|  |
| --- |
| George Mason University |
| Malware Analysis Manual unpacking |
| ISA 764 Security Experiment Lab Design Project |
|  |
| **Chu-Feng Huang** |
| **G00379170** |

|  |
| --- |
|  |

# Introduction

Recent years, malware threats are significantly increased, antivirus applications have hard time to catch up to identify a given executable is a malware or not. Malware authors tend to heavily use various obfuscation techniques such like packer to alter malware payload to bypass antivirus detection, and to increase the difficulty for others to perform static analysis. More than 80% of malware is packed in today’s malware, this number is observed by Symantec Research Laboratories[[1]](#footnote-1) from various antivirus vendors. Significant number of malware is packed has alter the way to perform static analysis, generally there are three approaches to perform unpack, manual unpacking, static unpacking, and generic unpacking,

in this lab, we will perform manual unpacking approach which utilize debugger, Ollydbg, and other tools[[2]](#footnote-2) to analyze and to restore hidden payload to its original form.

Since this packer problem is a serious threat, I design this lab for students to understand what packer is, and go a little bit further into reverse engineer to learn how to unpack it for static analysis.

# Packer

In this section we will look into how packer works, but first we will take a look portable executable (PE) format. Below is a brief on “PE Header” and “Section Headers”, for complete detail on PE format, refer to [7], [8].

## PE Header

* Signature: The PE header starts with “PE\0\0”.
* IMAGE\_FILE\_HEADER: This structure holds information includes PE header signature without this “PE\0\0” signature, an executable will never be executed. Other fields are listed below,
  + Machine
  + NumberOfSections
  + TimeDateStamp
  + PointerToSymbolTrable
  + NumberOfSymbols
  + SizeOfOptionalHeader
  + Characteristics
* IMAGE\_OPTIONAL\_HEADER // required by the linker and loader in Windows
  + AddressOfEntryPoint //Relative Virtual Address(RVA) of the entry point
  + ImageBase // the first byte of image when loaded into memory
  + SectionAlignment
  + FileAlignment
  + SizeOfImage
  + SizeOfHeaders
  + Subsystem
  + DataDirectory

## Section Headers

* IMAGE\_SECTION\_HEADER // each section represent code or data
  + Name
  + VirtualSize
  + VirtualAddress
  + SizeOfRawdata
  + PointerToRawData
  + Characteristics
  + …



