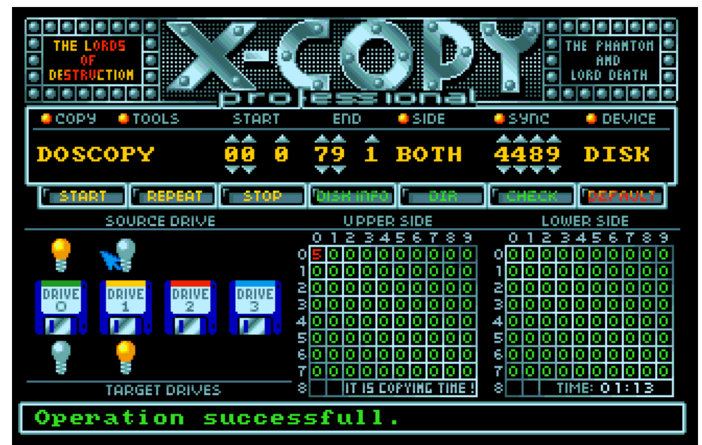
CRACK BAD DUDES VS DRAGON NINJA v1.1

Un tuto sur le modèle des tutos flashtro, en espérant que ça donne envie de s'y mettre à ceux qui veulent retrouver la magie de l'âge d'or des années 80...

<u>Nécessaire pour ce tuto</u> :
- un amiga 500 + Action Replay MK 3 (Dénommée AR ci-dessous)
- ou bien WINUAE + AR rom + CAPSImg.dll pour lire les fichiers .ipf (qui sont les fichiers des images de disque originaux (cf. tutos spécifiques WINUAE et fichiers .ipf))
- Des connaissances de base sur l'ASM 68000, l'adressage mémoire, la notation hexadécimale, etc...Commencez par des tutos la dessus si vous n'y connaissez rien
- Une connaissance de base de l'amiga OS et des librairies est optionnel (mais mieux !)

D'abord on essaie de copier le disque original (ipf image file N° 98) avec le célèbre X-COPY On voit rapidement une erreur sur le track N°1



Comment ça track N°1 ?? c'est marqué 0 sur X-COPY ??

Quelques mots sont nécessaires sur l'organisation d'un disque et la numérotation des pistes (ou "tracks" en anglais) :

Il y a 2 têtes de lecture dans un lecteur de disquette (supérieure et inférieure), qui lisent les datas simultanément sur les 2 faces du disque. Sur chaque face il y a 80 tracks.
L'association d'une piste supérieure et d'une piste inférieure s'appelle un cylindre, ces cylindres sont numérotés de 0 à 79
A chaque fois qu'on entend un petit bruit du lecteur, les têtes changent de cylindre (et non pas de track !)

En théorie les tracks sont numérotés comme suit :
Cylindre 00 : track 00 on lower side (LS), track 01 on upper side (US)
Cylindre 01 : track 02 on LS, track 03 on US
......
Cylinder 79 : track 158 on LS, track 159 on US

Mais souvent, la terminologie n'étant pas bien claire à l'époque, les tracks sont aussi numérotés uniquement par face (comme sur la capture d'écran X-COPY ci-dessus):
Cylinder 00 : Track 0 US ; Track 0 LS
Cylinder 01 : Track 1 US ; Track 1 LS
Cylinder 02 : Track 2 US ; Track 2 LS
etc....
Du coup les termes de cylindres et tracks se confondent.....peu importe, le contexte du tuto que vous lirez vous permet en général de comprendre dans quel sens le vocabulaire est utilisé

Pour revenir à nos moutons, pour le disque Bad dudes, on voit donc que le track N°1 (en terminologie "officielle", ou track 0 face supérieure) n'est pas copié (erreur d'écriture dans X-COPY)
Ce type de protection est typique d'une protection dite Rob Northen copylock ("RNC") du nom de son inventeur. La piste non standard est lisible par un lecteur amiga mais non copiable (il fallait des duplicateurs spéciaux pour ça). La lecture de cette piste prend un temps différent de la lecture d'une piste normale, le programme de protection va donc vérifier ce temps de lecture, et s'il est "normal", la disquette est une copie, le chargement s'arrête.
Dans ce cas, le programme de protection est aussi le "loader" du jeu, donc on ne peut pas simplement le bypasser. De plus ce programme est chargé crypté en mémoire et se décrypte uniquement si la vérification de la piste est ok. Pour lire cette piste, le programme est obligé d'utiliser des instructions superviseur (cf. plus loin)

