Sécurité Logicielle

– Examen (1) –

1 Assembleur x86-32 (Barême approximatif : 5 points)

Reconstituez (approximativement) ce que fait le programme suivant et essayez d'en deviner l'usage (il s'agit d'une fonction que l'on donne souvent en exercice de programmation). Trouvez aussi la mesure de protection qui a été rajoutée au programme par le compilateur.

```
Disassembly of section .text:

0000000000400510 <main>:
  400510: 41 54                         push   %r12
  400512: 55                            push   %rbp
  400513: bf 24 07 40 00                mov    $0x400724,%edi
  400518: 53                            push   %rbx
  400519: bd 01 00 00 00                mov    $0x1,%ebp
  40051e: 31 db                         xor    %ebx,%ebx
  400520: 45 31 e4                      xor    %r12d,%r12d
  400523: 48 83 ec 10                   sub    $0x10,%rsp
  400527: 64 48 8b 04 25 28 00 00 00    mov    %fs:0x28,%rax
  400530: 48 89 44 24 08                mov    %rax,0x8(%rsp)
  400535: 31 c0                         xor    %eax,%eax
  400537: e8 74 ff ff ff                callq  4004b0 <puts@plt>
  40053c: 48 8d 74 24 04                lea    0x4(%rsp),%rsi
  400541: bf 3e 07 40 00                mov    $0x40073e,%edi
  400546: 31 c0                         xor    %eax,%eax
  400548: e8 a3 ff ff ff                callq  4004f0 <__isoc99_scanf@plt>
  40054d: 3b 5c 24 04                   cmp    0x4(%rsp),%ebx
  400551: 7d 20                         jge    400573 <main+0x63>
  400553: 83 fb 01                      cmp    $0x1,%ebx
  400556: 89 de                         mov    %ebx,%esi
  400558: 7e 09                         jle    400563 <main+0x53>
  40055a: 41 8d 34 2c                   lea    (%r12,%rbp,1),%esi
  40055e: 41 89 ec                      mov    %ebp,%r12d
  400561: 89 f5                         mov    %esi,%ebp
  400563: bf 41 07 40 00                mov    $0x400741,%edi
  400568: 31 c0                         xor    %eax,%eax
  40056a: ff c3                         inc    %ebx
  40056c: e8 5f ff ff ff                callq  4004d0 <printf@plt>
  400571: eb da                         jmp    40054d <main+0x3d>
  400573: 31 c0                         xor    %eax,%eax
  400575: 48 8b 54 24 08                mov    0x8(%rsp),%rdx
  40057a: 64 48 33 14 25 28 00 00 00    xor    %fs:0x28,%rdx
  400583: 74 05                         je     40058a <main+0x7a>
  400585: e8 36 ff ff ff                callq  4004c0 <__stack_chk_fail@plt>
  40058a: 48 83 c4 10                   add    $0x10,%rsp
  40058e: 5b                            pop    %rbx
  40058f: 5d                            pop    %rbp
  400590: 41 5c                         pop    %r12
  400592: c3                            retq

Display of section .rodata:
  400724: .string "Enter the number of terms"
  40073e: .string "%d"
  400741: .string "%d\n"
```

2 How the ELF Ruined Christmas (Barême indicatif : 15 points)

Lisez l'article "*How the ELF Ruined Christmas*" par A. Di Federico, A. Cama, Y. Shoshitaishvili, C. Kruegel, et G. Vigna (USENIX Security, 2015). Puis rédigez des réponses aux questions suivantes.

Questions

1. Lister et expliquer les différentes techniques d'exploitation et de protection qui sont listées dans la section 2.
2. Expliquez le principe de la résolution paresseuse des symboles externes (*lazy symbol resolution*) pour le format exécutable ELF à travers les tables GOT et PLT.
3. Expliquez les mécanismes du RELRO (*partial* et *full*), et donnez quelques exemples d'attaques qui sont bloquées ou rendues plus difficiles par ce mécanisme.
4. Expliquez les étapes successives de l'attaque de base proposée dans la section 4 de l'article.
5. Expliquez comment appliquer l'attaque proposée dans le cas d'un programme durci avec du '*partial RELRO*', puis du '*full RELRO*' ?
6. Enfin, expliquez les limites d'une telle technique et expliques les contre-mesures proposées dans l'article pour rendre l'exploitation de bugs plus difficile en détaillant les avantages et les inconvénients de chacune des contre-mesures proposées.