14 March 2008

Comparison of File Infection on the Windows and Linux

Disclaimer:

The author of this document is not responsible of any kind of damage that could be made with the bad use of this information. The objective of this paper is for educational and research purposes only. It is made for use in viruses, but not as to promote any intentional harm or damage on computer systems.

Author: lclee_vx

lclee_vx@yahoo.com

1.0 Foreword / Introduction

This paper documents the common file infection strategies that virus writers have used over the years, conduct the comparison of Portable Executable (PE) file infection on the Windows platform and Executable and Linking Format (ELF) file infection on the Linux platform.

So, let's set the goal: I will go through the file format of PE and ELF, demonstration, source code, examples included along with the introduction of simple file infection method on Windows and Linux. Here are the two ways (file infection) I will present here:

1. Appending to the PE file with adding a new section – Windows Platform
2. Writes parasite code at entry point and the original data will be stored at end of file – Linux Platform

Note:
This article is never perfect, so notify me the possible mistakes in this document for further updates. Contact me:

Email : lclee_vx@yahoo.com
Group : F-13 Labs
Personal Web Site : http://www.f13-labs.net

2.0 Useful Things for Coding

You need some tools/references before start code the virus on the Linux/Windows platform. As below:

Windows:

1. The tasm 5.0 package – Win32 Assembly Language compiler
2. The API list (Win32 API help file)
3. PE file format – Strongly recommended Matt Pietrek document
4. Basic knowledge on Win32 Assembly Language
5. Assembly IDE – RadASM version 2.2.0.2
6. Debugger – OllyDbg version 1.10
7. Platform – Windows XP

Linux:

1. Nasm (Netwide Assembler) version 0.98.39 – Assembly Language Compiler (Linux)

2. i386-PC-Linux System Call Reference
3. ELF file format
4. Basic knowledge on Linux assembly language
5. IDE/Editor – vi
6. Debugger – Data Display Debugger (DDD) version 3.3 and GNU Project Debugger (GDB)
7. Platform – Red Hat 8.0

3.0 Introduction of Portable Executable (PE) File Format

Win32 refers to the Application Programming Interface (API) available in Windows operating systems. It is the set of system functions that are part of the operating system and that are available to be called form a Win32 (32-bit) Windows application.

Basically, the i386 architecture has four privilege levels, also known as rings (Ring0 – Ring3) that control the things such as memory access and access to certain sensitive CPU instructions. As you noticed we work in a 32 bit environment on the Windows platform, which means that memory address are 32 bit (00000000h – FFFFFFFFh) and the memory layout is as below:

00000000h – 3FFFFFFFh	Application code and data
40000000h – 7FFFFFFFh	Shared memory (system dll's)
80000000h – BFFFFFFFh	Kernel
C0000000h – FFFFFFFFh	Device Drivers

Detail information can get from Microsoft website. Here, we will code the simple virus, infect the PE file in level Ring3 (00000000h – 3FFFFFFFh) and adding another new section. Let's see how a virus can change an executable header in the following sections.

Before start the game, it is very important to have cleared the structure of the PE header, offset of the section and PE format layout. PE stands for Portable Executable. It is the native file format of Win32 such as binary programs (exe, dll, sys, scr) or object files (bpl, dpl, cpl, ocx, acm, ax). The meaning of 'Portable Executable' is that the file format is universal across win32 platform such as Windows 98, 2K and NT. The PE loader of every Win32 platform recognizes and uses this file format even when Windows is running on CPU platforms other

than Intel. Like other file formats, PE has different areas called sections such as .text, .data, .rdata, .bss and .reloc.

The most important thing to know about PE files is that the executable codes on disk do not need relocation for library calls anymore. Instead, the import address table (IAT) is used for that functionality by the system loader. It is also important to note that PE files are not just mapped into memory as a single memory-mapped file. Instead, the system loader looks at the PE file and decides that what portions of the file to map in.

