant la valeur 96 pour l'identifiant et la valeur 8 comme choix de jeu, le programme nous affiche un score de 0, sans effectuer les 10 itérations.

4 Conclusion

Dans ce laboratoire, il fallait analyser le code qui nous a été fourni pour comprendre son fonctionnement. Nous avons trouvé deux manières de gagner au jeu. La première est d'entrer la valeur 0 comme identifiant. La deuxième, plus complexe, consiste à réaliser un buffer overflow pour gagner le jeu.

Le programme qui nous a été fourni a été réalisé dans un but académique, pour nous introduire à la mise en œuvre de manipulation de la mémoire. Il contient des failles assez grossières dans ce but, mais il illustre l'importance des bonnes pratiques dans la sécurité logicielle.