Superfrog
by TEAM 17

Cracking RNC PDOS MFM Protection

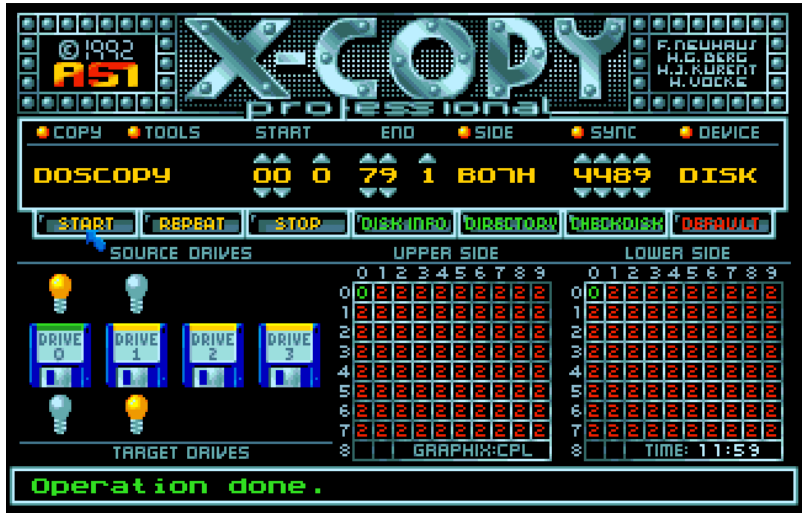
Contents

0 Introduction

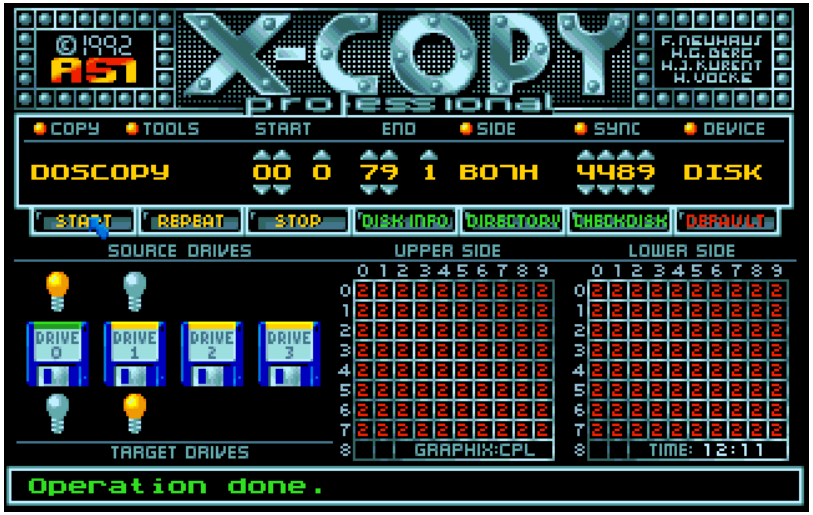
We will need:

Superfrog (CAPS image 35 or original disks)
Amiga 500 with 1MB/2MB Chip RAM
FIMP - File Imploder 2.34 (LSD Legal Tools Disk 28)
Action Replay 3
RNC Sector Loader (with stripped off writing code)
Few bottles of Lucozade

When trying to copy disk 1 and 3 we'll see that besides the first two tracks the disk uses some kind of custom MFM disk format, which cannot be copied. Disk 2 has no standard DOS tracks and uses throughout the whole disk a custom MFM disk format.



Disk 1 / Disk 3



Disk 2

1 Analysis of boot process and loading

First we load the first track into memory and make the first JMP (A3) to point to itself before we fix the checksum and write it back to disk.
Then we'll reboot and enter Action Replay 3 when we're stuck in the endless loop and change the instruction back again to represent the original instruction.

The boot block gets loaded to address $5C40.
Now we can step through the code in memory and see what's happening.

```
~005C4C MOVE.L  A1,-(A7)
~005C4E MOVE.L  #1000,D0
~005C54 MOVE.L  #10002,D1
~005C5A MOVEA.L 00000004.S,A6
~005C5E JSR     -C6(A6)      ----------> _LVOAllocMem
~005C62 MOVEA.L D0,A3
~005C64 MOVEA.L (A7)+,A1
~005C66 TST.L   D0
~005C68 BEQ     00005CA8
~005C6A MOVEA.L A1,A2
~005C6C MOVE.L  A3,28(A1)
~005C70 MOVE.L  #1000,24(A1)
~005C78 MOVE.L  #400,2C(A1)
~005C80 MOVE.W  #2,1C(A1)
~005C86 MOVEA.L 00000004.S,A6
~005C8A JSR     -1C8(A6)     ----------> _LVODoIO - Copy data after bootblock from disk to $400 - $1400 to $A498 (A3)
~005C8E MOVEA.L A2,A1
~005C90 MOVE.B  1F(A1),D0
~005C94 BNE     00005C6C
~005C96 MOVEA.L 00000004.S,A6
~005C9A JSR     -84(A6)      ----------> _LVOForbid
~005C9E MOVEA.L 00000004.S,A6
~005CA2 JSR     -96(A6)      ----------> _LVOSuperState
~005CA6 JMP     (A3)         ----------> Continue execution at $A498
```

```
~00A498 BSR     0000A68E
~00A49C LEA     A680(PC),A1
~00A4A0 MOVE.L  #400,(A1)
~00A4A6 MOVE.W  #4000,00DFF09A
~00A4AE MOVE.L  #7FFF7FFF,00DFF09A
~00A4B8 LEA     000003F4,A0
~00A4BE MOVE.L  #1000000,(A0)+
~00A4C4 MOVE.L  #1800000,(A0)+
~00A4CA MOVE.L  #FFFFFFFE,(A0)
~00A4D0 MOVE.L  #3F4,00DFF080
~00A4DA CLR.W   00DFF180
~00A4E0 MOVE.W  #81D0,00DFF096
~00A4E8 LEA     A67C(PC),A0
~00A4EC MOVEA.L (A0),A0
~00A4EE ADDA.L  #7D0,A0
~00A4F4 LEA     A736(PC),A2
~00A4F8 MOVE.L  A0,(A2)
~00A4FA LEA     A724(PC),A2
~00A4FE MOVE.L  A2,00000080
~00A504 TRAP    #0
~00A506 LEA     A67C(PC),A0
~00A50A MOVEA.L (A0),A0
~00A50C ADDA.L  #FA0,A0
~00A512 MOVEA.L A0,A7
~00A514 LEA     A67C(PC),A1
```

```
~00A518 MOVE.L  A0,(A1)
~00A51A CLR.L   D0
~00A51C MOVE.W  #B,D1           --> Sector start
~00A520 MOVE.W  #B,D2           --> Sectors to read
~00A524 MOVE.W  #8000,D3
~00A528 LEA     0007C180,A0     --> Load address
~00A52E LEA     00070000,A1     --> MFM Buffer
~00A534 BSR     0000A9CC        --> Call to loader
~00A538 LEA     0007C180,A0
~00A53E LEA     0007C180,A1
~00A544 BSR     0000A772        --> Decrunch
~00A548 LEA     0007C180,A0
~00A54E LEA     0007C180,A1
~00A554 BSR     0000A8D2        --> Hunk processing
~00A558 MOVEA.L A680(PC),A0
~00A55C MOVEA.L A67C(PC),A1
~00A560 MOVEA.L A678(PC),A2
~00A564 JMP     0007C180
```

The call to the loader basically loads track 1 (second track) from the disk (sectors 11 - 22).
Since track 1 is also standard AmigaDOS format, we don't have to replace the loader here.

