WHITEPAPER

DRAGOS
SAFEGUARDING CIVILIZATION

# Suggested Practices to Defend Against DLL Hijacking

## Technical Insights & Vulnerability Analysis

SAM HANSON | SENIOR VULNERABILITY ANALYST
DRAGOS, INC
DECEMBER 2024

## TABLE OF CONTENTS

# Executive Summary

Dragos researchers consistently encounter DLL hijacking, a prolific vulnerability that abuses a feature in the Microsoft® Windows operating system and tricks a vulnerable application into executing attacker-created code. This flexible technique can be used to gain initial access, escalate privileges, evade detection, and gain persistence on a host system. While there are many variants of DLL hijacking, all require tricking the Windows operating system into loading and executing a malicious, attacker-created DLL masquerading as a legitimate dependency.

Dragos maintains a database of industrial control systems (ICS) vulnerabilities. Dragos is aware of 98 DLL hijacking vulnerabilities impacting industrial software. The vast majority (90%) reside in Levels 2 and 3 of the Purdue Model. They are often found in device configuration and programming software on engineering workstation systems (EWS).

While recreating an exploit for a given DLL hijacking vulnerability is trivial, defense is doable—vulnerability detection is relatively quick, and mitigations exist for defenders that make exploitation harder. However, the root cause lies with the software developers, and it is important for operational technology (OT) vendors to be aware of the secure programming practices provided by Microsoft, which are highlighted in this paper, to reduce the occurrence of this vulnerability in their products.

# What Is DLL Hijacking?

To understand DLL hijacking, it's important to know what DLL files are and how Windows manages the loading and execution of these dynamic libraries.

## Dynamic Link Library (DLL) Files

A dynamic link library (DLL) is a Windows file type for storing shared or commonly used code. Any number of Windows programs and even other DLLs can import and use the code stored in a DLL file. DLLs are advantageous for many reasons, including "modularization of the code, code reuse, efficient memory usage, and reduced disk space."[1]

DLLs are commonly referred to as dependencies since programs may depend on them for certain functions and are, therefore, required for successful execution. For example, a program may depend on the Microsoft-provided user32.dll to build graphical user interfaces. Programmers can build shared libraries for their own applications as well. Consequently, a given program must be bundled with its dependencies or assume the file already exists on the user's computer.

Common system-provided Windows shared libraries can be found in a couple of locations, depending on program and operating system architecture:

- 32-bit - C:\Windows\SysWOW64
- 64-bit - C:\Windows\System32

For Windows on ARM, shared libraries can be found in:

- C:\Windows\SysArm32

With hybrid binaries located:

- C:\Windows\SyChpe64

1 **What is a DLL** – learn.microsoft.com

## Dynamically Loading Libraries

A shared library can be linked to a program dynamically or statically. Static linking refers to a binary that embeds all external dependencies directly inside the program itself. Statically linked binaries are not vulnerable to DLL hijacking. If a library is not statically linked to the application, programs have two methods for loading a DLL file: load-time dynamic linking and run-time dynamic linking.

In load-time linking, the operating system loads the DLL before the program's main function is executed. To take advantage of load-time linking, the programmer declares which libraries are needed at compile time.[2]

In run-time linking, code is executed that explicitly loads the shared library file and invokes functions using Windows API calls such as **LoadLibrary("library_name.dll")** and **GetProcAddress(...)** functions. Run-time linking is faster on application start-up and gives programmers the flexibility to conditionally load DLLs depending on the environment or user requirements.[3]

