Beyond Autorun: Exploiting vulnerabilities with removable storage

Jon Larimer
IBM X-Force Advanced R&D

jlarimer@us.ibm.com
jlarimer@gmail.com

Removable storage malware

- Malware has been spreading on removable storage since at least 1982 (**Elk Cloner**)
- First MS-DOS floppy virus emerged in 1986 (**Brain**)
- First PE infector developed in 1996 (**Bizatch**)
- First widespread virus to spread over USB drives was in 2007 (**SillyFD-AA**)
- In 2008, US Strategic Command banned all removable storage devices
- **Stuxnet**'s use of LNK vulnerability to spread over USB emerged in 2010

AutoRun / AutoPlay

- **AutoRun** originally designed for launching programs from CD
- `autorun.inf` file specifies program to run
- Windows XP SP2 allowed `autorun.inf` to work from USB devices (2004)
- Windows 7 changed so `autorun.inf` doesn't work from USB devices (2009)
- **AutoPlay** allows applications to handle media devices plugged into a PC, **AutoRun** is now a subset of this

Stuxnet and the LNK vulnerability

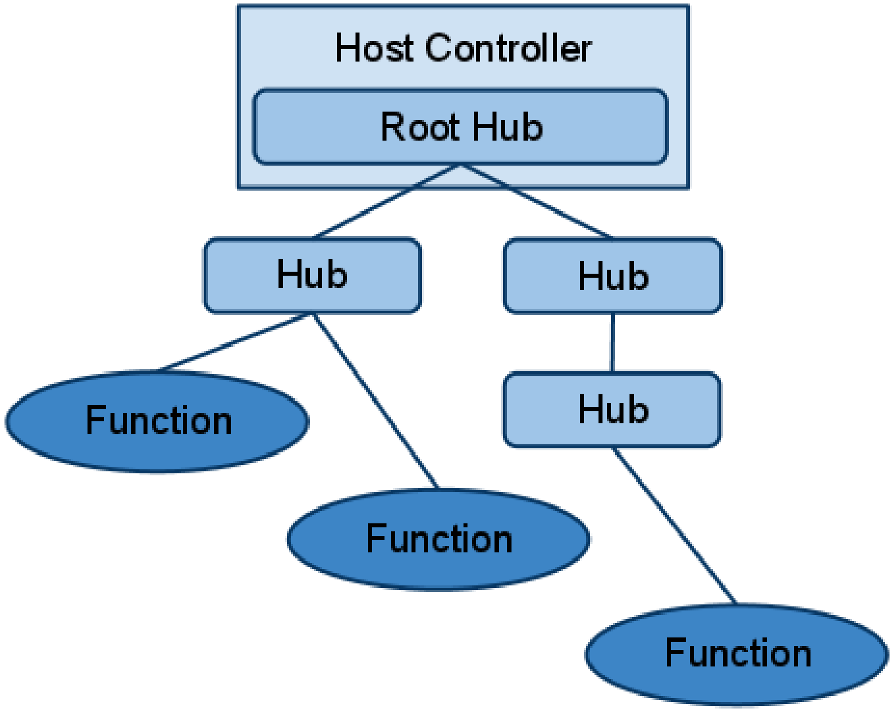
- Most entities disable **AutoRun** now
- LNK vulnerability (CVE-2010-2568) allows loading an arbitrary DLL just by browsing to a folder in Windows Explorer
- Also in File Open/Save dialogs...
- **Stuxnet** used this vulnerability to spread via USB drives without relying on **`autorun.inf`**
- If malware authors found one vulnerability like this... how many more are out there?

Attacks on physical systems

- Physical access is 'game over'
- What about full disk encryption?
- IEEE 1394 (FireWire) DMA physical memory access
  - Requires FireWire port and drivers
- Cold boot attack
  - Requires being able to boot from external media
- Removable storage attacks!
  - Most desktop OS's will automatically mount file systems on USB
  - Physical access not really necessary, just find someone to plug a device into their PC
  - If an exploit runs while the PC is already booted and the user is logged on, full disk encryption can be defeated

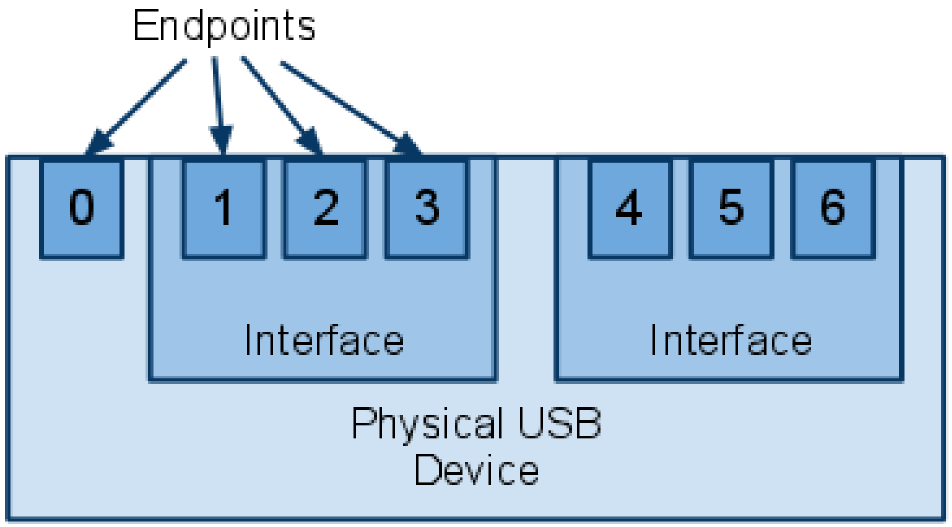
About USB

- Peripheral bus used by keyboards, mice, cameras, scanners, printers, mass storage devices
- Tiered star topology with the host controller at the top
- Polled bus, host initiates all transactions



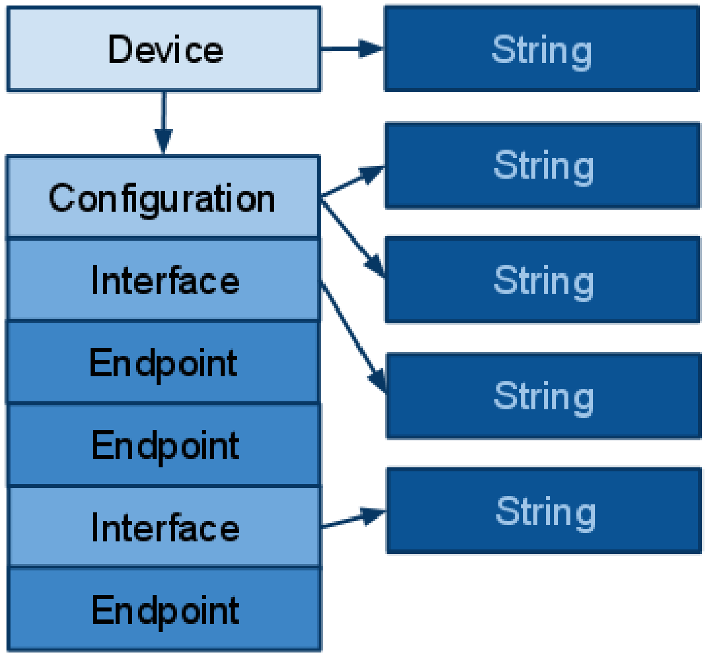
USB vocabulary

- **Device** – either a hub or a function
- **Hub** – connects multiple devices to another hub
- **Function** – a device that exposes a USB interface
- **Interface** – a collection of endpoints
- **Endpoint** – one end of a 'pipe'



USB descriptors

- Descriptors describe the device
- Used by the OS to load correct drivers
- Used by the drivers to communicate in a way the device can understand



USB device classes

- Device classes allow single device drivers to operate on devices of that class from any vendor
- The class defines the interfaces and protocols a device supports
- Most OS's include common class drivers
- Examples:
  - Human interface device (HID) – mouse/keyboard
  - Mass storage device (MSD) – flash drives
  - Printer class
  - Imaging class – scanners, cameras

Attacks using USB protocols

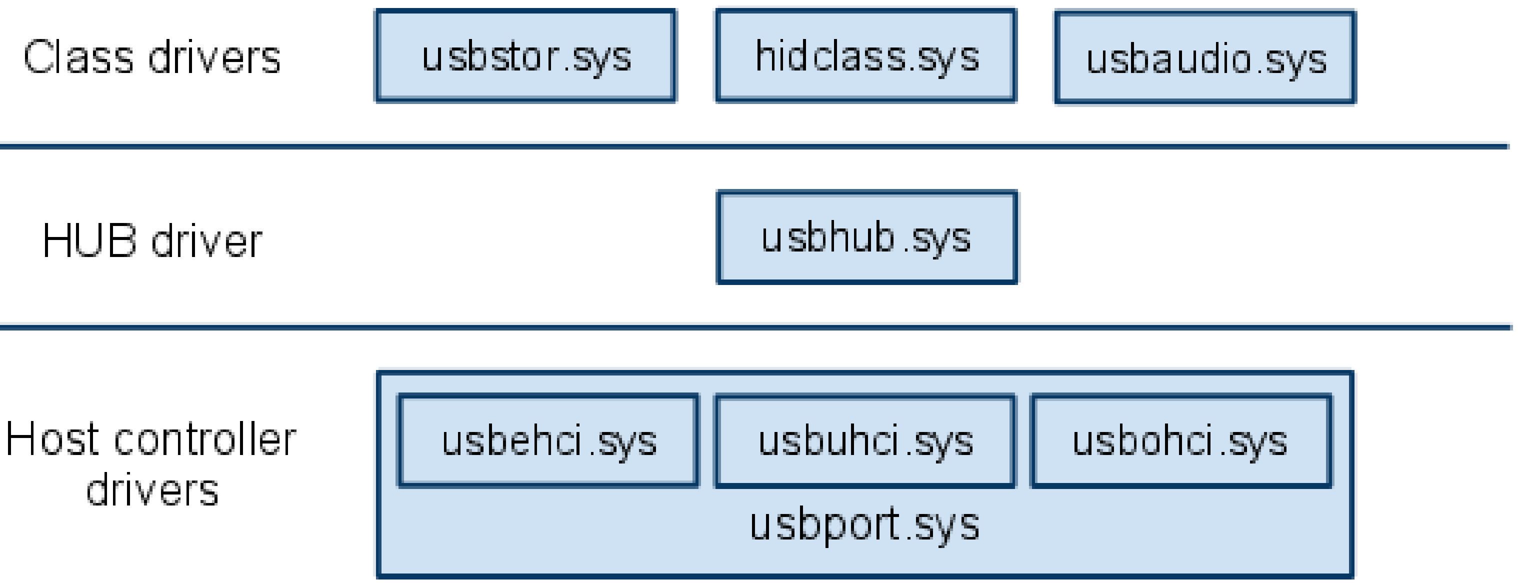
- BlackHat USA 2005, SPI Dynamics attacks on Windows XP USB drivers
  - –USB drivers expecting valid data from devices
- MWR InfoSecurity Auerswald Linux USB driver bug, 2009
  - –Problem handing USB descriptors
- PS3 Jailbreak in 2010
  - –Emulates a USB hub
  - –Connects and disconnects devices to trigger a heap overflow

