REPORT

FireEye

DLL SIDE-LOADING:

A Thorn in the Side of the Anti-Virus Industry

Author: Amanda Stewart

SECURITY REIMAGINED

CONTENTS

Abstract 3

DLL Side-loading Explained 3

WinSxS detailed 3

What This All Means For DLL Side-loading 5

Malware volume 5

Malware detection 5

The long-tail theory of malware distribution 6

PlugX and DLL Side-loading 7

How to Recognize DLL Side-loading Vulnerability 8

Samples exhibiting similar behavior 9

Recommendations 10

Software developer 10

QA analyst 11

Endpoint user 11

Call To Action 11

About FireEye, Inc. 11

Abstract

Dynamic-link library (DLL) side-loading is an increasingly popular cyber attack method that takes advantage of how Microsoft Windows applications handle DLL files. In such attacks, malware places a spoofed malicious DLL file in a Windows' WinSxS directory so that the operating system loads it instead of the legitimate file.

This paper describes the history of DLL Side-loading and its role in the malware and software engineering arenas. It also examines evolving trends along with similarities and differences between DLL Search-Order Hijacking, DLL-Hijacking, DLL pre-loading, and DLL side-loading.

A technical analysis of the Trojan PlugX variant used to target Chinese political rights activists shows the DLL-side-loading technique in action. Finally, the paper recommends preventative measures to ensure that legitimate files are not exploited.

DLL Side-loading Explained

Windows, like many operating systems, allows applications to load DLLs at runtime. Applications can specify the location of DLLs to load by specifying a full path, using DLL redirection, or by using a manifest. If none of these methods are used, Windows attempts to locate the DLL by searching a predefined set of directories in a set order.[1]

Cyber attackers have long abused this search feature by placing a malicious DLL in one of these directories. In these attacks, Windows reaches and loads the malicious DLL before finding the legitimateversion. The earliest such attacks (such as those exploiting CVE-2000-0854) appeared as early as 2000.[2]

A less common variant of this technique called DLL side-loading has been trending in recent attacks.[3,4,5] DLL side-loading takes advantage of Windows' side-by-side (SxS or WinSxS) assembly feature, which helps manage conflicting and duplicate DLL versions by loading them on demand from a common directory.

Traditionally, search-order hijacking attacks utilize an executable file's DLL search path to load spoofed DLLs through the known DLLs record. This record comprises a list of known DLLs on the current system, stored in the following registry key:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs.

DLL side-loading, in contrast, utilizes the WinSxS assembly to load the malicious DLL from the SxS listing, which is located in the following registry key:

%TEMP%\RarSFX%\%ALLUSERS PROFILE%\SXS\ or
%TEMP%\RarSFX%\%ALLUSERS PROFILE%\WinSxS\

WinSxS detailed

WinSxS mainfests, which are embedded in the executable as XML data, describe dependencies and libraries used by the application. The manifests contain the resource and library metadata. WinSxS is designed to give developers flexibility to update binaries by easily replacing the old binaries in the same location.

[1] Microsoft. "Dynamic-Link Library Security."

[2] National Vulnerability Database. "Vulnerability Summary for CVE-2000-0854." September 2008.

[3] Amanda Stewart. "Targeted Attack Trend Alert: PlugX the Old Dog With a New Trick." May 2013.

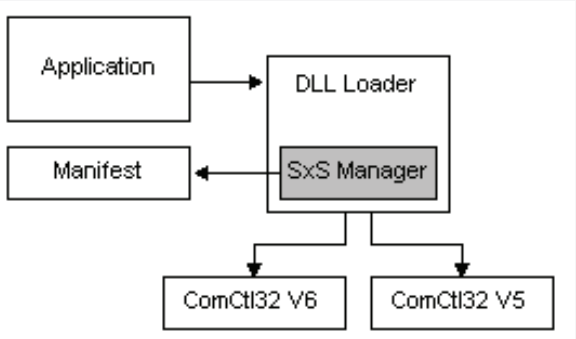
[4] Gabor Szappanos. "Targeted malware attack piggybacks on Nvidia digital signature." February 2013.

[5] Abraham Camba. "Unplugging PlugX Capabilities." September 2013.

