

Patching a DVD firmware to make it region-free.

by xvi (xvi91@hotmail.com).

Important notice

Use this document at your own risk.

Patching firmwares is a non-trivial operation that may cause permanent damage to your drive.

Version History

1.0 08-JUL-2001 First release.

Goal

This document tries to explain the specifics about patching a DVD firmware to make it region-free. It will give you or point you to required information.

Non goal

This document isn't a tutorial about patching. If you want to learn about patching, you're on the wrong way. Find appropriate material elsewhere.

Prerequisites

You will need to be experimented in some assembler language patching. Of course, DVD drives generally don't use the same processor as the one you're accustomed to patch, so you will need to learn another processor. If you know an assembly language, you'll have little to learn to understand another processor's instruction set and architecture. Registers are found in every processor, and instructions have the same basic features: you can add and subtract two values, you can compute the logical AND or OR of two values, call a subroutine and return from that subroutine to the caller, etc. To patch a DVD firmware, you don't need to know everything about the target processor, but at least you will need its architecture and instruction set description.

You must know hexadecimal as most numbers in this document are listed in hex form.

You must have some hex editor to view, search and edit the binary file.

Some knowledge about SCSI or ATAPI commands won't hurt.

How to use this document

This document is quite dense. Some important technical informations are given without further explanation. I suggest that you first use this document with a firmware that already has been patched, understanding the patch that was done, until you feel more familiar with such firmwares. Get both the original and patched files, compare them, analyze the original, and check that the patch works the way it should. If needed, repeat the same job on another region-free'd firmware, until you feel ready to go on your own.

Some may want to skip this step and immediately analyze and patch a firmware.

A word about SCSI and ATAPI

DVD drives exist in two flavors, SCSI and ATAPI. This will make no difference for patching them, as they both use the same command packets to communicate with the host. You may think of ATAPI as "SCSI over an IDE bus".

Both work the same way: the host sends a command packet that is a few bytes long, possibly followed by some bytes of parameters, and gets the answer bytes (if any). Most command packets are 6, 10 or 12 bytes long, and the first of these bytes

contains the code of the operation that the drive is asked to perform.

Some DVD specific commands

ATAPI/SCSI commands description can be found at:

<http://ftp.t10.org/ftp/t10/drafts/>

DVD specific commands are located in the mmc3/ directory.

You probably should fetch and read the latest mmc3 description, at least the "Report Key" and "Send Key" commands (opcodes A3 and A4), because these are the two commands that include the region-locking mechanism (and we will patch them). Here is a short reminder of both command packets:

Byte \ Bit	7	6	5	4	3	2	1	0
0	Operation code (A3)							
1	Reserved		Reserved					
2	Reserved							
3	Reserved							
4	Reserved							
5	Reserved							
6	Reserved							
7	Reserved							
8	Parameter List Length							
9								
10	AGID	Key Format						
11	Control							

Send Key command packet

Byte \ Bit	7	6	5	4	3	2	1	0
0	Operation code (A4)							
1	Reserved		Reserved					
2	Reserved or Logical Block Address							
3								
4								
5								
6	Reserved							
7	Key Class							
8	Allocation Length							
9								
10	AGID	Key Format						
11	Control							

Report Key command packet

The "Key Format" parameter for the "Send Key" command (A3) can take the documented values 01, 03, 06, 3F and sometimes undocumented values, such as 02 that is frequently used for the vendor reset command. The command packet is followed by a few bytes of parameters.

The "Key Format" parameter for the "Report Key" command (A4) can take the documented values 00, 01, 02, 04, 05, 08, 3F, on CPRM compliant devices, the documented value 11 and sometimes undocumented values. The drive returns some information bytes to the host when it receives this command.

Of particular interest are the Send Key command with Key Format set to 06, and the Report Key command with Key Format set to 04 or 08. These will be henceforth referred as the A3/06, A4/04 and A4/08 commands.

The A4/08 command is used to query the drive's region and counters, the A3/06 command is used to change the drive's region. The A4/04 command returns the Title Key, which is needed to decrypt a CSS-encrypted movie.

Here is how the drive reply to the A4/08 command is formatted:

Byte \ Bit	7	6	5	4	3	2	1	0
0	00							
1	06							
2	00							
3	00							
4	State	Resets left			Changes left			
5	Region Mask							
6	01							
7	00							

Report Key Format 08 reply

The 'Changes left' field contains the number (from 0 to 5) of region changes that still can be done. The 'Resets left' field contains the number (from 0 to 4) of vendor resets that still can be done.

