DLL Spoofing in Windows

By Andreas Björklund, Johan Klövstedt and Sven Westergren

What is spoofing?

A program run under Windows gains the full capabilities of the user that runs it. If an attacker without certain capabilities creates a malicious program and coaxes a second user with these capabilities to run the program he can expand his capabilities to include those of the second user. This process is generally known as "spoofing".

What is a DLL?

Dynamic Link Libraries (DLL's) are software object modules, or libraries, linked into a program while it is running. DLL's are powerful features that allows for programs to share common code making them easier to develop and make them more efficient. They are used extensively in newer versions of Windows, including Windows NT.

What is DLL spoofing?

Since the DLL code runs in the context of its host program it inherits the full capabilities of the program's user as discussed earlier with spoofing in the general sense. The DLL spoof causes a legitimate program (in the worst case scenario run by an administrator) to load a DLL with a Trojan Horse instead of the legitimate DLL. Once it gains control it has the same capabilities as the user who ran the program.

When a program loads DLL's, it searches through a sequence of directories looking for the DLL in question. The successful spoof occurs when an attacker succeeds in inserting the malicious DLL-file in one of those directories so that the program finds it before it finds the legitimate DLL of the same name. This means that the attack can still be successful even if the legitimate DLL is beyond the attackers reach (i.e. if the file is write-protected or the attacker doesn't have access to the directory which contains the legitimate DLL).

The opportunities for DLL spoofing stem from the algorithm used by the DLL linking algorithm to find the file that holds the DLL (usually with ".DLL" as the file suffix). The linking algorithm usually searches through the following three categories:

- Program directory: The directory that holds the program's executable file.
- System Directory: A well known system directory like %SYSTEMROOT% or its SYSTEM32 subdirectory.
- Working directory: The current working directory of the process.

The problematic case is when the algorithm checks the Working Directory for the DLL file. To spoof a user an attacker only need to insert a malicious DLL file in a directory that the users use as their working directory. If the DLL file inserted has the same name as the legitimate DLL the algorithm will link the fake DLL to the otherwise trusted program. The fake DLL can then create a new process, which will now run under the full capabilities of the user who ran the program, which can request the real DLL to make the program perform the required action so as to not arouse suspicion. Depending on what the fake DLL does, the attacker can now start to take over the system with his acquired capabilities.
The Program Directory and System Directory are well-known and can thus be protected more easily. But in the case of the Working Directory its location can be unknown to the user since the program itself may set the directory.

Targets

The most obvious targets for DLL spoofing attacks are machines running versions of Windows NT/2000 where the

registry has not been properly updated with a safe search-order for loading DLL's. Service packs updates have been issued that rectify the search order but for reasons of backwards compatibility it is disabled as per default.

For machines running Windows XP the problem with the search-order is not an issue but there have been a few mal-configured programs and registry entries which point to DLL's that do not exist. Such entries can also be exploited to make the system run malicious code by DLL spoofing. Obviously, simply having a mis-configured registry/search path does not mean that the machine will start running malicious code; an adversary must also gain access to the file system and place the code there. This can be achieved in a number of different ways, including Trojans, email and web caches to just name a few.

One can argue that this breach is in fact more serious than the DLL spoof but for NT/2000 machines an ordinary user could easily place malicious DLL's in for example the "Shared Documents" directory. When another user with higher privileges subsequently opens a document in the same directory this will be the "Current directory" and will be searched for DLL's before the system directories, allowing the ordinary user to run code with privileged rights.

For machines running Windows XP or properly updated NT/2000 the argument holds more weight. Simply placing a DLL in the shared directory or a web cache will not allow it to be loaded. For the DLL's to be loaded they must either be placed in the system directory, the application directory or a place pointed to by an absolute path in the application that tries to load the DLL. Being able to write to system and application file space already implies administrator privileges so there would be no need for DLL spoofing which leaves the case with absolute paths in the application. This however is a question about auditing and securing the software being used on the system and must be considered paramount for security administrators in any case.

Problems/Solutions

We all know that many operating systems, like Microsoft Windows, install auxiliary services that are not critical, such as FTP server, telnet and web server. If those services and all other services, defined by the administrator as not needed, are removed then the threat list is at once decreased. With DLL spoofing it is not all about shutting down and not installing, the countermeasure towards the actual spoofing would be the use signed DLL’s. For this purpose Microsoft (Who as already mentioned seems to have the greatest problem with DLL spoofing) tries to solve it by the use of their Microsoft Authenticode (Multi-Purpose) Certificates. These signings are also intended for code. First off we need to define what a manifest is to help the reader understand how Microsoft deals with the need for continuously updating their DLL’s as well. To let every DLL to become outdated could be dangerous in this world of hackers. A manifest is a descriptive file with a .mcf extension. This file consists of the GUID for the use of the file, the module list which could be explained as the requirement list for the file (which modules needed, and if they need hashing or not) and also a policy list of what should be included and excluded. By designating the DLL’s as NOHASH in the MODULELIST section, while including the Microsoft code-signing root public key into the inclusion, one could specify Microsoft Authenticode-signed modules in a manifest. Where a NOHASH value is applied to the DLL in the case Microsoft releases a new version of the DLL, you don’t have to create a new manifest. It is enough to just update the old one.

