CSec15233
Malicious Software Analysis

Analyzing Malicious Windows Programs

Qasem Abu Al-Haija

Windows Malware (1)

- Most malware targets Windows platforms and interacts closely with the OS.

- A solid understanding of basic Windows coding concepts will allow you to
  - *identify host-based indicators of malware,*
  - *follow malware as it uses the OS to execute code without a jump or call instruction, and*
  - *determine the malware's purpose.*

Windows Malware (2)

- Non-malicious programs are well-formed by compilers and follow Microsoft guidelines.

- Malware is typically poorly formed and tends to perform unexpected actions

The Windows API

- Broad set of functionality ➔ governs how program/ malware interacts with Microsoft libraries.
  - *Windows API is so extensive ➔ Developers of Windows-only apps have little need for third-party libraries.*
  - *Windows API uses certain terms, names, and conventions ➔ you should become familiar with before turning to specific functions.*

- Concepts,
  - *Types and Hungarian Notation*
  - *Handles*
  - *File System Functions*
  - *Special Files*

Types and Hungarian Notation

- Windows API has its own names to represent C data types
  - DWORD for 32-bit unsigned int and WORD for 16-bit unsigned int.

- Windows uses *Hungarian notation* for API function identifiers.
  - This notation uses a prefix to identify a variable's type.
  - Variables of a 32-bit unsigned integer start with prefix *dw*.
  - Hungarian notation makes it easier to identify variable types

Common API Types

Type (Prefix)

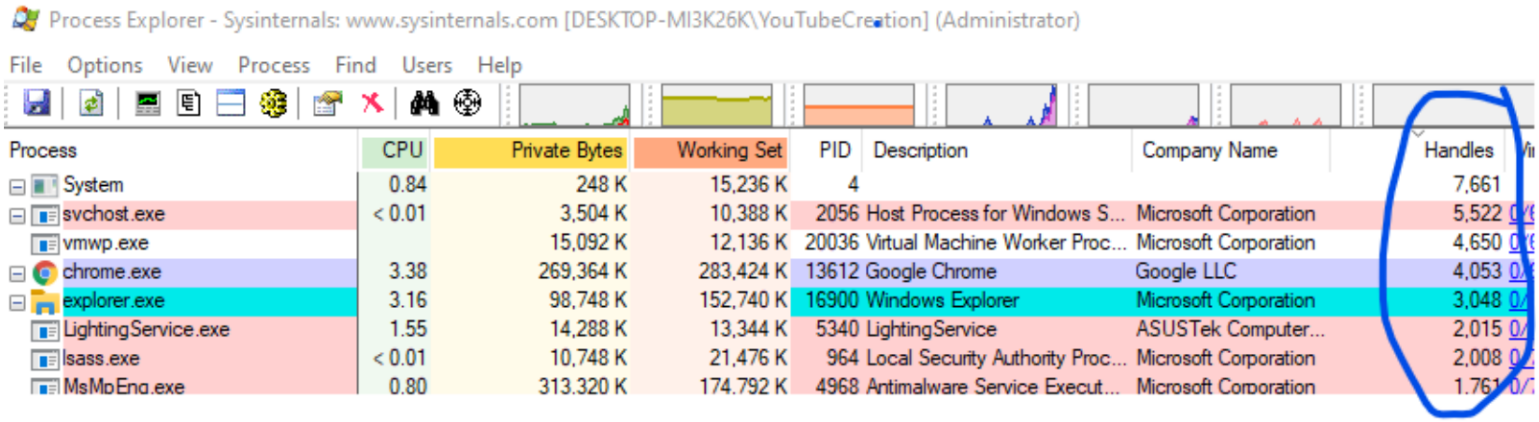
- WORD (w): 16-bit unsigned value
- DWORD (dw): 32-bit unsigned value
- Handle (H): A reference to an object
- Long Pointer (LP): Points to another type

Handles

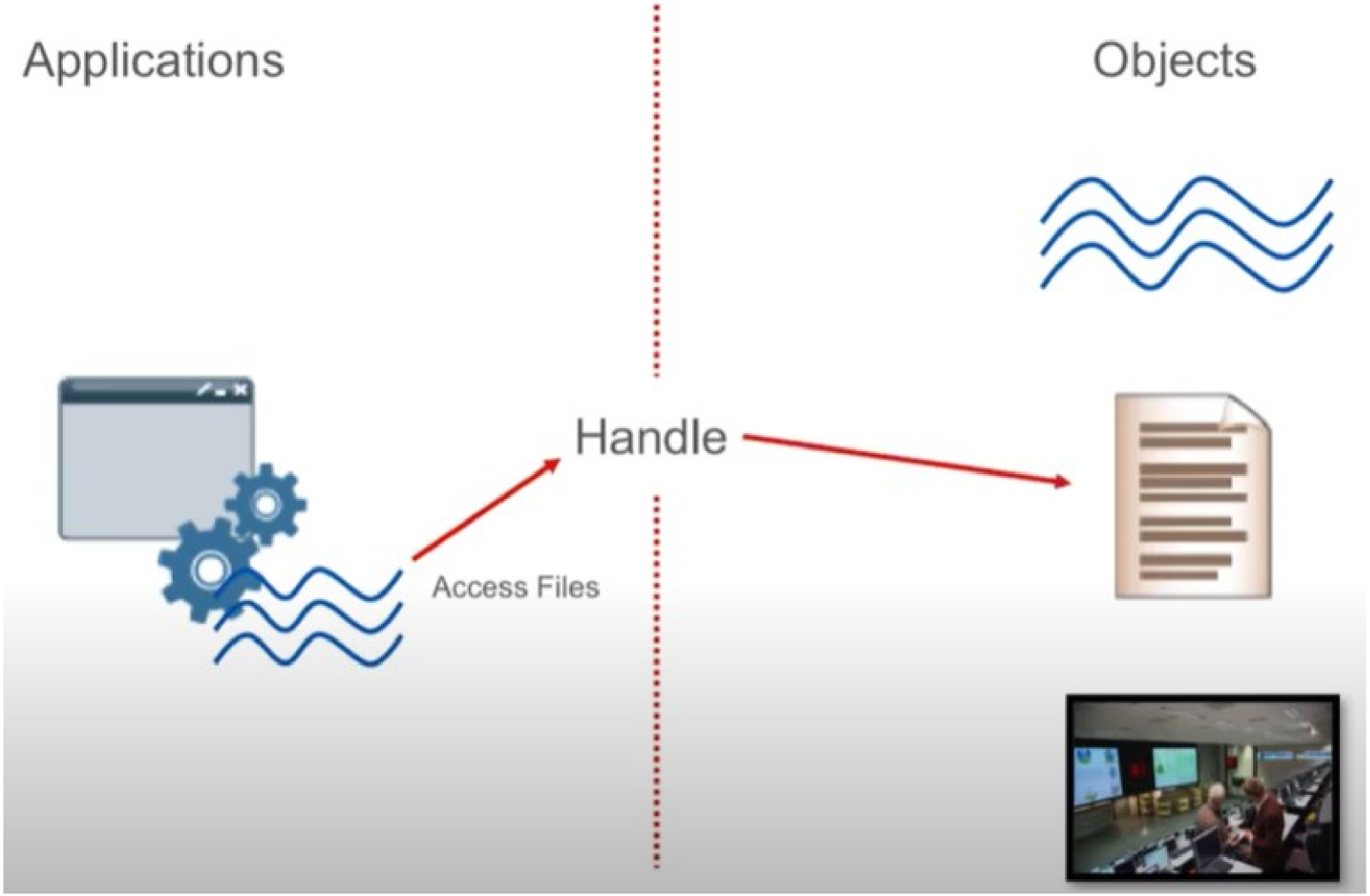
- Items opened or created in the OS, like
  - *Window, process, menu, file, …*
- Handles are like pointers to those objects
  - *However, unlike pointers, handles cannot be used in arithmetic operations.*
- The only thing you can do with a handle is to store it and use it in a later function call to refer to the same object

Handles

- **Handle Example: *CreateWindowEx* function**
  - ***CreateWindowEx*** *function returns an* ***HWND****, a handle to the window*
  - *To do anything to that window (such as* ***DestroyWindow****), use that handle*



Handles


Applications
Objects
Handle
Access Files
Thread
File
Graphic

File System Functions

- One of the most common ways that malware interacts with the system is by creating or modifying files,
  - Distinct filenames or changes to existing filenames can make good host-based indicators.

- File activity can hint at what the malware does.
  - For example, if the malware creates a file and stores web-browsing habits in that file, the program is probably some form of spyware.

File System Functions

- **CreateFile, ReadFile, WriteFile**
  - Normal file input/output

- **CreateFileMapping, MapViewOfFile**
  - Commonly used by malware, loads file into RAM
  - Used to replicate the functionality of the Windows loader.
  - Can be used to execute a file without using the Windows loader.

NOTE *File mappings are commonly used to replicate the functionality of the Windows loader. After obtaining a map of the file, the malware can parse the PE header and make all necessary changes to the file in memory, thereby causing the PE file to be executed as if it had been loaded by the OS loader.*

Special Files

- Not accessed by their drive letter and folder (like *c:\docs*).
  - *Malicious programs often use special files.*
  - *Some special files can be hidden (not shown in directory listings).*
  - *Others can provide greater access to the system HW & internal data.*
- Most common special files:
  - *Shared files,*
  - *Files accessible via namespaces,*
  - *Alternate data streams*

Special Files

Shared Files

- *serverName**share* or \\?*serverName**share*.
- They access directories or files in a shared folder stored on a network.
- The \\?\ prefix tells the OS to disable all string parsing, and it allows access to longer filenames.

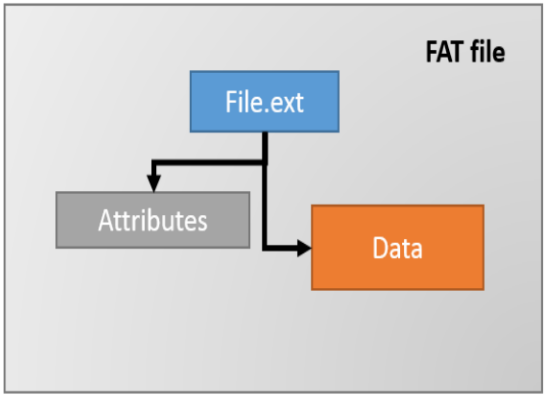
Special Files

Namespaces

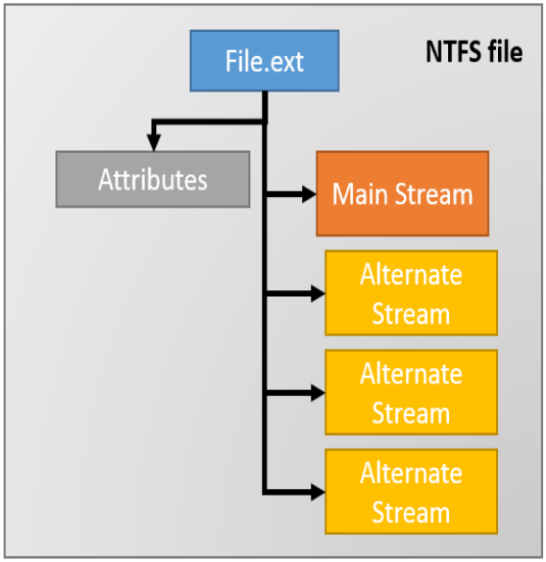
- *Special folders in the Windows file system.*
- *Fixed number of folders storing different object types.*
- *Lowest namespace prefix* **** *is known as NT namespace*
  - *Root folder: contains everything and has access to all devices.*
  - *all other namespaces exist within the NT namespace.*
- *Device namespace (***\\.***) used for direct disk I/O.*
  - *often used by malware to access physical devices directly and read and write to them like a file.*
- *E.g.,* ***Witty*** *worm wrote to* \\.*PhysicalDisk1 to corrupt the disk*

Special Files

Alternate Data Streams (ADS)



- A Microsoft NTFS Substructure, Originally for Macintosh Files. Not Visible to most Windows Applications.