Quand on essaie de booter sur le backup qu'on a fait avec X-COPY, le lecteur charge le track 00 puis s'arrête très rapidement sur un écran blanc, ce qui n'arrive pas avec la disquette originale. Donc la routine de protection doit être un programme très court chargé dès le boot

Il faut savoir que lors du boot, l'amiga charge et exécute les 2 premiers secteurs du track 00. (un Track est séparé en 11 secteur de 512 octets chacun (donc 1 track fait 11 x 512 = 5632 octets). Le programme du boot fait 2 secteurs donc au maximum 1024 octets (1 Ko)

Examinons donc ce programme boot dans notre cas avec l'Action Replay. On met le backup dans le DF0:, on allume l'AR et on charge le track 0 en mémoire, par exemple à l'adresse $50000 grâce à la commande :
RT 0 1 50000 : Cette commande lit 1 track depuis le track 0 et met les données à l'adresse mémoire $50000. Si on avait voulu lire 2 track depuis le track 5 et les mettre en mémoire à l'adresse $30000 par exemple, il aurait fallu taper :
RT 2 5 30000
!! Attention !!
L'AR fonctionne par défaut en hexadécimal ! Si on veut charger 16 tracks, on écrit :
RT 0 10 50000 ($10 = 16 en décimal)
Si on veut écrire en décimal dans l'AR, il faut rajouter le symbole ! devant les chiffres
Pour la lecture de track, on raisonne en décimal donc je vous conseille de toujours rajouter le "!" pour éviter les erreurs.

Donc une fois qu'on a chargé le track 0 à l'adresse $50000, on désassemble les données avec la commande D de l'AR :
D 50000 (Ensuite on presse entrée plusieurs fois)

```
Ready.
rt 0 1 50000
Disk ok
d 50000
~050000 NEG.W    A7
~050002 SUBQ.B   #1,D0
~050004 ADDA.L   -59(A1,A4.W),A6
~050008 ORI.B    #70,D0
~05000C MOVEM.L  D0-D1/A0-A2,-(A7)
~050010 LEA      50000(PC),A0
~050014 MOVE.L   A0,8(A7)
~050018 MOVE.L   #3EA00,10(A7)
~050020 MOVEA.L  C(A7),A1
~050024 MOVE.L   10(A7),D0
~050028 MOVE.L   D0,28(A1)
~05002C MOVE.L   #1200,24(A1)
~050034 MOVE.L   #400,2C(A1)
~05003C MOVE.W   #2,1C(A1)
~050042 MOVEA.L  00000004.S,A6
```

Les 12 premier bytes sont le header, ce n'est pas du code, le premier long mot est pour DOS0, le second est le checksum du boot et le 3ème est la localisation du "rooblock" sur le disque (ne nous concerne pas pour l'instant). On peut voir tout ça grâce à la commande *M* de l'AR

```
m 50000
:050000 44 4F 53 00 DD F1 C6 A7 00 00 03 70 48 E7 C0 E0 DOS....ß...pH...
```

"DOS0" Checksum RootBlock

L'amiga commence à lire le code en $5000C.
Les premières lignes sauvent les registres sur la pile.
Les lignes $50018 à $50046 sont une routine qui lit $1200 octets depuis l'offset $400 de la disquette vers l'adresse $3EA00 de la mémoire.
Pour comprendre cette routine, il faut savoir que quand l'amiga boot, il y a un trackdisk.device ouvert pour DF0: et on peut travailler avec ce device grâce à une structure en mémoire appelée IOrequest. L'adresse de cette structure est en A1 lors du boot. L'organisation de cette structure suit le schéma ci-contre.

```
struct IOStdReq
{                                        /* Offsets  */
struct   Message   io_Message;           /*  0  $00  */
struct   Device   *io_Device;            /* 20  $14  */
struct   Unit     *io_Unit;              /* 24  $18  */
         UWORD     io_Command;           /* 28  $1C  */
         UBYTE     io_Flags;             /* 30  $1E  */
         BYTE      io_Error;             /* 31  $1F  */
         ULONG     io_Actual;            /* 32  $20  */
         ULONG     io_Length;            /* 36  $24  */
         APTR      io_Data;              /* 40  $28  */
         ULONG     io_Offset;            /* 44  $2C  */
};
```

