

# Lecture Outline

- Memory handling
- The registry
- Windows boot
- Windows architecture
  - systems and subsystem details
  - PE files
    - exe and dll
- Windows security
- Windows domain
- Rootkits

Today's talk is  
mostly going to  
be Win7-centric

# Books used to build this lecture

1. Windows Internals 6 (part 1 and 2)
2. The Rootkit ARSENAL: Escape and Evasion in the Dark Corners of the System (2nd edition)
3. Designing BSD Rootkits
4. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software
5. Gray Hat Python: Python Programming for Hackers and Reverse Engineers

# Memory Handling

- Virtual Address Space
  - Each process gets one
  - Not shared between processes (by default)
  - Partition for the process and system

How are partitions enforced?

# Segmentation & Paging in Windows

Boundary between the kernel (OS) and user applications relies hardware mechanisms

- Intel 32 based processors (and variants) implements memory protection through both segmentation and paging
  - Windows uses both, but paging more often.

# Paging

**Paging:** memory is divided into small partitions (all the same size), and are referred to as page frames. When a process is loaded, it gets divided into pages --- these pages are loaded into the frames

- **Transparent to programmer** (system allocates memory)
- No separate protection
- No separate compiling
- No shared code

# Segmentation

**Segmentation:** memory is allocated in various sizes (segments), depending on the need for address space of a process. Segments can be individually protected, or shared between processes. *Segmentation Faults* occur when data that is about to be read or written is outside of the permitted address space for that process.

- Involves programmer (allocates memory to specific function inside code)
- Separate compiling
- Separate protection
- Share code

# Windows NT family

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Home Server
- Windows Server 2008
- Windows 7
- Windows 8
- Windows Phone 8
- Windows Server 2012

Then there are 32 bit vs 64 bit, home/professional/enterprise/etc flavors...

\*all use:

- ntdll.dll
- win32api
- Windows (NT) Domains
- Workgroups
- Active Directory

lots of minor system filename changes between 32bit and 64 bit

# Windows

- Multi-tiered design
- Mostly implemented in C / C++ / asm
- Can isolate chipset dependencies by using Hardware Abstraction Layers (HAL)
- Designed to support many architectures
  - MIPS, ARM, x86, ...
- **LOTS OF BACKWARDS COMPATIBILITY**



Look we still make all the old stuff work  
to keep our customers happy!  
Also lets not bother keeping the old stuff up  
to date!

# Backwards Compatibility

**IS THE BANE OF SECURITY**

**It's really quite a horrible problem**

# Examples of Backwards compatibility issues

The following things at some point were a security issue across versions...

- NTLMv1 password hashes
- WMF Exploits
  - (Metafile Image Code Execution AKA MICE)
- PE file format exploits
- CA-2002-03 ([SNMP](#))
- XBox One Bricking prank
- CVE-2014-6332 (19 years!)
  - IE11 exploit + Godmode
- JASBUG: MS15-011 (15 years!)
  - The fix required Microsoft to re-engineer core components of the operating system and to add several new features

# Main reasons for windows having so many security "problems"

1. marketshare
2. Massive size of Windows code
3. Hundreds of drivers (most are 3rd party!)
4. It has a lot of backwards compatibility, user friendliness, autorun, and "plug and play" features that make it target rich
5. Historically, it has a lot of MS products that are tied into kernel functions (outlook, IE, word, etc) to make them load faster than the competition

# **The Windows Overview**



# The registry

Basically a database for info and config for everything.

- regedit.exe

The 5 hives:

- HKEY\_CLASSES\_ROOT
- HKEY\_CURRENT\_USER
- HKEY\_LOCAL\_MACHINE
- HKEY\_USERS
- HKEY\_CURRENT\_CONFIG

# The registry

Works with:

- Access Control
- Group Policy
- Local Security Settings
- Permissions
- and User Rights Assignment

To define everything the computer and the user can and cannot do

# The registry

There are 5 main root keys in the hive categories

- visible by default
- Are associative arrays and abstract data types that can hold collections of objects
  - They contain subkeys, which contain values (variables)

# The hives

- HKEY\_CLASSES\_ROOT
  - Contains file type associations
- HKEY\_CURRENT\_USER
  - Contains preferences and settings of the currently logged on user
    - supporting files: Ntuser.dat, Ntuser.dat.log



# The hives

- HKEY\_LOCAL\_MACHINE
  - PnP and HAL info is gathered here about the system's hardware
  - contains software, hardware, and security info
  - Also pulls info from the 4 other hives:
    - System
    - Software
    - Security
    - SAM
  - is one of the most major hive structures

# The hives

- HKEY\_LOCAL\_MACHINE (HKLM)
  - supporting files:
    - HKLM \SAM: *Sam, Sam.log, Sam.sav*
    - HKLM \Security: *Security, Security.log, Security.sav*
    - HKLM \Software: *Software, Software.log, Software.sav*
    - HKLM \System: *System, System.alt, System.log, System.sav*
  - *all are stored in %System Root%\System32\config*
    - *stores all registry files*
    - *usually is C:\Windows\System32\config*

# The hives

- HKEY\_USERS

- Contains data from every user in the SAM
  - contains info for that user's:
    - desktop
    - environment
    - program settings
    - network connections
    - printers

- HKEY\_CURRENT\_CONFIG

- contains PnP data about system's hardware devices that are used in the loading/startup process

# Registry notes

- Each time a user logs on, a new hive ("user profile hive") is dynamically built for that user
  - located under HKEY\_USERS
- Is dynamically created each time the system is booted