Finding bugs in USB drivers

- Reversing / static analysis
- Fuzzing
  - Mortiz Jodiet – hardware+software (2009)
  - Tobias Mueller – QEMU-based fuzzer (2010)
- Other fuzzing options
  - Windows Driver Simulation Framework (DSF)
    - Included with Windows DDK
    - Emulate USB devices with scripting language
  - BOCHS
    - Implement fake device

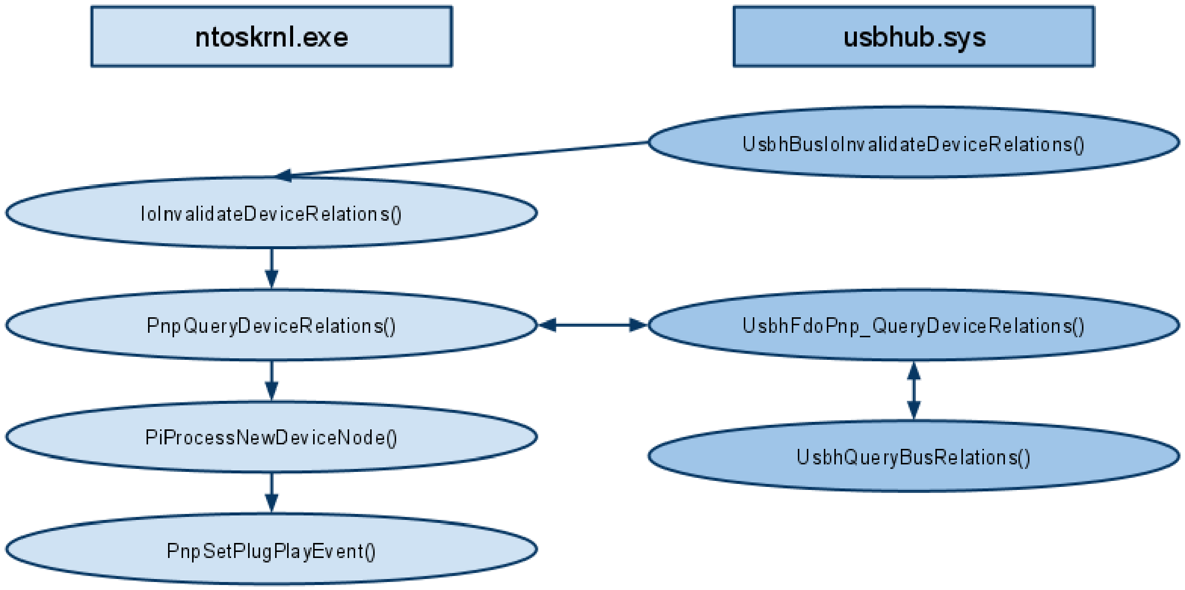
USB on Windows 7

- Core stack: **`usb[eou]hci.sys`**, **`usbport.sys`**
- Class drivers: **`usbstor.sys`**, **`hidclass.sys`**, etc



USB device recognition

- Kernel mode PnP Manager manages device relations
- Bus drivers notify PnP manager when devices are added/removed


ntoskrnl.exe
usbhub.sys
UsbhBusIoInvalidateDeviceRelations()
IoInvalidateDeviceRelations()
PnpQueryDeviceRelations()
UsbhFdoPnp_QueryDeviceRelations()
PiProcessNewDeviceNode()
UsbhQueryBusRelations()
PnpSetPlugPlayEvent()

Generating a device ID

- Windows queries USB device/interface descriptor
- Generates device ID string:
  - `USB\VID_`**`v(4)`**`&PID_`**`d(4)`**
  - `USB\VID_`**`v(4)`**`&PID_`**`d(4)`**`&REV_`**`r(4)`**
  - `USB\CLASS_`**`c(2)`**`&SUBCLASS_`**`s(2)`**`&PROT_`**`p(2)`**

Item	Value	Device Descriptor Value
v(4)	vendor ID	idVendor
d(4)	product ID	idProduct
r(4)	revision ID	bcdDevice
c(2)	class code	bDeviceClass
s(2)	subclass	bDeviceSubClass
p(2)	protocol	bDeviceProtocol

Locating USB device driver

- Device ID is used to match driver to device
- Kernel mode PnP manager checks registry to see if this device has a driver installed (`HKLM\System\CurrentControlSet\Enum\USB`)
- If not, driver user mode PnP manager searches for driver ("Plug and Play" service, `umpnpmgr.dll`)
  - First checks Windows Update (using `chkwudrv.dll`)
  - Then the local DriverStore (`%SystemRoot%\System32\DriverStore`)
  - Checks the DevicePath (`%SystemRoot%\Inf`)
  - If a driver can't be found, it's reported via Windows Error Reporting

Drivers from Windows Update?

- Windows 7 can automatically search Windows Update for the latest drivers for a new device
- Drivers are uploaded by the hardware vendors themselves (WinQual)
- Requirements are:
  – a Class 3 digital certificate
  –a driver that can pass the WHQL test
  –INF file must specify vendor and product IDs
- Companies don't submit code, they submit signed binaries and WHQL test logs

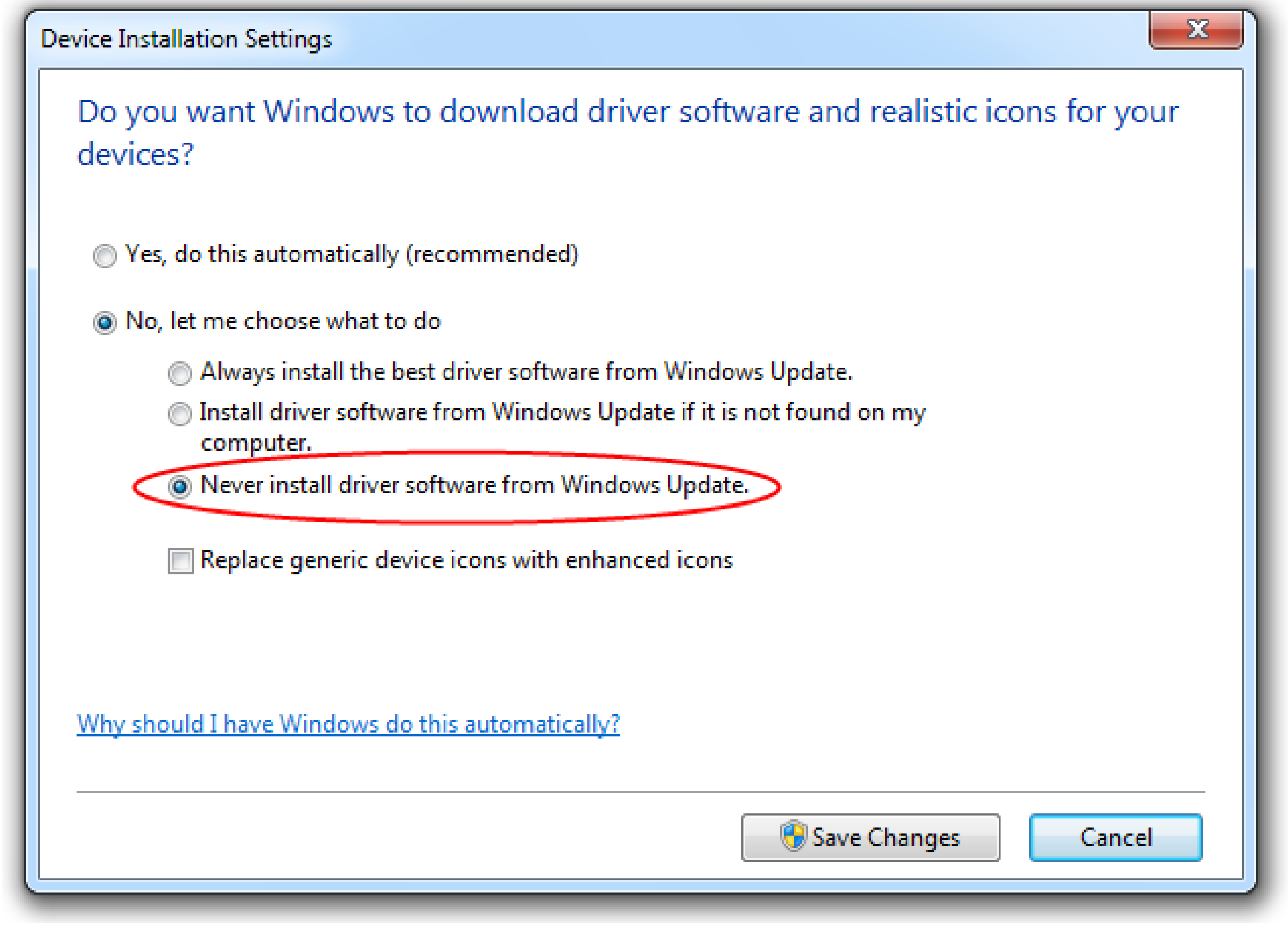
Windows Update Driver Attack #1

- Malicious entity obtains class 3 certificate ($99-$500)
- Develops driver for hardware that doesn't exist, but looks legit and passes WHQL ($250 for WHQL testing)
- Uploads driver to WinQual
- Develop hardware device that matches the submitted INF but triggers a cleverly hidden backdoor
- Can access any Vista/Win7 machine with a working USB port and Windows Update drivers enabled

Windows Update Driver Attack #2

- Malicious entity reverse engineers Windows Update driver check
- Writes script to enumerate through every USB vendor/product ID pair and download every available driver
- Analyzes the thousands of downloaded drivers for vulnerabilities, finds some, writes exploits
- Puts exploit on USB dev board firmware
- Can access any Vista/Win7 machine with a working USB port and Windows Update drivers enabled

Staying safe from malicious drivers



Device Installation Settings

Do you want Windows to download driver software and realistic icons for your devices?

