PROTECTIONS LOGICIELLES

Nous avons tous, un jour ou l'autre, tenté de copier des fichiers sans succès, lister un programme Basic sans rien obtenir de cohérent. Le pourquoi du comment est simple : ces fichiers et listings étaient protégés pour que vous ne puissiez rien voir de leur contenu.

Nous allons vous expliquer le fonctionnement de ces protections ; ainsi, vous pourrez vous amuser comme des petits fous pour mettre vos chefs-d'œuvre à l'abri des regards indiscrets.

Voici donc en exclusivité mondiale, pour les lecteurs d'*Amstrad Cent Pour Cent*, l'initiation qui, même dans vos rêves les plus fous, n'aurait été imaginable : comment protéger vos créations personnelles ?

XOR LE MAGNIFIQUE

Pour m'aider à préparer ce dossier, j'ai simplement demandé à l'un des plus férus en la matière, XOR, de me révéler quelques-uns de ses secrets. Il va de soi que sous ce pseudo (bien connu dans le milieu des tripatouilleurs d'octets) se cache une tête remplie d'idées, qui nous a fait la gentillesse de partager son savoir avec nous.

TOI PAS LIRE MOI

Cela fait des années que les programmeurs protègent leurs logiciels ; de même, tous les jours de nouvelles méthodes et ruses arrivent à leur esprit tordu. Or donc, pour vous parler de tout, il nous faudrait écrire un livre (tiens, c'est pas une mauvaise idée). Donc, sans être trop prétentieux, nous allons commencer par le commencement, à savoir la protection des fichiers en lecture.
La méthode la plus simple et la moins efficace est la « sauvegarde par l'option P ». En effet, encodez un fichier Basic de quelques lignes, sauvegardez-le comme suit :

```
SAVE "TOTO",P
```

Réinitialisez votre ordinateur et chargez le fichier par :

```
LOAD "TOTO"
```

Faites « LIST ». Etrange, non? Pas de liste disponible. C'est comme si le fichier n'avait jamais existé. Pourtant, le fichier ainsi protégé peut être exécuté normalement par :

```
RUN "TOTO"
```

Cette fameuse « option P » code le programme. L'ordinateur teste ce codage en exécutant la commande LOAD. S'il constate une protection, il exécute un NEW en fin de chargement, d'où l'impossibilité de lister le programme car, en fait, celui-ci n'est plus en mémoire.

Aujourd'hui, n'importe quel logiciel de traitement de fichier possède une instruction pour supprimer cette protection. Renommez, par exemple, un tel fichier à l'aide de Discology en supprimant le [P]. Voilà, le fichier vient d'être déprotégé.
Il existe d'autres méthodes plus radicales pour cacher une liste. Vous pouvez, par exemple, cacher des lignes Basic qui n'apparaîtront pas à la commande LIST, ajouter un programme en langage machine à la suite de votre source Basic, encoder une petite routine machine dans le programme lui-même — et cela sous les yeux du voyeur —, écrire vos fichiers dans les User inaccessibles au Basic, automodifier un programme Basic (poker de nouveaux Token) ; ou encore plus compliqué, masquer (ou coder) votre source Basic pour le rendre complètement incompréhensible.

Il reste enfin le classique des classiques, « le code d'accès au programme ». Nous allons faire maintenant le tour de ces méthodes. Avant de nous lancer, il faudra vous munir d'une disquette vierge et formatée et, si vous en avez les moyens, d'un éditeur de secteurs comme Discology.
Et pourquoi pas un assembleur si le cœur vous en dit ?

```
10 code$=UPPER$("passe")
20 WHILE flag<>LEN(code$)
30 a$="":WHILE a$="":a$=UPPER$(INKEY$):WEND
40 IF a$=MID$(code$,flag+1,1) THEN flag=flag+1 ELSE flag=0
50 WEND
```

Voici, pour vous mettre en haleine, un petit programme qui trouvera sa place en début de vos listings Basic. Il vous demandera un code qui autorisera l'accès au programme.
Attention, ce programme ne protège pas le source contre la lecture mais rend son utilisation impossible aux personnes non autorisées.
Voilà, il suffit de modifier la variable CODE$ en remplacant le mot « PASSE » par celui de votre choix.
Passons à des choses plus sérieuses. Faites un CAT après avoir inséré une disquette dans le lecteur. L'ordinateur affiche alors :

```
Drive A: User 0
```

Suivi du contenu de la disquette.
Le « Drive A » indique qu'il s'agit du lecteur A (premier lecteur), «User 0 » vous indique que les fichiers affichés sont dans la zone courante 0. Cela nous emmène à penser qu'il existe d'autres zones pour sauvegarder nos fichiers.

Sous Basic, vous pouvez accèder à 16 zones utilisateur (User de 0 à 15). Pour cela, utilisez la commande :

```
ùUSER,nb
```

Avec la variable *nb* allant de 0 à 15. vous constatez grâce à la commande CAT que les fichiers de l'User 0 ne figurent pas dans l'User, 1, 2, 3... 15.

Au-delà de cette limite (15), le Basic est incapable d'accéder aux Users supérieurs. Pour cela, utilisez le poke suivant :

```
POKE &A701,nb
```

Avec la variable *nb* allant de 0 à 255. Cool, non ? Saviez-vous que ces User

ne sont pas détectés par la commande CATALOGUE de Discology ?
Autre chose, quand un fichier est effacé du catalogue d'une disquette par la commande ùERA,"fichier", celui-ci est en fait encore disponible dans le User 229 jusqu'à la prochaine sauvegarde d'un fichier sur cette même disquette.

Allez voir cet User pour vous en persuader. La récupération d'un fichier effacé devient alors un jeu d'enfant.
Si TOTO.BIN a été effacé par :

```
ùERA,"TOTO.BIN"
```

Récupérez-le avec :

```
POKE &A701,229
ùREN,"0:TOTO.BIN","TOTO.BIN"
ùUSER,0
```

Vous venez ainsi de renommer le fichier "TOTO.BIN" en le replacant en USER 0.

Jusqu'à présent, nous n'avons pas fait appel à des connaissances trop particulières. Par contre, pour la suite de notre histoire, il va falloir être attentif. Pas de frayeurs inutiles car je vais juste vous démontrer comment est stockée une ligne Basic par l'ordinateur. Ces explications seront primordiales pour la suite de notre initiation.

LES LIGNES BASIC

En chargeant un programme Basic, ou plus simple en saisissant une ligne Basic grâce à son éditeur, la mémoire de l'ordinateur subit certaines modifications. Le Basic sait comment stocker vos lignes en mémoire, nous allons les décoder. Un programme Basic commence toujours à l'adresse 368 (&170 héxadécimal).
Les deux premiers octets indiquent la longueur de la ligne (le Basic n'en n'utilise qu'un et c'est une faille que nous étudierons plus loin), les deux suivants stockent le numéro de la ligne ; vient ensuite une suite de codes (les tokens) qui sont la représentation des instructions Basic. Ex : le Token 138 correspond à l'instruction CLS. Suite à toutes ces instructions, vient se rajouter un 0 qui signifie la fin de la ligne. Alors, la deuxième ligne peut commencer, etc.

La fin du programme sera marquée par trois 0 successifs. Cela vous paraît compliqué ? Alors suivez-moi de très près.

Réinitialisez votre ordinateur et pokez les valeurs suivantes :

pour un numéro de ligne égale à 50 :

```
POKE 368+2,50
POKE 368+3,0
```

pour mettre un CLS dans cette ligne :

```
POKE 368+4,138
```

Enfin, pour marquer la fin de la ligne :

```
POKE 368+5,0
```

Reste la longueur. 2 octets pour stocker la longueur + 2 octets pour le numéro de ligne + 1 octet pour le Token et 1 dernier octet pour le 0 de fin de ligne. Cela nous fait une petite addition dont le résultat est 8. Donc :

```
POKE 368+0,8
POKE 368+1,0
```

Pour voir le résultat, faites simplement LIST.
C'est magique, non ?
Vous venez de créer une ligne 50 contenant l'instruction CLS. Il va de soi que la prochaine ligne Basic sera stockée à partir de 368+6 et cela jusqu'en fin de programme (trois 0).
Vous voulez connaître le numéro de la 33e ligne d'un programme Basic ? Pas de problème, encodez en mode direct celui-ci :

```
nbligne=33:deb=368:FOR i=1 to lbligne-1:a=PEEK(deb):deb=deb+a:NEXT:PRINT
PEEK(deb+2)+PEEK(deb+3)*256
```

il va de soi qu'il faut remplacer le « nbligne » par 33 ou toute autre valeur de votre choix. Voyons maintenant d'autres astuces sorties tout droit du sac de XOR.