Set a breakpoint at $A564.
Upon break, trace the 2 jump instructions and you'll end up here:

```
~07F486 MOVE.L  A0,0007F500
~07F48C MOVE.L  A1,0007F504
~07F492 MOVE.L  A2,0007F508
~07F498 MOVE.L  #7F50C,00DFF080
~07F4A2 CLR.L   D0
~07F4A4 LEA     7F518(PC),A0
~07F4A8 MOVEA.L 7F504(PC),A1
~07F4AC LEA     0007C186,A2
~07F4B2 BSR     0007F7D8
~07F4B6 CLR.L   D0
~07F4B8 LEA     7F51E(PC),A0
~07F4BC MOVEA.L 7F500(PC),A1
~07F4C0 LEA     0007C186,A2
~07F4C6 BSR     0007F7D8
~07F4CA MOVEA.L 7F504(PC),A0
~07F4CE MOVEA.L 7F504(PC),A1
~07F4D2 BSR     0007F524
~07F4D6 MOVEA.L 7F500(PC),A0
~07F4DA MOVEA.L 7F500(PC),A1
~07F4DE BSR     0007F524
~07F4E2 MOVEA.L 7F504(PC),A0
~07F4E6 MOVEA.L A0,A1
~07F4E8 MOVEA.L 7F500(PC),A2
~07F4EC BSR     0007F69A
~07F4F0 MOVEA.L 7F500(PC),A0
~07F4F4 MOVEA.L 7F504(PC),A1
~07F4F8 MOVEA.L 7F508(PC),A2
~07F4FC JMP     (A1)
```

Read tracks

Decrunch $80FA0

Decrunch $400

Hunk processing

JMP $80FA0

```
~07F7D8 CLR.L   D0
~07F7DA MOVE.W  (A0)+,D1
~07F7DC MOVE.W  (A0)+,D2
~07F7DE MOVEA.L A1,A0
~07F7E0 LEA     7C186(PC),A1
~07F7E4 MOVE.W  #8000,D3
~07F7E8 MOVE.L  #12389A,D4
~07F7EE BSR     0007F800
~07F7F0 TST.L   D0
~07F7F2 BEQ     0007F4FE
~07F7F6 MOVE.W  #F00,00DFF180
~07F7FE BRA     0007F7F6
```

Loader

Parameters to first call of loader at $7F4B2:
A0 = $80FA0
A1 = $7C186
D1 = $38
D2 = $162
D3 = $8000
D4 = $12389A

Parameters to second call of loader at $7F4C6:
A0 = $400
A1 = $7C186
D1 = $18
D2 = $20
D3 = $8000
D4 = $12389A

After file loading, take a look at the memory addresses where the files where loaded to:

```
:080FA0 41 54 4E 21 00 05 0F 0C 00 02 B7 1E A8 40 80 74 ATN!.........@.t
:080FB0 FF 40 01 DC 5A 4A 03 E9 56 82 20 D3 4E A2 47 00 .@.üZJ..V. .N.G.
:080FC0 26 0B 02 D0 81 E2 01 01 13 FA 00 0B 76 6E B9 00 &...........vn..

:000400 41 54 4E 21 00 00 74 14 00 00 2D 96 02 C1 04 7E ATN!..t...-....~
:000410 09 99 04 7F 05 0F A8 F1 13 93 1C C0 1F 99 00 77 ...▒...........w
:000420 17 7E 30 07 99 08 77 0F 85 AD 0D 03 C3 04 3F 7B .~0...w.......?{
```

After passing through instructions $7F4D2 and $7F4DE you'll notice that the contents changed.
The data was decrunched in place:

```
:080FA0 00 00 03 F3 00 00 00 00 00 00 00 04 00 00 00 00 ................
:080FB0 00 00 00 03 00 01 20 D3 40 00 D4 A8 40 00 1C FF ...... .@...@...
:080FC0 40 01 DC 5A 00 00 03 E9 00 01 20 D3 4E F9 00 00 @.üZ...... .N...
:080FD0 00 26 00 01 00 02 00 00 00 01 00 00 00 01 00 00 .&..............
:080FE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:080FF0 00 00 4E B9 00 00 76 6E 4E B9 00 01 1B 6A 4E B9 ..N...vnN....jN.
:081000 00 00 0C 6A 4E B9 00 00 B6 F8 4E B9 00 00 B6 66 ....iN.....N....f

:000400 00 00 03 EA 00 00 1C FF 00 00 00 00 00 00 00 00 ................
:000410 00 0F 00 00 00 20 00 1F 00 D1 00 60 01 CF 00 80 ..... .....`....
:000420 03 BF 01 00 05 63 02 3C 02 C1 04 7E 09 99 04 7F .....c.<...~...▒
:000430 05 99 08 77 13 99 08 77 13 99 08 77 1F 99 00 77 ...w...w...w...w
:000440 1F 99 00 77 1F 99 00 77 1F 99 00 77 17 99 08 77 ...w...w...w...w
:000450 07 99 08 77 0F 85 00 7F 03 C3 04 3F 05 FF 02 1E ...w...▒...?....
:000460 02 FF 01 00 01 7F 00 80 00 FF 00 00 00 3F 00 00 .....▒.......?..
:000470 00 0F 00 00 00 00 00 00 00 00 00 00 07 00 00 00 ................
```

The red marked areas represent hunk header information.
$3F3 -> HUNK_HEADER
$3E9 -> HUNK_CODE
$3EA -> HUNK_DATA

Let's stay focused on the file being loaded to $80FA0 (main file).
When we go pass the $7F4EC instruction, the contents change one more time:

```
:080FA0 4E F9 00 08 0F C6 00 01 00 02 00 00 00 01 00 00 N...............
:080FB0 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:080FC0 00 00 00 00 00 00 4E B9 00 08 86 0E 4E B9 00 09 ......N.....N...
:080FD0 2B 0A 4E B9 00 08 1C 0A 4E B9 00 08 C6 98 4E B9 +.N.....N.....N.
:080FE0 00 08 C6 06 4E B9 00 08 BD 6A 4E B9 00 08 A1 12 ....N....jN.....
:080FF0 4E B9 00 08 A1 CE 4E B9 00 08 86 C2 61 00 10 40 N.....N.....a..@
:081000 4A 79 00 08 0F B6 67 14 4A B9 00 08 0F B8 66 00 Jy....g.J.....f.
:081010 00 76 4A 79 00 08 0F BC 66 00 00 6C 61 00 1E 1A .vJy....f..la...
:081020 61 00 20 5E 4A 79 00 08 32 58 66 00 FF D0 42 79 a. ^Jy..2Xf...By
:081030 00 08 19 90 4A B9 00 08 0F B8 66 00 00 0C 4E B9 ....J.....f...N.
:081040 00 08 1B 06 60 00 00 40 4E B9 00 08 C6 98 4E B9 ....`..@N.....N.
:081050 00 08 B5 20 4E B9 00 08 B5 8C 4E B9 00 08 C7 14 ... N.....N.....
```

After processing of the hunk header, the relative addresses are converted to absolute addresses and the hunk header is removed.

We can see the address calculation in action in the first instruction which is a jump:
$4E $F9 $00 $00 $00 $26 becomes $4E $F9 $00 $08 $0F $C6.
($26 + $80FA0 = $80FC6)

Important: Keep the main file's hunk header length in mind which is **$2C** bytes!

Program execution continues to jump into the unpacked file memory address $80FA0.

When you let the game run, you'll notice that it seems like there is another loader.
Debugging or searching for the hex signature (48 E7 7F FC 4E 56 FF DE) of the previous loader,
gives us the location of the main loader: $8C038.

That sums up to 3 loaders in 3 separate files located on disk 1:

1.) Sectors $2 - $A
2.) Sectors $B - $16
3.) Sectors $38 - $19A

The first loader we can keep in untouched because it does only read the standard encoded sectors up to sector $16 (size of 2 standard tracks).
The second loader has to be fixed or replaced, because it reads the specially encoded and encrypted sectors coming after sector $16.
The third loader has to be fixed or replaced as well.