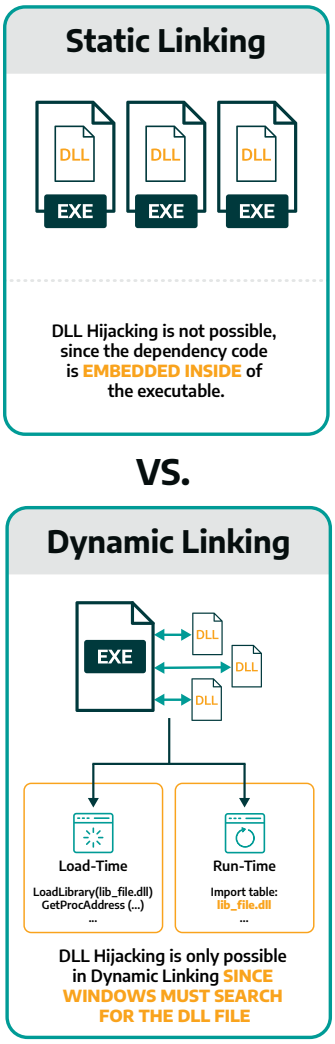



Figure 1: Techniques to load a DLL file; DLL hijacking is only possible in dynamic linking.

## The Windows Search Order

For a program to load a DLL, the file must be located on the Windows filesystem. To find where the file is stored, the operating system conducts a search by traversing through a default set of directories searching for the file by name: [4]

1. The directory from which the application loaded.
2. The system directory.
3. The 16-bit system directory.
4. The Windows directory.
5. The current directory.
6. The directories that are listed in the PATH environment variable.

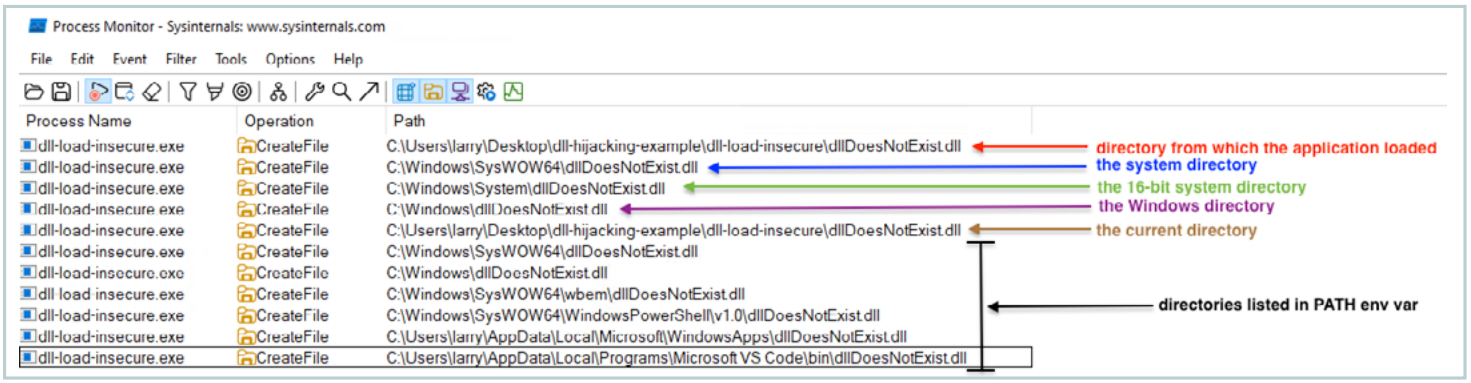



Figure 2: Process Monitor demonstrating Windows search order of an attempted load of a non-existent file

2 **Load-Time Dynamic Linking** – learn.microsoft.com
3 **Run-Time Dynamic Linking** – learn.microsoft.com
4 **Dynamic-Link Library Security** – learn.microsoft.com

That is, if the DLL file is not found in the same directory from which the application loaded, it will then check the system directory, followed by the 16-bit system directory, the Windows directory, and so forth, until each of the six locations has been checked. However, this simple search algorithm, often referred to as the Windows search order algorithm, yields a hijacking opportunity.

**Analyst Note:**

The Windows search order is more complicated than explained above; alternate search orders can be enacted, such as safe DLL search, DLL redirection, API sets, SxS manifests, KnownDLLs, and more. However, this whitepaper focuses on the search paths where DLL hijacking is possible and will discuss the alternate search order options only from a mitigation standpoint.