The 'State' field that will depend on the 'Changes left' field. If the 'Changes left' is 0, the 'State' will be binary 11. If the 'Changes left' is 1, the 'State' will be 10, if the 'Changes left' is 2, 3 or 4, 'State' will be 01, and when 'Changes left' is 5, 'State' is 00. The official name of these 4 states are 'None', 'Set', 'Last Chance', 'Permanent'.

The Region Mask format

The Region Mask is a format used to represent a region. It is made of a single byte, whose bit 0 corresponds to region 1, bit 1 to region 2, up to bit 7 that maps to region 8. Each bit can take two values: true and false. 'True' is represented by a 0 bit, while 'false' is represented by a 1 bit. Pay attention that this is *opposite* to common habits.

The A3/06 and A4/04 commands use the region mask format to represent the drive's region. The region mask is built by setting to 'true' the region you want to select, so that region 1 is represented by the FE value, region 2 by the FD value, etc.

Here are the most common values of a Region Mask:

No Region	Region 1	Region 2	Region 3	Region 4	Region 5	Region 6	Region 7	Region 8
FF	FE	FD	FB	F7	EF	DF	BF	7F

Note the 'No Region' corresponding to value FF. When return by the drive as its current region, it means that no region has been set yet. The factory defaults are: No Region set, 5 changes left, 4 vendor resets left. A vendor reset will reset the region change counter to 5, the region mask to FF, and will decrement the vendor resets count. A region change will change the region mask and decrement the changes left counter. The drive will reject region change requests if the changes left counter is 0.

As a DVD drive can only be set to one region simultaneously, A3/06 commands with region mask with multiple regions selected (such as 00) will be rejected.

The region mask is also used by movies to specify in which region they may be played. If a region's bit is true (i.e. 0), it may be played by a drive set to that region. This region authorization mask is stored in the sector header of the infamous CSS disc key sector, buried deep in the DVD's internal structure. A copy of this region mask is accessible at hex offset 23 of the VIDEO_TS.IFO file that is in the VIDEO_TS folder (or directory) of a movie.

Differences between a region-free and a region-locked drive

A **region-free drive** always will report the Title Key to the host, and, as it has no region, it doesn't know about the A3/06 or A4/08 command and cannot report neither change its region. When a A3/06 or A4/08 command is issued to a region-free drive, it will report an "INVALID FIELD IN CDB" error meaning that the issued Key Format parameter was unrecognized. Some region-free drives may report other errors, such as "MEDIUM NOT PRESENT" when no movie is inserted. Anyway, region-free drives won't report nor change their region settings.

A **region-locked drive** will be set to one and only one region, and will remember this region even after being powered down. It will honor the A4/08 command by returning its current region and counters, it will also honor the A3/06 command by changing its region (if the region change counter hasn't been decremented down to 0). The A4/04 command has a slightly modified behaviour, as it won't report the Title Key if the movie isn't authorized for playback in the drive's region. If the movie is authorized for the drive's region, the A4/04 command is honored normally. If the movie isn't marked as playable in a drive's region, the player software will miss the Title Key, so won't be able to decrypt nor play it.

Your computer will detect that your drive is region free by sending it the A4/08 command. If the drive replies with the region settings, the drive is region-locked. If the drive aborts the command with error, the drive is region-free. All utilities that check your drive's RPC status work this way.

How do region-locked drives remember their region and counters?

Of course, they all use some kind of non volatile memory, but what value do different drives store in that memory?

Different drives may know their regions by different means. Some store the region mask, other store the region by its number (1, 2, 3, etc), and other means can possibly exist.

Counters can also be remembered in two ways. They can be stored as countdown, representing the number of operations left, decremented at each region change or vendor reset. They also can count from 0 upward, representing the number of times the operation as been done.

These different ways of storing the region and counters yield different algorithms to manipulate them, so you will have to analyze the code that handles these values to find out the way they are stored.

How does a region-locked drive determine if it can "play" a movie?

In fact, a drive doesn't disallow reading any DVD, it only refuses to return the CSS Title Key of unauthorized region's movies, so the host is unable to decrypt the movie. To check if it can return the Title Key of a movie, the drive's firmware will check the movie's region mask bit corresponding to its own region. Depending on how the drive remember its own region,

different algorithms can be used.

If the drive remember its region mask, it can simply compute the binary OR of its region mask and the movie's one, if the result is FF, the movie is not authorized, if the result is not FF, the movie is authorized. This operation is easy to identify in an assembly listing as it is built of an OR instruction, followed by a COMPARE, and then a EQUAL/NOT EQUAL CONDITIONAL BRANCH.

If the drive remember its region by its number, it can compute its region mask and then check region compatibility as described just above, or it can test the specific bit of the movie's region mask. The later is less obvious in a firmware listing, but is still easily identifiable.