Before we proceed to the next steps, make sure you grab the 2nd and 3rd file containing the loaders from memory just after it was unpacked for later patching:

2nd loader containing file finished loading at $A548 (Address: $7C180 / Length: $3AF4)
3rd loader containing file finished loading at $7F4D6 (Address: $80FA0 / Length: $50F0C)

2 Disk format and loader explanation

Like most Team17 titles, Superfrog also uses the Rob Northen PDOS disk format.

A standard AmigaDOS disk has 160 tracks, where each track is composed of 11 sectors and each sector having a size of 512 bytes.
On the other hand a PDOS disk has also 160 tracks, but 12 sectors per tracks with 512 bytes capacity per sector.
So you can fit 880 KB data on a standard AmigaDOS disk (1760 or $6E0 sectors) and 960 KB on a PDOS disk (1920 or $780 sectors).

To read the sectors from the disk, a special loader is needed in the case of the original game it is a PDOS sector loader.
PDOS disk format allows encrypting sectors on the disk, to successfully read those sectors one has to supply the correct decryption key.

Let's have a look at the parameters we need to pass to the loader:

D0 = Drive to read (on entry)
D0 = Error code (on exit)

D1 = Sector start
D2 = Sectors to read
D3 = Drive motor on or off after read
D4 = Serial key

A0 = Load address
A1 = MFM buffer decode address

The parameters should be quite self explanatory.

In order to create a cracked version of the game, we'll have to obtain the game data from the disks and put them on standard copyable AmigaDOS disks.

It is to be expected that the game developers arranged the data of the game in a way that makes use of PDOS's extra capacity capabilities to make the cracker's life hard (but that's what software cracking is all about, isn't it?).

First we have to think about how we will read the data from the disks after we ripped the game data and put it back onto AmigaDOS disks.

There are 2 different possibilities we may consider now:

1) We may use a byte based loader. This way we may even squeeze out some extra space and possibly still make the game fit on 3 disks. But on the contrary we have to write lots of interface and plumbing code.

2) We may use the standard RNC sector loader. This way we'll have to extend the game to use 4 disks, but positively we don't have to write any interface or plumbing code, since the RNC sector loader uses the exact same parameters as the PDOS loader does.

I decided to take the second option.
There will be a surprising discovery later on while following this route, as you'll see..

The RNC sector loader written by Rob Northen was widely used in many different commercial games.
It was well known for its robustness and efficiency and it even provided the ability to write sectors back to the disk.

Since crackers are always trying to reduce their code to the minimal size possible in order to hide it somewhere in memory where it doesn't bother the rest of the program, N.O.M.A.D. came up with a version of the RNC sector loader with its writing code stripped off.

The reduced size RNC sector loader was used for example in the cracked game Mortal Kombat from Fairlight.
You may either rip the loader by yourself from memory address $86F26 - $871E6 after the Mortal Kombat game is loaded, or grab it from Flashtro at http://www.flashtro.com/index.php?e=page&id=4044#c21751.

3 Ripping game data

Now we understand where the loaders are and how they work.
Therefore we'll use the loader at $7F800 to rip all the data from the 3 disks.

We'll start with the first disk. We know the first 2 tracks ($16 sectors) are in standard format, so let's try to rip the sectors ranging from 16 - $780 ($76A sectors).

Boot the first disk and set a breakpoint at address $7F7EE (call to loader) and $7F7F0 (right after it).
As soon as the first breakpoint is triggered, change the register contents to the following parameters:

D1 = $16
D2 = $76A
D4 = $12389A
A0 = $100000 (requires 2MB Chip RAM)

Exit AR3 the second breakpoint gets triggered rather quickly without the disk being read. Examine the register contents - register D0 is set to $1E (some error code). After several trial and error attempts, it looks like the sectors $16 - $18 are not readable for some reason, but it doesn't pose a problem since these 2 sectors are never read by the game anyway.

Reset register D0 to $0.
Set register D1 to $18 and register D2 to $768 ($18 + $768 = $780) and jump back to the loader call at address $7F7EE (command: *G 7F7EE*).
Now the loading from disk works! Wait until the second breakpoint triggers and save the data at address $100000 to 2 floppy disks (split it somewhere in the middle).

Insert disk 3. Disk layout is the same as disk 1.
You can leave the parameters as before and divert program execution flow back to $7F7EE and save the ripped data as before.

Finally insert disk 2. The disk doesn't have a boot sector and all sectors on the disk are in PDOS format.
Change D1 to $0 and D2 to $780 and rip the data as before.

Disk 1 and Disk 3:

Sectors	$0	-	$15	Regular AmigaDOS sectors	- No decryption key
Sectors	$16	-	$17	Non readable sectors	
Sectors	$18	-	$77F	PDOS encoded sectors	- Decryption key: $12389A

Disk 2:

Sectors	$0	-	$77F	PDOS encoded sectors	- Decryption key: $12389A (same decryption key used)

4 Analyze data loading

To analyze which data is being loaded from the disk, insert a breakpoint on top of the main loader at $8C038 and take note of the D1 and D2 register contents.

Depending on your gaming skills (or patience), either play the game until the end, make a level skip trainer (change at $819A6: $FFFFFFFE to $00000001) or use level passwords (http://www.whdload.de/games/Superfrog.html).

The cells marked in blue indicate the disk checks.
The cells marked in red are the disk accesses which exceed $6E0 sectors and therefore need to be moved onto our (soon) newly created disk 4.

Level 1.1		Level 1.2		Level 1.3		Level 1.4		Complete	
77E/1	D1	77E/1	D1	77E/1	D1	77E/1	D1	77E/1	D1
2C7/2F	D1	2C7/2F	D1	2C7/2F	D1	2C7/2F	D1	4F3/25	D1
37E/45	D1	37E/45	D1	37E/45	D1	37E/45	D1		
3C3/1F	D1	3C3/1F	D1	3C3/1F	D1	3C3/1F	D1		
3E2/7D	D1	3E2/7D	D1	3E2/7D	D1	3E2/7D	D1		
45F/3B	D1	45F/3B	D1	45F/3B	D1	45F/3B	D1		
49A/9	D1	49A/9	D1	49A/9	D1	49A/9	D1		
77E/1	D1	77E/1	D1	77E/1	D1	77E/1	D1		
4A3/C	D1	4B1/14	D1	4C7/12	D1	4DB/16	D1		

Level 2.1		Level 2.2		Level 2.3		Level 2.4		Complete	
77E/1	D1	77E/1	D1	77E/1	D1	77E/1	D1	77E/1	D1
2C7/2F	D1	2C7/2F	D1	2C7/2F	D1	2C7/2F	D1	6AA/31	D1
518/46	D1	518/46	D1	518/46	D1	518/46	D1		
55E/1D	D1	55E/1D	D1	55E/1D	D1	55E/1D	D1		
57B/84	D1	57B/84	D1	57B/84	D1	57B/84	D1		
5FF/3E	D1	5FF/3E	D1	5FF/3E	D1	5FF/3E	D1		
63D/4	D1	63D/4	D1	63D/4	D1	63D/4	D1		
77E/1	D1	77E/1	D1	77E/1	D1	77E/1	D1		
641/C	D1	64F/19	D1	66A/1B	D1	687/21	D1		

Level 3.1		Level 3.2		Level 3.3		Level 3.4		Complete	
77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2
0/2F	D2	0/2F	D2	0/2F	D2	0/2F	D2	228/2C	D2
B7/3A	D2	B7/3A	D2	B7/3A	D2	B7/3A	D2		
F1/1F	D2	F1/1F	D2	F1/1F	D2	F1/1F	D2		
110/7F	D2	110/7F	D2	110/7F	D2	110/7F	D2		
18F/3A	D2	18F/3A	D2	18F/3A	D2	18F/3A	D2		
1C9/5	D2	1C9/5	D2	1C9/5	D2	1C9/5	D2		
77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2		
1CE/B	D2	1DB/15	D2	1F2/15	D2	209/1D	D2		

