# Sécurité Système

– Examen (3h, documents autorisés) –

## 1 Rétro-ingénierie logicielle (10 points)

**Questions**

1. (.5 points) Lister 2 avantages et 1 inconvénient de compiler un exécutable en « dynamique » plutôt qu'en « statique » ?

2. (.5 points) Que fait l'instruction '`lea eax, [ebx*8]`' ?

3. (1 point) Qu'est-ce que la résolution de symboles dite « fainéante » (*lazy loading*) ?

4. (.5 points) Quelles opérations sont effectuées par le compilateur lors de la phase de pré-processeur ?

5. (1 point) Quel est le but de l'instruction suivante : '`and eax, 1`' ?

6. (.5 points) Qu'est-ce qu'une '*immediate*' en assembleur ?

7. (.5 points) Que contient la section `.text` d'un binaire ?

8. (.5 points) Qu'est-ce que l'*entry point* d'un exécutable ?

9. (.5 points) Qu'implique le fait qu'un binaire soit « strippé » (*stripped*) ?

10. Le mnémonique `movsb` est décrit comme tel par Intel :

**Description movs (MOVS/MOVSB/MOVSW/MOVSD/MOVSQ)**

Moves the byte, word, or doubleword specified with the second operand (source operand) to the location specified with the first operand (destination operand). Both the source and destination operands are located in memory.
The address of the source operand is read from the DS:ESI or the DS:SI registers (depending on the address-size attribute of the instruction, 32 or 16, respectively). The address of the destination operand is read from the ES:EDI or the ES:DI registers (again depending on the address-size attribute of the instruction).
The DS segment may be overridden with a segment override prefix, but the ES segment cannot be overridden. [...]
The MOVS, MOVSB, MOVSW, and MOVSD instructions can be preceded by the REP prefix [...] for block moves of ECX bytes, words, or doublewords.

(a) (.5 points) Que fait la suite d'instruction ci-dessous :

```
mov ecx, [ebp + 16]
mov esi, [ebp + 12]
mov edi, [ebp + 8]
rep movsb
```

(b) (.5 points) À quelle fonction de la bibliothèque standard C cela vous fait-il penser ?

11. Les mnémoniques '`cdqe`' et '`imul`' sont décrits de la sorte par Intel :

**Description cdqe (CBW/CWDE/CDQE)**

Convert Byte to Word / Convert Word to Doubleword / Convert Doubleword to Quadword.

| Opcode | Instruction | Description |
|---|---|---|
| 0x98 | CBW | AX := sign-extend of AL |
| 0x98 | CWDE | EAX := sign-extend of AX |
| REX.W + 0x98 | CDQE | RAX := sign-extend of EAX |

Double the size of the source operand by means of sign extension. The CBW (convert byte to word) instruction copies the sign (bit 7) in the source operand into every bit in the AH register. The CWDE (convert word to double-word) instruction copies the sign (bit 15) of the word in the AX register into the high 16 bits of the EAX register. [...]
In 64-bit mode, the default operation size is the size of the destination register. Use of the REX.W prefix promotes this instruction (CDQE when promoted) to operate on 64-bit operands. In which case, CDQE copies the sign (bit 31) of the doubleword in the EAX register into the high 32 bits of RAX [...]

**Description imul (IMUL)**

IMUL: Signed Multiply.
Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands. [...]
Two-operand form - With this form the destination operand (the first operand) is multiplied by the source operand (second operand). The destination operand is a general-purpose register and the source operand is an immediate value, a general-purpose register, or a memory location. The intermediate product (twice the size of the input operand)

```
is truncated and stored in the destination operand location. [...]
In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R
prefix permits access to additional registers (R8-R15). Use of the REX.W prefix
promotes operation to 64 bits. Use of REX.W modifies the three forms of the
instruction as follows [...]
Two-operand form - The source operand is promoted to 64 bits if it is a register or a
memory location. The destination operand is promoted to 64 bits.
```