Benefits of WinSxS include the following:

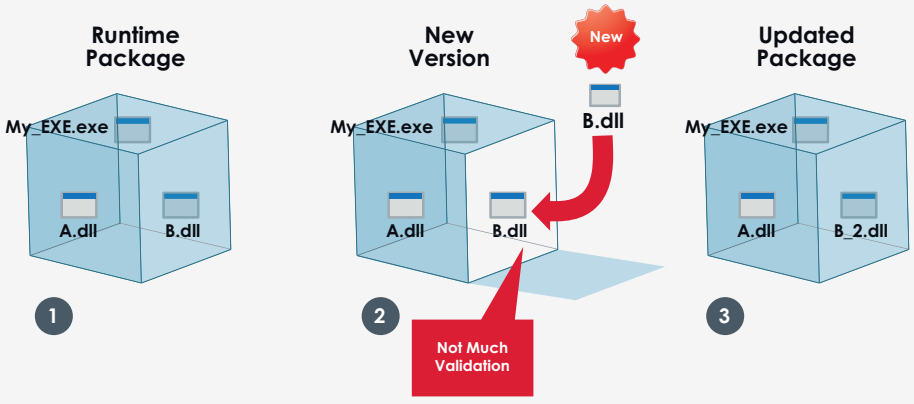
- Reduces the possibility of DLL version conflicts
- Enables sharing of multiple versions of COM or Windows assemblies to run simultaneously
- Updates assembly configuration on either a global or per-application configuration basis after deployment

Figure 1: Representation of typical side-by-side assembly[6]



The problem with this technique is that it offers little to no validation of the loaded DLL other than what is explicit in the manifest's DLL metadata (see Figure 2). This omission may inadvertently grant trusted installer privileges to malicious payloads.

Figure 2: Search for DLL within the executable run path



[6] Microsoft. "Side-by-side Assemblies." July 2010.

What This All Means For Dll Side-loading

When polymorphic malicious payloads are delivered with legitimate executables, endpoints face greater risks. With WinSxS, malware can bypass anti-virus scanners for a longer period.

This dynamic has several implications for the volume, detection, and variety of malware deployed by threat actors worldwide.

Malware volume

Security professionals' main challenge can be summed up with a simple question: How can we tackle clusters of malware quickly?

Malware is mushrooming. According to one study, the volume of malware samples in the wild has grown 60 percent since 2010.[7]

Even as the number of unique malware samples rises, they are growing more difficult to detect. Anti-detection techniques such as binary packing, compression, encryption, compiler variation, and polymorphism have made malware harder to identify.

To keep up with these dual challenges, anti-virus vendors mix automated and manual techniques to generate new malware signatures.

Malware detection

Most signature creation and detection techniques fall into one of the following categories:

- Basic whole-file hash generation. The most common signature-creation technique involves generating a basic hash value for the file as a whole. This technique uses whole-file hashes to create blacklists and whitelists. (One such example is the NSRL database, which contains hashes of wellknown legitimate binaries.) AV products then compare the hash values of questionable files against these lists to identify them as malicious or benign.

- Section-based hashes. The second signature-creation technique splits the binary into sections and generates hash signatures for each section. A common example is portable executable (PE) sections, in which hash values are generated from the binary's PE sections (such as .text, .rdata, and data) and size.

- Code-section hashes. The third-most common technique involves hash values based on code sections of the malware sample.

All three of these static signature-generation techniques can be used in automation or manual analysis. Regardless of which technique comes into play, signature-based detection is quickly reaching its practical limits when it comes to the central challenge of finding malware clusters quickly.

To understand why, consider how security vendors generate malware signature. Signature generation starts with sample origin. This sample origin can be a specific malware family or previously seen malicious code that has spawned the need for the signature.

Attackers' anti-detection techniques (such as code-morphing) can generate a large volume of

unique binary samples from the same malware family—each with a unique hash value. Because signature matching is limited to specific hash

[7] Sophos. "Year in Review: 2011." December 2011.

samples, timely detection becomes less and less possible for as the volume o f newly introduced unique samples grows.

This increasingly larger volume of unique samples also becomes more difficult to cluster based on static techniques because of a phenomenon known as the long-tail theory.

