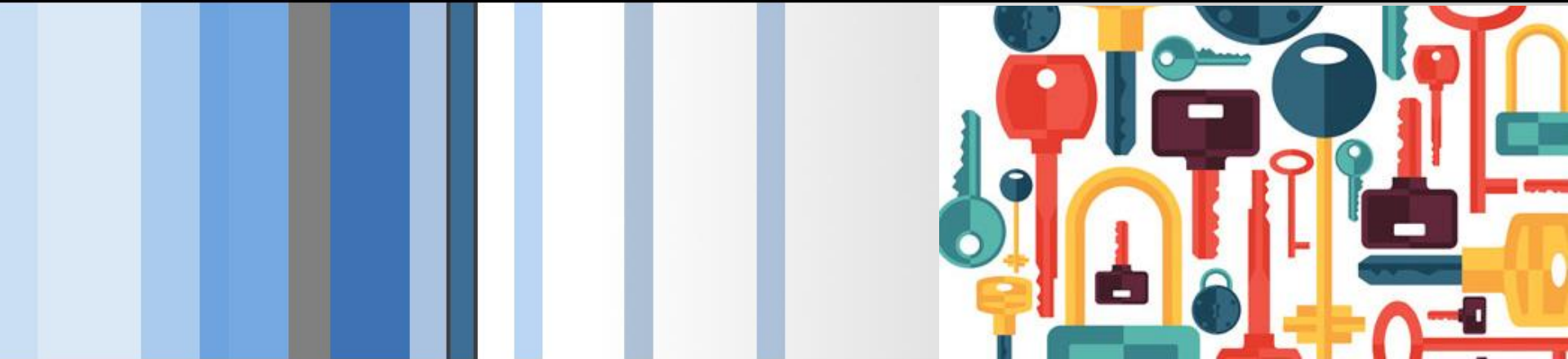
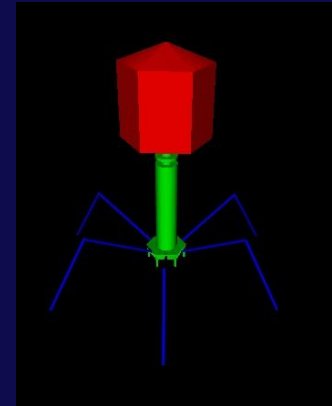


CSC 600 Advanced Seminar
CVE-2008-4250 RPC and Conflicker Worm
Si Chen (schen@wcupa.edu)

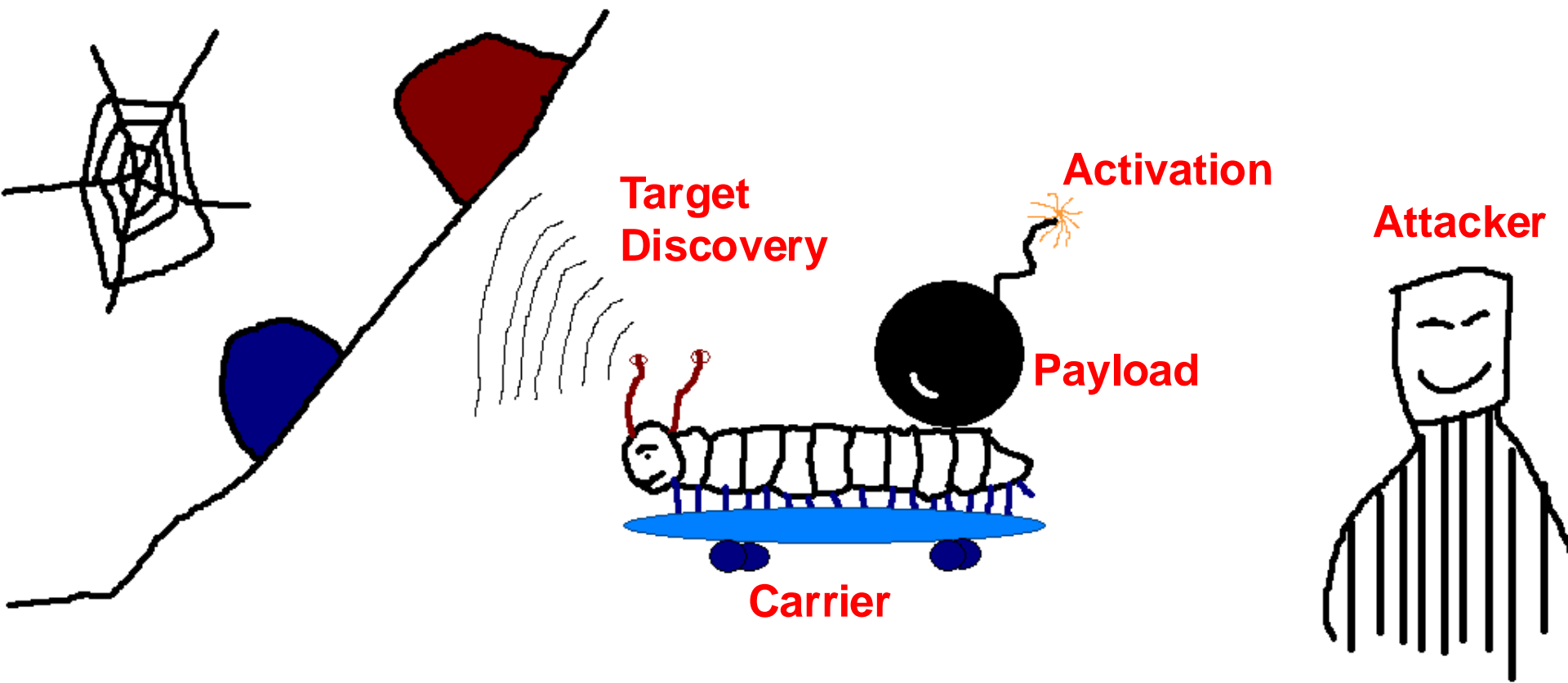


Worm vs a virus



1. Self propagates across the network
2. Exploits security or policy flaws in widely used services
3. Less mature defense today

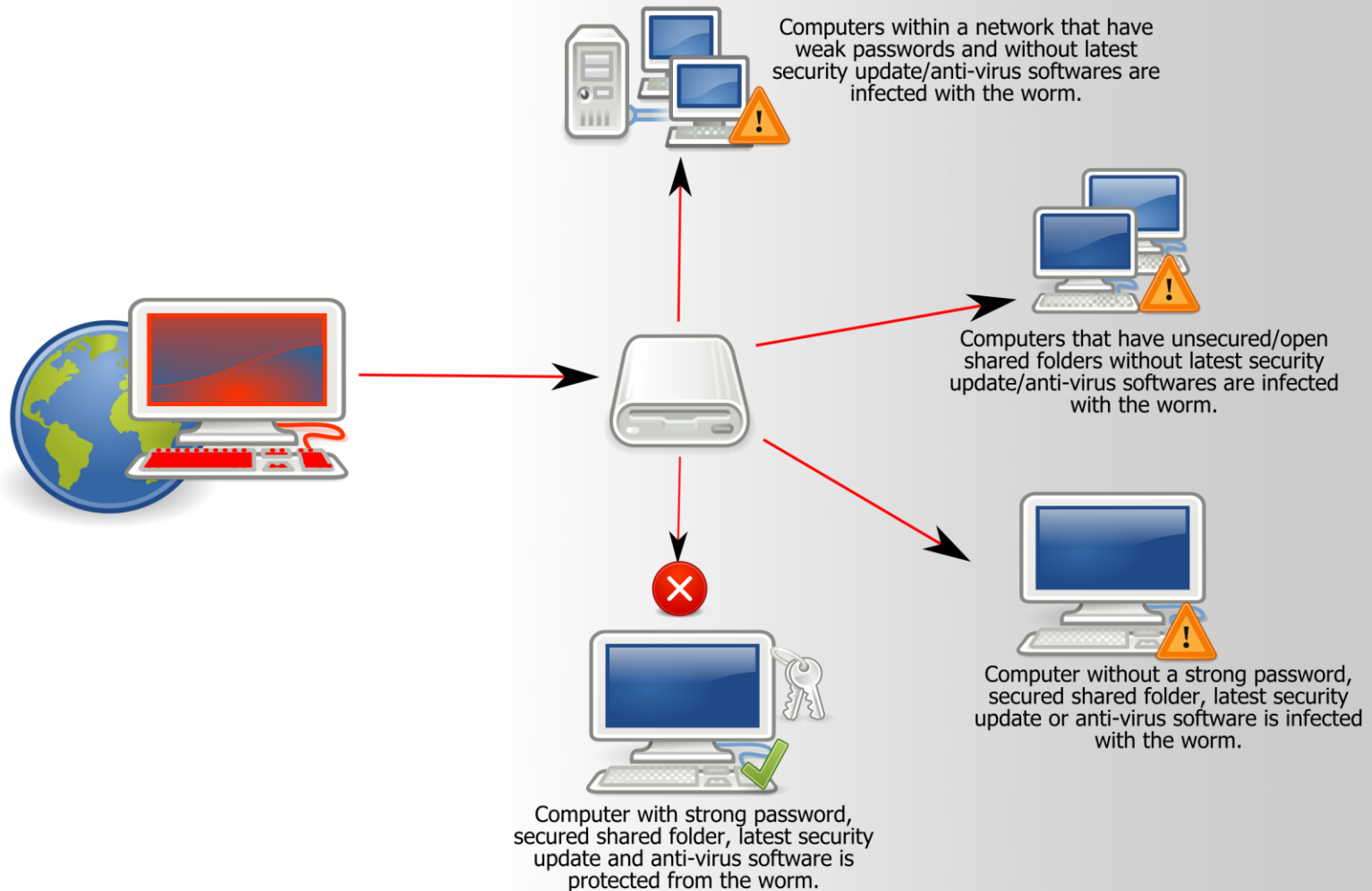




OVERVIEW

CVE-2008-4250 (MS08-067) & Conficker Worm

Worm:Win32 Conficker

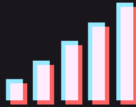


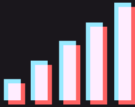
- In October 2008, Microsoft urgently released a critical security patch to fix the threat posed by the CVE-2008-4250 vulnerability (internally known as MS08-067). Since this patch was not released on Microsoft's regular Patch Tuesday, it is called an **Out-of-Band** Update.


ATTACKERKB


CVE-2008-4250

Microsoft RPC Code Execution MS08-067

**ATTACKER VALUE
VERY HIGH**

**EXPLOITABILITY
VERY HIGH**

 0

 2

Preface

- The CVE-2008-4250 vulnerability that broke out at that time and the subsequent Conficker worm variants were a very serious security event that lasted for several months. Dustin Childs, the then Security Program Manager (SPM) at Microsoft Security Response Center (MSRC), recalled:

"At the time, I was personally surprised to see Microsoft's various departments working together to deal with this vulnerability. Our Microsoft headquarters, Indian and European branch teams were almost working around the clock. One thing that impressed me was that when we held the first Security Incident Response Process (SSIRP) meeting for the MS08-067 vulnerability, there were 15 people in the conference room, and many experts joined the meeting via telephone conference lines. After the person in charge explained the basic situation of the vulnerability, the atmosphere in the meeting suddenly fell into a momentary silence, because we knew **that a large number of worm viruses would accompany this vulnerability.**

Preface

From that moment on, **we understood that the battle had begun**. People who have not experienced such a large-scale event may not have the same experience. The people in the room were all information security experts, and they had personally dealt with super worm viruses such as Melissa, Nimda, Slammer, Sasser, and Code Red. Another interesting thing is that, due to the priority of emergency response, I only needed to explain the situation of the MS08-067 vulnerability, and I could immediately coordinate and allocate staff to participate in the response process. In response to this vulnerability, all Microsoft employees worked around the clock for 17 days..."

