Penetration Testing Professional

Malware

Section 1: System Security – Module 6

IMPORTANT

In the members area you can download a .rar package (malware.rar) which contains all the samples we are going to use in this module.

Specifically, the malware.rar package will be available through the *Resources* drop-down menu, right next to this module's slides.

Malware is the short form of “**Mal**icious soft**ware**”.

It is basically a software written to cause damage or infiltrate computer systems without the owner’s informed consent.

It is a general term used to represent various forms of intrusive, hostile and/or annoying code.

6.1 Classification

6.2. Techniques used by Malware

6.3. How Malware Spreads?

6.4. Samples

CLASSIFICATION

Main malware categories

A Computer Virus is a computer program that copies itself and spreads without the permission or knowledge of the owner.

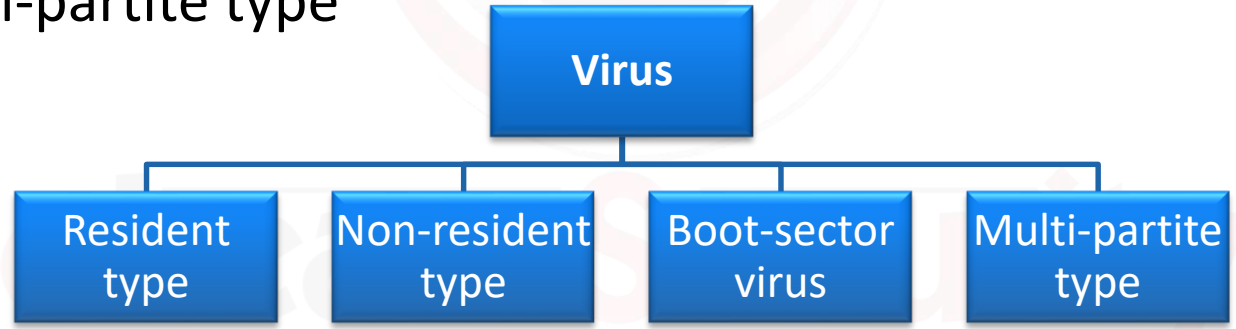
Viruses do not spread via exploiting vulnerabilities (the ones that do that are called worms).

The only way viruses are supposed to spread is with the host - at least in their rigorous classification.

Let us say, that a virus has infected a file; now if the owner moves the file to any system, the virus has thus a chance to spread and survive.

Viruses can be classified into the following sub-types:

- Resident type
- Non-resident type
- Boot-sector virus
- Multi-partite type



Resident type: which when executed becomes memory resident (and waits for some triggers such as loading of other program). It then infects other programs and so on.

Non-resident type: once a virus is executed, it will search for files it can infect. Then after infecting them, it will quit. When the infected program is run again, it will again find new targets and so on.

Boot-sector virus: which spreads via boot sectors. For example, if a user leaves a infected CD-ROM while turning off a system, the next time system will boot-up, the boot sector virus will activate and will thus spread to the hard-disk which will then spread it to another floppy disks. When floppies/pen drives are moved, the cycle gets repeated.

Multi-partite type: These viruses have several types of infection mechanisms such as they can have both Boot-sector and resident type virus or even more.

6.1.2 Trojan Horse

A **trojan horse** (or simply trojan) is a kind of malware that appears to the user to perform a function but in-fact facilitates unauthorized access to the owner's system.

The word actually comes from the Greek mythology.

Moreover, they are quite different from viruses.

- They are not self-replicating unlike viruses.

Example, someone sends you a screen-saver and it might actually be a trojan horse.

6.1.3 Rootkit

A **rootkit** is a malware which is designed to hide the fact that a compromise has already been done or to do the compromise at a deeper level.

A rootkit is basically used as a supplement to other malware.

6.1.3 Rootkit

Basically, rootkits can be used to hide processes, files on the file system, implement backdoors and/or create loopholes.

Rootkits exist for all major operating systems such as Windows, Linux, Solaris, OS X, etc.

They are basically installed as drivers (or kernel modules).

They are known to exist at the following levels (even lower levels are possibly):

- Application level
- Library level
- Kernel level
- Hypervisor level
- Firmware level

6.1.3 Rootkit

Application level	• They replace actually programs with copies of other programs
Library level	• Let us say that 10 applications are sharing a library, taking control of the library means taking control of all 10 apps
Kernel level	• This is the most common type and was first developed by Greg Hoglund around 1999 for Windows NT. They are known for their resistance to removal since they run at the same privilege level at which Anti-virus solutions run
Hypervisor level	• These days, processors have come up with support for virtualization. Rootkits which use such processor specific technologies are called hyper-visor rootkits. E.g., blue-pill and subvirt
Firmware level	• Rootkits for firmware such as BIOS,ACPI tables or device ROMS are known to exist. They have the highest chance of survival because currently, no tools exist to verify/scan up the firmware level rootkits

6.1.4 Bootkit

Bootkits are rootkits which grab the OS during the boot process itself and were introduced by Nitin Kumar and Vipin Kumar in 2007 (authors of this section).

They differ from the rootkits in the installation process and how and when they take control of the OS.

They start attacking the OS when the OS has not even started, so they are able to completely violate the security of the target operating system.

A **backdoor** is a software (or modification to the software) which helps in bypassing authentication mechanism, keeping remote access open (for later unauthorized purpose) which trying to remain hidden.

For example, a backdoor in a login system might give you access when a specified username/password is entered, even though they might not be a valid combination.

6.1.6 Adware

Adware is basically advertising supported software which displays ads from time-to-time during the use of the software.

Some adware also act as spyware. Adware also install other unwanted software on the users system which might/might not be malware without owner's consent.

A **spyware** is a software(kind of malware) which keeps on spying the user activities such as collecting user information (what he types), his website visiting record and other information without the consent of the computer owner.

The information is sent to the author after a certain amount has been collected.

They are also called privacy-violating software or privacy-invading software.

Normally, a system which has spyware also has other kinds of malware such as rootkits/trojans to hide the tracks and to keep in control of the machine.

6.1.8 Greyware

This is a collective name of spyware and adware. A **greyware** can either be a spyware or adware or both. Thus, all spyware and adware software are collectively referred as greyware

A **dialer** is a software which is used to connect to the internet but instead of connecting to normal numbers, they connect to premium numbers which are charged highly.

Thus, the owner of the dialer who has setup the stuff makes big sums of money.

6.1.10 Key-logger

Key-loggers are malware which log down key pressed by the key-owner without the consent of the owner. Thus, the person is unaware that his actions are being monitored.

