Sécurité Logicielle

– Examen (**1**) –

1 Assembleur x86-32 (Barême approximatif : 8 points)

Reconstituez (approximativement) le code C de la fonction qui correspond au code assembleur suivant.
Indice : 'A' = 0x41, 'Z' = 0x5a, 'a' = 0x61, 'z' = 0x7a, '0' = 0x30, '!' = 0x21.

```
tiezao:
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 <tiezao+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 <tiezao+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 <tiezao+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 <tiezao+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 <tiezao+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 <tiezao+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 <tiezao+0x17>
80486b6:    c9                      leave
80486b7:    c3                      ret
```

2 Framing Signals – A Return to Portable Shellcode (Barême approximatif : 12 points)

Lisez l'article "*Framing Signals – A Return to Portable Shellcode*" par Erik Bosman et Herbert Bos (IEEE Symposium on Security and Privacy, 2014). Puis rédigez des réponses aux questions suivantes.

Questions

1. Rappellez les principes des attaques de type `ret-into-libc` et de ROP (Return-Oriented-Programming), donnez la liste des protections qu'elles contournent et leurs limites face à certaines autres protections.
2. Expliquez les points principaux qui constituent un appel de signal en système UNIX.
3. Donnez les conditions pour qu'un SROP soit réalisable et donnez quelques exemples de fonctions (au moins deux) par lesquelles on pourrait réaliser l'injection tout en satisfaisant ces conditions.
4. Quels sont les principaux avantages/défauts d'un gadget `sigreturn` par rapport à un gadget classique '`syscall;ret`' ?
5. Expliquez à quoi sert la section de code `vsyscall` (le type de fonction qu'on y trouve et pourquoi les avoir séparées des autres). Et, expliquez pour quelles raisons elle a été supprimée.
6. Pourquoi est-il nécessaire de pouvoir écrire des octets `NULL` sur la pile pour pouvoir réaliser cette technique.
7. Détaillez l'attaque décrite à la section E. en résumant les principales étapes.
8. Expliquez le principe de "*SROP as a backdoor*" et donnez les hypothèses nécessaires pour qu'une telle technique puisse marcher.
9. Expliquez comment la technique du SROP permet de contourner la signature de binaires sous iOS.
10. Pourquoi est-il important d'avoir un ensemble de gadgets qui permette de simuler un langage Turing-complet lorsqu'on considère ce genre de techniques d'exploitation ?