A lot of malicious programs are targeted towards windows OS. Hence, a basic understanding windows OS will help a malware analyst to understand how a sample uses OS features to execute code and develop host-based indicators for that sample and understand the purpose of the malware.

Software that is non-malicious and useful is well formed and follows Microsoft’s guidelines for building applications for windows computers. Malware, on the other hand, is not well formed and it uses windows OS functionality in specific ways. This chapter helps us to understand some basic ways in which malware tries to use windows functionality.

**Windows API:**

API-stands for application programming interface, which is a group of functions and protocols which act as an abstraction layer and access point to systems other than the native system. Windows API is an extensive set of functionalities which is available to a user to make use of the system functionalities, and malware uses it in specific ways. Understanding windows API usage convention helps an analyst to understand what functionality a sample is trying to access.

**Types and Hungarian notation:**

Windows API uses Hungarian notation, which is a prefix naming scheme which allows us to identify key constructs in a piece of code. Eg;

dw- is a prefix for DWORD, which is a 32-bit unsigned integer. If you see a variable called dwSize being passed as an integer to any function, we can assume what it is. There are similar schemes to other data types and even functions in the windows API.

Table 7-1 in page 169 has a list of such prefixes which add a significant meaning to the context they are being used in and help an analyst to understand them as such.

**Handles:**

A Handle is like a pointer in as it is used to refer to an object or a memory location and usually refers to a process/file/window/menu. However, unlike usual pointers, they cannot be used in arithmetic operations and don’t always refer to the memory address. You can only use a handle to store it somewhere and use it later to refer to the same object.

A handle is an abstraction which hides the real[physical] memory address from the API user.

Resolving a handle- locks a physical address to a handle.

Releasing the handle- absolves the memory to a pointer.

It could be thought of an index to a table of pointers and the OS can change the pointers at will, but the handle will always point to the real process, thereby allowing the OS to be able to use memory efficiently. In the modern OS, even the pointers themselves are opaque and do not resolve perfectly into physical addresses.

**File System Functions:**

Usually a malware accesses file system resources to create new files or modify existing files. If the name of the created files is unique, they are good file system indicators and analyzing file system activity can give us a hint as to what the malware intends to do. Common file system resources are CreateFile, ReadFile and WriteFile(read and modify files).

FileMapping- is the process of loading a file from disk to memory. createFileMapping function loads a file from memory into disk. MapviewofFile gives the base address of the file in memory and can be used to read or write anywhere in the file easily.

Whereas read and write file functions work as a stream, which means the program calling the above two functions must access the file as a stream (a group of bytes at a time, which are sequential).

**Special Files**:

Windows has many files which can be accessed just like regular files but are not accessible by browsing through the directory structure. When these are targeting, it is harder for some one to be able to notice the physical manifestations of an infection (Increase/decrease in file size or change of instructions inside a file, or metadata files). Some of these files has greater access to system resources and infecting them would help malware to command these resources. These files can be accessed by file opening functions when the name of the file is sent to them as an operand.

Shared Files:

Are the files shared between many users and are typically located on a server.

Files Accessible via Namespace:

A Namespace can be thought of as a virtual folder structure visible to the user even if the physical files are in different servers. Hence, even if the files are on different locations, a single path ca be used to access them.

NT(New technology) is the lowest level namespace and is what is being referred generally in programming as \(root in linux systems). Some malware use the \ NT namespace to read/write from devices directly, since the NT namespace has access to all of them.

EG; \\ .\physicaldisk1 can be used by a malware, to access physical disk 1 directly and edit the disk in ways which are not allowed or even possible by accessing the files VIA windows API. Since any API requests go through the kernel, there can be lots of constraints o what could be done with the file depending on the file type. If the sample has access to NT namespace, then its free for all at the kernel junction.

ADS:

Stands for additional data streams and the naming convention is filename.txt:stream$DATA and it can be used to add additional data to a file. If data is added to a file using ads, it is not displayed to the user and wont be displayed in the directory structure. It can only be access by using a stream.

**Registry:**

Registry can be thought of as a ledger and keeps a list of all os and new programs processes and functions and where the code for those are in physical memory. The registry was created to store configuration information and as more applications have been using it, its importance has grown.