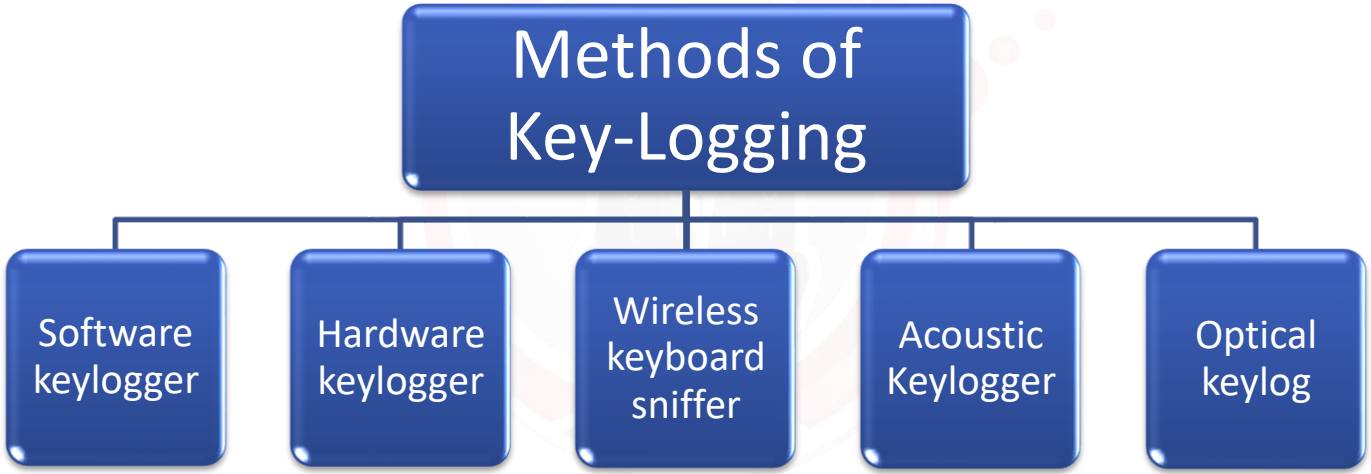
For example, a person might type his credit-card numbers which might then be misused by the creator of keylogger.

6.1.10 Key-logger

There are numerous kinds and methods of keylogging such as:

- Software keylogger
- Hardware keylogger
- Wireless keyboard sniffer
- Acoustic Keylogger
- Optical keylogger

6.1.10 Key-logger



6.1.10 Key-logger

Software keylogger: kernel mode or user mode keyloggers

Hardware keylogger: firmware based keylogger can be put in the BIOS.

PS/2 and USB keyboards can be sniffed with an additional device placed between the keyboard port and CPU.

6.1.10 Key-logger

Wireless keyboard sniffer: Passive sniffers can be used to collect keyboard data in case of wireless keyboards.

Acoustic Keylogger: These kinds of keylogger are based on the sound made when a key is struck by the user.

After sometime of data logging, clear patterns can be distinguished when a key is pressed or release which leads to remote passive keylogging.

Optical keylogger: Optical keylogging can be done by a person standing beside you.

This technique is also called shoulder surfing and is normally used to steal ATM PINs or passwords or other small but critical pieces of information.

Botnet refers to a collection of compromised computers which run commands automatically and autonomously (with the help of command and control server).

Botnets are typically created when a number of clients install the same malware.

This is usually done via drive-by-downloads (drive-by-download means a compromised website will try to exploit your web browser and install a software without user consent).

The controller or owner of the botnet is called a bot master and is usually the one who gives commands to the bots.

Botnets are used by the botmaster for reasons such as distributed denial of service, sending SPAM, etc.

6.1.12 Ransomware

This is a software which locks down important files with a password and then demands from the user to send money and in return promises to unlock files.

They are also called extortive malware since they demand extortion money (ransom) for restoration of user data.

The most famous example being gpcode which used public-key cryptography to encrypt the user files.

6.1.13 Data-Stealing malware

Data stealing malware basically steals data such as private encryption keys, credit-card data, competitors data such as internal secret algorithms, new product designs and other internal data which could be used by 3rd party to cause damage to the original data owner.

Some of these are highly targeted attacks and are never detected.

Worms are basically software which use network/system vulnerabilities to spread themselves from system to system.

They are typically part of other software such as rootkit and are normally the entry point into the system.

They basically compromise the system (locally or remotely) and then provide access to other software such as bot clients, spyware, key-loggers and so on.

Basically, the point to note that is that there is no clear line which distinguishes one form or malware from other.

Normally malwares are found in pairs with multiple variants simultaneously active on the target system.

This is done because once a system has been compromised, the new owner does not want to lose control of the machine.

Techniques used by Malware

6.2 Techniques used by Malware

Basically, in this section we are going to discuss almost all the techniques used by malware.

This includes hidden techniques.

Since, we are not teaching how to write malware, actual source code will be missing but yes, where required for a better understanding, we will propose snippets or pseudo-code.

6.2 Techniques used by Malware

This chapter does not cover the techniques used to detect malware.

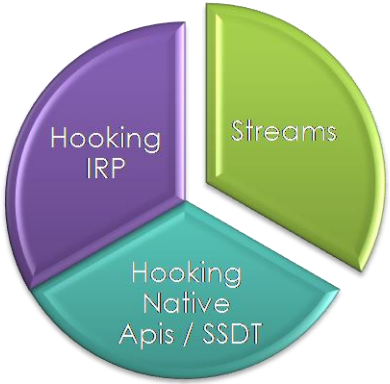
We are documenting the techniques used by them.

This should provide enough knowledge to understand the threat level and provide solutions accordingly.

This sort of techniques are basically used by malware to hide themselves from either the Anti-Virus, the user or both.

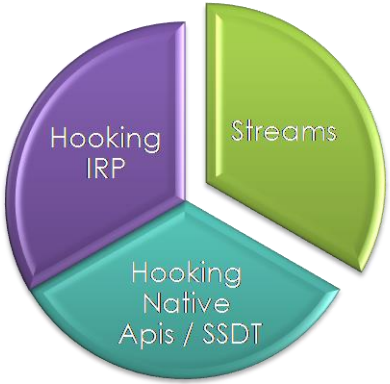
The most important covert methods are:

- Streams
- Hooking native Apis / SSDT
- Hooking IRP



Now we will cover the methods in some details. These topics are highly moving targets with new methods/techniques invented every few months.

Streams are a feature of NTFS file system, they are not available on FAT file systems.



Microsoft calls them Alternate Data Stream.

The original data stream is file data itself(it is the data stream with no name), all other streams have a name. Alternate data streams can be used to store file meta data/or any other data.

To explain the concept, let us give you a demo. The demo has been tested on Windows XP SP3 (but should work flawlessly on other Windows NT based OS too).

Type the following command in the command prompt:

```
echo This data is hidden in the stream. Can you Read IT ???
>> sample.txt:hstream
```

Now you can check the file named sample.txt.

You will be surprised to see that file size is reported as 0 bytes. So how do you retrieve back your data:

```
more < sample.txt:hstream
```

Now, let us explain how to use the streams programmatically.

In the `CreateFile` API in windows, just append "`:stream_name`" to the file name, where `stream_name` is the name of the stream.

6.2.1 Streams

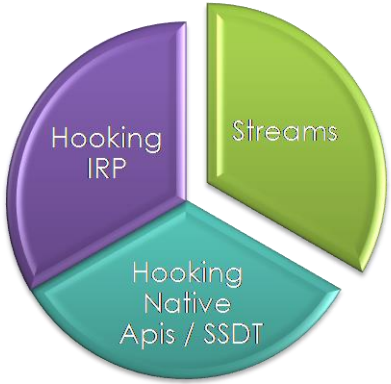
```
#include <windows.h>
#include <stdio.h>

void main( )
{
…
   hStream = CreateFile( "sample.txt:my_stream",
                              GENERIC_WRITE,
                          FILE_SHARE_WRITE,
                                      NULL,
                                OPEN_ALWAYS,
                                         0,
                                      NULL );
   if( hStream == INVALID_HANDLE_VALUE )
      printf( "Cannot open sample.txt:my_stream\n" );
   else
      WriteFile (hStream,"This data is hidden in the stream. Can you Read IT ???", 53, &dwRet,
NULL);
}
```

Snippet 1

6.2.2 Hooking native APIs / SSDT

SSDT stands for System Service Descriptor Table. Native API is API which resides in ntdll.dll and is basically used to communicate with kernel mode.