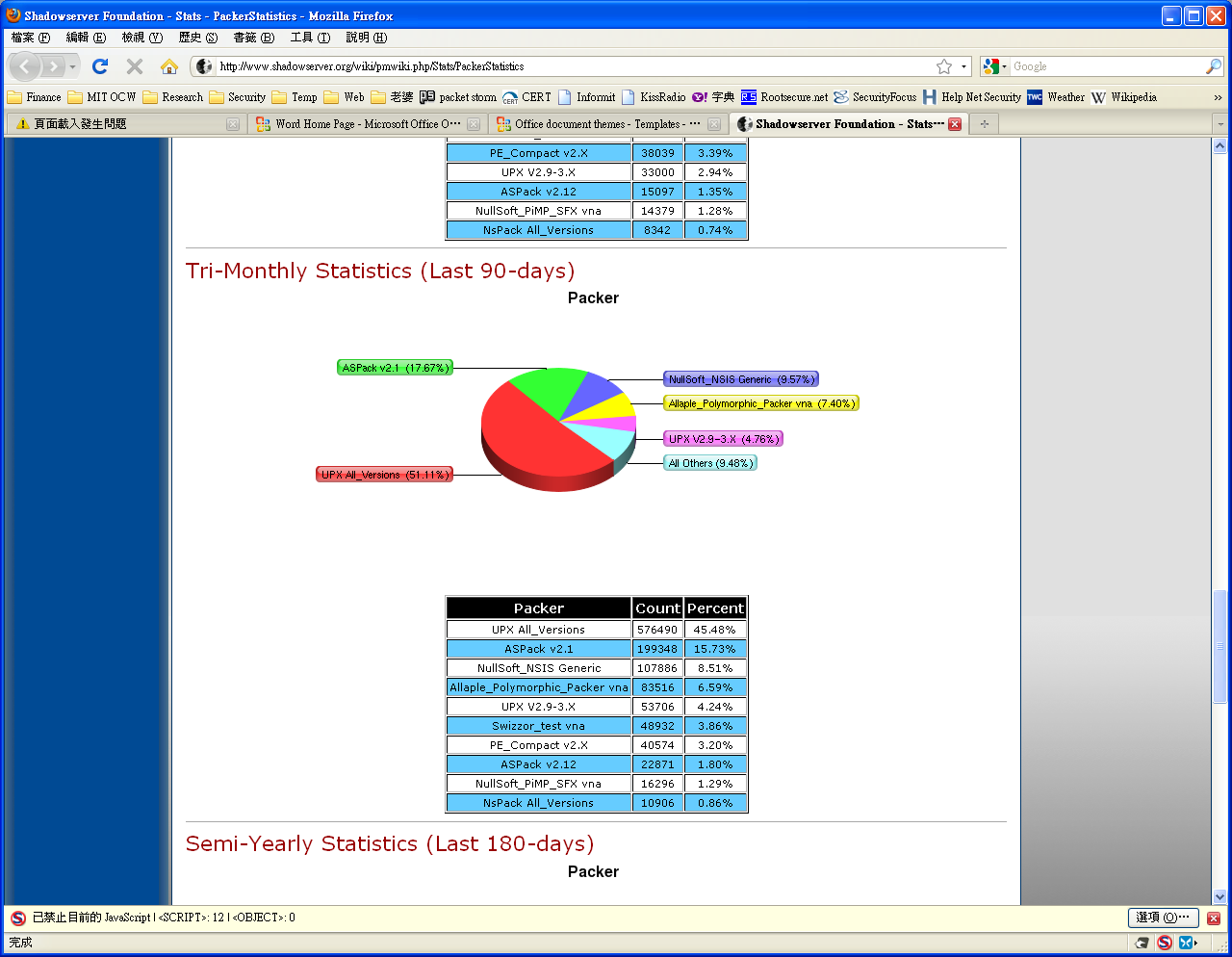
Figure PE format

There are comments after the “AddressOfEntryPoint”, and “ImageBase”, which “AddressOfEntryPoint” is a relative virtual address (RVA), and “ImageBase” is where to map the PE image. In PE format, virtual address is the sum of “AddressOfEntryPoint” and “ImageBase”.

VA = “AddressOfEntryPoint” + “ImageBase”.

For a typical packer, the following task would be perform during packing for inserting new code into PE image,

* Add new section: It would add a new IMAGE\_SECTION\_HEADER.
* Modify Entry Point: Save the original entry point (OEP) and then modify AddressOfEntryPoint, and updating the SizeOfImage in IMAGE\_OPTION\_HEADER.
* Insert loader: A loader is an execution routing to uncompress and/or decrypt the executable in memory, and to load the import of the original program. The following is a list of tasks performed by the load,
  + Self decryption of loader
  + Decompression and decryption of the sections in memory
  + Relocation handling for DLLs
  + Import table handling
  + Jump to the OEP
* Compression/Encryption: Transform the original executable into compressed or encrypted form.

In Figure 2, it shows more than 50 percent of malware are packed using UPX (All versions) during 9/06/2009 and 12/05/2009, and UPX packer is considered not as challenge to perform reverse engineer as other packer, and it also widely used. For this lab, the latest version of UPX (version 3.04, at the time of writing.) would be used. UPX packer uses compression (NRV compression library) and API Redirection to obfuscate the binary and prevent from easily rebuilding the import able.

09/06/2009 ~ 12/05/2009

Figure [[3]](#footnote-3)

# Lab Setup

## Environment

This lab is targeted to perform on Windows OS, but this can also be done in Linux with WineHQ[[4]](#footnote-4) which is software that allow you to run windows applications on Linux platform and necessary DLL files copy from Windows.

## Tools used

Refer to Tools section for all the tools used during the lab.

## Sample Binary used for analysis

Please find attachment 1 for source code of executable sample, and the compiled executable is also included in the submitting,

* “ISA764Demo\_NotPacked.exe” for compiled but not packed.
* “ISA764Demo\_packed.exe” for UPX packed version.

# Tools

|  |  |
| --- | --- |
| OllyDBG | <http://www.ollydbg.de/> |
| OllyDump | <http://www.openrce.org/downloads/details/108/OllyDump> |
| PEiD | <http://www.peid.info/> |
| Import REConstructor | <http://tuts4you.com/download.php?view.415> |
| HexEdit | <http://sourceforge.net/projects/hexedit/> |
| PEditor | <http://www.tuts4you.com/download.php?view.421> |
| UPX | <http://upx.sourceforge.net/> |

# ToDos & Solution

**† Solutions are highlighted with light blue background.**

## ****Task I****

**Task I.1:** Use PEiD tool to identify what packer tool is used to pack “ISA764Demo\_packed.exe” and what is its exact version?