This demonstrates the severity of this vulnerability. Therefore, we have chosen this very unique and significant vulnerability for study.

Introduction

- Brief overview of CVE-2008-4250 vulnerability
- Connection between vulnerability and differences between "." and ".." in command-line operations

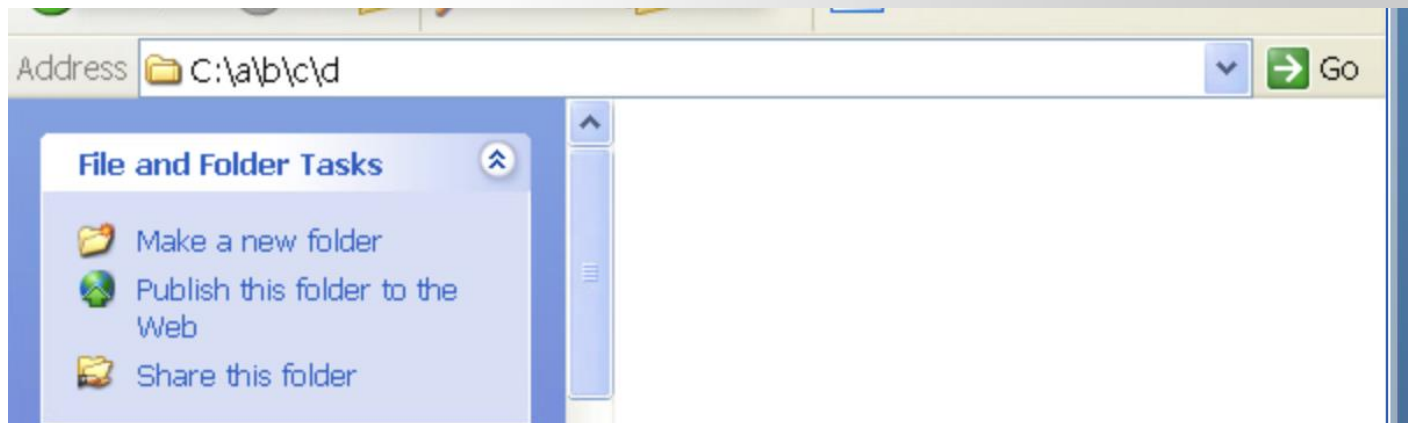
Brief overview of CVE-2008-4250 vulnerability

CVE-ID	
CVE-2008-4250	Learn more at National Vulnerability Database (NVD) <ul style="list-style-type: none">• CVSS Severity Rating• Fix Information• Vulnerable Software Versions• SCAP Mappings• CPE Information
Description	
<p>The Server service in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP1 and SP2, Vista Gold and SP1, Server 2008, and 7 Pre-Beta allows remote attackers to execute arbitrary code via a crafted RPC request that triggers the overflow during path canonicalization, as exploited in the wild by Gimmiv.A in October 2008, aka "Server Service Vulnerability."</p>	

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4250>

Differences between "." and ".."

- Before we delve into the CVE-2008-4250 vulnerability, I need to introduce some basic knowledge, as the cause of this vulnerability is related to the differences between "." and ".." in command-line operations, and how the program handling these two symbols.
- To illustrate this issue, I created a folder named "a" in the root directory of my C drive, and then created a folder named "b" inside "a" folder, which contains a "c" folder, and finally a "d" folder, as shown in the following hierarchy:



Differences between "." and ".."

- Then we open the command-line window and go to the root directory of the C drive. Normally, if we want to enter the "a" directory, we can simply enter the following command:

```
C:\Users\Administrator>cd\  
C:\>cd a  
C:\a>
```

- If we want to enter multiple directories, we can enter the following command:

```
C:\a>cd b\c\d  
C:\a\b\c\d>
```

Differences between "." and ".."

- And if we want to go back to the previous directory, we can enter:

```
C:\a\b\c\d>cd ..  
C:\a\b\c>
```

- If we enter a dot, it means we are still in the current directory and do nothing:

```
C:\a\b\c>cd .  
C:\a\b\c>
```

- That is, a dot represents the current directory, and two dots represent the previous directory. We can also use the following command to go directly back to the root directory:

```
C:\a\b\c>cd\  
C:\>
```

Differences between "." and ".."

- If we want to enter multiple directories at this time, besides the method mentioned above, there are actually several other ways, such as if we only want to enter the "a" directory, we can also write like this:

```
C:\>cd .\a
```

```
C:\a>
```

- Or write like this:

```
C:\>cd \. \a
```

```
C:\a>
```

- Since we can also enter the "a" directory like this:

```
C:\>cd \a
```

```
C:\a>
```

Differences between "." and ".."

- Therefore, before executing our command, the command line can actually perform a simplification operation, which is to convert ".a" or ".a" to "a" or "a" form, and remove the "." here.
- So much for the use of a dot. Next, there are two dots. For example, if we are in the current "a" directory and enter the following command:

```
C:\a>cd b\..\
```

```
C:\a>cd b\..
```

```
C:\a>
```

- It can be seen that these two commands do not change the current directory structure. This is because the "cd" command will help us enter the "b" directory, and the two dots mean to return to the previous level directory, which is the "a" directory, and then it is still the current directory. Separated writing is like this:

```
C:\a>cd b
```

```
C:\a\b>cd ..
```

```
C:\a>
```



Differences between "." and ".."

- That is to say, assuming that the directory hierarchy structure is not wrong, the writing method like "b.." or "b.." can be directly omitted. Then let's take a look at a slightly more complicated writing method. Still in the current "a" directory, enter the following command:

```
C:\a>cd b\.. \.. \a\b\c\d
```

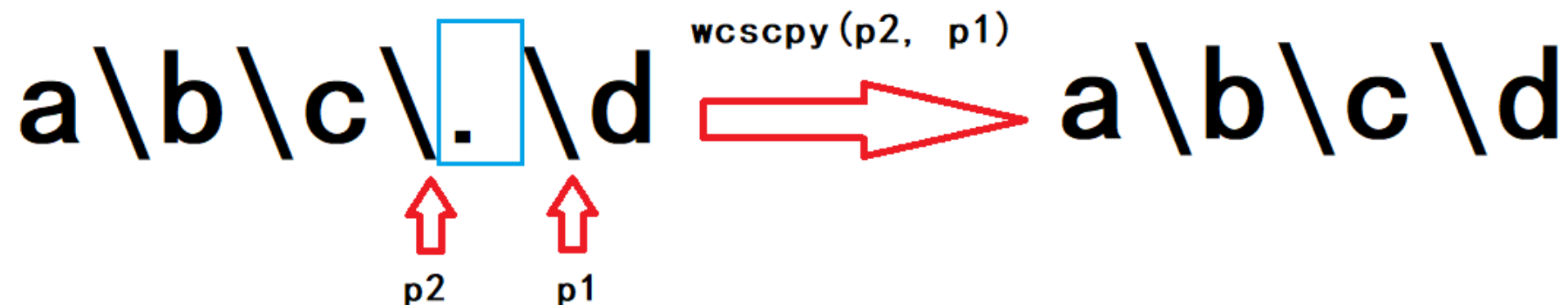
```
C:\a\b\c\d>
```

- The meaning of this command is to first enter the "b" directory in the current "a" directory, then return to the previous directory, that is, return to the "a" directory, and then return to the root directory of the C drive, and finally enter the "d" directory. According to the conclusion we just obtained, the writing method like "b.." can be directly omitted, so the path that the above command wants to enter is actually equivalent to "..abcd".

So, these are the basic knowledge we need to know about the dot symbol.

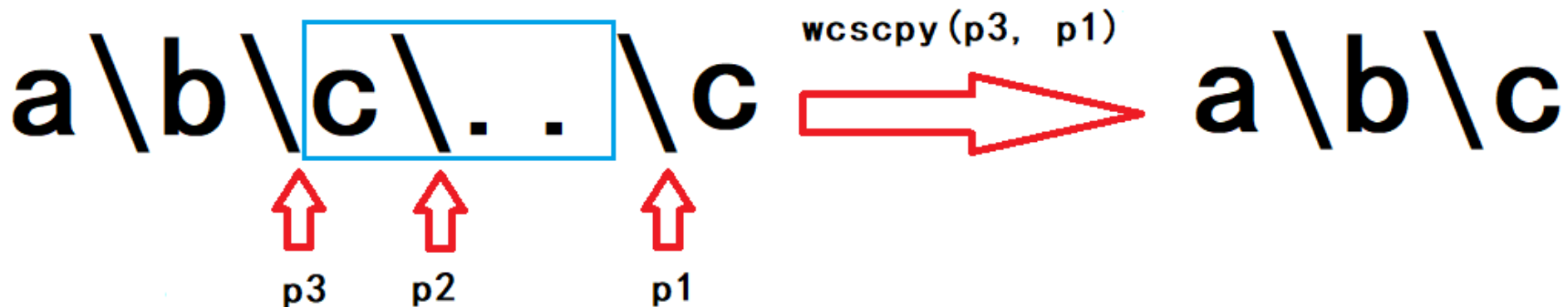
Programming the idea of simplifying directory structure

- Regardless of whether our command-line tool simplifies directories before executing our commands, one of the sub-functions in the NetpwPathCanonicalize function in our netapi32.dll has this feature. So here we need to implement two functions, one is the processing method for a dot. This situation is the simplest. Just remove the "." directly. However, our NetpwPathCanonicalize function **does not use deleting** functions to simplify strings, but **uses the `wcscpy()` function** to copy the contents of the left pointer to the right pointer, as shown in the following figure:



Programming the idea of simplifying directory structure

- Since the case with two dots also needs to remove the directory name in front of these two dots, in addition to the basic need for two pointers p1 and p2 to mark the addresses of the slashes on both sides of the dot, a pointer p3 is also needed to mark the position of the slash in front of the directory name to be removed, and then we can use the `wcscpy()` function to copy the contents pointed to by p1 to the position of p3.