Level 4.1		Level 4.2		Level 4.3		Level 4.4		Complete	
77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2
0/2F	D2	0/2F	D2	0/2F	D2	0/2F	D2	3D0/2F	D2
254/3E	D2	254/3E	D2	254/3E	D2	254/3E	D2		
292/23	D2	292/23	D2	292/23	D2	292/23	D2		
2B5/71	D2	2B5/71	D2	2B5/71	D2	2B5/71	D2		
326/3D	D2	326/3D	D2	326/3D	D2	326/3D	D2		
363/4	D2	363/4	D2	363/4	D2	363/4	D2		
77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2		
367/E	D2	377/1B	D2	394/15	D2	3AB/23	D2		

Level 5.1		Level 5.2		Level 5.3		Level 5.4		Complete	
77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2
0/2F	D2	0/2F	D2	0/2F	D2	0/2F	D2	543/28	D2
3FF/3B	D2	3FF/3B	D2	3FF/3B	D2	3FF/3B	D2		
43A/17	D2	43A/17	D2	43A/17	D2	43A/17	D2		
451/5A	D2	451/5A	D2	451/5A	D2	451/5A	D2		
4AB/30	D2	4AB/30	D2	4AB/30	D2	4AB/30	D2		
4DB/3	D2	4DB/3	D2	4DB/3	D2	4DB/3	D2		
77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2		
4DE/A	D2	4EA/1A	D2	506/19	D2	521/20	D2		

Project-F		Complete	
77E/1	D2	77E/1	D2
56B/1A	D2	668/29	D2
585/3C	D2		
5C1/2C	D2		
5ED/48	D2		
635/2A	D2		
77E/1	D2		
65F/7	D2		

Level 6.1		Level 6.2		Level 6.3		Level 6.4		Boss		Complete	
77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D3	77E/1	D3
0/2F	D2	0/2F	D2	0/2F	D2	0/2F	D2	6B3/35	D3	589/95	D3
691/3B	D2	691/3B	D2	691/3B	D2	691/3B	D2	6E8/1B	D3	61E/95	D3
6CC/10	D2	6CC/10	D2	6CC/10	D2	6CC/10	D2	703/2	D3	77E/1	D1
6DC/4C	D2	6DC/4C	D2	6DC/4C	D2	6DC/4C	D2	77E/1	D3		
728/3C	D2	728/3C	D2	728/3C	D2	728/3C	D2	705/4	D3		
764/6	D2	764/6	D2	764/6	D2	764/6	D2				
77E/1	D3	77E/1	D3	77E/1	D3	77E/1	D3				
709/D	D3	718/1C	D3	736/1D	D3	755/25	D3				

5 Compression format

All files being loaded by the game were packed by File Imploder.

"ATN!"	0	[Original ID: "IMP!"]	Amiga: A lot of Team 17 games
"IMP!"	0	Amiga: File Implode	Amiga: A lot of games

http://www.amiga-stuff.com/crunchers-id.html

FImp

FImp compresses a single file into the following format:

Offset	Length	Description
0x00	4	The identifying value 0x494D5021, or "IMP!" in ASCII. Clones of the FImp format use the IDs "ATN!", "BDPI", "CHFI", "Dupa", "EDAM", "FLT!", "M.H.", "PARA" and "RDC9". [1] [2]
0x04	4	The uncompressed length of the file, in bytes.
0x08	4	The offset of the following compressed data section: *endoffset*. Always even.
0x0C	*endoffset* - 0x0C	The compressed data section.
endoffset	4	Compressed data longword 3.
endoffset + 0x04	4	Compressed data longword 2.
endoffset + 0x08	4	Compressed data longword 1.
endoffset + 0x0C	4	The initial literal run length.
endoffset + 0x10	2	Bit 15 is an indicator of compressed data length; bits 7-0 are the first byte of the compressed data ("initial bit-buffer").
endoffset + 0x12	28	The explosion table (8 16-bit values and 12 8-bit values)
endoffset + 0x2E	4	Unknown; appears to be a checksum of the preceding bytes, but out by a little.

http://www.exotica.org.uk/wiki/Imploder_file_formats

6 Analysis of data access

When we analyzed data loading, you may have noticed that sector $77E is being accessed many times.
The beginning of sector $77E for disk 1 looks like this:

```
:04055C 32 32 32 32 00 00 00 00 01 F0 96 82 00 00 17 14 2222............
:04056C FF FF FF FE 07 C1 15 A4 01 F0 0C 8C 00 00 00 00 ................
:04057C 00 00 00 00 00 00 00 00 07 C0 09 70 07 C3 5A A8 ...........p..Z.
```

Sector access $77E is being used for disk identification.

```
~08BDC4 MOVE.W  #77E,D1
~08BDC8 MOVE.W  #1,D2
~08BDCC MOVE.W  #8000,D3
~08BDD0 MOVE.L  #12389A,D4
~08BDD6 MOVEA.L 0008C694,A0
~08BDDC ADDA.L  #3500,A0
~08BDE2 MOVEA.L 0008C694,A1
~08BDE8 BSR     0008C038
```

Further investigation of the code shows that there is a lookup table of all disk id's, as well as a memory location for the currently requested disk id the currently inserted disk id.

There are in total 3 calls to the loader at $8C038 coming from different locations:

```
FA 8C038
Search from: 000000 to: 100000
08BD34 JSR      0008C038
08BDE8 BSR      0008C038
08C02C BSR      0008C038
Searched up to adr: 0FAC84
Ready.
```

Call from $8BD34 is used during the gambling game after each level.

Call from $8BDE8 is used for disk identification purposes.

Call from $8C02C is used for loading game data.

7 Reconstructing disk images

To reconstruct the disk images, we'll load all ripped data (per disk) into memory and write it back to disk with AR3.
Remember that at the beginning of sector $6DE we have to store the disk key for disk identification purposes by the game.
$6DE is the second to last sector on our cracked disks, like $77E is the second to last sector on the original disks.
$EBC00 = $10000 + $6DE * $200

Disk 1:
LM D1_FIRST2TRACKS,10000
LM D1_DATA,12C00+400 (sectors $16 + $17 are empty)
M EBC00 -> Set memory to key: $32 $32 $32 $32
WT 0 !160 10000

Disk 2:
LM D2_DATA,10000
M EBC00 -> Set memory to key: $33 $33 $33 $33
WT 0 !160 10000

Disk 3:
LM D3_FIRST2TRACKS,10000
LM D3_DATA,12C00+400 (sectors $16 + $17 are empty)
M EBC00 -> Set memory to key: $31 $31 $31 $31
WT 0 !160 10000

The data exceeding the maximal size of standard DOS disks, will be put on our new disk 4 in the next step..

8 Additional disk

We create a new disk and label it "Disk 4".
The new disk will be split into 3 approximate same sized areas:

Sectors	000 - 585	-> For excess sectors from disk 1	($0 - $249)
Sectors	586 - 1171	-> For excess sectors from disk 2	($24A - $493)
Sectors	1172 - 1757	-> For excess sectors from disk 3	($494 - $6DD)
Sector	1758	-> Disk 4 id	($6DE)

Let's take a closer at the disk accesses for the final part of the game (level 6, boss, end sequence):

Level 6.1		Level 6.2		Level 6.3		Level 6.4		Boss		Complete	
77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D2	77E/1	D3	77E/1	D3
0/2F	D2	0/2F	D2	0/2F	D2	0/2F	D2	6B3/35	D3	589/95	D3
691/3B	D2	691/3B	D2	691/3B	D2	691/3B	D2	6E8/1B	D3	61E/95	D3
6CC/10	D2	6CC/10	D2	6CC/10	D2	6CC/10	D2	703/2	D3	77E/1	D1
6DC/4C	D2	6DC/4C	D2	6DC/4C	D2	6DC/4C	D2	77E/1	D3		
728/3C	D2	728/3C	D2	728/3C	D2	728/3C	D2	705/4	D3		
764/6	D2	764/6	D2	764/6	D2	764/6	D2				
77E/1	D3	77E/1	D3	77E/1	D3	77E/1	D3				
709/D	D3	718/1C	D3	736/1D	D3	755/25	D3				

The blue cells indicate the disk checks. Per level there are 2 disk checks and in between them there are several files being loaded.