○ Yes, do this automatically (recommended)

● No, let me choose what to do

  ○ Always install the best driver software from Windows Update.

  ○ Install driver software from Windows Update if it is not found on my computer.

  ● Never install driver software from Windows Update.

  ☐ Replace generic device icons with enhanced icons

Why should I have Windows do this automatically?

USB mass storage device stack

Filesystem Driver	ntfs.sys
Volume Management	volmgr.sys
	fvevol.sys
	volsnap.sys
Partition Mgmt	partmgr.sys
Storage Class	disk.sys
	usbstor.sys
Bus Drivers	usbhub.sys
	usbehci.sys
	usbport.sys

File system drivers in Windows

- Windows natively supports NTFS, FAT12/16/32, ExFAT, CDFS (ISO 9660), and UDF
- File systems recognized by **`fs_rec.sys`**, which then loads the right driver

Filesystem	**Device Type**	**Driver**
CDFS	CD-ROM	cdfs.sys
UDF	CD-ROM	udfs.sys
UDF	DISK	udfs.sys
FAT	DISK	fastfat.sys
FAT	CD-ROM	fastfat.sys
NTFS	DISK	ntfs.sys
ExFAT	DISK	exfat.sys

Finding bugs in file system drivers

- Reverse engineering
- Source for CDFS and FastFAT drivers are included in DDK
- Fuzzing?
  - **FileDisk** by Bo Brantén
  - Allows mounting a disk image in a file as a volume
  - Either randomly perturb a disk image or modify the code to modify data read from disk image
  - Make your fuzzer smart (recognize and modify file system metadata, etc)
  - Code coverage/taint analysis with QEMU or BOCHS?

AutoPlay

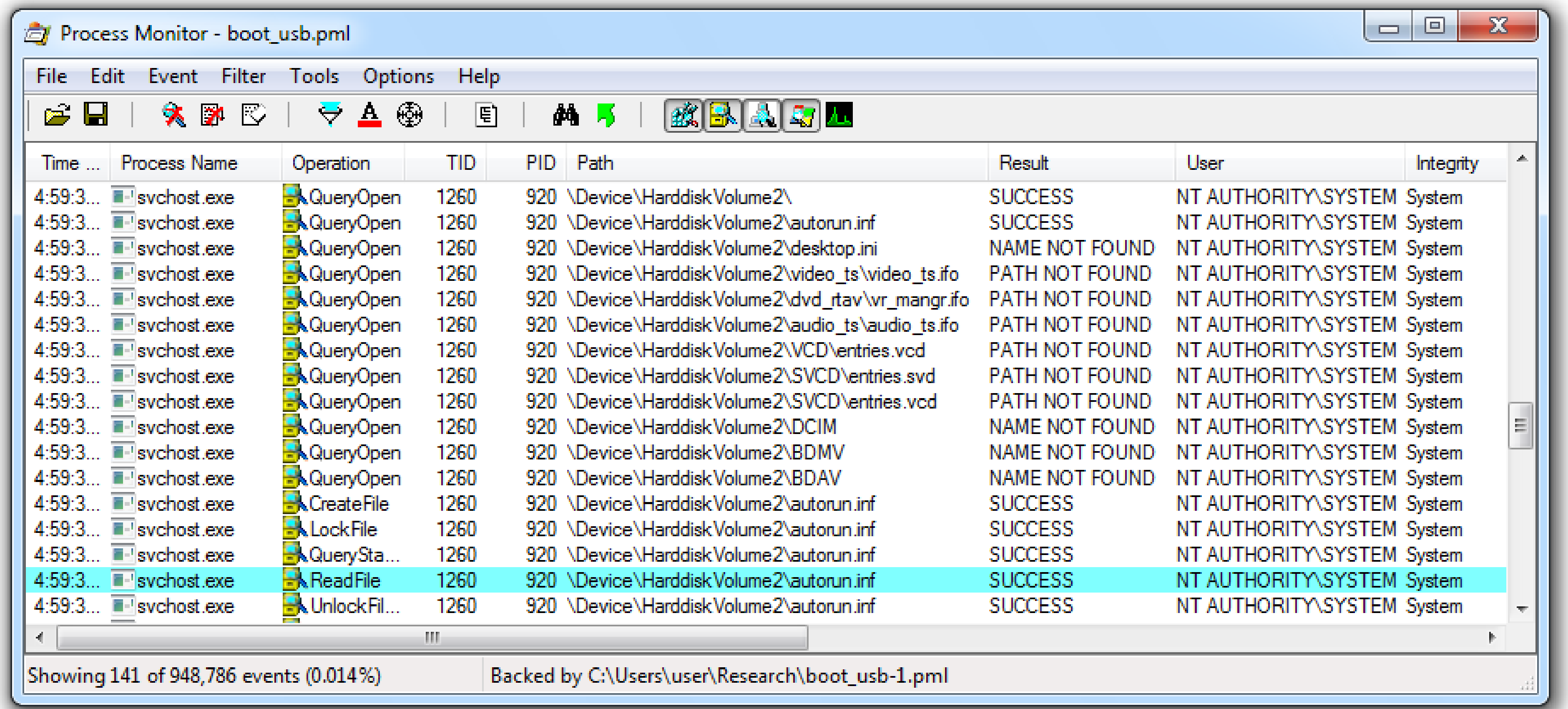
- **AutoPlay** is largely implemented in the Shell Hardware Detection Service (**`shsvcs.dll`**)
- Registers for PnP events with **`RegisterDeviceNotification()`**
- Checks for the existence of certain files and directories on newly mounted volumes
- Determines media type – Video CD, DVD, digital camera media
- Takes the configured AutoPlay action based on determined media

AutoPlay media checks

File	Purpose
autorun.inf	Autorun file
desktop.ini	Desktop.ini file
video_ts\\video_ts.ifo	DVD Video
dvd_rtav\\vr_mangr.ifo	DVD Video
audio_ts\\audio_ts.ifo	DVD Audio
VCD\entries.vcd	Video CD
SVCD\entries.svd	Super Video CD
SVCD\entries.vcd	Super Video CD
DCIM	Photos
BDMV	Blu-ray disc
BDAV	Blu-ray disc

AutoPlay media checks screenshot

- Media checks even when nobody is logged in...



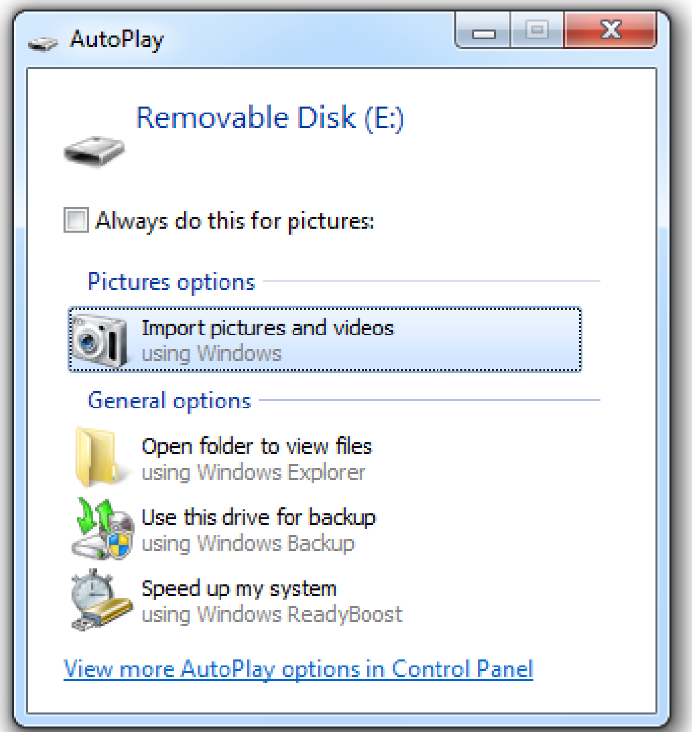
Process Monitor - boot_usb.pml

Time ...	Process Name	Operation	TID	PID	Path	Result	User	Integrity
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\	SUCCESS	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\autorun.inf	SUCCESS	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\desktop.ini	NAME NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\video_ts\video_ts.ifo	PATH NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\dvd_rtav\vr_mangr.ifo	PATH NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\audio_ts\audio_ts.ifo	PATH NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\VCD\entries.vcd	PATH NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\SVCD\entries.svd	PATH NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\SVCD\entries.vcd	PATH NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\DCIM	NAME NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\BDMV	NAME NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QueryOpen	1260	920	\Device\HarddiskVolume2\BDAV	NAME NOT FOUND	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	CreateFile	1260	920	\Device\HarddiskVolume2\autorun.inf	SUCCESS	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	LockFile	1260	920	\Device\HarddiskVolume2\autorun.inf	SUCCESS	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	QuerySta...	1260	920	\Device\HarddiskVolume2\autorun.inf	SUCCESS	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	ReadFile	1260	920	\Device\HarddiskVolume2\autorun.inf	SUCCESS	NT AUTHORITY\SYSTEM	System
4:59:3...	svchost.exe	UnlockFil...	1260	920	\Device\HarddiskVolume2\autorun.inf	SUCCESS	NT AUTHORITY\SYSTEM	System

Showing 141 of 948,786 events (0.014%) Backed by C:\Users\user\Research\boot_usb-1.pml

AutoPlay dialog

- By default Windows will ask what you want to do with media
- Whatever AutoPlay option is selected won't take effect if the screen is locked
- Thumbnails/icons won't be rendered if screen is locked
- Can't rely on shell extension exploits for physical attacks


AutoPlay

Removable Disk (E:)

Always do this for pictures:

Pictures options

Import pictures and videos
using Windows

General options

Open folder to view files
using Windows Explorer

Use this drive for backup
using Windows Backup

Speed up my system
using Windows ReadyBoost

View more AutoPlay options in Control Panel

Is AutoPlay useful for hackers at all?

