

Lab 8: Malware Packers

ITSC 303: Malware Analysis

Table of Contents

[Lab Outcomes 2](#_Toc491932861)

[Background Reading 2](#_Toc491932862)

[Introduction 2](#_Toc491932863)

[1.0 Using UPX 3](#_Toc491932864)

[1.1 Packing a Benign Binary 3](#_Toc491932865)

[1.2 Identifying UPX using PEiD 3](#_Toc491932866)

[1.3 Identifying UPX using Section Names 5](#_Toc491932867)

[1.4 Unpacking UPX 5](#_Toc491932868)

[2.0 Unpacking UPack 10](#_Toc491932869)

[2.1 Identifying UPack using PEiD 10](#_Toc491932870)

[2.2 Identifying UPack using Hiew 10](#_Toc491932871)

[2.3 Unpacking UPack 11](#_Toc491932872)

[2.4 Analysis 16](#_Toc491932873)

[3.0 Sign-Off: Lab 8 – Malware Packers 18](#_Toc491932874)

[References 19](#_Toc491932875)

Malware Analysis

Lab 8: Malware Packers

Lab Outcomes

This lab will focus on the following outcomes:

* Learn unpacking tools and techniques.
* Import table reconstruction.

Background Reading

* [The Art of Unpacking](https://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf) (<https://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf>)

Introduction

The term “packer” is often directly related to compression, but it has also become a generic term in industry to refer to most binary code obfuscation mechanisms. The overall goal of legitimate packers is to reduce the size of a distributable executable, which then unpacks in memory to produce the original code to be executed, restoring the original memory footprint after decompression. The primary goal of malicious packers is to conceal the original binary’s intentions, to evade anti-malware engines and to make analysis difficult for reverse engineers. For these reasons, as much as possible of the original binary has to be recovered for analysis.

A stub is executed at a packed binary’s entry point. This stub is often position-independent, which is how shellcode operates. Although the overall purpose of the stub differs between samples, it typically allocates memory to unpack code, unpack that code into the allocated memory and execute it. The memory allocation area can even be within a remote process, which is often referred to as process injection.

This lab walks you through this process with two samples that vary in difficulty. The first is a benign executable packed (by you) using the well‑known packer, UPX (Ultimate Packer for eXecutables); the second is a malware variant packed using UPack.

1. Using UPX

A debugger is most commonly used to help identify and eliminate software bugs or errors in software. Since malware is often obfuscated, or attempts to hide its true functionality, a debugger can be an invaluable tool for a malware analyst. A malware analyst’s most common uses for a debugger are de-obfuscation of malware samples and static analysis.

For this lab you will use OllyDbg, a versatile user mode debugger often used by malware analysts.

## Packing a Benign Binary

1. Open a Windows command prompt and navigate to the lab materials directory.
2. Execute the following command:

> xcopy.exe C:\Windows\System32\cmd.exe %temp%

Before you can pack this binary and analyze it, strip the PE flag that causes the binary’s address space layout to be randomized (ASLR). This is needed for addresses to align while dumping the binary and fixing the IAT.

1. To do this, execute the following command:

> setdllcharacteristics -d cmd.exe

1. Then execute the following command to compress cmd.exe using UPX:

> upx -9 %temp%\cmd.exe

## Identifying UPX using PEiD

1. Open PEiD and click the **…** button.

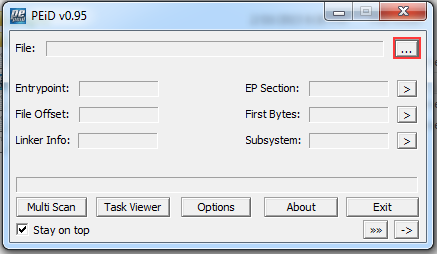


Figure 1: PEiD Open File Window

Source: PEiD software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Navigate to *%temp%*, select **calc.exe** and then click **Open**.

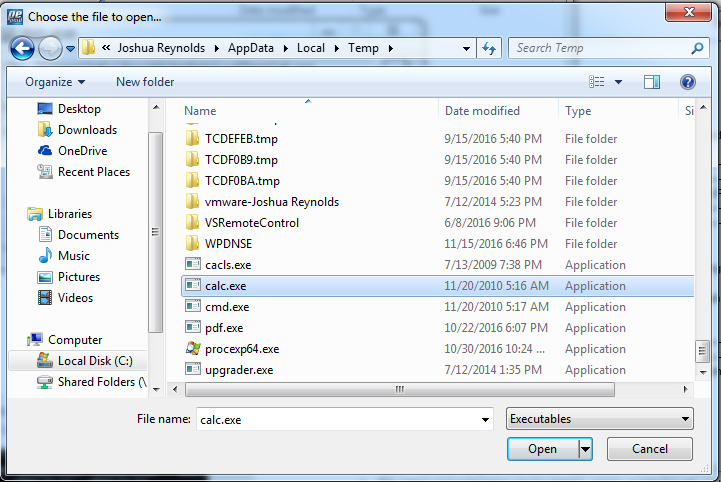


Figure 2: Open calc.exe

Source: PEiD software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

PEiD automatically runs its signature set against calc.exe, which it identifies as being packed using UPX.