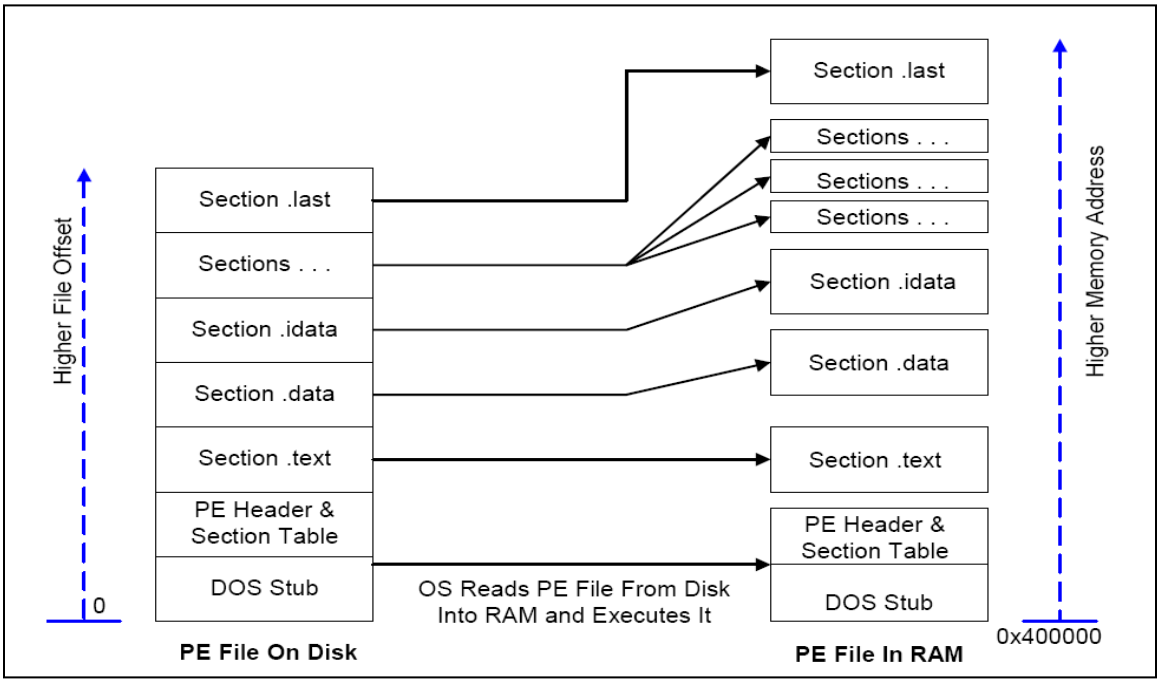

Figure 1 PE File Layouts on Disk and in RAM

Figure 1 is a diagram showing a Portable Executable (PE) file layout on disk and RAM when executed by a Win32 operating system. The details information of every section in PE file format can refer to 'Overview of PE file format' by Iczelion.

The Table 1 as below is the summary of important fields in the PE header, Section Table and Import Table in the process of PE file injection.

	Important Fields	**Functionality**

PE Header	Machine	Which CPU this file intended for
		Checked by viruses to ensure they only infect x86 platforms
	NumberOfSections	The number of sections in the file
		This field is updated after virus adds a new section.
	Characteristics	What type of file this is (Exe or DLL)
	SizeOfCode	Size of all the code sections
	AddressOfEntryPoint	The relative virtual address (RVA) where execution begin
		Viruses change this to point to the virus code
	ImageBase	First byte of image in memory
	SizeOfImage	The size of the image
Section Table	VirtualSize	Total size of the section in memory
	SizeOfRawData	Size of the section on disk
	Characteristic	What kind of section this is
The Import Table		Viruses use the import table to lookup the address of any API functions they need to call.

Table 1 Summary of Important Fields in PE Files

4.0 Introduction of Executable and Linking Format (ELF)

The Executable and Linking Format was originally developed by UNIX System Laboratories (USL) as part of the Application Binary Interface (ABI). The Tool Interface Standards committee (TIS) has selected the evolving ELF standard as a portable object file format that works on 32-bit Intel Architecture environments for a variety of operating systems. There are three types of object files:

1. relocatable file – holds code and data for linking with others object files
2. executable file – holds a program suitable for execution
3. shared object file – holds code and data suitable for linking in two contexts.

The object file format provides parallel views of a file's contents, reflecting the differing needs of these activities.

