



Shockwave Flash & Director Overlays, the Armadillo Aspect

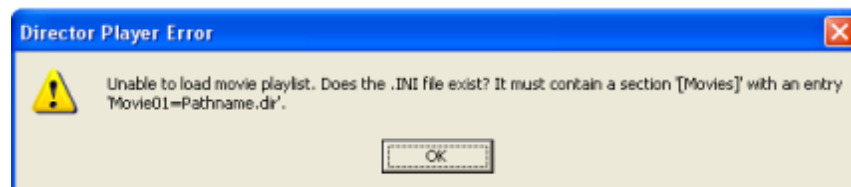
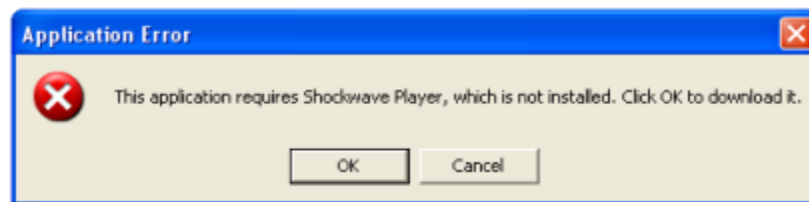
Version 1.0

Last Rev. July 2007

1. Introduction

Let me start by saying that this is not going to be a tutorial on unpacking Armadillo protected applications, it is more intended to give a little understanding to why some games don't work **after** unpacking them.

If you have unpacked a game and been greeted with one of the following messages, then you have encountered these before and are most probably wondering what went wrong, even confused:



There are also ones which, when unpacked, will result in a SWF player that loads showing a blank screen with a file menu on the top toolbar. These allow you to manually load Shockwave Flash files, both locally and from the internet, but are missing the original movie that was integrated into the original file.

Editor: Shub-Nigurrath

Without further ado, lets see what is happening...

Happy Reversing,
Ghandi

Disclaimers

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This tutorial is also free to distribute in its current unaltered form, with all the included supplements.

All the commercial programs used within this document have been used only for the purpose of demonstrating the theories and methods described. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched, and cracked versions were available since a lot of time. ARTeam or the authors of the paper cannot be considered responsible damages the companies holding rights on those programs. The scope of this tutorial as well as any other ARTeam tutorial is of sharing knowledge and teaching how to patch applications, how to bypass protections and generally speaking how to improve the RCE art. We are not releasing any cracked application.

Verification

ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: <http://releases.accessroot.com>

Table of Contents

1.	Introduction.....	1
1.	Abstract	3
2.	Tools used, skills required.	3
3.	Targets	3
4.	Understanding the overlay mechanism	3
4.1.	Armadillo and these overlays	4
5.	Restoring the overlay to an unpacked file.....	5
5.1.	Director Type A.....	5
5.2.	Director Type B Games.....	7
5.3.	Shockwave Flash Games	9
6.	Removing the Armadillo Sections	9
7.	OpenMutexA 'trick'	11
8.	References	11
9.	Conclusions	12
10.	Greetings/Thanks	12

1. Abstract

In a nutshell, we're going to see how this mechanism works and after we understand it, we will see what is needed to be done to correct the issues that arise when unpacking these kinds of files. This isn't really a hard thing to do, not after you know what is going wrong, so put on your thinking caps and read on...

2. Tools used, skills required.

OllyDbg

LordPE

Any good hex editor

Basic Armadillo unpacking experience

3. Targets

Because there are different types of overlays, i have chosen 3 targets for this tutorial.

Director Type A – Hidden Expedition: Everest (BigFish Games version) *Note: The installer is silent, no dialogs!*
<http://downloads.bigfishgames.com/downloads/F2002T1L1.exe>

Director Type B – Carrie the Caregiver (Gamenext/Oberon version) *Note: The installer is in German!*
http://www.gamenext.de/exe/Carrie_the_Caregiver-setup.exe

Shockwave Flash – Solitaire Pop (PlayFirst version)
<http://www.playfirst.com/game/solitairepop/download/windows>

4. Understanding the overlay mechanism

I haven't figured out a way to tell whether a file is a Director or Shockwave prior to unpacking, unless you can take the file information displayed by Windows as an indicator.