En regardant la structure IOrequest, on voit que l'offset à partir duquel lire les datas sur le disque doit être mis en A1 + $2C , la longueur des données à lire doit être mise en A1 + $24, l'adresse mémoire de destination doit être mise en A1 + $28 et enfin la commande à effectuer par le device (ici $2 cad lire les données) doit être mise en A1 + $1C
c'est ce que fait la routine du boot.
Enfin pour lancer cette requête au device DF0:, il faut utiliser la commande de la librairie Exec DoIO qui est situé à l'offset -$1C8 de l'Exec library (cf. ligne $50046)
(L'adresse de base de l'Exec library est à l'adresse 4 et est mise en A6)

Allons donc voir ce qu'il y a en mémoire en $3EA00 :
D 3EA00 puis entrée plusieurs fois

```
d3eaca
~03EACA MOVE.L   A6,-(A7)
~03EACC LEA      3EA4C(PC),A6
~03EAD0 MOVEM.L  D0-D7/A0-A7,(A6)
~03EAD4 LEA      40(A6),A6
~03EAD8 MOVE.L   (A7)+,-8(A6)
~03EADC MOVE.L   (A7),(A6)+
~03EADE MOVE.L   00000010,D0
~03EAE4 PEA      3EAF0(PC)
~03EAE8 MOVE.L   (A7)+,00000010
~03EAEE ILLEGAL
```

C'est une routine copylock typique, car cette routine, pour s'autodécrypter et lire le track anormal, utilise les instructions superviseur PEA et ILLEGAL, qu'on ne trouve jamais dans du code "normal". C'est pour ça que l'opcode $48 $7A (l'instruction PEA) est considéré comme une "signature" copylock.

On va essayer de trouver la fin de cette routine grâce à la fonction N de l'AR qui permet de voir la mémoire en mode ASCII. La routine semble s'arrêter en $3FAF0

L'idée pour le crack de ce type de protection, est de booter avec le disque original, d'attendre que la routine de protection/loader se soit auto décrypté, puis de remplacer sur le disque de backup la routine cryptée par la routine déjà décryptée !

Donc en pratique :

1 – Booter avec le disque ORIGINAL en DF0: et entrer dans l'AR juste quand le chargement Redémarre (la routine se sera autodécryptée). Attention, si on rentre dans l'AR trop tôt, la routine ne se sera pas encore auto décryptée, trop tard, elle se sera effacée
2 – On met le BACKUP en DF0: et on lit le track 0 à l'adresse $50000 : *RT !0 !1 50000*
3 – on copie la routine décryptée dans notre buffer avec *TRANS 3EA00 3FAF0 50400*
Cette fonction transfère le contenu de la mémoire depuis $3EA00 jusqu'a $3FAF0 (la fin de la routine décryptée que l'on a trouvé tout à l'heure grâce à N) en $50400 (on sait qu'il faut mettre les données à l'offset $400, puisque c'est à partir de là que le code du boot charge les données du PRG de protection sur la disquette (cf. ci-dessus))
4 - Il ne reste plus qu'à écrire ce nouveau track 00 sur la disquette : *WT 0 1 50000*
(WT = Write track)

Et c'est bon le jeu se charge........et c'est pas bon car le jeu démarre bien mais il n'y a pas d'ennemis et après quelques secondes de jeu.....il plante !
Les programmeurs de ce jeu sont des vicieux, ils ont rajouté une autre routine de protection !
On peut d'ailleurs s'en apercevoir avec winuae, car, en cours de chargement le cylindre 00 est de nouveau testé....
Là ça devient plus coton......d'autant qu'en plus les données sont "crunchées" sur le disque, donc on ne va pas pouvoir remplacer directement le code sur celui-ci simplement....

