

Battle for Middle Earth

[Home](#) [Forums](#) [Replays](#) [Videos](#) [Strategies](#) [Tournaments](#) [Online Community](#) [Game Info](#)

GameReplays Site News

Happy Holidays!

Stellar Warfare - Brand new...

Steam Summer Sale!

[GameReplays.org](#) » [Battle for Middle Earth Forums](#) » [Welcome to the Battle for Middle-earth Community](#) » [Battle for Middle-earth General Discussion](#)

BFME1 SafeDisc unpacked + patched

Options ▼

[Add Reply](#)[New Topic](#)

withmorten

Mar 17 2022, 06:37 AM # 1

BFME1 SafeDisc unpacked + patched


Hi everyone,
not really an active member here, but wanted to post this here nonetheless, since this seems the proper place.
My only real other post was a few years ago about having hacked the launchers to never cause the auto defeat issue, this goes in a similar direction.


Edit: Of course right as I post this a few minutes I get a null pointer exception in MemoryPool::_Free ... since this has never happened to me before with the original game, I guess I'll need to investigate further 🙄

Edit2: Played again, no crash. So maybe it was just dumb luck, but still seems worth investigating outside of this.

I finally managed to unpack BFME1's game.dat *properly*, and after wondering for years why nobody else had done this before, I realized why.

Before you read this way too long post detailing the process, the hopefully final version is attached in this post. As are hopefully most of the files needed to retrace these steps: the modified unpacker, the original unpacked exe without any patches, and my patching code.

 [game.7z](#)
Size: **4.09mb**
Number of downloads: **30**

 [release.7z](#)
Size: **7.1mb**
Number of downloads: **32**

Here is also a virustotal scan of the final game.dat, which of course returns nothing, since it's not malicious 🤪
(unlike the original safedisc protection): <https://www.virustotal.com/gui/file/53643d7...063c3?nocache=1>

Storytime:

First of course, one needs to unpack it, for that, I started off with this program:
<http://niotso.org/2014/02/18/finished-safe...-a-new-present/>, which I had to modify heavily (my current version is a complete clusterfuck of hacky stuff I did specifically for BFME1).

Long story short: SafeDisc uses 0xCC (int3 - debugbreak) bytes all over the code that need to be replaced with their proper bytes, which works because safedisc debugs the game the entire time.

Problem: BFME1 is an incrementally linked build, that means, it was built with the linker switch "/INCREMENTAL". This is meant for Debug builds, not Release builds, but for some reason this happens from time to time in Release builds it seems. I know it's the default CMake setting for Visual Studio projects, but I doubt BFME1's project was generated with that.

Anyway, this switch is meant to enable the following: During debugging, change functions in the exe at runtime. To do this, each function is not called directly, but via a jump table in the beginning of the exe, so not every offset needs to be updated. Each function has lots of padding behind it, so offsets must be changed less often. At the end of the "actual" code, a huge block of padding separates the actual code and exception handling code generated by the compiler (in BFME1 slightly below 1MB), so the data sections are less likely to need to be shifted (and again, less offsets need to be fixed). The jump table and actual code are also separated by padding, slightly above UINT16_MAX, so if a new function gets added there, not every offset ... you get the point.

Additionally, several *entire* functions main bodies have been replaced with the same byte used for padding.

Now guess what this padding is ... 0xCC bytes. So the normal way of traversing the exe for each 0xCC byte used by that unpacker, even with calculating potential padding areas, takes forever (and actually breaks the safedisc debugger that replaces the bytes,). So I wrote a tool to precompute some padding areas that will be entirely by the function checking for 0xCC bytes. During that I realised that SafeDisc only had infested a certain code range in the beginning, which helped.

After all this, and the other protection, stolen bytes, which were fixed perfectly by the unpacker, I realised the unpacker's import table reconstruction was broken, so I had to rewrite all that and add some hardcoded cases for imports that weren't called, but were just pointers (like _acmdln), but finally, the exe was properly unpacked and had no safedisc protection left.

Or so I hoped ... the game still crashed in an address out of range.

Remember those several entire functions that had been also replaced with 0xCC bytes? They weren't nanomites, they were something else. They are all hashing related functions, for an ExeCrc, an IniCrc and who knows what else. There is a weird mechanism, that checks if a specific global value is nonzero, and if yes, calls that nonzero value, if it is zero, it calls the debugbreak_checksum function, which is full of 0xCC bytes.

Pattern:

```

v15 = (int (__cdecl *)(int, int, int *, int))debugbreak_trigger_12C233C;
VersionNumber = v14;
if ( debugbreak_trigger_12C233C || debugbreak_trigger_12C2340 )
{
    v21 = debugbreak_12C23C4;
    v22 = debugbreak_12C23C8;
    sub_462D90(v24, &v14, &VersionNumber);
    v16 = v15(v21, v22, v24, (int)v24 >> 31);
}
else
{
    v16 = debugbreak_checksum_461F00(v14, v14);
}

```

They are placed in various locations, during game startup, opening options, the spellbook, single player, multiplayer, account creation, deletion etc. And will for example enter an infinite loop when not returning the proper values, or probably break multiplayer sync.