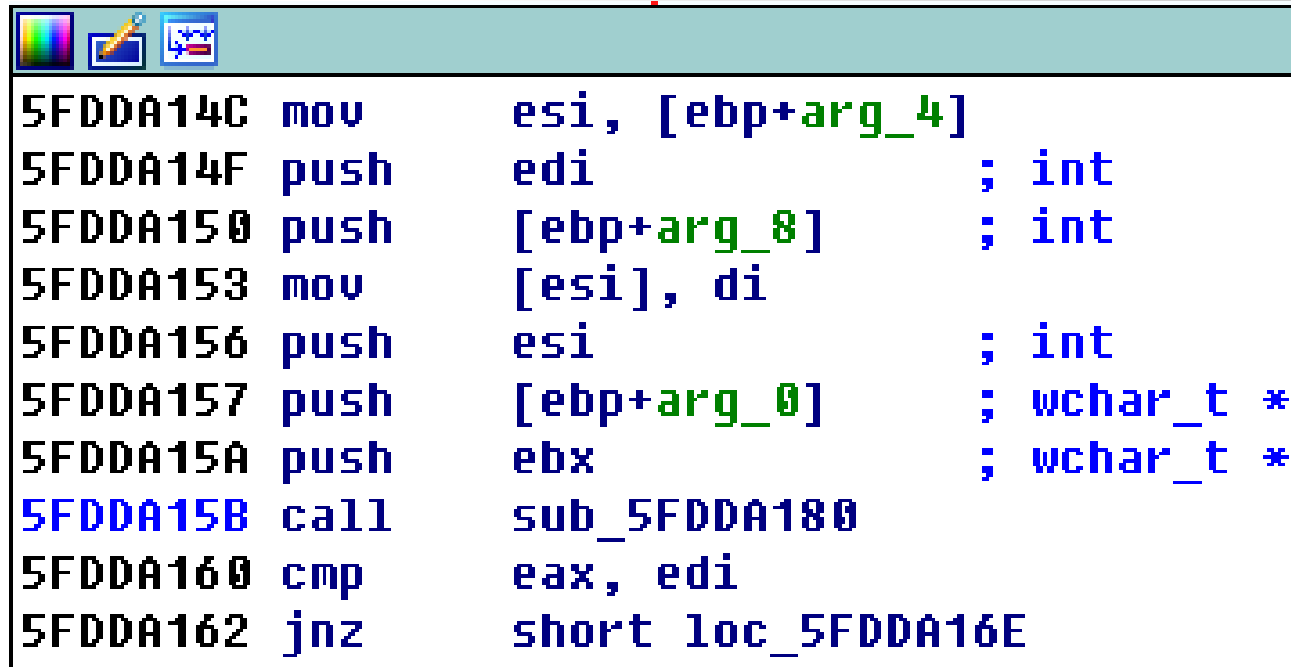
Conclusion

- The content we discussed in this course seems very simple, but even for such programming problems, a Microsoft engineer's negligence caused a serious vulnerability. In the next part, we will focus on the static analysis of this problem. But the premise is that you must thoroughly understand the content of this course.
- After all, the vulnerability research is a one-step-by-step process, and only by mastering these basics can we help us with our research and analysis in the future.

- The CVE-2008-4250 vulnerability we are studying this time is still in the **NetpwPathCanonicalize** function of the **netapi32.dll** file, but the location has changed and the idea is different.
- Its cause is due to a developer's negligence and lack of rigor in the string movement operation, which did not strictly check the out-of-bounds situation.

Static Analysis

- The function we are researching this time is the same as before, which is the path character function used to splice and normalize path characters in the NetpwPathCanonicalize function, and the call location of this function is at 0x5FDDA15B in the NetpwPathCanonicalize function:



```
5FDDA14C  mov     esi, [ebp+arg_4]
5FDDA14F  push    edi                ; int
5FDDA150  push    [ebp+arg_8]        ; int
5FDDA153  mov     [esi], di
5FDDA156  push    esi                ; int
5FDDA157  push    [ebp+arg_0]        ; wchar_t *
5FDDA15A  push    ebx                ; wchar_t *
5FDDA15B  call    sub_5FDDA180
5FDDA160  cmp     eax, edi
5FDDA162  jnz     short loc_5FDDA16E
```

Enter the sub_5FDDA180 function, starting at 0x5FDDA1E0, we can see that the program uses the wcscat() function to splice the path, and the spliced path will be placed in the local variable var_418. Next, a loop operation (green bold arrow) is used to check whether the "/" character or "slash" character exists in the spliced string. If it exists, it will be converted to the backslash character or "\" character

After the conversion is completed, the program will push var_418, the converted path string, as the only argument to the stack, and call the sub_5FDDA26B function. It is this function that has an overflow problem.

\\$% (&^*&*) &) (* \A\.\.\.\B BBB...

Return Address

wcscpy (p3, p1)



p3



p2



p1

\\$% (&^*&*) &) (* \.\.\B BBB...



p3

Return Address

wcscpy (p3, p1)



p2



p1

\BBBBBBBBBBBBBBBBBBBBBBB...

Return Address

Fingerprinting the Malware -- Cryptographic Hash



Fingerprinting the Malware



- Fingerprinting involves generating the **cryptographic hash** values for the suspect binary based on its file content.
- Same cryptographic hashing algorithms:
 - MD5
 - SHA1
 - SHA256
- **Why not just use the file name?**
 - **Ineffective**, same malware sample can use different filenames, cryptographic hash is **calculated based on the file content**.
- File hash is frequently used as an indicator to share with other security researchers to help them identify the sample.

Tools and Python code

md5sum

sha256sum

sha1sum

```
1  import hashlib
2  import sys
3
4  filename = sys.argv[1]
5  content = open(filename, "rb").read()
6  print hashlib.md5(content).hexdigest()
7  print hashlib.sha256(content).hexdigest()
8  print hashlib.sha1(content).hexdigest()
9
```

■ Finding Strings ^[1]

- A string in a program is a sequence of characters such as “the.”
- A program contains strings if it prints a message, connects to a URL, or copies a file to a specific location.
- Searching through the strings can be **a simple way to get hints about the functionality of a program.**
 - For example, if the program accesses a URL, then you will see the URL accessed stored as a string in the program.
- You can use the **Strings** program to search an executable for strings, which are typically stored in either ASCII or Unicode format.

Static analysis (myhack.dll)

```
C:\Work>strings.exe myhack.dll_
```

```
modf
ldexp
_cabs
_hypot
fmod
frexp
_y0
_y1
_yn
_logb
_nextafter
index.html
http://www.naver.com/index.html
<myhack.dll> Injection!!! -- CSC 497/583 -- Si Chen
QI\
QI\
QI\
QI\
```

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    HANDLE hThread = NULL;

    g_hMod = (HMODULE)hinstDLL;

    switch( fdwReason )
    {
        case DLL_PROCESS_ATTACH :
            OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
            hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
            CloseHandle(hThread);
            break;
    }

    return TRUE;
}
```

Static analysis (myhack.dll)

```
4%5
7.787K7R7^7v7<7
7g8n8
9&9R9
9%:::g:r:e<
>&>+>6>A>U>
?I?
0G0^0i0q0!0
1A1^1
2"2+363
5<535c5
6"6j6
7<7
7.8
9:9I9I9s9
:#:A:h:}:
;>;;H;a;r;i;
<"<></<J<Q<
=U>
1o2M3t3
6k6
7^7>7
8>83808J8Z8
;3;s;
<!<'<+<1<5<?<R<L<v<
```

Sometimes the strings detected by the Strings program are not actual strings.

strings in Linux and flare-floss



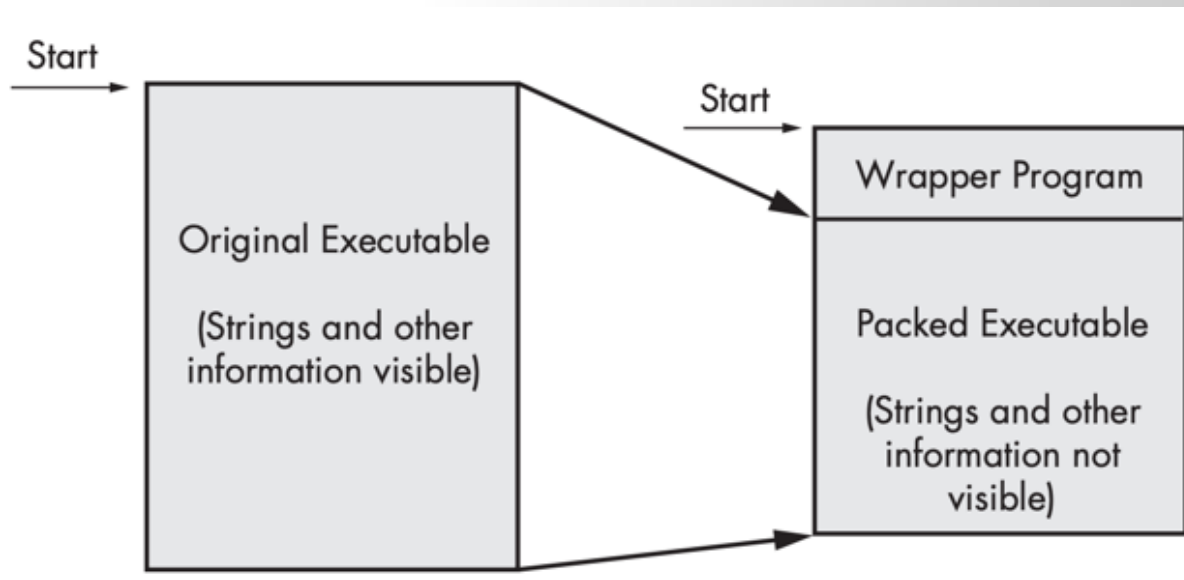
▪ FireEye Labs Obfuscated String Solver

- Many malware authors **evade heuristic detections** by obfuscating only key portions of an executable
 - These portions are strings and resources used to configure domains, files, and other artifacts of an infection
- The FireEye Labs Obfuscated String Solver (FLOSS) uses advanced static analysis techniques to **automatically deobfuscate strings** from malware binaries.

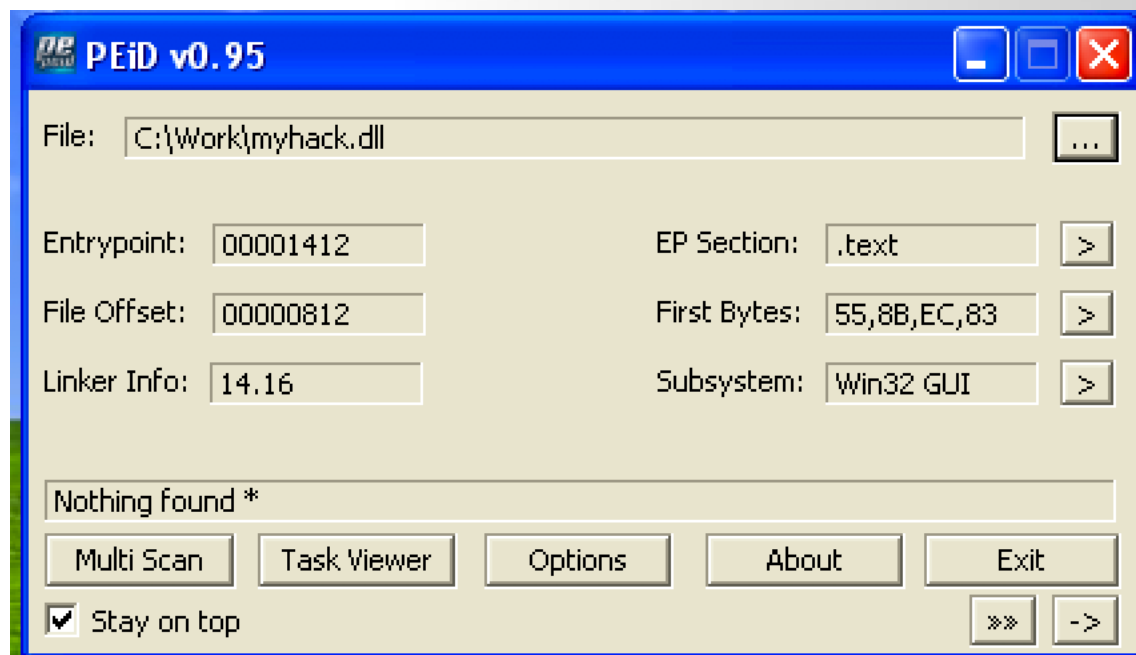
```
root@localhost:~# ./floss a99c01d5748b1bfd203fc1763e6612e8
FLOSS static ASCII strings
!This program cannot be run in DOS mode.
Rich
.text
.rdata
@.data
.rsrc
SPWV
uNSW
j0Xf
RPSW
90t0
j Xf
PPPPP
Y__^[
9csm
u)jAXf;
u+9u
8csm
uTVWhA7@
PPPPP
<v*v
^SSSSS
tAVWP
Y[ ^
PPPPP
8"u8
t      j\Yf
t$9U
QQSVWh
i@i ^V
```

Packed and Obfuscated Malware

- Malware writers often use **packing or obfuscation** to make their files more difficult to detect or analyze.
- **Obfuscated** programs are ones whose execution the malware author has attempted to hide.
- **Packed** programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed.
- Both techniques will severely limit your attempts to statically analyze the malware.



Packed and Obfuscated Malware



Packers and Cryptos

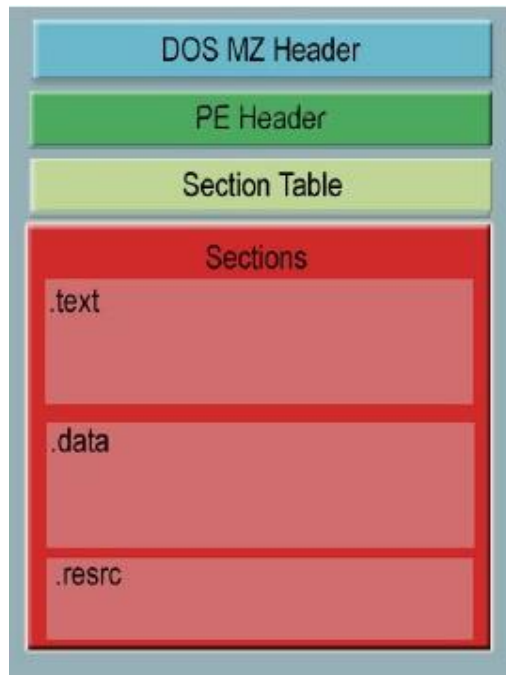
```
→ ~ upx -o myhack_packed.dll myhack.dll
      Ultimate Packer for eXecutables
      Copyright (C) 1996 - 2018
UPX 3.95      Markus Oberhumer, Laszlo Molnar & John Reiser      Aug 26th 2018

