Sécurité Logicielle

– Examen (**1**) –

1 Assembleur x86 (8 points)

Voici un le code source d'une fonction `main()` qui fait appel aux fonctions `func1()` et `func2()`

```
int main() {
  int array[] = {64, 34, 25, 12, 22, 11, 90};
  const int n = sizeof(array) / sizeof(array[0]);
  func1(array, n);
  func2(array, n);
  return 0;
}
```

Vous ne disposez que du code assembleur des fonctions `func1()` et `func2()` (ci-dessous). Essayez de reconstituer ce qu'elles font (au pire, donnez les grandes idées, au mieux reconstituez le code C originel). Notez que les ';;' en fin de ligne sont des commentaires qui vous donnent des informations sur le reste du contexte mémoire.

```
080484bb <func1>:
 80484bb <+0>:   55                      push   %ebp
 80484bc <+1>:   89 e5                   mov    %esp,%ebp
 80484be <+3>:   57                      push   %edi
 80484bf <+4>:   56                      push   %esi
 80484c0 <+5>:   8b 45 0c                mov    0xc(%ebp),%eax
 80484c3 <+8>:   53                      push   %ebx
 80484c4 <+9>:   8b 5d 08                mov    0x8(%ebp),%ebx
 80484c7 <+12>:  8d 50 ff                lea    -0x1(%eax),%edx
 80484ca <+15>:  89 d1                   mov    %edx,%ecx
 80484cc <+17>:  89 d0                   mov    %edx,%eax
 80484ce <+19>:  29 c8                   sub    %ecx,%eax
 80484d0 <+21>:  39 c2                   cmp    %eax,%edx
 80484d2 <+23>:  7e 1e                   jle    80484f2 <func1+55>
 80484d4 <+25>:  31 c0                   xor    %eax,%eax
 80484d6 <+27>:  39 c1                   cmp    %eax,%ecx
 80484d8 <+29>:  7e 15                   jle    80484ef <func1+52>
 80484da <+31>:  8b 34 83                mov    (%ebx,%eax,4),%esi
 80484dd <+34>:  8b 7c 83 04             mov    0x4(%ebx,%eax,4),%edi
 80484e1 <+38>:  39 fe                   cmp    %edi,%esi
 80484e3 <+40>:  7e 07                   jle    80484ec <func1+49>
 80484e5 <+42>:  89 3c 83                mov    %edi,(%ebx,%eax,4)
 80484e8 <+45>:  89 74 83 04             mov    %esi,0x4(%ebx,%eax,4)
 80484ec <+49>:  40                      inc    %eax
 80484ed <+50>:  eb e7                   jmp    80484d6 <func1+27>
 80484ef <+52>:  49                      dec    %ecx
 80484f0 <+53>:  eb da                   jmp    80484cc <func1+17>
 80484f2 <+55>:  5b                      pop    %ebx
 80484f3 <+56>:  5e                      pop    %esi
 80484f4 <+57>:  5f                      pop    %edi
 80484f5 <+58>:  5d                      pop    %ebp
 80484f6 <+59>:  c3                      ret
```

```
080484f7 <func2>:
 80484f7 <+0>:   55                      push   %ebp
 80484f8 <+1>:   89 e5                   mov    %esp,%ebp
 80484fa <+3>:   57                      push   %edi
 80484fb <+4>:   56                      push   %esi
 80484fc <+5>:   53                      push   %ebx
 80484fd <+6>:   31 db                   xor    %ebx,%ebx
 80484ff <+8>:   83 ec 0c                sub    $0xc,%esp
 8048502 <+11>:  8b 7d 08                mov    0x8(%ebp),%edi
 8048505 <+14>:  8b 75 0c                mov    0xc(%ebp),%esi
 8048508 <+17>:  39 f3                   cmp    %esi,%ebx
 804850a <+19>:  7d 15                   jge    8048521 <func2+42>
 804850c <+21>:  50                      push   %eax
 804850d <+22>:  50                      push   %eax
 804850e <+23>:  ff 34 9f                pushl  (%edi,%ebx,4)
 8048511 <+26>:  68 c0 85 04 08          push   $0x80485c0 ;; push @"%d "
 8048516 <+31>:  43                      inc    %ebx
 8048517 <+32>:  e8 e4 fd ff ff          call   8048300 <printf@plt>
 804851c <+37>:  83 c4 10                add    $0x10,%esp
 804851f <+40>:  eb e7                   jmp    8048508 <func2+17>
 8048521 <+42>:  c7 45 08 c4 85 04 08    movl   $0x80485c4,0x8(%ebp) ;; mov @"\n", %ebp+8
 8048528 <+49>:  8d 65 f4                lea    -0xc(%ebp),%esp
 804852b <+52>:  5b                      pop    %ebx
 804852c <+53>:  5e                      pop    %esi
 804852d <+54>:  5f                      pop    %edi
 804852e <+55>:  5d                      pop    %ebp
 804852f <+56>:  e9 dc fd ff ff          jmp    8048310 <puts@plt>
```

2 `return-to-csu` : A New Method to Bypass 64-bit Linux ASLR (12 points)

Lisez l'article "***`return-to-csu`*** *: A New Method to Bypass 64-bit Linux ASLR*" par H. Marco-Gisbert et I. Ripoll (BlackHat Asia 2018, Singapour).

Questions

1. Rappelez les principes de l'ASLR (Address-Space Layout Randomization), du SSP (Stack Smashing Protector), du NX (No-eXecute), du RELRO (RELocation Read-Only) ainsi que du PIE (Position-Independant-Executable) et contre quels types d'attaques il ont chacun été introduit.
2. Expliquez quelles sont les parties intéressantes des Gadgets 1 et 2 (listing 3 p.7) et ce qu'elles peuvent permettre de faire si on les chaînes comme sur la figure 2 (p.8).
3. Rappelez en quoi consiste un ROPshell et donnez la liste des instructions manquantes pour créer un ROPshell si on considère juste les gadgets 1 et 2.
4. Les auteurs supposent (p.8) que les applications qu'ils vont attaquer auront toujours `read()`, `write()`, `send()` et `recv()` dans leur PLT. Déduisez-en de quel type de programmes il s'agit essentiellement (et expliquez pourquoi).
5. Les auteurs supposent que l'adresse fuitée permet d'accéder de manière globale à l'ensemble de la `libc`. Expliquer pourquoi, et est-ce vraiment la seule information qu'il est nécessaire de connaître pour réaliser l'attaque ou bien les auteurs ont-ils oublié une autre hypothèse ?
6. Que se passe-t-il si le programme attaqué contient des canaris (`-fstack-protector-all`) ? Décrire ce que cela change dans l'attaque et sous quelles conditions l'attaque est réalisable.
7. Critiquez la figure 5 (p.10), les auteurs ne sont ils pas exagérément optimistes pour certains arcs du dessin ? Expliquez, par exemple, quel est l'intérêt d'appliquer l'attaque `return-to-csu` après une reconstruction complète de l'espace mémoire via du brute-force (chemin de gauche sur la figure). Motivez toutes vos critiques !
8. Quelles sont les trois solutions proposées pour supprimer ce risque de sécurité ? Expliquez les principes de base des trois solutions proposées dans l'article et discutez-en la validité.