- ADS allows additional data to be added to an existing file within NTFS, essentially adding one file to another.
  - ADS are a file attribute only found on the NTFS file system.
  - The extra data does not show up in a directory listing, and
  - Not shown when displaying the contents of the file.
  - Visible only when you access the stream

Special Files

- ADS data is named according to the following convention:

  normalFile.txt:Stream:$DATA

  - This allows a program to read and write to a stream.

- Malware authors like ADS because it can be used to hide data (e.g., Backdoor.Rustock.A)

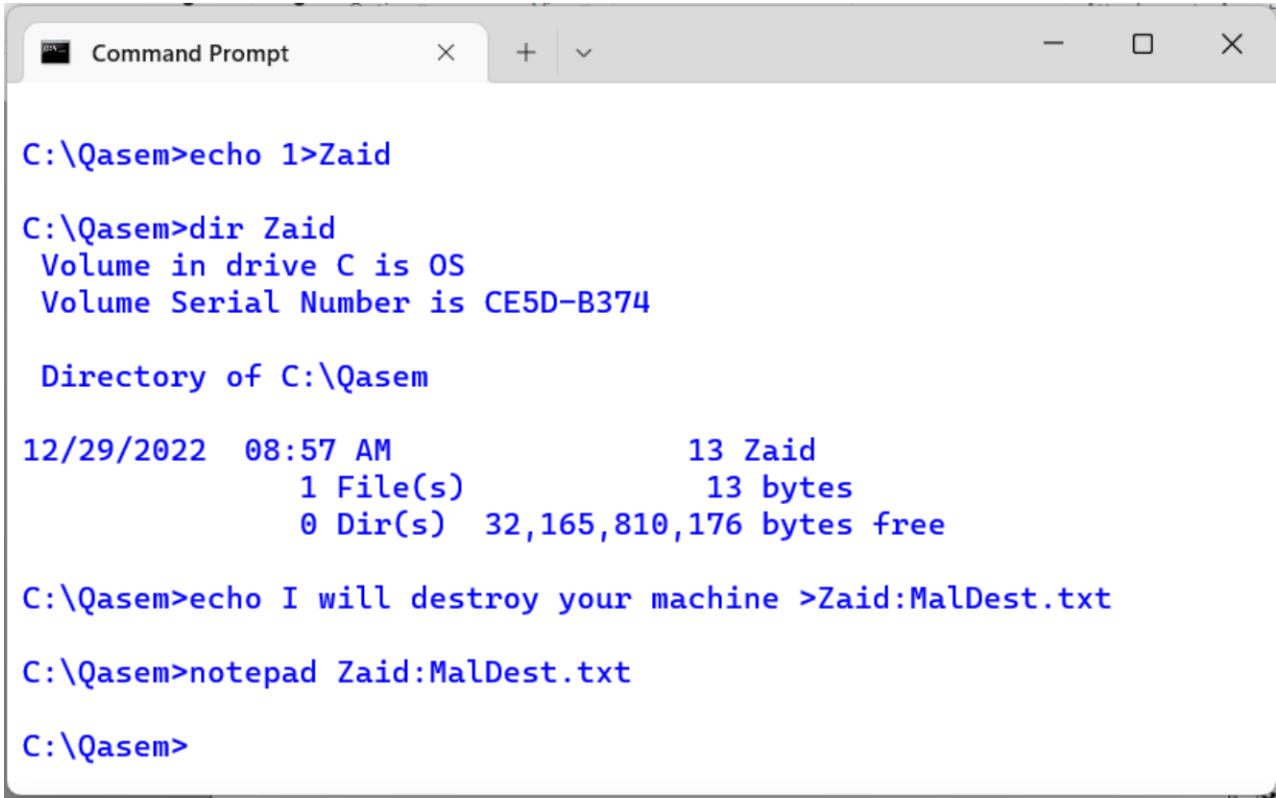
- The first tool you can use was developed by *Sysinternals* (later bought by Microsoft) and is called Streams

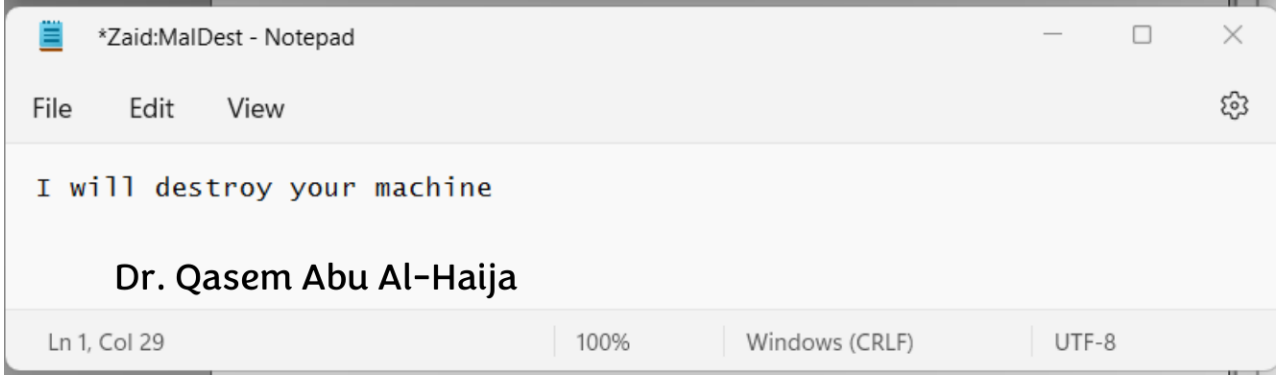
Special Files

- **Alternate Data Streams (ADS)**

Example:

Creating a hidden file with ADS using **Echo**

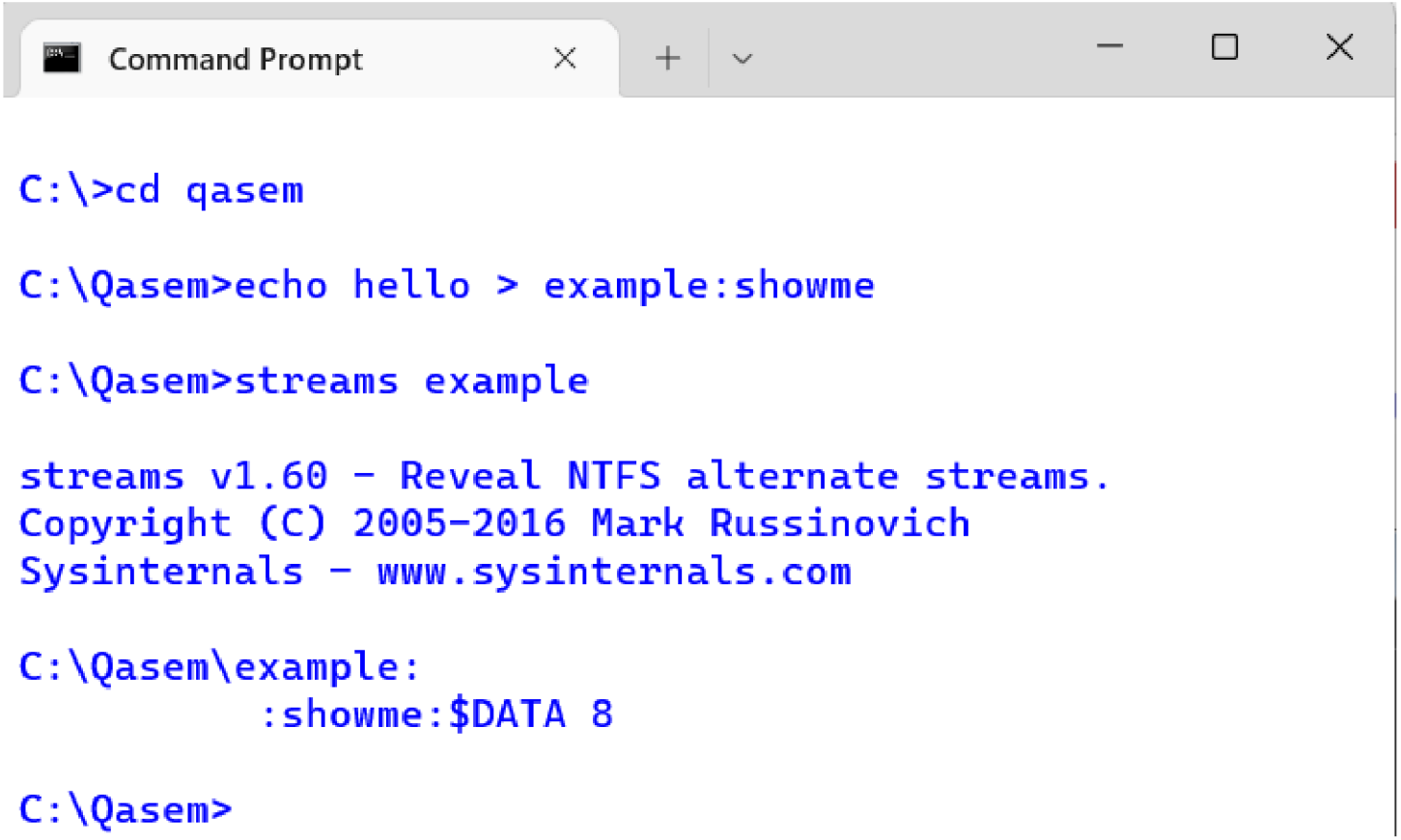




Special Files

- **Alternate Data Streams (ADS)**

Example: Checking Files having ADS using Streams



The Windows Registry

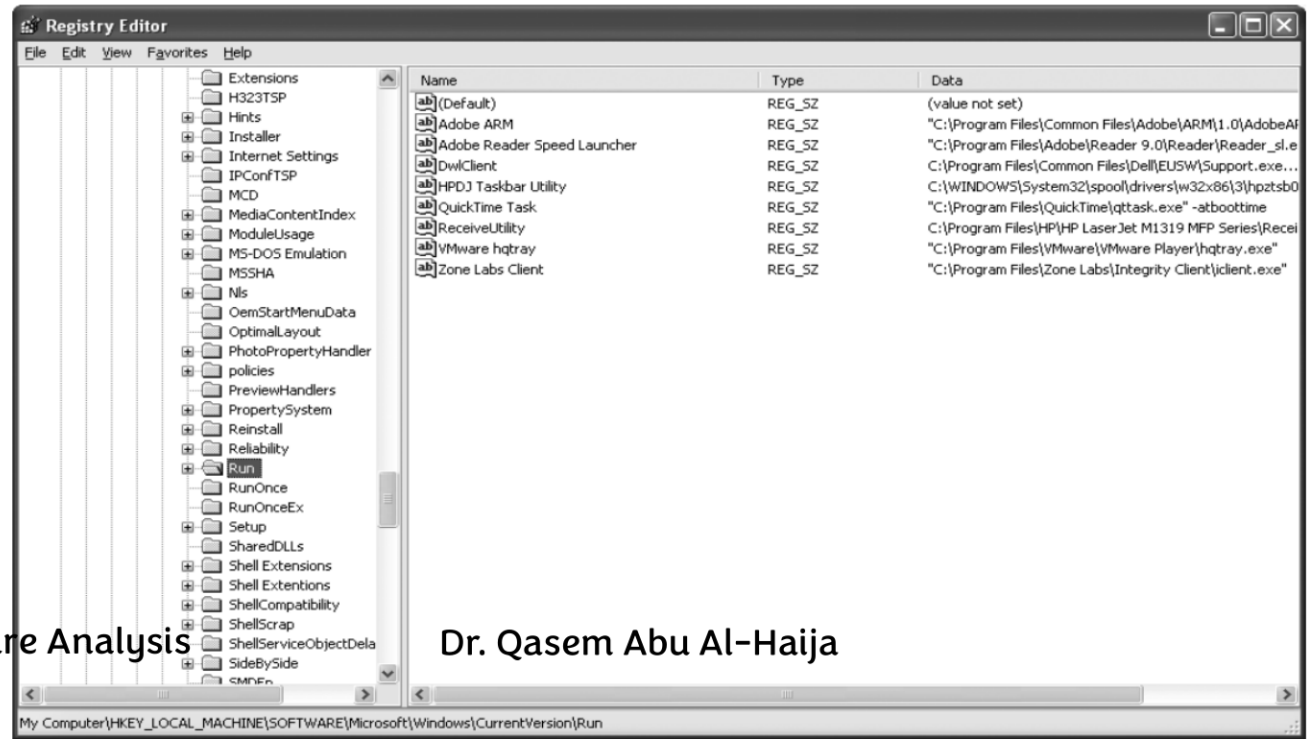
The Windows Registry

- Store OS and program configuration settings.
  - including networking, driver, startup, user account…
  - Good source of host-based indicators and can reveal useful information about the malware's functionality.