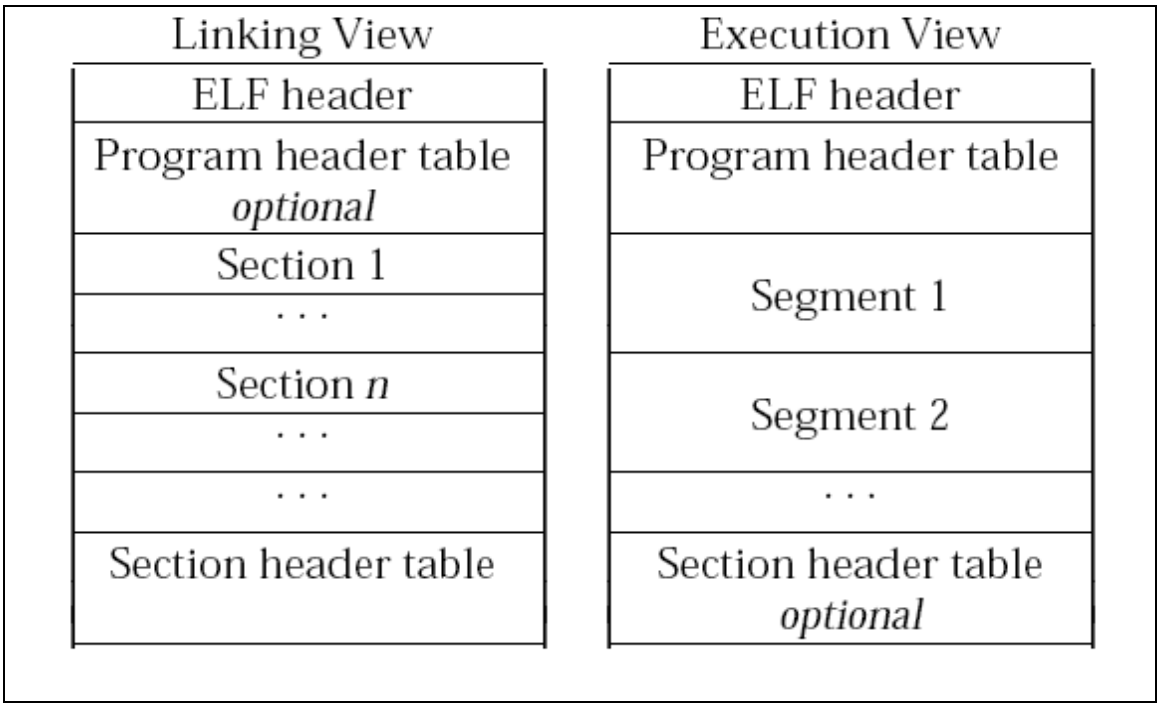

Figure 2 ELF File Layout

An ELF header resides at the beginning and holds a "road map" describing the file's organization. It provides information such as offsets to program header and section header tables, sizes and number of entries.

A section header table is used to locate and interpret all of the files sections. The table is an array [e_shnum] of Elf32_Shdr structures, holding information about section sizes, locations and virtual addresses.

A program header table is used to describe segment information the system needs to prepare in program loading for execution. It holds information such as virtual addresses, file size, segment attributes and so on.

The summary of the important fields in ELF file format as below:

	Important Fields	**Functionality**
Elf32_Ehdr	e_ident	Holds the magic values 0x7f, 'ELF' and some flags
	e_entry	Virtual address of entry point
	e_ehsize	Size of the ELF header
	e_phentsize	Size of one entry in the program header
	e_phnum	Numbers of entrys in the program header
Elf32_Phdr	p_vaddr	Virtual address in memory
	p_addr	Physical address
	p_memsz	Size of the segment in memory

5.0 Demonstration of PE/ELF File Infection

Let's make a quick review of the File Infection process on the Windows and Linux. Please refer to the attachments for the source code of PE/ELF infector.

	PE Infection	**ELF Infection**
1.	Get the delta offset – where executing the code	Get the delta offset – where executing the code
2.	Get the Kernel32.dll address	Control access to a region of memory, all the system call can access with **int 80h**
3.	Get the API functions as below: - LoadLibraryA - GetProcAddress - GetCurrentDirectoryA - SetCurrentDirectoryA - FindFirstFileA - FindNextFileA - FindClose - GetFileAttributesA - SetFileAttributesA - CreateFileA - GetFileSize - GlobalAlloc - ReadFile - SetFilePointer - WriteFile - GlobalFree	Scan the target file in current directory

	- CloseHandle - ExitProcess	
4.	Scan the target file in current directory	Open the file to see if it is infected
	Open the file to see if it is infected	If infected, exit and return to the host program
	If Infected, search for another file (maximum 3 files)	Else, virus infects target file by overwriting host code by viral code. The original host code is stored at the end of host file.
5.	Else, File Injection with adding the new section	Exit and return control to the host program
6.	Copy virus body into the section	
7.	Exit and return control to the host program	

The detail demonstrations please refer to the attachment.

6.0 Conclusion

There are several reasons for the non-issue of the Linux virus. For a Linux binary virus to infect ELF executables and spread, those executables must be writable by the user activating the virus. That is not likely to be the case. Chances are, the files/programs are owned by power user such as root and the user is running from a non-privileged account. Second, even if the Linux virus successfully infects a program owned by the user, its task of propagation is made much more difficult by the limited access right of the user account. As we know, Linux applications and software is almost all open source. Binary-only products are rare and this is a tough place for a Linux virus to hide. Each of the above reduces the reproduction rate of the Linux virus.

Reference

1. Szor, Peter. Attacks on Win32. Virus Bulletin Conference, October 1998, Munich/Germany, page 57-84.

2. Inside Windows: An In-Depth Look into the Win32 Portable Executable File Format: http://msdn.microsoft.com/msdnmag/issues/02/02/PE/default.aspx.

3. Microsoft Portable Executable and Common Object File Format Specification: http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx.

4. Silvio Cesare, 1999. Unix Viruses

5. Billy Belcebu, 1999. Viruses under Linux, Xine – Issue #5

6. @Computer Knowledge 2000, 2000. Computer Knowledge Virus Tutorial

Credit:

1. The Linux ELF infector is inspired of Winux virus of Benny/29A
2. Billy Belcebu, Virus under Unix
3. izee, skyout, robinh00d, synge, moaphie