Contrary to the previous version of the table, there are now also yellow cells. The yellow cells indicate file accesses which we will duplicate onto the new disk 4, even they don't exceed $6E0 sectors.

Why? Because this way we can keep the structure of "disk check-load files-disk check" as it is, which is easier than restructure the whole thing and there is plenty of space on our new disk anyways.

Disk 1 Area:

$000 - $24A: You may move some files here if you need to free up some space for a Cracktro/Trainer.

Disk 2 Area:

$24A - $279: $000 / $02F (Duplicate)
$279 - $2B4: $691 / $03B (Duplicate)
$2B4 - $2C4: $6CC / $010 (Duplicate)
$2C4 - $310: $6DC / $04C
$310 - $34C: $728 / $03C
$34C - $352: $764 / $006

Disk 3 Area:

$494 - $4C9: $6B3 / $035 (Boss)
$4C9 - $4E4: $6E8 / $01B (Boss)
$4E4 - $4E6: $703 / $002 (Boss)
$4E6- $4EA: $705 / $004 (Boss)
$4EA - $4F7: $709 / $00D (Level 6.1)
$4F7 - $4F9: Empty gap
$4F9 - $515: $718 / $01C (Level 6.2)
$515 - $517: Empty gap
$517 - $534: $736 / $01D (Level 6.3)
$534 - $536: Empty gap
$536 - $55B: $755 / $025 (Level 6.4)

Finally add the disk id key $35 $35 $35 $35 at sector $6DE for disk 4 the same way we did it for disks 1-3.

9 Patch game

Disk 1 and disk 3 are both bootable.
Let's start with disk 3, which contains the story intro.

Patching disk 3:

Boot the game and instantly enter AR3. The boot block will be located at $5C40.

```
~005C4C MOVE.L  A1,-(A7)
~005C4E MOVE.L  #1000,D0
~005C54 MOVE.L  #10002,D1
~005C5A MOVEA.L 00000004.S,A6
~005C5E JSR     -C6(A6)
~005C62 MOVEA.L D0,A3
~005C64 MOVEA.L (A7)+,A1
~005C66 TST.L   D0
~005C68 BEQ     00005CA8
~005C6A MOVEA.L A1,A2
~005C6C MOVE.L  A3,28(A1)
~005C70 MOVE.L  #1000,24(A1)
~005C78 MOVE.L  #400,2C(A1)
~005C80 MOVE.W  #2,1C(A1)
~005C86 MOVEA.L 00000004.S,A6
~005C8A JSR     -1C8(A6)
~005C8E MOVEA.L A2,A1
~005C90 MOVE.B  1F(A1),D0
~005C94 BNE     00005C6C
~005C96 MOVEA.L 00000004.S,A6
~005C9A JSR     -84(A6)
~005C9E MOVEA.L 00000004.S,A6
~005CA2 JSR     -96(A6)
~005CA6 JMP     (A3)
```

Jump to $A498

```
~00A498 BSR     0000A69C
~00A49C LEA     A68E(PC),A1
~00A4A0 MOVE.L  #400,(A1)
~00A4A6 MOVE.W  #4000,00DFF09A
~00A4AE MOVE.L  #7FFF7FFF,00DFF09A
~00A4B8 LEA     000003F4,A0
~00A4BE MOVE.L  #1000000,(A0)+
~00A4C4 MOVE.L  #1800000,(A0)+
~00A4CA MOVE.L  #FFFFFFFE,(A0)
~00A4D0 MOVE.L  #3F4,00DFF080
~00A4DA CLR.W   00DFF180
~00A4E0 MOVE.W  #81D0,00DFF096
~00A4E8 LEA     A68A(PC),A0
~00A4EC MOVEA.L (A0),A0
~00A4EE ADDA.L  #7D0,A0
~00A4F4 LEA     A744(PC),A2
~00A4F8 MOVE.L  A0,(A2)
~00A4FA LEA     A732(PC),A2
~00A4FE MOVE.L  A2,00000080
~00A504 TRAP    #0
~00A506 LEA     A68A(PC),A0
~00A50A MOVEA.L (A0),A0
~00A50C ADDA.L  #FA0,A0
~00A512 MOVEA.L A0,A7
~00A514 LEA     A68A(PC),A1
~00A518 MOVE.L  A0,(A1)
~00A51A CLR.L   D0
~00A51C MOVE.W  #18,D1
~00A520 MOVE.W  #60,D2
~00A524 MOVE.W  #8000,D3
~00A528 MOVE.L  #12389A,D4
~00A52E LEA     000519D0,A0
~00A534 LEA     00070000,A1
~00A53A BSR     0000A9DA
~00A53E LEA     000519D0,A0
~00A544 LEA     000519D0,A1
~00A54A BSR     0000A780
~00A54E LEA     000519D0,A0
~00A554 LEA     000519D0,A1
~00A55A BSR     0000A8E0
~00A55E MOVEA.L A68E(PC),A0
~00A562 MOVEA.L A68A(PC),A1
~00A566 MOVEA.L A686(PC),A2
~00A56A JMP     000519D0
;---------------------------
~00A570 CLR.W   D0
~00A572 MOVE.W  #18,D1
~00A576 MOVE.W  #8000,D3
~00A57A MOVE.L  #12389A,D4
~00A580 LEA     00070000,A0
```

The first loader on disk 3 reads $60 sectors starting from sector $18.
The actual call to the loader takes place at address $A53A.

We'll replace the old loader with our standard RNC sector loader:

RT 0 1 10000
A 10942 → MOVEQ #0,D4
LM RNC_LOADER,10944
WT 0 1 10000

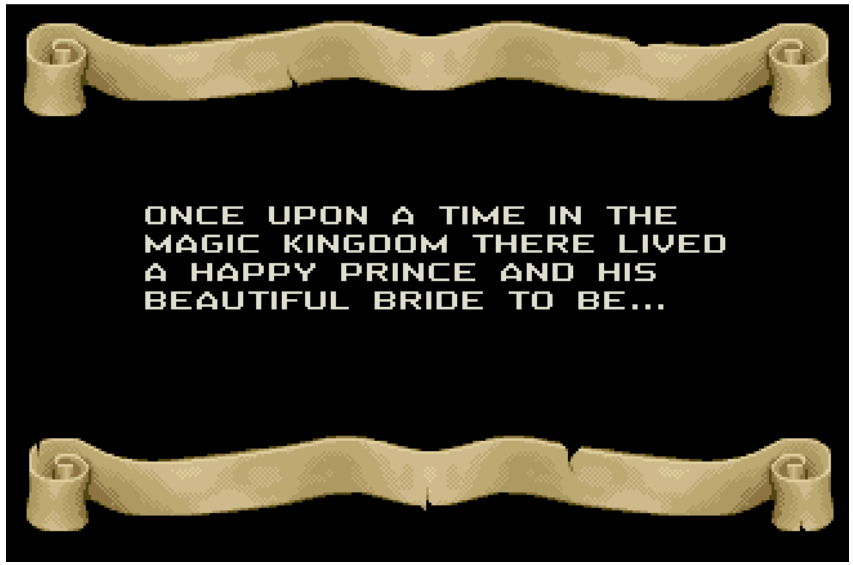
Let's see what happens after we restart:

Ok the logo gets loaded and then further loading gets stuck..
Most likely there's another loader inside the loaded file that is supposed to take over.