      File size      Ratio      Format      Name
      -----
      75264 ->      39424      52.38%      win32/pe      myhack_packed.dll

Packed 1 file.
```

Portable Executable (PE) file

- A Portable Executable (**PE**) **file** is the standard binary **file** format for an **Executable (.exe) or DLL** under Windows NT, Windows 95, and Win32.
- Derived from COFF (Common Object File Format) in UNIX platform, and it is not really “portable”.



Now here is the kicker. Even though this specification is spelled out by Microsoft, compilers/linkers chose to ignore some parts of it.

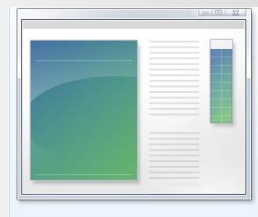
To make things even worse, the Microsoft loader doesn't enforce a good portion of this specification and instead makes assumptions if things start getting weird.

So even though the spec outlined here says a particular field is supposed to hold a certain value, the compiler/linker or **even a malicious actor could put whatever they want in there and the program will likely still run...**

Portable Executable (PE) file

- PE formatted files include:

- .exe, .scr (executable)
- .dll, .ocx, .cpl, drv (library)
- .sys, .vxd (driver files)
- .obj (objective file)



- All PE formatted files can be executed, except obj file.

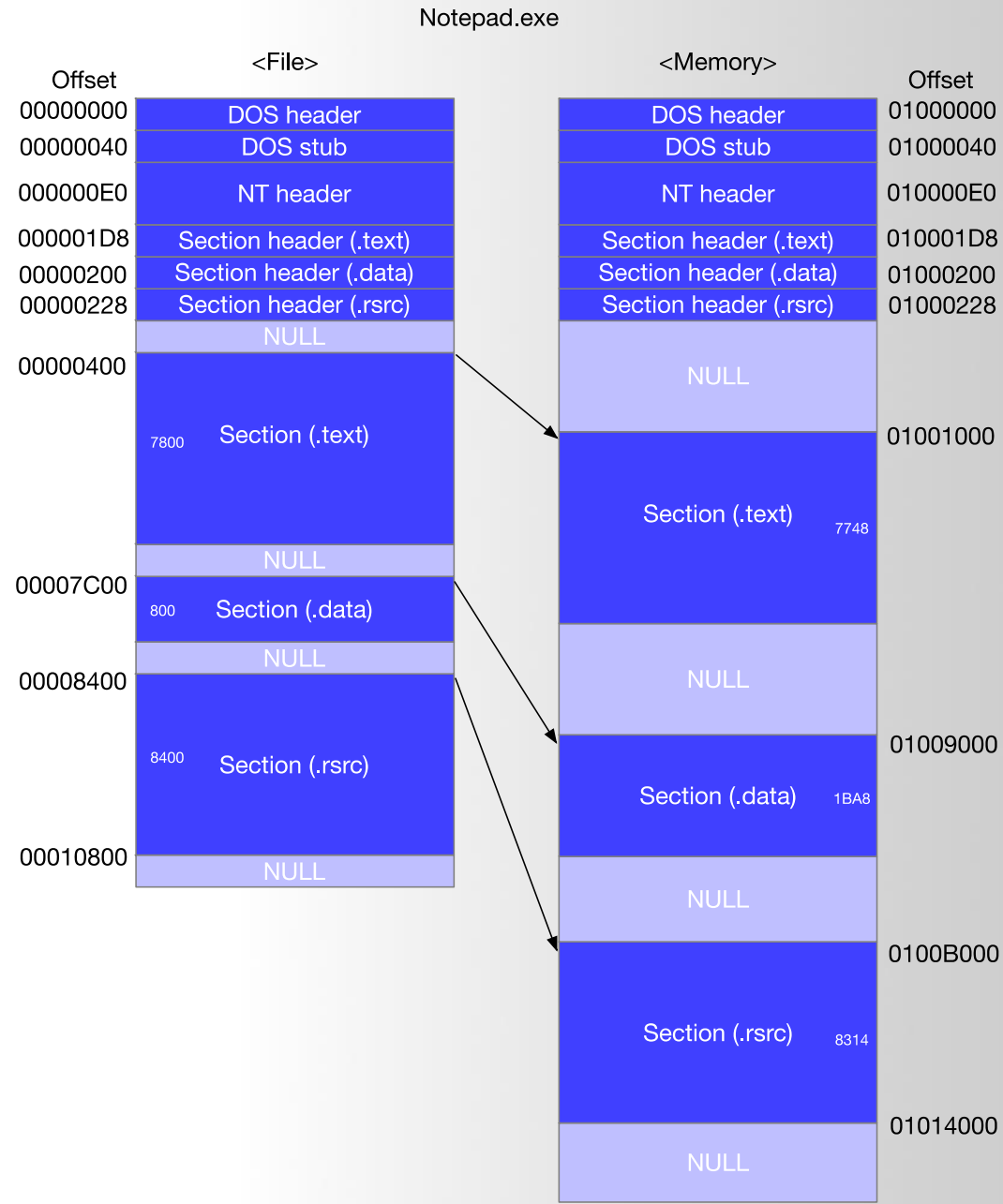
- .exe, .scr can be directly executed inside Shell (explorer.exe)
- others can be executed by other program/service

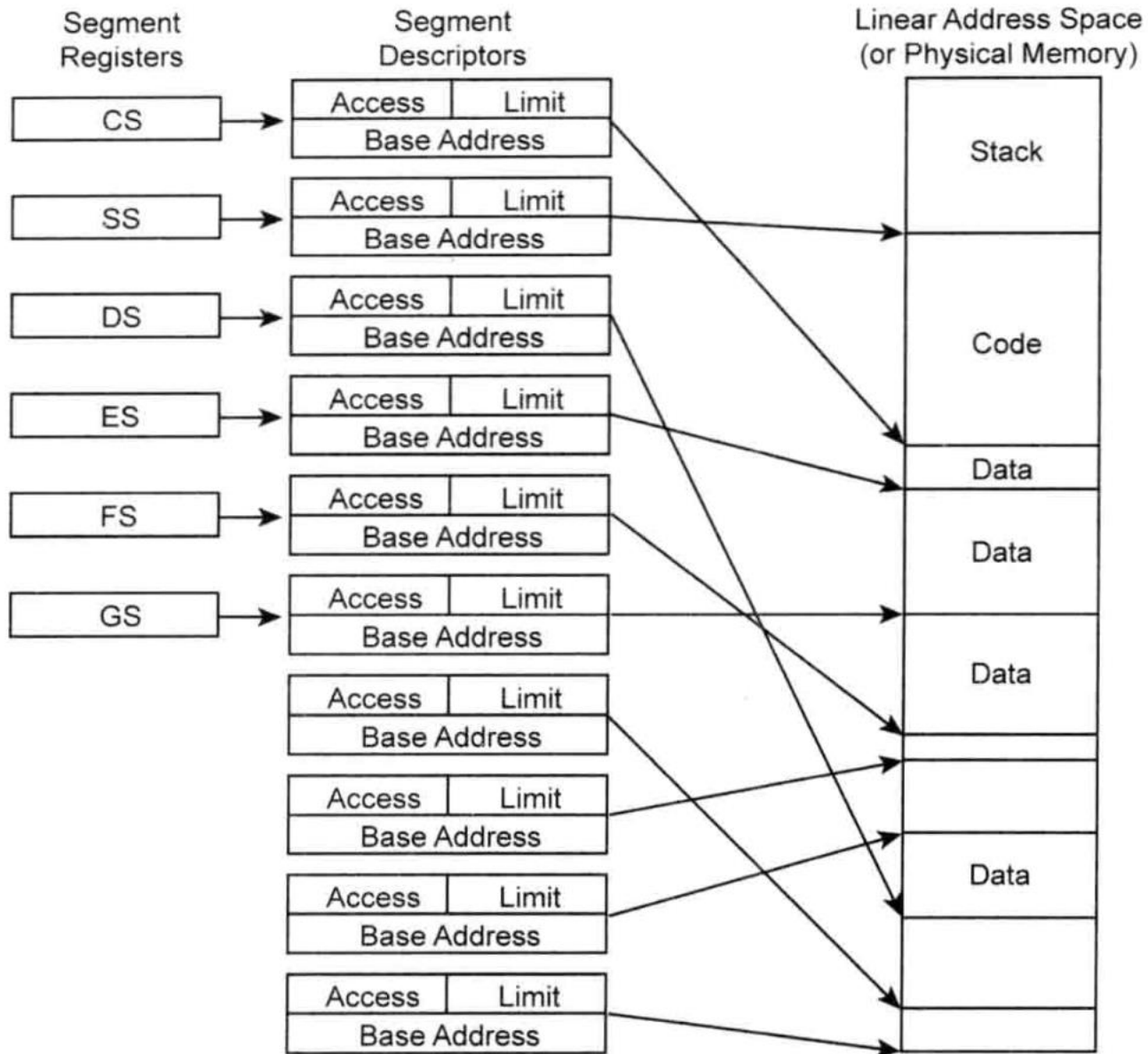
- **PE refers to 32 bit** executable file, or **PE32**. **64 bit** executable file is named as **PE+ or PE32+**. (Note that it is not PE64).

PE Example – Notepad.exe

00000000	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 00	MZÉ..... ..
00000010	B8 00 00 00 00 00 00 00	40 00 00 00 00 00 00 00	7.....@.....
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00	00 00 00 00 E8 00 00 00Φ...
00000040	0E 1F BA 0E 00 B4 09 CD	21 B8 01 4C CD 21 54 68!=!7.L=!Th
00000050	69 73 20 70 72 6F 67 72	61 6D 20 63 61 6E 6E 6F	is.program.canno
00000060	74 20 62 65 20 72 75 6E	20 69 6E 20 44 4F 53 20	t.be.run.in.DOS.
00000070	6D 6F 64 65 2E 0D 0D 0A	24 00 00 00 00 00 00 00	mode....\$......
00000080	A5 6D 16 9B E1 0C 78 C8	E1 0C 78 C8 E1 0C 78 C8	ñm.¢ß.xℒß.xℒß.xℒ
00000090	1B 2F 38 C8 E0 0C 78 C8	E1 0C 78 C8 E0 0C 78 C8	./8ℒα.xℒß.xℒα.xℒ
000000A0	1B 2F 61 C8 F2 0C 78 C8	E1 0C 79 C8 23 0C 78 C8	./aℒ≥.xℒß.yℒ#.xℒ
000000B0	76 2F 3D C8 E0 0C 78 C8	3B 2F 64 C8 F2 0C 78 C8	v/=ℒα.xℒ;/dℒ≥.xℒ
000000C0	1B 2F 45 C8 E0 0C 78 C8	52 69 63 68 E1 0C 78 C8	./Eℒα.xℒRichß.xℒ
000000D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000E0	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L...
000000F0	0D 84 7D 3B 00 00 00 00	00 00 00 00 E0 00 0F 01	.ä};.....α...
00000100	0B 01 07 00 00 6E 00 00	00 A6 00 00 00 00 00 00n... ^a
00000110	E0 6A 00 00 00 10 00 00	00 80 00 00 00 00 00 01	αj.....Ç.....
00000120	00 10 00 00 00 02 00 00	05 00 01 00 05 00 01 00
00000130	04 00 00 00 00 00 00 00	00 30 01 00 00 04 00 00θ.....
00000140	55 D8 01 00 02 00 00 80	00 00 04 00 00 10 01 00	U†.....Ç.....
00000150	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00
00000160	00 00 00 00 00 00 00 00	20 6D 00 00 C8 00 00 00m..ℒ...
00000170	00 A0 00 00 48 89 00 00	00 00 00 00 00 00 00 00	.á..Hë.....
00000180	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000190	40 13 00 00 1C 00 00 00	00 00 00 00 00 00 00 00	@.....

Load PE file (Notepad.exe) into Memory





VA & RVA

- VA (Virtual Address): The address is called a “VA” because **Windows creates a distinct VA space for each process, independent of physical memory**. For almost all purposes, a VA should be considered just an address. A VA is not as predictable as an RVA because the loader might not load the image at its preferred location.
- RVA (Relative Virtual Address): The address of an item after it is loaded into memory, with the base address of the image file subtracted from it. The RVA of an item almost always differs from its position within the file on disk (file pointer).

$$\text{RVA} + \text{ImageBase} = \text{VA}$$

In 32bit Windows OS, each process has 4GB virtual memory which means the range of VA is: **00000000 - FFFFFFFF**

DOS Header