This communication happens using SSDT table.



For each entry in SSDT table, there is a suitable function in kernel mode which completes the task specified by the API; the representation can be pictured as:

6.2.2 Hooking native APIs / SSDT

SSDT table resides in the kernel and is exported as **KeServiceDescriptorTable**. The following are the services available for reading/writing files:

- **NtOpenFile**
- **NtCreateFile**
- **NtReadFile**
- **NtWriteFile**
- **NtQueryDirectoryFile** (This is used to query contents of the directory).

Microsoft keeps on adding new services on every OS release.

Let us consider the case of a directory query.

For that, we have to hook **NtQueryDirectoryFile**.

Hooking means that we want our (malicious) function to be called instead of the actual function.

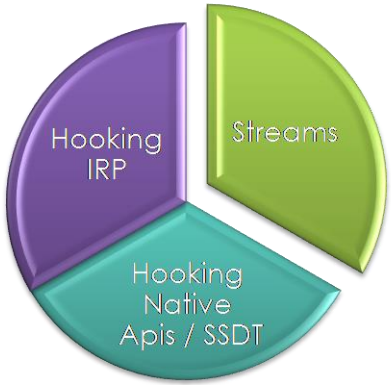
6.2.2 Hooking native APIs / SSDT

1. Hook SSDT table entry corresponding to NtQueryDirectoryFile
2. Now, whenever the above function is called, your function will be called
3. Right after your function gets called, call original function and get its result (directory listing)
4. If the result was successful, modify the results (hide the file/sub-directory you want to hide)
5. Now pass back the results to the caller
6. You are hidden

However, this is a very basic method.

Nowadays almost all anti-virus/rootkit-detectors scan SSDT table for modifications (they can compare it with the copy stored in the kernel) and thus detection can be done.

Windows architecture in kernel mode introduced the concepts of IRPs (I/O Request Packets) to transmit piece of data from one component to another.



The concept of IRPs is well explained in the Windows Driver Development Kit (it is available for free).

Almost everything in the windows kernel use IRPs for example network interface (TCP/UDP, etc.), file system, keyboard and mouse, and almost all existent drivers.

Here is a little snippet from Microsoft winddk showing how critical IRPs are:

```
DriverObject->MajorFunction[IRP_MJ_CREATE]= DiskPerfCreate;
DriverObject->MajorFunction[IRP_MJ_READ]= DiskPerfReadWrite;
DriverObject->MajorFunction[IRP_MJ_WRITE]= DiskPerfReadWrite;
DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL]= DiskPerfWmi;
DriverObject->MajorFunction[IRP_MJ_SHUTDOWN]= DiskPerfShutdownFlush;
DriverObject->MajorFunction[IRP_MJ_FLUSH_BUFFERS]=                    DiskPerfShutdownFlush;
DriverObject->MajorFunction[IRP_MJ_PNP]= DiskPerfDispatchPnp;
DriverObject->MajorFunction[IRP_MJ_POWER]= DiskPerfDispatchPower;#
```

There are basically 2 ways to play with IRPs:

- **Become A Filter Driver**
- **Hooking The Function Pointer**

Become A Filter Driver:- Register with the operating system as a filter driver or an attached device.

Hooking The Function Pointer: in the previous snippet, the array is just a table with function pointers and can be easily modified

Code snippet showing function pointer hooking:

```
old_power_irp = DriverObject->MajorFunction[IRP_MJ_POWER];
DriverObject->MajorFunction[ IRP_MJ_POWER] = my_new_irp;
```

As you can see, function pointer is one of the easiest method to hook functions.

The basic IRP design is so that after an IRP has been created, it is passed to all the devices registered at lower levels.

The design has pre-processing mode and post-processing mode.

Pre-processing is done when an IRP arrives and post-processing is done when the IRP has been processed by all the levels below current level.

Each device object has its own function table (as shown above in example).Hooking the function pointers of such objects is called DKOM (**D**irect **K**ernel **O**bject **M**anipulation).

6.2.3 Hooking IRP

All file-systems, network layers, devices like keyboard, mouse, etc. have such objects.

For example:

- \device\ip
- \device\tcp
- \Device\KeyboardClass0
- *FileSystem*\ntfs

Filter drivers are basically used by Anti-viruses to get control whenever a new file is written.

6.2.4 Hiding a Process

Hiding a process requires a more difficult approach.

It requires a combination of different techniques.

E.g., first thing you have to do is to hook NtOpenProcess native API (probably using SSDT table hooks).

Please refer to earlier segments of this chapter to know more about SSDT hooking.

6.2.4 Hiding a Process

Other things to do is to hide the process from EPROCESS list.

This list is maintained by the OS for all the active processes.

6.2.4 Hiding a Process

The EPROCESS list has the following structure:

```
kd> dt _EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x06c ProcessLock      : _EX_PUSH_LOCK
   +0x070 CreateTime       : _LARGE_INTEGER
   +0x078 ExitTime         : _LARGE_INTEGER
   +0x080 RundownProtect   : _EX_RUNDOWN_REF
   +0x084 UniqueProcessId  : Ptr32 Void
   +0x088 ActiveProcessLinks : _LIST_ENTRY
   +0x090 QuotaUsage       : [3] Uint4B
...
   +0x0c4 ObjectTable      : Ptr32 _HANDLE_TABLE
   +0x0c8 Token            : _EX_FAST_REF
...
   +0x174 ImageFileName    : [16] Uchar
```

Note: The important members are made bold that are normally most used.

As you can see in the above structure, **`ActiveProcessLinks`** is the circular doubly linked list with `*FLINK` and `*BLINK` as pointers to other structures.

The easiest thing to do is to unlink the structure relative to our process from the list (Refer to Module 3 for more information on this list).

If the driver is loaded, you will also have to unlink it from the `PsLoadedModuleList`.

API hooking is essentially the act of intercepting an API function call, and modifying its functionality somehow, either by redirecting it to a function of our choice, stopping the function from being called, or logging the request - the possibilities are infinite.

6.2.5 API Hooking

There can be different types of hooking such as:

- IAT Hooking
- EAT Hooking
- Inline Hooking

The techniques depend on where the hooks are actually applied.

6.2.6.1 IAT Hooking

IAT stands for Import Address Table. It is basically used to resolve runtime dependencies.

For example, when you use `MessageBoxA` API in windows, your compiler automatically links to `user32.dll`.

This makes your program dependent on `user32.dll`.

6.2.5.1 IAT Hooking

IAT hooking involves modifying the IAT table of the executable and replace the function with our own copy.

EAT stands for Export Address Table. This table is maintained in DLLs (dynamic link library).

These files just contain support functions for other executable files.

6.2.5.2 EAT Hooking

The difference between **IAT** and **EAT** hooking is:

- Since EATs exist only in DLL files (under normal settings) most of the times EAT hooking is utilized only on DLLs while IAT hooking can be done on both EXEs and DLLs.

6.2.5.3 Inline Hooking

Inline hooking is the most difficult to do due to the way it works.

In this form of hooking, we modify the first few bytes of the target function code and replace them with our code which tells the IP (instruction pointer) to execute code somewhere else in memory.

Whenever the function gets executed, we will get control of execution; after doing our job, we have to call the original function so we have to fix up the modified function.