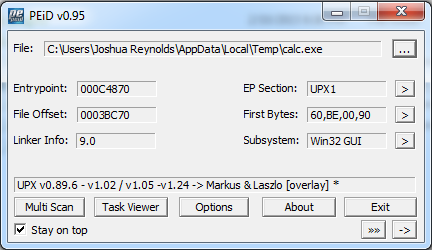


Figure 3: UPX Identified

Source: PEiD software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

## Identifying UPX using Section Names

1. Open Hiew, navigate to the packed **cmd.exe** and open the **Object Table** view.

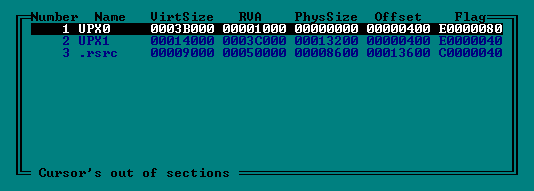


Figure 4: UPX Sections in Hiew

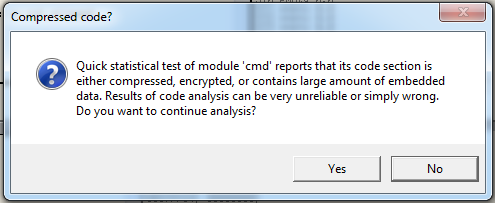
Source: Hiew software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

Notice that, once packed, the binary’s section names change to UPX[0-9].

## Unpacking UPX

1. Open the UPX packed **cmd.exe** in OllyDbg.

The following message appears:

  
Figure 5: UPX Statistical Test Warning

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

This is another indication that the code being analyzed is likely packed.

1. Press F9 to go to the binary’s entry point.

The instruction at this address is PUSHAD, which is used to push the current registry values onto the stack.

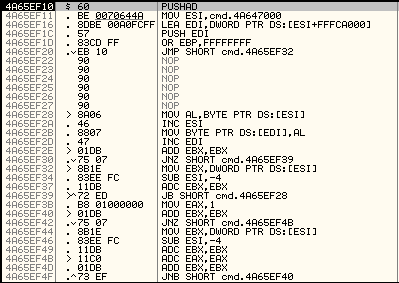


Figure 6: UPX PUSHAD Instruction

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

Scrolling through the linearly swept disassembly, you can see a POPAD instruction followed by an unconditional JMP.

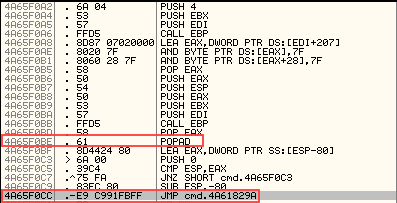


Figure 7: UPX POPAD Instruction Followed by JMP

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Press F2 to set a breakpoint on this address, and then press F9 to run the program until this breakpoint is hit.

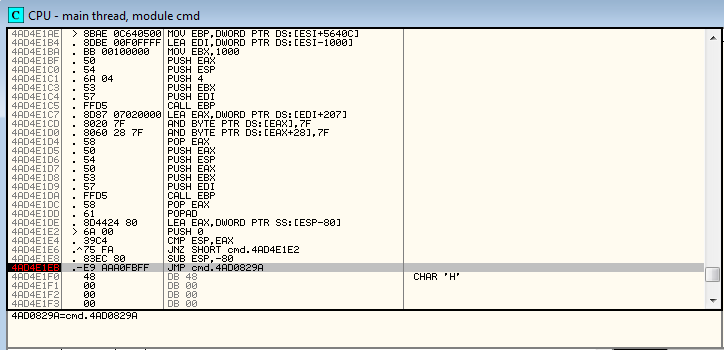


Figure 8: UPX Breakpoint Hit at JMP to OEP

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

The destination address of this JMP instruction is the original entry point (OEP) of the binary. Press F7 to be taken to the OEP.

At this point, you can dump the binary in its current state using LordPE.

1. Open LordPE and select the running **cmd.exe** from the list.

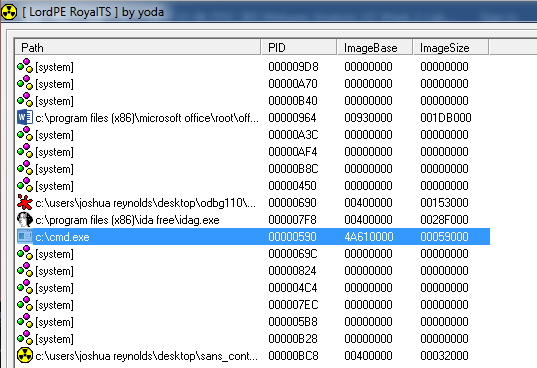


Figure 9: LordePE Highlighted Process

Source: LordePE software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Right-click the highlighted process and select **dump full**.
2. Save the dumped binary to a destination.
3. Open Import REConstructor (ImpREC) and select **cmd.exe** from the *Attach to an Active Process* drop-down menu.

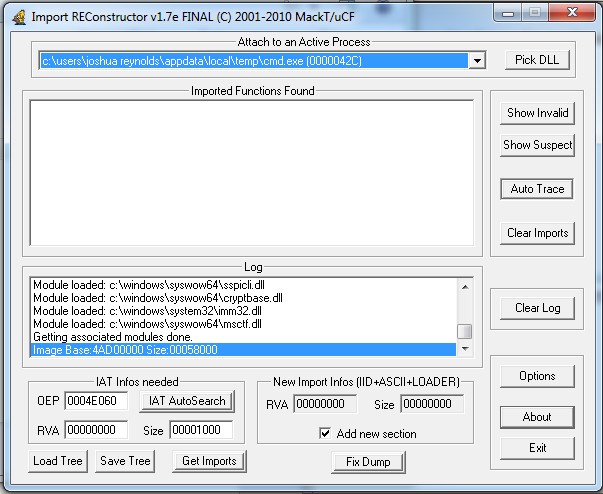


Figure 10: ImpREC Select cmd.exe from Dropdown

Source: ImpREC software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Enter the OEP relative to the image base address. For example, if the image above the image base is 0x4AD00000, and the OEP address is 0x4AD0829A, the resulting relative OEP is 0x829A.
2. Click **IAT AutoSearch** to perform a search from the OEP for addresses that could correspond to the import address table.