# The Windows Boot

1. Post *16 bit real mode*
2. CMOS
  - UEFI / EFI will load other firmware
3. MBR - points to bootmgr - the windows boot manager
4. Bootmgr - loads and reads the Boot Configuration Data (BCD) file/store
5. BCD Store - reads which OSes are specified in the BCD store, and displays a menu to select which one

# The Windows Boot

6. bootmgr resumes - loads Winload.exe, the windows boot loader
7. Winload.exe - *protected* & *real modes*
  - loads the kernel (ntoskrnl.exe), and loads HAL.dll into memory.
  - Then loads the SYSTEM registry hive
8. These processes are used to create registry key HKEY\_LOCAL\_MACHINE\SYSTEM
9. Winload uses the HKLM\SYSTEM key to load device drivers into memory (**without starting them**)

# The Windows Boot

10. Winload checks if user wants to start using Last Known Good Configuration (pressing F8 key)
11. Winload starts:
  - memory paging (pagefile.sys) and
  - startup control passes to the ntoskrnl.exe (the windows kernel)
12. ntoskrnl.exe - causes the HAL to become active
  - builds HKEY\_LOCAL\_MACHINE\HARDWARE from info collected thusfar
13. ntoskrnl.exe starts critical services and drivers
  - located in C:\Windows\System32\Drivers

# The Windows Boot

## 14. ntoskrnl.exe starts **smss.exe** (Session Manager SubSystem)

- responsible for handing sessions running on a machine
- starts the **kernel** and **user** modes of the Win32 subsystem **protected mode**
  - win32k.sys (kernel mode)
  - winsrv.dll and csrss.exe (both user mode)
- starts any subsystems listed with the "Required" value in the following registry key:
  - HKLM\System\CurrentControlSet\Control\Session Manager\Subsystems
- creates environment variables, virtual memory paging files
- *smss.exe = historically common target for malware*
  - *first native application in boot/startup*



# The Windows Boot

15. smss.exe starts the Win32 graphics subsystem
16. smss.exe starts **csrss.exe** (Client Server Runtime SubSystem)
  - provides the user mode side of the Win32 subsystem
  - console handling and GUI shutdown
  - the second **native application**
17. smss.exe starts **Winlogon.exe** (the logon manager)
18. Winlogon.exe starts **services.exe** (Service Control Manager)

# The Windows Boot

19. Winlogon.exe starts lsass.exe (Local Security Authority Process)
  - a. displays the logon screen, prompting for user id and password.
  - b. handles authentication
20. Winlogon.exe executes userinit.exe
21. Userinit.exe
  - a. applies Group Policy settings and startup and policy settings
    - i. in the local user registry
    - ii. not overridden by the Active Directory Group Policy

# The Windows Boot

22. Winlogon launches **Explorer.exe**, the windows graphical Window Manager and shell

Whew thats a lot  
that happens!