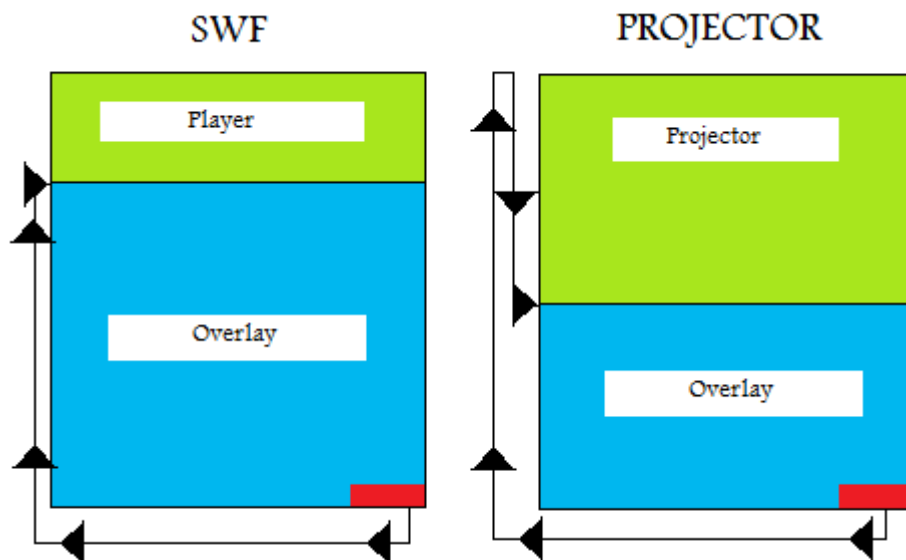
The example Shockwave game will have **Macromedia Flash Player v8.0**, whereas the Director examples merely have **Macromedia Projector**. There is a big difference between the two overlay types, Flash are relocatable and can be attached to the end of any size file, whereas Director files need to be treated differently if you want to modify them.

Both types, however, have as their primary index (*my terminology, it will suffice for now.*) a DWORD value that is stored as the last 4 bytes of the file. When the player/projector is run, the process will open its file with read-access, and set the file pointer to the last 4 bytes of the file. Then the 4 bytes are read and used to set the file pointer again and another 4 bytes are read, then compared with the overlay signature(s) it is capable of playing.

If there is not a recognized signature found, it will then take appropriate action, whether that be launching the player in standalone mode (Flash) or attempting to load the .INI file (Director). In the case of Director files, if there is no .INI file found, the projector will display the appropriate error messages before exiting.

Here's where they branch away from each other: Director projectors use the DWORD value read from the end of the file to set the file pointer from the start of the file (FILE_BEGIN), Flash players use it to set the file pointer from the end of the file (FILE_END). Director overlays closely follow the RIFF/RIFX format, they are a collection of files and directories, with each directory containing more raw offsets to the files contained within. Changing the location of the overlay would mean that ALL the offsets would need to be adjusted to reflect this.

Here's a visual of that, just in case you are confused, sorry for the graphics, I'm not much of an artist:



4.1. Armadillo and these overlays

Armadillo doesn't leave the overlay in the same location for various reasons, such as:

1. The actual size of the overlay isn't a constant, nor is the end size of a projector due to icons and other resources, so leaving it at the correct location for Director files would be out of the question. Armadillo adds sections (file bloat) too, which would change the location of the overlay. Armadillo by default inserts its sections (basically merging two executable files) between the .rsrc section and the section preceding it. If this were changed, then it would be possible to add its sections AFTER the overlay, as long as the all important DWORD value was copied to the last 4 bytes of the file.
2. Leaving the overlay unprotected would defeat the whole purpose of protecting the file. If this were the case, we could just extract the overlay and attach it to another player/projector (as Deroko's tutorial shows.) thus bypassing Armadillo without the need to unpack it at all.

I know you're asking now: "But if the overlay isn't where it needs to be, then how does the protected file still access it at all?"