This is normally done by executing a number of instructions which were replaced and then resuming execution in non-modified original function code.

6.2.6 Anti-Debugging Methods

There are several methods which are used by malware to increase the time required to analyze the code (by security analysts).

If such techniques are not already known by security analysts then the time required increases drastically.

For example:

We will document a trick that lets us detect the presence of a running debugger. It is called INT 2D trick and works flawlessly on Windows OS.

The trick is coded in assembly language.

6.2.6 Anti-Debugging Methods


```
push debugger_not_detected
push    fs:[0]                      set SEH
mov     fs:[0],esp
int     2dh                         If debugger is attached it will run
                                    normally else we have got an exception
nop                                 The above instruction causes it to skip
                                    this instruction if debugger is attached.
pop     fs:[0]                      clear SEH
add     esp, 4
...
debugger_detected :
...
debugger_not_detected:
```

6.2.6 Anti-Debugging Methods

What we do is:

1. Set an exception handler.

2. Cause an exception with `INT 2dh`

3. If a debugger is attached and does not pass the exception to us we get to `debug_detected` because an exception occurred for sure (we caused it)

6.2.7 Anti-Virtual Machine

Normal users always run the programs on a real system but security analysts analyzing malwares do not run the malware on a real system.

They always run the code in virtualized OS.

The tools are VMware, Virtual Pc, Xen, bochs, qemu to name a few.

6.2.7 Anti-Virtual Machine

These software let you install a virtual OS side-by side your OS (without disturbing your OS) and will run just like any other normal program.

These techniques are basically used by security analysts, so malware authors have found out few bugs in these applications which can be used to detect whether the OS is virtualized or not.

6.2.7 Anti-Virtual Machine

One of the tricks is given in next slide (in the screen shot).

The techniques basically work on the SIDT instruction, which returns the IDT table address.

On real machines, it is in low memory less than 0xd0 while for virtualized OS (VMware/Virtual Pc), it is higher than that.

This abnormal behavior leads to detection whether the malware is running on a real or virtualized system or not.

6.2.7 Anti-Virtual Machine


```
text:10007116 ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
text:10007116
text:10007116
text:10007116 Anti_Emulation_SIDT_Based_Check proc near ; CODE XREF: DllMain(x,x,x)+161p
text:10007116                 call    Get_IDT_base
text:10007116
text:1000711B                 and     eax, 0FF000000h
text:10007120                 xor     ecx, ecx
text:10007122                 cmp     eax, 80000000h  ; Real Windows Machine always have 0x80 for their MSB
text:10007127                 setnz   cl
text:1000712A                 mov     eax, ecx        ; If EAX != 0 we are emulating windows
text:1000712C                 retn
text:1000712C
text:1000712C Anti_Emulation_SIDT_Based_Check endp
text:1000712C
```

6.2.8 Obfuscation

Code obfuscation techniques transform/change a program in order to make it more difficult to analyze while preserving functionality.

Code obfuscation is used both by malware and legal software to protect itself.

The difference is that malware use it to either prevent detection or make reverse engineering more difficult.

6.2.8 Obfuscation

Basically code obfuscation/data obfuscation makes programs more difficult to reverse engineer.

One major drawback of existing obfuscating techniques is the lack of theoretical basis about their efficiency (Several implementations which looked impressive had very basic weak points, leading to their total downfall).

6.2.8 Obfuscation

The malware obfuscates itself every time it infects a new machine making it harder for a detector to recognize it.

Existing malware detectors(Anti-virus Engines) are based on signature matching, thus they are based on purely syntactic information and can be fooled by such techniques.

6.2.9 Packers

Packers are software which compress the executable. They were initially designed to decrease the size of executable files.

However, the malware authors recognized very quickly that decreasing file size will decrease number of patterns in the file, so less chances of detection by anti-virus.

6.2.9 Packers

Anti-virus basically work by matching patterns (signatures).

So, this effectively increases the chances of malware to go undetected.

Some virus writers have gone to the limit of creating their own packers(such as Yoda packer) while others use readily available packers such as UPX.

Some packers use anti-debugging tricks also.

6.2.9 Packers

Packer facts:

- Packers allow to compress/encrypt applications
- You cannot see the code of the application using a disassembler, you need to unpack it first.
- Packers compress applications and add a small loader to the file.
- The loader will decompress the binary in memory, resolve imports, and call the Original Entry Point (OEP).

Polymorphic code aims at performing a given action (or algorithm) through code that mutates and changes every time the action has to be taken.

The mutation makes them very difficult to detect.

6.2.10 Polymorphism

There have been only a few polymorphic viruses and they still are not detected 100% by most of the anti-viruses.

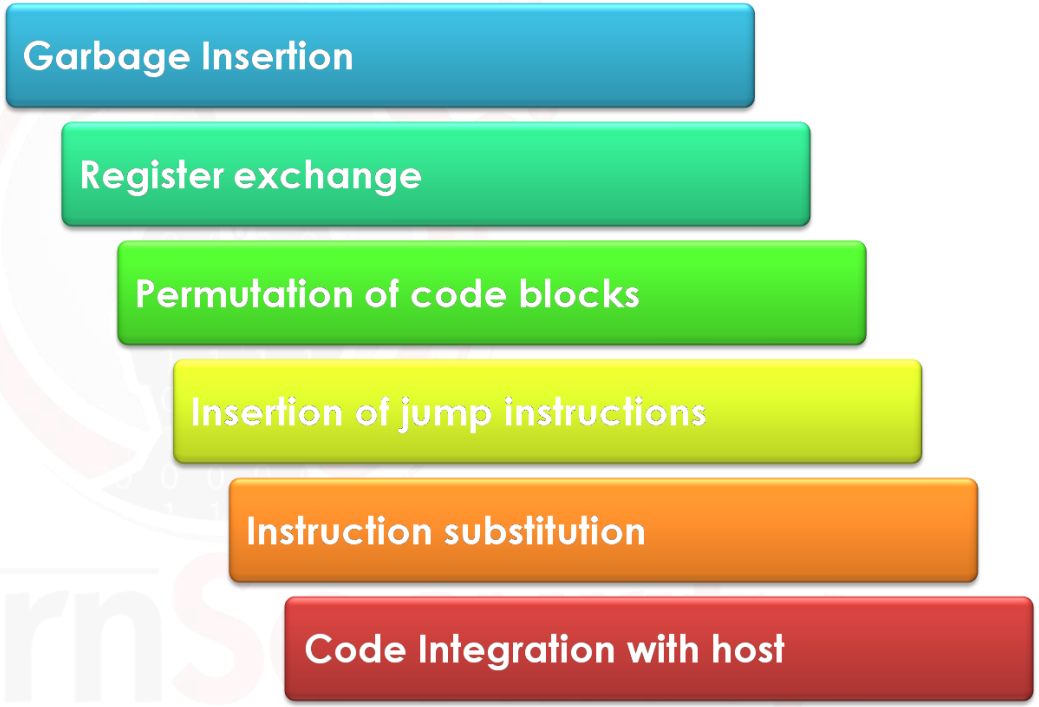
All polymorphic viruses have a constant encoding and variable decryptor. So a virus using a different XOR key to encrypt its variant also falls into polymorphic category.

This brings me down to one of our favorite topics - metamorphism.

It can be best defined as polymorphism with polymorphism applied to the decryptor/header as well.

There are numerous ways to implement metamorphism/polymorphism (both are similar with some minor differences).