- When targeting AutoPlay or Windows Portable Devices (WPD) applications, exploits will only work when someone is using the PC (not when screen is locked)
- AutoPlay does cause certain files to be read and parsed whenever a new volume is mounted
  - –Even when the screen is locked
  - –Even when nobody is logged in
  - –This fact could be used to trigger vulnerabilities in file system drivers

Windows Explorer

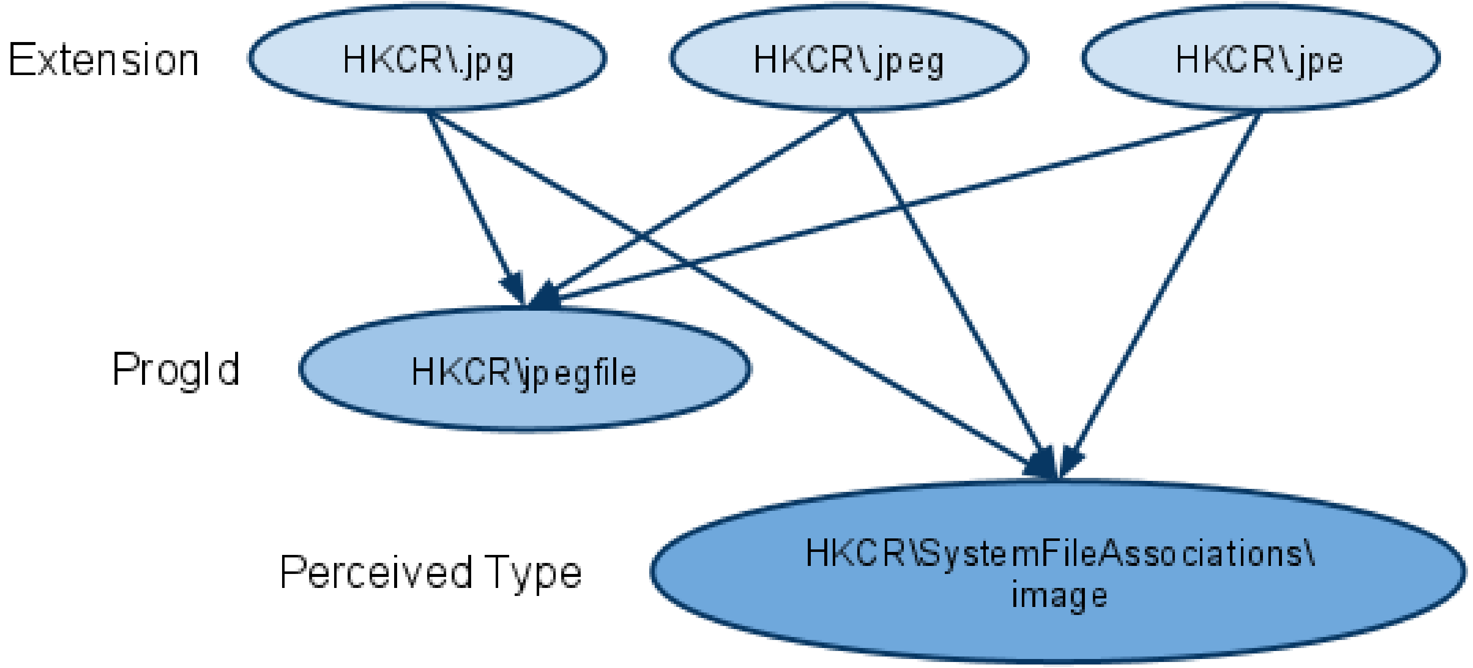
- The OS shell, your main interface for interacting with files and folders
- Keeps getting (arguably) prettier and prettier
- Supports image thumbnails, document previews, file metadata retrieval
- Some of these features will read and parse files without you explicitly trying to open them
- Bad things can happen when the OS tries to parse untrusted data

Files, file types, and perceived types

- How Windows handles files is determined by registry settings
- File type is determined by extension (`.doc`, `.jpg`)
- Extensions map to a "ProgId" in the registry (`Word.Document.8`, `jpegfile`),
- Perceived types match an extension to a generic type (`image`, `document`)
- Shell extension handlers are usually registered for each extension, ProgId, or perceived types

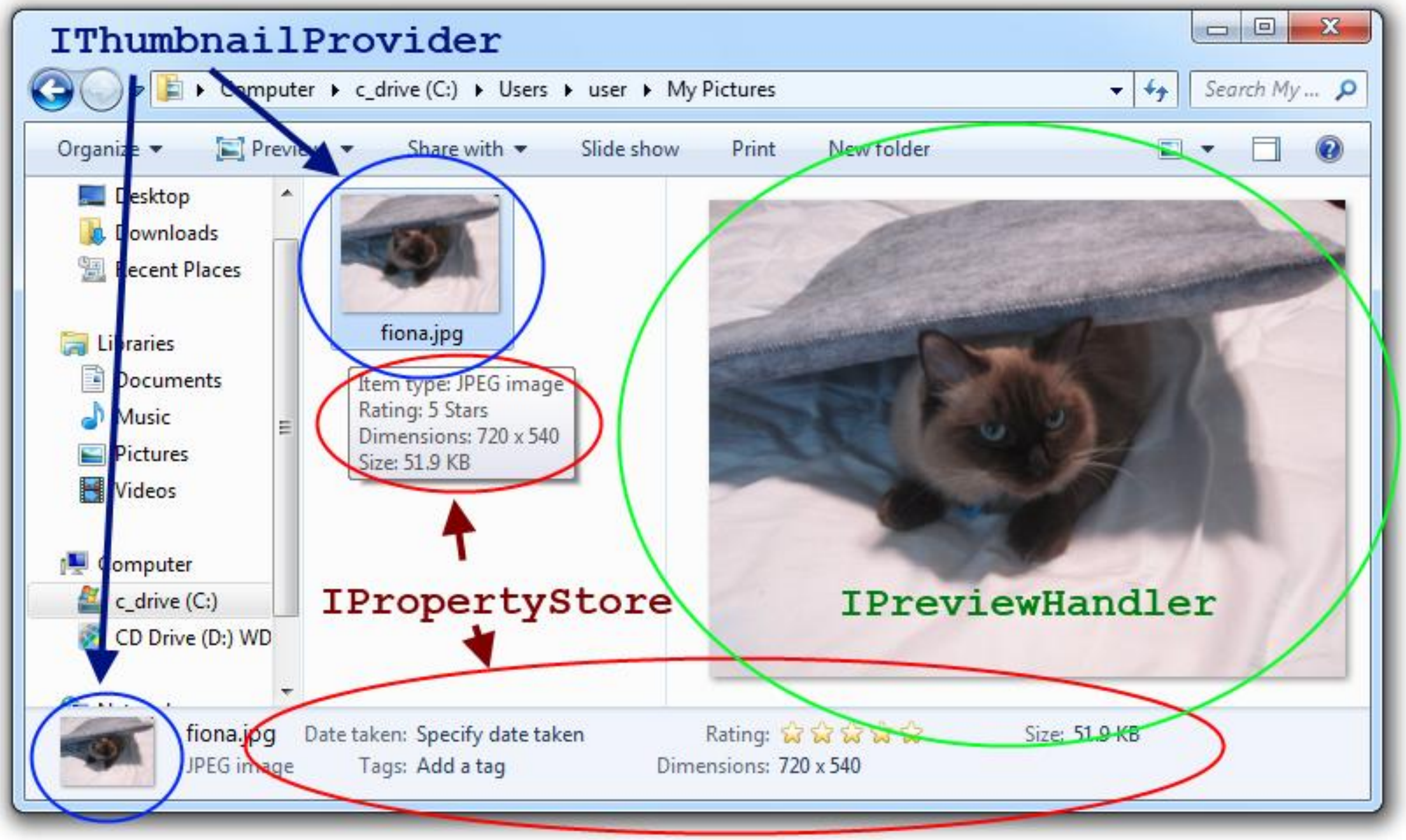
Registering file types

- Extension and ProgIds are under **`HKEY_CLASSES_ROOT`** or **`HKEY_CURRENT_USER\Software\Classes`**



Shell extension handlers

- Registered to provide custom icons, thumbnails, previews, tooltips, and other features for files
- COM objects that implement an interface



Item type: JPEG image
Rating: 5 Stars
Dimensions: 720 x 540
Size: 51.9 KB

fiona.jpg
JPEG image
Date taken: Specify date taken
Tags: Add a tag
Rating: ☆☆☆☆☆
Dimensions: 720 x 540
Size: 51.9 KB

Type of shell extension handlers

- **Icon handlers**: used in **Small** icons / **Details** view
- **Thumbnail handlers**: used in **Medium**, **Large**, **Extra Large** icon views and the **Tiles** and **Content** views
- **Intotip handlers**: used for file metadata when mouse hovers over file
- **Preview handlers**: used when **Preview** pane is enabled
- **Property handlers**: used in **Details** and other views when file metadata is required
  - Can also be used by **Intotip** and **Thumbnail** handlers...

Vulnerabilities in shell ext handlers

- LNK vulnerability used by Stuxnet – LNK file icon handler
- PDF preview/thumbnail has been known to trigger malicious PDFs without clicking
- Moti and Xu Hao "A vulnerability in my heart" at POC2010 – embedded BMP thumbnail vulnerability in the property handler for OLE document files
- Many times these can be exploited remotely too (e-mail attachments, links to network shares, etc)
- I'm sure there will be more, that's why I'm here!