- Malware uses the registry for *persistence*.
  - The malware adds entries into the registry that allows it to run automatically when the computer boots.
  - The registry is so large that there are many ways for malware to use it for persistence.

Window Registry Tool

- ***Regedit*** (The *Registry Editor) tool:*
  - Built-in Windows tool used to view and edit the registry.
    - The window on the left shows the open subkeys.
    - The window on the right shows the value entries in the subkey.
    - Each value entry has a name, type, and value.
    - The full path for subkey currently being viewed is shown at the bottom of window.



Registry Terms

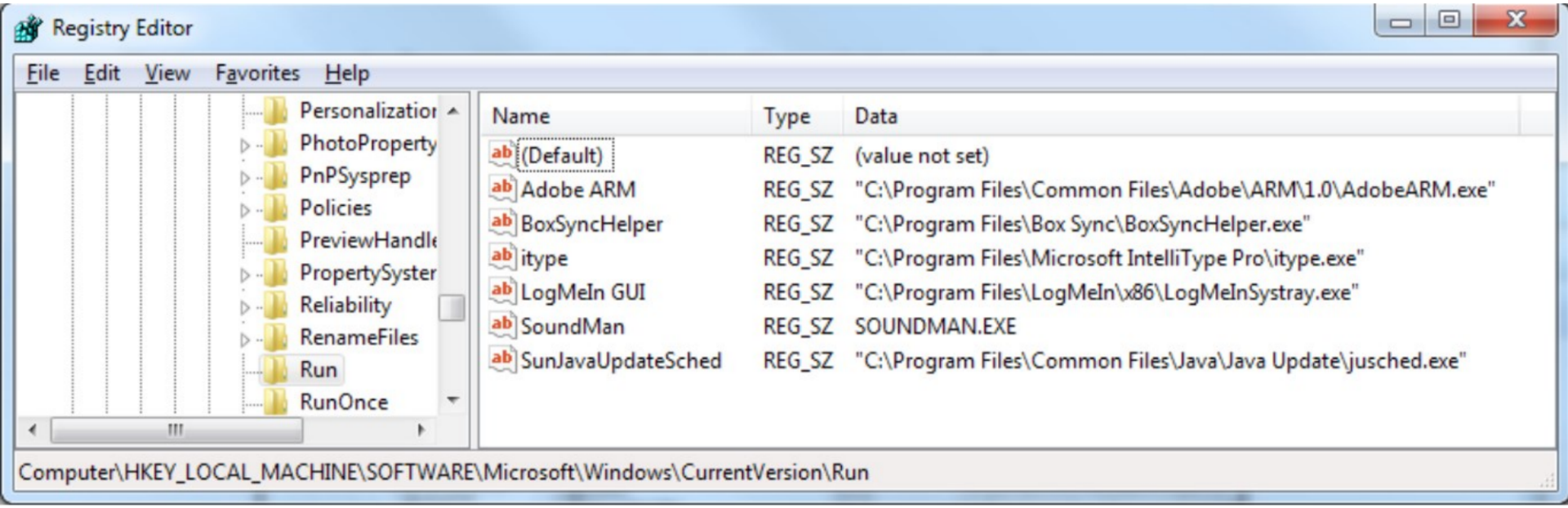
- **Root key**: Top-level sections called *root keys*.
  - There are five root keys, and each has a particular purpose.
- **Subkey**: subfolder within a folder.
- **Key**: A folder that can contain folders or values.
  - The root keys and subkeys are both keys.
- **Value entry**: An ordered pair with a name and value.
- **Value or data**: The data stored in a registry entry.

Registry Root Keys

- **HKEY_LOCAL_MACHINE (HKLM)**
  - Stores settings that are global to the local machine.
- **HKEY_CURRENT_USER (HKCU)**
  - Stores settings specific to the current user.
- **HKEY_CLASSES_ROOT**
  - Stores information defining types.
- **HKEY_CURRENT_CONFIG**
  - Stores settings about the current hardware configuration.
- **HKEY_USERS**
  - Defines settings for the default user, new users, and current users.

Run Key

- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
  - Executables that start when a user logs on
    - Programs that Run Automatically



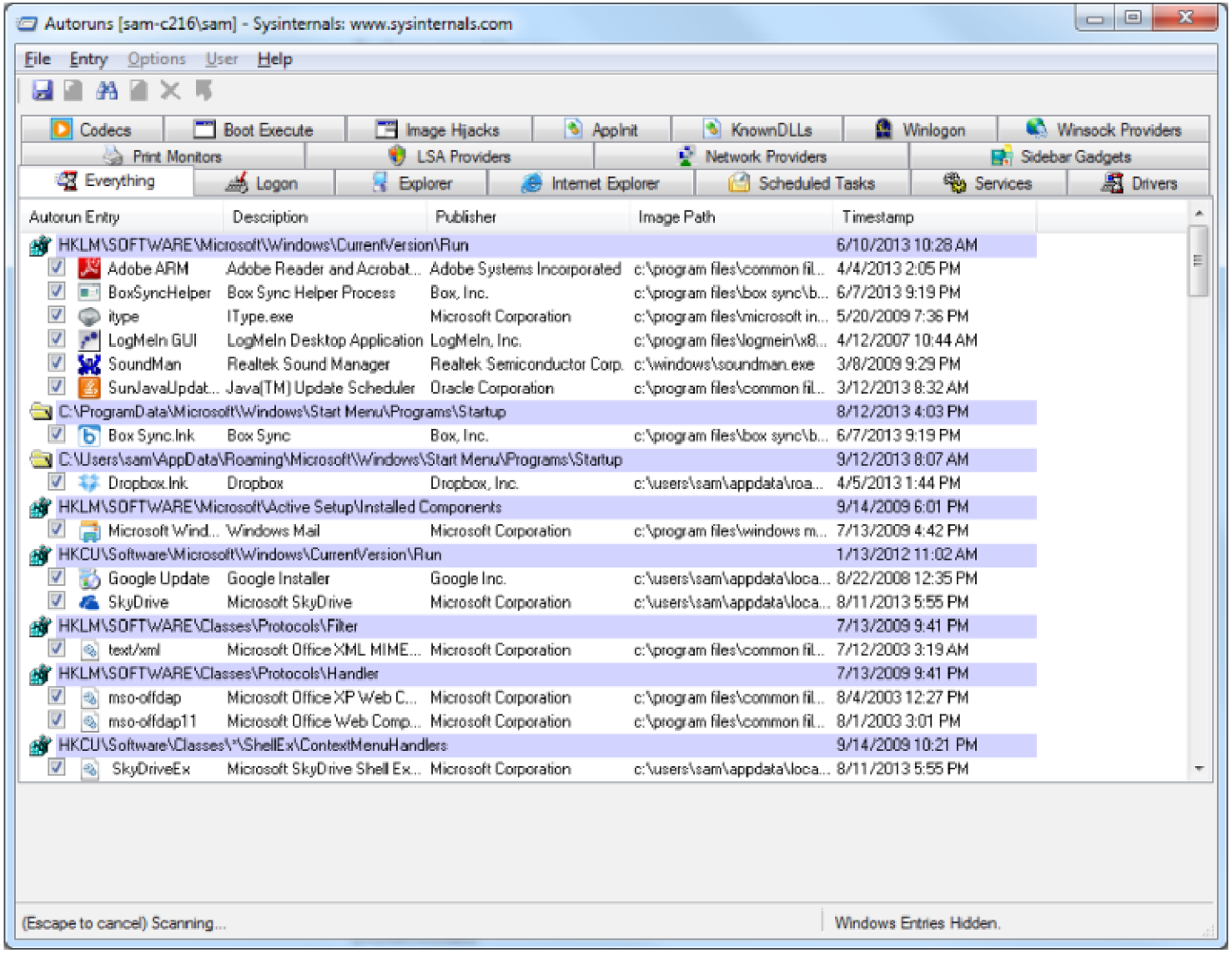
Autoruns tool

- Free from Microsoft (*Sysinternals tool*).

- Lists code that will run automatically when OS starts.
  - Executables that run.
  - DLLs loaded into IE and other programs.
  - Drivers loaded into the kernel.

- Autoruns checks 25 to 30 locations in the registry,
  - but it won't necessarily list all of them.

Autoruns tool



Common Registry Functions

- **RegOpenKeyEx**
  - Opens a registry key for editing and querying.
- **RegSetValueEx**
  - Adds a new value to the registry and sets its data.
- **RegGetValue**
  - Returns the data for a value entry in the registry.

Note: Documentation will omit the trailing W (wide) or A (ASCII) character in a call like RegOpenKeyExW

Common Registry Functions

FUNCTION NAMING CONVENTIONS

When evaluating unfamiliar Windows functions, a few naming conventions are worth noting because they come up often and might confuse you if you don't recognize them. For example, you will often encounter function names with an `Ex` suffix, such as `CreateWindowEx`. When Microsoft updates a function and the new function is incompatible with the old one, Microsoft continues to support the old function. The new function is given the same name as the old function, with an added `Ex` suffix. Functions that have been significantly updated twice have two `Ex` suffixes in their names.

Many functions that take strings as parameters include an A or a W at the end of their names, such as `CreateDirectoryW`. This letter does *not* appear in the documentation for the function; it simply indicates that the function accepts a string parameter and that there are two different versions of the function: one for ASCII strings and one for wide character strings. Remember to drop the trailing `A` or `W` when searching for the function in the Microsoft documentation.

Analyzing Registry Code in Practice

```
0040286F   push    2                  ; samDesired
00402871   push    eax                ; ulOptions
00402872   push    offset SubKey      ; "Software\\Microsoft\\Windows\\CurrentVersion\\Run"
00402877   push    HKEY_LOCAL_MACHINE ; hKey
0040287C ❶call     esi ; RegOpenKeyExW
0040287E   test    eax, eax
00402880   jnz     short loc_4028C5
00402882
00402882 loc_402882:
00402882   lea     ecx, [esp+424h+Data]
00402886   push    ecx                ; lpString
00402887   mov     bl, 1
00402889 ❷call     ds:lstrlenW
0040288F   lea     edx, [eax+eax+2]
00402893 ❸push     edx                ; cbData
00402894   mov     edx, [esp+428h+hKey]
00402898 ❹lea      eax, [esp+428h+Data]
0040289C   push    eax                ; lpData
0040289D   push    1                  ; dwType
0040289F   push    0                  ; Reserved
004028A1 ❺lea      ecx, [esp+434h+ValueName]
004028A8   push    ecx                ; lpValueName
004028A9   push    edx                ; hKey
004028AA   call    ds:RegSetValueExW
```

Listing 7-1: Code that modifies registry settings

Analyzing Registry Code in Practice

- **Listing 7-1 contains comments after the semicolon.**
  - Parameter names being pushed on stack (from MS documentation for a function being called)
  - For example, the first four lines have the comments samDesired, ulOptions, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", and hKey.
    - These comments give information about the meanings of the values being pushed.
    - The samDesired value indicates the type of security access requested,
    - The ulOptions field is an unsigned long integer representing the options for the call
    - The hKey is the handle to the root key being accessed.

- **In short, the code works as follows:**
  - The code calls **RegOpenKeyEx** function at ① with the parameters needed to open a handle to the registry key HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run.
  - The value name at ⑤ and data at ④ are stored on the stack as parameters to this function and are shown here as having been labeled by IDA Pro.
  - The call to **lstrlenW** at ② is needed in order to get the size of the data, which is given as a parameter to the **RegSetValueEx** function at ③ .