| Opcode | Instruction | Description |
|---|---|---|
| 0xF6 /5 | IMUL r/m8 | AX := AL r/m byte |
| 0xF7 /5 | IMUL r/m16 | DX:AX := AX r/m word |
| 0xF7 /5 | IMUL r/m32 | EDX:EAX := EAX r/m32 |
| REX.W + 0xF7 /5 | IMUL r/m64 | RDX:RAX := RAX r/m64 |
| 0x0F AF /r | IMUL r16, r/m16 | word register := word register r/m16 |
| 0x0F AF /r | IMUL r32, r/m32 | doubleword register := doubleword register r/m32 |
| REX.W + 0x0F AF /r | IMUL r64, r/m64 | quadword register := quadword register r/m64 |

Soit l'extrait d'assembleur x86-x64 suivant :

```
secret:
  push rbp
  mov  rbp, rsp
  mov  dword ptr [rbp-20], 9
  mov  qword ptr [rbp-16], 1
  mov  dword ptr [rbp-4], 1
  jmp  short loc_401646
loc_401631:
  mov  eax, [rbp-4]
  cdqe
  mov  rdx, [rbp-16]
  imul rax, rdx
  mov  [rbp-16], rax
  add  dword ptr [rbp-4], 1
loc_401646:
  mov  eax, [rbp-4]
  cmp  eax, [rbp-20]
  jle  short loc_401631
  mov  rax, [rbp-16]
  pop  rbp
  retn
```

(a) (1 point) Écrire le pseudo-code correspondant à la fonction `secret`.

(b) (.5 points) Que calcule cette fonction ?

12. (2 points) Considérons le programme désassemblé suivant :

```
; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
```

```
public main
main proc near

var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

; __unwind {
    push  ebp
    mov   ebp, esp
    sub   esp, 20h
    mov   [ebp+var_1C], 0Ah
    mov   [ebp+var_18], 4
    mov   [ebp+var_14], 6
    mov   [ebp+var_10], 2
    mov   [ebp+var_C], 9
    mov   [ebp+var_8], 1Dh
    mov   [ebp+var_4], 1Bh
    push  7
    lea   eax, [ebp+var_1C]
    push  eax
    call  mystery
    add   esp, 8
    mov   eax, 0
    leave
    retn
; } // starts at 8049DB1
main endp

; =============== S U B R O U T I N E =======================================
; Attributes: bp-based frame

public mystery
mystery   proc near         ; CODE XREF: main+3D↓p

var_C     = dword ptr -0Ch
var_8     = dword ptr -8
var_4     = dword ptr -4
arg_0     = dword ptr  8
arg_4     = dword ptr  0Ch

; __unwind {
    push  ebp
    mov   ebp, esp
    sub   esp, 10h
    mov   [ebp+var_8], 1
    jmp   loc_8049DA0

loc_8049D07:                ; CODE XREF: mystery+B1↓j
    mov   [ebp+var_4], 0
```

```
    jmp   short loc_8049D8D

loc_8049D10:                ; CODE XREF: mystery+A1↓j
    mov   eax, [ebp+var_4]
    lea   edx, ds:0[eax*4]
    mov   eax, [ebp+arg_0]
    add   eax, edx
    mov   edx, [eax]
    mov   eax, [ebp+var_4]
    add   eax, 1
    lea   ecx, ds:0[eax*4]
    mov   eax, [ebp+arg_0]
    add   eax, ecx
    mov   eax, [eax]
    cmp   edx, eax
    jle   short loc_8049D89
    mov   eax, [ebp+var_4]
    lea   edx, ds:0[eax*4]
    mov   eax, [ebp+arg_0]
    add   eax, edx
    mov   eax, [eax]
    mov   [ebp+var_C], eax
    mov   eax, [ebp+var_4]
    add   eax, 1
    lea   edx, ds:0[eax*4]
    mov   eax, [ebp+arg_0]
    add   eax, edx
    mov   edx, [ebp+var_4]
    lea   ecx, ds:0[edx*4]
    mov   edx, [ebp+arg_0]
    add   edx, ecx
    mov   eax, [eax]
    mov   [edx], eax
    mov   eax, [ebp+var_4]
    add   eax, 1
    lea   edx, ds:0[eax*4]
    mov   eax, [ebp+arg_0]
    add   edx, eax
    mov   eax, [ebp+var_C]
    mov   [edx], eax

loc_8049D89:                ; CODE XREF: mystery+42↑j
    add   [ebp+var_4], 1

loc_8049D8D:                ; CODE XREF: mystery+19↑j
    mov   eax, [ebp+arg_4]
    sub   eax, 1
    cmp   [ebp+var_4], eax
    jl    loc_8049D10
    add   [ebp+var_8], 1

loc_8049DA0:                ; CODE XREF: mystery+D↑j
    mov   eax, [ebp+var_8]
    cmp   eax, [ebp+arg_4]
    jle   loc_8049D07
    mov   eax, [ebp+arg_0]
    leave
```

```
    retn
; } // starts at 8049CF5
mystery endp
```