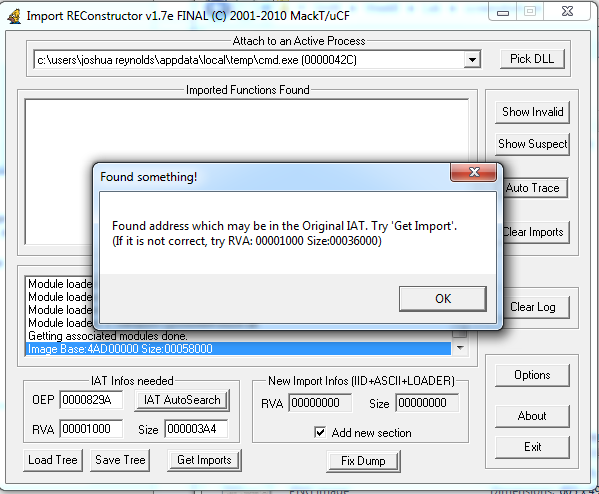


Figure 11: IAT Found using IMPRec Autosearch

Source: ImpREC software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Click Get Imports.

The *Imported Functions Found* table populates.

1. Click Fix Dump and navigate to the location of your LordPE dump.

ImpRec indicates that the fixed dump has been saved successfully.

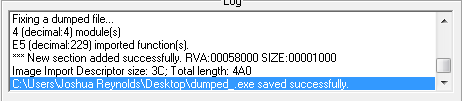


Figure 12: IMPRec Successful Dump Save

Source: ImpREC software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

You now have a fully functional unpacked binary when you click on it or when you want to analyze it further.

As an analysis comparison, run the following two commands and compare the number of strings from PowerShell in their output.

> strings.exe -n 7 %temp%\cmd.exe | measure-object -Line

> strings.exe -n 7 [path to fixed dump file] | measure-object -Line

**Instructor sign-off:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Unpacking UPack

NOTE FOR INSTRUCTOR

Section 2 cannot be completed as UPack does not exist, and as such this section has been skipped.

## Identifying UPack using PEiD

1. Open the supplied UPack binary using PEiD.

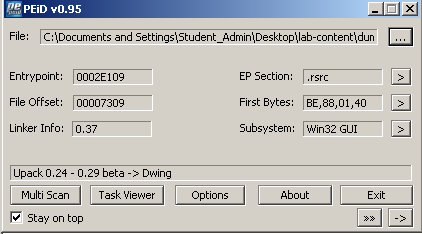


Figure 13: PEiD Identified

Source: PEiD software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

As demonstrated with UPX, PEiD positively identifies UPack as the packer in question using its signature set.

## Identifying UPack using Hiew

1. Open the supplied UPack binary using Hiew software and press Enter.
2. The UPack signature appears close to the MZ header in the rendered ASCII, along with a number of Windows library names.

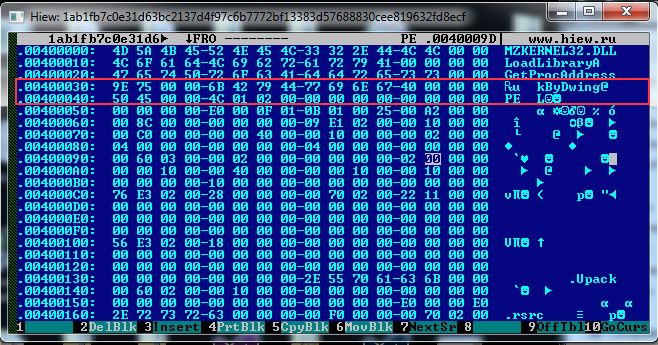


Figure 14: UPack Signature

Source: Hiew software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Open the **Object Table** to view section names.



Figure 15: UPack Section Names

Source: Hiew software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

UPack can also be identified by the use of its name within the binary sections.

## Unpacking UPack

UPack’s OEP can be identified using section hopping. In order to identify from which section it will be executing its original code, set a breakpoint on a function that you know will be executed once unpacked. In this case, that function is CreateFileW.

1. Navigate to that function:

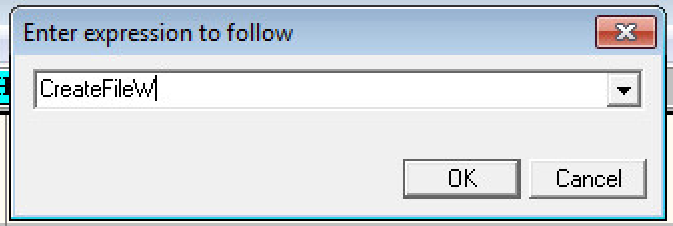


Figure 16: CreateFileW Enter Expression in OllyDbg

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Right-click the first instruction of the function and insert a hardware breakpoint at this address.