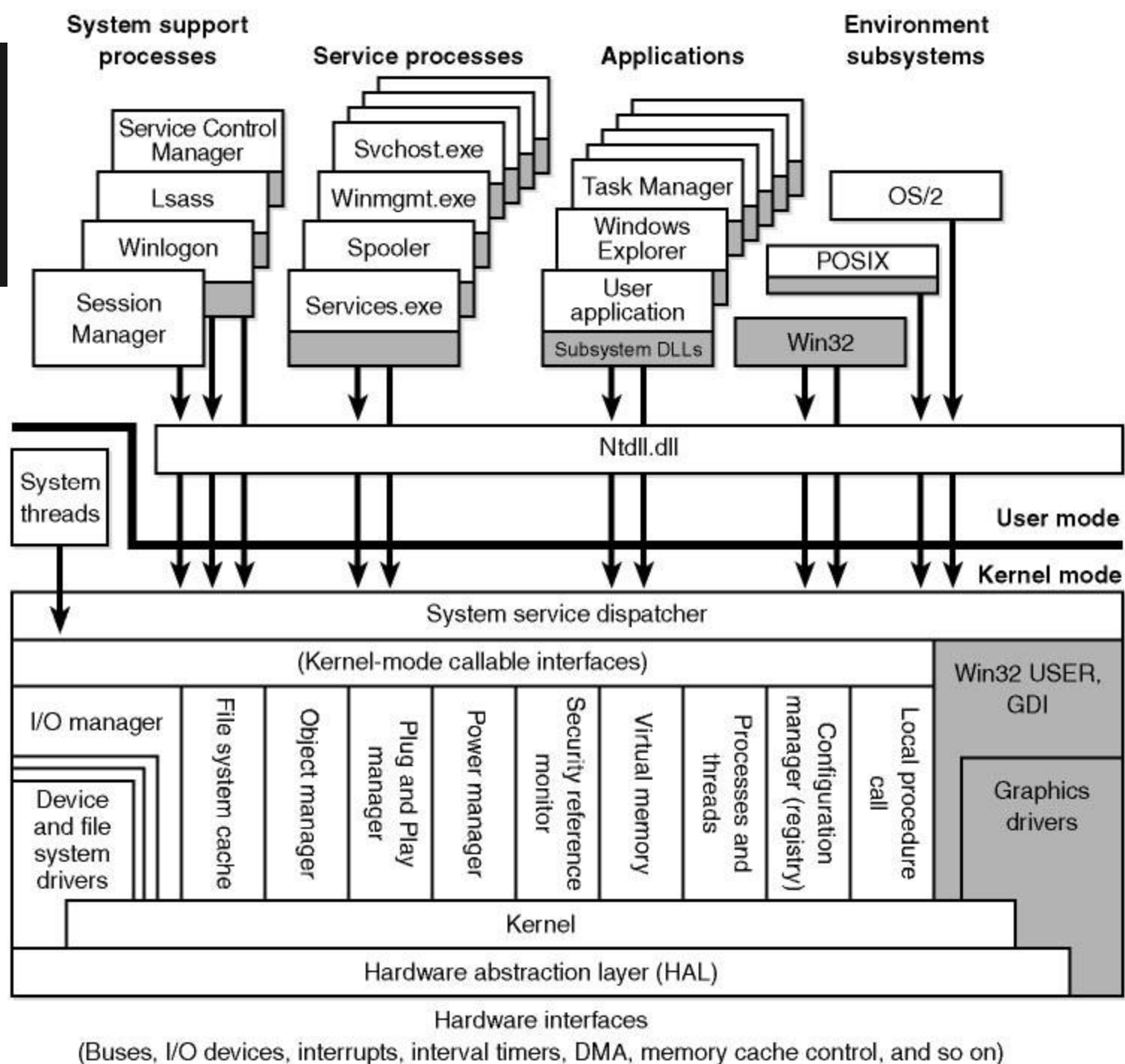
# Subsystem Startup

Subsystems are started by the Session Manager (Smss.exe) process

- Smss information is stored at:

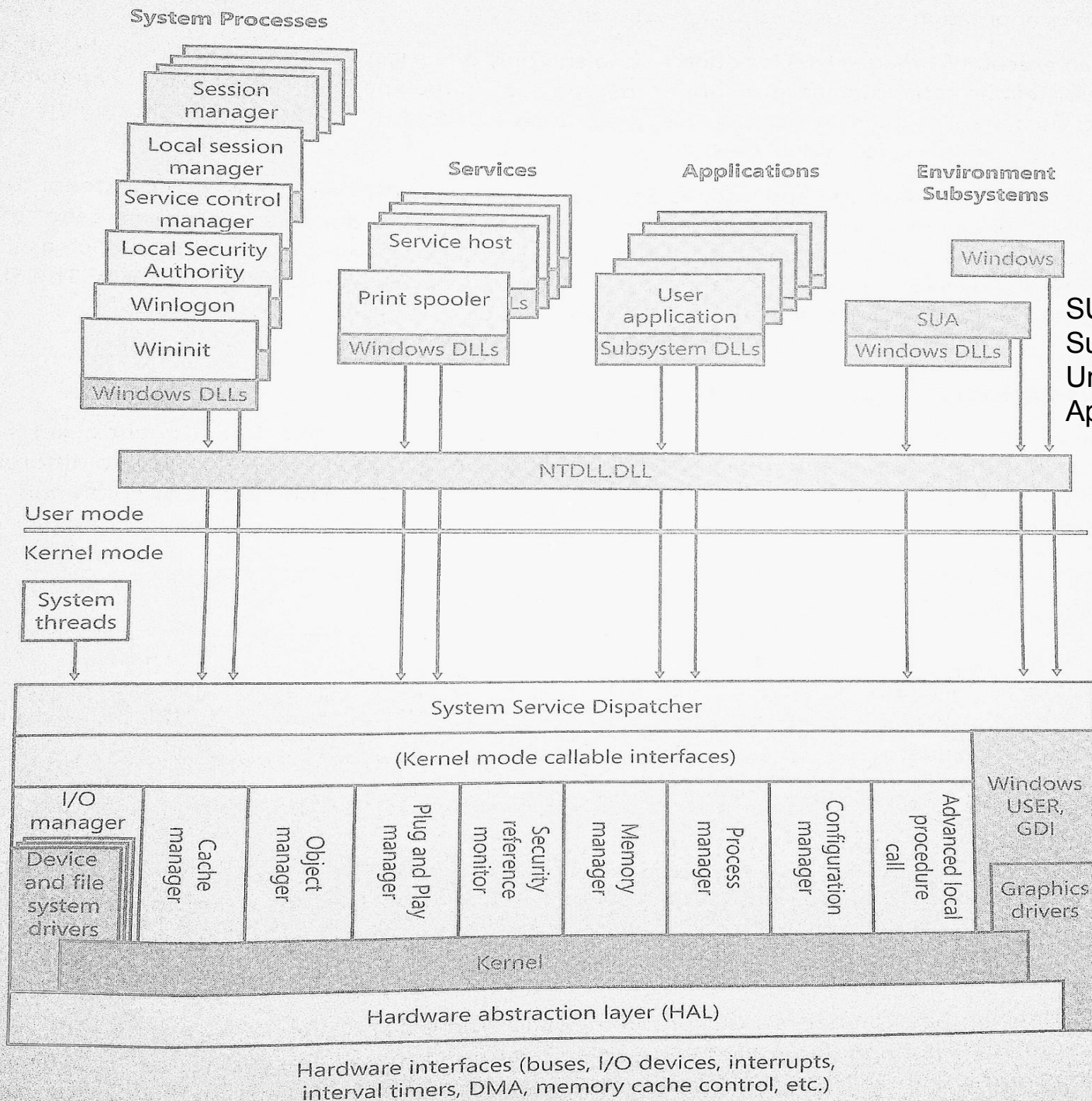
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Subsystems

- starts any subsystems listed with the "Required" value in the following registry key