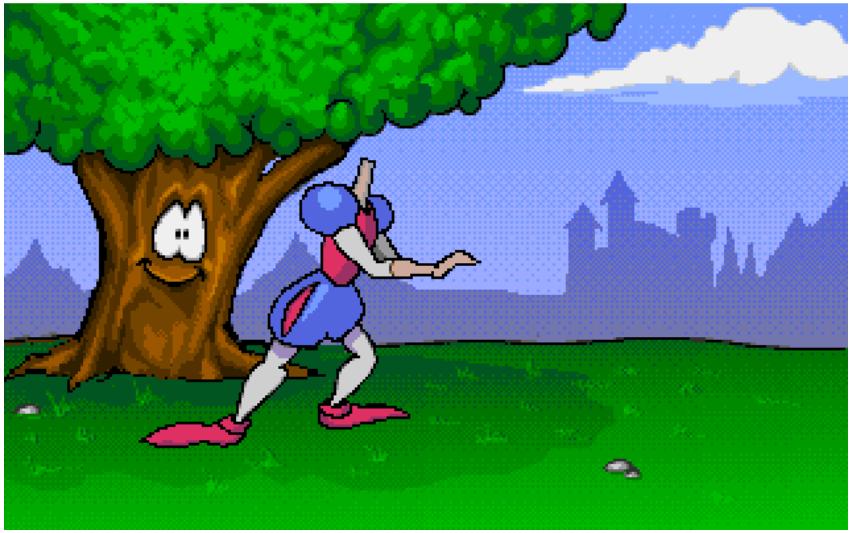
Since the loaded file is crunched, we'll have to:

- Replace the loader at $7CEC0 (like the first loader) right after it was decrunched at address $A54A
- Rip the patched file from $519D0 to $519D0+$2C740=$7E110
- Crunch it with Imploder (FImp)
- Write the file back to disk:
  - *RT 2 9 10000*
  - Replace crunched file at $10400
  - Change signature of crunched file from "IMP!" to "ATN!"
  - *WT 2 9 10000*

Let's give it another try.
Looks good, loading progresses..



But wait a minute, the static background pictures are loaded but all the animated parts are missing..
Also when we observe the track counter, we see that it tries to load more files at different locations, but fails doing so.



Sigh.. Looks like there's yet another loader waiting to be replaced..

Ok when we'll keep on following the code from $519D0, we'll soon end up at the following place, where we already should recognize the pattern:

```
~07CDC4 MOVE.L  #7D9DC,00DFF080
~07CDCE CMPI.B  #FF,00DFF006
~07CDD6 BNE     0007CDCE
~07CDD8 CMPI.B  #20,00DFF006
~07CDE0 BNE     0007CDD8
~07CDE2 BSR     0007CE7A
~07CDE6 BSR     0007CE64
~07CDEA BSR     0007D4EC
~07CDEE TST.W   0007D7EE
~07CDF4 BEQ     0007CDE2
~07CDF6 MOVE.W  #11D,D1
~07CDFA MOVE.W  #212,D2
~07CDFE MOVEA.L 7DCB0(PC),A0
~07CE02 LEA     000799F6,A1
~07CE08 BSR     0007CE94
~07CE0C MOVE.W  #78,D1
~07CE10 MOVE.W  #A5,D2
~07CE14 MOVEA.L 7DCAC(PC),A0
~07CE18 LEA     000799F6,A1
~07CE1E BSR     0007CE94
~07CE22 MOVEA.L 7DCB0(PC),A0
~07CE26 MOVEA.L 7DCB0(PC),A1
~07CE2A BSR     0007DCC4
~07CE2E LEA     7DA80(PC),A0
~07CE32 LEA     000799D6,A1
~07CE38 MOVE.L  #10,D0
~07CE3E BSR     0007D60E
~07CE42 BSR     0007CE7A
~07CE46 BSR     0007CE64
~07CE4A BSR     0007D6E8
~07CE4E TST.W   0007D7EE
~07CE54 BEQ     0007CE42
~07CE56 MOVE.L  #7DCB8,00DFF080
~07CE60 BRA     0007DC84
```

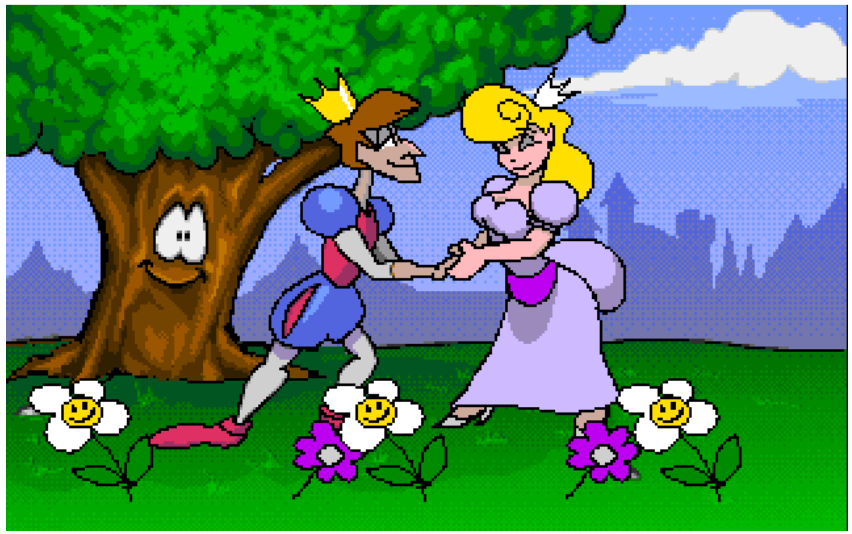
A0 = $80FA0

A0 = $400

Since the loaded file is crunched (again!), we'll have to:

- Replace the loader at $8D42E (like previous loaders, first instruction *MOVEQ #0,D4*) right after it was decrunched at address $7CE2A
- Patch instruction: $8D368 *MOVE.W #77E,D1 → MOVE.W #6DE,D1*
- Rip the patched file from $80FA0 to $80FA0 + $53C7C = $D4C1C
- Crunch it with Imploder
- Write the file back to disk:
  - *RT !25 !49 10000*
  - Replace crunched file at $11400
  - Change signature of crunched file from "IMP!" to "ATN!"
  - *WT !25 !49 10000*

Give it another try.
Looks much better.. The animations and disk loading are working now.



After we learned why our hero turned into a frog, we will be prompted to insert disk 1.

The game provides 2 ways of being booted, either via disk 3 or directly via disk 1 without the story intro.
Therefore we have to make sure we make disk 1 properly bootable as well.

Patching disk 1:

Track 1 ($B - $16) contains the first loader we have to replace on the first game disk.
Since the file containing the loader is packed with Imploder, we'll have to replace the loader in memory after it was unpacked and then right away grab it from memory, repack it and finally write it back to disk.

Load in the first track from disk and change first jump to jump to itself:

RT 0 1 10000

```
~01005E MOVEA.L 00000004.S,A6          ~01005E MOVEA.L 00000004.S,A6
~010062 JSR     -96(A6)          →     ~010062 JSR     -96(A6)
~010066 JMP     (A3)                   ~010066 BRA     00010066
```

BOOTCHK 10000

WT 0 1 10000

Restart Amiga and when being stuck, change the instruction back and trace until you arrive at this place:

```
~00A51A CLR.L   D0
~00A51C MOVE.W  #B,D1
~00A520 MOVE.W  #B,D2
~00A524 MOVE.W  #8000,D3
~00A528 LEA     0007C180,A0
~00A52E LEA     00070000,A1
~00A534 BSR     0000A9CC
~00A538 LEA     0007C180,A0
~00A53E LEA     0007C180,A1
~00A544 BSR     0000A772
~00A548 LEA     0007C180,A0
~00A54E LEA     0007C180,A1
~00A554 BSR     0000A8D2
~00A558 MOVEA.L A680(PC),A0
~00A55C MOVEA.L A67C(PC),A1
~00A560 MOVEA.L A678(PC),A2
~00A564 JMP     0007C180
```

Set a breakpoint at address $A548 after the file was unpacked.