**Note:** Packers can sometimes emulate the first few instructions of library import calls, so it is wise to set a breakpoint on the return as well.

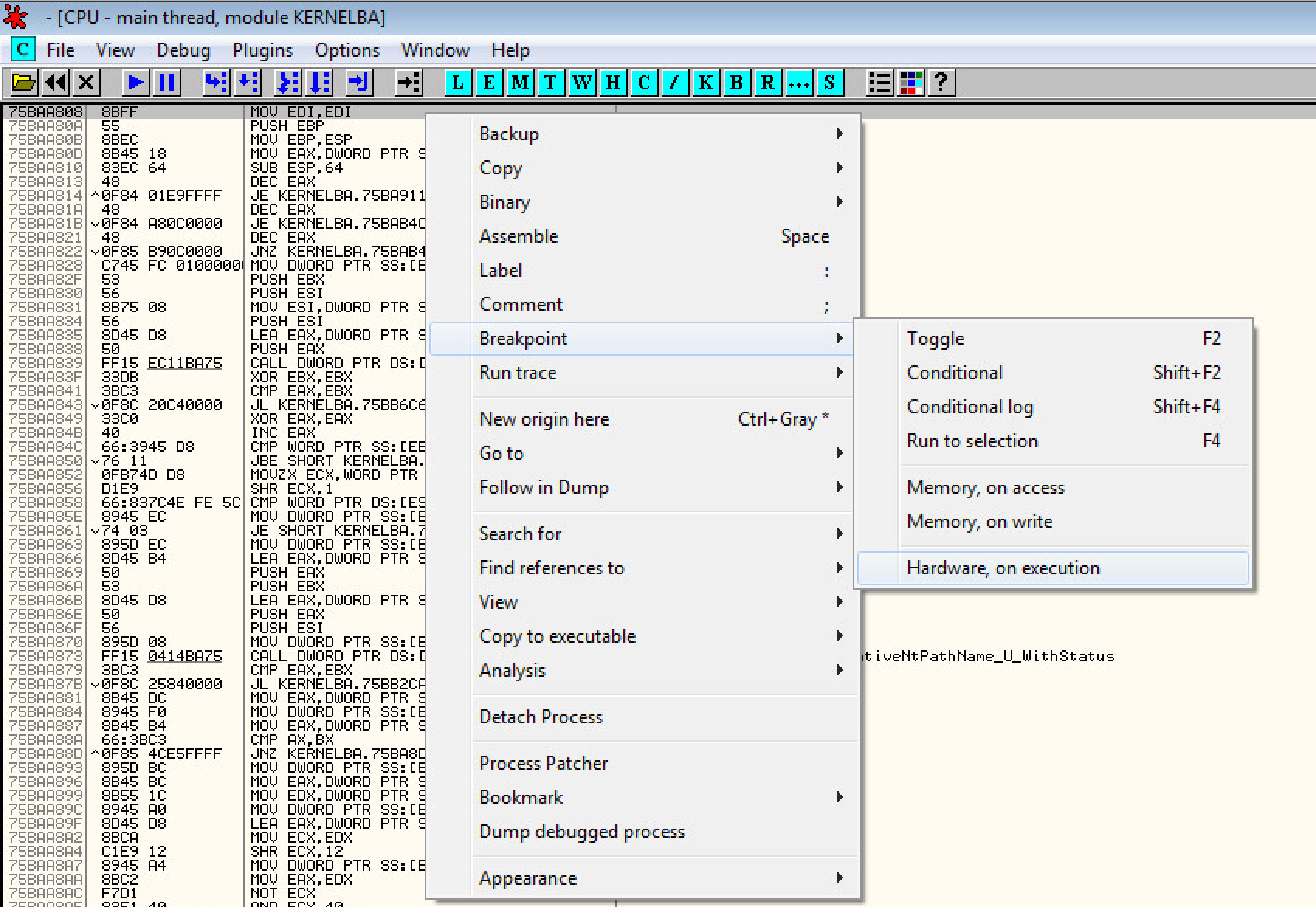


Figure 17: CreateFileW Hardware Breakpoint on Execution

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Run the executable until this breakpoint is hit.