Registry Scripting with .reg Files

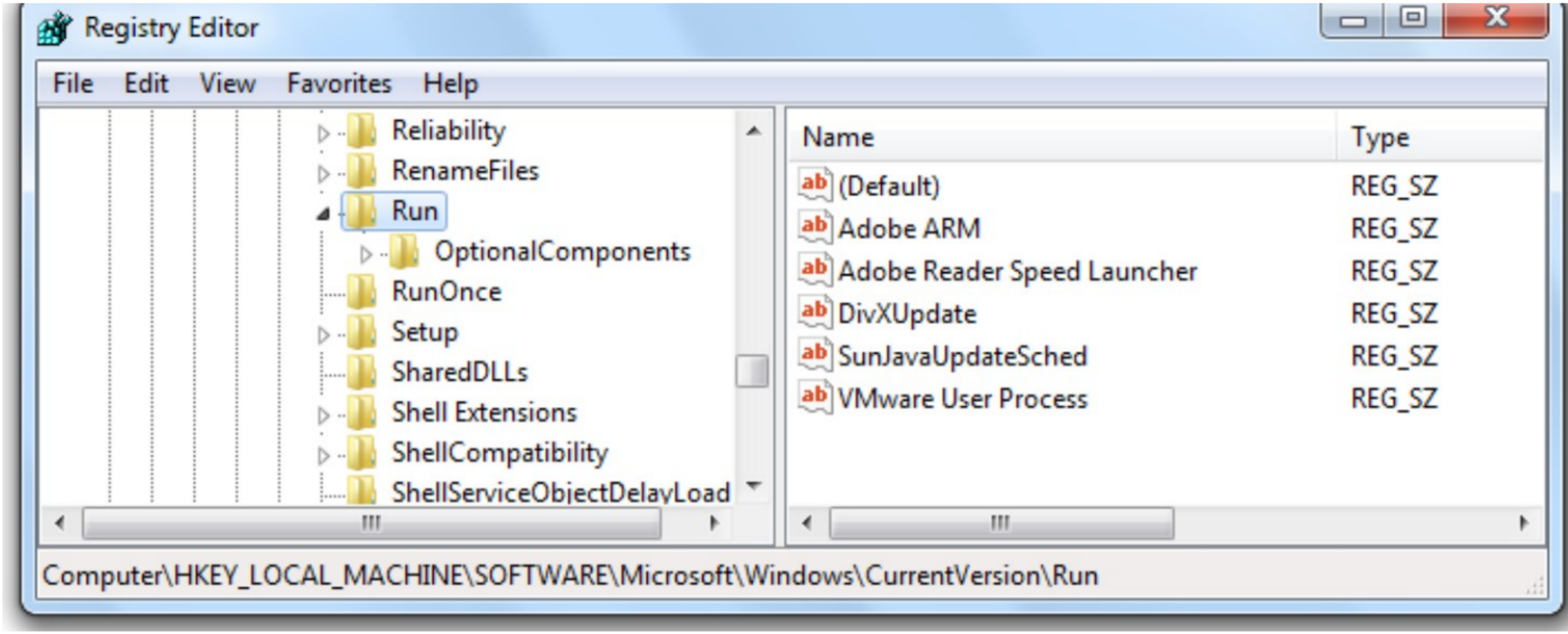
- Files with a *.reg* extension contain human-readable registry data.

- When a user double-clicks a *.reg* file:
  - it automatically modifies the registry by merging info from the file into the registry.
  - Almost like a script for modifying the registry.
  - Malware sometimes uses .reg files to modify the registry

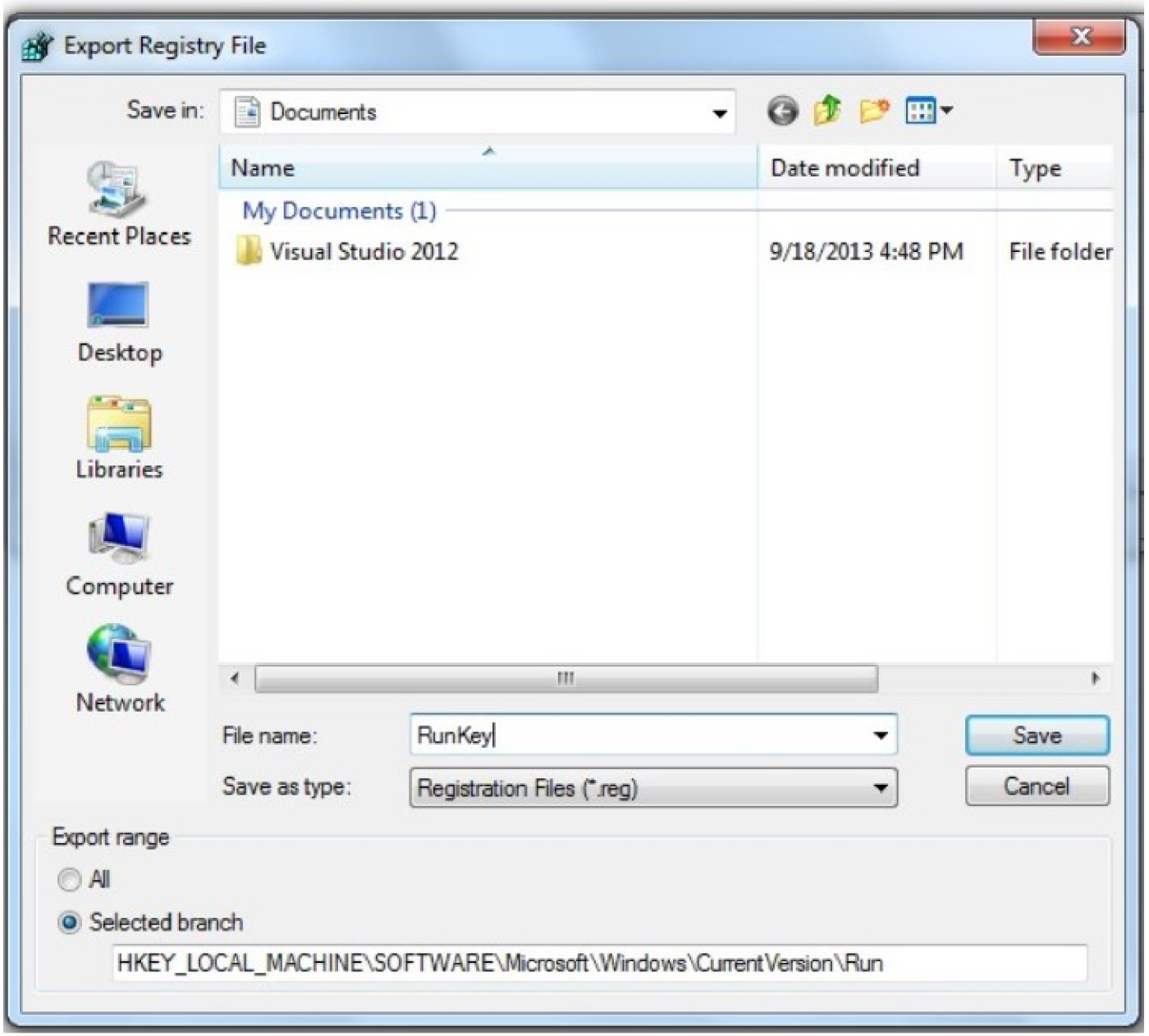
```
Windows Registry Editor Version 5.00

[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"MaliciousValue"="C:\Windows\evil.exe"
```

Listing 7-2: Sample .reg *file*

.reg Files


Registry Editor
File Edit View Favorites Help
Reliability
RenameFiles
Run
OptionalComponents
RunOnce
Setup
SharedDLLs
Shell Extensions
ShellCompatibility
ShellServiceObjectDelayLoad
Name
Type
(Default)
REG_SZ
Adobe ARM
REG_SZ
Adobe Reader Speed Launcher
REG_SZ
DivXUpdate
REG_SZ
SunJavaUpdateSched
REG_SZ
VMware User Process
REG_SZ
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

.reg Files


Export Registry File
Save in:
Documents
Recent Places
Desktop
Libraries
Computer
Network
Name
Date modified
Type
My Documents (1)
Visual Studio 2012
9/18/2013 4:48 PM
File folder
File name:
RunKey
Save as type:
Registration Files (*.reg)
Save
Cancel
Export range
All
Selected branch
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

.reg Files

RunKey.reg - Notepad

File Edit Format View Help

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"VMware User Process"="\"C:\\Program Files\\VMware\\VMware Tools\\vmtoolsd.exe\" -n
vmusr"
"SunJavaUpdateSched"="\"C:\\Program Files\\Common Files\\Java\\Java Update\
\jusched.exe\""
"DivXUpdate"="\"C:\\Program Files\\DivX\\DivX Update\\DivXUpdate.exe\" /CHECKNOW"
"Adobe Reader Speed Launcher"="\"C:\\Program Files\\Adobe\\Reader 9.0\\Reader\
\Reader_sl.exe\""
"Adobe ARM"="\"C:\\Program Files\\Common Files\\Adobe\\ARM\\1.0\\AdobeARM.exe\""

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents]
@=""

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents
\IMAIL]
@=""
"Installed"="1"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents
\MAPI]
@=""
"Installed"="1"
"NoChange"="1"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents
\MSFS]
@=""
"Installed"="1"
```

Networking APIs

Berkeley-compatible sockets

- Implemented in ***Winsock*** libraries, primarily in ***ws2_32.dll***.
  - Almost identical in Windows and Unix

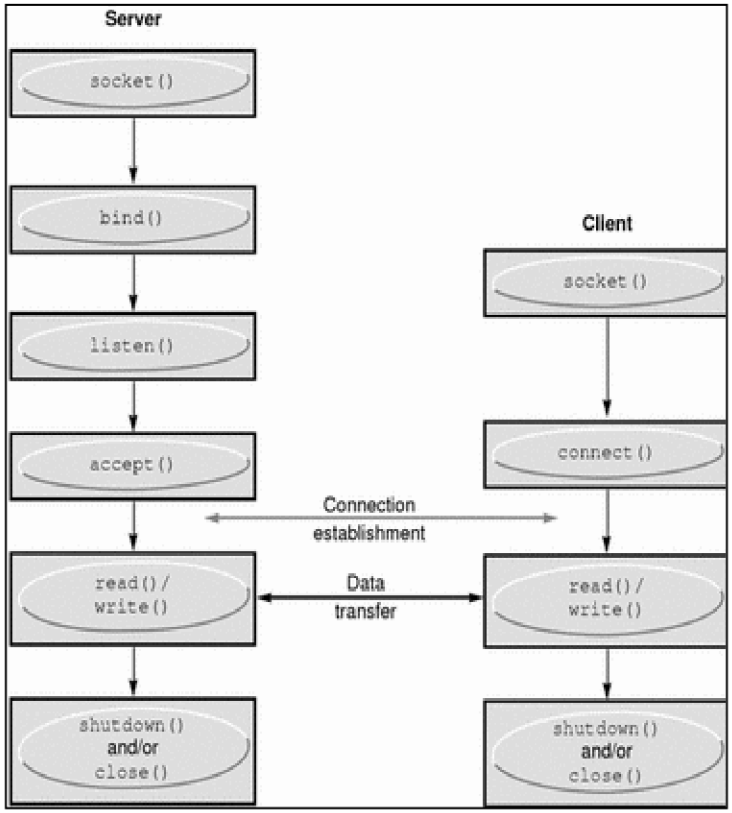
Table 7-2: Berkeley Compatible Sockets Networking Functions

Function	Description
`socket`	Creates a socket
`bind`	Attaches a socket to a particular port, prior to the accept call
`listen`	Indicates that a socket will be listening for incoming connections
`accept`	Opens a connection to a remote socket and accepts the connection
`connect`	Opens a connection to a remote socket; the remote socket must be waiting for the connection
`recv`	Receives data from the remote socket
`send`	Sends data to the remote socket

NOTE *The `WSAStartup` function must be called before any other networking functions in order to allocate resources for the networking libraries. When looking for the start of network connections while debugging code, it is useful to set a breakpoint on `WSAStartup`, because the start of networking should follow shortly.*

Server and Client Sides

- **Server side**
  - Maintains an open socket waiting for connections
  - Calls, in order, socket, bind, listen, accept
  - Then send and recv as necessary
- **Client side**
  - Connects to a waiting socket
  - Calls, in order, socket, connect
  - Then send and recv as necessary



Connection-Oriented Comm. Using Stream Sockets

Simplified Server Program

NOTE:

This example leaves out all error handling and parameter setup.

Realistic code would call **WSAGetLastError** many times.