LES LIGNES CACHEES DU BASIC

Pour commencer ce chapitre, saviez-vous que le Basic n'affiche les lignes (par les commandes LIST ou EDIT) qu'à partir de la ligne numéro 1 ? Le zéro, il ne connaît pas. Alors, rien ne nous empêche de lui balancer une ligne zéro pour l'embrouiller jusqu'à la moelle. La méthode est fort simple. Il suffit de poker en 370 un zéro. Vous vous souvenez qu'à cette adresse (tout comme la suivante) se trouve le numéro de la première ligne du Basic.
Voici un exemple. Tapez :

```
10 CLS
```

Faites LIST pour constater la présence de cette ligne. Puis pokez directement :

```
POKE 370,0
```

Refaites LIST. Hé ! hé ! hé ! Y'a p'us de liste. Mais attention, le programme est toujours là et vous pouvez même le lancer par RUN. Quelle classe ! Je ne le dirais jamais assez.
Voilà pour la ligne zéro : sachez tout de même qu'un éditeur de fichiers tel Discology est capable de lire ce type de ligne.
On peut cacher une ligne au sein même du programme. Pour cela, il existe encore une méthode simple mais à laquelle il fallait penser.
Reportez-vous plus haut dans les explications pour vous remémorer la structure d'une ligne Basic (longueur, numéro de ligne, etc.). Le Basic, en exécutant un programme, ne tient pas compte de la longueur donnée en début de ligne. Son repère de fin de ligne est le zéro placé en fin de ligne. La longueur est utilisée par l'éditeur Basic pour lister le programme ou les sauts (GOTO, GOSUB...). Pour cacher une ligne, il suffit de tromper le listeur Basic en lui indiquant une longueur égale à la ligne courante, plus la longueur de la ligne qui restera cachée.
Soyons clair.
Vous voulez cacher la ligne 25 de votre programme ? Il suffit de calculer sa longueur (disons 35 octets) puis celle de la ligne précédente (disons la ligne 20 qui a une longueur de 56 octets), d'additionner l'ensemble et de poker la somme de cette addition comme longueur de la ligne 20.
C'est pas clair, hein ?
Je vais vous donner un exemple. Tapez ce petit programme:

```
10 MODE 1:CLS
20 INK 0,0:INK 1,26
30 BORDER 0
35 PRINT "COUCOU"
40 END
```

Nous allons cacher la ligne 35. Calculons donc, dans un premier temps, l'adresse des octets qui renferme la longueur des lignes 30 et 35 de notre programme Basic. Pour cela, j'ajoute en fin de programme, le sous-programme que voici :

```
100 DEB=368:MODE 2
110 LL=PEEK(DEB+1)*256+PEEK(DEB+0)
120 NL=PEEK(DEB+3)*256+PEEK(DEB+2)
130 PRINT "LIGNE ";NL;" = ";LL;" OCTETS > ";
140 PRINT "ADRESSE ";DEB;DEB+1
150 CALL &BB18:DEB=DEB+LL:GOTO 110
```

Faites alors un RUN 100 (pour exécuter le programme à partir de la ligne 100) et vous saurez tout sur votre programme Basic : à savoir, la ligne 30 fait 8 octets qui sont stockés en 395-396 (même si 396 est égal à zéro), tout comme la ligne 35 qui a une longueur de 15 octets. J'ajoute les 8 au 15 pour obtenir une longueur fictive de 23. Je poke en 395 cette valeur et hop ! la ligne 30 va cacher la suivante. Essayez maintenant :

```
POKE 395,23:LIST
```

La ligne 35 a disparu !
N'ayez pas peur, lancez le programme pour vérifier que le COUCOU s'affiche, même s'il ne figure plus dans la liste (!). Vous pouvez y aller... EDIT 35, RENUM, etc. Elle ne montrera plus le bout de son nez. La ligne 35 est devenue... fantôme.

ET LES TOKENS DOCTEUR ?

Il y a très longtemps, avec mon ami Septh, j'avais concocté un petit programme intitulé « Le CLS le plus rapide

du monde ». Il était truffé de lignes automodifiant le programme Basic, donc carrément incompréhensible à première vue. Im maginez, en effet, une ligne du type :

```
OUT 1,100,10
```

Cela ne veut rien dire. Par contre, si la ligne précédente modifie le TOKEN de OUT, qui est 185, en le remplacant par 204 (TOKEN de SOUND), les choses n'auront plus la même signification. Voici un exemple :

```
10 MODE 1
15 POKE 393,204
20 OUT 1,100,10
25 POKE 393,185
30 LIST
```

Cachez les deux lignes 15 et 25 par la méthode décrite plus haut et donnez-m'en des nouvelles.

ETRANGES APPLICATIONS

Cela dit, vous ne voyez toujours pas comment il est possible, à partir de ces éléments, de tromper une personne fouinant dans vos programmes. O.K. ! je suis là pour ça.
Commençons par un petit exercice simple. Je vais vous montrer comment inclure un petit programme en langage machine dans votre source Basic. Pour ne pas compliquer les choses, j'ai choisi volontairement une routine bidon qui ne fait qu'un simple bip, équivalent de PRINT CHR$(7).

```
LD A,7
JP &BB5A
```

En désassemblant cette routine, on obtient les cinq codes hexadécimaux suivants :

```
3E,07,C3,5A,BB
```

Nous allons également créer une ligne Basic bidon qui réservera cinq octets pour stocker notre routine. Que pensez-vous de :

```
10 REM 12345
```

Cette ligne est la première de notre programme Basic. Elle commence donc en 368. La zone 368-369 est réservée pour la longueur, 370-371 pour le numéro de ligne. Le token de REM (ou de l'apostrophe le remplaçant «'») est codé avec un 192 dans l'octet 372. Vient ensuite un espace en 373 et cinq octets (de 374 à 378) qui stockent les cinq valeurs Ascii de 1, 2, 3, 4, 5 soit 49,50,51,52,53.
Vérifiez par vous-même :

```
PRINT PEEK (374) -> 49
PRINT PEEK (378) -> 53
```

C'est dans cette zone que nous allons poker notre routine assembleur. C'est également cette zone qui sera appelée pour l'exécution de la routine du bip sonore. Voici la démarche à suivre :

```
POKE 374,&3E
POKE 375,&07
POKE 376,&C3
POKE 377,&5A
POKE 378,&BB
```

Faites LIST. Vous voyez, votre ligne de REM ne ressemble plus à grand-chose. Par contre, elle renferme maintenant la petite routine. Vous pouvez sauvegarder ce programme et une éventuelle suite sans problème avec l'instruction : SAVE "TOTO"
Pour exécuter la routine, il suffit de l'appeler par :

```
CALL 374
```

Essayez, vous m'en donnerez des nouvelles. Il va de soi qu'au lieu de faire un petit bip, vous auriez pu décoder un programme (préalablement codé) ou lancer un appel à une autre routine machine...

A propos de codage/décodage, voici un petit exemple pour éclairer le principe. L'opération logique XOR (tiens donc) a la particularité de restituer une valeur à condition de l'utiliser deux fois de suite sur un même octet.
Je m'explique :
soit une valeur A et B. En appliquant A XOR B, on obtient C. A étant la valeur initiale, et B une constante de notre choix. Nous nous retrouvons donc avec deux valeurs, C et B. La valeur A étant apparemment perdue. Eh bien non, A n'est pas perdue, il suffit de faire B XOR C et le résultat est A. Voilà l'exemple pour coder une ligne de LONGS octets.

```
        ORG 374
        LD HL,ADRESSE DEBUT
        LD B,LONG
BOUC    LD A,(HL)
        XOR 123
        LD (HL),A
        INC HL
        DJNZ BOUC
        RET
```

Il va de soi que vous devriez au préalable calculer l'adresse de début de cette dernière :
Voici le code assemblé, soit 13 octets :

```
21,??,??,06,??,7E,EE,7B,77,23,10,F9,C9
```

Prenez n'importe lequel de vos programmes Basic et ajoutez-y les deux lignes suivantes :

```
1 REM 1234567890123
2 CALL 374
```

Pokez ensuite les 13 octets comme dans l'exemple précédent. Exécutez en mode direct un CALL 374 et sauvez le fichier sous un nom de votre choix. Jouez ensuite le voyeur. « Loadez » le programme et « listez »-le. Drôle de tête, non ? Exécutez-le, c'est comme si de rien n'était. Vous pourrez par exemple placer dès le début de votre programme (après les deux lignes) un ON BREAK GOTO 60000 et en ligne 60000 un CALL 374.

De là, n'importe quelle interruption du programme par la touche ESC recodera ce dernier. La classe, quoi !

Bon, c'est pas tout mais faudrait p'têt penser à boucler la boucle. Je vais donc vous donner le *nec plus ultra*, à savoir...

LE BASIC MACHINAL

Je suis assez fier de cet intertitre. Il va de soi qu'il fait référence au langage machine collé aux fesses du Basic. Donc indétectable mais, surtout, illisible par le petit frère curieux (si, si,

```
0000:  00 51 34 20 20 20 20 20 20 42 41 53 00 00 00 00   .Q4      BAS....
0010:  00 00 00 00 00 70 01 00 22 00 00 00 00 00 00 00   .....p..".......
0020:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040:  22 00 00 D0 02 07 00 0A 00 01 C0 00 12 00 14 00   ".............
0050:  01 C0 20 70 72 6F 67 20 62 69 64 6F 6E 00 07 00   .. prog bidon...
0060:  1E 00 01 C0 00 00 00 1A A8 A8 A8 A8 A8 A8 A8 A8   ................
0070:  A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8   ................
0080:  07 00 0A 00 01 C0 00 12 00 14 00 01 C0 20 70 72   ............ pr
0090:  6F 67 20 62 69 64 6F 6E 00 07 00 1E 00 01 C0 00   og bidon........
00A0:  00 00 1A A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8   ................
00B0:  A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8   ................
```

Doc 1

regardez, il est déjà en train de lire le journal par-dessus votre épaule).
Le principe d'ajouter une routine en langage machine à la suite d'un programme Basic est, comme d'habitude, fort simple. Il suffit de poker la routine machine à la suite des trois zéros finissant le programme Basic sans oublier de déclarer la nouvelle longueur du fichier Basic dans le HEADER (en-tête du fichier en français). Avant de nous lancer, voyons de quoi est constitué ce HEADER.
Creéz un fichier bidon comme suit :

```
10 '
20 ' prog bidon
30 '
```