# WINDOWS 7 / WINDOWS VISTA

Source: *Windows Internals 6th edition, Part 1*



# Win32 API & native API

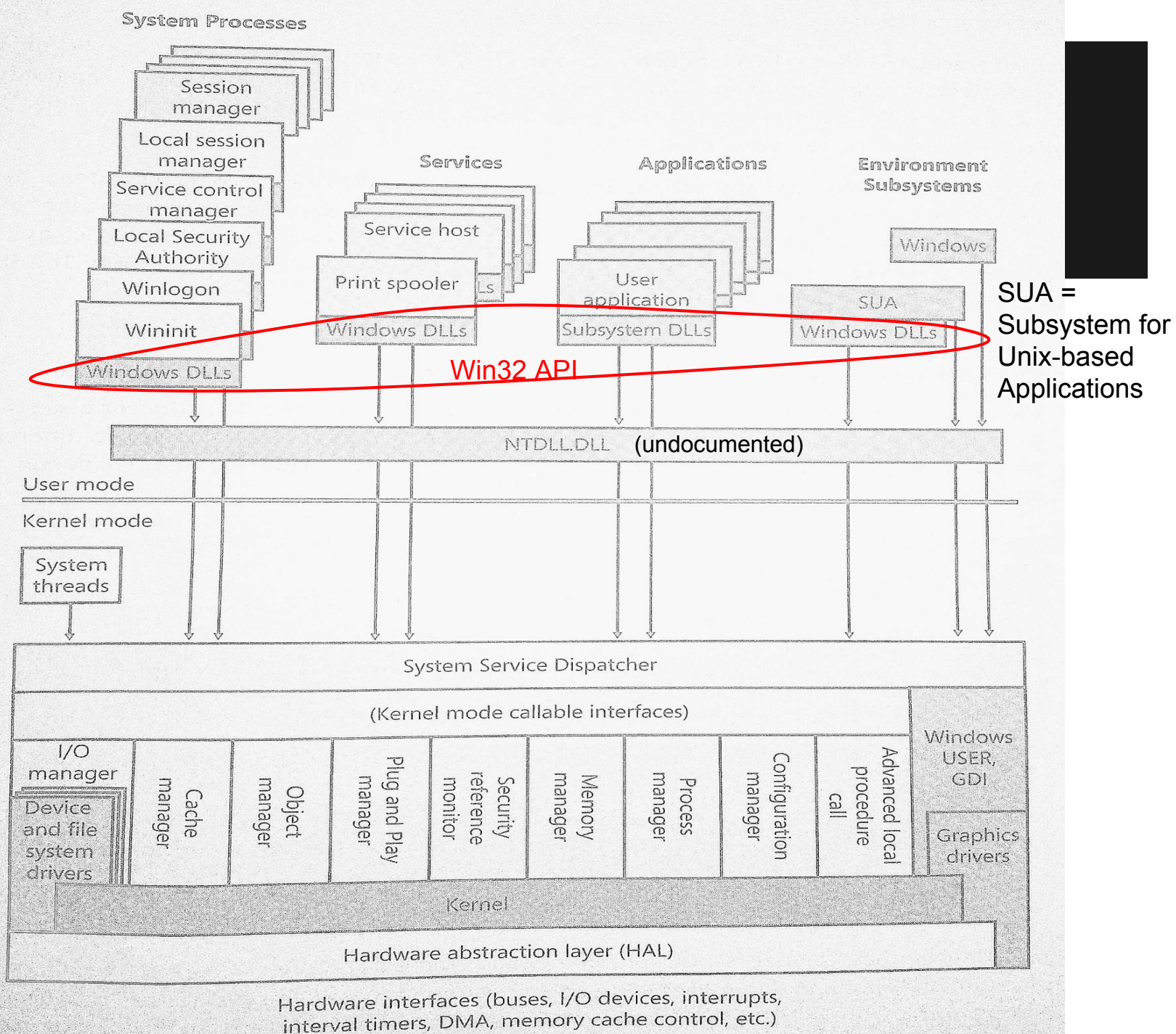
Windows has a layered architecture

- in order to program applications on it's OS, Windows provides the **Win32 API**:
  - **These are the subsystem DLL's**
  - Are well documented, and call the native API
- In order to communicate with the kernel, windows provides a native API (located usually in ntdll.dll)
  - ntdll.dll functions have no documentation
    - Window's famous hidden API



# WINDOWS 7 / WINDOWS VISTA

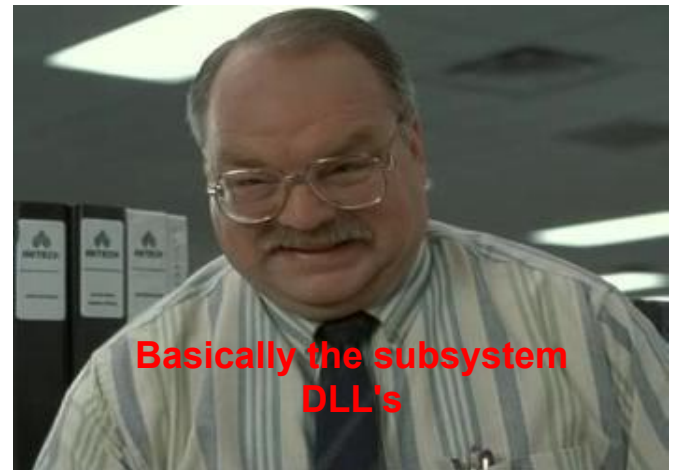
Source: Windows Internals 6th edition, Part 1





# Win32 API & native API

*The role of the subsystem DLL's is to translate a documented function into the appropriate internal (and mostly undocumented) native system calls (for security in depth)*



# Win32 API breakdown

Functions fall in 8 categories:

## 1. Base Services

- a. file system, devices, processes, threads, err handling

## 2. Advanced Services

- a. registry, shutdown, service management, user account management (advapi32.dll)

## 3. Graphics Device Interface

- a. monitor output, printers, and etc

## 4. User Interface

- a. windows, buttons, menus, scrollbars, etc...