Icon handler registration

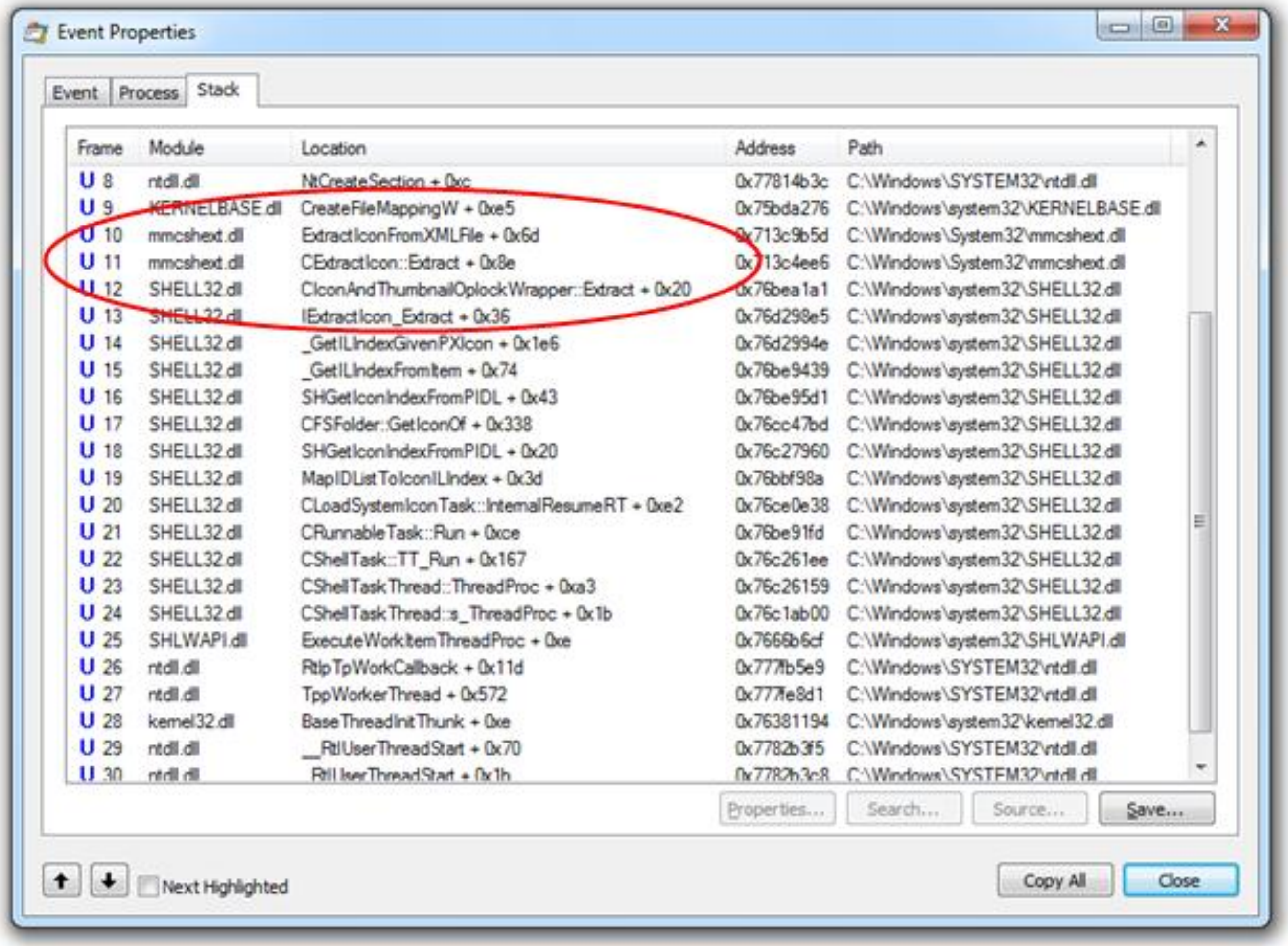
- Registered with the subkey `ShellEx\IconHandler` under the ProgId or perceived type key
- Places Explorer looks for icon handlers:
  - `HKCU\Software\Classes\jpegfile\ShellEx\IconHandler`
  - `HKCR\jpegfile\ShellEx\IconHandler`
  - `HKCU\Software\Classes\SystemFileAssociations\.jpeg\ShellEx\IconHandler`
  - `HKCR\SystemFileAssociations\.jpeg\ShellEx\IconHandler`
  - `HKCU\Software\Classes\SystemFileAssociations\image\ShellEx\IconHandler`
  - `HKCR\SystemFileAssociations\image\ShellEx\IconHandler`
- Example of registration for .MSC:
  - `HKEY_CLASSES_ROOT\.msc = "MSCFile"`
  - `HKEY_CLASSES_ROOT\mscfile\shellex\IconHandler = "{7A80E4A8-8005-11D2-BCF8-00C04F72C717}"`
  - `HKEY_CLASSES_ROOT\CLSID\{7A80E4A8-8005-11D2-BCF8-00C04F72C717}\InprocServer32 = "%SystemRoot%\system32\mmcshext.dll"`

Icon handler implementation

- Implements `IExtractIcon[WA]` interface
- Also implements either `IPersistFile`, `IInitializeWithFile`, `IIinitializeWithItem`, or `IIinitializeWithStream`
- `IPersistFile::Load()` is called to specify the file name
- `IExtractIcon::GetIconLocation()` to get path to file with icon
- `IExtractIcon::Extract()` to get icon handles

Icon handler stack trace

- From SysInternals Process Monitor



Frame	Module	Location	Address	Path
U 8	ntdll.dll	NtCreateSection + 0xc	0x77814b3c	C:\Windows\SYSTEM32\ntdll.dll
U 9	KERNELBASE.dll	CreateFileMappingW + 0xe5	0x75bda276	C:\Windows\system32\KERNELBASE.dll
U 10	mmcshext.dll	ExtractIconFromXMLFile + 0x6d	0x713c9b5d	C:\Windows\System32\mmcshext.dll
U 11	mmcshext.dll	CExtractIcon::Extract + 0x8e	0x713c4ee6	C:\Windows\System32\mmcshext.dll
U 12	SHELL32.dll	CIconAndThumbnailOplockWrapper::Extract + 0x20	0x76bea1a1	C:\Windows\system32\SHELL32.dll
U 13	SHELL32.dll	IExtractIcon_Extract + 0x36	0x76d298e5	C:\Windows\system32\SHELL32.dll
U 14	SHELL32.dll	_GetILIndexGivenPXIcon + 0x1e6	0x76d2994e	C:\Windows\system32\SHELL32.dll
U 15	SHELL32.dll	_GetILIndexFromItem + 0x74	0x76be9439	C:\Windows\system32\SHELL32.dll
U 16	SHELL32.dll	SHGetIconIndexFromPIDL + 0x43	0x76be95d1	C:\Windows\system32\SHELL32.dll
U 17	SHELL32.dll	CFSFolder::GetIconOf + 0x338	0x76cc47bd	C:\Windows\system32\SHELL32.dll
U 18	SHELL32.dll	SHGetIconIndexFromPIDL + 0x20	0x76c27960	C:\Windows\system32\SHELL32.dll
U 19	SHELL32.dll	MapIDListToIconILIndex + 0x3d	0x76bbf98a	C:\Windows\system32\SHELL32.dll
U 20	SHELL32.dll	CLoadSystemIconTask::InternalResumeRT + 0xe2	0x76ce0e38	C:\Windows\system32\SHELL32.dll
U 21	SHELL32.dll	CRunnableTask::Run + 0xce	0x76be91fd	C:\Windows\system32\SHELL32.dll
U 22	SHELL32.dll	CShellTask::TT_Run + 0x167	0x76c261ee	C:\Windows\system32\SHELL32.dll
U 23	SHELL32.dll	CShellTaskThread::ThreadProc + 0xa3	0x76c26159	C:\Windows\system32\SHELL32.dll
U 24	SHELL32.dll	CShellTaskThread::s_ThreadProc + 0x1b	0x76c1ab00	C:\Windows\system32\SHELL32.dll
U 25	SHLWAPI.dll	ExecuteWorkItemThreadProc + 0xe	0x7666b6cf	C:\Windows\system32\SHLWAPI.dll
U 26	ntdll.dll	RtlpTpWorkCallback + 0x11d	0x777fb5e9	C:\Windows\SYSTEM32\ntdll.dll
U 27	ntdll.dll	TppWorkerThread + 0x572	0x777fe8d1	C:\Windows\SYSTEM32\ntdll.dll
U 28	kernel32.dll	BaseThreadInitThunk + 0xe	0x76381194	C:\Windows\system32\kernel32.dll
U 29	ntdll.dll	__RtlUserThreadStart + 0x70	0x7782b3f5	C:\Windows\SYSTEM32\ntdll.dll
U 30	ntdll.dll	_RtlUserThreadStart + 0x1b	0x7782b3c8	C:\Windows\SYSTEM32\ntdll.dll

Thumbnail handler registration

- Registers at:
  - `ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}` (`IThumbnailProvider`)
  - `ShellEx\{BB2E617C-0920-11D1-9A0B-00C04FC2D6C1}` (`IExtractImage`)
- Explorer checks (for `.ini` files):
  - `HKCU\Software\Classes\inifile\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCR\inifile\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCU\Software\Classes\.ini\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCR\.ini\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCU\Software\Classes\SystemFileAssociations\text\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCR\SystemFileAssociations\text\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCU\Software\Classes\*\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCR\*\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCU\Software\Classes\AllFilesystemObjects\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`
  - `HKCR\AllFilesystemObjects\ShellEx\{E357FCCD-A995-4576-B01F-234630154E96}`

Thumbnail handler registration

- **`IThumbnailProvider`**, using the file extension:
  - `HKEY_CLASSES_ROOT\.avi\ShellEx\{e357fccd-a995-4576-b01f-234630154e96} = "{9DBD2C50-62AD-11D0-B806-00C04FD706EC}"`
  - `HKEY_CLASSES_ROOT\CLSID\{9DBD2C50-62AD-11d0-B806-00C04FD706EC}\ InProcServer32 = "SystemRoot%\system32\shell32.dll"`
- **`IExtractImage`**, using the ProgId:
  - `HKEY_CLASSES_ROOT\.ttf = "ttffile"`
  - `HKEY_CLASSES_ROOT\ttffile\shellex\{BB2E617C-0920-11d1-9A0B-00C04FC2D6C1} = {B8BE1E19-B9E4-4ebb-B7F6-A8FE1B3871E0}`
  - `HKEY_CLASSES_ROOT\CLSID\{B8BE1E19-B9E4-4ebb-B7F6-A8FE1B3871E0}\ InProcServer32 = "%SystemRoot%\system32\fontext.dll"`
- By default, thumbnail handlers run in isolated process (COM Surrogate, **`dllhost.exe`**)
  - Can be disabled with **DisableProcessIsolation=1** in the CLSID for the COM object class
  - Or by calling **`IShellItem::BindToHandler()`** with a **`NULL`** context
  - Isolated process runs as same user/context as **`explorer.exe`**