Sauvez-le sous le nom de Q4.BAS (c'était mon appellation, vous pouvez en prendre une autre).
Lancez Discology et grâce à l'éditeur de fichier (EDITION —> MODES —> EDITION FICHIER), éditez le fichier Q4.BAS. Vous devez obtenir une page similaire au document 1 :

Le HEADER est formé des 128 premiers octets, soit les huit premières lignes affichées à l'écran.
L'adresse 0 correspond à l'USER, les 11 octets suivant, c'est le nom du fichier. Les six qui suivent ne sont pas utilisés. A l'adresse &12, nous trouvons le type de fichier (00=BASIC, 01=BASIC protégé, 02=binaire, 03=binaire protégé). Les octets &13 et &14 ne sont pas utilisés. En &15 et &16, on trouve l'adresse d'implantation du programme (dans notre exemple 70 et 01 soit &170 ou 368 décimal). L'octet &17 ne sert à rien. &18 couplé à &19 donne la longueur du fichier (ces octets nous intéressent). Dans notre exemple, le fichier a une longueur de &0022 octets (soit 34 octets en décimal).
On continue dans la joie et la négresse, &1A et &1B donnent l'adresse d'exécution. Cette adresse est à zéro pour les fichiers Basic. Les valeurs allant de &1C à &3F ne sont pas utilisées. Viennent, enfin, &40 et &41 qui redonnent pour la deuxième fois la longueur du fichier. Attention, ces adresses sont prioritaires par rapport à &18 et &19.
L'adresse &42 ne sert à rien ; par contre, en &43 et &44 nous trouvons le cheaksum de l'ensemble du HEADER, à savoir la somme de l'addition de tous les octets allant de la zone &0000 à &0042. On nomme également cette addition CSH. Le reste des octets (&45 à &7F) peut être rempli avec n'importe quoi car il s'agit d'une zone utilisée comme buffer lors de la sauvegarde. Ouf ! c'est fini.

Pour ne pas être trop lourd, nous allons implanter un tout petit programme machine :

```
LD A,7
JP &BB5A
```

Soit en codes machine les cinq octets suivants :

```
&3E,&07,&C3,&5A,&BB
```

Pour cela, on doit augmenter de cinq les zones &18-&19 ainsi que &40-&41, sans oublier d'augmenter de &0A ou 10 décimal (deux fois 5) la zone de CSH (&43-&44).

Il ne restera alors plus qu'à placer les cinq octets à la suite des trois zéros de fin du fichier Basic (à partir de l'adresse &A0) voir Doc 2.
Revenez maintenant sous Basic. Rechargez le fichier Q4. Listez-le. Apparemment rien n'a changé. Nous avons vu dans le HEADER que la longueur initiale du fichier était de &22 soit 34 (en décimal).

Sachant qu'un fichier Basic commence en &170 ou 368 nous en déduisons facilement que notre routine est installée en &192 ou 402 décimal. Vérifiez par vous-même :

```
FOR I=402 TO 406:PRINT
HEX$(PEEK(I),2):NEXT
```

Si vous avez encore des doutes, exécutez donc la fameuse routine assembleur par :

```
CALL 402
```

Voilà, nous devons arrêter les frais pour aujourd'hui. Il va de soi que l'appel à la routine peut être (doit être !) dans le source Basic (une ligne cachée, par exemple). Pour cela, insérez dès le début un CALL bidon qu'il suffira de réajuster une fois les opérations terminées.

Avant de finir avec ce chapitre et comme bonus d'adieu, je vous parlais plus haut (dans le HEADER) du type de fichier. Sachez que tout comme un fichier Basic protégé (type 1), un fichier binaire protégé (type 3) ne pourra être loadé. Faites donc le test pour vous en persuader.

DES QUESTIONS ?

A partir de là, vous vous posez sans doute la question suivante : « Il est bien gentil le p'tit Poum, il nous file plein de trucs pour cacher des instructions ou rendre le programme illisible. Mais rien n'empêchera le premier venu de prendre Discology et de copier mon programme. Alors, j'ai fais tout ça pour du beurre ? »

Bien sûr que non, car il suffit ensuite de rendre le programme incopiable (lisez plus bas) et cacher tous les tests par une ou un mélange des méthodes données ici même.

C'EST FINI !

Si ces quelques lignes vous ont plu ; si vous commencez déjà à taper du pied dans l'espoir d'en savoir plus ; si vous faites sauter le standard de la rédaction ; si les traiteaux de mon bureau cèdent sous la pression de votre courrier ; si les gros chèques arrivent sur mon compte en suisse (n° 126584654294)... « IF ces Si sont TRUE THEN » je vous promets de reprendre mon courage en m'aventurant dans l'antre du grand Xor, histoire de vous narrer d'autres contes machiavéliques du genre.

Poum qui remercie Xor
(il va de soi)

```
0000:  00 51 34 20 20 20 20 20 20 42 41 53 00 00 00 00   .Q4      BAS....
0010:  00 00 00 00 00 70 01 00 27 00 00 00 00 00 00 00   .....p..'.......
0020:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040:  27 00 00 DA 02 07 00 0A 00 01 C0 00 12 00 14 00   '...............
0050:  01 C0 20 70 72 6F 67 20 62 69 64 6F 6E 00 07 00   .. prog bidon...
0060:  1E 00 01 C0 00 00 00 1A A8 A8 A8 A8 A8 A8 A8 A8   ................
0070:  A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8   ................
0080:  07 00 0A 00 01 C0 00 12 00 14 00 01 C0 20 70 72   ............ pr
0090:  6F 67 20 62 69 64 6F 6E 00 07 00 1E 00 01 C0 00   og bidon........
00A0:  00 00 3E 07 C3 5A BB A8 A8 A8 A8 A8 A8 A8 A8 A8   ................
00B0:  A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8 A8   ................
```

Doc 2

PROTECTIONS LOGICIELLES (II)

Vous voulez ma mort ou quoi ? Je viens de réaliser que, depuis des années, je vis dans l'illusion... L'illusion de croire en des lecteurs qui me voudraient du bien. Que nennie. Vous avez été nombreux (trop nombreux) à me demander une nouvelle rencontre avec Xor. Pourrais-je vous le pardonner un jour ?

Vous ne comprenez pas pourquoi cela m'est si pénible ? Si vous étiez passé un jour dans l'antre du maître, si vous aviez vu tous les monstres errant dans ses dongeons, si vous aviez flippé la mort en entendant les bruits étranges venant du fin fond des puits, si... et si..., vous comprendriez que cela me coûte de préparer ces deux pages, car à chaque moment passé avec Xor j'ai risqué ma vie. Mais, encore une fois, je m'en suis sorti indemne, et c'est tout ce qui compte pour le moment. Voici les infos recueillies lors d'une froide nuit de février, quelque part dans un petit village du Périgord.

ELLE RAP ? RAPELLE

Je dois rectifier deux tout petits détails qui ont pu vous poser un problème lors de notre dernier entretien.

Primo, je vous disais qu'en renommant un fichier sous Disco, vous pouviez enlever la protection contre l'écriture. Il est vrai que cela est... faux ! Seule la protection contre l'écriture pourra ainsi être supprimée.

Secundo, le petit programme assembleur (l'encadré de la page 32), qui devrait masquer un source Basic ne doit pas être assemblé, et encore moins appelé à l'adresse 374, mais en 398, qui est l'adresse de la troisième ligne Basic. Cela dit, je vous rappelle qu'il faudra, pour faire une bonne protection, mélanger les différentes méthodes afin de planter complètement les petits curieux.

Par exemple, dans le cas de ce masquage, il faut créer une ligne 0 qui cachera en plus la suivante. Le reste des lignes étant masqué par le programme assembleur, le source Basic ne ressemblera plus à grand-chose, mais restera exécutable.

LES FICHIERS FANTOMES

Ce que l'on nomme un fichier fantôme est un fichier fictif, de très grande taille sur la disquette, et qui empêche toute sauvegarde sur cette dernière.

Il n'est pas question de vous parler de ce type de fichier, mais plutôt de vous montrer comment cacher des fichiers pour que personne ne puisse retrouver leur trace.

Il est évident que la méthode la plus simple consiste à changer d'User, mais cela reste insuffisant.

On peut planquer un fichier de deux autres façons. Soit en masquant son nom, soit en déplaçant le fichier dans un nouveau catalogue (un mélange des trois méthodes sera fortement conseillé).

MASQUER UN NOM DE FICHIER

Vous savez que certains fichiers n'apparaissent pas au catalogue de l'Amsdos. Pour parvenir à ce type de résultat, deux méthodes vous sont proposées. Soit vous renommez les fichiers à l'aide de Disco en leur ajoutant l'option [S], soit vous tripatouillez manuellement le catalogue. Pour cela, placez-vous à l'aide de Disco sur le catalogue de la disquette (piste 0, premier secteur). Trouvez le nom du fichier et ajoutez la valeur 128 (&80) à l'octet formant le deuxième caractère de l'extension. Je vous vois déjà grincer des dents. C'est pourtant simple, prenons un fichier portant le nom de TOTO.BAS. Le deuxième caractère de l'extension est le A de BAS. Son code Ascii est de 65 décimal, soit &41 hexa. En ajoutant 128 (bits 7 à 1), on obtient 193 décimal soit &C1. Remplacez le &41 par &C1, et votre fichier n'apparaît plus au catalogue. Sachez tout de même qu'en faisant la même opération sur le troisième caractère de l'extension (S du BAS, dans notre exemple), le fichier passera en protection contre l'écriture. Vous comprenez facilement qu'un tel fichier reste toujours visible sous Disco, car rien n'a changé au niveau des huit premiers caractères qui forment le nom de ce dernier. Pour cela, sachez que ce type de masquage (+128 soit bits 7 à 1) est entièrement applicable aux 12 caractères du nom du fichier dans le catalogue. Ainsi, je vous défie de reconnaître le nom d'un fichier même sous Disco. Tout ça est clair. Le nom d'un fichier ne ressemble plus à grand-chose sous l'éditeur, mais il reste tout de même un gros problème, la commande catalogue de Disco sait lire ce type de nom. Argh ! Pour cela, n'hésitez pas à modifier l'User de vos fichiers (au-dessus de 15) et/ou lisez le chapitre qui suit.