Source:

[http://en.wikipedia.org/wiki/Windows\\_API](http://en.wikipedia.org/wiki/Windows_API)

# Win32 API breakdown

5. Common Control Library
  - status bars, progress bars, toolbars, tabs..
6. Common Dialog box library
  - dialog boxes and fonts...
7. Windows Shell
  - os system shell (shell32.dll, shlwapi.dll)
8. Network Services
  - networking, NetBIOS, Winsock, NetDDE, RPC (netapi32.dll)

Source:

[http://en.wikipedia.org/wiki/Windows\\_API](http://en.wikipedia.org/wiki/Windows_API)

# Win32 API files

- **Kernel32.dll**
  - exports most of the wind32 base APIs, such as memory management, I/O, process and thread creation, & synchronization functions
- **GDI32.dll**
  - exports Graphics Device Interface functions
- **User32.dll**
  - exports functions that create/manipulate the Windows Users Interface
- **Comctl32.dll**
  - File open / save / save as / progress pars, etc...

# ntdll.dll

- mostly undocumented (by Microsoft)
  - historically "well" documented on the underground
- majority of applications do not call ntdll.dll
- Applications that link/call ntdll.dll directly are called **native applications**
  - cannot be run by a User
- vast majority of **exported functions (aka symbols)** are prefixed with **"NT"**
  - NtDisplayString, NtReadFile, NtWriteFile
  - ReadFile calls NtReadFile
  - WriteFile calls NtWriteFile

# Exported Functions

- How API's are provided...
- EXEs and DLLs export functions to interact with other programs and code
- most common in DLLs, rare in EXEs
  - malware that exports functions will use intentionally misleading function names

# PE files / Importing functions

## Portable Executable

- .cpl, **.exe**, .dll, .ocx, .sys, .scr, .drv
- Contains info as to what functions are imported / exported in its headers (call tables)
- When reverse engineering PE's, looking at the imported functions is informative
- Can play tricks and import functions by indirect reference
  - common in malware
  - i.e. import the 6th function in winsrv.dll

# PE files / Importing functions

When applications are compiled,

- API calls do not use hardcoded addresses
  - Work through a function pointer
  - Organized in a table
    - **The Import Address Table (IAT)**
  - This way each API call doesn't have to be touched up by the compiler with the correct jump address for the API code
  - packed executables mess with the process IAT table to make it smaller (we'll cover executable packing later :)



# Interesting functions malware may often use

From Kernel32.dll

CreateFileW	opening and manipulating files
FindFirstFileW	for traversing directories
FindNextFileW	for traversing directories
GetCurrentProcess	for open and manipulating processes
GetProcessHeap	for open and manipulating processes
OpenProcess	for open and manipulating processes
ReadFile	opening and manipulating files
WriteFile	opening and manipulating files

# Interesting functions malware may often use

from User32.dll (user environment / GUI)

RegisterClassExW	Gui Creation/Manipulation (does not have to appear for user)
RegisterHotKey	Whenever a hotkey is pressed (even in the focus of other applications), this application will be notified (i.e. CTRL-SHIFT-P)
SetWindowTextW	Gui Creation/Manipulation (does not have to appear for user)
SetWindowsHookExW	Commonly used by malware to hook things (i.e. for keylogging)
ShowWindow	Gui Creation/Manipulation (does not have to appear for user)

# Interesting functions malware may often use

from Advapi32.dll (for manipulating the registry)

RegCloseKey	for viewing / changing the registry
RegDeleteValueW	for viewing / changing the registry
RegOpenCurrentUser	for viewing / changing the registry
RegOpenKeyExW	for viewing / changing the registry
RegQueryValueExW	for viewing / changing the registry
RegSetValueExW	for viewing / changing the registry

# Interesting functions malware may often use

Shell32.dll (for launching other applications)

CommandLineToArgvW	Parses a Unicode command line string and returns an array of pointers to the command line arguments, along with a count of such arguments, in a way that is similar to the standard C run-time <i>argv</i> and <i>argc</i> values.
SHChangeNotify	Notifies the system of an event that an application has performed. An application should use this function if it performs an action that may affect the Shell.
ShellExecuteExW	Performs an operation on a specified file.
ShellExecuteW	Performs an operation on a specified file.

# Security in Win 7


Data Execution Prevention (DEP)

Address Space Layout Randomization (ASLR)

/GS Compiler option and /SAFESEH linker option

Kernel Patch Protection

Enhanced mitigation Experience Toolkit (EMET)



We will  
cover later  
on

## ● User Account Control (UAC)

- These are the consent command prompts
- All users (including administrators) run as Standard user by default (can still download malware)
  - Prompt for administrator access
- From Vista: vast reduction in number of OS applications that require elevated privileges

# Security in Win 7

- SAM ([local] Security Accounts Manager)
  - stores the users' passwords in a hashed format
  - in %System Root%\System32\config\
    - LM hash and NTLM hash
    - Windows kernel has exclusive filesystem lock on the SAM file to prevent any sort of online copying
      - can only be copied/opened by users while system is offline
    - SYSKEY function introduced to improve the security of the hash
      - on disk copy of SAM file is partially encrypted with a key (the SYSKEY)
      - in memory version of SAM can still be dumped while online
        - via pwdump and similar tools
- Note: In windows 2000 and before, if an attacker



problem?