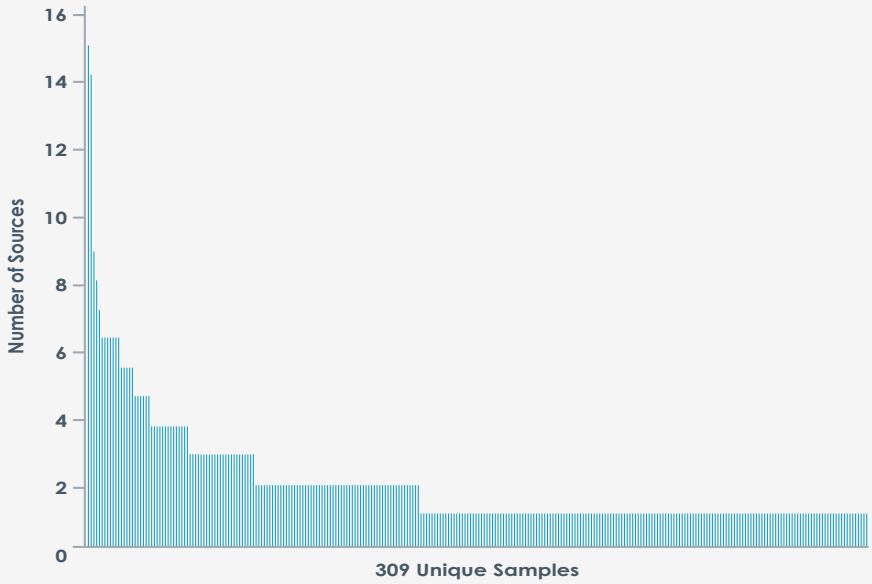
The long-tail theory of malware distribution

The long-tail theory describes the unique distribution of occurrences. Applied to malware, the theory explains why many samples may only appear once and never occur again.

The PlugX family of malware, which uses DLL side-loading techniques to infect target machines, demonstrates the long-tail theory in action.

Figure 3 shows a collection of unique PlugX related samples discovered within the last year. Of those samples, only 46 percent were uploaded to VirusTotal. While this example represents a very small sampling, the long-tail phenomenon has held true in broader FireEye detection statistics.

Figure 3: Sample distribution.



FireEye

PlugX And DLL side-loading

In 2013, FireEye Labs discovered a spear-phishing attack targeting Chinese political rights activists. This email contained an attachment exploiting a vulnerability in Windows ActiveX controls (CVE-2012-0158) to drop several binaries that appear benign in isolation but combine to form a malicious executable. In Figure 4, the green highlighted icon represents the seemingly benign executable that contains the DLL side-loading vulnerability.

OINFO11.exe (hash value: a31cad2960a660cb558b32ba7238b49e) originated from an Office 2003 Service Pack 2 update. In this attack, this sample loads a spoofed DLL component (Oinfo11.ocx). When Oinfo.ocx is loaded into memory, it loads, decompresses, and decodes a secondary component (OInfo.ISO).

These two malicious payloads combine to form the DLL that exists only in the benign executable's memory. This distinction is crucial—OINFO11.exe's hash value is listed in the National Software Reference Library (NSRL) database, which means it is listed in a publicly used binary whitelist.

OINFO11.exe (hash value: 31cad2960a660cb558b32ba7238b49e) originated from an Office 2003 Service Pack 2 update. In this attack, this sample loads a spoofed DLL component (Oinfo11.ocx). When Oinfo.ocx is loaded into memory, it loads, decompresses, and decodes a secondary component (OInfo.ISO).

These two malicious payloads combine to form the DLL that exists only in the benign executable's memory. This distinction is crucial—OINFO11.exe's hash value is listed in the National Software Reference Library (NSRL) database, which means it is listed in a publicly used binary whitelist.

