

Unpacking PECompact v2.79 beta d

Gabri3l of ARTeam

Version 1.0 - August 2006

1. Introduction

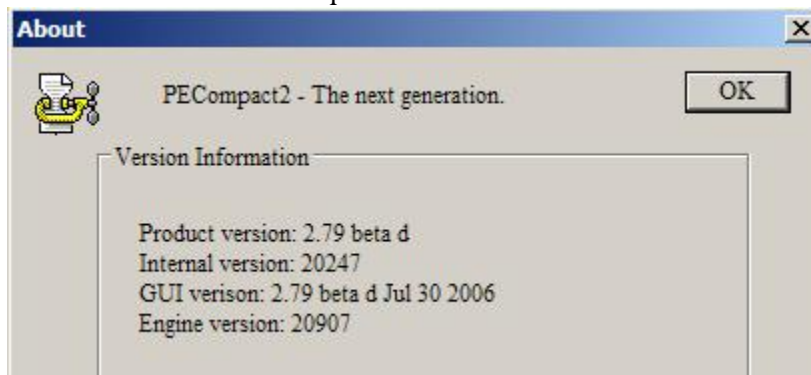
We are going to be unpacking the newest version of PECompact available at www.bitsum.com/. Hopefully after reading this tutorial you will have a better idea of how PECompact works and the effective ways to unpack it, we will then cover unpacking and rebuilding a pecompact target with advanced options to prevent debugging and rebuilding.

Tools needed:

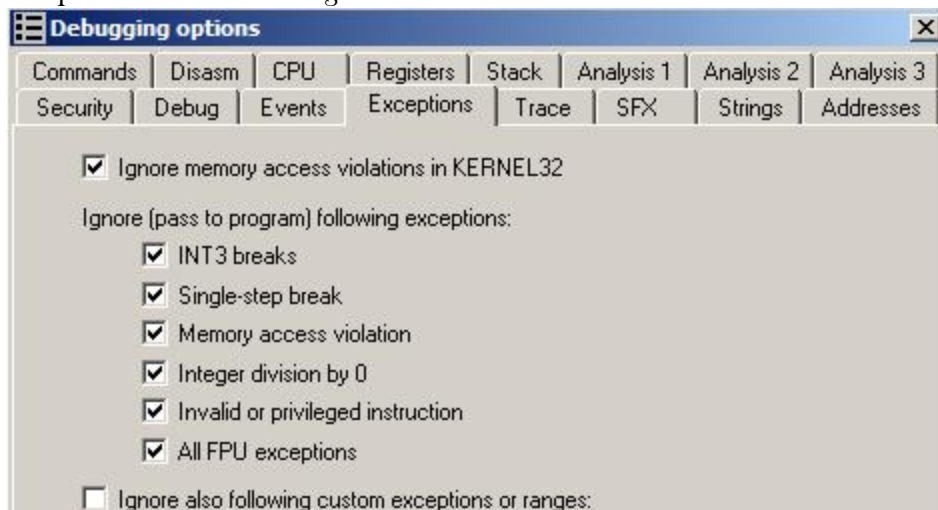
1. Ollydbg
 - Ollydump plugin
2. ImpREC

2. Unpacking

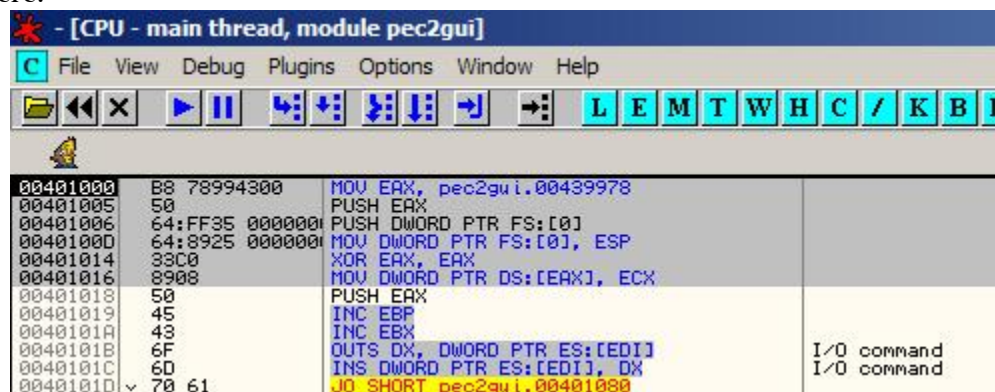
The target today is the newest beta of PECompact version 2.79 beta d