This nonzero value is not within a loaded module when not being debugged by the SafeDisc debugger. At this time my DaemonTools on Win7 broke so I could no longer run the unpacker (it requires safedisc properly working) and test anything in there.

But since we have the worldbuilder, which is not encumbered with any DRM schemes, I managed to find this same function in there, using names from ZeroHour I found out it this specific trigger was in "GlobalData::GlobalData". And tada, the same pattern was there, and the debugbreak_checksum function untouched, and the trigger 0! This pattern is even still in rotwk and its worldbuilder, but the trigger is always 0 there also.

I identified all the checksum functions, their triggers, and the static data array they used for their calculation, found their locations in the worldbuilder, then basically copied the assembly from ida once and wrote a bunch of .asm files that restored all the functions, and then placed them in their proper places in the exe. I also nulled all the triggers, and copied the arrays from the worldbuilder (these arrays are still the same in rotwk, so I also had to check there, since one byte in various static arrays was 0 in the game, and 1 in the worldbuilder). This took slightly more than a week.

A restored array:

```

    db 78h ; x
; unsigned int debugbreak_array_12C2304[14]
debugbreak_array_12C2304 dd 67231341h
    dd 184h
    dd 1
    dd 3Fh
    dd 0
    dd 5CAC5AC5h
    dd 0F135B4E3h
    dd 0B10C27h
    dd 0F5A9E4Fh
    dd 7F586E0Dh
    dd 0DA2B64D5h
    dd 1D1434E5h
    dd 32FB1026h
    dd 0CB9A0Fh
; int (__cdecl *debugbreak_trigger_12C233C)(_DWORD, _DWORD, _DWORD, _DWORD)
debugbreak_trigger_12C233C dd 0
debugbreak_trigger_12C2340 dd 0
    db 0

```

A restored function:

```
int __cdecl debugbreak_checksum_461F00(int a1, int a2)
{
    int v2; // eax
    int v4; // [esp+10h] [ebp-14h]
    int v5; // [esp+1Ch] [ebp-8h]
    unsigned int *array_4616C0; // [esp+20h] [ebp-4h]

    array_4616C0 = debugbreak_get_array_4616C0();
    v5 = debugbreak_hash_data_1_461720(array_4616C0, a1);
    v4 = debugbreak_hash_data_1_461720(array_4616C0, a2);
    v2 = debugbreak_return_data_1_461710(v5, v4);
    debugbreak_return_data_2_461880(5, 1, v5, v4, v2);
    return debugbreak_hash_data_2_461900(array_4616C0, a2);
}
```

hash_data_1 and hash_data_2 were also 0xCC'd, but these two functions never changed between all the slightly different checksum functions, so I only had to restore each of them once. There were however 32 of these each slightly different checksum functions.

Now I was finally done with that, the game starts, and using the *same* exe, I can even play multiplayer. But there is another problem: BFME1 uses an ExeCrc in addition to the IniCrc , unlike BFME2 and ROTWK which just use the IniCrc.

This ExeCrc is actually what's being calculated in the screenshot above, and right above that is a call to GetModuleFileNameA and after that a call to FileSystem::openFile with the filename returned by that ... and then it calculates a checksum over the literal entire file data in UINT16_MAX chunks.

This is btw the reason that replays between the original uncracked safedisc version and the patched safedisc version are incompatible, and multiplayer too.

This is a problem, because now all old replays are incompatible, and despite no actual sync differences, mp is impossible with the previous crack.

So what I did was make the game read the original safedisc crack once instead of the currently running file, save the value, and use that value instead of actually reading the file. This makes it now possible to play this unpacked version with somebody who still uses the original crack. If you think this might be unsafe because of cheating/sync or so, reminder, neither bfme2 nor rotwk use this exeCrc anymore, so this shouldn't really be a problem.

I also patched 3 other locations where the exeCrc is checked, specifically the "load replay" and "save replay" screen to get rid of the grey color if the exeCrc is wrong, and in "RecorderClass::testVersionPlayback", which if this returns false, makes the game pop up that "this replay might not work correctly". Essentially, exeCrc is now ignored for replays.

Now I am 99% certain that all the protections of this game have been defeated, and it should work correctly.

Please test this and let me know if there are any issues!

This post has been edited by **withmorten**: Mar 17 2022, 07:27 AM



Don't like this display mode? [Switch to: Standard](#)

Posts in this topic

- withmorten** BFME1 SafeDisc unpacked + patched Mar 17 2022, 06:37 AM
- Beterwel** Books of gibberish to solve an issue nobody suffer... Mar 19 2022, 08:38 AM
- withmorten** Books of gibberish to solve an issue nobody suffe... Mar 20 2022, 04:59 AM
- ValsaDoom** Hah, yeah, sorry. I had to condense about a mont... Mar 20 2022, 15:45 PM
- withmorten** I agree, kids of tomorrow will be so much smarter... Mar 21 2022, 02:15 AM
- ValsaDoom** This is a much too specific and specialized subjec... Mar 19 2022, 12:54 PM
- Johann Tilly** thanks for your post. I dont play bfme1 anymore, b... Apr 15 2022, 15:15 PM



1 User(s) are reading this topic (1 Guests and 0 Anonymous Users)