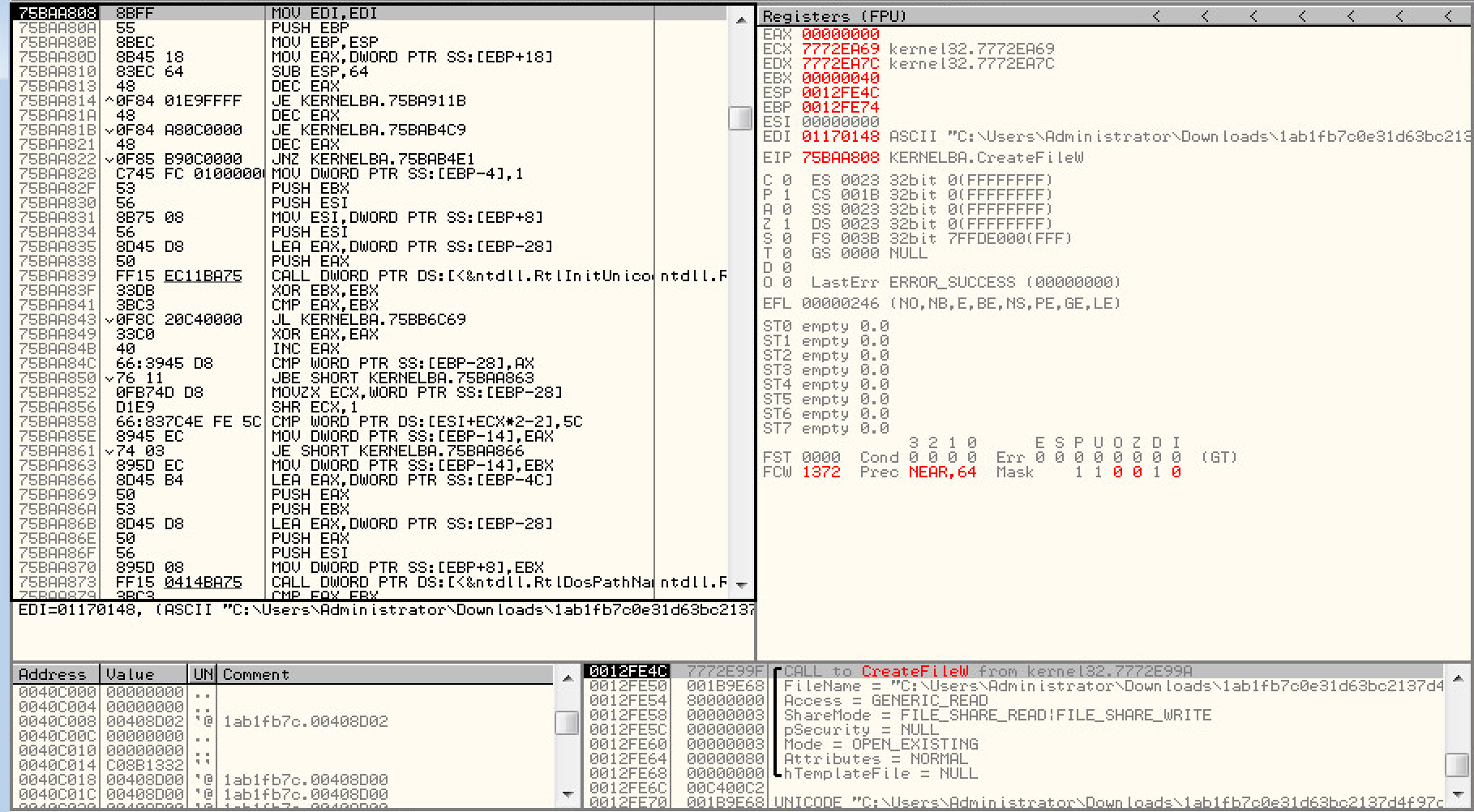


Figure 18: CreateFileW Hardware Breakpoint Hit

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. Press CTRL+F9 to “Execute till return” until you have left the library code.