Some of which are documented below:



Garbage Insertion:
Garbage data/instructions are inserted into the code, for example NOP instructions (0x90) are inserted.

Register exchange:
The registers are exchanged in all the instructions.

For example, see the 2 snippets of code given below which are using register exchange.

6.2.11.2 Register Exchange

First Snippet

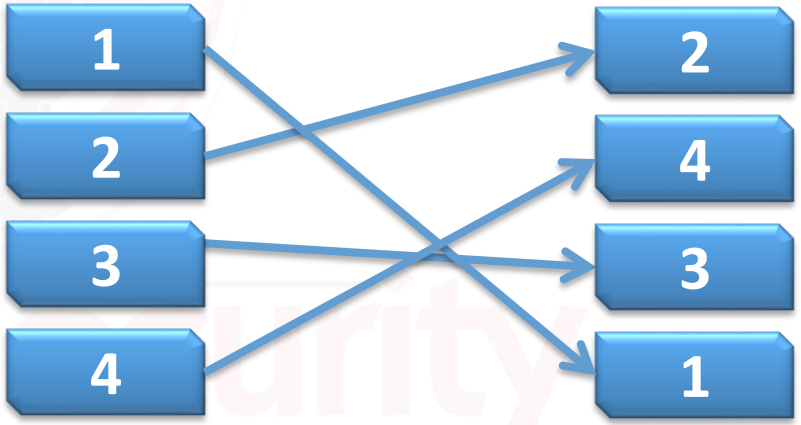
code bytes	disassembly
5A	pop edx
BF04000000	mov edi,0004h
8BF5	mov esi,ebp
B80C000000	mov eax,000Ch
81C288000000	add edx,0088h
8B1A	mov ebx,[edx]

Second Snippet

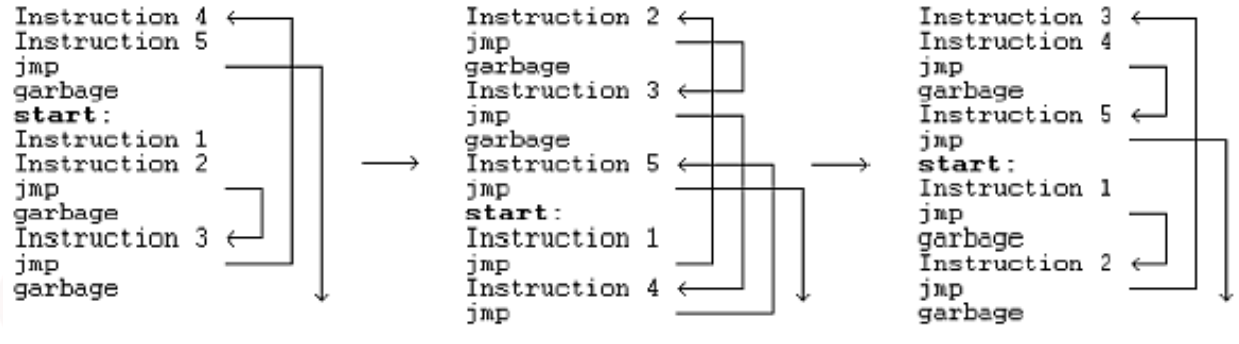
code bytes	disassembly
58	pop eax
BB04000000	mov ebx,0004h
8BD5	mov edx,ebp
BF0C000000	mov edi,000Ch
81C088000000	add eax,0088h
8B30	mov esi,[eax]

Permutation of code blocks:
In this type of mutation, code blocks are randomly shuffled and then fixed up, so that the execution logic is still the same.

This technique is very powerful.



Insertion of jump instructions:
Some malware mutate by inserting jumps after instructions (the instruction is also relocated), so that the code flow does not change.



ZPerm can directly reorder the instructions in its own code

Instruction substitution:
In this type of mutation, one instruction (or set of instructions) are replaced by 1 or more different instructions which are functionally equivalent to the replaced set.

The example shows few example rules, which can be used for substitution.

6.2.11.5 Instruction Substitution

REG stands for registered. **imm** stands for immediate number such as 0x800 (can be any numeric value). **imm8** stands for 1 byte (8 bits).

Original CODE

```
ADD    reg, imm
MOV    reg, reg/imm

SUB    reg, reg
TEST   reg, reg
LODSx

STOSx

CMD    reg, imm8
DEC    reg
INC reg
```

Transformed CODE

```
SUB    reg, -(imm)
PUSH   reg/imm
POP    reg
XOR    reg, reg
OR reg, reg
MOV    ACUM, [esi]
ADD    esi, SIZE
MOV    ACUM, [esi]
ADD    edi, SIZE
CMD    reg, imm32
SUB    reg, 1
ADD    reg, 1
```

Code Integration with host:
In this type of mutation, the malware modifies the target executable(which is being infected) by spraying its code in regions of the EXE.

Zmist virus used this technique very effectively.

To hide more changes such as changes in file size, the malware might compress the original code (or can even damage the file completely) to survive.

How Malware Spreads?

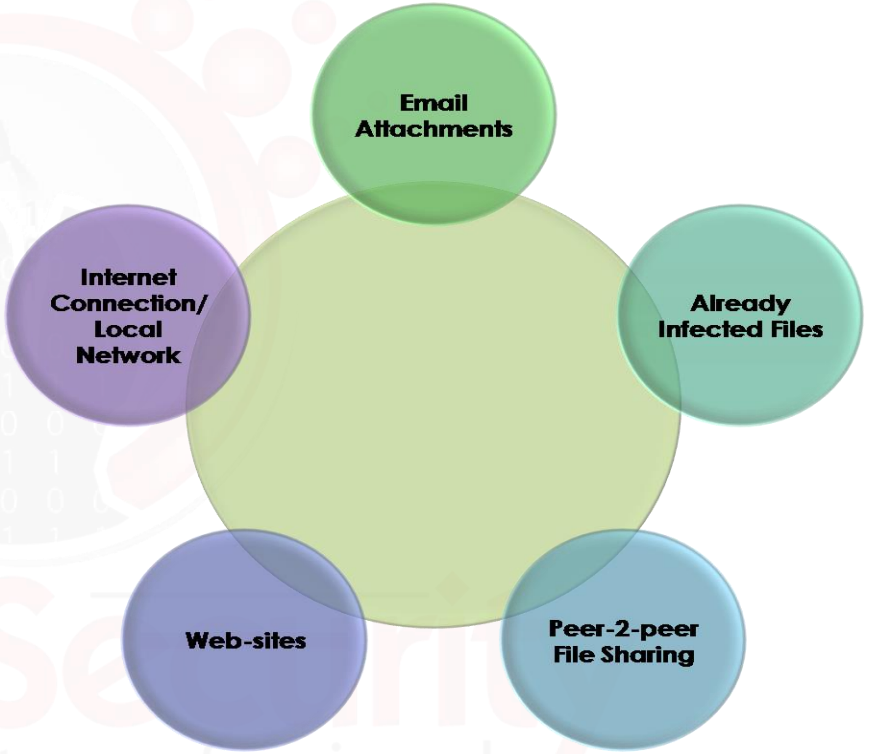
6.3 How Malware Spreads?

In this section, we will cover the techniques used by malware to spread in general.

Malware basically spreads via a large number of mechanisms such as:

- Email Attachments
- Already Infected Files
- Peer-2-peer File Sharing
- Web-sites
- Internet Connection/Local Network

These methods will now be discussed in detail.

