Année Universitaire 2021-2022 session 1 de printemps
Parcours : TCY Master CSI **UE :** 4TCY801U
Épreuve : Examen Sécurité logicielle
Date : 10 mai **Heure :** 9h **Durée :** 3h
Modalités : Épreuve sans document, calculatrice autorisée
Épreuve de : Mr Samuel Thibault 4 pages
université de BORDEAUX
Collège Sciences et technologie

Exercice 1. Les buffers contre-attaquent

J'exécute le programme ci-dessous dans gdb :

```
void f(void) {
    char buf[16];
    gets(buf);
    printf(buf);
}
int main(void) {
    f();
}
```

J'ai placé un breakpoint sur `printf` et j'ai tapé 15 fois la lettre `A` et tapé entrée. `pframe` me montre ceci :

```
0xffffc27c            0xf7dac905
0xffffc278            0x00000000
0xffffc274    arg2    0x56556070
0xffffc270    arg1    0x00000001
0xffffc26c    ret@    0x565561f8
0xffffc268      bp    0xffffc278
0xffffc264            0x00000000
0xffffc260            0xf7fdc480
0xffffc25c            0x00414141
0xffffc258            0x41414141
0xffffc254            0x41414141
0xffffc250            0x41414141
0xffffc24c            0x565561b9
0xffffc248            0x56559000
0xffffc244            0x00000001
0xffffc240            0xffffc250
0xffffc23c         sp 0x565561da
```

Q1.1 Expliquez ce qu'est `ret@`

Q1.2 Indiquez où se retrouvent les `A` dans la sortie de `pframe`.

Q1.3 La fonction `f` ne prend pas de paramètre, pourtant `pframe` montre 2 paramètres, comment cela se fait-il ?

Q1.4 Pourquoi `pframe` ne montre par contre pas 3 paramètres ?

Q1.5 Pourquoi est-ce une mauvaise idée d'utiliser la fonction `gets`, quel genre de faille cela permet ? Indiquez dans la sortie de `pframe` où le problème se poserait le plus.

Q1.6 À l'adresse `0xffffc260` on trouve une valeur non nulle, d'où peut-elle bien venir alors que dans `f` il n'y a pas d'autre variable locale que `buf` ?

Q1.7 Lorsque je relance le tout, les valeurs aux adresse `0xffffc26c` et `0xffffc23c` changent exactement de la même façon, à quoi est-ce dû ? Pourquoi fait-on cela ?

Q1.8 La valeur à l'adresse `0xffffc27c` change de manière similaire, à quoi correspond-elle ?

Q1.9 Pourquoi est-ce une mauvaise idée d'utiliser la fonction `printf` ainsi, quel genre de faille cela permet ?

Q1.10 Donnez un exemple simple d'attaque utilisant cette faille pour récupérer une information sur le programme en cours d'exécution, expliquez vos calculs et ce qui se passe.

Q1.11 Pourquoi est-ce intéressant d'obtenir cette information ?

Exercice 2. Lecture d'assembleur

Un programme, dont je n'ai pas le code source, contient la fonction suivante :

```
00000000 <f>:
   0: 55                    push   %ebp
   1: 31 c0                 xor    %eax,%eax
   3: 89 e5                 mov    %esp,%ebp
   5: 53                    push   %ebx
   6: 8b 55 08              mov    0x8(%ebp),%edx
   9: 8b 4d 0c              mov    0xc(%ebp),%ecx
   c: 3b 45 10              cmp    0x10(%ebp),%eax
   f: 74 0b                 je     1c <f+0x1c>
  11: 8b 1c 81              mov    (%ecx,%eax,4),%ebx
  14: 39 1c 82              cmp    %ebx,(%edx,%eax,4)
  17: 75 07                 jne    20 <f+0x20>
  19: 40                    inc    %eax
  1a: eb f0                 jmp    c <f+0xc>
  1c: 31 c0                 xor    %eax,%eax
  1e: eb 05                 jmp    25 <f+0x25>
  20: b8 01 00 00 00        mov    $0x1,%eax
  25: 5b                    pop    %ebx
  26: 5d                    pop    %ebp
  27: c3                    ret
```

Q2.1 Quelle·s instruction·s récupère·nt le·s argument·s de la fonction ?

Q2.2 Repérez le corps de la boucle, que se passe-t-il pour le registre `%eax` ?

Q2.3 Que signifie `(%ecx,%eax,4)` ?

Q2.4 Dans quel·s cas la boucle se termine-t-elle ?

Q2.5 Que fait la fonction, en fait ?

Q2.6 Dessinez l'état de la pile au moment de l'entrée dans la fonction.