# Security in Win 7

- Better Desktop Auditing (Event Viewer)
  - XML based logs
  - tasks tied to events
  - fine grained support for audit of administrator privilege
  - easier to filter "noise" out to find the vents you want
  - reworked event viewer
  - also I suggest checking out **Splunk!**
- Network Access Protection
  - Helps insure that machines that connect to corporate networks meet corporate policy, are fully patched, and have up to date AV

# Security in Win 7

- Direct Access
  - When enabled, IT departments are allowed full access (via an IPSEC tunnel) to a machine.
  - Happens without knowledge of a user (a "seamless" experience) --- makes IT high value target
- App Locker
  - Helps IT eliminate unknown and unverified applications in the network
    - black lists, white lists, publisher **rules** based off of digital signatures



# Security in Win 7

- RMS (Rights Management Service)
  - Fine grained access control for documents
    - can control whether users can copy, print, etc a document
- Bit Locker
  - full HD encryption (desktops, laptops, and even thumbdrives)
  - focused on protecting the OS volume / partition
    - best used on dual / multi partition setups

# Least Privilege in Win 7

All registry keys implement access control through:

- Access Control Entries (ACE) in Discretionary Access Control Lists (DACLS)
  - ACE controls or monitors access to an object by a specified trustee
  - ACE entries contain:
    - SID (security identifier for the trustee)
    - access mask (specifies the access rights)
    - a flag that indicates the type of ACE (6 types)
    - a set of flags to dictate if ACE applies to child containers/directories

# Least Privilege in Win 7

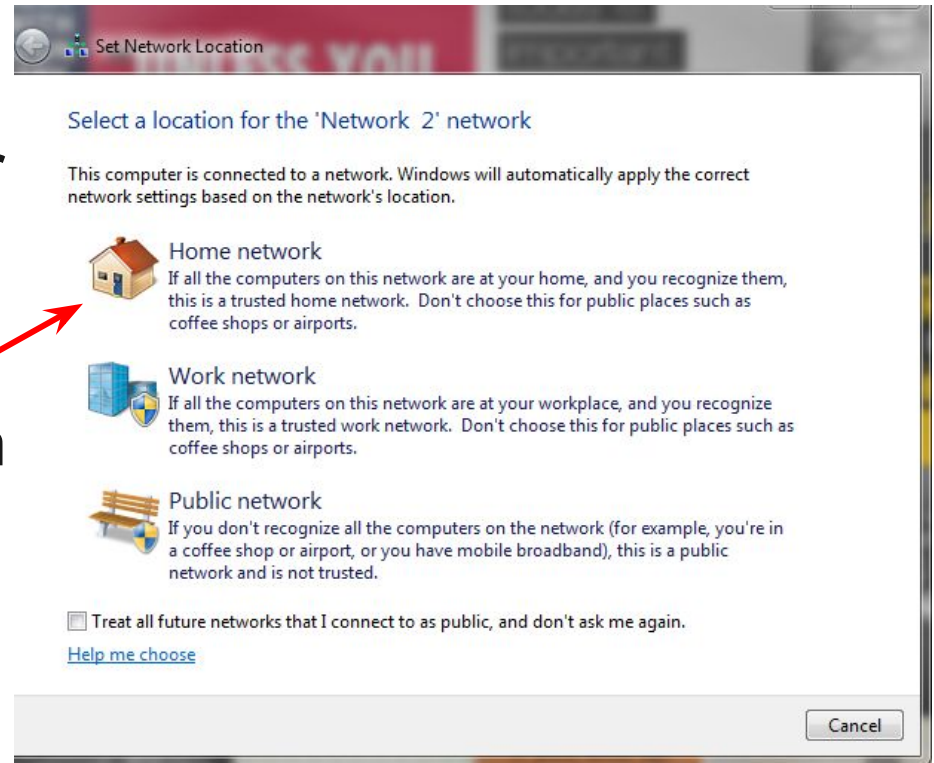
## DACL

- controlled by the owner of an object
- a NULL DACL permits all types of access to all users (do not use NULL DACLs)
- is an important part of application development

# Domains, Workgroups, and Homegroups

All computers running Windows on a network **MUST** be part of a workgroup or domain

- Windows PC's running on "home networks" can be also part of a homegroup, but not required



# Workgroup

- All PCs are peers
- No PCs has control of another PCs
- each PC has a set of user accounts
  - to log on a PC in the workgroup, you must have an account on that PC
- There is usually a 20 PC cap
- A workgroup is not protected by a password
- all PCs must be on the same LAN or subnet

# Homegroup

- PCs in a homegroup must belong to a workgroup as well.
- Homegroup makes it easy to share pictures, music, videos, documents, and printers
- Homegroup IS password protected
  - password is only typed in once, when adding the pc to the homegroup

# Domain

- One or more PCs are servers
  - Network admins use servers to control the security and permissions for all PCs in the domain.
    - any changes are automatically made to all PCs in the domain
- Password/Credentials required each logon
- If user has a domain account, he/she can log onto any PC on the domain
  - without needing a local account on that PC
- Thousands of PCs on a domain
- Can span across networks

# Windows Domain

## History:

- Microsoft wanted a way to just make everything work seamlessly together.
- Plug it into the network and you can start sharing, collaborating, etc.
- Started with LANMAN (LAN Manager), and then NTLM (NT LANMAN) protocols.
  - LANMAN hashes were easy to break
  - NTLM replaced it, but lots of things were backwards compatible with LANMAN until NTLM v2 came along



# Windows Domain

Collection of security principals that share some centralized directory database on a network.