```
struct DOS_Header
{
    // short is 2 bytes, long is 4 bytes
    char signature[2] = { 'M', 'Z' };
    short lastsize;
    short nblocks;
    short nreloc;
    short hdrsize;
    short minalloc;
    short maxalloc;
    void *ss; // 2 byte value
    void *sp; // 2 byte value
    short checksum;
    void *ip; // 2 byte value
    void *cs; // 2 byte value
    short relocpos;
    short noverlay;
    short reserved1[4];
    short oem_id;
    short oem_info;
    short reserved2[10];
    long e_lfanew; // Offset to the 'PE\0\0' signature relative to the beginning of the file
}
```

The first 2 letters are **always** the letters "**MZ**", the initials of Mark Zbikowski, who created the first linker for DOS. To some people, the first few bytes in a file that determine the type of file are called the "**magic number**,"


```
long e_lfanew;
```

long → 32 bit → ? Byte

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....yy..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00à...

E0 00 00 00

value for e_lfanew → ?

DOS Header

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....yy..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	EO	00	00	00à...

e_lfanew → 000000E0

DOS stub

00000040	OE 1F BA OE 00 B4 09 CD 21 B8 01 4C CD 21 54 68	[.]°..'.í!_.Lí!Th
00000050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
00000060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00000070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$.....
00000080	EC 85 5B A1 A8 E4 35 F2 A8 E4 35 F2 A8 E4 35 F2	i...[;`ä5ò`ä5ò`ä5ò
00000090	6B EB 3A F2 A9 E4 35 F2 6B EB 55 F2 A9 E4 35 F2	kë:ò@ä5òkëUò@ä5ò
000000A0	6B EB 68 F2 BB E4 35 F2 A8 E4 34 F2 63 E4 35 F2	këhò»ä5ò`ä4òcä5ò
000000B0	6B EB 6B F2 A9 E4 35 F2 6B EB 6A F2 BF E4 35 F2	këkò@ä5òkëjòçä5ò
000000C0	6B EB 6F F2 A9 E4 35 F2 52 69 63 68 A8 E4 35 F2	këoò@ä5òRich`ä5ò
000000D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

<https://virtualconsoles.com/online-emulators/dos/>

```
C:\>notepad.exe
This program cannot be run in DOS mode.
```

IMAGE_NT_HEADERS32 structure

12/04/2018 • 2 minutes to read

Represents the PE header format.

Syntax

C++

 Copy

```
typedef struct _IMAGE_NT_HEADERS {  
    DWORD      Signature;  
    IMAGE_FILE_HEADER      FileHeader;  
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;  
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Members

Signature

A 4-byte signature identifying the file as a PE image. The bytes are "PE\0\0".

FileHeader

An [IMAGE_FILE_HEADER](#) structure that specifies the file header.

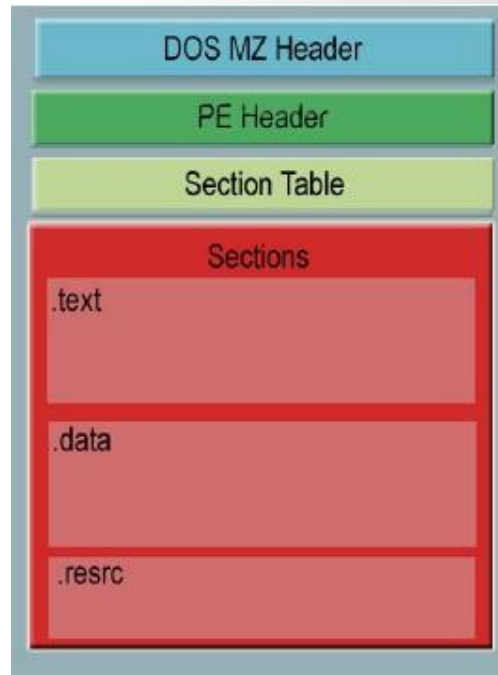
OptionalHeader

An [IMAGE_OPTIONAL_HEADER](#) structure that specifies the optional file header.

NT Header

NOTEPAD.EXE																	
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000000E0	50	45	00	00	4C	01	03	00	A3	C3	B0	4A	00	00	00	00	PE..L...ËÃ°J....
000000F0	00	00	00	00	E0	00	0F	01	0B	01	07	0A	00	78	00	00à.....x..
00000100	00	A6	00	00	00	00	00	00	9D	73	00	00	00	10	00	00s.....
00000110	00	90	00	00	00	00	00	01	00	10	00	00	00	02	00	00
00000120	05	00	01	00	05	00	01	00	04	00	00	00	00	00	00	00
00000130	00	40	01	00	00	04	00	00	33	30	01	00	02	00	00	80	.@.....30.....€
00000140	00	00	04	00	00	10	01	00	00	00	10	00	00	10	00	00
00000150	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00
00000160	04	76	00	00	C8	00	00	00	00	B0	00	00	58	89	00	00	.v..È....°..X%..
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000180	00	00	00	00	00	00	00	00	50	13	00	00	1C	00	00	00P.....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001A0	00	00	00	00	00	00	00	00	A8	18	00	00	40	00	00	00"....@...
000001B0	00	00	00	00	00	00	00	00	00	10	00	00	48	03	00	00H...
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001D0	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00text...

Section Header



Name	Privilege
.code	Executable, read
.data	Non-Executable, read/write
.resource	Non-Executable, read

IMAGE_SECTION_HEADER structure

12/04/2018 • 4 minutes to read

Represents the image section header format.

Syntax

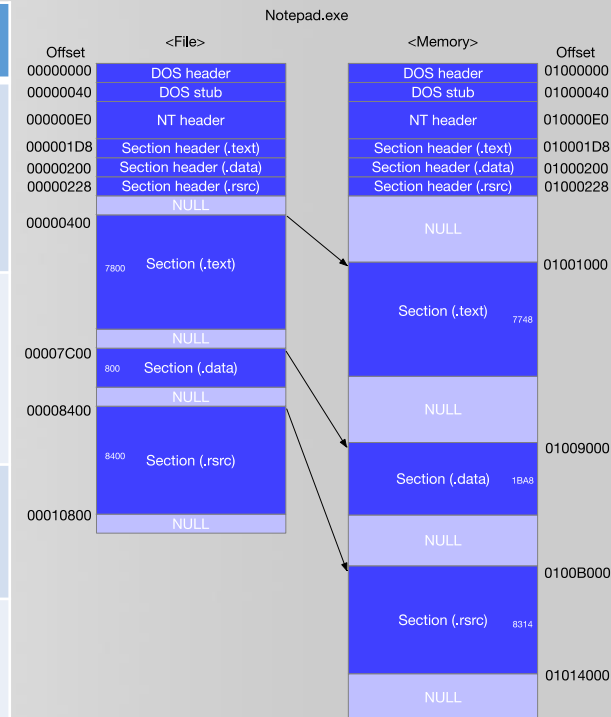
C++

 Copy

```
typedef struct _IMAGE_SECTION_HEADER {  
    BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];  
    union {  
        DWORD PhysicalAddress;  
        DWORD VirtualSize;  
    } Misc;  
    DWORD VirtualAddress;  
    DWORD SizeOfRawData;  
    DWORD PointerToRawData;  
    DWORD PointerToRelocations;  
    DWORD PointerToLinenumbers;  
    WORD  NumberOfRelocations;  
    WORD  NumberOfLinenumbers;  
    DWORD Characteristics;  
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Section Header

Members	Meaning
VirtualSize	The total size of the section when loaded into memory, in bytes.
VirtualAddress	The address of the first byte of the section when loaded into memory (RVA)
SizeOfRaw Data	The size of the section data on disk , in bytes.
PointerToRawData	The address of the first byte of the section on disk.
Characteristics	The characteristics of the image.



https://docs.microsoft.com/en-us/windows/desktop/api/winnt/ns-winnt-_image_section_header

Section Header

000001D0	00 00 00 00 00 00 00 00	2E 74 65 78 74 00 00 00text...
000001E0	48 77 00 00 00 10 00 00	00 78 00 00 00 04 00 00	Hw.....x.....
000001F0	00 00 00 00 00 00 00 00	00 00 00 00 20 00 00 60`
00000200	2E 64 61 74 61 00 00 00	18 1B 00 00 00 90 00 00	.data..."
00000210	00 08 00 00 00 7C 00 00	00 00 00 00 00 00 00 00
00000220	00 00 00 00 40 00 00 C0	2E 72 73 72 63 00 00 00@..À.rsrc...
00000230	58 89 00 00 00 B0 00 00	00 8A 00 00 00 84 00 00	X%...°...Š...//..
00000240	00 00 00 00 00 00 00 00	00 00 00 00 40 00 00 40@..@

Inspecting PE Header Information in Linux