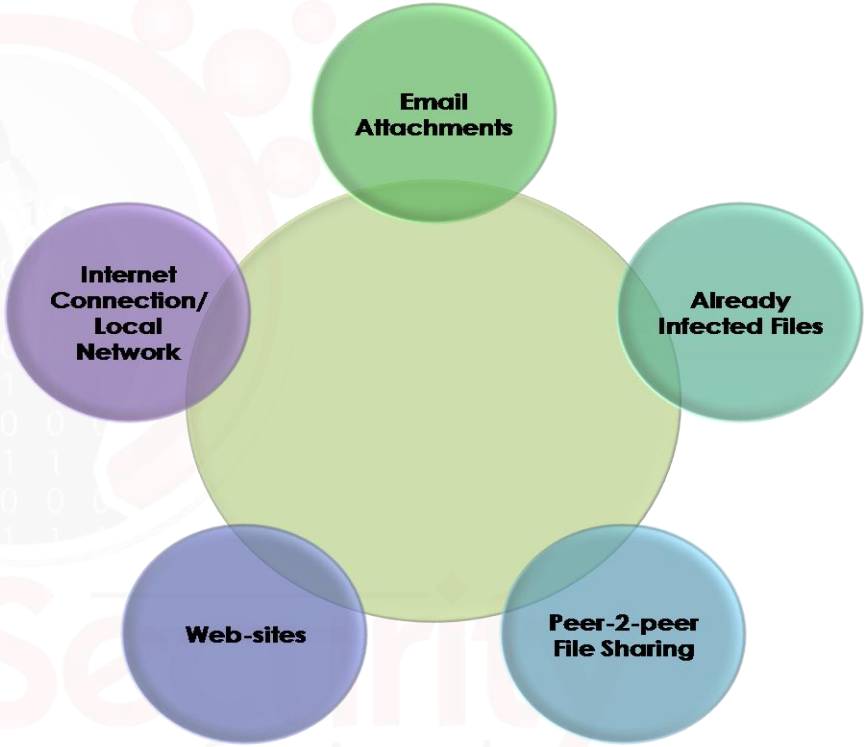


6.3 How Malware Spreads?

One of the easiest and most effective ways is via email

Attachments are attached to the junk message.

Social engineering should invite the user to execute the attachment.

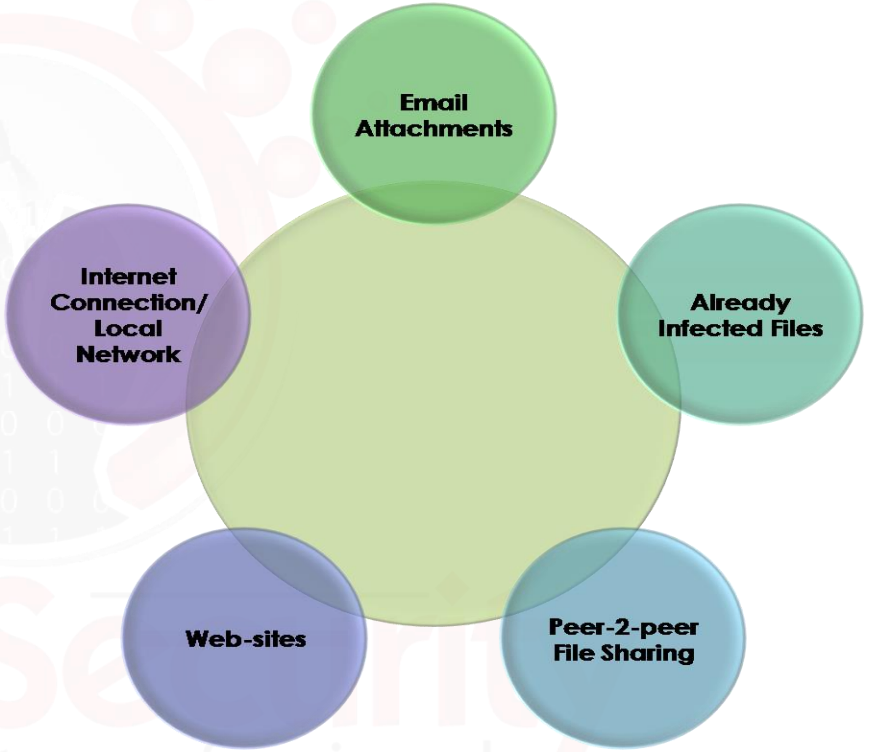


6.3.1 Email Attachments

For example, a junk mail is received with an attachment “kitten_playing.jpg.exe.”

Windows by default hides the extension, so the user just sees “kitten_playing.jpg” and the user falls into the trick believing that it is an image and innocently clicks it.

Let us say that you were infected by a Z virus and it was there for around 1 day, before you noticed and removed it.

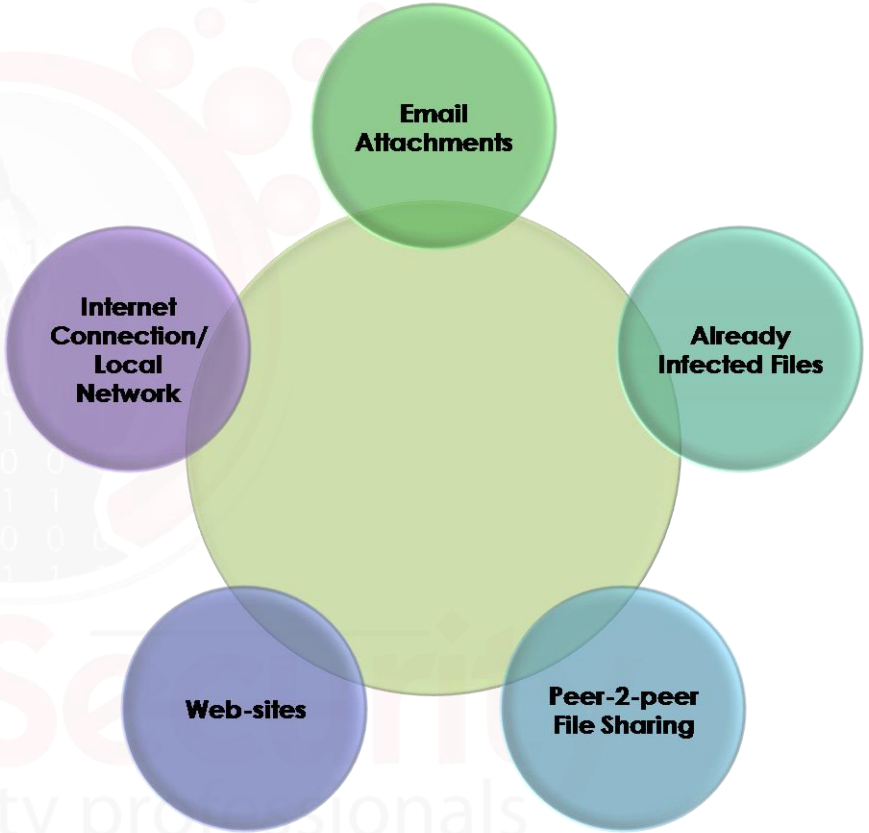


You should really think about what it might have already done during the 1-day time-frame.

If the virus was a re-infection kind, then it must have infected numerous other files, which open access/execution will activate the virus again.

Once you are infected with a virus, it is then very hard to remove the infection from the system.

P2P file-sharing was started with sharing of music files (via Napster).