Go to the load address of $7C180 and scroll down until you encounter the loader:

```
~07F820 MOVEM.L D1-D7/A0-A5,-(A7)
~07F824 LINK    A6,#FFFFFFDE
~07F828 MOVE.W  D0,D5
~07F82A ANDI.W  #3,D5
~07F82E MOVE.W  D5,-22(A6)
~07F832 MOVE.W  D1,-20(A6)
~07F836 MOVE.W  D2,-1E(A6)
~07F83A MOVE.W  D3,-1C(A6)
~07F83E MOVE.L  D4,-12(A6)
~07F842 MOVE.L  A0,-1A(A6)
~07F846 MOVE.L  A1,-16(A6)
~07F84A ROR.W   #2,D0
~07F84C ANDI.W  #1,D0
```

As usual change the first instruction at $7F820 to set the key to zero and then replace the loader at the following instruction.

A 7F820 -> MOVEQ #0,D4
LM RNC_LOADER,7F822
SM D1_B_B_UNPACKED, 7C180 7C180+3AF4

-= PACK FILE WITH IMP =-

RT 1 1 10000
LM D1_B_B_PACKED, 10000
M 1000 -> Change signature from "IMP!" to "ATN!"
WT 1 1 10000

Now restart the Amiga and we'll notice that we're stuck at the following screen:

Load the main file we ripped during the boot process analysis to $80FA0 - $2C = $80F74 (remember $2C is the hunk header size).
By loading the file into this memory address, we may use the addresses like after the hunk processing done at $7F4EC.
Therefore patching can be done directly without unnecessary searching and offset calculations.

Extend the lookup table for the disk id's with the following value for disk 4:

```
:08BEF4 32 32 32 32 31 31 31 31 32 32 32 32 33 33 33 33 2222111122223333
:08BF04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:08BF14 35 35 35 35 00 00 00 00 00 00 00 00 00 00 00 00 5555............
:08BF24 2F 09 4E B9 00 00 AF 98 22 5F 20 49 4E B9 00 00 /.N.....".. IN...
:08BF34 CE 58 4E 75 4A 79 00 00 00 08 66 00 00 44 61 00 .XNuJy....f..Da.
```

Next step is to replace the old PDOS loader with our RNC loader.
At $8C038 we'll put a *MOVEQ #0,D4* to always pass the key $00 $00 $00 $00 to the loader (we don't want to patch every single call to the loader, so we apply the patch centrally).

Then load the RNC loader to address $8C03A - $8C2FA.

```
~08C038 MOVEQ   #0,D4
~08C03A MOVEM.L D1-D7/A0-A6,-(A7)
~08C03E LINK    A2,#FFFFFFD0
~08C042 LEA     00DFF000,A6
~08C048 LEA     00BFD100,A5
~08C04E LEA     8BFEC(PC),A4
~08C052 MOVE.L  A0,-2C(A2)
~08C056 MOVE.L  A1,-30(A2)
~08C05A MOVE.W  D0,-20(A2)
~08C05E MOVE.W  D1,-1E(A2)
~08C062 MOVE.W  D2,-1C(A2)
~08C066 ADD.W   D1,D2
~08C068 CMP.W   #6E0,D2
'''
```

The following 3 routines are used by the game to select the disk id where the data should be loaded from:

```
~08BD5E MOVEM.L D0-D7/A0-A6,-(A7)
~08BD62 LEA     0000AF58,A0        ------------------>  $31 $31 $31 $31 (Disk 3 (Story))
~08BD68 BRA     0008BD84
!=====================================
~08BD6A MOVEM.L D0-D7/A0-A6,-(A7)
~08BD6E LEA     0000AF5C,A0        ------------------>  $32 $32 $32 $32 (Disk 1)
~08BD74 BRA     0008BD84
!=====================================
~08BD76 MOVEM.L D0-D7/A0-A6,-(A7)
~08BD7A LEA     0000AF60,A0        ------------------>  $33 $33 $33 $33 (Disk 2)
~08BD80 BRA     0008BD84
```

Now we need a place where we can put the same routine for our disk 4, but this time pointing to the new disk id $35 $35 $35 $35.
Ideally we put it just behind the replaced loader since there is still enough space being occupied by the original loader.

```
~08C2FA MOVEM.L D0-D7/A0-A6,-(A7)
~08C2FE LEA     0008BF14,A0
~08C304 BRA     0008BD84
```


$35 $35 $35 $35 (Disk 4)

Since standard format disks contain $6E0 sectors and not $780 sectors like PDOS disks, we have to patch the location where originally sector $77E is read for obtaining the disk id. We patch it to read sector $6DE:

```
~08BD84 CMPI.W  #2,00080FA8
~08BD8C BNE     0008BEEE
~08BD90 MOVEA.L 0008C694,A1
~08BD96 CLR.L   3500(A1)
~08BD9A MOVE.L  (A0),0008BEF4
~08BDA0 CLR.L   D7
~08BDA2 LEA     0008C668,A5
~08BDA8 MOVE.L  D7,D6
~08BDAA ADD.L   D6,D6
~08BDAC ADD.L   D6,D6
~08BDAE ADDA.L  D6,A5
~08BDB0 TST.L   (A5)
~08BDB2 BNE     0008BE30
~08BDB6 MOVE.L  D7,-(A7)
~08BDB8 MOVEA.L 0008C694,A1
~08BDBE CLR.L   3500(A1)
~08BDC2 MOVE.L  D7,D0
~08BDC4 MOVE.W  #6DE,D1
~08BDC8 MOVE.W  #1,D2
~08BDCC MOVE.W  #8000,D3
~08BDD0 MOVE.L  #12389A,D4
~08BDD6 MOVEA.L 0008C694,A0
~08BDDC ADDA.L  #3500,A0
~08BDE2 MOVEA.L 0008C694,A1
~08BDE8 BSR     0008C038
```

In the same routine we'll need to extend the functionality to recognize our new disk 4.
Otherwise it would keep on searching for disk 4 in an endless loop.

```
~08BDEC MOVEA.L 0008C694,A6
~08BDF2 CMPI.L  #31313131,3500(A6)
~08BDFA BEQ     0008BE12
~08BDFC CMPI.L  #32323232,3500(A6)
~08BE04 BEQ     0008BE12
~08BE06 CMPI.L  #33333333,3500(A6)
~08BE0E BEQ     0008BE12
~08BE10 BRA     0008BDB8
```

Patch to →

```
~08BDEC MOVEA.L 0000B6F4,A6
~08BDF2 CMPI.L  #31313131,3500(A6)
~08BDFA BEQ     0008BE12
~08BDFC CMPI.L  #32323232,3500(A6)
~08BE04 BEQ     0008BE12
~08BE06 CMPI.L  #33333333,3500(A6)
~08BE0E BRA     0008C308
```

Add functionality ↓

```
~08C308 BEQ     0008BE12
~08C30C CMPI.L  #35353535,3500(A6)
~08C314 BEQ     0008BE12
~08C318 BRA     0008BDB8
```