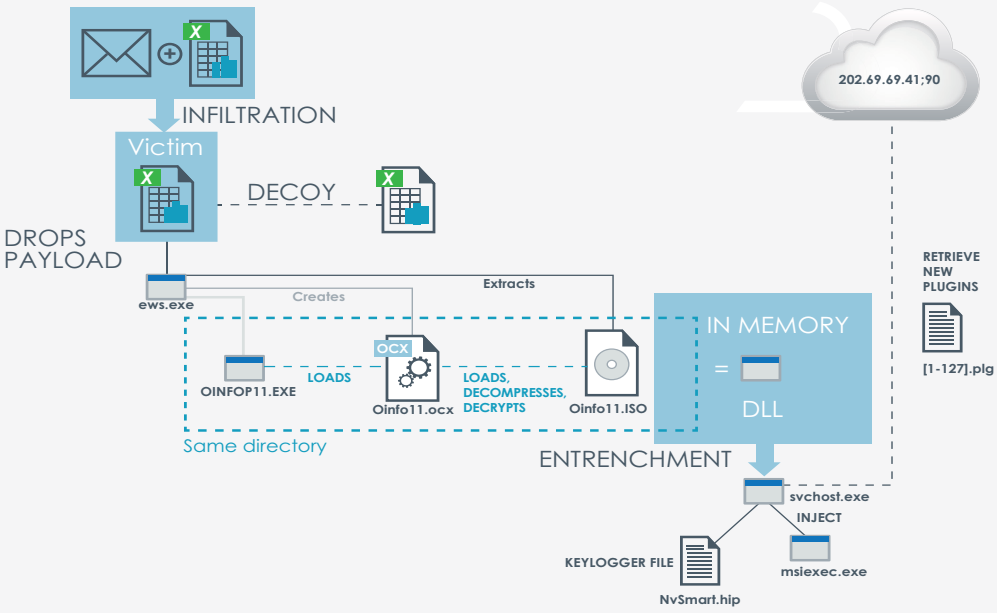
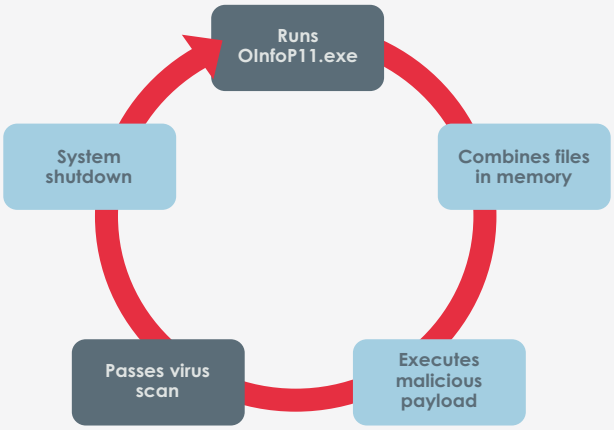


Figure 4: PlugX targeted attack

Figure 5: PlugX flow



Some anti-virus software may utilize this database to ignore benign executables and reduce false positives. Because the malicious payload exists only in memory, the sample is not detected or removed, and the attack persists (see Figure 5).

How To Recognize DLL Side-loading Vulnerability

Security professionals have several methods of examine software for the DLL side-loading vulnerability. This section explains the fastest and easiest method: validating the DLL imports.

Figure 6 shows the OINFO11.exe DLL import table, which includes the functions GetOfficeData and DeleteOfficeData. Any file that is loaded from the side-by-side directory and adjacent to the primary executable should be validated for these functions. Usually, executables using the side-by-side feature will have these resources located in the embedded manifest file.

Validating the file and functions must involve more than simply checking for the correct filename and functions names.

Figure 6: DLL Exports for OINFO11.exe

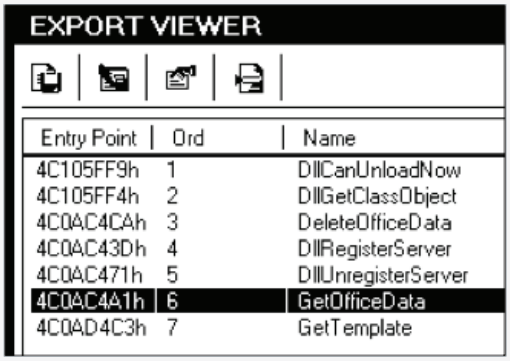


Figure 7: Original exported function

```
.text:4C0AC4A1 ; int __cdecl GetOfficeData(int, int lpUserNameBuf, LPCSTR lpOrgNameBuf)
.text:4C0AC4A1                 public GetOfficeData
.text:4C0AC4A1 GetOfficeData   proc near
.text:4C0AC4A1
.text:4C0AC4A1 arg_0           = dword ptr  4
.text:4C0AC4A1 lpUserNameBuf   = dword ptr  8
.text:4C0AC4A1 lpOrgNameBuf    = dword ptr  0Ch
.text:4C0AC4A1
.text:4C0AC4A1                 push    [esp+arg_0]
.text:4C0AC4A5                 push    offset aGetofficedataD
.text:4C0AC4AA                 call    nullsub_1
.text:4C0AC4AF                 pop     ecx
.text:4C0AC4B0                 pop     ecx
.text:4C0AC4B1                 push    0               ; int
.text:4C0AC4B3                 push    [esp+4+lpOrgNameBuf] ; lpOrgNameBuf
.text:4C0AC4B7                 mov     ecx, offset unk_4C133848
.text:4C0AC4BC                 push    [esp+8+lpUserNameBuf] ; lpUserNameBuf
.text:4C0AC4C0                 push    [esp+0Ch+arg_0] ; int
.text:4C0AC4C4                 call    sub_4C0C8407
.text:4C0AC4C9                 retn
.text:4C0AC4C9 GetOfficeData   endp
```

Figure 8: Spoofed exported function

```
.text:10001000                 public GetOfficeData
.text:10001000 GetOfficeData   proc near
.text:10001000                 retn                    ; DeleteOfficeData
.text:10001000 GetOfficeData   endp
.text:10001000
.text:10001000
```

Take OINFO11.exe, for example. Compare the functions of the original supplementary file Oinfo11.ocx and a spoofed version. Figure 7 shows the original assembly for GetOfficeData. In the spoofed version (Figure 8), the same function does nothing.