Q2.7 Donner 2 raisons pour lesquelles il est intéressant d'utiliser `xor %eax,%eax` plutôt qu'un `mov`

Exercice 3. qmail fun

En 1997, Daniel J. Bernstein a promis 500$ à quiconque trouverait une faille de sécurité dans la dernière version de son serveur de mail `qmail` [1].

Il indique notamment plusieurs principes qui lui permettent d'affirmer que son logiciel est sûr, contrairement à d'autres implémentations d'un serveur de mail, telles que `sendmail`.

Do as little as possible in setuid programs.

Q3.1 Expliquez la fonctionalité `setuid`, et pourquoi le principe ci-dessus est important.

Don't parse.

Q3.2 Décrivez un exemple de faille potentielle due au *parsing*.

En 2005 Georgi Guninsky a rapporté une faille [2] sur machine 64 bit concernant les tailles de tableau dépassant des milliards d'octets. Daniel a répondu que « Nobody gives gigabytes of memory to each qmail-smtpd process », et effectivement les scripts de lancement de qmail-smtpd définissent des quotas à quelques Mo de mémoire seulement. Aucun quota n'a cependant été défini pour `qmail-local`, qui se trouve donc vulnérable.

Une partie du code problématique ressemble à ceci :

```
#define ALIGNMENT 16
char *alloc(unsigned int n)
{
  char *x;
  n = ALIGNMENT + n - (n & (ALIGNMENT - 1));
  return malloc(n);
}
```

Q3.3 Expliquez dans quel cas on peut effectivement avoir un débordement et quelles conséquences cela peut avoir selon le comportement de `malloc`.

C'est par contre difficile à exploiter, regardons ailleurs.

Une autre partie ressemble à ceci, elle sert à agrandir une allocation existante. `x->a` contient la taille réellement allouée, qui est volontairement un peu plus grande que la taille demandée `n` pour amortir le coût de réallocation. `alloc_re` s'occupe de faire la réallocation réelle du pointeur `x->field`.

```
1  void stralloc_readyplus(ta *x, unsigned int n)
2  {
3    unsigned int i;
4    if (x->field) {
5      i = x->a; n += x->len;
6      if (n > i) {
7        x->a = 30 + n + (n >> 3);
8        alloc_re(&x->field, i * sizeof(type), x->a * sizeof(type));
9      }
10     return;
11   }
12   x->len = 0;
13   x->field = (type *) alloc((x->a = n) * sizeof(type));
14 }
```

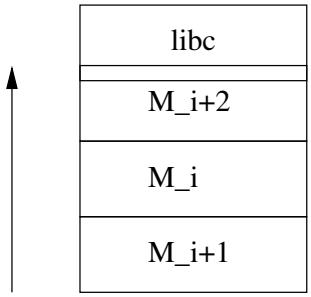
1. https://cr.yp.to/qmail/guarantee.html
2. https://www.guninski.com/where_do_you_want_billg_to_go_today_4.html

Q3.4 Décrivez la situation obtenue lorsque la ligne 7 déborde. Qu'est-il possible alors de faire pour exploiter cette faille ?

L'attaque [3] consiste alors à envoyer au serveur de mail un mail avec un en-tête d'un peu moins de 4Go. Celui-ci est découpé par le protocole SMTP en lignes d'au plus 998 octets voire seulement 78 octets. Il est alors lu par `qmail-local` ligne par ligne, jusqu'à atteindre la taille à laquelle le débordement a lieu. À chaque ligne le buffer de résultat est réalloué pour avoir la place d'ajouter la nouvelle ligne :

```
1 ||  if (!stralloc_readyplus(sa,n + 1)) return 0;
2 ||  byte_copy(sa->s + sa->len,n,s);
```

À partir d'une certaine taille, les allocations se font avec `mmap`, et elles se retrouvent alors juste à côté de la libc. Lors du débordement, avec l'exploitation mentionnée précédemment [4], on peut aboutir à la situation suivante où l'on a numéroté successivement les allocations M_i, M_{i+1}, et M_{i+2} (ces allocations ne sont pas à l'échelle sur la figure). On remarque que l'allocation M_{i+2} a écrasé le début de la libc :



Q3.5 Comment exploiter cette attaque pour pouvoir faire exécuter ce que l'on veut par `qmail-local` ?

Q3.6 Pourquoi l'ASLR rend une telle attaque bien plus difficile ?

Q3.7 Dans quelle mesure parvenir à lire la valeur mentionnée à la question 1.8 de l'exercice 1 permettrait de rendre l'attaque possible même avec ASLR ?

Épilogue : en pratique il n'est pas possible d'attaquer `root` avec cette méthode, mais attaquer n'importe quel utilisateur ayant une adresse mail sur le serveur de mail, oui.

`ret`

3. `https://www.qualys.com/2020/05/19/cve-2005-1513/remote-code-execution-qmail.txt`
4. `http://tukan.farm/2016/07/27/munmap-madness/`