Well, the Armadillo authors have taken this into account, to allow people to use their product with Macromedia applications.

The overlay is stripped from the executable, stored in another location (an Armadillo section) and memory is allocated at runtime, then the overlay is inserted into this memory. Armadillo hooks **_lopen**, **_lseek**, **_lread** and **_lclose** so that when the projector needs to access itself on disk to read this data, it can simply return the pointer to this memory and the projector will function happily, oblivious to the fact it has been altered. (SWF players use other file i/o API, the principle is the same.)

Now, lets see how we can then restore the unpacked files so they can run as they were originally intended, once again!

Because the portals require the protection to be compatible with as many programs as possible (read: profit), they can't really use protection options that are too severe, so this makes the files relatively easy to unpack...

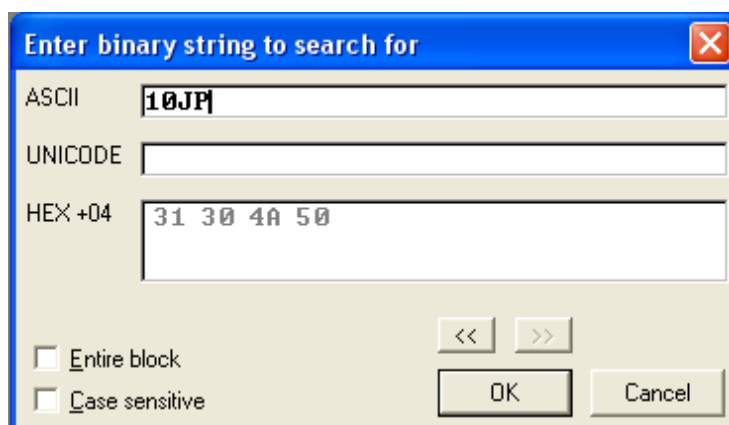
5. Restoring the overlay to an unpacked file.

As i stated in the introduction, I'm not showing how to unpack Armadillo. If you have read the tutorial this far and don't know how to unpack Armadillo protected files, i suggest that you put this paper down and go study some tutorials that cover that part. ARTeam has some excellent tutorials for this, plus Teddy Roger's site 'Tuts4You' has a good range that can show you the required skills so you can do this part. Unpack the targets and use LordPE to wipe the Armadillo section headers, then rebuild the file so it is a valid executable once again, see the chapter at the end of this article if you are unsure how to do so.

We'll tackle Hidden Expedition: Everest first, for no particular reason other than i wanted to. ;)

5.1. Director Type A

Open the protected file with OllyDbg and bypass Debug-Blocker (OpenMutexA trick) so this remains a single process. After this, breakpoint CreateThread and run the target until it hits your breakpoint, go to the memory view and do a binary search for the ASCII string **10JP**:



This is the signature for Director v10 overlays, it has been pointed out to me by Nacho_DJ that Director v6 overlays have the signature bytes **59JP**, so a wildcard search would be ??JP at +20h bytes. You should get a hit:

```

311C0020 31 30 4A 50 59 E1 27 00 9C 27 00 00 40 00 00 00 10JPVp'.e'..@...
311C0030 04 00 00 00 04 00 00 00 24 41 01 00 00 50 02 00 4...$A0..P0.
311C0040 70 72 6F 6A 00 00 00 00 00 00 00 00 00 00 00 00 proj.....$#...
311C0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .dll.dll....$#...
311C0060 00 64 6C 6C 00 64 6C 6C 00 00 00 00 24 91 03 00 .0..dirapi.....
311C0070 00 E0 16 00 64 69 72 61 70 69 00 00 00 00 00 00 .dll.dll....
311C0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 $q+...'..iml32...
311C0090 00 00 00 00 00 64 6C 6C 00 64 6C 6C 00 00 00 00 .....dll.dll....
311C00A0 24 71 1A 00 00 60 09 00 69 6D 6C 33 32 00 00 00 $q+...'..iml32...
311C00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....dll.dll....
311C00C0 00 00 00 00 00 00 00 00 00 64 6C 6C 00 64 6C 6C .....dll.dll....
311C00D0 00 00 00 00 24 D1 23 00 35 10 04 00 6D 73 76 63 ...$0#.5$...msvc
311C00E0 72 74 00 00 00 00 00 00 00 00 00 00 00 00 00 00 rt.....dll
311C00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 64 6C 6C .dll.....
311C0100 00 64 6C 6C 00 00 00 00 00 00 00 00 00 00 00 00 .....
311C0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
311C0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
311C0130 00 00 00 00 00 00 00 00 00 00 00 00 00 4F 00 00 .....
311C0140 00 00 00 00 4D 5A 90 00 03 00 00 00 04 00 00 00 ....MZE.♥...
311C0150 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 ..@.....@...
311C0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
311C0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
311C0180 F8 00 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C °...$||$.+.=+00L
311C0190 CD 21 54 68 69 73 20 70 72 6F 67 72 61 6D 20 63 =!This program c
311C01A0 61 6E 6E 6F 74 20 62 65 20 72 75 6E 20 69 6E 20 annot be run in
311C01B0 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 DOS mode....$...
311C01C0 00 00 00 00 CF 8F 26 38 80 FF 48 68 80 FF 48 68 $A0$-Hv$-Hv