Lets start by opening up Olly. The trial beta of PECompact does not use any anti-debugging tricks, but we will set our exceptions to the following:



Once our exceptions are set we can open PECompact2 GUI in Ollydbg. You will find yourself at the entrypoint here:



```
- [CPU - main thread, module pec2gui]
File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B] [R]
00401000  B8 78994300  MOV EAX, pec2gui.00439978
00401005  50          PUSH EAX
00401006  64:FF35 00000000 PUSH DWORD PTR FS:[0]
0040100D  64:8925 00000000 MOV DWORD PTR FS:[0], ESP
00401014  33C0       XOR EAX, EAX
00401016  8908       MOV DWORD PTR DS:[EAX], ECX
00401018  50          PUSH EAX
00401019  45          INC EBX
0040101A  43          INC EBX
0040101B  6F          OUTS DX, DWORD PTR ES:[EDI]
0040101C  6D          INS DWORD PTR ES:[EDI], DX
0040101D  70 61      JG SHORT pec2gui.00401080
I/O command
I/O command
```

Now we are going to talk a little about how PECompact2 unpacks itself. First thing you will notice is that your entrypoint is located in the code (.text) section of the executable. You can verify this by checking the memory map in Olly:

00150000	00001000				Priv	RWE	RWE
00160000	00001000				Priv	RWE	RWE
00170000	00001000				Priv	RWE	RWE
00180000	00003000				Priv	RW	RW
00280000	00006000				Priv	RW	RW
00290000	00003000				Map	RW	RW
002A0000	00016000				Map	R	R
002C0000	0003D000				Map	R	R
00300000	00041000				Map	R	R
00350000	00006000				Map	R	R
00360000	00001000				Priv	RWE	RWE
00370000	00001000				Priv	RWE	RWE
00380000	00001000				Priv	RWE	RWE
00400000	00001000	pec2gui		PE header	Imag	R	RWE
00401000	00037000	pec2gui	.text	code	Imag	R	RWE
00438000	00002000	pec2gui	.rsrc	imports, res	Imag	R	RWE
7C800000	00001000	kernel32		PE header	Imag	R	RWE
7C801000	00082000	kernel32	.text	code, import	Imag	R	RWE
7C883000	00005000	kernel32	.data	data	Imag	R	RWE
7C888000	00066000	kernel32	.rsrc	resources	Imag	R	RWE
7C8EE000	00006000	kernel32	.reloc	relocations	Imag	R	RWE

PECompact has written its loader into the main section of the executable. What does this mean when we are unpacking? It means that at some point PECompact execution will leave the code section in order to properly unpack the original file. To do this PECompact will create a new section of memory to store some unpacking and decryption code. We can verify that it is creating new sections by examining the memory map before and after execution. The image above shows memory before execution. The next image shows memory after execution. We can see the addition of multiple sections:

00170000	00001000				Priv	RWE	RWE
00180000	00015000				Priv	RW	RW
00280000	00006000				Priv	RW	RW
00290000	00003000				Map	RW	RW
002A0000	00016000				Map	R	R
002C0000	0003D000				Map	R	R
00300000	00041000				Map	R	R
00350000	00006000				Map	R	R
00360000	00001000				Priv	RWE	RWE
00370000	00001000				Priv	RWE	RWE
00380000	00001000				Priv	RWE	RWE
00390000	00002000				Priv	RWE	RWE
003A0000	00001000				Priv	RW	RW
003B0000	00001000				Priv	RW	RW
003C0000	00004000				Priv	RW	RW
003D0000	00001000	pec2rsrc		PE header	Imag	R	RWE
003D1000	00001000	pec2rsrc	.rdata	data, export	Imag	R	RWE
003D2000	00008000	pec2rsrc	.rsrc	resources	Imag	R	RWE
003DA000	00001000	pec2rsrc	.reloc	relocations	Imag	R	RWE
003E0000	00001000				Map	R	R
003F0000	00001000				Priv	RW	RW
00400000	00001000	pec2gui		PE header	Imag	R	RWE
00401000	00037000	pec2gui	.text	code	Imag	R	RWE
00438000	00002000	pec2gui	.rsrc	imports, res	Imag	R	RWE
00440000	00004000				Map	R E	R E

I highlighted some of the memory that was added. Now keep in mind that not all the new sections were created by PECompact, they may have been created by other dlls and programs operating in the PECompact memory space. Now we know that it is using new sections of memory, but the question is how does it do that in Windows? Well the answer is simple, and also found in the MSDN:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/memory/base/virtual_memory_functions.asp

Microsoft explains that virtual memory functions are used to allocate and free memory for programs. These allocated memory sections are reserved and can be written to and read from but not be overwritten or reallocated until they are freed from memory.

So the unpacker is allocating new memory in order to unpack the original executable code. But what functions does it use specifically? Again the answer is found in MSDN:

VirtualAlloc
Reserves or commits a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero, unless MEM_RESET is specified.
VirtualFree
Releases, decommits, or releases and decommits a region of pages within the virtual address space of the calling process.

Programs often use these 2 API functions in conjunction, after memory has been allocated and used it can be freed in order to allow other programs to use the memory and reduce the footprint for the program.



So you are asking yourself: “why do I care?” The reason is that knowing how the memory is allocated and freed allows us to predict how PECompact will operate when it is using memory. It will most likely allocate memory using VirtualAlloc and then after it is done using the memory it will free it using VirtualFree. So it will allocate memory to store unpacking code and it will probably not free the memory until it is no longer needed at all. Hopefully when the memory is freed it means that the PECompact does not need to do anymore unpacking and we will be close to where it re-enters the code section.

Well let's test this theory out. Go back to Olly, and make sure PECompact is loaded and we are at the entry point. Next we need to set a Breakpoint on VirtualFree. Hopefully when we reach VirtualFree it means that PECompact is done with allocated memory and very close to reaching the OEP. You can easily set a breakpoint by using the CommandLine plugin that comes with Olly. Enter BP VirtualFree in the CommandLine and press ENTER:



After you have set the breakpoint go ahead and press RUN to start the program. You will see it execute for a few and then you will hit a breakpoint at VirtualFree. To see the arguments of the function VirtualFree look at the stack in Olly:



0012FF14	0033087A	CALL to VirtualFree from 00330877
0012FF18	003A0000	Address = 003A0000
0012FF1C	00000000	Size = 0
0012FF20	00008000	FreeType = MEM_RELEASE
0012FF24	003741E0	

Interesting, here we see that this function was called from location 330877 and it is going to free memory at 3A00000. Let's see where we land if we execute the VirtualFree Function. Press Execute-Till-Return:  This will put you at the end of the VirtualFree function. Now press Step-Into to follow the return back to it's calling location:  We will end up here:

0033087A	SF	POP EDI
0033087B	5E	POP ESI
0033087C	8BC3	MOV EAX,EBX
0033087E	5B	POP EBX
0033087F	C9	LEAVE
00330880	C2_0C00	RETN_0C

Well we are back in a section of memory that is not part of our original executable. I am going to save you a few steps here, and a few keystrokes for myself, and tell you that we are not at the right location yet. If you were to continue stepping through the code you would find that the program is still not fully unpacked. So lets continue to run the executable and see if we break at VirtualFree again. Press RUN to continue execution. You will break pretty quickly at VirtualFree again. This time the arguments on the stack are:

0012FF84	00330B9B	CALL to VirtualFree from 00330B95
0012FF88	00360000	Address = 00360000
0012FF8C	00000000	Size = 0
0012FF90	00008000	FreeType = MEM_RELEASE
0012FF94	F04386FD	

We see that the calling location for the function is 330B95, so we are still in the 330B95 range. Just as before Press Execute-Till-Return:  Press Step-Into to follow the return back to the original calling location:  We will end up here:

00330B9B	8B46 0C	MOV EAX,DWORD PTR DS:[ESI+C]
00330B9E	03C7	ADD EAX,EDI
00330BA0	5D	POP EBP
00330BA1	5E	POP ESI
00330BA2	5F	POP EDI
00330BA3	5B	POP EBX
00330BA4	C3	RETN

Let's see where this leads... Press Execute-Till-Return and step into the return just as we have been doing before. Doing so will place us here:

00439A18	8985 3F130010	MOV DWORD PTR SS:[EBP+1000133F],EAX	pec2gui.0041B6C3
00439A1E	8BF0	MOV ESI,EAX	
00439A20	8B4B 14	MOV ECX,DWORD PTR DS:[EBX+14]	
00439A23	5A	POP EDX	
00439A24	EB 0C	JMP SHORT pec2gui.00439A32	
00439A26	03CA	ADD ECX,EDX	
00439A28	68 00800000	PUSH 8000	
00439A2D	6A 00	PUSH 0	
00439A2F	57	PUSH EDI	
00439A30	FF11	CALL DWORD PTR DS:[ECX]	
00439A32	8BC6	MOV EAX,ESI	
00439A34	5A	POP EDX	
00439A35	5E	POP ESI	
00439A36	5F	POP EDI	
00439A37	59	POP ECX	
00439A38	5B	POP EBX	
00439A39	5D	POP EBP	
00439A3A	FFE0	JMP EAX	
00439A3C	0000	ADD BYTE PTR DS:[EAX],0	

We are very close now! That JMP EAX is our jump to the Original EntryPoint. You can continue to