IThumbnailProvider implementation

- Explorer checks for this first when generating a thumbnail
- Also requires implementing **IInitializeWithStream**, **IInitializeWithItem**, or **IInitializeWithFile**
- Safer to implement **IInitializeWithStream**, since Windows doesn't have to give the thumbnail provider access to the file system itself – just the file handle
- Only exposes one method
  - `HRESULT GetThumbnail(UINT cx, HBITMAP *phbmp, WTS_ALPHATYPE *pdwAlpha);`
- Many file types and perceived types use the "Property Thumbnail Handler"

`IExtractImage` implementation

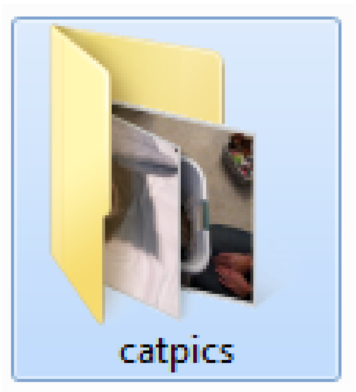
- Only used if there's no **`IThumbnailProvider`** registered
- Also requires **`IPersistFile`** or one of the regular shell extension initialization interfaces
- There are still some file types with only **`IExtractImage`** implementations on Windows 7
- Exposes two methods:
  - `HRESULT GetLocation(LPWSTR pszPathBuffer, DWORD cchMax, DWORD *pdwPriority, const SIZE *prgSize, DWORD dwRecClrDepth, DWORD *pdwFlags);`
  - `HRESULT Extract(HBITMAP *phBmpImage);`

Property thumbnail handler

- Used as the **`IThumbnailProvider`**/ **`IExtractImage`** interface for many file types
- Uses the Windows Property System to read thumbnails from files
- Located in the **`CPropertyThumbnailHandler`** class in **`shell32.dll`**
- Looks for three different property keys:
  - **`PKEY_Thumbnail (VT_CF)`**
  - **`PKEY_ThumbnailStream (VT_STREAM)`**
  - **`PKEY_ImageParsingName (VT_VECTOR|VT_LPWSTR)`** or **`(VT_ARRAY|VT_BSTR)`**

Folder thumbnails

- Explorer can generate icons for folders that contain thumbnails of files in that folder
- A thumbnail vulnerability could be exploited without even having the file in the current folder
- Explorer picks two files to thumbnail
- Can use icon or thumbnail handlers to generate embedded thumbnails
- See **`CFolderThumbnail()`** in **`shell32.dll`**



Infotip handlers

- **Infotips** can be static or dynamic
  - –Static strings in the registry, could point to a DLL resource
  - –Static strings could also reference properties
    - `HKEY_CLASSES_ROOT\SystemFileAssociations\.exe\InfoTip = "prop:System.FileDescription;System.Company;System.FileVersion;System.DateCreated;System.Size"`
  - –Dynamic **Infotip** handlers can implement the `IQueryInfo` interface and register in `ShellEx\{00021500-0000-0000-C000-000000000046}`
- There are a few `IQueryInfo` handlers registered by default, but most **Infotips** come from the Property System

Preview handlers

- Shown in preview pane when a file is clicked
- Runs in an isolated, low integrity level process by default (`prevhost.exe`)
- Can also implement their own COM host – check integrity level with Process Monitor
- Low integrity level can be disabled in the registry, **DisableLowILProcessIsolation=1**
- Adobe Reader 9 had low-IL disabled in the preview handler, Reader X enables it

Preview handler registration & implementation

- Registered under `ShellEx\{8895B1C6-B41F-4C1C-A562-0D564250836F}`
- Also requires an entry under `HKLM\Microsoft\Windows\CurrentVersion\PreviewHandlers`
- Example registration:
  - `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\PreviewHandlers\{BFD468D2-D0A0-4bdc-878C-E69C2F5B435D} = "Microsoft Windows Mail Html Preview Handler"`
  - `HKEY_CLASSES_ROOT\.html = htmlfile`
  - `HKEY_CLASSES_ROOT\htmlfile\shellex\{8895B1C6-B41F-4C1C-A562-0D564250836F} = "{f8b8412b-dea3-4130-b36c-5e8be73106ac}"`
  - `HKEY_CLASSES_ROOT\CLSID\{f8b8412b-dea3-4130-b36c-5e8be73106ac}\InprocServer32 = "%SystemRoot%\system32\inetcomm.dll"`
- Implements `IPreviewHandler` and a few other interfaces
- `DoPreview()` is the magic function that does the rendering

Auditing shell extension handlers

- Reversing COM can be hard, use Process Monitor to get stack traces
- Windows debug symbols help A LOT for extensions included with Windows
- Fuzzing can work since we know how Windows uses the COM interfaces
  - –Load COM object
  - –Initialize with stream/file
  - –Fuzz!

Exploiting shell extension handlers

- ASLR+DEP is tough to get around
- Brute force?
  - –Will only work if the handler has it's own exception handler
  - –If a handler crashes, Windows notifies the user and requires interaction to continue
    - Could crash **`explorer.exe`**, closing the window
    - Crashing **`dllhost.exe`** results in another **`dllhost.exe`** being loaded for the next icon, etc
- Force process to load non-ASLR DLL
  - –All system DLLs in Win7 are built with **`/DYNAMICBASE`**
  - –Might be easier to load/find 3rd party non-ASLR DLL's in **`explorer.exe`**
  - –**`dllhost.exe`** can load more than one thumbnail handler DLL at a time

Windows Property System

- Allows reading/writing of metadata for files without relying on file system features (NTFS alternate streams...)
- Examples: JPEG Exif data, MP3 ID3 tags, document authors, etc
- Used by Explorer (Details, Content, Infotips, etc)
- Also used by Windows Search
  - –Indexing files on disk, email inbox, etc
  - –Uses low integrity isolated process by default
- Not a shell extension, but feels a lot like it
  - –Registered in different part of registry
  - –Registration by extension only

Property handler registration

- Property handlers are registered at **`HKLM\Software\Microsoft\Windows\CurrentVersion\PropertySystem\PropertyHandlers`**
- Example registration:
  - `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\PropertySystem\PropertyHandlers\.jpg = "{a38b883c-1682-497e-97b0-0a3a9e801682}"`
  - `HKEY_CLASSES_ROOT\CLSID\{a38b883c-1682-497e-97b0-0a3a9e801682}\InProcServer32 = "C:\Windows\system32\PhotoMetadataHandler.dll"`
- Can use **DisableProcessIsolation** in COM object's key

Property handler vulnerabilities?

- The recent BMP thumbnail thing was exploitable through the Thumbnail Property Handler for MS Office files
- Didier Stevens noticed that the PDF shell extension handler could be exploited through details view, but that was in XP through `IColumnHandler` (no longer in Windows 7)
- If they're run be `explorer.exe`, exploits can be useful
- Might be less useful if exploited through Windows Search...
  - If they're run by `SearchFilterHost.exe` (isolated low isolation level search host), a privilege escalation is required to escape
  - If **DisableProcessIsolation** is enabled, they can run in `SearchProtocolHost.exe` which has access to the file system
  - External media aren't searched by default, but email could work for a remote exploit
  - In 2005, F-Secure observed that Google Desktop Search could trigger a vulnerability in WMF files by indexing

Folder customization

- `desktop.ini`
  - –Can specify icons and Infotips for folders
  - –Can be used to create virtual folder (Recycle Bin) by specifying `[.ShellClassInfo]` entry
  - –Can contain UNC paths for some fields, triggering external connections
  - –There was a buffer overflow in `explorer.exe` in XP when processing this file
  - –Another vulnerability allowed loading arbitrary COM objects

Shell namespace extensions

- Provides an interface for creating a 'virtual folder' that can be browsed in Explorer
- Used for Recycle Bin, My Computer, Control Panel, etc
- Also used for handling `.zip` files and the `.cab` file viewer
- Virtual folders can be created through
  - Registry settings
  - `desktop.ini [.ShellClassInfo]` entry
  - Creating a folder named `xxx.{CLSID}`
- `desktop.ini` and the folder both need the `+s` (system) attribute to work

USB on Linux

- **usbcore** in `drivers/usb/core`
- Host controller driver framework is `drivers/usb/core/hdc.c`
  - UHCI: `drivers/usb/host/usb-uhci.c`
  - EHCI: `drivers/usb/host/usb-ehci.c`
- Hub driver in `drivers/usb/core/hub.c`
- Interface drivers register by calling `usb_register()` or `usb_register_driver()`, specifying which vendor/product IDs they work with
- `drivers/core/usb/driver.c usb_match_id()` takes care of the matching, then the driver is loaded

USB mass storage on Linux

- Storage class driver in `drivers/usb/storage/usb.c`
- `storage_probe()`
  - Sets up a SCSI host structure
  - adds SCSI host to SCSI subsystem
  - `scsiglue.c` and `protocol.c` take care of converting SRBs to URBs for the USB drivers
- SCSI subsystem adds a block device (`/dev/sdb`)
- **udev** is notified