The question as to how we know which DLL's that are up-to-date, Microsoft solved with Windows 2000, by digitally sign the drivers that pass the Windows Hardware Quality Lab (WHQL) tests. The drivers that passed were given a Microsoft digital signature. Such devices covered then were: Video adapter, Keyboard, Monitor, Printer, etc. To do the actual signing of the drivers, the existing digital-signature cryptographic technology was used. A hash of the driver binary and relevant information was stored in a catalogue file (CAT file), and that file was signed with the Microsoft signature. Since the actual driver binary is left untouched, the reference between the driver package and the CAT file that would be needed was solved using a third file to do the referencing. That file is called the driver's INF file and is maintained by the system after the driver has been installed.

But as mentioned, in the present time this signing is done with Microsoft Authenticode (Multi-Purpose) Certificates. These signings could be done by an authorized signer such as Thawte. Thawte can help out with the authentication that is needed for the third-party Authenticode-signed modules. As Microsoft states themselves "The Rights Management client (uses the Windows Rights Management Service, RMS) supports Authenticode module signing using both SHA1 and MD5, however SHA1 results in better performance". However they do not provide any tools for extracting or producing third-party keys for use in the inclusion/exclusion list (in the policy list). In the mid-nineties when the approach with the RMS, sometimes referred to as the Digital Rights Management (DRM), was explored designers came up with a variety of tamper resistance. Often, however, they concluded that even though a particular approach may seem effective, only Microsoft would have the resources, scope and platform control to make it practical.

Now, ten years later, details are coming out about Microsoft's plans for DRM protection in the upcoming Windows Vista OS, which indeed contain a lot of tamper resistance mechanisms. Then it would seem they were right ten years ago.

Here are two concepts that stand out, which contain some handling of drivers:

- Protected Path: More specifically known as Protected Video Path (PVP) and Protected User Mode Audio (PUMA). These are mechanisms to support DRM rules about safe content presentation. Many of the protected path mechanisms are implemented in kernel-mode device drivers.

- Protected Environment: This is a kernel mechanism to ensure that kernel-mode drivers are safe for protected content. Drivers that handle protected content must be digitally signed and authorized by Microsoft, and must implement specific security functions. Other kernel-mode drivers must also be signed, to ensure that they have a known origin and have not been tampered with. Unless all these signature and authorization checks pass, protected content cannot be played. Some Output Content Protection (OCP) mechanisms are planned for Windows Vista in 2006 (the basic video and audio protections, and the protected environment).

At a higher level, OCP's Protected Path and Protected Environment ideas seems to make sense. In implementation, however, there is a great deal of software complexity, management process, and supporting infrastructure. One software complexity regarding drivers then becomes that the Protected Environment requires that drivers be both authentic and sound. This again puts Microsoft in the business of analyzing, testing, and authorizing new

driver versions--not just determining that they appear not to break Windows or cause incompatibility problems (which are the main goals of WHQL testing today), but that they provide a secure implementation of required protection features. Also to implement OCP would mean that device drivers get numerous new security responsibilities. The drivers must ensure that the hardware they control is authentic and isn't tampered with. (E.g. the tilt-bits which are used like tilt on a pinball machine.)

But to go back to DLL's (device drivers), another complexity is revocation. Authorization is not particularly useful unless it can be revoked when a compromise is discovered.
For making sure that revocation would work, Microsoft plans to run a revocation infrastructure that distributes a Microsoft Global Revocation List to identify no-longer-authorized driver software. This feature is supposed to be implemented in the Windows Update mechanism. Drivers will be added to the list if they are discovered to contain exploitable security flaws, and DRM rules can be written to require that the list is appropriately fresh (actually DRM already contains this type of capability). Software revocation is problematic because of the potential effect on customers who may suddenly be unable to play content through no fault of their own, so revocation will likely only occur well after updates are distributed, leaving a long window of content vulnerability.

References:

Authenticode Signing a DLL:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rms_sdk/rm/authenticode_signing_a_DLL.asp

Code Signing for Authenticode, Netscape and Sun Java:

http://www.jensign.com/JavaScience/Thawte/

Microsoft moves forward on DRM:

http://news.com.com/2100-1012_3-5071342.html

Microsoft Previews DRM Directions for Windows Vista:

http://www.drmwatch.com/special/article.php/3529586

Windows NT Security Guidelines:

http://www.trustedsystems.com/download/NSAGuideV2.PDF