```

You can see the names of the dll's contained in the overlay, plus the beginning of the first dll's file header, the MZ signature and the DOS stub. Go back to the memory view and right click the region, select Dump Memory-Area to dump that region of memory to a temporary file, we'll need to clean it a little before attaching it to the unpacked projector. Once you have dumped the region, close OllyDbg and open our overlay in the hex editor, then delete the first 20h bytes:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	50	00	9A	00	50	00	9A	00	00	00	00	00	00	00	00	00	; P.Š.P.Š.....
00000010h:	00	F0	CB	00	00	F0	CB	00	36	0C	00	00	00	0B	00	00	; .šĚ..šĚ.6.....
00000020h:	31	30	4A	50	59	E1	27	00	9C	27	00	00	40	00	00	00	; 10JPYá'.œ'..@...
00000030h:	04	00	00	00	04	00	00	00	24	41	01	00	00	50	02	00	;\$A...P..
00000040h:	70	72	6F	6A	00	00	00	00	00	00	00	00	00	00	00	00	; proj.....
00000050h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000060h:	00	64	6C	6C	00	64	6C	6C	00	00	00	00	24	91	03	00	; .dll.dll....\$'..
00000070h:	00	E0	16	00	64	69	72	61	70	69	00	00	00	00	00	00	; .à..dirapi.....
00000080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
-----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	

They were only in use when this was a memory region, but now they're no longer needed. Now, the overlay file begins at the signature bytes **10JP**:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	31	30	4A	50	59	E1	27	00	9C	27	00	00	40	00	00	00	; 10JPYá'.œ'..@...
00000010h:	04	00	00	00	04	00	00	00	24	41	01	00	00	50	02	00	;\$A...P..
00000020h:	70	72	6F	6A	00	00	00	00	00	00	00	00	00	00	00	00	; proj.....
00000030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000040h:	00	64	6C	6C	00	64	6C	6C	00	00	00	00	24	91	03	00	; .dll.dll....\$'..
00000050h:	00	E0	16	00	64	69	72	61	70	69	00	00	00	00	00	00	; .à..dirapi.....
00000060h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000070h:	00	00	00	00	00	64	6C	6C	00	64	6C	6C	00	00	00	00	;dll.dll....
00000080h:	24	71	1A	00	00	60	09	00	69	6D	6C	33	32	00	00	00	; \$q...`..iml32...
00000090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
000000a0h:	00	00	00	00	00	00	00	00	00	64	6C	6C	00	64	6C	6C	;dll.dll

Go to the end of the file and scroll upwards until you reach the beginning of the NULL bytes:

00cbe380h:	42	4A	4A	42	4A	4A	42	4A	41	4b	4c	41	3b	00	00	00	; 50050050AFLAb...
00cbe390h:	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	; ù.ù.ù.ù.ù.ù.ù.ù.
00cbe3a0h:	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	; ù.ù.ù.ù.ù.ù.ù.ù.
00cbe3b0h:	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	F9	00	; ù.ù.ù.ù.ù.ù.ù.ù.
00cbe3c0h:	F9	00	F9	00	F9	00	00	40	01	00	00	00	00	00	00	00	; ù.ù.ù..@.....
00cbe3d0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00cbe3e0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00cbe3f0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00cbe400h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;

We need to allow a single NULL byte, because a projector file isn't going to be 01400000h bytes in size, so selecting from the 2nd NULL byte **after** the 01, delete the rest of the file. Now your overlay ends on the 1st NULL byte, like so:

```

00cbe340h: 4A 42 4A 4A 42 4A F9 00 07 29 31 29 31 29 31 29 ; JBJJBjJù..) 1) 1) 1)
00cbe350h: 31 F9 31 07 4A 4A 42 4A 4A 42 4A 4A F9 00 07 31 ; 1ù1.JJBjJBjJù..1
00cbe360h: 29 31 29 39 29 31 29 F9 31 07 4A 42 4A 4A 42 4A ; ) 1) 9) 1) 1) 1) 1) 1) 1) 1)
00cbe370h: 4A 42 F9 00 07 29 31 29 31 29 31 29 31 F9 31 07 ; JBù..) 1) 1) 1) 1) 1) 1)
00cbe380h: 42 4A 4A 42 4A 4A 42 4A 41 46 4C 41 36 00 00 00 ; BJBjJBjJBjAFLA6...
00cbe390h: F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 ; ù.ù.ù.ù.ù.ù.ù.ù.ù.
00cbe3a0h: F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 ; ù.ù.ù.ù.ù.ù.ù.ù.ù.
00cbe3b0h: F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 F9 00 ; ù.ù.ù.ù.ù.ù.ù.ù.ù.
00cbe3c0h: F9 00 F9 00 F9 00 00 40 01 00 ; ù.ù.ù..@..

```

We're nearly there. ;) Save the 'cleaned' overlay in case anything goes wrong while doing the next step, then open the dumped projector executable and go to the end of the file. Take notice of the last bytes offset and add 1 byte to include it in the count, then subtract that value from the DWORD ptr at the end of our overlay. This is the amount of bytes we need to pad the file to:

```

00012f90h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00012fa0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00012fb0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00012fc0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00012fd0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00012fe0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00012ff0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....

```

Fixed dump file (with added import section):	00013000h
RAW offset required:	00014000h
Bytes required for padding:	00001000h

Pad the file with the necessary NULL bytes, then copy & paste the overlay data to it making sure it is EXACTLY at the offset that is at the end of the overlay. Save the file and you are ready to test your newly repaired Macromedia game!

5.2. Director Type B Games

This is the German version of Carrie the Caregiver, i only found out about this second type of Director file from a request for help in the forum. This kind has the .DCR (movie) file in the game directory, so at first glance it appears it isn't using any overlay at all... Unpack the file however, repair imports and try to run it... You will get an error message telling you that the .INI file is nowhere to be found.

The most basic Director .INI file that you can write contains:

```

[Movies]
Movie01=moviename.dcr

```

If you were to try creating this INI file alone though, the Projector will load but display a different message:

'Expired or unauthorized.'