Early windows versions used .ini files to store system configuration information and can be a source of reliable host-based indicators. Registry is used by malware to be persistent across reboots and there are lots of ways for a sample to achieve this function. Below is the key terminology of registry;

Key- is an entry in the registry and top-level keys are called root keys which are divided into subkeys.

Root key- Registry is divided into five sections also called HKEY or hive and consists of subkeys.

Value entry and Data: A value entry is an ordered pair with a NAME and VALUE (also called data).

The root keys are as below;

HKEY\_Current\_Machine- Configuration settings that are global for the machine.

HKEY\_Current\_user- configuration for a present user.

HKEY\_Classes\_Root- Stores information for defining types.

HKEY\_Current\_config- stores information about hard ware configuration, specifically differences about original and present configuration.

HKEY\_USERS- Settings for all users.

1. Some keys are virtual keys which are used as a reference to actual keys. HKCU is stored in HKEY\_USERS\SID, SID- is the security identifier for the user currently logged in.
2. For persistence, the malware adds itself to the key; HKEY\_Local\_Machine\Software\MICROSOFT\windows\CurrentVersion\RUN- which is a group of programs run while the system is booting.

Use regedit to check your windows OS registry and autoruns tool to check code that starts when your OS boots.

**Registry Functions:**

Common registry functions accessed by malware to edit os registry are; RegopenKeyex, regsetvalue and reggetvalue.

Note: Information after comments is most likely the value of the parameter.

Registry configuration can also be edited by running a .reg script, which merges exiting registry information with the new entry(Example of this file is given in page number 176).

**Networking API:**

Malware sometimes downloads/uploads information to and from the infected host computer and it uses native windows API to connect to the internet. Comonly used api’s are defined in ws2\_32.dll(winsoc-widows socket) and the convention is Berkley compatible sockets and its implementation is similar in both windows and Unix like systems. Common functions accessed are listed in page 176. WASstartup function must be called before any other network related functions as it is needed to allocate resources for networking libraries. Set a breakpoint here. Also, there are two sides to networking; client and server and a sample could be implementing any of these two.

Client: A socket call followed by a connect call for initial connection and send and receive during transmission.

Server: socket, bind, listen and accept to form an initial connection and send and receive.

Note: Above pattern is common for malicious and non-malicious software and cannot act as network-based indicators.

An example of a network functions being accessed for malicious purposes is illustrated in Page 177.

**WinInet API:**

In addition to winsoc API there is an additional API called wininet api which implements application layer protocols such as http and Ftp. Functions used for accessing these protocols are InternetOpen, InternetOpenUrl and InternetReadfile and InternetFiledownload and is used to access, read from or write to files on the internet and download files.

**Triggering Execution of other files:**

A sample often gets other processes and executables to run to copy themselves or infect those files. A simple way to do that is to infect a DLL.

A DLL (Dynamic linked library) is a group of functions in the windows OS which export key functionality to be used by other processes. The advantages to using DLL’s are;

1. Many processes can use a DLL loaded in memory (as long as they are asynchronous and locks are implemented properly) and each process doesn’t need to load the library twice, they just need access to a function which can be requested.
2. If the system of DLL’s is reliable, developers need only to import them into their applications and the size of those deliverables will be much smaller.

The above two points mean that through DLL’s are significantly powerful, there are lots of ways in which a malicious sample will try to infect them. Common ways are;

1. Use a DLL to copy itself to any process which needs that DLL (Step 1: Copy itself to the DLL. STEP 2: Copy itself to the process from the DLL).
2. By using windows DLL’s, they can spread to any program by copying themselves to a the dll.
3. They can also use a third party dll to circle back to infect the system.

A basic DLL structure:

As we all know, there is only one flag in a file which differentiates between a .exe and DLL (Hence the suggestion in chapter 2 to edit this flag and run it as a .exe). DLL’s use a PE format ad the only difference between a DLL and exe is the exported functions. The function which needs special consideration here is Dllmain, and it is the file’s entry point. This function is called to notify the DLL (By DLL, it refers to the dynamic process which is being executed in memory) when;

1. A process loads or unloads a library.
2. A thread starts or finishes execution.

Dynamic linking process can be explained as below; It is a multi-process multi thread illustration:

1. A process(P1) is loaded into memory, it is allocated some virtual address space (VA1).
2. This process is multi-threaded, and it starts all the threads (T1 and T2) by first creating two stack frames for those processes and creating arguments needed for those threads for the threads to be executed.
3. When a thread needs access to a DLL’s resources (exported function), the process calls a loadlibrary/loadlibraryEx function.
4. The load library function calls the dll’s main method to notify the dll as to what functions are needed from the dll. This is used by the dll to manage its own per process or per-thread resources.
5. The dll is loaded into the virtual address space of the process, but it uses the stack frame of the thread which has needed its resources (The DLL is also notified of this when dll main method is called when a new thread is created or finished execution).
6. The system maintains a per-process reference count to each DLL. Eg;

P1 has T1, T2, T3 and T4 and P2 has T5, T6, T7 and T8. If Both of the process need Dll1, and

No process and no thread has called dll1----

P1 T1 has loaded Dll1. P1-Dll1 ref count 1(the dll has been loaded into the virtual address space of p1).

P1 T2 has loaded dll1. P1-Dll1 ref count 2()Dll main has been called and been notified that another thread has requested to use dll1’s functions.

P2 T5 has loaded Dll1. P2-Dll1 ref count-1.

So on and so forth.

1. When the reference count for a process-Dll pair has reached zero, the dll will be unloaded of the processes’ virtual address space.

Important points to note:

1. When a process is loaded into memory, it is given an address space to use by the OS. Though the pointer looks physical, it is not a reference to an actual physical space. The OS has a virtual address VS physical address table to resolve the virtual and physical pointers.
2. Though it looks like many processes are using the same space, they are not. Look at note 1.
3. A sample which infects a process and gets access to an address, say 0x0040A010 will only be able to access the physical address referred to by the virtual address.
4. Resolution of virtual address to physical address is as below;
5. Process loads into memory after the OS allocates space for a process, and returns the a pointer.
6. This pointer, though virtual, is resolved to a physical address via a table.
7. OS manages this table (called page table).

Malware infects processes:

1. Create process is used to create a process. Malware often uses process creation to execute its own code in ways described below.
2. Commonly, create process is used to also open a socket connection and download further instructions. An Illustration;
3. Create process function has many parameters, one of which is STARTUPINFO structure which has a handle to the standard input, output and error streams. A malware author can set these up to include a socket, and either upload/download data just like form the disk.
4. Also, malware sometimes hides its malicious code in the resource section, copies the file to disk and creates a process to execute that file (There was one sample like this in chapter two). We need to examine these samples with resource hacker and try to analyze the file saved from the resource section.

**Threads:**

A Thread is an independent flow of execution (which can be explained as a light weight process). Important differences are;

1. Each process has its own virtual memory space where as threads within a process must share the space within a process.
2. Each process could contain more than one threads.
3. We have not much control over which thread is executed one or when the scheduler switches execution of threads.

**Thread Context:**

Consider a scenario in which there are two threads which are being executed simultaneously and name them T1 and T2. T1 and T2 both copy from an address addr onto eax and push it onto the stack.

1. Lea eax addr
2. Push eax.

If T1 pushes its value to eax and its scheduled out of execution and T2 pushes its variable onto eax. Now when T1 is scheduled in and what would happen if it pushes the value stored by T2 onto the stack? Chaos!

The answer to the above scheduling concerns is thread context. Thread context could be explained as a framework within a thread can maintain its environment (variables, registers and flags and its own instruction pointer EIP) between scheduled in and out of execution. When a thread is scheduled for execution, its thread context is loaded and EIP is used to fetch the next instruction. When A thread is scheduled out of execution, its context is saved to be reloaded when its scheduled to be executed again.

**Malware Using threads:**

Malware can use threads in below two ways; both of which involves using the function create thread. Create thread has the following parameter; startaddress- the location at which the thread is supposed to start executing from.

1. A thread can be used to load a malicious dll’s by calling the loadlibrary function.
2. Createthread is called by giving the name of the library as a start address (When loadlibrary is called which then calls the DLLmain function as specified in the DLL section).
3. Malware can create two threads, one to read from a socket/pipe and execute instructions received and to also read a file in disk and upload it to a socket/pipe. Both threads may be called by a single process, but their functionality is different.
4. To examine which thread does what, look at the start address of the threads. Code for these functions would explain which thread does what in the sample.