Note: Region 7 and 8 may or may not be recognized by the drive, as they are special cases among regions: region 7 is currently not defined, and region 8 is a special international region, rarely (if ever) used.

The patching process will now be described. Remember, this is an exemple, and some firmwares may require additional steps.

Patching, step 1

First of all, get a firmware to patch. A common source of firmware is a firmware file extracted from a firmware update kit published by the drive's manufacturer. If the firmware comes in .HEX or .MOT format, it must be translated into binary form. Doing this is beyond the scope of this document, and utilities are available all over the net. Beware that .HEX files can hold more than 64 KB of data, and some conversion utilities aren't able to handle these large files.

Patching, step 2

You must now disassemble the firmware file. This implies that you must know which processor the drive uses. If you don't know, here is a list of the most common processors used in DVD drive: Hitachi H8S, Intel 8051, Zilog Z80, Matsushita 102000, Mitsubishi 7902, Hitachi SuperH. Try to disassemble the file according to each of this processors, and find out which one is used by looking at the assembly listing, one and only one should make sense.

Sources for H8S, 102000, 790, SuperH, 8051 and Z80 disassemblers are available at:

<http://perso.club-internet.fr/farzeno/firmware/>

Raw disassembly listing are enough, you don't need an intelligent disassembler that will identify data and code sections, a brute force disassembly of all bytes of the firmware is fine for patching a firmware. The above disassemblers are such raw disassemblers. I have found them even more useful than intelligent disassemblers and they may be customized to handle different firmware file format. Your mileage may vary.

Patching, step 3

Now that you disassembled the firmware, you'll need to find what and were to patch. First, you need to identify where the drive's region is stored. The easiest way to achieve this goal is to find the A4/08 handler that returns the drive's region mask to the host.

Common firmware are 64 KB, 128 KB or even larger, so finding the right code area in several tens of thousand assembly lines may seem like a hard job. Hopefully, there is a particular operation that has always been a faithful helper: the format parameter of the A3 and A4 operations must be extracted from the byte it shares with the AGID parameter. This is commonly done by ANDing the byte with 3F. A "regular expression" search for "AND.*3F" (or a similar syntax adapted to the target processor) will yield a few code area candidates (of course, you need a text editor that supports regular expression searches). Among these candidates, you can find out the two that corresponds to the A3 and A4 commands by examining the code that follows. The different format options will either be checked one by one, comparing them with the different values that it can take, either by indexing into a table. Both methods are quite easily identified, and by checking the different values that are checked, you will quickly find the two command handlers. Note that the table dispatching or successive compares are frequently preceded by other parameter's validity check. The AGID for example, as it hasn't to be valid with the particular formats that we are looking for; so the A4 handler might be identified by an immediate compare of the ANDed value with 08.

Now that you found the A4 handler, follow the code for the 08 Key Format. You'll end up at the A4/08 handler and you will find the code that returns the region. Note how the region is stored (by its mask or number, etc) and its address.

This step is the most important step in patching a firmware because this is where you first pry into the firmware. From this initial code area analysis, most things can be inferred.

Patching, step 4

Now let's look for all references to the drive's region (or region mask) in the firmware. We must try to identify all of these, at least know what they do with the drive's region. Possible uses are (all are not present in every firmware):

- Initialization. At startup, the drive reads the non-volatile memory to restore the last region that was remembered. Some firmware reads the region from NVRAM or Flash memory every time it is needed, so this code may be called from several locations.
- Defaulting to no region set. The first time it is powered on (during the manufacturing process), the drive's non-volatile memory is uninitialized, so the firmware enforces the following settings: "no region selected", "5 changes left", "4 vendor resets left".
- Storing the region. When the region is changed, it must be remembered and written to non-volatile memory.
- Exact compares. These comparisons are followed by conditional branches whose condition is either "EQUAL" or "NOT EQUAL". Such a comparison is needed as part of the A3/06 handler, when the host has requested the last region change, that is, there is exactly one change left (or four changes done). In this case, to prevent user from trivial errors, it is mandatory to have a movie inserted in the drive that is authorized for the target region, and for that region only.
- Unexact compares. These are region compatibility checks. Some are part of the A3/06 handler as a drive will only change its region if a movie that is authorized for that region is inserted. Some are part of the region locking scheme, e.g. in the A4/04 handler that checks if returning the Title Key is allowed.

Patching, step 5

Enough rambling, it's now time to patch!

First, modify the A3 handler so that it doesn't recognize the 06 Key Format code. You have to patch a conditional branch and/or the table that is looked up.