Commençons par essayer de trouver la routine en mémoire : on boot, on attend le moment ou le track 00 est testé, et on rentre dans l'AR juste à ce moment....on voit qu'on se situe en $1E6xx....donc la routine de protection est dans la zone mémoire $1Exxx
(On aurait aussi pu faire *F 48 7A* (F = Find, l'AR cherche dans la mémoire les endroits où se trouve une éventuelle signature copylock : l'AR nous renvoie alors une seule adresse : $1E336 donc il n'y a a priori qu'une seule routine de copylock)

```
d
~01E64A BEQ     0001E63E
~01E64C MOVEQ   #31,D2
~01E64E ADDQ.L  #1,D1
```

On essaie de trouver le début de cette routine en utilisant la fonction N et D, et on voit que la routine doit commencer en $1E31E
Voyons si cette adresse est appelée quelque part dans le programme grâce à la fonction FA de l'AR : *FA 1E31E* (pour éviter de nombreux faux positif, il ne faut pas qu'un registre ait cette valeur, or c'est le cas ici (on voit le contenu des registres en faisant R), donc il faut d'abord faire *R A2 0* puis *FA 1E31E* (on met le registre A2 à zéro avant de faire la recherche)

```
r
D0=000000C0 00000000 0005B4FD 00000003  00FC0820 00FC0822 00FC090E 00FC0826
A0=00DFF000 0001E266 0001E31E 00C3FAEC  0001CFE8 00C014B6 0001E2EC 00C7FFD6
PC = 0001E64A USP = 00C3D678 SR = 270C T=0 S=1 I=111 X=0 N=1 Z=1 V=0 C=0
r a2 0
D0=000000C0 00000000 0005B4FD 00000003  00FC0820 00FC0822 00FC090E 00FC0826
A0=00DFF000 0001E266 00000000 00C3FAEC  0001CFE8 00C014B6 0001E2EC 00C7FFD6
PC = 0001E64A USP = 00C3D678 SR = 270C T=0 S=1 I=111 X=0 N=1 Z=1 V=0 C=0

fa 1e31e
Search from: 000000 to: C80000
01E262 BRA      0001E31E
Searched up to adr: C80000
Ready.
```

```
fa 1e262
Search from: 000000 to: C80000
01D210 BSR     0001E262
02ADFC MOVE.W  -4(A1),D0
033E6A JSR     -8A(A6)
03D6A4 MOVE.W  -4(A1),D0
059B4A TST.L   -4(A1)
C2DC92 TST.L   -4(A1)
Searched up to adr: C80000
Ready.
d 1d210
^01D210 BSR    0001E262
^01D214 MOVE.L D0,000000E0.S
^01D218 NOP
```

L'AR nous renvoie une seule adresse : $1E262
En désassemblant $1E262, on trouve juste un BRA isolé......probablement un petit jeu de piste des programmeurs pour les crackers ??
Qu'à cela ne tienne, on recommence *FA 1E262* et on désassemble la première adresse qui nous est donnée, on voit qu'il y a appel à la routine de protection et aussitôt après le contenu de D0 est mis en $E0......mmmhhh......surement une clé de protection......avec notre backup il y a 00 en $E0

Quand on boot avec le jeu original et qu'on regarde en $E0 une fois le test de protection passé, le contenu de $E0 est : **A8 D3 98 FB** : **c'est le nombre magique** !

Le principe de ce 2ème crack va être de bypasser complètement la protection en essayant de mettre le code suivant à l'adresse 1E31E :

```
1E31E : move.l #A8D398FB,D0     ;  renvoie la bonne valeur dans D0
        RTS                     ; retour, on "bypasse" le reste de la protection
```

La suite n'est pas forcément compliquée mais il faut être très rigoureux et aller doucement pour éviter de faire des erreurs....
Le code en $1E31E n'apparait pas tout de suite en mémoire, il est d'abord chargé, décrunché puis exécuté, donc il faut changer ce code après son chargement mais avant son exécution......
Pour cela on va encore jouer avec l'AR.....on va stopper le code APRES le chargement des cylindres 28 à 30 mais avant le test du track 01.....il faut faire quelques essais-erreurs pour trouver le bon timing, mais on va finalement se retrouver dans du code aux alentours de $C2F6xx....en désassemblant un peu le code dans la région, on cherche une zone propice à l'insertion d'un saut vers un code de notre cru, pour ma part j'ai choisi $C2F690. Au moment du BRA en $C2F696, on sait que le code de protection est chargé mais non encore exécuté.....on va donc insérer un petit JSR vers un code de notre cru que l'on va mettre par exemple à l'adresse $300 (il y a souvent de la mémoire libre dans les premiers octets, ici $300 c'est bien, si on essaye $100 et $200, le programme du jeu ré écrit du code dans ces zones...).

```
;=====================================
^C2F678 MOVEQ   #2,D1
^C2F67A BSR     00C2F6E4
^C2F67E CMP.B   #2,D2
^C2F682 BLT     00C2F69A
^C2F684 CMP.B   #3,D2
^C2F688 BEQ     00C2F672
^C2F68A MOVEQ   #8,D1
^C2F68C BSR     00C2F6E4
^C2F690 MOVE.W  D2,D3
^C2F692 MOVE.W  #C,D1
^C2F696 BRA     00C2F6A4
;=====================================
```