```
00401041  push    ecx             ; lpWSAData
00401042  push    202h            ; wVersionRequested
00401047  mov     word ptr [esp+250h+name.sa_data], ax
0040104C  call    ds:WSAStartup
00401052  push    0               ; protocol
00401054  push    1               ; type
00401056  push    2               ; af
00401058  call    ds:socket
0040105E  push    10h             ; namelen
00401060  lea     edx, [esp+24Ch+name]
00401064  mov     ebx, eax
00401066  push    edx             ; name
00401067  push    ebx             ; s
00401068  call    ds:bind
0040106E  mov     esi, ds:listen
00401074  push    5               ; backlog
00401076  push    ebx             ; s
00401077  call    esi ; listen
00401079  lea     eax, [esp+248h+addrlen]
0040107D  push    eax             ; addrlen
0040107E  lea     ecx, [esp+24Ch+hostshort]
00401082  push    ecx             ; addr
00401083  push    ebx             ; s
00401084  call    ds:accept
```

The WinINet API

- Higher-level API than Winsock.
  - Its functions are stored in *Wininet.dll*.
  - Implements application layer protocols, such as HTTP and FTP.

- Can understand Malware based on the connections it opens.
  - *InternetOpen – connects to the Internet*
  - *InternetOpenURL –connects to a URL*
  - *InternetReadFile –reads data from a downloaded file*

Following Running Malware

Transferring Execution

- **jmp** and **call** transfer execution to another part of the code, but there are other ways
  - DLLs
  - Processes
  - Threads
  - Mutexes
  - Services
  - Component Object Model (COM)
  - Exceptions

DLLs (Dynamic link libraries)

- Share code among multiple applications
  - DLLs export code that other applications can use.

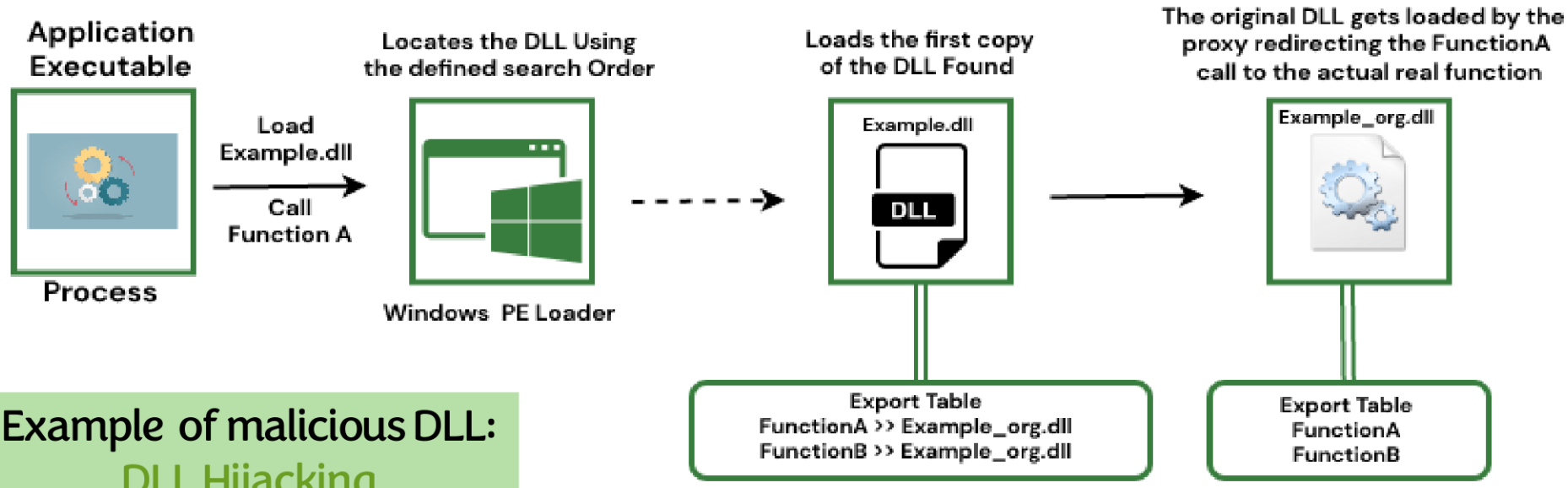
- Static libraries were used before DLLs
  - They still exist but are much less common
  - They cannot share memory among running Processes
  - Static libraries use more RAM than DLLs

Advantages of DLLs

- Using Windows DLLs makes code smaller.
  - Saves space, only loaded into memory once
- Software companies can also make custom DLLs
  - Distribute DLLs along with EXEs

How Malware Authors Use DLLs

- Store malicious code in DLL
  - Sometimes load malicious DLL into another process
- Using Windows DLLs
  - Nearly all malware uses basic Windows DLLs
- Using third-party DLLs
  - Use Firefox DLL to connect to a server instead of Windows API



Example of malicious DLL:
DLL Hijacking

Basic DLL Structure

- DLL files are very similar to *.exe* files.
  - Both uses the PE file format
  - A single flag indicates that the file is a DLL, not a *.exe*.
  - DLLs have more exports and fewer imports.

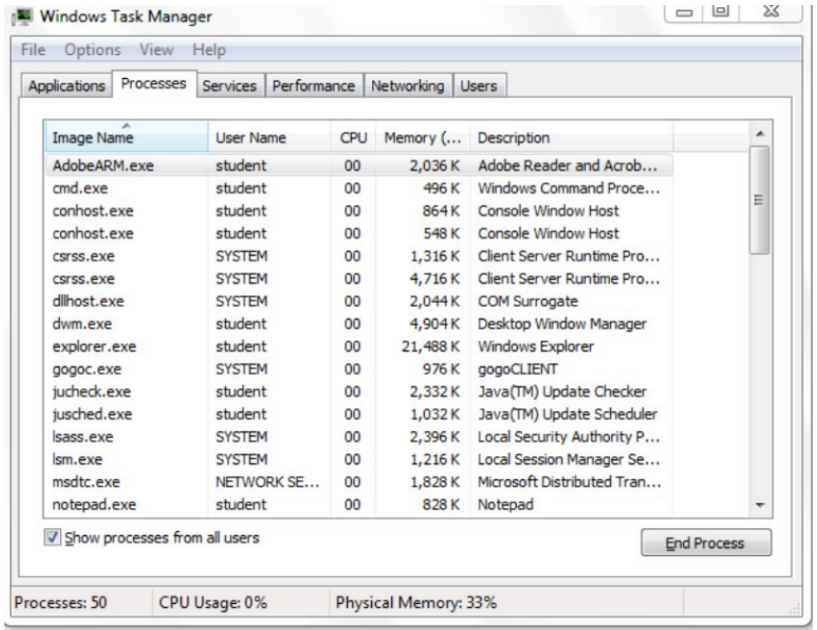
- The main DLL function is *DllMain*.
  - It has no label and is not exported,
  - But specified as the entry point in the PE Header.
  - Called whenever a function loads or unloads the library

Processes

- Every program being executed by Windows is a process
  - Each process has its resources: Handles, memory, thread(s),…

- Malware can also execute code outside the current program by:
  - Creating a new process or modifying an existing one.
  - Older malware ran as an independent process
  - Newer malware executes its code as part of another process

- Many Processes Run at Once.
  - All sharing the same resources.
    - CPU, file system, memory, & HW.
  - OS allows all processes to access resources without interfering



Processes

- OS allocates memory to each process.
- Two processes accessing the same memory address actually access different locations in RAM
  - Virtual address space
- Some Common process APIs:
  - CreateProcess, CreateProcessAsUser, EnumProcesses

Creating a New Process

The "CreateProcess" function

- Commonly used by malware to create a simple remote shell with just a single function call.
- This function has many parameters; one important parameter is the STARTUPINFO.
- STARTUPINFO parameter contains handles for standard I/O and standard error streams.
  - A malicious program can set these values to a socket allowing an attacker to execute a shell remotely by calling CreateProcess only.

Code to Create a Shell

Listing 7-4: Using CreateProcess to create a remote shell.

- Prior to this, code would have opened a socket to a remote location.
- Handle to the socket is stored on stack and entered into STARTUPINFO structure.
- Then CreateProcess is called, and all process's I/O is routed through the socket.

Listing 7-4: Sample code using CreateProcess call

```
004010DA   mov     eax, dword ptr [esp+58h+SocketHandle]
004010DE   lea     edx, [esp+58h+StartupInfo]
004010E2   push    ecx                 ; lpProcessInformation
004010E3   push    edx                 ; lpStartupInfo
004010E4 ❶mov      [esp+60h+StartupInfo.hStdError], eax
004010E8 ❷mov      [esp+60h+StartupInfo.hStdOutput], eax
004010EC ❸mov      [esp+60h+StartupInfo.hStdInput], eax
004010F0 ❹mov      eax, dword_403098
004010F5   push    0                   ; lpCurrentDirectory
004010F7   push    0                   ; lpEnvironment
004010F9   push    0                   ; dwCreationFlags
004010FB   mov     dword ptr [esp+6Ch+CommandLine], eax
004010FF   push    1                   ; bInheritHandles
00401101   push    0                   ; lpThreadAttributes
00401103   lea     eax, [esp+74h+CommandLine]
00401107   push    0                   ; lpProcessAttributes
00401109 ❺push     eax                 ; lpCommandLine
0040110A   push    0                   ; lpApplicationName
0040110C   mov     [esp+80h+StartupInfo.dwFlags], 101h
00401114 ❻call     ds:CreateProcessA
```

Code to Create a Shell

Example 8-4. Sample code using the CreateProcess call

```
004010DA  mov     eax, dword ptr [esp+58h+SocketHandle]
004010DE  lea     edx, [esp+58h+StartupInfo]
004010E2  push    ecx                   ; lpProcessInformation
004010E3  push    edx                   ; lpStartupInfo
004010E4 1mov     [esp+60h+StartupInfo.hStdError], eax
004010E8 2mov     [esp+60h+StartupInfo.hStdOutput], eax
004010EC 3mov     [esp+60h+StartupInfo.hStdInput], eax
004010F0 4mov     eax, dword_403098
004010F5  push    0                     ; lpCurrentDirectory
004010F7  push    0                     ; lpEnvironment
004010F9  push    0                     ; dwCreationFlags
004010FB  mov     dword ptr [esp+6Ch+CommandLine], eax
```

- Loads socket handle, StdError, StdOutput and StdInput into lpProcessInformation

Code to Create a Shell