Samples exhibiting similar behavior

Table 1 lists executables from large corporations that have been used in APT PlugX attacks that occurred before the OINFO11.exe attacks. Attacks that use these files do not always mirror the DLL-side-loading methods exactly, but the concept is the same

Filename	MD5Sum	Detail
mcvsmap.exe	4e1e0b8b0673937415599bf2f24c44ad	McAfee
NvSmart.exe	09b8b54f78a10c435cd319070aa13c28	NVIDIA Corporation
RASTLS.EXE	62944e26b36b1dcace429ae26ba66164	Symantec Corporation

Table 1: Files commonly used in PlugX attacks

Recommendations

Developers, quality assurance analysts, and endpoint users can help prevent or mitigate DLL-side-loading attacks in a number of ways.

Software developer

For the software developer, FireEye recommends the following mitigation techniques when loading updated DLLs for packaging:

- Ensure that the full path is hardcoded. Avoid using relative paths for any resources.
- Confirm that the imported DLL actually exists.
- Ensure that imported functions are valid. As noted from the PlugX sample, the spoofed function was simply empty.
- Ensure that the operating system is correct.
- Utilize DLL redirection or a manifest. Recent versions of Visual Studio enable developers to create manifests to ensure that the loaded library is valid.

Here is an example of this manifest file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity publicKeyToken="75e377300ab7b886" type="win32"
name="Test4Dir"
version="1.0.0.0" processorArchitecture="x86"/>
<file name="DirComp.dll" hash="35ca6f27b11ed948ac6e50b75566355f0991d5d9"
hashalg="SHA1">
<comClass clsid="{6C6CC20E-0F85-49C0-A14D-D09102BD7CDC}" progid="DirComp.
PathInfo"
threadingModel="apartment"/>
<typelibtlbid="{AA56D6B8-9ADB-415D-9E10-16DD68447319}" version="1.0"
helpdir=""/>
</file>
</assembly>
```

The file name DirComp.dll is accompanied with a hash for validation. This may seem a simple concept, but it can provide a minimum check on DLL validation.

- Call SetDllDirectory with an empty string. Recommended by Microsoft, keeping the string parameter empty disables the safe DLL search mode and prevents automatic DLL loads by API calls to LoadLibrary.[8]

[8] Microsoft. "SetDllDirectory function."

Table 2: DLL import validation tools

PE explorer	http://www.heaventools.com/overview.htm
Dependency Walker	http://www.dependencywalker.com/
SxStrace.exe	(Found in MS Vista) Validate manifests and DLL tracing

QA analyst

Quality-assurance analysts can use any of the tools listed in Table 2 to check the DLL imports of executables.

Endpoint user

FireEye advises endpoint users to ensure that all validated and clean applications are installed in administrator-protected directories. This step restricts write and execute permissions to user folders and implements least-privilege access.

Call To Action

Software publishers must remain alert to any DLL-side-loading vulnerabilities in their products. Staying aware of this potential attack vector and heeding the recommendations outlined in the previous section can help reduce opportunities for malware authors to use them for hard-to-detect malware.

About FireEye

FireEye has invented a purpose-built, virtual machine-based security platform that provides real-time threat protection to enterprises and governments worldwide against the next generation of cyber attacks. These highly sophisticated cyber attacks easily circumvent traditional signature-based defenses, such as next-generation firewalls, IPS, anti-virus, and gateways. The FireEye Threat Prevention Platform provides real-time, dynamic threat protection without the use of signatures to protect an organization across the primary threat vectors and across the different stages of an attack life cycle. The core of the FireEye platform is a virtual execution engine, complemented by dynamic threat intelligence, to identify and block cyber attacks in real time. FireEye has over 1,500 customers across more than 40 countries, including over 100 of the Fortune 500.

FireEye, Inc. | 1440 McCarthy Blvd. Milpitas, CA 95035 | 408.321.6300 | 877.FIREEYE (347.3393) | info@fireeye.com | **www.fireeye.com**