UPX 3.04

**Task I.2:** Use PEidtor tool to find and report sections contain in “ISA764Demo\_packed.exe” and “ISA764Demo\_NotPacked.exe”.

|  |  |
| --- | --- |
| ISA764Demo\_packed.exe | ISA764Demo\_NotPacked.exe |
| UPX0 | .textbss |
| UPX1 | .text |
| .rsrc | .rdata |
|  | .data |
|  | .idata |
|  | .rsrc |

## Task II

**Task II.1**: Use OllyDBG to find and report Original Entry Point (OEP) address of “ISA764Demo\_packed.exe”.

OEP: 0042D920

**Task II.2:** Use OllyDump plugin to dump the unpacked binary into a file. To use OllyDump, the OllyDump.dll file needs to be placed in the same folder OllyDBG is, OllyDBG will pick up OllyDump automatically when start up. When perform dump, enter the OEP found in Task II.1, and UNCHECK “Rebuild Import” option. Dump to a file named “dump.exe”, and submit this binary.

A binary named “dump.exe”.

**Task II.3:** Look into PE Header in the OllyDBG memory window, find and report Import Address Table (IAT) address, and IAT size.

IAD: 0049B1F4

Size: E0(224)

## Task III

**Task III.1:** Use ImpREC tool to fix API redirection. To run ImpREC tool and to perform fix dump, OllyDBG must run and “ISA764Demo\_packed.exe” is loaded. Fill in “IAT Info Needed”, and “Get Imports” to retrieve all import functions. Right click on invalid function and select “hex view” to verify the content, remove chunk if not valid, until all invalid functions are removed. Report all the valid import functions.

* Kernel32.dll
  + LoadLibraryA
  + GetProcAddress
  + VirtualProtect
  + VirtualAlloc
  + VirtualFree
* User32.dll
  + MessageBoxA

**Task III.2:** After all invalid functions are removed, using “Fix Dump” to fix the “dump” from TaskII.2 and save as “dump\_fixed.exe” and submit this working unpacked binary.

A binary named “dump\_fixed.exe”.

# Bibliography

1. (n.d.). Retrieved 12 01, 2009, from TUTS4YOU: http://www.tuts4you.com/
2. (n.d.). Retrieved 12 01, 2009, from The Reverse Code Engineering Community: http://www.reverse-engineering.net/
3. (n.d.). Retrieved 12 01, 2009, from Shadow Server: http://www.shadowserver.org
4. (n.d.). Retrieved from Frank Boldewin's www.reconstructer.org: http://www.reconstructer.org/
5. (n.d.). Retrieved from OpenRCE: http://www.openrce.org/downloads/
6. Guo, F., Ferrie, P., & Chiueh, T.-c. (2008). A Study of the Packer Problem and ITs Solutions. *11th international symposium on Recent Advances in Intrusion Detection*, (pp. 98 - 115). Cambridge, MA.
7. *Microsoft Portable Executable and Common Object File Format Specification.* (n.d.). Retrieved from Microsoft: http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx
8. Pietrek, M. (n.d.). *An In-Depth Look into the Win32 Portable Executable File Format.* Retrieved from MSDN: http://msdn.microsoft.com/en-us/magazine/cc301805.aspx
9. Yan, W., Zhang, Z., & Ansari, N. (2008). Revealing Packed Malware. *IEEE Computer Society* .
10. Yason, M. V. (2007). The Art of Unpacking. *BackHat USA 2007.*

# Attachment 1

Source code for “ISA764Demo” program.

#include <windows.h>

int WINAPI

WinMain(HINSTANCE hInst, HINSTANCE hPrev, LPSTR pszCmdLine, int iCmdShow)

{

MessageBox(NULL, "ISA 764 Lab Design Project\nDemo Program", "ISA 764", MB\_OK | MB\_ICONEXCLAMATION);

return 0;

}

1. From reference “A Study of the Packer Problem and Its Solutions”. [↑](#footnote-ref-1)
2. Please refer “Tools” section for list of tools used in this lab. [↑](#footnote-ref-2)
3. <http://www.shadowserver.org/wiki/pmwiki.php/Stats/PackerStatistics> [↑](#footnote-ref-3)
4. WineHQ can obtain from http://www.winehq.org/. [↑](#footnote-ref-4)