AJOUT D'UN CATALOGUE

Normalement, l'Amsdos travaille avec un seul catalogue. Vous pouvez néanmoins en ajouter plusieurs. Une seule contrainte à cela, notez qu'un fichier placé dans un catalogue s'enregistrera à la suite de la piste du catalogue. Ce qui, en d'autres termes, veut dire qu'un catalogue placé en piste 36 ne laissera que très peu de place pour la sauvegarde des fichiers. De même, un catalogue en piste 3 aura de fortes chances d'écraser les fichiers enregistrés dans le catalogue de la piste 0.
Pour comprendre le principe, suivez cet exemple. Formatez une disquette. Tapotez un petit programme Basic et, avant de le sauvegarder, encodez cette ligne :
POKE &A8A8,255:POKE &A89D,N
La variable N représente la piste sur laquelle sera placée le catalogue (disons 10). Faites votre sauvegarde (SAVE "TOTO.BAS"), réinitialisez l'ordinateur (CONTROL+SHIFT+ESC) et faites CAT. Vous constatez qu'il n'y a pas de TOTO apparent sur la disquette. Si vous voulez récupérer ce fichier, refaites la ligne de POKE et lancez ou LOADez le fichier. Tout va bien, c'est un miracle !
Faites tout de même attention. Cette méthode est assez dangereuse pour les données de la disquette. N'importe quelle opération sur le catalogue normal peut modifier les données du second catalogue. Vous pensez bien, l'Amsdos se fiche royalement de votre nouveau directory. Alors prudence, et faites des copies de sauvegarde au lieu de travailler sur vos originaux.
Vous en voulez une bonne ? La copie fichier de Disco est complètement aux fraises avec ce type de fichiers (qui plus est, sans messages d'erreur de sa part).
Avant de passer à la suite, je vous dois quelques explications concernant ces deux POKE.

En &A8A8 se trouve un zéro. En le remplaçant par 255, vous indiquez à l'Amsdos de ne pas réinitialiser les paramètres disque (n° de piste du catalogue, nombre de secteurs par piste, taille du catalogue...).
L'octet &A89d indique la piste sur laquelle se trouve la catalogue. C'est cet octet qui prendra la valeur de votre nouvelle piste.
Nous sommes avec Xor persuadés qu'il est possible de changer par la même méthode la taille du catalogue. Sachant que Disco ne reconnaît qu'un seul type de directory, vous comprenez qu'il sera complètement perdu lors de ces copies. Malheureusement, nos tentatives n'ont pas donné les résultats escomptés, donc histoire à suivre...

PROTECTIONS PHYSIQUES

Le nec plus ultra pour protéger une disquette est de la rendre physiquement incopiable. Croyez-moi, cela est une très longue histoire pas facile à narrer. En trafiquant une disquette, vous aurez peu de risques de planter Disco, car les programmeurs de ce dernier ont assuré un max et ont fait un sacré boulot. Promis, dans le prochain numéro, on vous donne une routine pour planter Disco, mais avant tout, il faudra mettre quelques termes au point.

Premièrement, vous devez connaître les méthodes de formatage sur le bout des doigts. Pour cela, relisez l'article de Sined paru en janvier 91 (n° 33 d'*Amstrad Cent Pour Cent*, encore disponible). Deuxièmement, vous devez tout savoir sur les formats de pistes.

LES FORMATS

En formatant une disquette (DATA) vous obtenez un certain nombre de pistes (souvent de 0 à 39, soit 40 pistes). Chaque piste est formatée en 9 secteurs de 512 octets. Faites le compte :
40*9*512=184320
Sachant qu'un kilo-octet représente 1024 octets on en déduit par une simple division que la disquette est formatée à 180 Ko. 2 kilos sont réservés pour le catalogue et il nous reste les fameux 178 kilos disponibles. En oubliant l'Amsdos et le catalogue vous pouvez récupérer les deux kilos du CAT et formater la disquette jusqu'en piste 41 ce qui vous donnera une disquette de :
42*9*512=184.5 Ko

Sachez, enfin, qu'un secteur peut avoir plus ou moins d'octets que 512. On désigne alors les secteurs par leur taille (vous avez entendu parler des secteurs taille 6 ?).
Pour cela, appliquez la formule suivante :
Nb d'octets par secteur = 128 * 2 puissance TAILLE.

Ainsi, notre format classique donne des secteurs de taille 2. Voici un petit tableau résumant les différentes tailles de secteurs :

nombre d'octets par secteur	taille
128	0
256	1
512	2
1024	3
2048	4
4096	5
8192	6

Plus vous poussez la taille des secteurs, plus l'écriture de ces derniers devient difficile. Pensez donc, une disquette avec 42 secteurs de taille 6 devrait donner 42*8192 soit une disquette de 336 kilos par face. Ne vous faites pas d'illusions, cela est impossible, car même en programmant comme un dieu, seul 4 kilos des secteurs taille 6 seront utilisables.
Il me reste à vous dire une chose ; très peu de gens en France ont une routine permettant de créer de tels secteurs, encore plus rare sont ceux qui peuvent écrire dans ces secteurs et quasiment personne ne pourra les lire.

Pour la petite histoire, sachez que Xyphos Phantasy de Feffesse était formaté avec des pistes de 5 secteurs de taille 2 (5*512 octets), 1 de taille 3 (1*1024) et 1 de taille 4 (1*2048), ce qui sur 42 pistes (0 à 41) représentait 231 kilos.
Je ne vous parle pas du nombre de disquettes retournées à l'éditeur, car même les gros dupliqueurs utilisés par les boîtes d'édition se sont sentis mal.

Poum,
qui ne remercie pas ses lecteurs

PROTECTIONS LOGICIELLES (III)

Blindage Software présente : le Retour de la revanche du fils de la protection logicielle II : The Disk Quete. Nous vous avons parlé, en long, en large et en travers, des différents moyens de plomber un logiciel. Voici, pour poursuivre et finir, les quelques derniers petits points à connaître, histoire de totalement dégoûter les pirates en culotte courte.

Quand Xor a pris Poum par la main pour l'emmener au fin fond de la Grotte du Savoir, notre héros s'est fait porter pâle. Il en a eu la courante de sa vie. On ne dit plus blanc comme un linge, on dit blanc comme la tronche de Poum lorsqu'il s'est trouvé face à l'entrée de la Grotte de Xor. C'est plus long, mais on ne fait pas mieux. Moi, c'est avec deux litres du meilleur whisky dans le ventre, rond comme un manche de pelle, que Xor me fit rouler dans son antre. Lorsque les éléphants roses me saluèrent en allant vaquer à leurs occupations quotidiennes, il était trop tard. Nous marchions déjà dans le couloir menant au Puits de la Connaissance, aveuglés par l'éclat des torches atomiques, seules armes efficaces contre les puissants mais mortels Copyrats...
Oh ! pardon... reprenons.

C'est bien beau d'empêcher un voyeur de venir tripoter votre code, mais si cet hurluberlu peut copier la disquette à volonté, autant ne pas se casser à plomber. Pour être sûr que le programme ne sera pas violé, il faut le protéger autant physiquement que logiciellement.
Mais si, c'est possible ! J'en vois des petits qui rigolent dans le fond. Nous allons avant tout nous prendre la tête avec de nouvelles connaissances, histoire de savoir de quoi on parle. Ensuite, nous pourrons bosser tranquille. Comment voulez-vous savoir comment travaille le contrôleur de disquette si vous ne connaissez pas le format exact d'une piste après un formatage courant ?
Il faut parfaitement assimiler ses capacités pour pouvoir le comprendre et le planter, bien évidemment. Contentons-nous pour le moment de le regarder faire. Le reste suivra...

FORMATION DE CROISIERE

Comme vous pouvez vous en douter, le FDC, pour retrouver les données qu'il éparpille sur une piste, range tout minutieusement lors du formatage. Cette forme de rangement permet non seulement d'augmenter la fiabilité des données, car des sommes de contrôles sont calculées pour chacun des secteurs, mais elle permet aussi de partitionner la piste pour ne pas gérer la totalité des données simultanément. Une piste mesurant 6 144 octets, il faudrait en mémoire un buffer de 2 x 6 144 octets pour la lecture et l'écriture, ce qui serait bien trop important

pour les machines de petite capacité comme le CPC. Les concepteurs du contrôleur FDC 765 ont donc implémenté une sectorisation qui permet d'accéder à des secteurs de tailles paramétrables. Il existe ainsi six tailles de secteurs différentes numérotées de 0 à 5. Pour calculer la taille des secteurs en fonction de leur numéro, il suffit d'appliquer le calcul suivant :

Taille = 128 x 2Numéro.

Où Numéro est le numéro du secteur comme spécifié ci-dessus et Taille est la taille du secteur en octet. Comme vous le voyez, une valeur de 0 donnera un secteur de longueur 128 et un secteur de type 5 vous fournira une zone de 4 096 données.