```
004010FF  push    1                     ; bInheritHandles
00401101  push    0                     ; lpThreadAttributes
00401103  lea     eax, [esp+74h+CommandLine]
00401107  push    0                     ; lpProcessAttributes
00401109  push    eax                   ; lpCommandLine
0040110A  push    0                     ; lpApplicationName
0040110C  mov     [esp+80h+StartupInfo.dwFlags], 101h
00401114  call    ds:CreateProcessA
```

- CommandLine contains the command line
- It's executed when CreateProcess is called

Storing one program inside another

- Malware often creates a new process by storing one program inside another in the resource section.

- When the program runs:
  - it will extract the additional executable from the PE header,
  - write it to disk, and then
  - Call CreateProcess to run the program.
  - This is also done with DLLs and other executable code.

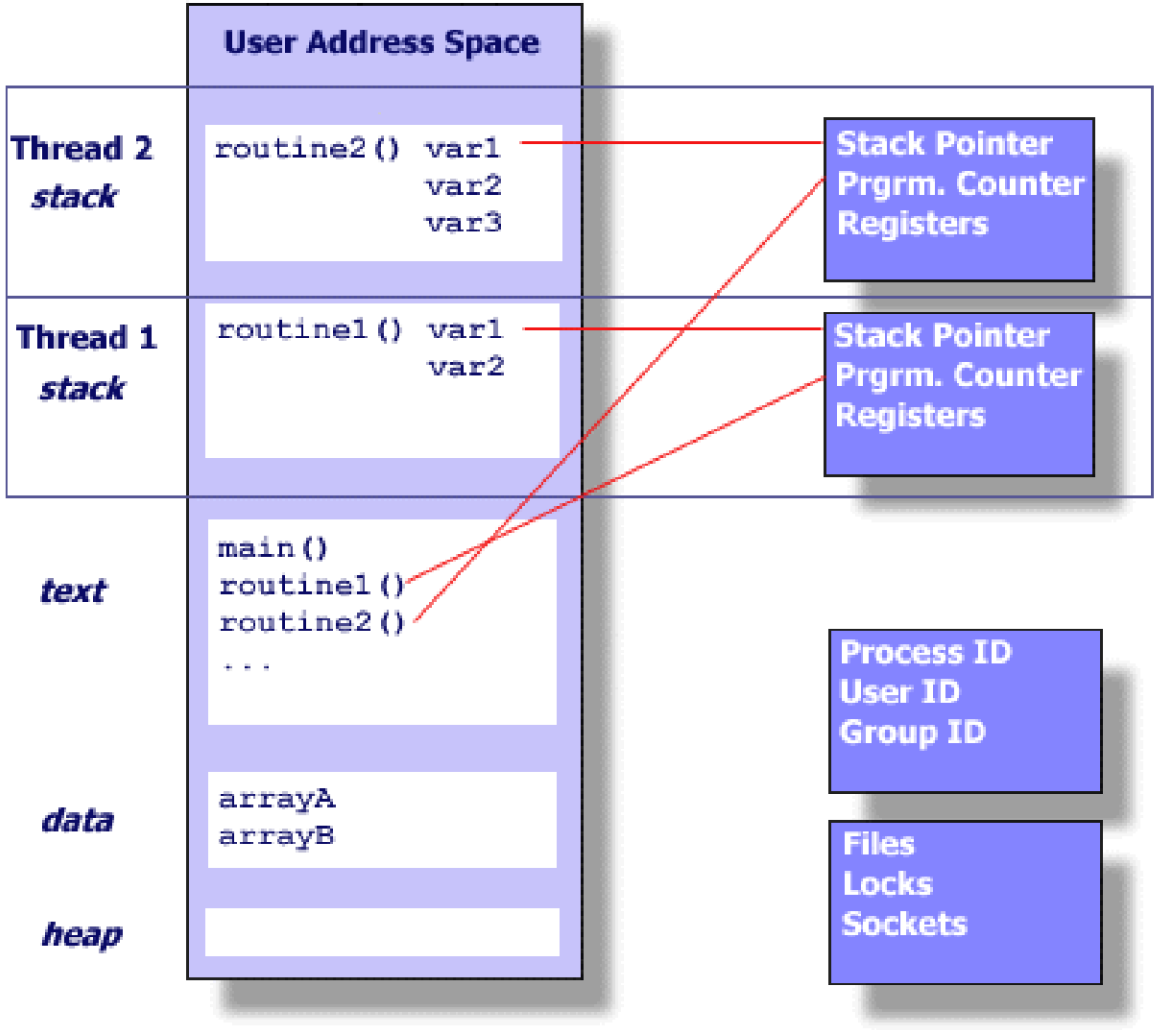
- When this happens:
  - you must open the program in the Resource Hacker utility and
  - Save the embedded executable file to disk in order to analyze it.

Threads

- **Processes** are the container for execution, but *threads* are what the Windows OS executes.
  - A process contains one or more threads, which execute part of the code within a process.
- **Threads** are independent sequences of instructions.
  - Executed by CPU without waiting for other threads
  - Threads within a process share the same memory space
  - Each thread has its own registers and stack
  - Each thread belongs to a single process
  - Scheduled and executed by the OS
  - Have their own thread context and stack
  - Common APIs: CreateThread, CreateRemoteThread

Thread Context

- Keeps track of the state of a thread
  - Necessary when there are multiple threads on a system
  - State is defined by register values
- Some API: GetThreadContext, SetThreadContext



Thread Context

- When a thread is running, it has complete control of the CPU
- Other threads cannot affect the state of the CPU
- When a thread changes a register, it does not affect any other threads
- When the OS switches to another thread, it saves all CPU values in a structure called the **thread context**

Creating a Thread

- **CreateThread**
  - Caller specified a **start** address, also called a **start** function

- **How Malware Uses Threads**
  - Use **CreateThread** to load a malicious DLL into a process
  - Create two threads for input and output
    - Used to communicate with a running application

Mutexes: Interprocess Coordination

- Mutexes are global objects that coordinate multiple processes and threads
  - In the kernel, they are called mutants
  - Mutexes often use hard-coded names, which can be used to identify malware
  - Only one thread can own a mutex at a time.

Functions for Mutexes

- WaitForSingleObject
  - Gives thread access to the mutex
  - Any subsequent threads attempting to gain access to it must wait
- ReleaseMutex
  - Called when a thread is done using the mutex
- CreateMutex
  - A function to create a new mutex
- OpenMutex
  - Gets a handle on another process's mutex

How malware uses Mutex

Making Sure Only One Copy of Malware is Running

- Malware will commonly create a mutex and attempt to open an existing mutex with the same name to ensure that only one version of the malware is running at a time

- **OpenMutex** checks if HGL345 exists

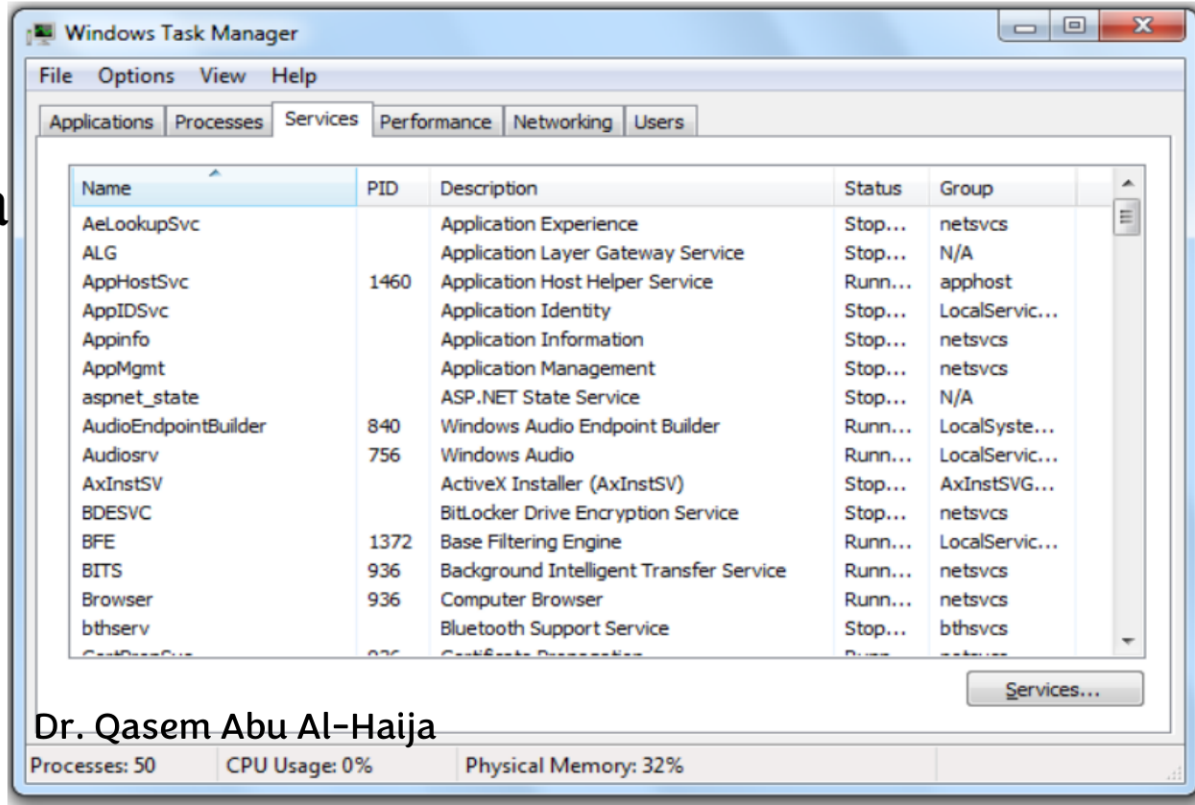
- If not, it is created with **CreateMutex**

- **test eax, eax** sets Z flag if eax is zero