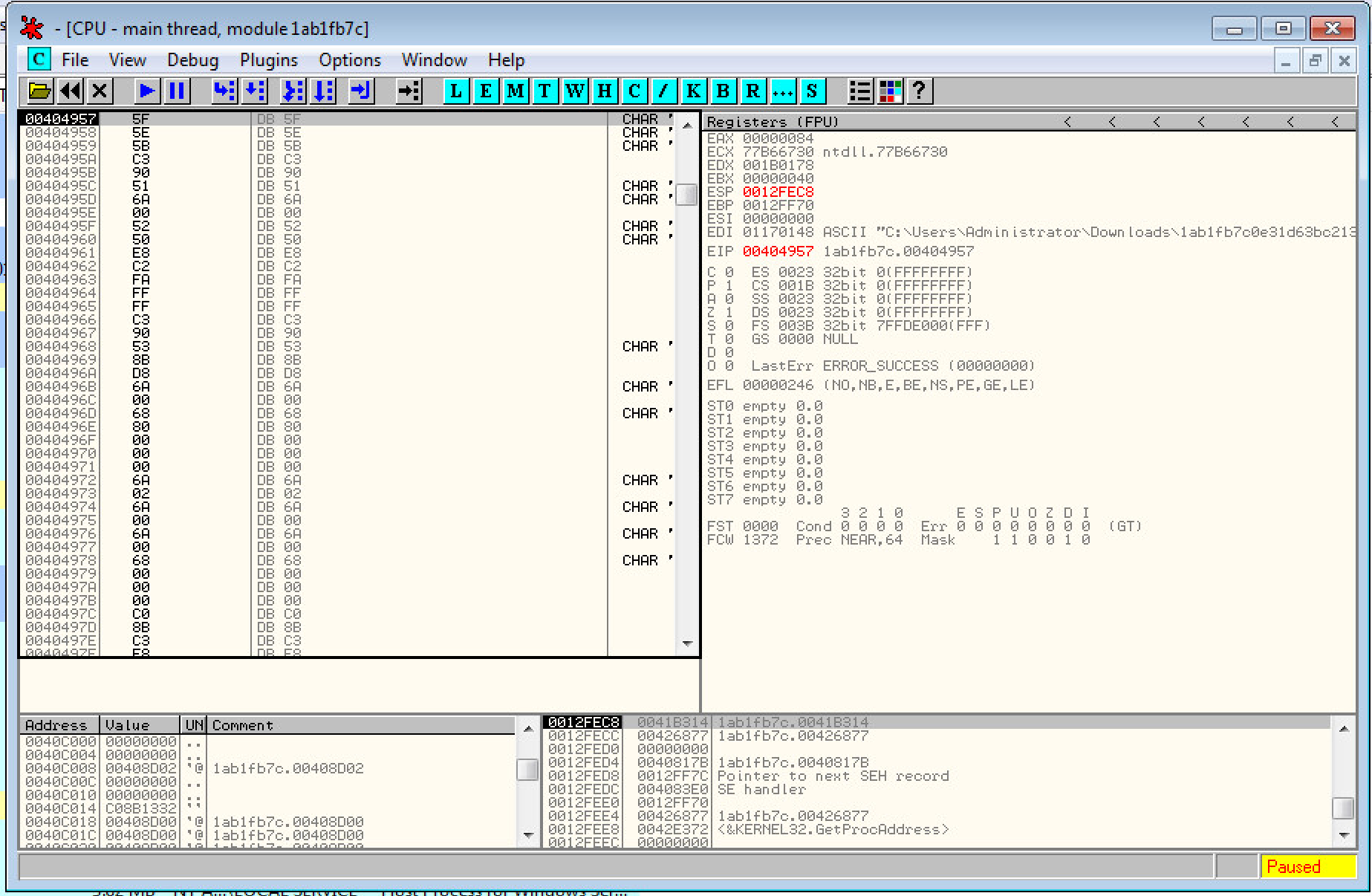


Figure 19: Returned from CreateFileW

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. The code above has not been properly interpreted by OllyDbg’s linear sweep disassembler. Fix this by removing the analysis for the current module, using the menu options shown in Figure 20.

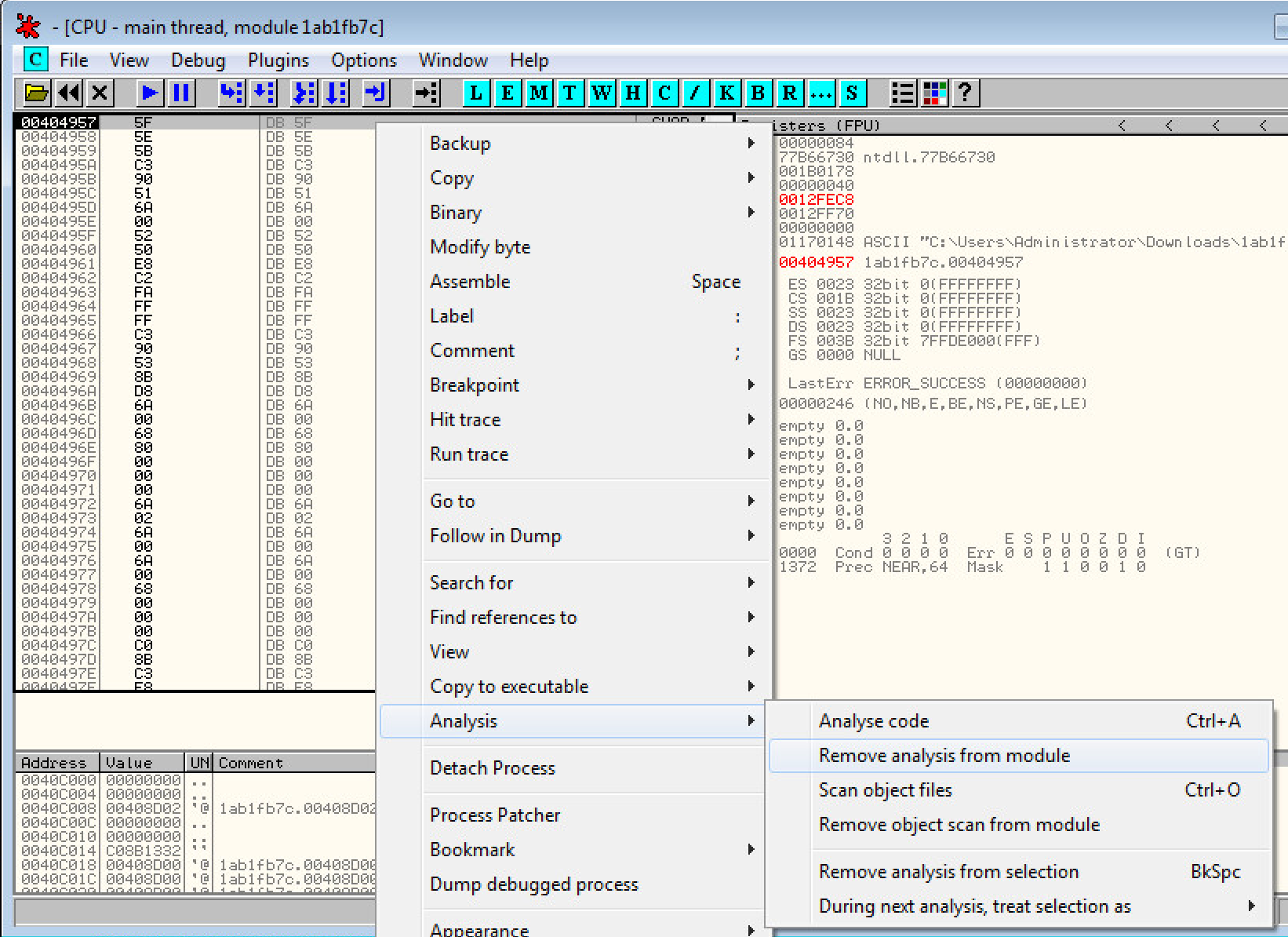


Figure 20: Remove Analysis from Module

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

The resulting address provides the section from which you will execute unpacked code.