Mais ce n'est pas tout, comme des tas d'octets servent à délimiter les secteurs, à donner des informations sur la piste en cours, à préparer des synchronisations entre le contrôleur et la disquette, nous ne disposons pas effectivement de 6 144 octets consacrés aux données. Selon la taille des secteurs utilisés et selon la longueur qu'il est possible de fixer aux zones de synchro, plus ou moins de secteurs identiques ou non viendront se loger sur la piste. Il est ainsi possible, par exemple, de loger sur une piste un secteur de type 5 (4 096) suivi d'un de type 2 (512) et un de type 1 (256). A vous de calculer la bonne place pour ces secteurs et d'organiser le formatage en fonction.

A titre indicatif, voici un bref descriptif de la manière dont sont placées les données sur la piste pour que le FDC s'y retrouve. Nous n'entrerons pas dans les détails, mais nous montrerons simplement pourquoi tant de place est perdue en dehors de la taille prise effectivement par les secteurs. Voici donc le descriptif d'une piste d'un trou d'index à l'autre (qui est le même, mais la piste est ronde), donc en piste :

- 80 octets de Gap 4A (valeur 4E) ;
- 12 octets de synchronisation (valeur 00) ;
- 3 octets d'Index Adress Mark (???)
- 1 octet (valeur FC) ;
- 50 octets de Gap 1 (valeur 4E) ;

——— Début du secteur ———

- 12 octets de synchronisation (valeur 00) ;
- 3 octets d'Index Adress Mark (???) ;
- 1 octet (valeur FE) ;
- 1 octet (numéro de piste) ;
- 1 octet (numéro de tête) ;
- 1 octet (numéro de secteur) ;
- 1 octet (taille du secteur) ;
- 2 octets de CRC (checksum des 8 dernières valeurs) ;
- 22 octets de Gap 2 (valeur 4E) ;
- 12 octets de synchronisation (valeur 00) ;
- 3 octets d'Index Adress Mark (???) ;
- 1 octet (valeur FB) ;
- X octets de données selon la taille du secteur ;
- 2 octets de CRC (checksum des données et des 4 octets les précédant) ;
- Z octets de Gap 3 (valeur 4E).

——— Fin de secteur ———

- N autres secteurs...
- W octets résiduels de Gap 4B comblant le vide entre la fin et le début de piste (valeur 4E).

GAP #3 : PERDRE POUR MIEUX GAGNER

Si nous partons à la chasse aux octets, on s'aperçoit vite qu'un secteur prendra effectivement 62 octets de descripteurs et de CRC, plus le nombre d'octets de données, plus la taille du GAP #3 choisie. Le Gap #3 est très important, car il permet de séparer physiquement deux secteurs. Il faut savoir que cette zone a été réalisée dans le but de garder une compatibilité entre divers lecteurs. Il suffit qu'un de ces joujoux tourne un tout petit peu plus vite qu'un autre pour qu'une écriture de secteur vienne écraser l'ID du suivant. Pourquoi faire dans l'instable alors qu'il est possible de toucher une fiabilité relativement idéale.

C'est pour cela que des valeurs de sécurité ont été instaurées. Lorsque vous réalisez le format qui tue avec 2 méga-octets par piste et des GAP #3 de 0,5 octets, dites-vous bien qu'il y a de fortes chances pour que le CPC de M. Bolocksbroken (Belgique) ne lise et n'écrive que des pistes irrécupérables. Alors, pensons aux copains, jouons la carte de la sécurité au détriment de quelques octets perdus.

Voici une théorie simple permettant de calculer un GAP #3. Vous choisissez votre taille de secteurs, par exemple 3 (1 024 octets). Vous lui ajoutez le header par secteur, soit 62. Cela nous donne 1 086 octets effectifs par secteur. Sachant qu'une piste mesure en moyenne (selon la vitesse théorique des lecteurs) 6 144 octets auxquels nous retranchons 146 octets de header piste, il nous reste 5 998 octets réellement exploitables. Nous divisons 5 998 par la taille d'un secteur (1 086) ce qui nous donne 5 secteurs par piste. Pour calculer la taille du GAP #3, facile. Il suffit de diviser en 5 la place laissée libre par la soustraction de l'espace total disponible, moins la place prise par les secteurs. Elle est de (5 998 - 5 430)/5 soit dans notre cas, d'environ 113,6. Comme ce GAP #3 est de loin supérieur à 80, nous ne risquons rien et pouvons utiliser cette dernière valeur tout en sachant que ce format sera certainement ultra fiable. Le reste du libre ira dans le GAP 4B. Pourquoi prendre 80 ? Je présume que cette valeur n'est pas à négliger, car elle a été choisie par le constructeur comme valeur initiale de séparation entre le trou d'index de piste et la première synchro (GAP 4A). Dans ce cas, mieux vaut présumer que se planter. Si vous vous retrouvez, dans certains cas de figure, avec des GAP #3 très courts (de loin inférieurs à 80), je vous conseille de répartir une partie du dernier GAP #3 sur les autres. Cela augmentera quelque peu les espaces intersecteurs, ce qui n'est pas un mal, sans pour autant trop tirer sur la piste. Notez que si un petit débordement de GAP #3 du dernier secteur apparaît en début de piste, cela ne sera pas gênant, car les octets écrits auront la même valeur dans les deux zones (le remplissage des deux GAP se fait avec 4E). Dans le cas où le GAP #3 est paisible (supérieur à 80), laissez la fin de piste telle qu'elle est, cela ne gênera personne.

Pour en finir avec le GAP #3, sachez qu'il en existe de deux valeurs différentes : une de formatage permettant de séparer physiquement les secteurs, et une d'écriture permettant de recouvrir des données de valeur 4E dans le cas où la disquette aurait tourné trop vite et où des octets différents seraient visibles après le CRC. Il n'y a pas de réel rapport calculable entre les deux valeurs, si ce n'est que pour des valeurs de formatage aux alentours de 80, prenez la moitié pour l'écriture. Plus la valeur au formatage est petite, plus la valeur à l'écriture doit se restreindre, sous peine d'écraser l'ID suivant si elle est trop gonflée. (Exemple : GAP #3 de 50 au formatage donnerait un GAP #3 d'environ 12 à l'écriture.)

Tout ce que je peux vous dire c'est que la meilleure protection pour CPC dont vous puissiez rêver est celle réalisée sur un ORIC Atmos. En effet, le FDC 765 ne peut pas écrire dans la première zone GAP 4A, alors que l'Oric sait le faire. Ecrivez votre prénom à cet endroit et allez le lire avec une lecture piste sur CPC. Je défie n'importe quel FDC 765 de faire de même (c'est théoriquement impossible). La protection de rêve, quoique plus chère car il faut acheter un Oric.

Même Disco 145.3 ne sait pas faire ce truc-là. Diabolique... Voilà pour les conseils. On attaque la suite.

LE PRET-A-PORTER

Des listings sont fournis dans ces pages pour vous aider à réaliser le formatage ou la protection nec plus ultra. En voici un rapide descriptif, car nous nous sommes quelque peu trop étendus sur le premier chapitre (ND Sined : pardon à Alexe, Poum, Gambas, Fifou, Agnès, moi, lui, vous et à tous ceux qui s'en foutent).

La routine de formatage écrite par Valère est un petit bijou parfaitement orienté protection. Bravo camarade. Son utilisation est ultra simple et en voici le descriptif : à titre d'essai, nous allons formater une disquette au format DATA.