```
00401007   push  1F0001h          ; dwDesiredAccess
0040100C  1call  ds:__imp__OpenMutexW@12 ;
OpenMutexW(x,x,x)
00401012  2test  eax, eax
00401014  3jz    short loc_40101E
00401016   push  0                ; int
00401018  4call  ds:__imp__exit
0040101E   push  offset Name      ; "HGL345"
00401023   push  0                ; bInitialOwner
00401025   push  0                ; lpMutexAttributes
00401027  5call  ds:__imp__CreateMutexW@12 ;
CreateMutexW(x,x,x)
```

Services

- Malware can execute code by installing it as a service.
- Services run in the background without user input.
  - Tasks run without their own processes or threads
  - Code is scheduled and run by Win. service manager without user input
  - Similar to a process
- Can interact with them via Service Manager (services.exe)
  - Start, stop, suspend, schedule, autostart



SYSTEM Account

- Services run as SYSTEM (more powerful than the Administrator)
  - Therefore, its convenient for malware writers

- Services can run automatically when Windows starts
  - May not even show up in the Task Manager as a process.
  - An easy way for malware to maintain persistence
  - Persistent malware survives a restart

- Searching the running applications wouldn't find any suspicious,
  - because the malware isn't running in a separate process.

Service API Functions

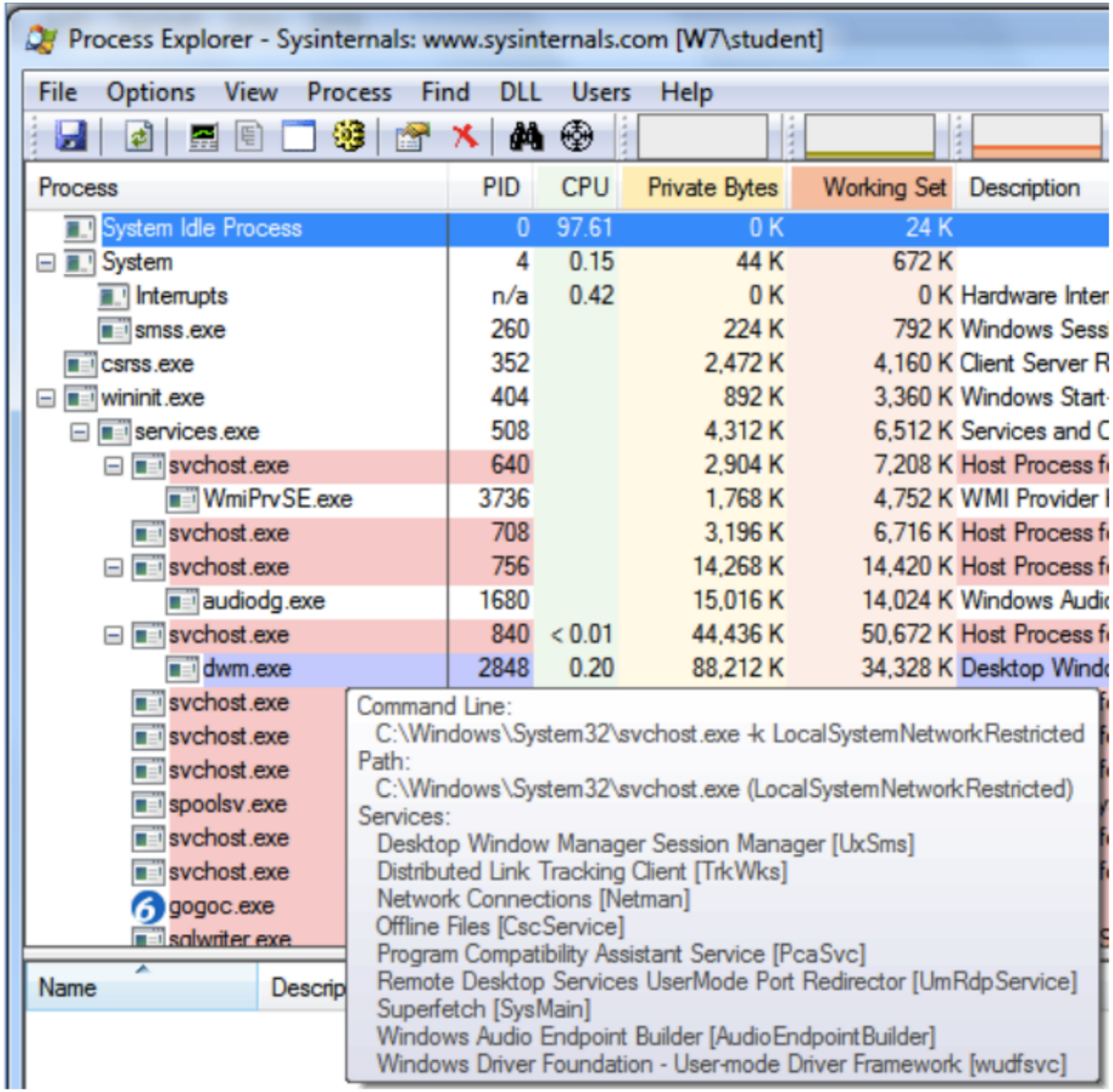
- OpenSCManager
  - Returns a handle to the Service Control Manager
- CreateService
  - Adds a new service to the Service ControlManager
  - Can specify whether the service will start automatically at boot time
- StartService
  - Only used if the service is set to start manually

Common Service Type

- WIN32_SHARE_PROCESS
  - Most common type of service used by malware
  - Stores code for service in a DLL
  - Combines several services into a single shared process, 'svchost.exe.'
  - In Task Manager: several instances of svchost.exe process, which is running WIN32_SHARE_PROCESS-type services.

- Other Common Service Types
  - WIN32_OWN_PROCESS: Runs as an EXE in an independent process
  - KERNEL_DRIVER: Used to load code into the Kernel

Svchost.exe in Process Explorer



Service Information in the Registry

- Info. about services on a local system is stored in the registry.
- Each service has a subkey under HKLM\SYSTEM\CurrentControlSet\Services.
- For example, Figure 7-2 shows the registry entries for: HKLM\SYSTEM\CurrentControlSet\Services\VMware NAT Service.

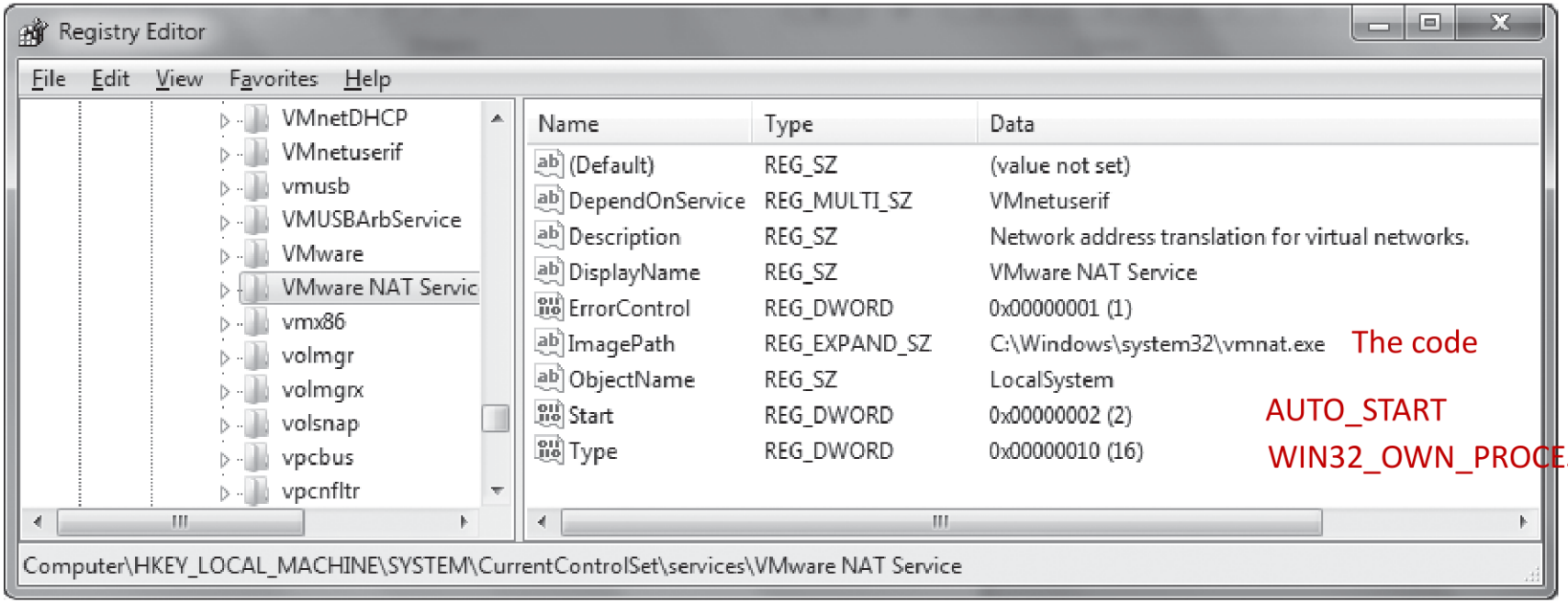
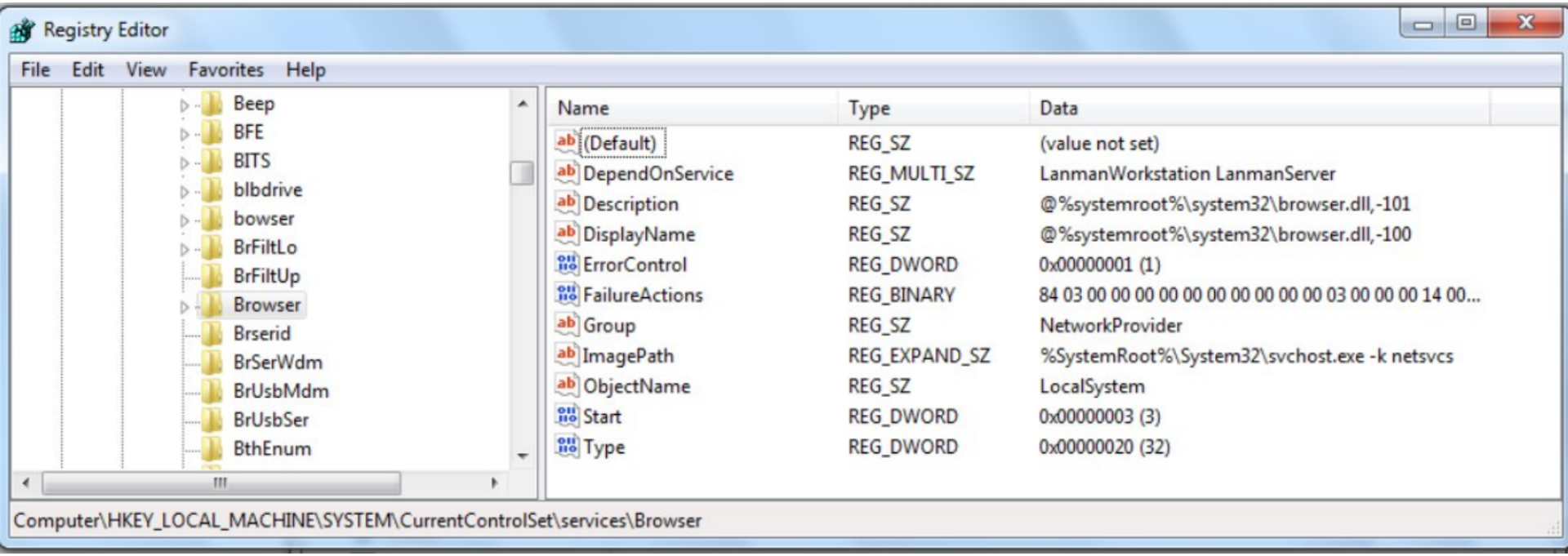


Figure 7-2: Registry entry for VMware NAT service

Service Information in the Registry

- Another example: "Browser" Service
  - HKLM\System\CurrentControlSet\Services\Browser
  - Start value = 0x03 for "Load on Demand"
  - Type = 0x20 for WIN32_SHARE_PROCESS



SC Command

- sc query of services
  - Included in Windows, More readable
  - Gives information about Services