Donc ce qu'on veut c'est :

```
^C2F690 JSR     00000300
^C2F696 BRA     00C2F6A4
```

; Attention 2 lignes du code original ont été effacées,
; il faudra les reprendre dans notre crack

avec en $300 un code qui écrit le code du crack ci-dessus en $1E31E

Ce n'est pas tout ! Pour insérer le JSR 300, il faut d'abord que le code en $C2Fxxx soit chargé et décrunché mais non encore exécuté
Même procédure que ci-dessus mais c'est plus simple…en effet, le code dans la $C2Fxxx est le premier code chargé par le programme, lors du chargement des cylindres 53 à 68

Le saut vers ce code se situe dans le loader chargé en mémoire depuis l'offset $400 du disque par le boot (instruction JMP (A0)), c'est assez simple à trouver, il suffit d'examiner le boot ou de jouer un peu avec l'AR pour s'en apercevoir (non détaillé ici pour ne pas alourdir le tuto)

Donc pour résumer dans l'ordre :

1 - Le boot charge un loader qui charge les cylindre 53 à 68 dans la zone mémoire $Cxxxxx, les décrunche et saut dans ce code décrunché par un JMP (A0)(situé en $1652)
A ce moment le code en $C2F690 est en mémoire et est "pachable", donc on va détourner le JMP A0 par une JMP vers une routine de notre cru qui met le JSR 300 en $C2F690

2 - Le programme charge plusieurs fois mais on sait que une fois en $C2F690, la routine de protection sera chargée et décrunchée mais non éxécutée, donc on pourra à ce moment-là "pacher" par un code en $300 qui écrit les 2 lignes de codes en $1E31E
(le crack proprement dit).

Comment faire tout ça ?   --->   Grace a la place restante sur le boot !
En examinant le boot de la disquette (donc les $400 premiers octets), on voit qu'il reste de la place pour écrire du code à partir de $C0 :

```
Ready.
rt 0 1 50000
Disk ok
n 50000
.050000 DOS....§...pH...A.../H../l......"o.. /..#@.(#l.....$#l......3l..
.050040 ..,x..N..8"o...)..f.Lß..NuProtection (C)Copyright 1989 Rob North
.050080 en Computing. All Rights Reserved..............................
.0500C0 ................................................................
.050100 ................................................................
.050140 ................................................................
.050180 ................................................................
.0501C0 ................................................................
```

```
d 50000
~050000 NEG.W    A7
~050002 SUBQ.B   #1,D0
~050004 OR.L     D7,(A1)+
~050006 LINEF
~050008 ORI.B    #70,D0
~05000C MOVEM.L D0-D1/A0-A2,-(A7)
~050010 LEA      50000(PC),A0
~050014 MOVE.L   A0,8(A7)
~050018 MOVE.L   #3EA00,10(A7)
~050020 MOVEA.L C(A7),A1
~050024 MOVE.L   10(A7),D0
~050028 MOVE.L   D0,28(A1)
~05002C MOVE.L   #1200,24(A1)
~050034 MOVE.L   #400,2C(A1)
~05003C MOVE.W   #2,1C(A1)
~050042 MOVEA.L 00000004.S,A6
~050046 JSR      -1C8(A6)
~05004A MOVEA.L C(A7),A1
~05004E MOVE.B   1F(A1),D0
~050052 BNE      00050024
~050054 MOVEM.L (A7)+,D0-D1/A0-A1
~050058 BRA      000500C0
```