Le programme vous demande avant tout la taille des secteurs que vous désirez installer par piste. Vous saisissez ensuite 9 à la question portant sur le nombre des secteurs, 80 pour le Gap #3 (valeur par défaut), et enfin &E5 pour l'octet de formatage. A ce moment apparaît une grille de saisie demandant, pour chaque secteur physique numéroté de 1 à 9, son numéro suivi de sa taille. Voici ce qu'il faut répondre :

```
No  1 - C1/ 2
No  2 - C6/ 2
No  3 - C2/ 2
No  4 - C7/ 2
No  5 - C3/ 2
No  6 - C8/ 2
No  7 - C4/ 2
No  8 - C9/ 2
No  9 - C5/ 2
```

```
100 'COPYRIGHT AMSTRAD 100 %
110 ADR= 32768:FOR I=0 TO 25
120 FOR J=1 TO 8:READ A$
130 A=VAL("&"+A$)
140 B=(B+I+A*J) AND 255
150 POKE ADR+I*8+J-1,A:NEXT J
160 READ B$:IF B=VAL("&"+B$) THEN 180
170 PRINT "ERREUR EN ";1000+I*10:STOP
180 NEXT I:SAVE"ECRISCT.BIN",B,ADR,202
190 '
1000 DATA F3,01,7E,FA,3E,01,ED,79,D6
1010 DATA 06,02,21,00,00,2B,7C,B5,59
1020 DATA 20,FB,10,F9,3E,0F,CD,9C,9E
1030 DATA 80,3E,00,CD,9C,80,DD,7E,ED
1040 DATA 08,CD,9C,80,ED,78,F2,24,B2
1050 DATA 80,21,C6,80,3E,08,CD,9C,CF
1060 DATA 80,CD,7D,80,21,C6,80,CB,B1
1070 DATA 6E,28,F1,DD,66,01,DD,6E,6D
1080 DATA 00,3E,45,CD,9C,80,3E,00,EA
1090 DATA CD,9C,80,DD,7E,08,CD,9C,4C
1100 DATA 80,3E,00,CD,9C,80,DD,7E,D3
1110 DATA 06,CD,9C,80,DD,7E,02,CD,5A
1120 DATA 9C,80,DD,7E,04,CD,9C,80,0B
1130 DATA 3E,2A,CD,9C,80,3E,FF,CD,31
1140 DATA 9C,80,CD,BC,80,21,C6,80,44
1150 DATA CD,7D,80,FB,C9,ED,78,FE,A2
1160 DATA C0,38,FA,0C,ED,78,77,0D,8A
1170 DATA 23,3E,05,3D,20,FD,ED,78,7D
1180 DATA E6,10,20,E9,3A,C7,80,E6,93
1190 DATA 04,C0,37,C9,01,7E,FB,F5,F6
1200 DATA ED,78,87,30,FB,87,30,02,39
1210 DATA F1,C9,F1,0C,ED,79,0D,3E,29
1220 DATA 05,00,3D,20,FC,C9,0C,7E,FB
1230 DATA ED,79,0D,23,ED,78,F2,BC,34
1240 DATA 80,E6,20,20,F1,C9,00,00,8B
1250 DATA 00,00,00,00,00,00,00,00,53
```

```
10 'FORMATEUR HARD.
20 '
30 '(c) 1992 PIM PAM POUM Software.
40 '
50 MODE 1:INK 0,0:BORDER 0:PAPER 0
60 INK 1,26:PEN 1
70 MEMORY &4000-1
80 LOAD"FORMAT.BIN",&8000
90 DIM SECTEUR(32),TAILLE(32)
100 INPUT"TAILLE DES SECTEURS  :",TDS
110 INPUT"SECTEURS PAR PISTE   :",SPP
120 INPUT"GAP #3               :",GAP
130 INPUT"OCTET DE REMPLISSAGE :",ODR
140 CLS
150 IF SPP<0 OR SPP>32 THEN GOTO 100
160 LOCATE 13,1:PRINT"SECTEUR / TAILLE"
170 X=1:Y=4:FOR Z=1 TO SPP
180 LOCATE X,Y
190 PRINT"No";Z:LOCATE X+6,Y:PRINT"- ";H
EX$(SECTEUR(Z));" / ";HEX$(TAILLE(Z))
200 IF Y<19 THEN Y=Y+1 ELSE Y=4:X=24
210 NEXT Z
220 WINDOW#0,1,40,22,25
230 CLS:INPUT"NUMERO DE SECTEUR :",NUMSE
CT
240 IF NUMSECT<0 OR NUMSECT>SPP THEN 230
250 IF NUMSECT=0 THEN 330
260 CLS:PRINT">";HEX$(SECTEUR(NUMSECT));
",";HEX$(TAILLE(NUMSECT))
270 PRINT CHR$(11);CHR$(9);:INPUT"",S,T
280 IF S<0 OR S>255 OR T<0 OR T>255 THEN
 260
290 SECTEUR(NUMSECT)=S:TAILLE(NUMSECT)=T
300 IF NUMSECT>16 THEN X=32:Y=NUMSECT-13
 ELSE X=9:Y=NUMSECT+3
310 WINDOW#0,1,40,1,25:LOCATE X,Y:PRINT
HEX$(S);" / ";HEX$(T);"  "
320 GOTO 220
330 MODE 1
340 INPUT"DE LA PISTE :",PIST1
350 INPUT"A LA PISTE  :",PIST2
360 IF PIST1<0 OR PIST1>42 OR PIST2<0 OR
 PIST2>42 THEN GOTO 330
370 PRINT:PRINT"ENTREZ LE DISK A FORMATE
R PUIS":PRINT"APPUYEZ SUR UNE TOUCHE."
380 CALL &BB06
390 BUFFER=&4000
400 FOR Z=1 TO SPP
410 POKE BUFFER,PIST1
420 POKE BUFFER+1,0
430 POKE BUFFER+2,SECTEUR(Z)
440 POKE BUFFER+3,TAILLE(Z)
450 BUFFER=BUFFER+4
460 NEXT Z
470 BUFFER=&4000
480 CALL &8000,PIST1,TDS,SPP,GAP,ODR,BUF
FER
490 IF PIST1=PIST2 THEN OUT &FA7E,0:CLS:
GOTO 100
500 PIST1=PIST1+1
510 GOTO 390
```

```
100 'COPYRIGHT AMSTRAD 100 %
110 ADR= 32768:FOR I=0 TO 25
120 FOR J=1 TO 8:READ A$
130 A=VAL("&"+A$)
140 B=(B+I+A*J) AND 255
150 POKE ADR+I*8+J-1,A:NEXT J
160 READ B$:IF B=VAL("&"+B$) THEN 180
170 PRINT "ERREUR EN ";1000+I*10:STOP
180 NEXT I:SAVE"LIRESECT.BIN",B,ADR,202
190 '
1000 DATA F3,01,7E,FA,3E,01,ED,79,D6
1010 DATA 06,02,21,00,00,2B,7C,B5,59
1020 DATA 20,FB,10,F9,3E,0F,CD,9C,9E
1030 DATA 80,3E,00,CD,9C,80,DD,7E,ED
1040 DATA 08,CD,9C,80,ED,78,F2,24,B2
1050 DATA 80,21,C6,80,3E,08,CD,9C,CF
1060 DATA 80,CD,7D,80,21,C6,80,CB,B1
1070 DATA 6E,28,F1,DD,66,01,DD,6E,6D
1080 DATA 00,3E,46,CD,9C,80,3E,00,ED
1090 DATA CD,9C,80,DD,7E,08,CD,9C,4F
1100 DATA 80,3E,00,CD,9C,80,DD,7E,D6
1110 DATA 06,CD,9C,80,DD,7E,02,CD,5D
1120 DATA 9C,80,DD,7E,04,CD,9C,80,0E
1130 DATA 3E,2A,CD,9C,80,3E,FF,CD,34
1140 DATA 9C,80,CD,BC,80,21,C6,80,47
1150 DATA CD,7D,80,FB,C9,ED,78,FE,A5
1160 DATA C0,38,FA,0C,ED,78,77,0D,8D
1170 DATA 23,3E,05,3D,20,FD,ED,78,80
1180 DATA E6,10,20,E9,3A,C7,80,E6,96
1190 DATA 04,C0,37,C9,01,7E,FB,F5,F9
1200 DATA ED,78,87,30,FB,87,30,02,3C
1210 DATA F1,C9,F1,0C,ED,79,0D,3E,2C
1220 DATA 05,00,3D,20,FC,C9,0C,ED,76
1230 DATA 78,77,0D,23,ED,78,F2,BC,36
1240 DATA 80,E6,20,20,F1,C9,00,00,8D
1250 DATA 00,00,00,00,00,00,00,00,55
```

```
100 'COPYRIGHT AMSTRAD 100 %
110 ADR= 32768:FOR I=0 TO 23
120 FOR J=1 TO 8:READ A$
130 A=VAL("&"+A$)
140 B=(B+I+A*J) AND 255
150 POKE ADR+I*8+J-1,A:NEXT J
160 READ B$:IF B=VAL("&"+B$) THEN 180
170 PRINT "ERREUR EN ";1000+I*10:STOP
180 NEXT I:SAVE"FORMAT.BIN",B,ADR,184
190 '
1000 DATA F3,01,7E,FA,3E,01,ED,79,D6
1010 DATA 06,02,21,00,00,2B,7C,B5,59
1020 DATA 20,FB,10,F9,3E,0F,CD,8D,26
1030 DATA 80,3E,00,CD,8D,80,DD,7E,2A
1040 DATA 0A,CD,8D,80,ED,78,F2,24,C4
1050 DATA 80,21,B7,80,3E,08,CD,8D,3C
1060 DATA 80,CD,6E,80,21,B7,80,CB,97
1070 DATA 6E,28,F1,DD,66,01,DD,6E,53
1080 DATA 00,3E,4D,CD,8D,80,3E,00,9D
1090 DATA CD,8D,80,DD,7E,08,CD,8D,69
1100 DATA 80,DD,7E,06,CD,8D,80,DD,3C
1110 DATA 7E,04,CD,8D,80,DD,7E,02,E5
1120 DATA CD,8D,80,CD,AD,80,21,B7,E0
1130 DATA 80,CD,6E,80,FB,C9,ED,78,84
1140 DATA FE,C0,38,FA,0C,ED,78,77,CC
1150 DATA 0D,23,3E,05,3D,20,FD,ED,A9
1160 DATA 78,E6,10,20,E9,3A,B8,80,0E
1170 DATA E6,04,C0,37,C9,01,7E,FB,DD
1180 DATA F5,ED,78,87,30,FB,87,30,C3
1190 DATA 02,F1,C9,F1,0C,ED,79,0D,DF
1200 DATA 3E,05,00,3D,20,FC,C9,0C,22
1210 DATA 7E,ED,79,0D,23,ED,78,F2,D6
1220 DATA AD,80,E6,20,20,F1,C9,00,2A
1230 DATA 00,00,00,00,00,00,00,00,E2
```

Certains se demanderont alors pourquoi les noms de secteurs ne sont pas donnés dans l'ordre ? C'est ici qu'entrent les notions d'ordres logique et physique. L'ordre physique, représenté de 1 à 9, est l'ordre dans lequel seront rangés les secteurs sur la piste. L'ordre logique est celui dans lequel seront chaînés les secteurs et dans lequel ils seront aussi appelés. Cela est dû à ce qu'on nomme communément l'entrelacement. Après que le contrôleur a lu un secteur, il lui faut un petit moment avant de se recalibrer et d'être prêt à se resynchroniser sur le secteur suivant. Ce temps est malheureusement plus long que le GAP #3, mais plus court qu'un secteur. Ainsi, après avoir lu le secteur &C1, il se retrouve prêt à relire des données lorsque la tête se trouve au milieu du secteur &C6. Le secteur suivant rencontré sera le bon au niveau logique, soit le &C2. Si cet entrelacement n'avait pas été installé, il aurait fallu un tour complet avant que la tête ne repasse alors sur le secteur &C2 à la place de &C6. Je pense que vous avez suivi. En fait, il suffit de laisser un secteur physique entre deux logiques pour ne pas perdre de vitesse. Essayez donc ce format sans entrelacement et vous comprendrez vite.