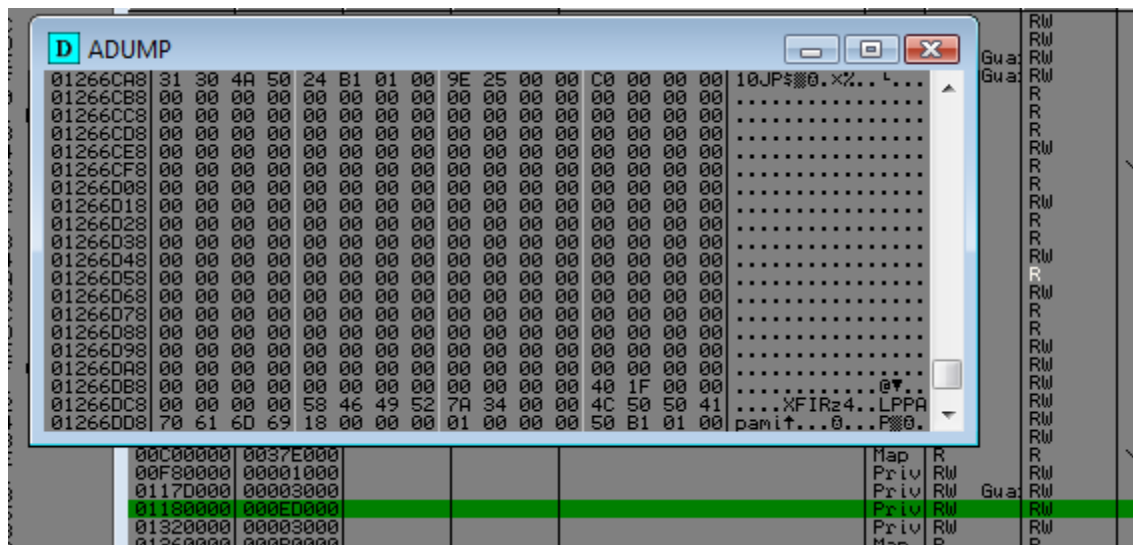
Okay... It seems that part of the overlay is also the license file for games. When the Projector can't find this data, it will display the appropriate message and go no further. Before you go running off to look for the overlay at +20h in all memory sections, I'll tell you that it isn't going to be found there... :)

Instead, Armadillo has taken this data and stored it in the .PDATA section (encrypted). When the application is executed, it is decrypted and stored in a section of memory where it has also stored some resources for the virtual ArmAccess.dll (Security.dll).

So, load this one into OllyDbg and take it to OEP, then go to the Memory window of OllyDbg. Perform a binary search for the overlay signature i showed you in the first example: **10JP**, as this is also a Director v10 game.

(You'll have to excuse the dialog looking different, i upgraded to Vista halfway through writing this tutorial... ;))

Can you see that the location is NOT at the +20h offset, but nearly at the end of this section?:



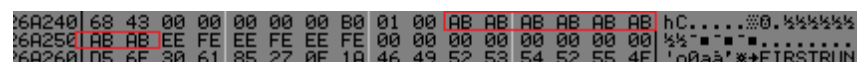
This is the beginning of the overlay data, note it's address and then scroll down to the end of this section. If you care to look on your way down, you will see the RIFX references, they are all stored in reverse ascii:

LPPA = APPLiCation
pami = image map
pamm = memory map
tsiL = List
tciD = Dictionary
eliF = File
tSAC = CASt
YEK = KEY <--- The most important ones, i think...

Anyways, back to repairing this game... Looking at the end of the section, it would be easy to assume that the last DWORD there is the correct DWORD offset (00 30 01 00), but it isn't. I was actually lucky enough not to get this one by trial and error! Instead, the FE EE FE EE made it seem that there was other data stored on the end of the overlay in memory... Scroll back upwards until you see the Armadillo string '**PROTECTEDFILE=**':



Then look for the bytes that **usually** indicate the memory was allocated but not zero initialized: AB AB AB AB...



If you examine the memory immediately above them, you'll see some more RIFX file type signatures. In the DWORD **just** before the AB AB AB AB bytes, you'll find your overlav RAW offset:

00 B0 01 00 = 0001B000

Taking the Virtual Address that this DWORD ends on, subtract the Virtual Address noted before, where the overlay begins:

$$0126A24A - 01266CA8 = 35A2$$

This is the size of the overlay, so you can either dump the entire section and calculate the offset where the overlay begins and copy the data to a new hex file, or (providing the memory area is accessible to LordPE) perform a partial dump of that section so you have just the overlay. Once you have the overlay dumped, follow the same steps as the first example to reattach it to its correct position: 00013000. Job's almost done and the file is repaired!

Lastly, open notepad and create an .INI file with the contents:

```
[Movies]
Movie01=loader.dcr
```

Save it with the same filename as your fixed dump, but give it the .INI extension and close notepad, then test your fixed file, it should run without a problem.

