Sécurité Logicielle

– Examen (**1**) –

1 Assembleur x86-32 (5 points)

Reconstituez (approximativement) le code C qui correspond au code assembleur suivant (et signalez quelle option particulière de `gcc` a été utilisée pendant la compilation) :

```
.LC0:
        .string "i vaut: %d\n"
        .text
        .globl  func
        .type   func, @function
func:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $28, %esp
        movl    8(%ebp), %ebx
        movl    %gs:20, %eax
        movl    %eax, -12(%ebp)
        xorl    %eax, %eax
        pushl   %ebx
        pushl   $.LC0
        call    printf
        addl    $16, %esp
        xorl    %eax, %eax
        jmp     .L2
.L3:
        imull   %ebx, %eax
        incl    %eax
.L2:
        cmpl    %ebx, %eax
        jl      .L3
        imull   %ebx, %eax
        subl    $10, %eax
        movl    -12(%ebp), %edx
        xorl    %gs:20, %edx
        je      .L4
        call    __stack_chk_fail
.L4:
        movl    -4(%ebp), %ebx
        leave
        ret
```

2 G-Free : Defeating Return-Oriented Programming through Gadget-less Binaries (15 points)

Lisez l'article "*G-Free : Defeating Return-Oriented Programming through Gadget-less Binaries*" par Kaan Onarlioglu, Leyla Bilge, Andrea Lanzi, Davide Balzarotti et Engin Kirda (Annual Computer Security Applications Conference, 2010). Puis rédigez des réponses aux questions suivantes.

Questions

1. Quelle syntaxe assembleur est utilisée dans cet article (AT&T ou Intel) ?
2. Rappelez rapidement le principe d'un ret-into-libc.
3. Rappelez rapidement le principe du Return-Oriented-Programming (ROP). Et, expliquez en quoi il diffère du ret-into-libc (qu'est-ce qu'il permet de plus).
4. À votre avis, qu'appelle-t-on un gadget '*non-intentionnel*' (aussi appelé '*non-aligné*' dans l'article) ?
5. Montrez, par l'exemple, qu'il est possible de créer des gadgets sans '`ret`' final. Et dites s'il est possible d'être Turing-complet sans utiliser de '`ret`' ? Argumentez !
6. Expliquez la technique des '*Alignment Sleds*'. Et, précisez comment ils diminuent le nombre de gadgets présents dans l'exécutable.
7. Expliquez la technique des '*Frame Cookies*'. Et, précisez quelle famille de gadgets il tente de rendre inopérant.
8. Expliquez le principe général du '*Code Rewriting*'. Et, précisez quelle famille de gadgets il tente de supprimer.
9. Expliquez les limites de cette approche en donnant un cas où un gadget (ou une série de gadgets) ne pourra pas être enlevé.
10. Quels sont, à votre avis, les problèmes majeurs rencontrés par les auteurs de l'article pendant l'implémentation du mécanisme de protection ? Argumentez !
11. Cette approche vous semble-t-elle efficace et quelles sont ses limites et quel impact a-t-elle sur l'assembleur ainsi produit ? En outre, à votre avis, existe-t-il des cas où ce mécanisme ne pourra venir à bout de certains gadgets ? Si oui, décrivez un scénario qui vous semble réaliste en pratique.