1. Now that you’re aware of this section, select **Debug > Set condition** to restart the binary and set up a trace that runs until EIP points to an address within this section.

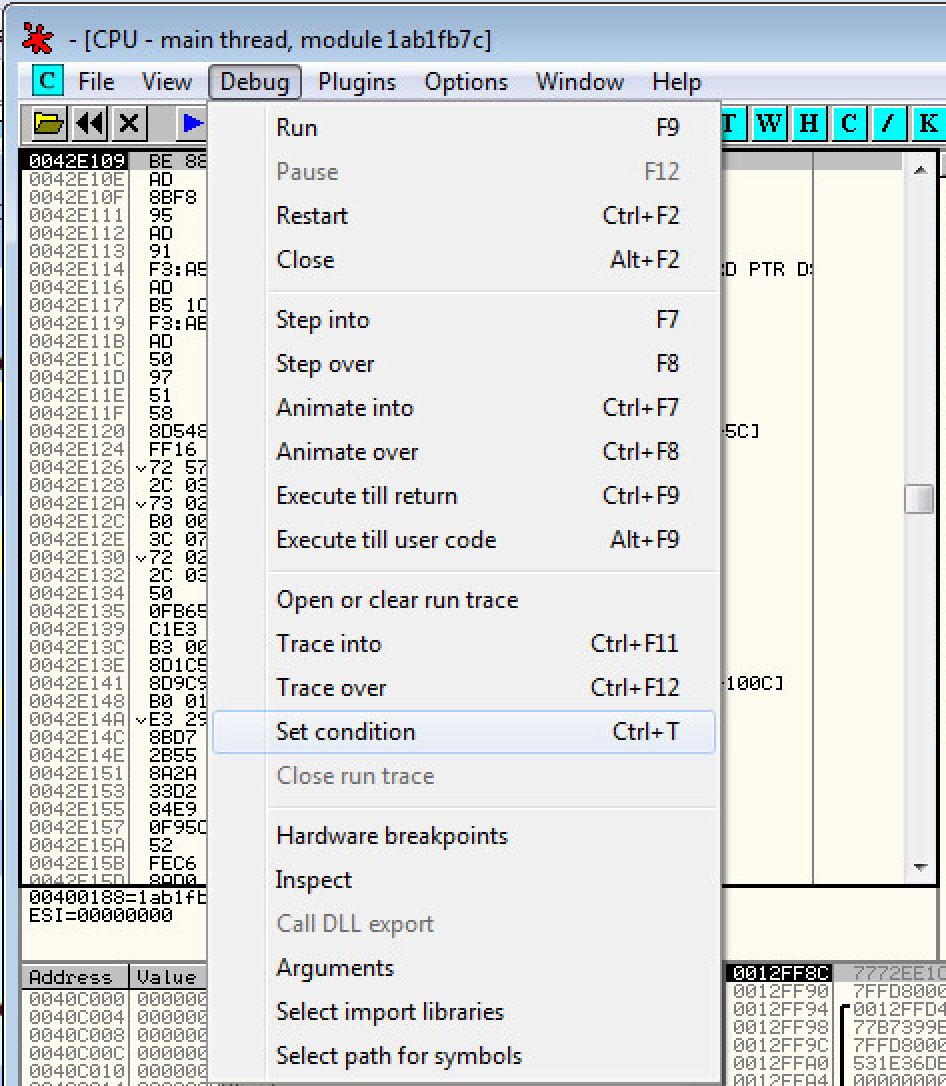


Figure 21: Set Trace Condition Select

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

The *Condition to run pause trace* window appears.

1. Enter the section’s address range as follows and click **OK**.

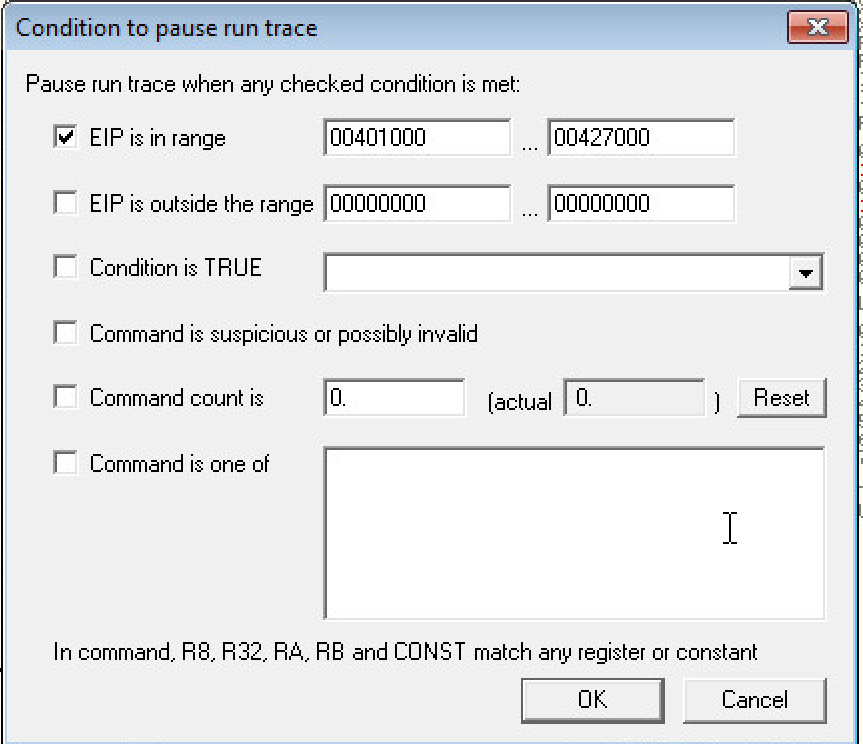


Figure 22: Condition to Pause EIP Section Range

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

1. From the menu, select **Debug > Trace into** to trace into the given range.

This process can take a substantial amount of time, because after every instruction is executed the debugger needs to check the EIP register needs to see if it is within that range.