```
1 import pefile
2 import sys
3
4 malware_file = sys.argv[1]
5 pe = pefile.PE(malware_file)
6 for section in pe.sections:
7     print "Name: %s VirtualSize: %s VirtualAddr: %s SizeofRawData: %s PointerToRawData: %s" %
8         (section.Name, hex(section.Misc_VirtualSize), hex(section.VirtualAddress), section.SizeOfRawData, section.PointerToRawData)
```

```
root@localhost ~# python display_sections.py a99c01d5748b1bfd203fc1763e6612e8
```

```
Name: .text VirtualSize: 0x7378 VirtualAddr: 0x1000 SizeofRawData: 29696 PointerToRawData: 1024
Name: .rdata VirtualSize: 0x261c VirtualAddr: 0x9000 SizeofRawData: 10240 PointerToRawData: 30720
Name: .data VirtualSize: 0x2cac VirtualAddr: 0xc000 SizeofRawData: 3584 PointerToRawData: 40960
Name: .rsrc VirtualSize: 0x1b4 VirtualAddr: 0xf000 SizeofRawData: 512 PointerToRawData: 44544
```

Inspecting PE Header Information

PEview - C:\WINDOWS\notepad.exe

File View Go Help

NOTEPAD.EXE

- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
 - Signature
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER
 - IMAGE_SECTION_HEADER**
 - IMAGE_SECTION_HEADER
 - IMAGE_SECTION_HEADER
- SECTION .text
 - IMPORT Address Table
 - IMAGE_DEBUG_DIRECTORY
 - IMAGE_LOAD_CONFIG_DIRECTORY
 - IMAGE_DEBUG_TYPES
 - IMPORT Directory Table
 - IMPORT Name Table
 - IMPORT Hints/Name Table
- SECTION .data
- SECTION .rsrc
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY

pFile	Data	Description	Value
000001D8	2E 74 65 78	Name	.text
000001DC	74 00 00 00		
000001E0	00007748	Virtual Size	
000001E4	00001000	RVA	
000001E8	00007800	Size of Raw Data	
000001EC	00000400	Pointer to Raw Data	
000001F0	00000000	Pointer to Relocations	
000001F4	00000000	Pointer to Line Numbers	
000001F8	0000	Number of Relocations	
000001FA	0000	Number of Line Numbers	
000001FC	60000020	Characteristics	
			IMAGE_SCN_CNT_CODE
			IMAGE_SCN_MEM_EXECUTE
			IMAGE_SCN_MEM_READ

Viewing IMAGE_SECTION_HEADER .text

Inspecting file imports with pefile library

```
1  import pefile
2  import sys
3
4  malware_file = sys.argv[1]
5  pe = pefile.PE(malware_file)
6  if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT'):
7      for entry in pe.DIRECTORY_ENTRY_IMPORT:
8          print "%s" % entry.dll
9          for imp in entry.imports:
10             if imp.name != None:
11                 print "\t%s" % (imp.name)
12             else:
13                 print "\tord(%s)" % (str(imp.ordinal))
14         print "\n"
```

Inspecting file export with pefile library

```
1  import pefile
2  import sys
3
4  malware_file = sys.argv[1]
5  pe = pefile.PE(malware_file)
6  if hasattr(pe, 'DIRECTORY_ENTRY_EXPORT'):
7      for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
8          print "%s" % exp.name
9
```

Inspecting PE Header Information in Linux

```
1 import pefile
2 import sys
3
4 malware_file = sys.argv[1]
5 pe = pefile.PE(malware_file)
6 for section in pe.sections:
7     print "Name: %s VirtualSize: %s VirtualAddr: %s SizeofRawData: %s PointerToRawData: %s" %
8         (section.Name, hex(section.Misc_VirtualSize), hex(section.VirtualAddress), section.SizeOfRawData, section.PointerToRawData)
```

```
root@localhost ~# python display_sections.py a99c01d5748b1bfd203fc1763e6612e8
```