Voici le code commenté que l'on met à partir de $C0:
Pour aller vers ce code, il faut remplacer le dernier RTS du boot par un BRA 500C0 ---------->

(pour écrire du code, on utilise la commande *A <adresse>* de l'AR)

```
d 500c0
~0500C0 MOVEM.L D0/A0-A1,-(A7)
~0500C4 MOVE.L  #1C,D0
~0500CA LEA     50130(PC),A0
~0500CE MOVEA.L #300,A1
~0500D4 MOVE.B  (A0)+,(A1)+
~0500D6 DBF     D0,000500D4
~0500DA MOVEM.L (A7)+,D0/A0-A1
~0500DE MOVE.L  #23FC4EB9,0000031C
~0500E8 MOVE.L  #C2,00000320
~0500F2 MOVE.L  #F69033FC,00000324
~0500FC MOVE.L  #30000C2,00000328
~050106 MOVE.L  #F6944EB9,0000032C
~050110 MOVE.L  #16AC,00000330
~05011A MOVE.W  #4ED0,00000334
~050122 RTS
```

On copie $1C octets depuis l'offset $130 du boot vers $300 (cf. encadré ci-dessous en $50130 pour voir le code qui est copié en $300)

Ce code met le code suivant en $31C (à la suite du code précédent)(cf. ci-dessous)

on reprend le RTS original du boot

```
~050130 MOVE.W  D2,D3
~050132 MOVE.W  #C,D1
~050136 MOVE.L  #203CA8D3,0001E31E
~050140 MOVE.L  #98FB4E75,0001E322
~05014A RTS
```

Un peu plus loin dans le boot, voici le code qu'on copie en $300

```
'000300 MOVE.W   D2,D3                    ;  On reprend les 2 instructions que l'on a fait
'000302 MOVE.W   #C,D1                    ;  "sauter" avec le JSR 300 en $C2F690
'000306 MOVE.L   #203CA8D3,0001E31E       ; ce code met le code de crack en $1E31E
'000310 MOVE.L   #98FB4E75,0001E322       ;
'00031A RTS                               ; retour en $C2F696
;===================================
'00031C MOVE.L   #4EB90000,00C2F690       ; ce code insère un JSR 300 en $C2F690
'000326 MOVE.W   #300,00C2F694
'00032E JSR      000016AC                 ; cf. ci dessous pour expliquer cette ligne
'000334 JMP      (A0)
```

Voici le code de notre boot terminé, mais si on le ré écrit tel quel sur la disquette, l'amiga ne chargera pas car le checksum du boot a changé, pour le recalculer, on utilise la fonction Bootchk de l'AR puis seulement on le ré écrit :
Bootchk 50000
WT 0 1 50000

Si on relance notre backup, en faisant *D 300*, on voit bien nos 2 routines présentes

Il reste une dernière chose à faire, c'est changer le loader du boot (offset $400) pour qu'il saute en $31C : (après un *RT 0 1 50000 puis D 50400*)

```
^05050A BRA      00050512
;======================================
^05050C MOVEA.L #4BEB8,A0
^050512 BSR      00050570
^050516 JMP      (A0)
;======================================
^050518 JSR      00049000
```

```
^05050A BRA      00050512
;======================================
^05050C MOVEA.L #4BEB8,A0
^050512 JMP      0000031C
;======================================
^050518 JSR      00049000
```

Comme on le voit, on a été obligé de faire "sauter" une instruction, c'est pour ca qu'elle est reprise dans notre patch en $31C mais sous forme de JSR et non plus de BSR pour ne pas avoir d'adressage relatif

N.B : Ce crack est théoriquement un "mauvais" crack car il ne fonctionne que sur un amiga avec 1Mo de mémoire (le code est en mémoire haute, à partir de $C00000)
Pour le faire fonctionner sur tous les amiga, il faudrait écrire le JSR 300 sur une adresse relative au début du code et non une adresse absolue. C'est assez simple a faire mais à l'heure de winuae, cela n'a plus tellement de raison d'être, donc je n'ai pas alourdi le tuto avec ca.

Bon jeu !

JEL 08/2018 avec la participation de Giants

Merci encore au site flashtro et à Gi@nts pour leur travail et le partage de connaissance !
Version PDF