**IPC with Mutexes:**

We have processes to execute instructions and programs and threads to execute instructions separately to increase efficiency. We also need these processes to coordinate effectively to access system resources like memory, input and output and access to devices. Enter mutexes; Which are global object which control and grant access rights to processes and within processes, threads.

A Mutex is a **Mutual Exclusion object**, which as the name suggests, means that if one object possesses access an item, it disqualifies other objects from gaining or controlling resources handled by the mutex. Mutexes are usually hardcoded, and when a suspicious sample tries to gain access, then it usually makes for a good host-based indicator. Threads which want access to a mutex call waitforasingleobject function and locks the mutex. When a mutex is locked, other threads wanting to access the mutex must wait and are queued by the scheduler until the thread which has the lock calls for a RelelaseMutex function.

A Malware tries to open(OpenMutex) to get access to another processes’ mutext or creates a mutex(CreateMutex).

Question: How is this relevant or even how does creating and opening a mutex will help it to maintain only one execution process?

Malwares can gain access to resources by interfering with mutexes in the following way;

1. A malware checks for a specific mutex or can get a list of mutexes currently operational and try to either close/tamper with the mutex to gain access and lock it.

**Services:**

Service can be described a piece of code which is run by the windows OS without any user prompt. Running malicious code as a service has certain advantages, which are;

1. They are usually run by using SYSTEM privileges, which has more access than admin/user accounts.
2. They can also be set up to run automatically when the OS boots and a user looking at tasks in the task manager cannot find these services being executed.
3. Malware authors sometimes name their samples to run as services (Svhost.exe) in order to appear as legitimate programs.
4. Use autoruns to observe all the services which are running on your computer even at boot time.

**Services API:**

OpenSCManager- Returns a handle to the service control manager. All programs which interact with services must access this function.

CreateService- Creates a new service in the service manager and allows the caller to specify if the service will be called manually or during boot time.

StartService- This function is used to start If a service is set up to be called manually.

Different services in windows execute in different ways (named service types)

Win32\_Share\_Process- Code for these services is combined to run in a single DLL and run as a single process. EG; svchost.exe- which combines a lot of services and run as a single process. Other service types include win32\_own\_process and KERNEL\_DRIVER(used to load code in the kernel).

All services in a system is noted in the registry at HKLM\SYSTEM\Currentcontrolset\services. If you want to add, delete, starting and querying services, SC is a command line tool in windows.

**MS COM:**

Microsoft component object model is an interface model standard which makes it possible for different software components to call each other code without knowing the specifics of how each of them is implemented (kinda like SOAP WSDL).

COM standard works as a client server model, in which COM Objects themselves are servers and the objects that use them are clients. Number of COM objects are already provided by Microsoft to be used by programmers to develop applications. Each thread that utilizes com must call oleinitialize or CoinitializeEx function. When analyzing a sample, searching for the above two calls might help to determine whether it is using COM, but it would not tell us much about the malware itself. We need a couple of object identifiers to determine which COM object is being used and to what purpose.

1. Since COM objects are provided by Microsoft, they are usually common across different systems and OS versions and are referred to by Globally Unique Identifiers (GUIDs) which consist of two items Class Identifier (CLSID) and Interface Identifiers (IID)

There are some pointers in page 188 as to how malware uses COM client and servers to infect systems.

**Kernel Mode VS User mode:**

Kernel Mode malware require advanced malware writing capabilities. Most Antivirus tools run in kernel mode. Most Malware runs on user mode.

Most user mode programs must run through windows API and windows api redirects the request to the kernel. Microsoft has developed this muti-step process to distinguish between processes which have kernel access and processes which don’t have kernel access.

Page 192 has some pointers. A few of them are defined below;

NTDLL- is the boundary between user and kernel modes and is a group of functions which make up the Native API. User Applications interact with NTDLL via kernel32.dll, which in turn interacts with the kernel mode process called ntoskrnl.exe which in turn manipulates the kernel data structures. Though User applications are not supposed to call ntoskrnl directly, there is nothing that prevents them from doing so. Calling Native API is useful for malware writers because it allows them to do thing which otherwise are not possible. There is a lot of functionality that is not available to user mode applications that interact with kernel 32 dll but is available by calling ntdll.

A well designed Anti virus program should monitor calls from processes at all levels of abstraction. Common functions that are called are in page 193.

Good- Labs are remaining in both chapter 6 and 7.

Labs Chapter 7:

May be we need the full version.