```
Name: .text VirtualSize: 0x7378 VirtualAddr: 0x1000 SizeofRawData: 29696 PointerToRawData: 1024
Name: .rdata VirtualSize: 0x261c VirtualAddr: 0x9000 SizeofRawData: 10240 PointerToRawData: 30720
Name: .data VirtualSize: 0x2cac VirtualAddr: 0xc000 SizeofRawData: 3584 PointerToRawData: 40960
Name: .rsrc VirtualSize: 0x1b4 VirtualAddr: 0xf000 SizeofRawData: 512 PointerToRawData: 44544
```

Inspecting PE Header Information

PEview - C:\WINDOWS\notepad.exe

File View Go Help

NOTEPAD.EXE

- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
 - Signature
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER
 - IMAGE_SECTION_HEADER**
 - IMAGE_SECTION_HEADER
 - IMAGE_SECTION_HEADER
- SECTION .text
 - IMPORT Address Table
 - IMAGE_DEBUG_DIRECTORY
 - IMAGE_LOAD_CONFIG_DIRECTORY
 - IMAGE_DEBUG_TYPES
 - IMPORT Directory Table
 - IMPORT Name Table
 - IMPORT Hints/Name Table
- SECTION .data
- SECTION .rsrc
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY

pFile	Data	Description	Value
000001D8	2E 74 65 78	Name	.text
000001DC	74 00 00 00		
000001E0	00007748	Virtual Size	
000001E4	00001000	RVA	
000001E8	00007800	Size of Raw Data	
000001EC	00000400	Pointer to Raw Data	
000001F0	00000000	Pointer to Relocations	
000001F4	00000000	Pointer to Line Numbers	
000001F8	0000	Number of Relocations	
000001FA	0000	Number of Line Numbers	
000001FC	60000020	Characteristics	
			IMAGE_SCN_CNT_CODE
			IMAGE_SCN_MEM_EXECUTE
			IMAGE_SCN_MEM_READ

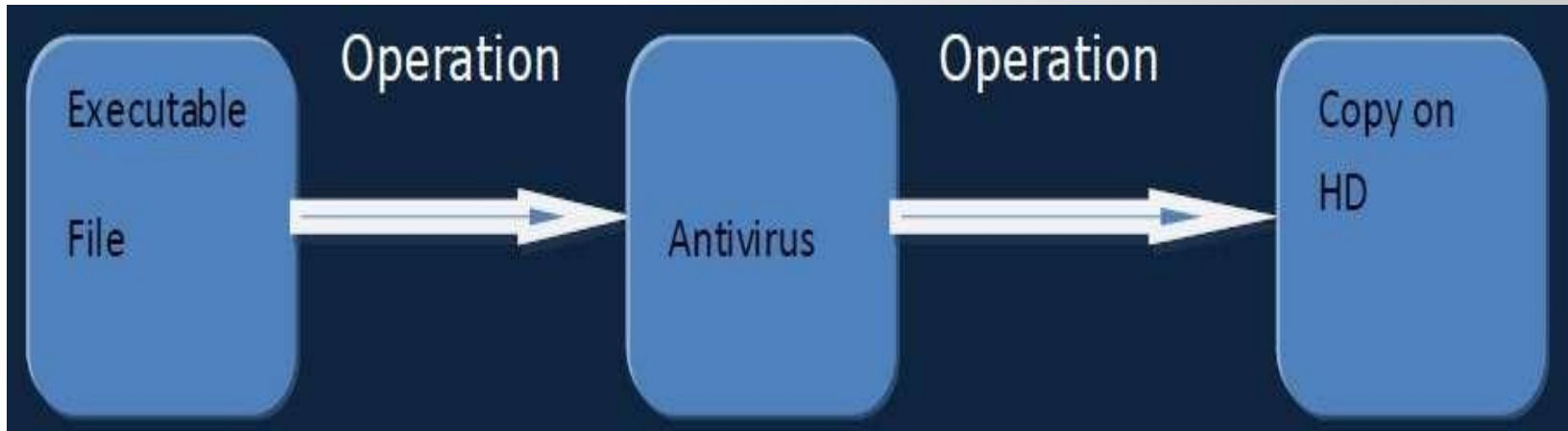
Viewing IMAGE_SECTION_HEADER .text

Anti-virus: How they actually work

- Nowadays AV scans our system on real-time basis.
- Information is analyzed based on the origin of the information
 - i.e. source of information.
- Operates differently depending upon source of information.



Anti-virus working from top level view.

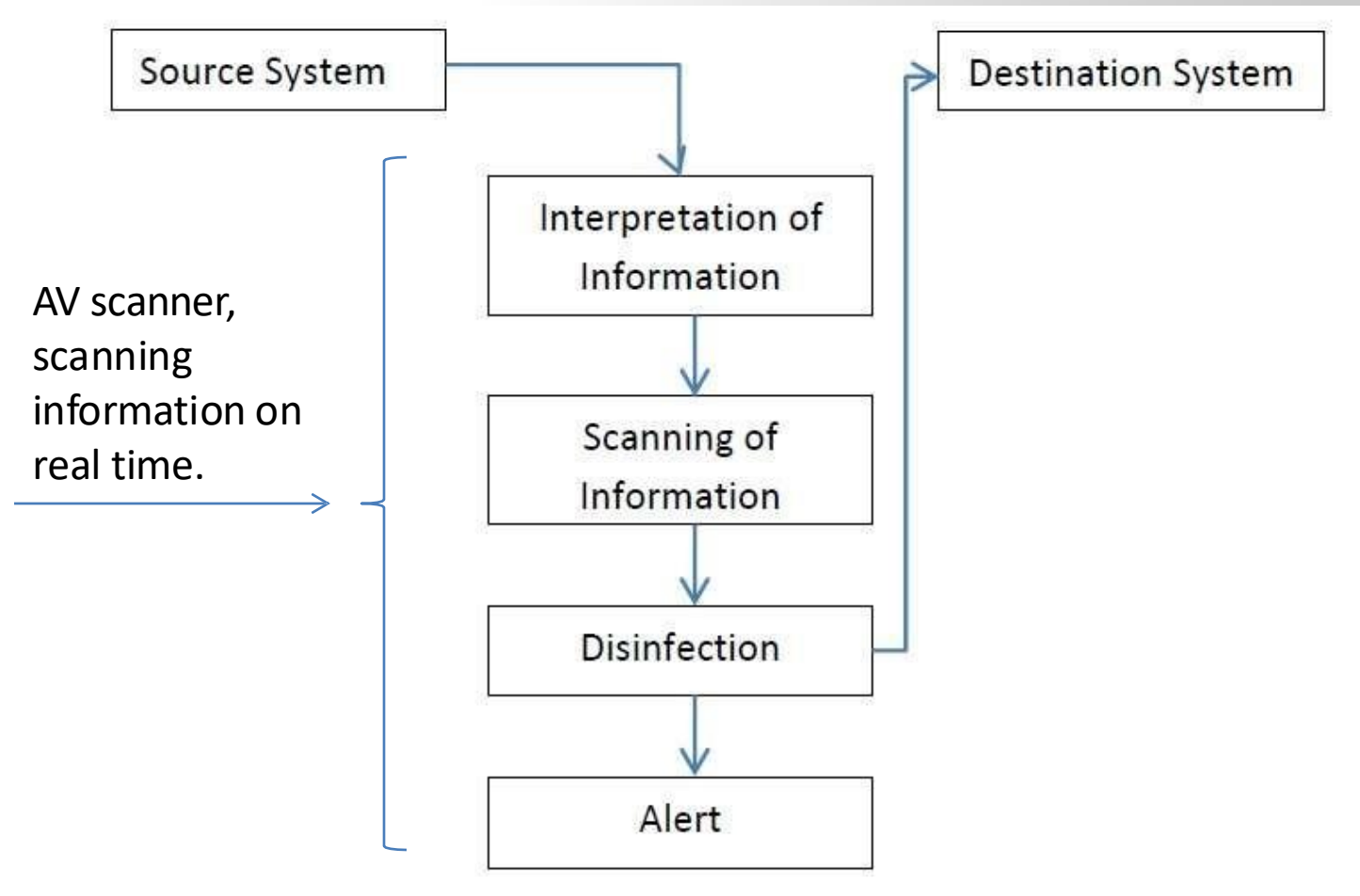


If the file is found malicious then the information will not be copied onto the destination location.
(Here destination in our case is HD)

One of the two possibilities takes place

- When the **data is found to be legitimate**, the scanner forwards that data to the destination location.
- When **virus is detected** then a warning is sent to UI for user's action. Interface may vary.

Traditional Antivirus Methods

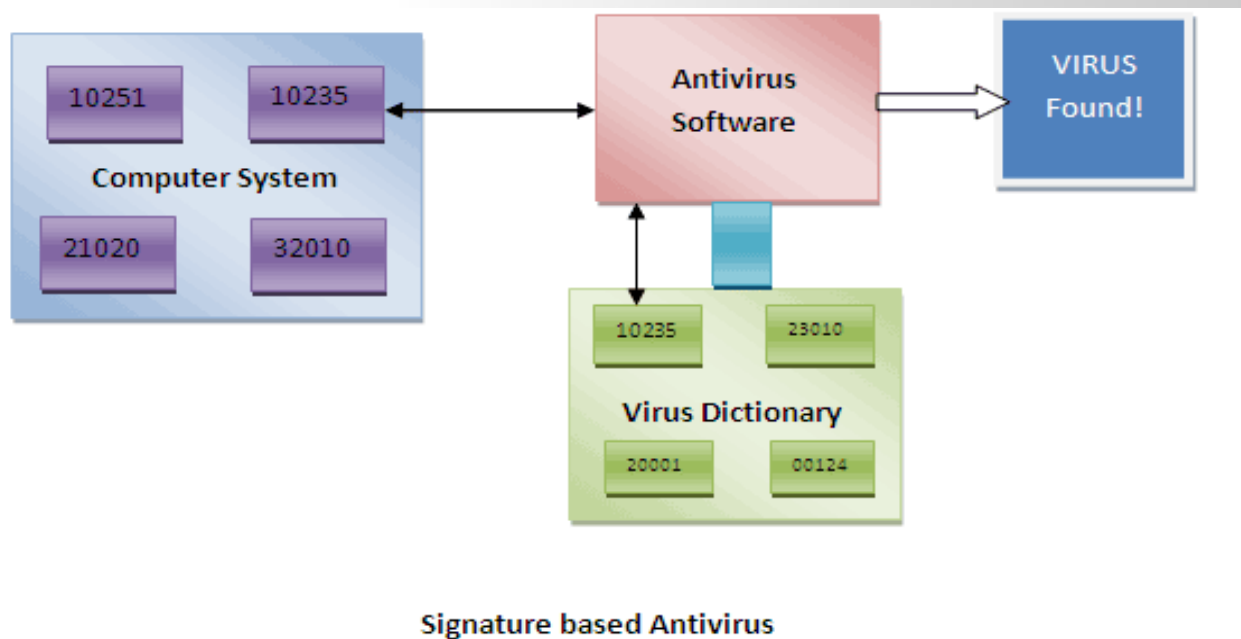


Hash-based blacklisting

- Simple and efficient method
- Requires maintaining a large virus signature database
- Always reactive, not proactive
- Sensitive to virus variations
- Fast update process for new samples

AV detection techniques(Scan - Engines)

- Signature Based detection (also sometimes called as “string based” detection)
- AV maintains a dictionary of the signatures of known Viruses, malwares, spywares etc.
- This dictionary is stored at client side and is usually in binary.
- Next-generation signature based detection
- Disadvantage?



00401090	00 00 00 00 00 00 00 00	4D 5A 90 00 03 00 00 00MZ.....
004010A0	04 00 00 00 FF FF 00 00	B8 00 00 00 00 00 00 00
004010B0	40 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	@.....
004010C0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
004010D0	00 00 00 00 F0 00 00 00	0E 1F BA 0E 00 B4 09 CD
004010E0	21 B8 01 4C CD 21 54 68	69 73 20 70 72 6F 67 72	!..L.!This progr
004010F0	61 6D 20 63 61 6E 6E 6F	74 20 62 65 20 72 75 6E	am cannot be run
00401100	20 69 6E 20 44 4F 53 20	6D 6F 64 65 2E 0D 0D 0A	in DOS mode....
00401110	24 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	\$.....
00401120	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00401130	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00401140	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00401150	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00401160	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00401170	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00401180	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L...

0041B498	40 45 43 48 4F 20 4F 46 46 0D 0A 3A 52 45 50 0D	@ECHO OFF...:REP.
0041B4A8	0A 44 45 4C 20 25 31 0D 0A 49 46 20 45 58 49 53	.DEL %1..IF EXIS
0041B4B8	54 20 25 31 20 47 4F 54 4F 20 52 45 50 0D 0A 44	T %1 GOTO REP..D
0041B4C8	45 4C 20 25 30 00 00 00 43 4D 44 20 2F 43 20 22	EL %0...CMD /C "
0041B4D8	22 25 73 22 20 22 25 73 22 22 00 00 25 73 5C 25	"%s" :"%s""..%s\%
0041B4E8	75 2E 63 6D 64 00 00 00 52 55 4E 44 4C 4C 33 32	u.cmd...RUNDLL32
0041B4F8	20 22 25 73 22 2C 25 73 00 00 00 00 25 73 5C 25	:"%s",%s....%s\%
0041B508	75 2E 74 6D 70 00 00 00 47 6C 6F 62 61 6C 5C 25	u.tmp...Global\%
0041B518	75 2D 25 75 00 00 00 00 FF FF FF FF DE B9 41 00	u-%u.....A.

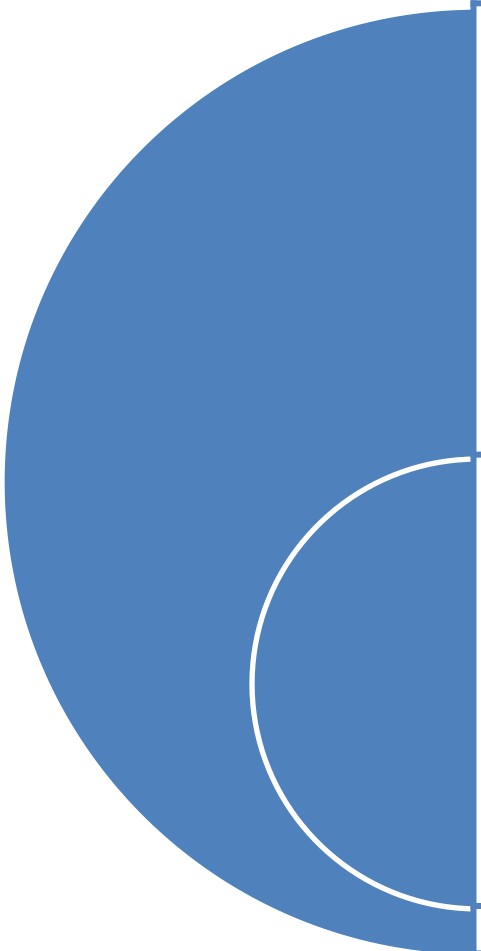
Signature-based detection

- Based on file offset and unique binary code
- Can detect new variants if the signature is well chosen
- One-to-many detection for the same virus family
- Requires experienced analysts
- Possibility of false positives/negatives
- Time-consuming update process

Heuristic based Detection

- Used to detect new, unknown viruses in your system that has not yet been identified.
- Based on the piece-by-piece examination of a virus.
- Looks for the sequence of instruction that differentiate the virus from 'normal programs'
- Disadvantage?

AV bypassing techniques



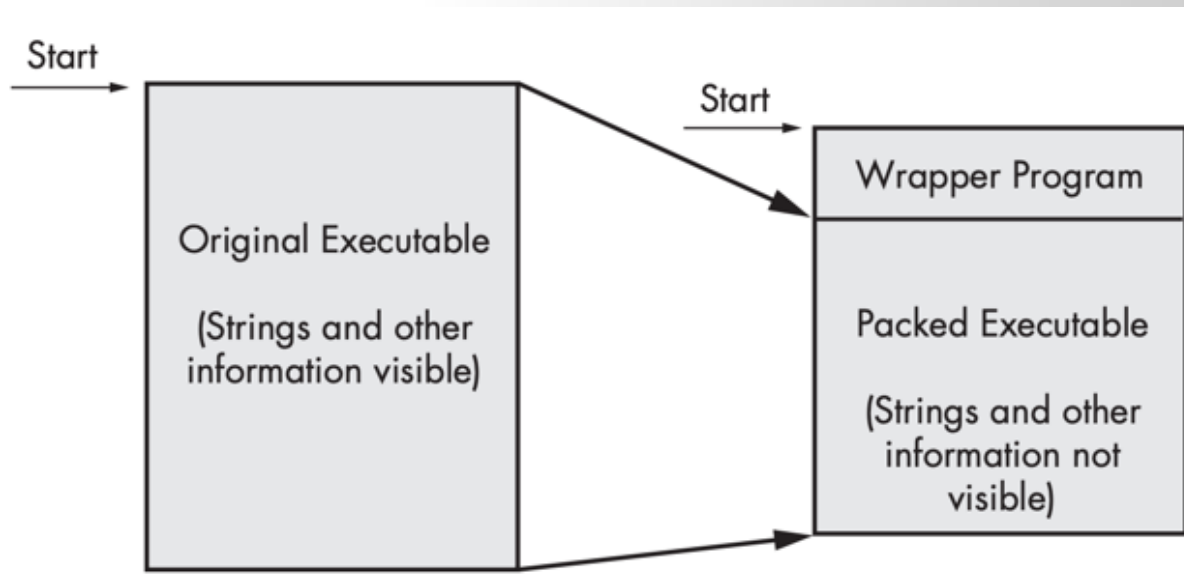
This are those techniques that the hackers and crackers already knew.

These are:

- Binders and packers
- Using splitter
- Code conversion from EXE to client side script
- Code obfuscation
- Using metasploit framework
- Code or DLL Injection

Packed and Obfuscated Malware

- Malware writers often use **packing or obfuscation** to make their files more difficult to detect or analyze.
- **Obfuscated** programs are ones whose execution the malware author has attempted to hide.
- **Packed** programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed.
- Both techniques will severely limit your attempts to statically analyze the malware.



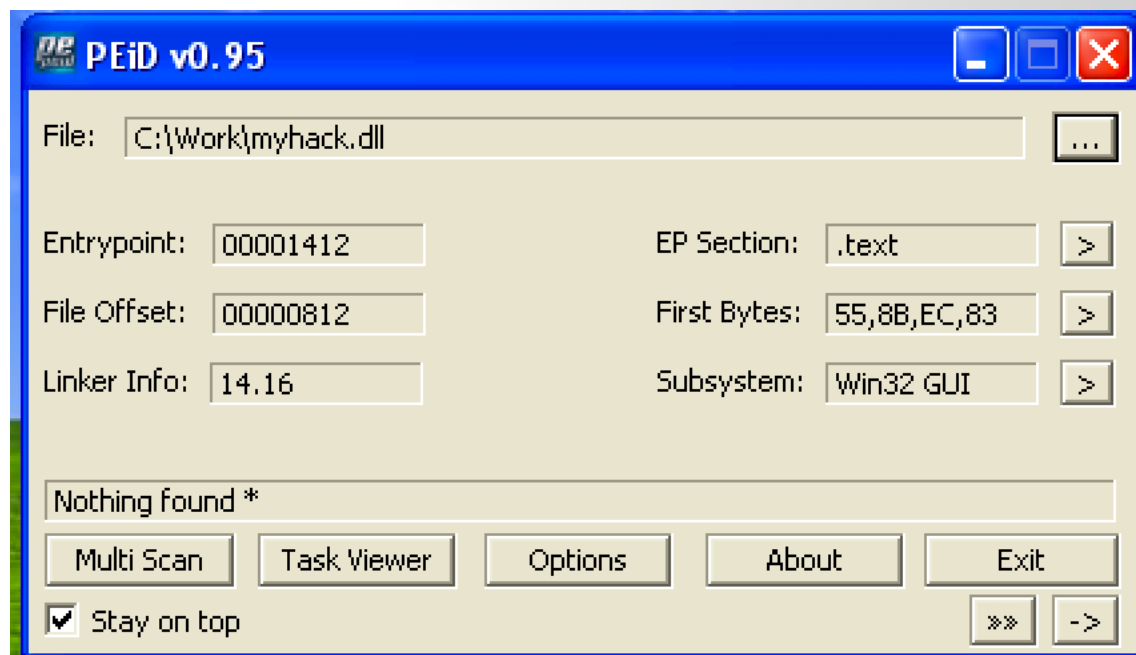
Packers and Cryptos