6.3.3 Peer-2-peer File Sharing

However, due to its tremendous popularity, a number of new networks started sprouting which shared all files unlike previous network.

Nowadays, around 30%-40% of all available files in file-sharing networks can be assumed to be infected with malware.

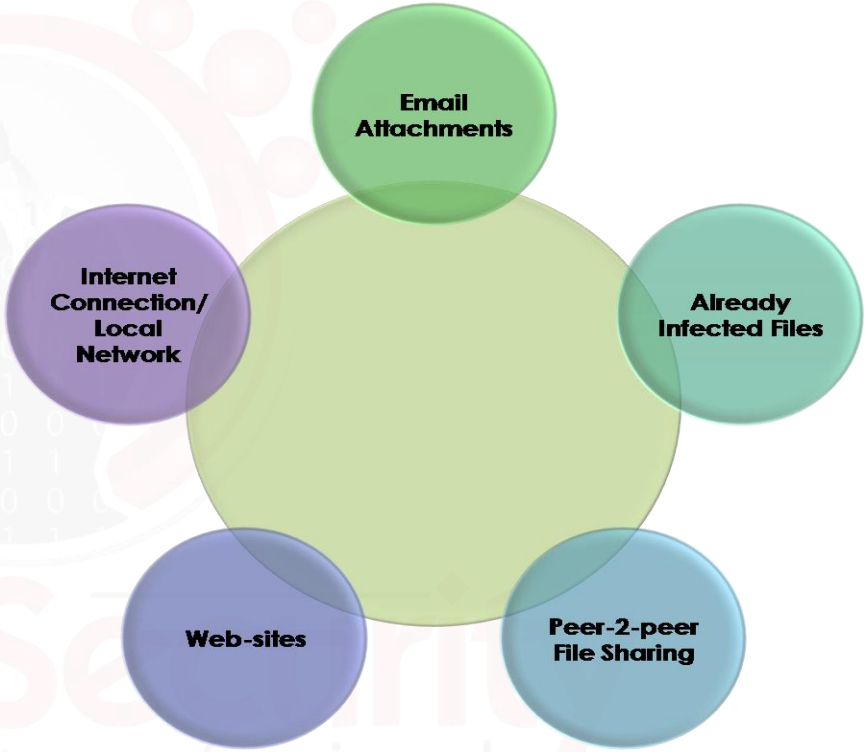
6.3.3 Peer-2-peer File Sharing

Moreover, some of the clients (programs used to access some specific network) are ad-supported or are pre-bundled with malware.

Also, since these clients have to be connected to internet to function, their network-interfacing code has not been verified by any 3rd party. They might contain hidden back-doors (both knowingly and unknowingly).

6.3 How Malware Spreads?

Yes, websites are the second most (maybe topmost) method used by malware to spread.



These kind of attacks are also called drive-by-downloads. Let us say, you are using a browser X to visit website Y.

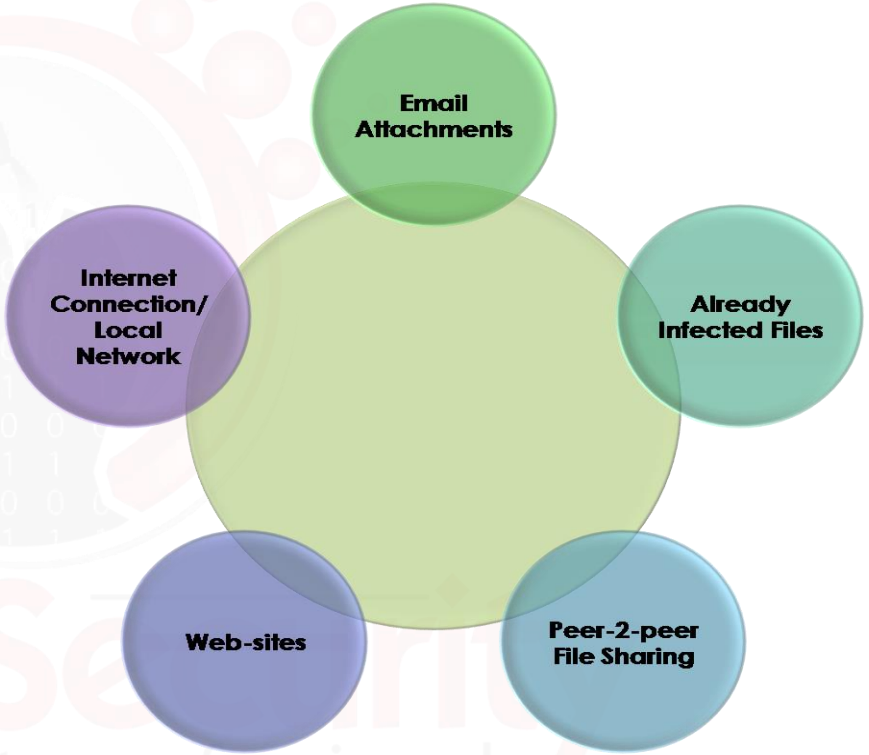
Now if website Y, exploits your web-browser, then it gets the chance to run its code on your system which will download remaining part/body of the malware.

Drive-by downloads are triggered upon visiting an HTML page.

This includes email addresses!

6.3 How Malware Spreads?

Almost all computers get connected to a network these days, it can be a local internal network or internet itself.



These kind of glitches also occur in Windows networking stack (Linux also has its own set of bugs) which are found from time and time , and then rapidly exploited by malware to spread directly.

Some of the known examples are RPC-DCOM vulnerability in Windows 2000/XP which was exploited on a very large scale all over the world.

6.3.5 Internet Connection / Local Network

These kind of vulnerabilities do not require any user action.

However, these kind of attacks can be stopped by using correctly configured firewalls or simply applying patches.

Remember that we have written “Correctly configured firewall” because a badly configured firewall does not offer any protection at all.

However, do note that firewalls themselves have their own sets of bugs, so they themselves can be vulnerable, so the best way to be protected is to update your software as soon as possible.

6.3.5 Internet Connection / Local Network

This brings us to another important point, some people think that they have updated their OS, Antivirus, firewall and other software, so they are 100% secure.

There is no 100% security ever.

The only machine 100% secure is that disconnected from the network and locked down physically and of course is not in use.

