Sécurité Logicielle

– Examen (1) –

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

Reconstituez (approximativement) le code C de la fonction qui correspond au code assembleur suivant et expliquez à quoi il sert à votre avis :

```
xvnm:
804863b:    55                      push   %ebp
804863c:    89 e5                   mov    %esp,%ebp
804863e:    83 ec 10                sub    $0x10,%esp
8048641:    c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%ebp)
8048648:    c1 65 fc 02             shll   $0x2,-0x4(%ebp)
804864c:    83 4d fc 03             orl    $0x3,-0x4(%ebp)
8048650:    eb 5a                   jmp    80486ac <xvnm+0x71>
8048652:    8b 45 08                mov    0x8(%ebp),%eax
8048655:    0f b6 00                movzbl (%eax),%eax
8048658:    3c 40                   cmp    $0x40,%al
804865a:    7e 23                   jle    804867f <xvnm+0x44>
804865c:    8b 45 08                mov    0x8(%ebp),%eax
804865f:    0f b6 00                movzbl (%eax),%eax
8048662:    3c 5a                   cmp    $0x5a,%al
8048664:    7f 19                   jg     804867f <xvnm+0x44>
8048666:    8b 45 08                mov    0x8(%ebp),%eax
8048669:    0f b6 10                movzbl (%eax),%edx
804866c:    8b 45 fc                mov    -0x4(%ebp),%eax
804866f:    01 c0                   add    %eax,%eax
8048671:    21 d0                   and    %edx,%eax
8048673:    89 c2                   mov    %eax,%edx
8048675:    83 ca 21                or     $0x21,%edx
8048678:    8b 45 08                mov    0x8(%ebp),%eax
804867b:    88 10                   mov    %dl,(%eax)
804867d:    eb 29                   jmp    80486a8 <xvnm+0x6d>
804867f:    8b 45 08                mov    0x8(%ebp),%eax
8048682:    0f b6 00                movzbl (%eax),%eax
8048685:    3c 60                   cmp    $0x60,%al
8048687:    7e 1f                   jle    80486a8 <xvnm+0x6d>
8048689:    8b 45 08                mov    0x8(%ebp),%eax
804868c:    0f b6 00                movzbl (%eax),%eax
804868f:    3c 7a                   cmp    $0x7a,%al
8048691:    7f 15                   jg     80486a8 <xvnm+0x6d>
8048693:    8b 45 08                mov    0x8(%ebp),%eax
8048696:    0f b6 10                movzbl (%eax),%edx
8048699:    8b 45 fc                mov    -0x4(%ebp),%eax
804869c:    21 d0                   and    %edx,%eax
804869e:    89 c2                   mov    %eax,%edx
80486a0:    83 ca 30                or     $0x30,%edx
80486a3:    8b 45 08                mov    0x8(%ebp),%eax
80486a6:    88 10                   mov    %dl,(%eax)
80486a8:    83 45 08 01             addl   $0x1,0x8(%ebp)
80486ac:    8b 45 08                mov    0x8(%ebp),%eax
80486af:    0f b6 00                movzbl (%eax),%eax
80486b2:    84 c0                   test   %al,%al
80486b4:    75 9c                   jne    8048652 <xvnm+0x17>
80486b6:    c9                      leave
80486b7:    c3                      ret
```

2 A Eulogy for Format Strings (Barême approximatif : 15 points)

Lisez l'article "*A Eulogy for Format Strings*" par Captain Planet (Phrack #67, 2010). Puis rédigez des réponses aux questions suivantes.

Questions

1. Quelle syntaxe assembleur est utilisée dans cet article (AT&T ou Intel) ? Et, est-ce de l'assembleur 32bits ou bien 64bits ?
2. Rappelez rapidement les principes de l'exploitation d'un bug de format.
3. Quel est l'effet, relatif aux bugs de format, de l'option '`-D_FORTIFY_SOURCE=2`' de `gcc` sur l'assembleur directement généré ?
4. La fonction '`__printf_chk()`' n'est finalement qu'un wrapper autour de '`vfprintf()`' mais que fait-elle de plus exactement ?
5. Dans la protection #1, décrite à la section B, que se passe-t-il lorsque le flag '`_IO_FLAGS2_FORTIFY`' est actif ? Et, en quoi cela pose-t-il problème lors d'une tentative d'exploitation d'un bug de format ?
6. La protection #2, décrite à la section C, empêche certaines possibilités au niveau des arguments passés dans la chaîne de format, expliquez lesquelles.
7. Expliquez, de manière plus détaillée que dans l'article, comment réaliser le '*Arbitrary 4-byte NUL overwrite*' en expliquant les différentes étapes. Et, dites à quoi va servir cette fonctionnalité dans le contexte de l'article.
8. Expliquez en quoi mettre la variable `nargs` à zéro (`nargs = 0`) va permettre de désactiver la deuxième protection ?
9. Dans l'exemple d'entraînement, expliquer à quoi sert chacun des trois éléments de la chaîne de format donnée dans l'article :

```
Disabling both :nargs: and fortify source: [%*472$ %1$*269145004$ %1073741824$]
```

10. Expliquez l'exploit réalisé sur le serveur CUPS. Et, dites pourquoi l'ASLR rendrait problématique l'exploitation du bug.
11. Donnez votre opinion concernant les techniques proposées pour contrer l'ASLR. Sont-elles réalistes ? Peut-on proposer mieux ?
12. Suggérez de petites modifications du code de la `glibc` pour compliquer la vie de l'attaquant.