Saisissez la piste de départ (zéro pour la première), la piste d'arrivée (42 pour la dernière) et c'est parti. Attention ! n'oubliez surtout pas de mettre une disquette quelconque dans le lecteur, et au besoin, protégez Format avant qu'il ne se volatilise.

Nous allons maintenant voir un petit truc en détail. Vous avez dû vous demander pourquoi il était nécessaire de saisir deux fois la taille des secteurs. C'est fait pour l'ID piste et pour chaque ID secteur. D'accord, c'est redondant, mais c'est la clef de tout plombage. La première valeur va préciser la taille effectivement prise sur le disque par chaque secteur. La seconde n'est pas prise en compte lors du formatage mais seulement lors de l'écriture. Ainsi, une piste formatée en taille 2 dans l'ID piste contiendra des secteurs théoriques de taille 2, dont la taille formatée sera effectivement de 512 octets. Si vous dites dans les ID secteurs qu'ils sont de taille 3, toute écriture dans l'une de ces zones engendrera un débordement de données de 512 octets (512 réservés contre 1 024 écrits). Vous voyez que la moindre erreur peut être fatale. C'est cette erreur que nous allons utiliser.

Les copieurs se voient obligés d'écrire dans tous les secteurs analysés, cela dans l'espoir de créer une copie conforme à l'original. Si nous lui faisons une ruse dans les tailles annoncées par les ID secteurs comme expliqué ci-dessus, il se verra écraser bêtement une disquette qu'il voulait engendrer. C'est gros, mais ça protège super sec. Cette explication n'est pas la plus pure, mais elle fait palper le principe fondamental d'anticopies sur CPC.

Lorsque vous formatez une piste de secteurs de taille 0, voici ce qui se passe : l'écriture d'un secteur de taille différente écrase quelques secteurs de taille 0. Un exemple. Un secteur écrasé pour l'écriture d'un secteur de taille 1 ; 2 pour 2 ; 5 pour 3 ; 10 pour 4 ; et 21 pour 5. Si vous écrivez dans un secteur leurre de taille 4 sur ce genre de piste, il ne vous restera que 22 secteurs au lieu de 32. Facile alors, en lisant chaque secteur, de savoir s'il en manque.

Si vous avez pigé le truc, on peut passer à du beaucoup plus sérieux.

TOUT CE QUI SE LIT S'ECRIT

C'est bien beau de mettre des secteurs partout. Encore faut-il pouvoir les utiliser. C'est ce que se propose avant tout cette routine. Elle va permettre de remplir les récipients vides que nous avons installés sur la disquette. Son fonctionnement en est simple.

Elle se nomme ECRISECT.BIN et se loge en &8000. Lorsque vous l'appelez, il suffit de lui passer les paramètres suivants :

- CALL &8000, Pist, Premsect, dernsect, Tailsect, Buffer ;
- Pist : numéro de la piste entre 0 et 42 (plus ne sert à rien car les lecteurs sont équipés de butée et certains stoppent à 41) ;
- Premsect : numéro logique du premier secteur à remplir ;
- dernsect : numéro logique du dernier secteur à remplir ;
- Tailsect : type des secteurs à remplir (de 0 à 5 pour 128 à 4 096) ;
- Buffer : adresse de la zone mémoire contenant les données à déposer dans les secteurs visés (&4000 par exemple).

Pour écrire un seul secteur, donnez le même nom au premier et au dernier.

Facile d'écrire, mais pour lire ? Même principe, même adresse, mêmes paramètres, mais pas le même nom. Le programme se nomme LIRESECT.BIN.

Notre cher Valère nous a aussi envoyé un petit format de derrière les fagots qui envoie Disco 5 au rencart. Poubelle, la bête. Le principe utilisé est celui précédemment cité. C'est parti pour les valeurs.

Taille : 0, nombre : 32, Gap #3 : 1, remplissage : 0.

Laissez tout à zéro sauf :

```
NO  1 : &01/ 2
NO  4 : &02/ 2
NO  7 : &03/ 2
NO 10 : &04/ 2
NO 13 : &05/ 2
NO 16 : &06/ 2
NO 19 : &07/ 2
NO 22 : &08/ 2
NO 25 : &09/ 2
NO 26 : &0A/ 2
NO 27 : &0B/ 2
NO 28 : &0C/ 2
NO 29 : &0D/ 2
NO 30 : &0E/ 2
NO 31 : &0F/ 2
NO 32 : &10/ 2
```

Ecrivez les secteurs de 1 à 8 comme sous un format IBM. Comme deux secteurs seront écrasés par ceux-ci, mais qu'ils ont été réservés, pas de problème. Utilisez-les comme pour tout autre secteur qui se respecte. N'écrivez surtout pas dans les 8 autres (de &09 à &10) sans quoi le plombage sera naze, c'est ce que fera Disco 5.0 et il ne laissera que 7 secteurs en état de marche. Pour vérifier si on est bien sur l'original, il suffit de lire les 8 secteurs légaux. S'ils sont actifs, on peut rouler. Dans le cas contraire, dites bonjour au pirate en herbe.

DISCO 6.0 : AUX FRAISES

C'est l'ami et éternel génie Xor qui a trouvé le truc que voici :

- Taille : 0, nombre : 32, Gap #3 : 1, remplissage : 0.

Laissez tout à zéro sauf:

```
NO  1 :  01/ 2
NO  4 :  02/ 2
NO  5 :  03/ 2
NO  6 :  04/ 2
NO  9 :  05/ 2
NO 12 :  06/ 2
NO 15 :  07/ 2
NO 16 :  08/ 6
NO 17 :  09/ 5
NO 18 :  10/ 4
NO 19 :  11/ 3
NO 20 :  12/ 2
NO 21 :  13/ 1
NO 23 :  14/ 6
NO 24 :  15/&FF
NO 25 :  16/ 2
NO 28 :  17/128
NO 29 :  18/ 1
NO 31 :  19/ 6
NO 32 :  20/ 2
```

Ecrivez dans les secteurs logiques 1, 4, 5 et 6, et vous vous retrouverez avec une piste de 24 secteurs que Disco 6.0 ne saura recopier. Pour tester l'original, faites simplement un test de présence des secteurs (Disco n'en laisse que 7 ou 8 d'actifs).

Voilà, c'est fini pour ce mois-ci, mais nous vous en réservons encore pour le mois prochain.

La vie est ainsi faite. Merci à Xor, à Valère et à Thierry Frache pour leur collaboration si précieuse.

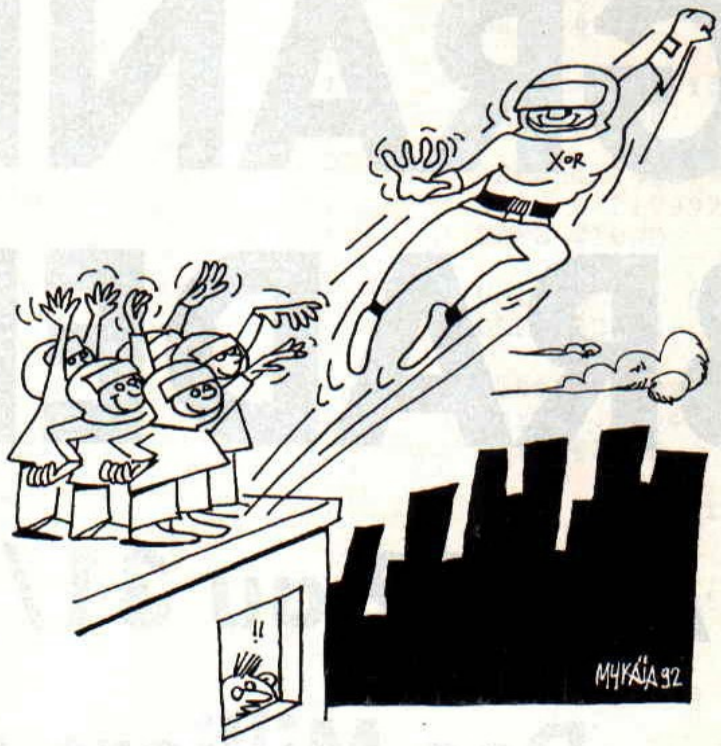
Merci à tous pour cette intrusion dans le monde merveilleux de la technique. Bonne lecture.

Sined le Barbare
(99,9 % - 1 verre de whisky)
Poum (0,1 % - 5 bouteilles de whisky))

INITIATION

PROTECTIONS LOGICIELLES (IV)

Pour qu'un logiciel shareware ou du commerce vous rapporte un tantinet de monnaie, il faut au minimum qu'il ne soit pas dupliqué par le premier copieur venu. Mais cela n'est pas tout, encore faut-il bien organiser la protection au sein du système et du programme, pour qu'elle ne soit pas trop vulnérable. Voici quelques conseils en tout genre pour mieux vous protéger.