udev, udisks, and D-Bus

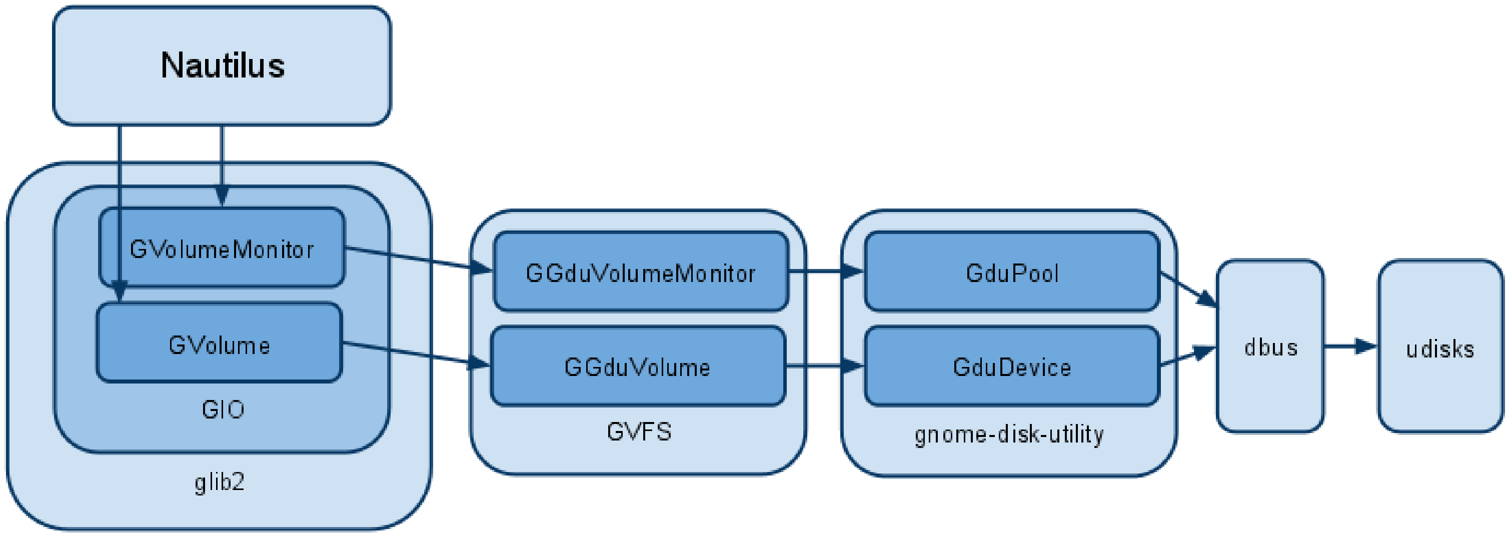
- **udev**
  - device manager for Linux
  - adds/remove entries in /dev
  - can trigger events based on rules or through a netlink socket
- **D-Bus**
  - IPC mechanism
  - allows applications to register for system device events
- **udisks**
  - provides a **D-Bus** interface for dealing with disk devices
  - uses **GUdev** library (part of **udev**) to subscribe to **udev** events through a **netlink** socket, republishes them through **D-Bus**

File systems in Linux

- Traditionally lived in `fs/` branch of kernel source tree
- File systems operate between low level disk bus drivers and virtual file system
- **FUSE** – file system in userspace
- **GVFS** – GNOME Virtual File System
  - not a traditional file system
  - can only be access through **GVFS**, **GIO**, or the `~/.gvfs` **FUSE** mountpoint

GNOME Nautilus

- File manager / browser for the **GNOME** desktop
- Uses **GVFS** to access browse file systems over SMB, FTP, DAV, etc
- Uses **GVFS** to be notified of newly mounted file systems



Auto mounting file systems

- Auto mount settings are configured through **gconf**
  - `gconftool -g /apps/nautilus/preferences/media_automount`
- Can also use the Folder Options dialog
- File systems on auto mounted device are determined through the use of "`mount -t auto`"
  - uses **libblkid** first
  - then tries each file system in `/proc/filesystems`
- Auto mounted file systems can also be auto browsed
  - `gconftool -g /apps/nautilus/preferences/media_automount_open`

Autorun capabilities

- Nautilus supports an AutoPlay-like ability to play CDs, DVDs, browse photos, etc
- Configured through **gconf** in **`/apps/nautilus/preferences`**
- Content type determined by using **`/usr/share/mime/treemagic`**
- Nautilus also supports executing files named **`autorun`**, **`.autorun`**, or **`autorun.sh`**!
  - –Fortunately there's no way to configure your system to run these automatically

Thumbnailers

- Nautilus uses **GdkPixBuf** for rendering image thumbnails
- Also supports using external thumbnailer applications
- Thumbnailers configured through **gconf**
  - `gconftool -R /desktop/gnome/thumbnailers`
- 3 thumbnailers configured by default
  - **`evince-thumbnailer`** for document files
  - **`totem-video-thumbnailer`** for audio and video files
  - **`gnome-thumbnail-font`** for font files

Putting this all together...

- Nautilus will automatically mount new file systems on USB sticks inserted into a PC
- Nautilus will open a window to browse that file system
- Nautilus will render icons for all files in the root directory of the file system that are visible
- Nautilus will use thumbnailer applications that could be full of old, insecure code for file formats that nobody uses
- A vulnerability in a thumbnailer could be exploited to unlock a 'locked' GNOME desktop

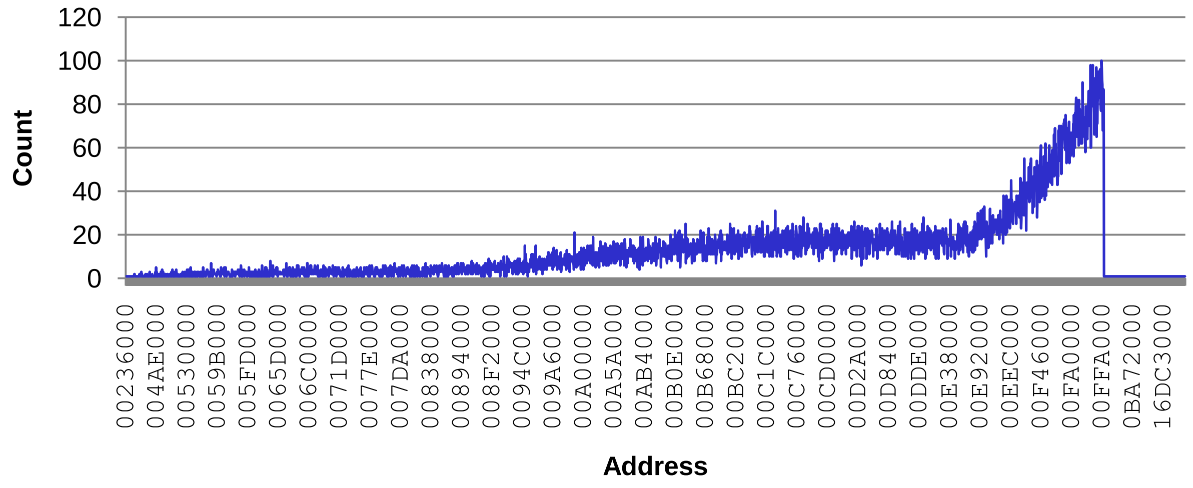
Exploiting thumbnailers

- Linux offers a few mitigation techniques
- On Ubuntu 10.10, we have **NX**, **ASLR**, and **AppArmor**
- **NX** can be defeated with return-oriented-programming (**ROP**) techniques
- **ASLR** can mitigate **ROP**
- Even if you can execute code, **AppArmor** limits what you can do to a system

Exploiting thumbnailers – ASLR?

- What about ASLR?
  - Brute force, since Nautilus doesn't care if a thumbnailer crashes
  - ASLR appears to be particularly weak in some cases:

Base address of `libc` per 40960 runs of `evince-thumbnailer`



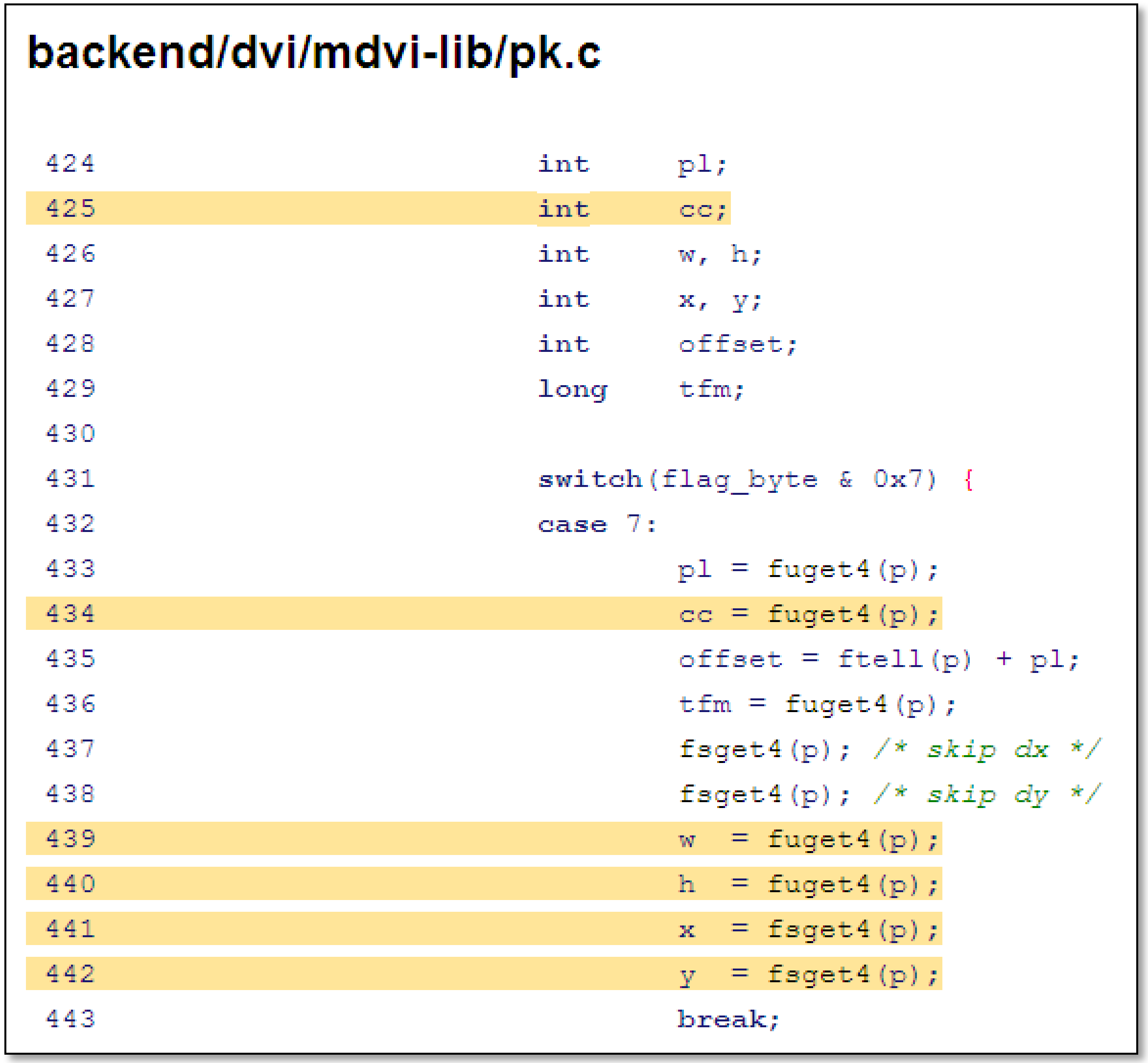
Exploiting thumbnailers – AppArmor?