6.3.5 Internet Connection / Local Network

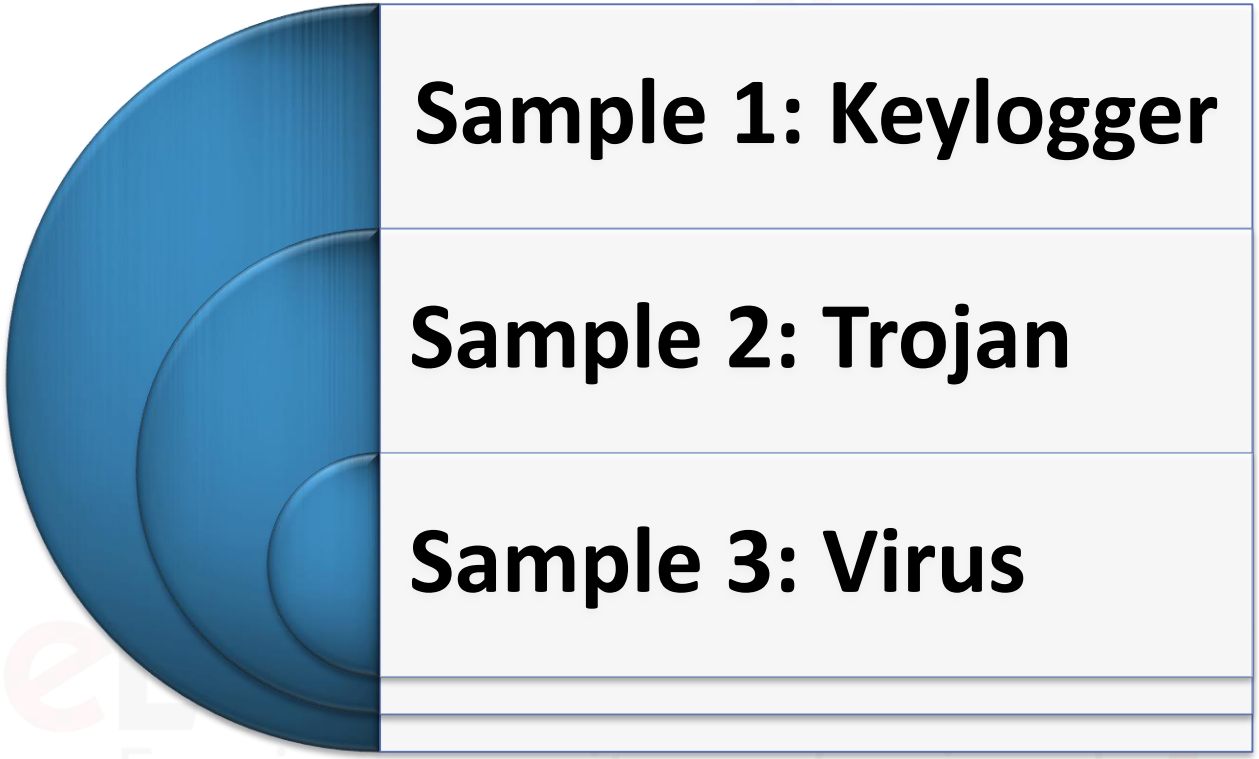
Let us say an Anti-viruses updates itself every 24 hours.

Now just after the update, a new virus starts hitting the networks.

So, just think who will be protecting you during the next 23 hours; I do not think that it will be your OS or antivirus.

It would be your luck for sure.

SAMPLES



6.4.1 Sample 1 : Keylogger

Since this is a Penetration Testing course, you should pay particular attention to the privacy violation of this kind of tool.

If you're thinking of using it against one of your target organization employees make sure to ask for written permissions during your engagement negotiation phase.

6.4.1 Sample 1 : Keylogger

We are providing you a very simple keylogger for Windows based OS.

Keyloggers are being used world wide to capture keystrokes, passwords but also as spy tools.

A very small keylogger can fit in less than 50 lines of C code.

Ours fits in around 80 lines.

This is a user mode CPU-intensive keylogger.

6.4.1 Sample 1 : Keylogger

A better design would be a silent one, but this is just supposed to give you an overview, right?

This keylogger is based on `GetAsyncKeyState` API. This function can be used to obtain the state of any key on the keyboard asynchronously.

So, what we do is we check the state of all the keys of the keyboard one-at-a-time and if the key-state is pressed then note it down.

6.4.1 Sample 1 : Keylogger

You can download the source code of our Keylogger in the members area.

Sample 1 Keylogger 

6.4.1 Sample 1 : Keylogger


```
while(1)
{
 for(i=8;i<=190;i++)
        {
                  // check keys from code 8 to code 190
                 if (GetAsyncKeyState(i) == -32767)
                {
                 print_key (i);
                }
        } // end for
 }// end while
```

As you can see in the code above, we fall into an infinite loop and then keep-on checking whether the user has pressed any key.

As soon as we detect that user has pressed a key, we invoke a `print_key` function which prints the key to screen. This code is not supposed to be used as a full-featured key-logger.

Since we are informing you about malware, the course would be incomplete for sure without a working trojan in action.

The one we are covering here is the famous
NetBus Trojan ver. 1.7.

The old-school reader will surely remember about this tool from the 90's.

This trojan is supposed to be easily usable with lots of features. The features list is :

- Open/Close CD-ROM
- how optional BMP/JPG image.
- Swap mouse buttons.
- Start optional application.
- Play music file.
- Control mouse.
- Shut down Windows.
- Show different types of messages to user.
- Download/Upload/Delete files
- Go to an optional URL.
- Send keystrokes and disable keys.
- Listen for and send keystrokes.
- Take a screen-dump.

6.4.1 Sample 2 : Trojan

If you want to test it we advise you to use a virtual OS.

Also what follows is the removal process.

From this you should understand how the trojan infects the machine.

An how to use guide (along with the Netbus executable files) is given in the members area.

Sample 2 Netbus 

1

Find out the name of the NetBus-server (which is most often `Patch.exe`).

6.4.1 Sample 2 : Trojan

2

Run

```
RegEdit.exe
```

and lookup the registry-key:

```
\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
   \CurrentVersion\Run
```

3

From that key you should be able to sort out the NetBus server program (again, most often `Patch.exe`) from others.

The offending program normally ends with

```
/nomsg
```

4

When you have found the suspicious entry, do a file-search for `"[Name of the NetBus-server].exe"` on your system.

Finally run: `"[Name of NetBus-server].exe /remove”`

Virus detection is done by matching the patterns within virus code with the database signature.

Instead of giving you some binary junk code, we will give you an open-source code for a Windows virus (you will have to compile it yourself).

The virus name is Win32.Dissolution (as detected by various Antiviruses).

NOTE:
The virus has not been written by us and is provided as-is.

The comments have been improved for better understanding.

You can download the full source code from the members area:

Sample 3 Dissolution 

The virus spreads by adding its code to the PE file and then changing the entry-point to the virus body.

The virus does not try very hard to escape the AVs.

Let us try to understand how it works.

6.4.1 Sample 3 : Virus

Virus…

1. Gets the delta offset and save the starting location of the virus
2. Saves registers incase the host program needs them
3. Gets the location of the `kernel32.dll` in memory
4. Uses the **`GetFunctionAddresses`** procedure to get the `kernel32` API function addresses.
5. Calls the **`FindHostFile`** procedure to find a valid PE file to infect.

6. Calls the **`GetHeader`** procedure which reads the PE header into memory

7. Calls the **`AddCodeToHost`** procedure which does many things:
   - Writes this program in memory to the end of the host file
   - Updates the last section header to include all the data up to the EOF, Updates its virtual size, and makes it Readable/Writable/Executable
   - Updates the program image size
   - Sets the entry point to the virus code
   - Adds a signature to location 79h to stop another infection
   - Calls **PutHeader** procedure which writes the updated PE Header to the host

8. Calls **`AddToRegistry`** procedure which adds the last infected file to the registry

9. Restore registers for the host program

10. Returns control to the host program

6.4.1 Sample 3 : Virus

However, you can easily check the strength of your antivirus by slightly modifying the file and compiling it. E.g.

```
RegistryName   db 'Start-up Program', 0
Db      'Vorgon, Canada, 2003'          ; Signature
```

and so on and you will be surprised with the results!

References

Practical Malware Analysis

https://www.amazon.com/Practical-Malware-Analysis-Hands-Dissecting/dp/1593272901/

Reversing: Secrets of Reverse Engineering

https://www.amazon.com/Reversing-Secrets-Engineering-Eldad-Eilam/dp/0764574817/ref=pd_bxgy_b_img_b