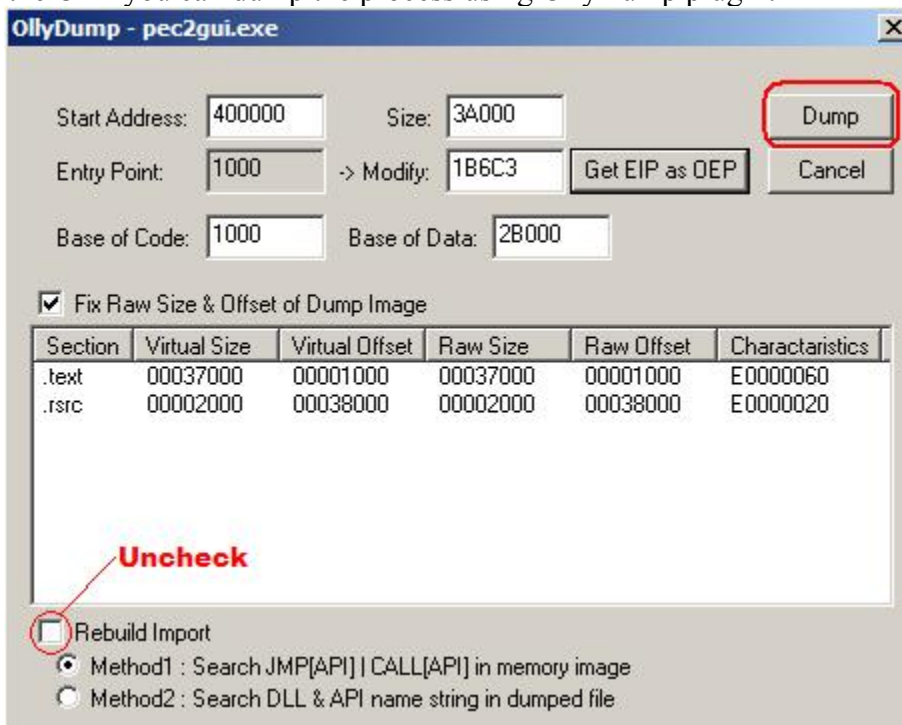
step through the code until you get to the JMP EAX and follow that to 41B6C3, or whatever the OEP is for your program. The entrypoint for PECompact2 GUI is:

```

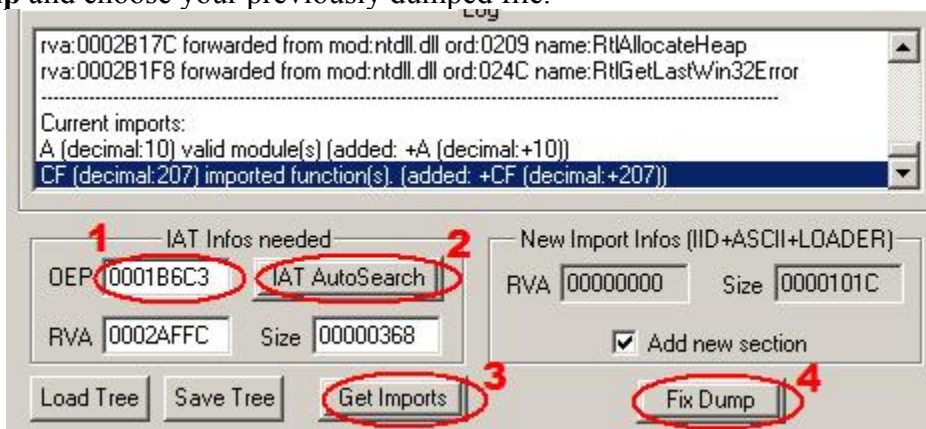
0041B6C3 E8 9B640000 CALL pec2gui.00421B63
0041B6C8 ^E9 40FEFFFF JMP pec2gui.0041B500
0041B6CD CC INT3
0041B6CE CC INT3
0041B6CF CC INT3
0041B6D0 8B5424 0C MOV EDX,DWORD PTR SS:[ESP+C]

```

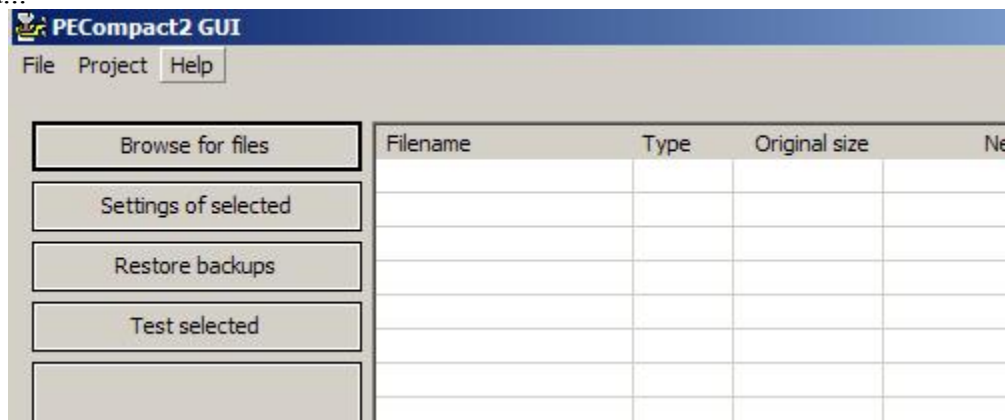
Once you get to the OEP you can dump the process using OllyDump plugin:



Make sure you uncheck the Rebuild Import option, and then press DUMP. Select a name for the dumped executable and press Okay. We now only need to rebuild the imports. Rebuilding the imports for this target is fairly simple. Use ImportREC and attach to the pec2gui.exe program. Once attached enter the Original EntryPoint of the program into the OEP box. In this case my original entrypoint was at 41B6C3. However my image-base was 400000. $41B6C3 - 400000 = 1B6C3$. After you have entered the OEP press **IAT AutoSearch**. You will be presented with a box saying it located what may be the original Import table. Press **Okay**. Now press **Get Imports**. You will be presented with a list of imports, make sure that they are all valid by looking for the “Valid:Yes” after each thunk. Finally press **Fix Dump** and choose your previously dumped file.