Dites ce que fait ce programme en décomposant la fonction `main()` et la fonction `mystery()` (sans forcément donner leur pseudo-code même si cela peut aider).

## 2 Programmation noyau (10 points)

Lisez l'article "*Finding hidden kernel modules (extrem way reborn): 20 years later*" de `g1nko`, Phrack #71, août 2024. Puis, répondez aux questions suivantes.

**Questions**

1. (1 point) Rappelez brièvement ce qu'est un *linux kernel rootkit* (LKM), ce qu'il cherche à faire et des droits dont il faut disposer pour pouvoir l'installer sur une machine.

2. (1 point) Expliquez la méthode que nous avons vue en cours pour cacher un module du noyau Linux. Puis, expliquez comment la contourner d'après la méthode évoquée dans la section 1 de l'article.

3. (.5 points) Comment KoviD contourne-t-il la méthode de détection évoquée précédemment ?

4. (.5 points) Quelle solution propose l'auteur pour détecter les modules cachés ?

5. (.5 points) L'anti-rootkit `rkspotter` utilisait une méthode du noyau pour détecter les modules cachés. Quelle était cette méthode ? Et pourquoi n'est-elle plus utilisable aujourd'hui ?

6. (.5 points) Dans la section 2.3 l'auteur signale qu'il risque d'avoir plus de faux-positifs sur une architecture 64 bits. Pourquoi ? Et comment l'auteur propose-t-il de résoudre ce problème ?

7. (1 point) Dans la section 3, l'auteur donne les septs champs de la structure qu'il vérifie pour identifier un module. Lesquels sont-ils ?

8. (.5 points) Quel est le problème évoqué par l'auteur dans la section 3.2, quelles en seraient les conséquences, comment propose-t-il de le résoudre et pourquoi est-ce plus rapide ?

9. (.5 points) Les modules sont-il mappé en mémoire physique, en mémoire virtuelle ou les deux ?

10. (.5 points) Scanner l'ensemble de la mémoire du noyau en force brute est une opération risquée. Pourquoi ? Et, comment l'auteur propose-t-il de le faire en supprimant ces risques ?

11. (.5 points) Que signifie l'acronyme MMU et à quoi sert ce composant ?

12. (.5 points) Quelles sont les différentes architectures supportées par le programme tel qu'il est livré dans l'article ? Expliquez ce qui vous fait dire cela.

13. (.5 points) À partir de quelle version du noyau Linux, la taille mémoire des modules n'est elle plus contenue dans un seul champ ?

14. (1 point) Expliquer à quoi sert la fonction `check_name_valid(char *s)` et que fait-elle ?

15. (1 point) D'après le code présenté à la fin de l'article, comment déclenche-t-on le scan des modules ?