5.3. Shockwave Flash Games

Now it's time to tackle the final example, but instead of leading you through it, I'll just explain that the DWORD value at the end of the overlay is NOT the RAW offset it needs to be at! (I explained this before, but i'm just making sure you know;))

The example game is SolitairePop from PlayFirst games, so you'll need to debug the DRM executable and retrieve at least one valid serial so you can unpack the game file. You can follow my tutorial i have written about this subject if you need to. After you have a valid serial, it's the same process to unpack PlayFirst SWF games as it is for other companies that aren't using the 'No Default Certificate' option when protecting their game releases.

Here's a summary of what to do for SWF game's protected with Armadillo:

1. Unpack executable file at it's OEP.
2. Remove unnecessary sections if desired. (Not a requirement, just cleaner if you do.)
3. Recover imports and repair dumped executable.
4. Search for SWF/SWC file signature bytes at +20h in each memory section: "CWS" or "FWS".
5. Dump overlay and remove excess bytes from beginning and end of dumped data.
- OR
6. Use DWORD immediately after signature DWORD as it's size and dump ONLY that many bytes.
7. Reattach overlay to the unpacked executable, ensuring final DWORD value is the overlay size.

That's all that is required! As a side-note, it is also possible to dump the overlay and give it the SWF file extension, then load it with an SWF player. If you use the unpacked one to do this, you can even choose the file menu option: '**Create Projector**'. This will attach the file to a SWF player as an attachment, exactly as if it were meant to be there but minus the correct icon, which can be changed with a resource editor such as ResHacker.

Another thing was pointed out to me by Nacho_Dj, it appears that ShockWave Flash v6 files have the signature bytes **CWS** and v7+ have the signature bytes **FWS**. A way to see what version the SWF file was created with is the byte immediately after the signature, it is the version number.

6. Removing the Armadillo Sections

When unpacking Armadillo files, you are faced with the choice of removing unnecessary sections. These redundant sections were used by Armadillo when the file was protected, but in removing the protection, you also negated the need for them to be there at all. (in MOST cases.) There are tools available to automate this task, but i will settle for the middle ground and show you a semi-manual way of doing this. (It is not really that hard to do it 100% manually with a hex editor, but that is for another time ;))

First, unpack and dump your executable but don't fix imports. It is important to NOT let LordPE realign file when you are dumping, because we will do that when we've removed the Armadillo sections and i have had it corrupt some file if they are realigned multiple times! So, in LordPE's rebuilder options, uncheck realign before performing

your dump. Now, load your dump file into LordPE's PE editor and go to the section table editor (Click the 'sections' button). These Armadillo protected executable files will have the following sections added to them, in between the last section of the original file and it's previous section:

1. **.text1**
2. **.adata**
3. **.data1**
4. **.pdata**

So, the ones we are using in this tutorial go from:

1. **.text**
2. **.rdata**
3. **.data**
4. **.rsrc**

to

1. **.text**
2. **.rdata**
3. **.data**
4. **.text1**
5. **.adata**
6. **.data1**
7. **.pdata**
8. **.rsrc**

We can remove the section headers and LordPE will happily change the sizes of the 2nd last section to compensate for this change, thus keeping the sections contiguous. But this presents a problem when trying to repair Director files, as a dumped file will still have a massive size difference if compared to its original state. Like, 91kb vs 13-14MB, depending on the size of the encrypted overlay it contains in the .pdata section.

Now you have the file loaded into the PE editor, one by one, select then right click the Armadillo sections and select 'Wipe Section Header' from the context menu. DO NOT SELECT TRUNCATE, AS THIS WILL CUT THE FILE AND LOSE THE RSRC SECTION!!!

When you have wiped the 4 Armadillo sections, close the PE editor for a second and go back to LordPE's options. Now you can check the 'Realign > Nice' options from the rebuilder section. Done that? Close the options dialog and use the Rebuild button to fix your dump file. If you have done this correctly, the file size will change dramatically and the .rsrc section will now be at the RAW offset the 1st Armadillo section used to occupy. The file should still be a valid executable and it's time to repair your imports with whatever method you are using.