ImportREC will name the new fixed file *filename_.exe*, in my case my fixed dump was *PeDump_.exe*. Now, finally with a fixed IAT we can test our unpacked file. Find your newly saved file, Double-Click to run it and...



It works!! We unpacked PECompact2.79 successfully! Hopefully this has given you a better idea as to how the packer works and the methods you can use to unpack it. If you want to continue on I am going to cover more advanced features that you may find in a PECompact compressed file.

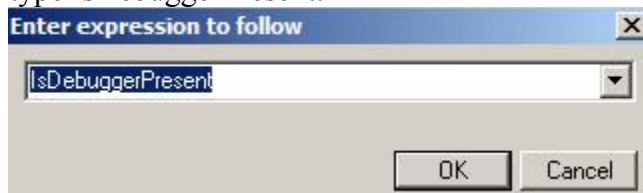
3. Advanced Features

Not all PECompact programs are alike. Some can use advanced features such as anti-debugging and import protection. Here I will cover some of the advanced options you may come across. First thing that we will look at the anti-debugging feature. If you constantly get a debugger present warning when trying to unpack a pecomact compressed file, it has most likely been packed using the anti-debug feature.



The anti-debugging feature in PECompact is fairly simple. It calls the API function `IsDebuggerPresent`, which then returns 1=yes debugger, or 0=no debugger. We can simple defeat it by going to the `IsDebuggerPresent` API function and modifying it to always return 0.

First load your program into Ollydbg. Before executing anything, Right-Click and choose Go-To. In the new box that opens up type `IsDebuggerPresent`:

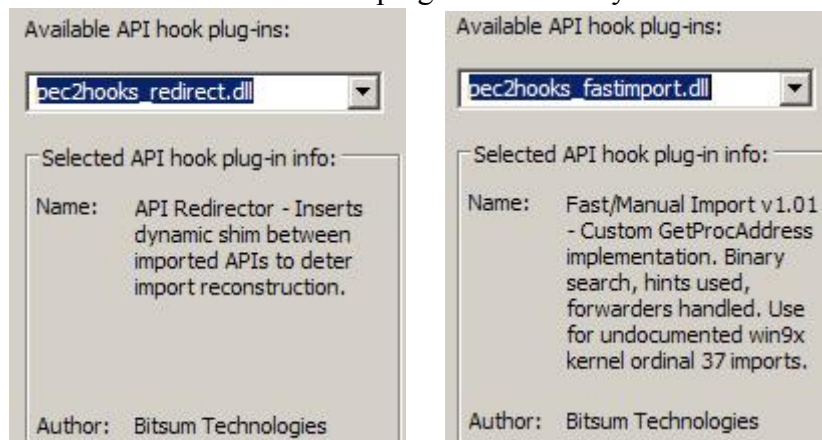


Press Okay and you will be taken to the start of the `IsDebuggerPresent` API function. You will see that it is a very short function. By zeroing out EAX before it returns we will make it always return 0. Select the line: **MOVZX EAX,BYTE PTR DS:[EAX+2]**

