# Weekend Project: A Custom IDA Loader Module for the Hidden Bee Malware Family — Möbius Strip Reverse Engineering

By Rolf Rolles

Published: 2018-09-02 · Archived: 2026-04-05 18:42:09 UTC

Given that custom loaders are the only variety of IDA plugin that I haven't yet written, this seemed like a nice small-scope project for the weekend to round out my knowledge. My very minor contribution with this entry is the IDA custom loader for the Hidden Bee format, which can be found on my GitHub. The IDAPython code requires that Ero Carrera's pefile module be installed, say via pip.

## Hidden Bee

In brief, the Hidden Bee malware family distributes payloads in a customized file format, which is a majorly stripped-down version of the PE file format. You can see all of the details in hasherezade's write-up. I did no original malware analysis for this project; I merely read her blog entry, figured out how to convert the details into a loader plugin, and then debugged it against the sample links she gave. As usual, Chris Eagle's The IDA Pro Book, 2nd Edition was useful. Some details about the loader API have changed with the IDA 7.x API port, but Hex-Rays' porting guide was informative, and the loader examples in the IDA 7.1 SDK have also been ported to the newest API.

## IDA Loader Modules in Brief

An IDA loader module is simply an IDA plugin with a well-defined interface. IDA loader modules will be called when loading any file into IDA. They have two primary responsibilities:

1. Given access to the bytes of a file, determine whether the file is of a format that the loader module can handle. Every IDA loader module must export a function named accept_file for this purpose. This function returns 0 if it can't recognize the file format, or a non-zero value if it can.
2. If the file type can be loaded by the module, and the user chooses to use this module to load the file, perform the actual loading process e.g. creating segments within the IDB, copying bytes out of the file into the segments, processing relocations, parsing imports, adding entrypoints, and so on. Every IDA loader module must export a function named load_file for this purpose.

Both of these functions take as input an "linput_t *" object that behaves like a C FILE * object, which supports seeking to specified positions, reading byte arrays out of the file, and so on. Since Hidden Bee's format includes relocations, I chose to implement a third, optional IDA loader module function: move_segm. This function will be called by the IDA kernel when the user requests that the database be relocated to another address.

## Writing a Loader Module for Hidden Bee

After reading the aforementioned write-up, I figured that the only difficulties in loading Hidden Bee images in IDA would be A) that the Hidden Bee customized header specifies API imports via hash rather than by name, and B) that it includes relocation information. Relocations and import lookup via hash are simple enough conceptually, but the precise details about how best to integrate them with IDA are not obvious. Sadly, I did not feel confident in these tasks even after reading the loader module examples in the SDK. Four out of the five hours I spent on this project were reverse engineering %IDADIR%\loaders\pe.dll -- the loader module for the PE file format -- focusing in particular on its handling of relocations and imports. As expected, the results are idiosyncratic and I don't expect them to generalize well.

## Imports

For dealing with the imports by hash, hasherezade's toolchain ultimately generates a textual file with the addresses of the import hash names and their corresponding plaintext API string. Then, she uses one of her other plugins to create repeating comments at the addresses of the import hash DWORDs. Instead, I wanted IDA to show me the import information the same way it would in a normal binary -- i.e., I wanted IDA to set the proper type signature on each import. I figured this might be difficult, but after a few hours reverse engineering the virtual functions for the pe_import_visitor_t class (partially documented in %IDASDK%\ldr\pe\common.hpp), it turns out that all you have to do to trigger this functionality is simply to set the name of the DWORD to something from a loaded type library.

Here's a screenshot showing IDA successfully applying the type information to the APIs:

## Relocations

For the IMAGE_REL_BASED_HIGHLOW relocations common in PE files, each can ultimately be processed via straightforward translation of the relocation information into IDA's fixup_data_t data structures, and then passing them to the set_fixup API. The SDK examples did not give a straightforward idea of what I needed to do to handle PE IMAGE_REL_BASED_HIGHLOW relocations properly, so I reverse engineered pe.dll to figure out exactly what needed to happen with the relocations. (Fortunately, reverse engineering IDA is trivial due to the availability of its SDK.) If you wish, you can see the results in the do_reloc function. Don't ask me to explain why it works; however, it does work.

Here's a before and after comparison of rebasing the database from base address 0x0 to base address 0x12340000. Note particularly that the red underlined bytes change. Before:

---