When we keep on scrolling down a bit, we come across this interesting piece of code:

```
~08BE94 CLR.L   D0
~08BE96 MOVE.B  0008BEF4,D0
~08BE9C CMPI.B  #31,D0
~08BEA0 BNE     0008BEAA
~08BEA2 MOVE.L  #2,D0
~08BEA8 BRA     0008BEAE
;====================================
~08BEAA SUBI.B  #32,D0
~08BEAE MOVEA.L 0008C68C,A0
~08BEB4 JSR     (A0)
~08BEB6 MOVE.L  #73D8,00DFF080
~08BEC0 BSR     0008C698
~08BEC4 MOVEM.L (A7)+,D0-D7/A0-A6
~08BEC8 BRA     0008BDA0
;====================================
```

→ Requested disk id (first byte)

→ Disk 3 id $31 special handling

→ D0 = $2 - Disk 3

→ D0 = $0 - Disk 1 / $1 - Disk 2 / $3 - Disk 4

→ A0 = $41D5C

```
~041D5C BSR     000420A8
~041D60 MOVE.L  #420BE,00DFF080
~041D6A MOVEM.L D0-D7/A0-A6,-(A7)
~041D6E CMPI.W  #0,D0
~041D72 BNE     00041D7A
~041D74 LEA     0004CC72,A1
~041D7A CMPI.W  #1,D0
~041D7E BNE     00041D86
~041D80 LEA     0004CC84,A1
~041D86 CMPI.W  #2,D0
~041D8A BNE     00041D92
~041D8C LEA     0004CC96,A1
~041D92 CMPI.W  #3,D0
~041D96 BNE     00041D9E
~041D98 LEA     0004CCA8,A1
~041D9E MOVE.L  A1,00041E9C
~041DA4 BSR     00041DEA
~041DA8 BTST    #7,00BFE001
~041DB0 BNE     00041DA8
```

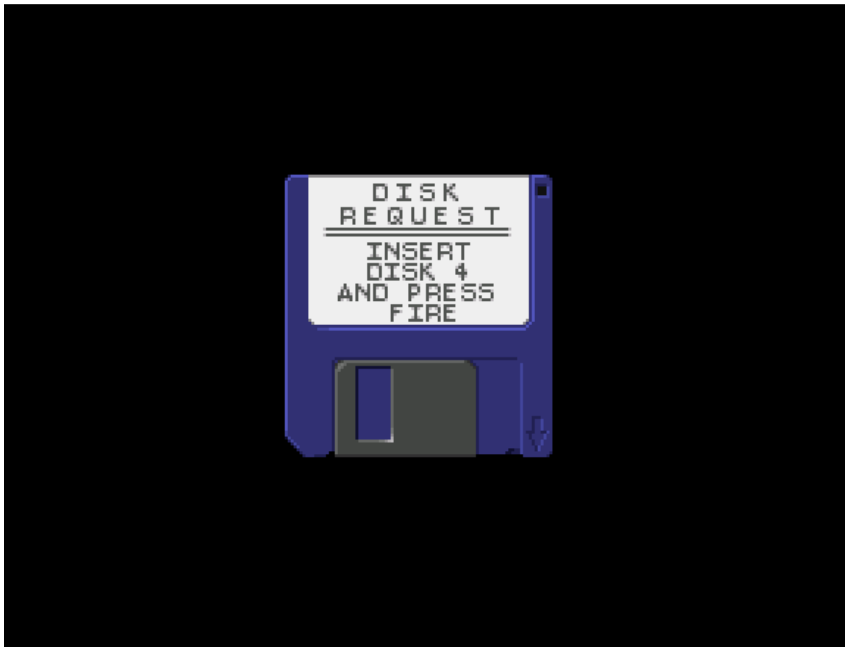

Load address for disk 1 request screen

Load address for disk 2 request screen

Load address for disk 3 request screen

Load address for disk 4 request screen

Wait a minute.. There is a request screen for disk 4?



Cool.. Looks like the developers didn't bother to remove it before mastering the game :)

Now we come to the final step of fixing the disk accesses.

All levels 6.1 to 6.4, the boss stage and end sequence will be loaded from disk 4.
The swap requests from disk 2 and disk 3 to disk 4 have to be fixed.
The individual data accesses which would otherwise exceed the disk capacity have to be patched as well.

Level	Disk Check	Disk (Old)	Disk (New)	Sector / Amount (Old)	Sectors occupied (New)	Memory location
Level 6.0:	$81614	2	4	$000 / $02F (Loaded from $8B022) $691 / $03B $6CC / $010 $6DC / $04C $728 / $03C $764 / $006	$24A - $279 $279 - $2B4 $2B4 - $2C4 $2C4 - $310 $310 - $34C $34C - $352	$8DB98 (add manually)* $8DDBA $8DDC0 $8DDC6 $8DDCC $8DDD2
Level 6.1:	$8B112	3	4	$709 / $00D	$4EA - $4F7	$8DBC2
Level 6.2	$8B1B4	3	4	$718 / $01C	$4F9 - $515	$8DBC8
Level 6.3	$8B256	3	4	$736 / $01D	$517 - $534	$8DBCE
Level 6.4	$8B2F8	3	4	$755 / $025	$536 - $55B	$8DBD4
Boss:	$81706	3	4	$6B3 / $035 $6E8 / $01B $703 / $002	$494 - $4C9 $4C9 - $4E4 $4E4 - $4E6	$8DBAA $8DBB0 $8DBB6
Boss:	$8B4A8	3	4	$705 / $004	$4E6 - $4EA	$8DBBC

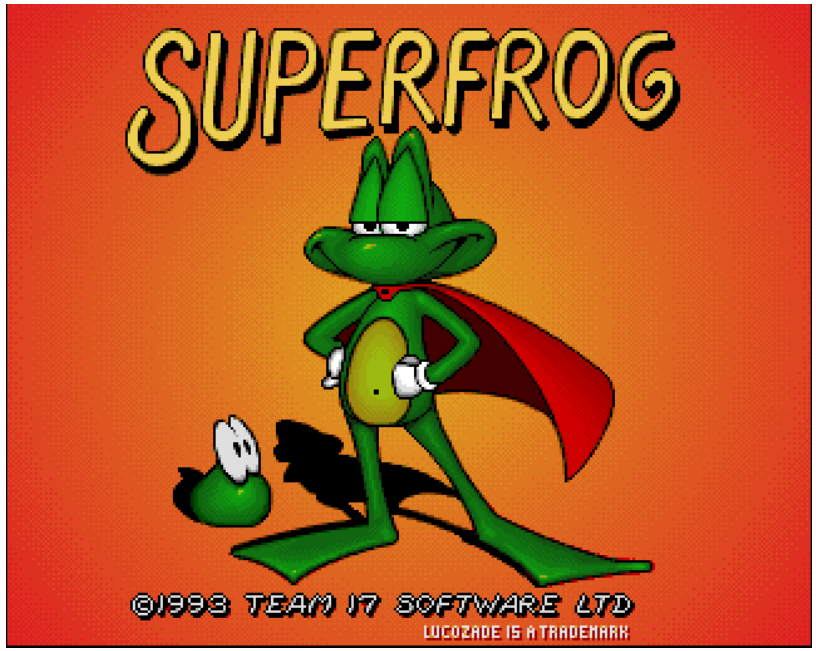
* $8DB98 - $80FA0 = $CBF8 (relative address)

Patch all above disk checks to: JSR B35A (relative address $8C2FA - $80FA0 = $B35A)
Search in $8DB9E area for sector/amount pair and patch accordingly for example: 06 91 00 3B becomes 02 79 02 B4

Alright we're almost done with the crack:

- Grab the main file from memory
- Pack the file with Imploder
- *RT 5 !30 10000*
- Replace patched and packed file at $10200
- Change signature from "IMP!" to "ATN!"
- *WT 5 !30 10000*

Et voilà, we're done with our Superfrog crack!



Greetings to the awesome Flashtro community!

scenex 2016