```
C:\Users\User1>sc qc "VMware NAT Service"
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: VMware NAT Service
        TYPE               : 10  ❶WIN32_OWN_PROCESS
        START_TYPE         : 2     AUTO_START
        ERROR_CONTROL      : 1     NORMAL
        BINARY_PATH_NAME   : C:\Windows\system32\vmnat.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : VMware NAT Service
        DEPENDENCIES       : VMnetuserif
        SERVICE_START_NAME : LocalSystem
```

Listing 7-10: The query configuration information command of the SC program

Component Object Model (COM)

- An interface standard allows different SW components to share code
  - without knowledge of specifics about each other.

- Every thread that uses COM must call OleInitialize or CoInitializeEx before calling other COM libraries

GUIDs, CLSIDs, IIDs

- COM objects are accessed via Globally Unique Identifiers (GUIDs)

- There are several types of GUIDs, including
  - Class Identifiers (CLSIDs)
    - in Registry at HKEY_CLASSES_ROOT\CLSID
  - Interface Identifiers (IIDs)
    - in Registry at HKEY_CLASSES_ROOT\Interface

Exceptions: When Things Go Wrong

- Exceptions allow a program to handle events outside the flow of normal execution.
  - Exceptions are caused by errors raised by HW, such as division by zero, or by OS, such as invalid memory access.
  - Exception can also be raised explicitly in code with the RaiseException call.

- When an exception occurs, execution transfers to the Structured Exception Handler (SEH)
  - special routine that resolves the exception.

Exceptions: When Things Go Wrong

Example 8-13. Storing exception-handling information in fs:0

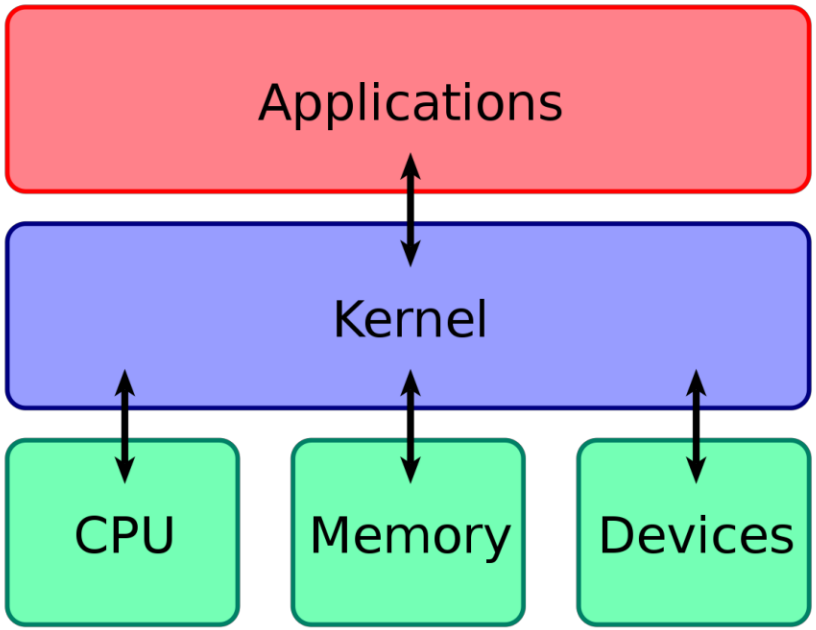
```
01006170  push  1offset loc_10061C0
01006175  mov     eax, large fs:0
0100617B  push  2eax
0100617C  mov     large fs:0, esp
```

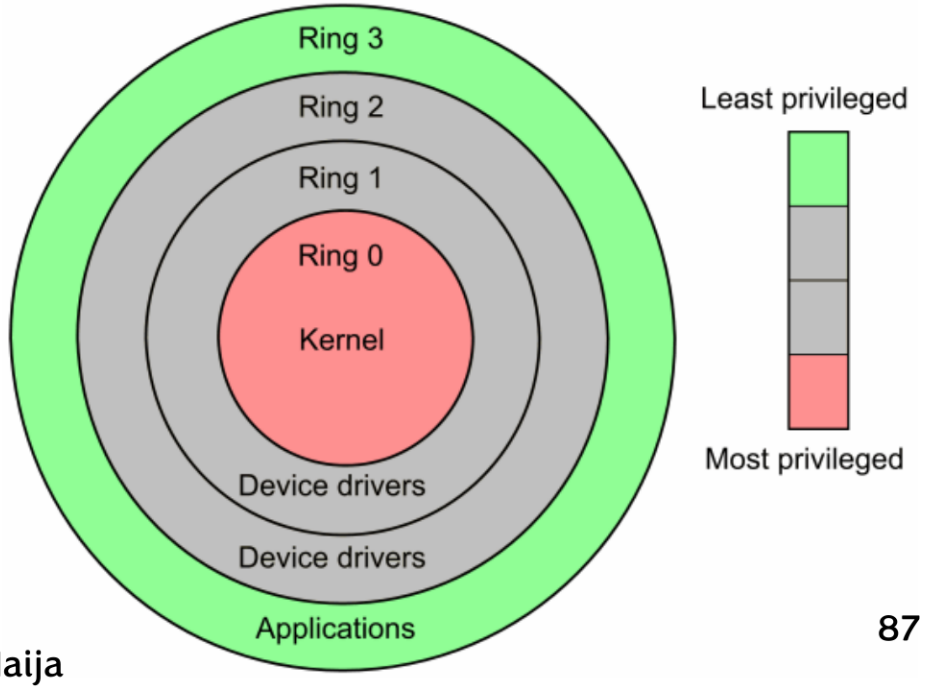
- fs is one of six Segment Registers
- fs:O Stores Exception Location

Kernel vs. User Mode

Kernel vs. User Mode

- Windows uses two processor privilege levels:
  - kernel mode (ring 0) and user mode (ring 3).
  - Rings 1 & 2 are not used by Windows.
  - All functions discussed so far: user-mode functions





User Mode

- Nearly all code runs in user mode
  - Except OS and hardware drivers, which run in kernel mode
- User mode cannot access hardware directly
  - Restricted to a subset of CPU instructions
  - Can only manipulate HW through Windows API

User Mode

- In user mode, each process has its own resources (memory, security permissions, ...).
  - If a user-mode program executes an invalid instruction and crashes, Windows can reclaim the resources and terminate the program.

- It's not possible to jump directly from user mode to the kernel
  - **SYSENTER**, **SYSCALL**, or **INT 0x2E** instructions use lookup tables to locate predefined functions.

Kernel Mode

- All kernel processes share resources and memory addresses (has Fewer security checks)
- If kernel code executes an invalid instruction, the OS crashes with the Blue Screen of Death
- Antivirus software and firewalls run in Kernel mode

Kernel Mode

- kernel-mode malware is more powerful than user-mode malware
- Most malware does not use kernel mode
- OS Auditing feature doesn't apply to the kernel
- Almost all rootkits use kernel code

Native API

Native API

- Lower-level interface for interacting with Windows
- Rarely used by nonmalicious programs
- Popular among malware writers
- Undocumented
- Intended for internal Windows use
- Can be used by programs
- Native API calls can be more powerful and stealthier than Windows API calls

Native API

- Ntdll.dll manages interactions between user space and kernel
- Ntdll functions make up the Native API

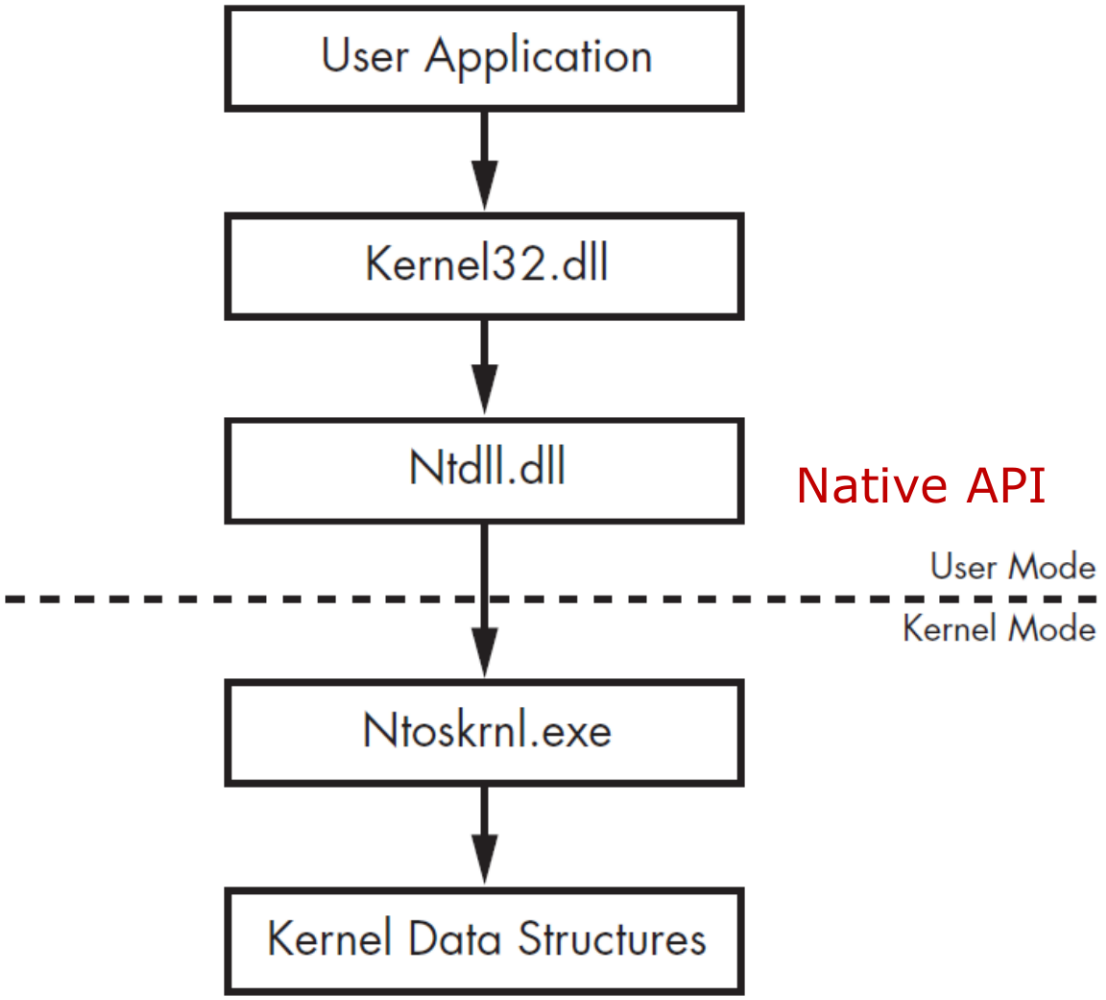


Figure 7-3: User mode and kernel mode

Bypassing Security Monitoring Program

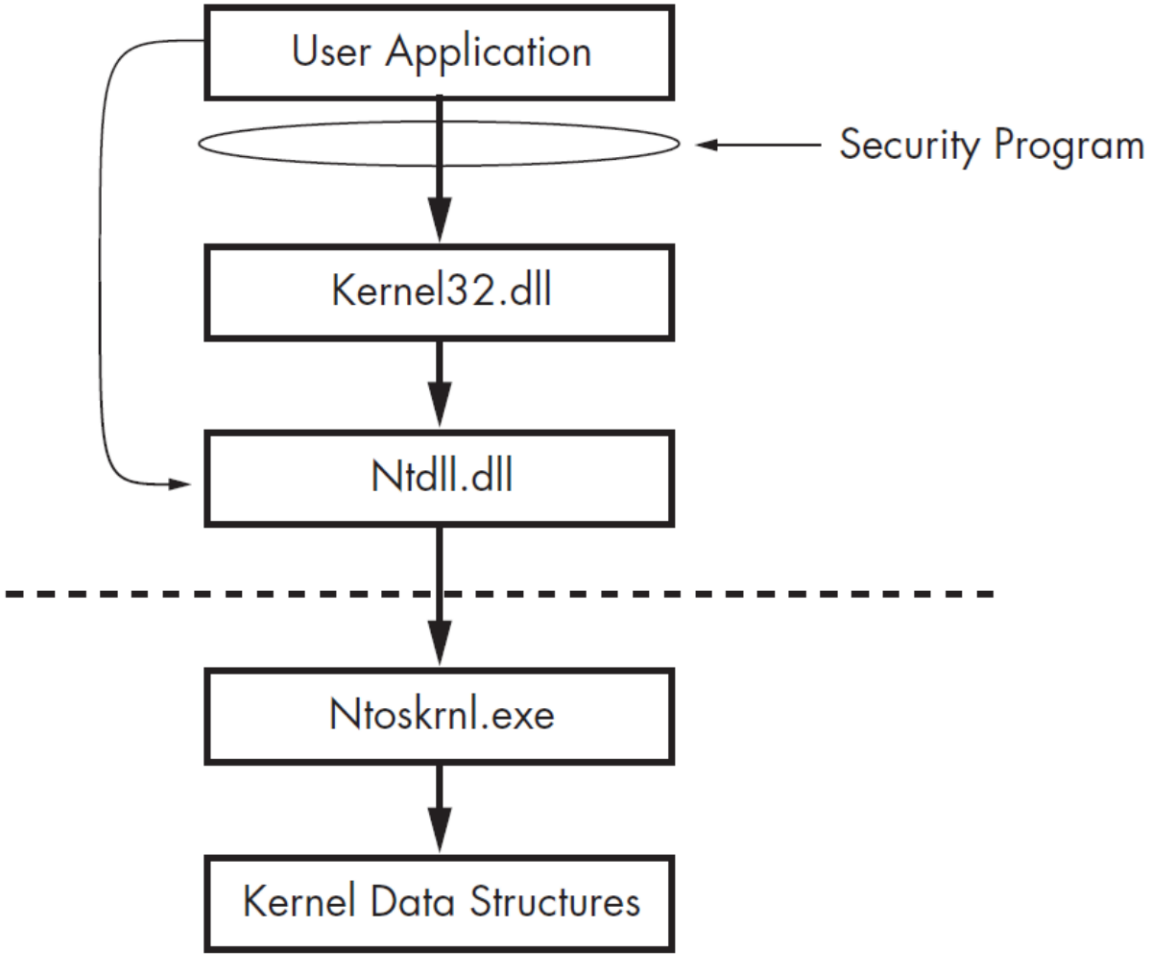


Figure 7-4: Using the Native API to avoid detection

Popular Native API Calls in Malware

- NTtQuerySystemInformation
- NTtQueryInformationProcess
- NTtQueryInformationThread
- NTtQueryInformationFile
- NTtQueryInformationKey
- NtContinue

Main Sources for these slides

- *Michael Sikorski and Andrew Honig, "Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software"; ISBN-10: 1593272901.*
- *Xinwen Fu, "Introduction to Malware Analysis," University of Central Florida*
- *Sam Bowne, "Practical Malware Analysis," City College San Francisco*
- *Abhijit Mohanta and Anoop Saldanha, "Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware," ISBN: 1484261925.*

Thank you