Then, do the same for the A4 handler so that it won't recognize the 08 Key Format code.

Now, patch the region compatibility checks so that they ever match.

You might also want to modify the firmware name, if so, beware that it may appear also byte-swapped on ATAPI devices, as an ATAPI drive has two ways of being "inquired": the ATAPI inquiry command packet, and the old-style IDE identification. The IDE identification string may be stored byte-swapped, so that the firmware named "ABCD" may also appear as "BACD" somewhere else in the firmware file. Renaming patched firmwares is a matter of personal choice. Some patchers do, some patchers don't. What's important is that the patched firmware is region-free, not its name.

Some tricks I didn't tell you yet...

The above patching process is idealized. The real thing can be a bit harder (or easier):

- Anti-patching techniques: I have found firmwares (Matsushita DVD-RAM drives) that call a subroutine *outside* the firmware area to check the region compatibility (as it's outside of the flashed firmware area, it's unpatchable). I found other firmwares that contain a hidden checksum (Toshiba 1502/2502) and make the drive eject the medium when the checksum mismatches. Both are pathetic attempts to hinder patching. I understand that vendors are tied to RPC-2 by the CSS license agreement, but I can't imagine why they would want to sell less drives by preventing them to be region-free'd.
- Checksums: Most drives will check uploaded firmware before flashing it. The firmware file can contain headers and/or checksums. You will have to find the checksum algorithm in the firmware itself, then make a little program to recompute a patched firmware file's checksum. This kind of checksums are not part of any anti-patching conspiracy, but acts as a safety mechanism to avoid flashing a drive with a corrupt firmware which could damage the drive.
- Some firmwares are already RPC-1 able. In such firmwares, a check is made to determine if the drive is RPC-1 or RPC-2. Look for such tests in the A4/08 handler code area. If you find such a test, it can either check if a jumper is set or removed, it can check a hard-coded flag somewhere in the firmware, or a flag set by an undocumented command, etc.
- When you found the A4/08 handler, do you remember you found that ANDing with 3F? The source byte was taken from the eleventh byte of the command buffer. Subtract 0A from its address to compute the address of the byte that holds the operation code. Search references to that byte to find the command dispatcher, then look for undocumented opcodes or functions (mostly in the C0-FF range, but also in vendor-specific MODE PAGES or DIAGNOSTICS PAGES). Some may be very interesting to analyze, such as commands that dump the drive's memory (about every drive), reset the change and vendor reset counters (Hitachi GD-7000 and 7500), or even switch the drive to RPC-1 temporary (LG drives) or permanently (Hitachi GD-3000 and GD-5000).

- Removing the A3/06 command from the firmware is maybe superfluous, as the drive will be identified as RPC-1 if the A4/08 command has been removed. I've seen patched firmware where this command was still present, and they worked fine. However, as the A3/06 command is part of the RPC-1/RPC-2 differences, I suggest you remove it nonetheless. It won't take much time, and it's safer.
- Some firmwares check if a region is set before checking for region compatibility (and if none is yet set, the command aborts with error), so either patch that code too, or ensure a region has been set to the drive before updating it with the patched firmware. In either case, it is a good idea to ensure a region has been set to the drive before applying a patched firmware. This is why I frequently set this as a prerequisite before flashing a drive.
- Some people may be tempted to patch a firmware by simply removing the change counter decrement of the A3/06 command (a.k.a. the *infinite lives* patch, in reference to games patched so that your lives aren't decremented anymore). I warn you against this, as most non volatile memory have a limited number of write cycle lifetime (sometimes as low as 100). Such a patch will lead a drive to write and write again this memory, and this will sooner or later cause a NVRAM breakdown. Other drives store the region change by eating up a reserved memory range. In this case, when the range is exhausted, the drive will either corrupt contiguous memory or fail to change its region. In either case, it will take some time before such a firmware fails, so publishing such a patched firmware is like spreading landmines. So, once again: *don't patch a firmware this way*.
- *Gotta patch'ém all*. A whole product line frequently shares firmware development resources (team, program sources, etc), so firmwares from a drive to the next are very similar. If you patched one, the next one should be pretty easy. Most users do not have the necessary skills to patch, so if you succeed in patching your drive's firmware, please consider patching more drive of the same product line. It shouldn't take you much time, and it will help thousands of people all around the world.

Final words

I would like to thank all the people that helped me in many ways to build my tools and firmware patches. They are too numerous to list here. You know who you are. I had a lot of fun doing all this job!

At last but not least, special thanks go to my wife *Cécile*. She did support me, even during the darkest days...

Feedback (both positive and negative) is welcome. You can mail me at: xvi91@hotmail.com