Database history:

- Active Directory (win2000) Active Directory Domain Services (win serv2008)
  - servers that run Active Directory are called "domain controllers"
  - Uses LDAP v2/v3, Kerberos, and DNS
- NT Directory Services or NTDS on Windows NT systems prior to Active Directory

# Windows Domain

Each person who uses computers with on a domain received their own unique user name.

- this account can be assigned access to resources in the domain
- Domain connections supported over LAN, WAN, or VPN
  - VPN makes it a pivoting target

# Windows Domain Protocol

Security/Protocol history:

100% BROKEN (at some point)

- LANMAN
  - LM hashes were notoriously weak
  - still appears some places for backwards compatibility (even in apple OSes!)
- NTLMv1 (NT LANMAN)
  - provided backwards compatibility w/ LANMAN
- NTLMv2 introduced Windows NT 4.0
  - used HMAC-MD5 for authentication
  - uses CRC & Message Digest algorithms for integrity
  - Uses RC4 for encryption (no support for AES/SHA2)
- Kerberos adopted since Windows 2000

NTLM 2 sessions are similar to **MS-CHAPv2** (recently broken!)

# Windows Domain Protocol

Kerberos has replaced NTLM as the default authentication protocol in Active Directory, but *NTLM widely/default used where the client:*

- cannot access Domain controller
- is authenticating to a server using an IP addr
- is authenticating that belongs to a different Active Directory forest that has **legacy** NTLM trust...
- is authenticating to a server not part of a domain
- cannot access kerberos ports due to firewall

# Windows Firewall

- On by default in win7
- Has Home/Work/Public profiles
- Ties connections to applications
  - incoming and outgoing
  - can require user consent
- Application white list
  - can block all other incoming connections

# When things go wrong

Program / Browser inexplicably crashes

BSOD

or other crashes

etc

usually the worst

is fixable with

system restore



# The Ugly

**Rootkits!**

The following material comes from:  
**The Rootkit ARSENAL**

# Rootkits

Who builds them?

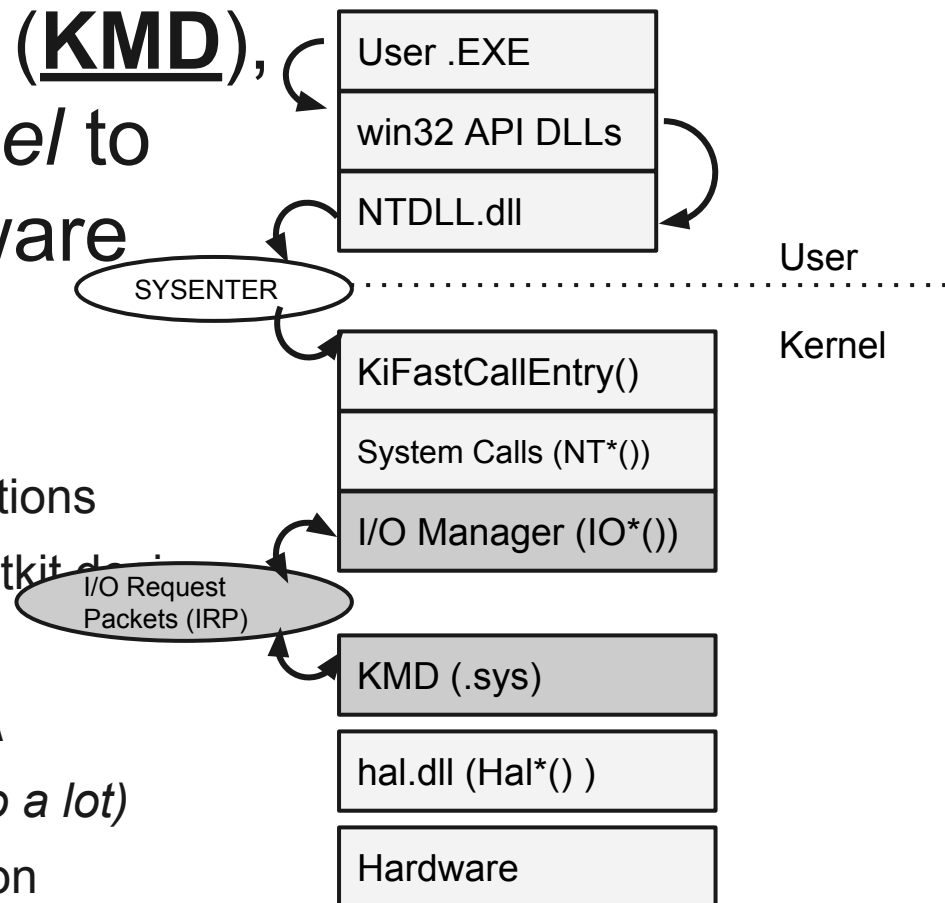
- Marketing ("what is security???" :D )
- Digital Rights Management
- "features" / anti-cheating
- Law Enforcement
- Industrial Espionage
- Political Espionage
- Cybercrime



# Getting into the kernel

## 1. Kernel Mode Drivers (**KMD**), are the *official channel* to gain access to hardware (ring 0)

- for system engineers
- can be loaded by user applications
- are like training-wheels for rootkit developers
- KMD's are usually .sys files
- C:\Windows\system32\drivers\
- offers: high complexity (*can do a lot*)
- cost: low stealth / high detection
- not difficult to code



# Windows Kernel Mode Security

## "Kernel-Mode Code Signing" (KMCS)

- When 64 bit windows came out, Microsoft required KMD's to be digitally signed with a *Software Publishing Certificate*
- Code signing does work!
  - associates drivers with authors/vendors
- not applicable to 32 bit versions
  - Though, now all versions do require:
    - signed: a subset (small) of core system binaries
    - signed: boot drivers
- attackers can still piggy back on signed drivers!