7. OpenMutexA 'trick'

Debug-Blocker is one of the protection measures which Armadillo tries to use as a defense against debugging/reversing a protected application. Andreageddon's paper on Armadillo 4.20 covers the mechanism pretty well, but I'll give a brief explanation here, plus a way to circumvent this 'problem'. Please note that this can not be used for programs that are also using the CopyMemll protection!

When a protected application is launched, it will try to open a mutex. *(The mutex name is comprised of the processes PID and a 'random' hexadecimal value, but this is unimportant.)* Applications using the Debug-Blocker feature will attempt to open the mutex **twice**, applications not using Debug-Blocker only **once**. The outcome of this call to OpenMutexA determines whether the process is either 'Parent' or 'Child', because the mutex doesn't exist until the 'Parent' process creates it.

When the call returns NULL (ERROR_FILE_NOT_FOUND), the calling process will then use CreateMutexA to create the mutex, then launch the 'Child' process. Now that the mutex has been created, any subsequent calls to OpenMutexA will return a handle, rather than NULL.

At least that is the theory, but we can change that. ;)

Open Hidden Expedition's executable file with OllyDbg and use Ctrl+G to open the 'Go to Expression' dialog, then enter OpenMutexA and hit enter. We won't use the starting point of this routine, instead scroll down and set a breakpoint on the return from it: **RET 0C**. Now press Shift+F9 so the process runs without halting on unknown commands. It should break at your **RET 0C** breakpoint, with EAX being NULL.

This first call isn't the one which determines Parent/Child. It is okay to leave the return value as it is, so press Shift+F9 again and you will get another break at your **RET 0C** breakpoint. This second break will have exactly the same mutex name *(look in the stack, if it doesn't then your executable isn't using Debug-Blocker or you missed the first call somehow.)* and if you are curious enough you could trace into the return to see where it branches off, Parent/Child.

Simply change EAX to a non-NULL value, anything will do, and clear your breakpoint on OpenMutexA's RET 0C. Now, you can continue to unpack/debug the process as a non-Debug-Blocker protected file.

Like i said above though, this approach won't work with CopyMemll, instead causing the process to crash. I have yet to find out why this occurs; perhaps the Parent does some other preparation for the child?

8. References

'An Armadillo DRM Tutorial (Part 1)' by Ghandi

Explores the DRM used by PlayFirst games, shows how to retrieve valid serial keys for unpacking PlayFirst games, how to find the Armadillo trial information for nearly any protected application.

'Armadillo and Shockwave Games' by Deroko of ARTeam

Alternative approach to unpacking overlays, including injecting a dll and coding a projector.

'Armadillo 4.20, Removing the armour – A naked mutant' by AndreaGeddon of RETeam

Detailed exploration of Armadillo 4.20, the methods and information are still mostly current.

9. Conclusions

Macromedia games protected with Armadillo, although needing a little more work, are quite easy to unpack and repair, once you have an idea of their mechanics. As pointed out in Deroko's tutorial covering this subject, it is not necessary at all to unpack the protected file and reattach the overlay, you can also create your own projectors and players! I won't go into that further, it is for Deroko's tutorial to explain. ;)

Another thing that Deroko's tutorial shows is an alternative method for locating and extracting the overlay data, perhaps quicker and more universal. By calling the hooked file i/o API from within the protected program's context, Armadillo will return the precise location of data it is protecting. Once again, I shall leave that to Deroko's tutorial to explain, which it does quite well.

Hopefully, after reading this tutorial, you walk away with a little more RCE knowledge and are armed with the necessary information to allow you to work with these types of files!

10. Greetings/Thanks

All my friends at ARTeam forums, ARTeam themselves (you guys rock!), MR_BLAH, Gunna35, Deroko, Nacho_Dj and last but not least YOU the reader. Without you to read these tutorials, there would be no point in creating them. I have learned more by trying to share the knowledge, both in trying to explain things clearly and properly, plus re-examining my work to make sure it is correct. Long live RCE!

Ghandi
ARTeam 2007