Imaginez une disquette incopiable, dans un format que même Discologie 38.2 ne saurait reproduire. Un produit quelconque sur une telle disquette serait-il réellement en sécurité ? Non et cinquante fois non. Révisez vos leçons. Si le code permettant de vérifier la protection n'est pas lui aussi protégé par quelques plombages puissants, le format d'enfer de la disquette ne sera qu'un leurre inutile et seulement bon à rebuter les inconditionnels des copieurs. N'importe quel petit bidouilleur malin saura taper dans le code et enlever le petit morceau de programme qui plante le produit.

Il faut donc protéger la routine de vérification de format. Pour cela, un second sous-programme est chargé de vérifier l'intégralité de la routine de vérification. Ce dernier doit aussi être surveillé, car s'il est contourné, sa routine protégée pourra l'être aussi. Bref, à tout endroit du programme, en n'importe quelle circonstance, il faut penser à aller voir du côté de la protection ou de sa routine.

OU PROTEGER QUOI ?

De préférence, il faudra protéger les parties sensibles du programme par des sommes de contrôle et des comparaisons de zones.

Pour ne pas que tout saute tout de suite, il ne faut pas appeler la protection par un CALL, mais recopier le programme de protection çà et là dans le code, pour que le pirate fasse un maximum de manipulations avant de déplomber totalement le logiciel. Pour brouiller les pistes et pour que les recherches de protections soient les plus ardues possibles, n'hésitez pas à coder la partie vérification. Créez ainsi deux routines ; la première décodera le sous-programme de vérification, puis l'appellera.

La seconde le recodera après son appel pour ne laisser aucune trace. Le programme de codage basé sur un XOR, ou quelques décalages, devra générer différentes versions de camouflage pour qu'une recherche numérique soit, elle aussi, inefficace.

POUSSER LE VICE

Pour que les outils mémoire ne soient pas facilement utilisables, régénérez les vecteurs système et remplissez de zéros les zones de mémoire non utilisées par le programme. Vérifiez quelques centaines d'octets plus loin que ces zones sont bien vierges. Lors de l'initialisation de votre logiciel, faites un maximum de CALLs imbriqués. Cela embrouille considérablement les pirates et peut faire déborder la pile si elle est déjà utilisée par un programme résident. N'hésitez pas à utiliser des routines de scanning sous interruption pour vérifier si telle ou telle routine clef n'est pas contournée. A la limite de la folie, faites des routines de calcul de sommes de contrôle pour vérifier que chaque page de 256 octets meublant la mémoire est bien telle que vous l'avez

écrite. Le cas échéant, faites des sommes de contrôle sur ces checksums. N'oubliez pas que tout ce que vous avez fait, un pirate peut le défaire. Les pirates qui veulent protéger leurs œuvres vont jusqu'à utiliser des timings. De cette manière, si le moindre octet de code est modifié, la machine plante. Bref, une protection est le contraire de ce qu'on ne doit pas faire dans la vie et se limite en une phrase : faites aux autres ce que vous n'aimez pas qu'on vous fasse. J'espère que ces quelques lignes vous auront donné des idées plus que malignes.

CATALOGUE CAMOUFLE

Pour ceux qui ne se sentent pas la force de réaliser des protections de folie comme celles citées ci-dessus, il est possible de dresser des pièges système. Comme vous le savez, le CP/M n'est capable de s'en sortir que si les programmes et données qu'il manipule sont organisés dans des fichiers à son format. Le fait d'éparpiller des données sur le disque ou de cacher des fichiers dans des catalogues parsemés sur la face de la disquette rend les copies de fichiers totalement aberrantes. Vous pouvez ainsi rendre irréutilisables vos programmes en embrouillant le petit pirate avec des pokes et des appels étranges et incompréhensibles pour un non-initié.
Il va sans dire que de tels procédés ne seront pas à l'abri de la copie intégrale de disquette. En revanche, toute copie de fichiers ou tentative de détournement ou de réutilisation de codes sera retardée, voire compromise. Ne restons pas sur notre faim et voyons quelques exemples.

DES INFORMATIONS NON FICHEES

Le truc le plus bête est de sauvegarder des programmes sur des secteurs qui n'appartiennent à aucun fichier. Le chargement de ces informations devra donc se faire en passant par des lectures de secteurs. Une copie de fichiers ou une sauvegarde quelconque de fichier sur la disquette risquera de compromettre le fonctionnement du logiciel entier.
En poussant un peu, la ruse encore plus folle est de loger des petits bouts de programme en langage machine à la fin des blocs d'un fichier. Pour l'exemple, une zone de données de 1 000 octets laissera 24 octets de libre dans un bloc de 1 024 octets. Vous pouvez vous servir de ce bout pour y coller n'importe quelle routine assembleur. Appeler une routine machine dans le buffer disque, sans qu'on ait chargé autre chose qu'un programme en Basic, en déconcertera plus d'un et le mettra aux fraises. J'ai même vu des fous dessiner une page écran autour d'une petite routine. Le fait de faire un CALL au beau milieu de l'écran se révélait complètement ébouriffant. Surtout que l'adresse de la routine était finement calculée puis stockée dans HL, et son corps, en dehors de quelques pokes fort à propos, faisait du n'importe quoi sur mesure.
Vous passerez de longues heures pour vérifier les calculs de l'adresse, autant encore pour essayer de déchiffrer une routine piège-couillon, pour vous apercevoir que tout était bon du premier coup. J'ai perdu 32 768 cheveux ce jour-là (Poum aussi, mais les siens n'ont pas repoussé).

LES CATALOGUES FANTOMES

Eh oui, ce n'est pas de la magie. Notre CPC est bien capable de faire naviguer des informations sur la disquette. Il est possible de déplacer le catalogue.
On peut aussi changer sa taille mais, malheureusement, on ne peut changer sa structure. Dans la longue liste des variables utilisées par le système au sujet du lecteur de disquette, il en existe quatre qui vont nous être d'un grand secours. Les voici dans leur ordre d'apparition:
- &A897-&A898 : nombre d'entrées du catalogue moins une (63 par défaut) ;
- &A899-&A89A : masque binaire des positions des blocs du catalogue sur les pistes (&C000 soit % 1 100 000 000 000 000 par défaut, où les 1 représentent les endroits où sont les blocs à utiliser)
- &A89D : nombre de pistes utilisées par le système, soit encore le numéro de la piste contenant le catalogue (0 en data, 2 en CP/M) ;
- &A8A8 : super génial, de la plus haute importance : c'est ce flag qui va nous permettre de travailler avec les trois autres variables.
Avant tout, il faut savoir que le système est très prudent. Il faut qu'il vérifie tout et qu'il réinstalle sans cesse des valeurs qu'il aime utiliser.
Si nous pokons 95 en &A897 et 224 en &A899, nous devrions avoir un catalogue long de trois blocs, donc de 96 entrées.
Que nenni, le système regardera la disquette, et une fois la commande CAT lancée, nous nous apercevons que ce que nous avons fait est vain et que les valeurs par défaut sont revenues. Quelques PEEK le prouvent. Serions-nous perdus ?

LE POKE BONHEUR

Non, nous n'avons pas été menés à notre perte par le système.
Les programmeurs de l'Amsdos ont fait des miracles et nous permettent de forcer le système à nous écouter.
Le fait de poker 255 en &A8A8 le supplie de laisser nos valeurs tranquilles, ce qu'il fait, tombant sous le charme de tant de bits tendus. Cela fait, nous pouvons poker ce que nous voulons, où nous voulons.

CAT	&A897	&A899
32	31	128
64	63	192
96	95	224
128	127	240
160	159	248
192	191	252
224	223	254
256	255	255

En voici quelques exemples :

CAT est le nombre d'entrées de réper toire, qu'il ne faut pas confondre avec le nombre de fichiers stockables (voir le cours assembleur de ce mois). Formatez une disquette, pokez un des couples de ces valeurs sans oublier le fatidique POKE &A8A8,255, et tapez la ligne suivante:

```
FOR I = 1 TO CAT:SAVE STR$(I):?
I:NEXT
```

Où CAT est la valeur correspondant à votre choix. Vous ne serez pas déçu du voyage. Sachez tout de même qu'utiliser plus de 160 entrées devient aberrant, aussi bien dans la lenteur de l'émission du catalogue que dans le fait que 172 Ko de données - 5 ko de catalogue = 168 blocs libres, soit environ 1 Ko par fichier. Cent soixante fichiers sur une disquette, c'est déjà pas si mal.
Le dernier truc que nous vous offrons consiste à propulser le catalogue à tout endroit de la disquette par un poke de 0 à 41 en &A89D. Vous pouvez tout à fait laisser un catalogue bidon sur votre piste 0, en créer un vrai en piste 20 et laisser profiter les données des zones ntermédiaires.

Attention toutefois de protéger les blocs de la piste 20 par un fichier factice du catalogue d'origine, sans quoi vos données situées sur la seconde partie seront bonnes pour la poubelle ! Remplissez la première partie de la disquette, sauvegardez un fichier factice de 2 Ko qui doit commencer sur le secteur &C1 d'une piste, et le tour est joué. Il ne reste plus, à ce moment, qu'à poker le décalage de votre fichier (en piste à l'adresse &A89D) et de travailler sur votre nouveau catalogue protégé.

FIN DE DOSSIER

Et voilà, notre folle épopée au milieu du monde des pirates et des plombeurs est maintenant terminée. C'est bête, j'aurais bien voulu que cela continue, mais je suis à cours d'astuces.
Anecdote : pour se crêper le chignon, Longshot et Ruby s'envoyaient à tour de bras des plombages en défi. Ils ne cessaient de faire des merveilles. Je laisse votre imagination chercher qui fut le vainqueur.
Nous espérons que toutes ces idées vous seront utiles et vous souhaitons bonne plombe.

Sined(redon)