```
→ ~ upx -o myhack_packed.dll myhack.dll
      Ultimate Packer for eXecutables
      Copyright (C) 1996 - 2018
UPX 3.95      Markus Oberhumer, Laszlo Molnar & John Reiser      Aug 26th 2018

      File size      Ratio      Format      Name
      -----
      75264 ->      39424      52.38%      win32/pe      myhack_packed.dll

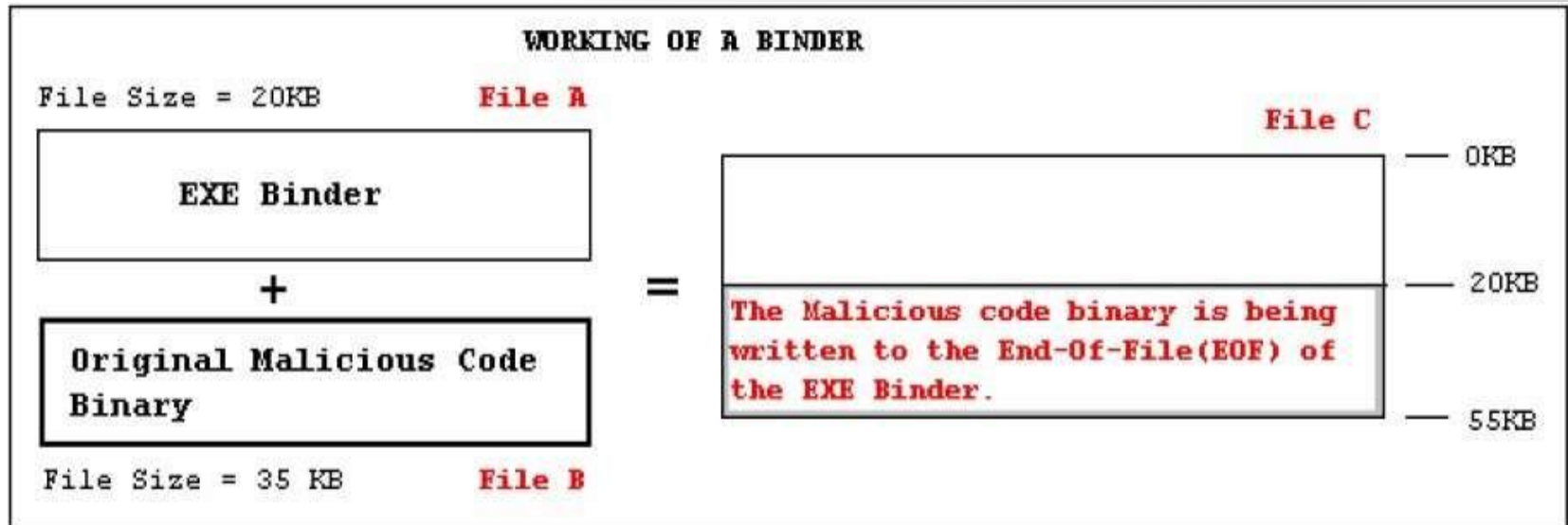
Packed 1 file.
```

Packed and Obfuscated Malware



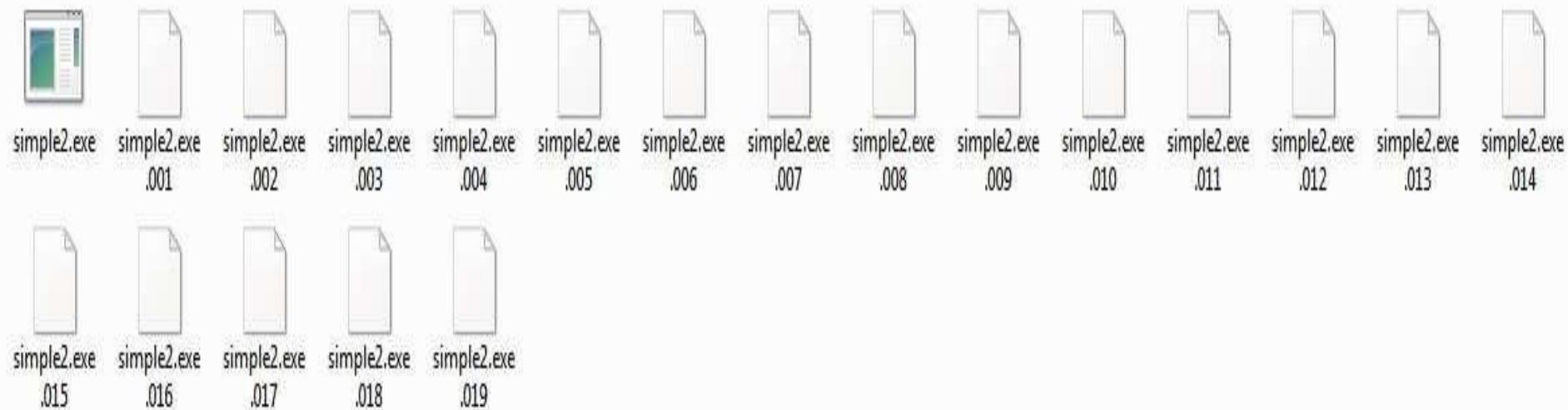
Binders and Packers

- Binders



Splitting the File and Code Obfuscation

- These are those programs that split a single files into no. of small sized files.



- One may change some code into some small chunked file to evade AV detection and again join it and scan it to check whether AV flags it malicious or not. A trial and Error method..

Behavioral based detection

- Just observes how the program executes, rather than merely emulating its execution.
- Identify malware by looking for suspicious behavior.
- Disadvantage?

Sandboxing Based detection

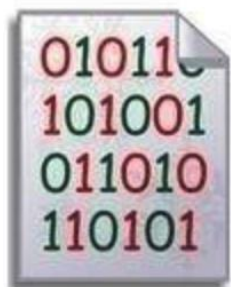
- What is “sandbox” ?
- Isolate the files which are to be scanned and monitors their activity.

Heuristic Engines

- Heuristic engines are basically statistical and rule based analyze mechanisms.
- Their main purpose is detecting new generation(previously unknown) viruses by categorizing and giving threat/risk grades to code fragments according to predefined criteria.
- Heuristic engines are the most advanced part of AV products they use significant amount of rules and criteria.
- Since no anti virus company releases blueprints or documentation about their heuristic engines all known selective criteria's about their threat/risk grading policy are **found with trial and error**.

Dynamic Heuristic Analysis

Unknown Sample



PE file

Sandbox



contains C:\, D:\, E:\
And windows,
System32 Folder and
system file

Log File



monitoring the behavior
of the unknow sample,
logging the function
call, parameters, etc...

Malware Fingerprint



malware expert use the
log file to find the key
features and add it to
the malware database

Some of the known rules about threat grading

- Decryption loop detected
- Reads active computer name
- Reads the cryptographic machine GUID
- Contacts random domain names
- Reads the windows installation date
- Drops executable files
- Found potential IP address in binary memory
- Modifies proxy settings
- Installs hooks/patches the running process
- Injects into explorer
- Injects into remote process
- Queries process information
- Sets the process error mode to suppress error box
- Unusual entropy
- Possibly checks for the presence of antivirus engine
- Monitors specific registry key for changes

Some of the known rules about threat grading

- Contains ability to elevate privileges
- Modifies software policy settings
- Reads the system/video BIOS version
- Endpoint in PE header is within an uncommon section
- Creates guarded memory regions
- Spawns a lot of processes
- Tries to sleep for a long time
- Unusual sections
- Reads windows product id
- Contains decryption loop
- Contains ability to start/interact device drivers
- Contains ability to block user input

	Pros	Cons
Static Heuristic Analysis	Fast, easy	Cannot handle shell, code obfuscation
Dynamic Heuristic Analysis	It can “reveal” the malware	May attacked by the anti-VM technology

Port 445: Overview, Use Cases, and Security Risks

1. What is Port 445?

1. TCP/UDP port used by the Server Message Block (SMB) protocol
2. Facilitates file, printer, and named pipe sharing in Windows networks

2. Port 445 Use Cases

1. File and printer sharing between Windows devices
2. Remote administration of network devices
3. Communication with Active Directory services

3. Security Risks

1. Vulnerable to unauthorized access if not properly secured
2. Exploitation of SMB vulnerabilities (e.g., WannaCry and NotPetya ransomware attacks)
3. Potential for information leakage if SMB traffic is not encrypted

4. Mitigating Security Risks

1. Use firewalls to restrict access to Port 445
2. Disable SMBv1 and use SMBv2 or SMBv3 with encryption
3. Keep systems updated with the latest security patches

Understanding IPC\$ in Windows Networking

1.What is IPC\$?

1. IPC\$ stands for Inter-Process Communication (IPC) Share
2. It is a hidden administrative share in Windows operating systems

2.IPC\$ Basics

1. Facilitates communication between processes on the same or different computers
2. Implemented using the Server Message Block (SMB) protocol

3.Role of IPC\$ in Windows Networking

1. Enables remote administration and management of resources
2. Provides a mechanism for authentication and authorization

4.Security Considerations

1. IPC\$ can potentially be exploited by attackers
2. Ensure proper security measures to mitigate risks

Q & A