# KMCS Countermeasures

Not easy, involves piggybacking on "existing" drivers

How attackers change that "existing" problem:

- With money and a cover, can buy a SPC
  - way too brazen
  - release KMD with backdoor disguised as a bug
    - has occurred before
- Bad guys can get a job at a third party vendor and introduce backdoor/bug

OR instead, ***KMD Exploits***

# Getting into the kernel

## 2. Kernel / KMD Exploits

- Hundreds of drivers, and most are by 3rd parties
  - Common to have vulnerabilities: [http://www.nccgroup.com/media/190706/usb\\_driver\\_vulnerabilities\\_whitepaper\\_january\\_2013.pdf](http://www.nccgroup.com/media/190706/usb_driver_vulnerabilities_whitepaper_january_2013.pdf)
  - Nothing Microsoft can do about it
  - \*Easy\* and statistically likely to find bugs / exploits
- requires being a master of stack overflows / shellcode / ROP / etc...
- "This is the most generic attack today." - Joanna Rutkowska in the Rootkit Arsenal book
- The basic exploit payload is also a **KMD**, but leave

# Getting into the kernel

## 3. Modifying Call Tables / Hooking

- A Call table is an array where each element stores the address of a routine/function
  - i. IAT for user space
  - ii. IDT, CPU MSRs, GDT, SSDT, IRP Dispatch Table for kernel space
- Hooking = changing legitimate call table addresses with your own addresses
  - i. challenging in kernel, multiple copies of the GDT, IDT, and MSRs on a machine
  - ii. common targets:

1. processes, drivers, files & directories, Registry keys, and

# Getting into the kernel

Basics to an attacker's hook routine:

- save existing entry in target call table
- swap in new address for attacker hook
  - restore old entry when done (Why???)
    - usually left over from rootkit development
- Block calls made by certain applications (AV)
- Entirely replace the original routine
- Monitor system by intercepting input params
- Filter output parameters
- Perform malicious routine code, then call original routine

# Hooking Countermeasures

- Hooking \*can\* be easy to detect
  - known good address ranges
- Can look for \*sane\* address values in the IDT, CPU MSRs, GDT, SSDT, IRP...

## Counter Countermeasure!

- The above hooking checks all use the same system call to check the call tables (ZwQuerySystemInformation() )
  - **hook on this to override it**
- **Detour patching**: leave call table alone, just add a jump to your hook in the original routine!

# Getting into the kernel

## 4. Patching Methods

(High complexity)

- Binary Patching
  - modify the binary on disk, wait for reboot  
(**bootkit**)
    - easy to detect
- Run-time Patching
  - modify memory image of a module
    - super hard to detect
- Ways to code patches:
  - In-place patching
    - if a routine is only 10 bytes long, use only 10 bytes to do malicious hook (quite limited)
  - **Detour patching**



# Once you get into the kernel

Attackers like to stay there

- Its really hard work getting there!
- try to find ways to retain access across reboots and even updates
  - **patching kernel**
  - patching some part of the boot process

# Kernel Patch Protection (KPP)

MS introduced PatchGuard to guard the kernel against malicious *patching*.

- Periodic checks on core components against known good signatures or copies:

- ntoskrnl.exe
- hal.dll
- ci.dll (windows 7 Code Integrity DLL)
- kdcom.dll
- pshed.dll
- clfd.dll
- ndis.sys
- tcpip.sys

*If the periodic check fails, the system dies*

*periodic checks can occur every few milliseconds*

# KPP Countermeasures

- Only modify the parts of the kernel long enough to get in and establish a foothold, and then replace what you changed with the original code (must occur kinda quickly)
- KMD's and Patchguard operate at ring 0
  - A custom KMD or KMD Exploit can just disable PatchGuard!
    - Microsoft combats this with just crazy amounts of obfuscation
- Wonderful resource: <http://uninformed.org/>

# Theoretical Countermeasures

Both hooks and detour patches modify constructs that are pretty static

Thus it is possible to safeguard against them by:

- explicitly reconstructing them from known good versions
- checksum-based signature detection
  - i.e. check against known good signature for contents of a table or routine
- direct binary comparison detection

# But we can break this by

- 5. Modifying Kernel Objects or,  
Direct Kernel Object Manipulation (DKOM)
  - Objects here are abstractions for system resources
    - i. do not confuse with object oriented terminology
  - Objects are C structures
    - i. C is not a object oriented language
  - Changes to Objects are handled by the Object manager subsystem (too many subsystems!!!!)
  - Common targets:
    - i. EPROCESS object (to hide a process)
    - ii. DRIVER\_SELECTION object (load & hide KMB)
    - iii. Token object (subverts entirety of authentication),

# And so on

There are countermeasures to these attacks

And counter countermeasures to their defenses!

and so on

and so on



# Questions?