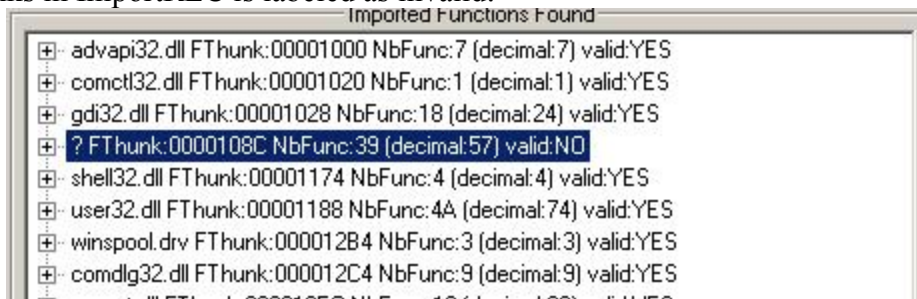
With that line selected, press SPACEBAR to Assemble the line. A new box will open up and in it type **XOR EAX,EAX**. Make sure you **check Fill with NOPS**. Then press **Assemble**. Your modified code will look like this:

7C812E03	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C812E09	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]
7C812E0C	33C0	XOR EAX,EAX
7C812E0E	90	NOP
7C812E0F	90	NOP
7C812E10	C3	RET
7C812E11	90	NOP
7C812E12	90	NOP

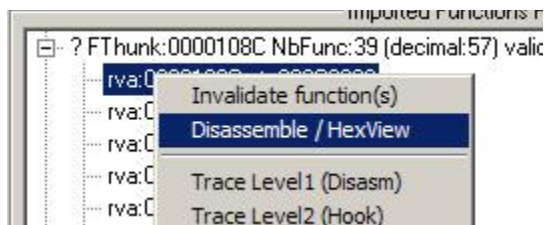
Once you have done that you can run the executable and unpack the program following the same steps as we covered previously. However you may find yourself against some of the packers other features when trying to fix the IAT. There are different plugins that modify the IAT:



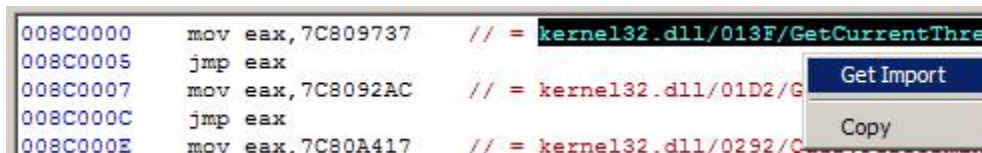
You will know when you encounter a program that has used these plugins because you will notice that one of the Thunks in ImportREC is labeled as invalid:



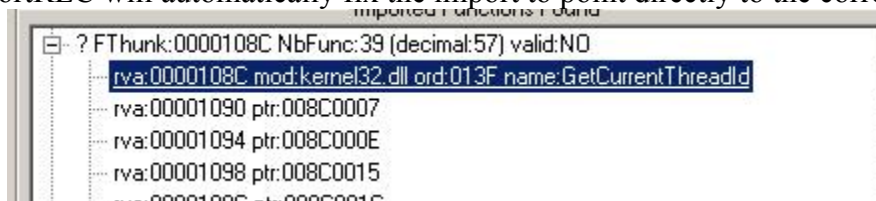
To examine these invalid entries press the [+] in front of the invalid thunk. That will list all the imports that are invalid. The reason they are invalid is because they have been redirected. They have been redirected to jump to an allocated memory location before they actually call the API function. We can see where they have been redirected too by right-clicking one on one of the invalid imports and choose Disassemble/Hex View:



Once you have done that a new window will open up showing the location of the redirected API functions. Thankfully the API redirection is not very complex at all. We can see immediately what API function has been redirected. To fix the redirection, right-click on the function name and choose Get Import:

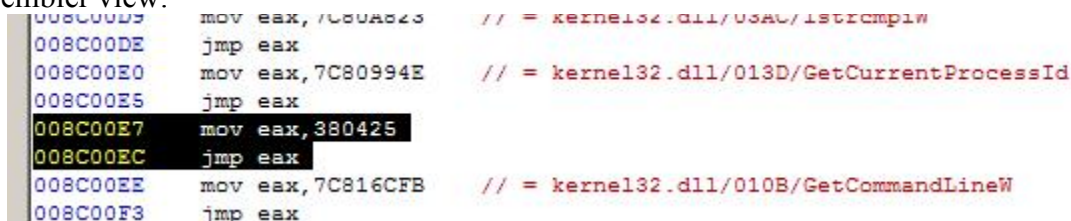


By doing so ImportREC will automatically fix the import to point directly to the correct API function:

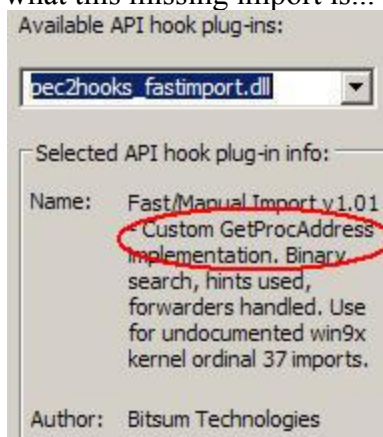


Continue to the next import and follow the same steps. Right-Click invalid import> Disassemble/Hex View->Right-Click Import Name. It may seem tedious work but it won't take you very long.

However, sometimes you may come across a single import that doesn't have a name when you view it in Disassembler view:



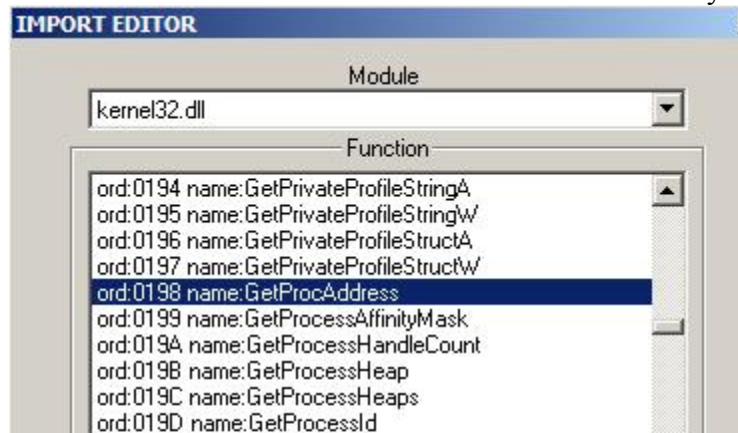
What could this import be? Well we don't really have to do that much detective work to find out because PECompact tells us exactly what this missing import is...



The missing import is GetProcAddress which was "replaced" by a custom version from the PECompact plugin! Thankfully, it is easily fixed by just making the import point to the regular windows GetProcAddress. To do so, close disassembler view and go back to the invalid import list. Highlight the import which has no name and double-click it.

You should be presented with a new window that has a dll name and a list of API functions contained

in that DLL. Choose kernel32.dll as the dll and then scroll down the list until you find GetProcAddress:



Press Okay to fix the import. It will now point to the correct GetProcAddress and not to PECompacts custom version. You can continue to fix the rest of the imports using the Disassembler view in ImportREC. Once you fix all the imports you should see that ImportREC says that the Thunk is now Valid. You can fix your dumped file just as we did previously.

Thats it! Hopefully you enjoyed this tut and have a better understanding of how to unpack PECompact. And hopefully you learned something new that can be applied abstractly outside the realm of just unpacking this single program.

4. Greetings

Thanks to ARTeam, its forum members, the people at #ARTeam on efnet, and all my friends in the reversing scene. ☺

Thanks to all the people who take time to write tutorials.

Thanks to all the people who continue to develop better tools.

Thanks to all the people at Exetools and Woodmann for providing great places of learning.

The contribution to the Reverse Engineering community by so many different teams and people has been so overwhelming I cannot list them all anymore. You all know who you are and I salute you!

And finally, thanks to you for taking the time and interest to read this tutorial.

5. Contact

If you have any questions, comments, or complaints feel more than free to email me at: Gabri3l2003[at]yahoo.com or stop by the ARTeam forum at: <http://forum.accessroot.com>

6. 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>