- The only thumbnailer protected by **AppArmor** is **`evince`**
- **AppArmor** limits which files can be read and what can be written
- No launching of arbitrary processes
- Weaknesses in **AppArmor**
  - **`evince`** only allowed to read files with certain extensions, but a symlink will get around that
  - **`evince`**'s profile allows writes to certain parts of the user's home directory
  - **AppArmor** can't prevent evince from using the X11 **`XKillClient()`** API call to kill the screen saver window

evince vulnerabilities

- Vulnerabilities in handling external font files for DVI documents (CVE-2010-2640, CVE-2010-2641, CVE-2010-2642, CVE-2010-2643)
- DVI files can reference external fonts that get loaded when the DVI file is processed
- External fonts can be specified with an absolute path (`/media/XXX`)
- **AppArmor** will prevent loading a `.pk600` file, but creating a symlink from the `.pk600` file to a file ending in `.png` will get around this restriction

CVE-2010-2640

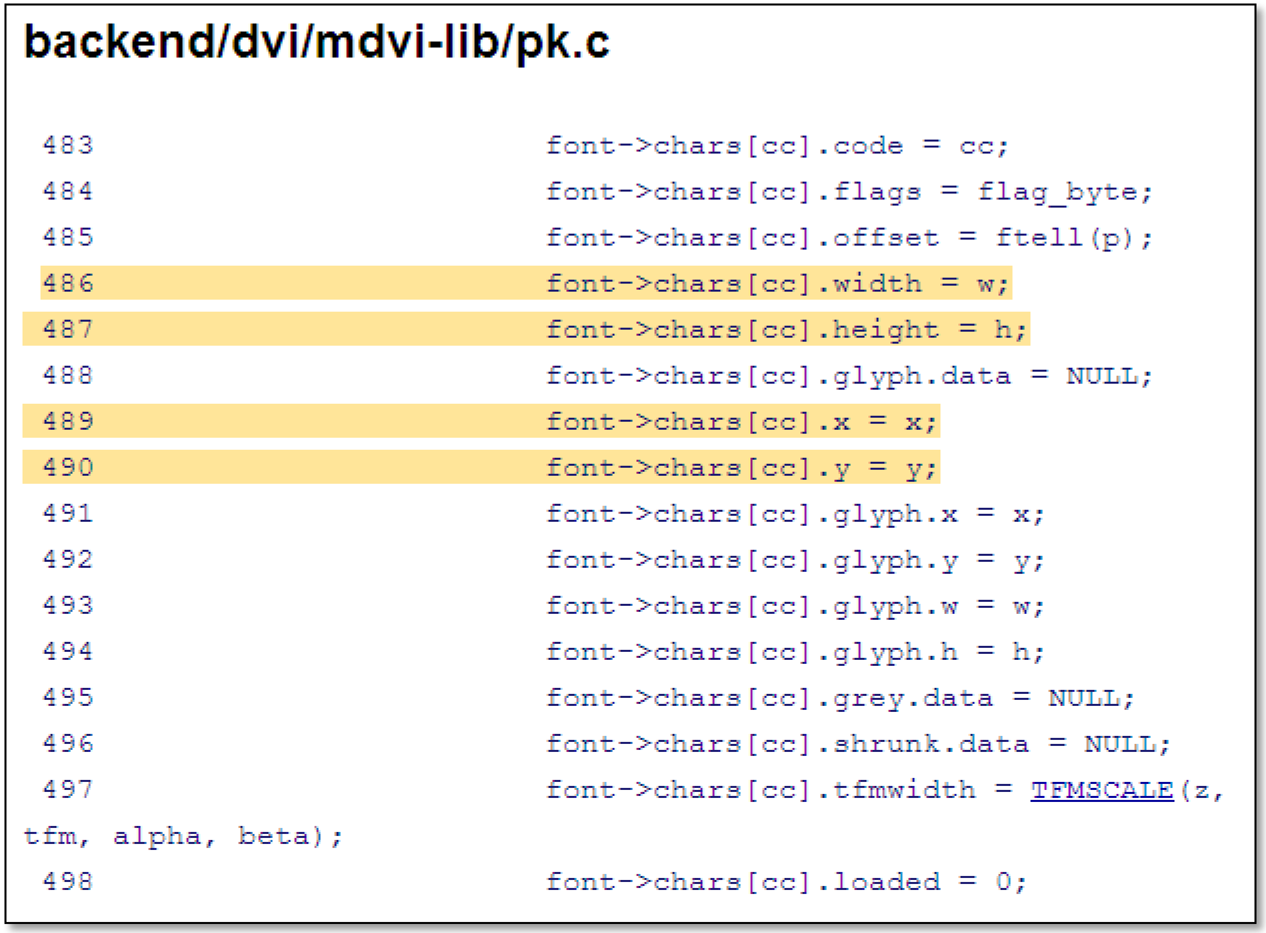


backend/dvi/mdvi-lib/pk.c

```
int     pl;
int     cc;
int     w, h;
int     x, y;
int     offset;
long    tfm;

switch(flag_byte & 0x7) {
case 7:
        pl = fuget4(p);
        cc = fuget4(p);
        offset = ftell(p) + pl;
        tfm = fuget4(p);
        fsget4(p); /* skip dx */
        fsget4(p); /* skip dy */
        w  = fuget4(p);
        h  = fuget4(p);
        x  = fsget4(p);
        y  = fsget4(p);
        break;
```

CVE-2010-2640

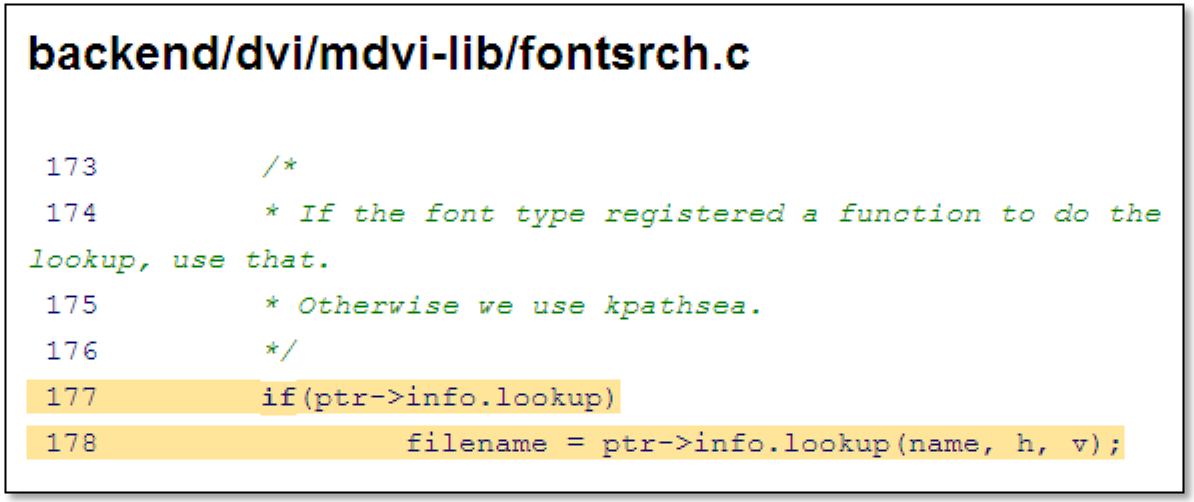


backend/dvi/mdvi-lib/pk.c

```
			font->chars[cc].code = cc;
			font->chars[cc].flags = flag_byte;
			font->chars[cc].offset = ftell(p);
			font->chars[cc].width = w;
			font->chars[cc].height = h;
			font->chars[cc].glyph.data = NULL;
			font->chars[cc].x = x;
			font->chars[cc].y = y;
			font->chars[cc].glyph.x = x;
			font->chars[cc].glyph.y = y;
			font->chars[cc].glyph.w = w;
			font->chars[cc].glyph.h = h;
			font->chars[cc].grey.data = NULL;
			font->chars[cc].shrunk.data = NULL;
			font->chars[cc].tfmwidth = TFMSCALE(z,
tfm, alpha, beta);
			font->chars[cc].loaded = 0;
```

CVE-2010-2640

- So we can write an arbitrary value to a semi-arbitrary location in memory
- The write is relative to the heap, so **ASLR** won't impact our ability to overwrite a function pointer on the heap
- What to overwrite?



backend/dvi/mdvi-lib/fontsrch.c

```
	/*
	 * If the font type registered a function to do the lookup, use that.
	 * Otherwise we use kpathsea.
	 */
	if(ptr->info.lookup)
		filename = ptr->info.lookup(name, h, v);
```

CVE-2010-2640

- We can overwrite `ptr->info.lookup` with the address of system
- name is a string representing the font file it's looking for
- To write this exploit:
  - –figure out what `cc` needs to be so that `w`, `h`, `x`, or `y` overwrites `ptr->info.lookup` for one of the fonts
  - –specify that `cc` value for the first font, and put the address in system in `w`, `h`, `x`, `y`
  - –for the 2nd font, speficy the name to be `/media/`*`XXX`*`/kill.sh`, where *`XXX`* is volume name of USB device
  - –`/media/`*`XXX`*`/kill.sh` can be a shell script to do whatever you want – mine kills the screensaver

Problems...

- **AppArmor** won't let you execute a process
- How do we get around this?
  - Write a ROP 2nd stage shellcode loader
  - **`mmap/open/read`**
  - **AppArmor** won't let you map executable files, but you can create an anonymous W+X mapping
  - 2nd stage shellcode can search for **X11** library, use **X11** APIs to enumerate root windows then kill the topmost one (it's the screensaver)
  - Still working on it...

Demo!

D E M O

DEMO

DEMO

Conclusion

- There are more ways than **AutoRun** to execute code on a USB flash drive
- A lot of these can be pre-emptively mitigated by disabling the features of your OS
- Epoxy those USB ports! (and IEEE1394, eSATA, PC-CARD/CardBus, memory cards, CD/DVD drives...)
- Questions?