After tracing into the instructions, the OEP is hit at address 0x40B0B0.

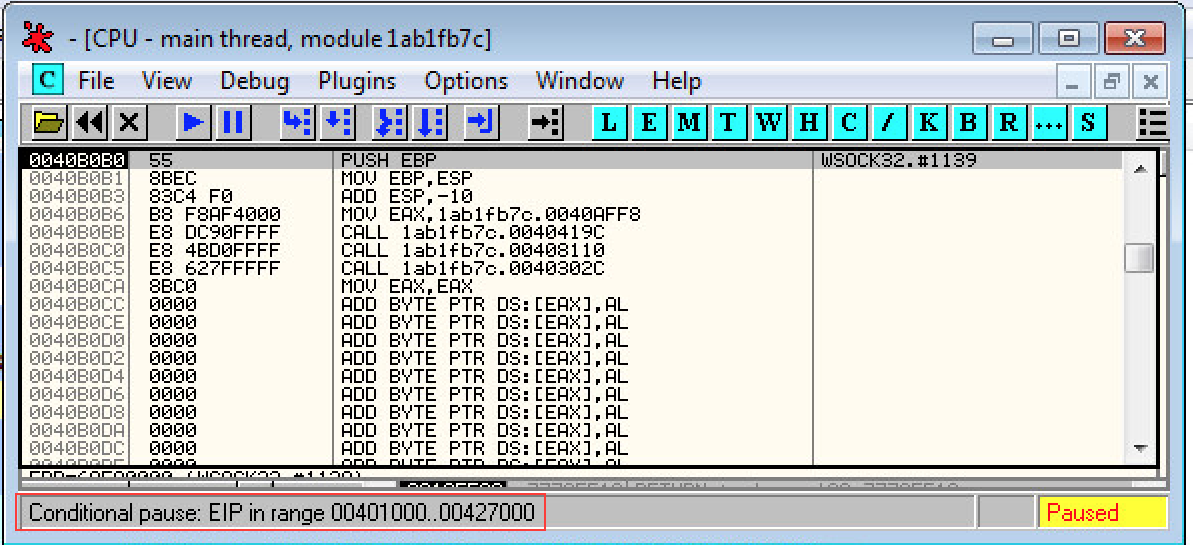


Figure 23: EIP Breakpoint within Section Range

Source: OllyDbg software, 2016. Reproduced and used in accordance with the fair dealing provisions in section 29 of the Canadian Copyright Act for the purposes of education, research or private study. Further distribution may infringe copyright.

Setting this condition stops execution once OllyDbg observes EIP as being within the range of 0x401000-0x427000 (e.g., the .UPack section identified with Hiew). The unpacking code writes to this section from the .rsrc section, which is then jumped to, thus accomplishing a section hop.

1. Demonstrate to your instructor that you’ve set up the trace correctly.

**Instructor sign-off:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Once you hit the OEP (from the trace or your own breakpoint due to time constraints), dump the binary and repair that IAT for analysis using the steps outlined with the UPX binary.

**Instructor sign-off:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Analysis

NOTE FOR INSTRUCTOR

**The samples this question cannot be accessed as the last section could not be completed, so I instead used the cmd.exe samples that we generated in the first section.**

1. Open two instances of IDA Pro, open the original dumped binary (not dumped) in the first, and open the dumped binary with the fixed IAT in the second. What imports are now available in the fixed binary that were not available in the first?

**Many many imports, too many to list, there is 71 more imports available in the fixed binary.**

1. What registry key is written to by this sample?

**Due to the fact we do not have access to IDA Pro, we cannot access the registry keys using the alternate program PEstudio so we cannot answer this question or the following one.**

1. What value is written to this key?

**See previous answer**

1. What is the name of the DLL being dropped by the sample?

**kernel32.dll**

1. Many file handles are being opened by this sample. What types of files are these and why are they being opened?

**No files are being opened**

1. Based on the above observations, what type of malware would you define this as, using the provided taxonomy?

**This is not malware.**

**Instructor sign-off:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Sign-Off: Lab 8 – Malware Packers

Detach this page and submit it to your instructor.

Name: Coleton Sanheim

Student ID: 000862545

|  |  |
| --- | --- |
| **Item** | **Initial** |
| Unpacking UPX |  |
| Tracing UPack |  |
| Unpacking UPack |  |
| Analysis |  |

# References

LordePE (Version 1.31) [Computer software]. Retrieved from http://www.softpedia.com/get/Programming/File-Editors/LordPE.shtml

MackT. (2010). Import REConstructor (Version 1.7e) [Computer software]. Retrieved from https://tuts4you.com/download.php?view.415

PEiD (Version 0.95) [Computer software]. Retrieved from http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml

Suslikov, E. (2017). Hiew (Version 8.53) [Computer software]. Retrieved from http://www.hiew.ru/

Yuschuk, O. (2014). OllyDbg (Version 2.01) [Computer software]. Retrieved from http://www.ollydbg.de/