## How DLL Hijacking Happens

Applications loading a shared library can be hijacked into loading a malicious file if, during the search algorithm execution, the malicious file is found before the legitimate DLL file. That is, if an adversary places a DLL file in a directory that is searched prior to the actual directory that holds the legitimate DLL, or if the DLL does not exist at all, then the DLL will be loaded and executed inside the program, allowing the adversary to gain code execution with the same privileges as the user who initially ran the program.

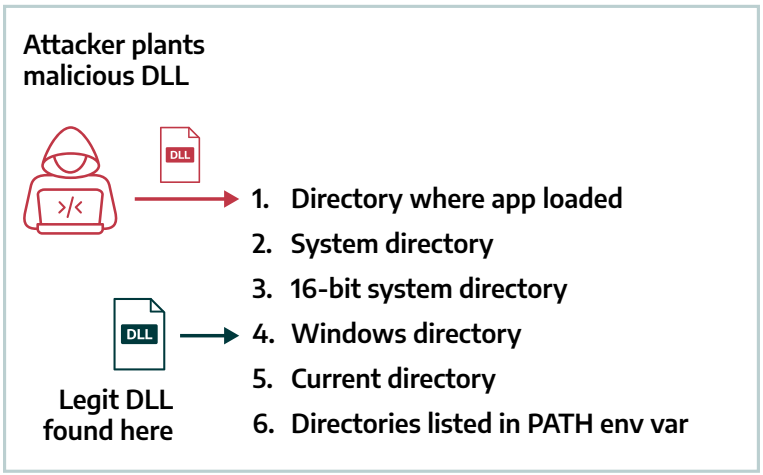



Figure 3: Malicious DLL planted in location searched prior to where legitimate DLL resides.

Many sub-variants of DLL hijacking exist, such as DLL replacement, phantom DLL hijacking, DLL side-loading, and DLL redirection hijacking. They differ by slight technical details on how the DLL is hijacked over the benign file. However, all variants refer to the same outcome – tricking the loader into executing an attacker-created DLL file. Dragos has defined each of the variants, noting their differences, in Appendix A.

## Attack Methods Leveraging DLL Hijacking

DLL hijacking is an extremely versatile attack methodology. With it, attackers can gain initial access to an organization, escalate privileges, persist on a host system, and remain undetected. Dragos will provide an example scenario for each attack use case in the following sections.

### Initial Access via Drive-by-Download

If a victim unknowingly downloads a malicious DLL while browsing the web, which is saved to the all-too-commonly-messy **Downloads** folder, and later runs a vulnerable application from the **Downloads** folder, then an attacker will gain initial access and arbitrary code execution on the victim's machine.

Installer executables, such as those generated by Revenera (previously named Flexera) InstallShield, are especially serious in a drive-by-download scenario. Dragos discovered that the installer for AVEVA's Edge HMI/SCADA software was vulnerable to DLL hijacking in mid-2021.[5] After further investigation, it was discovered that a vulnerable version of InstallShield was used to generate the Edge installer, creating the hijacking opportunity. This specific vulnerability can be thought of as a software supply chain vulnerability – one flawed version of InstallShield leads to numerous vulnerabilities in applications that use InstallShield.

### Privilege Escalation

If an application or system service requires administrator privileges and contains a DLL hijacking vulnerability, and an attacker with Standard User privileges can write to a directory that is in the search order, they could escalate privileges to an administrator once the application is executed by a legitimate administrator. Dragos assesses with moderate confidence that privilege escalation is the most common reason for performing DLL hijacking attacks, allowing adversaries to execute otherwise restricted functionality.

### Gaining Persistence

A DLL hijacking vulnerability in an application added as a service to the Windows host could allow a high-privileged attacker to gain persistence on the host system over reboots. For example, a service may run with SYSTEM privileges and execute binaries stored in non-standard locations such as C:\<app_or_vendor_name>\ rather than C:\Windows\Program Files\ or C:\Windows\Program Files (x86)\. Often, these non-standard locations have incorrect permission assignments – they allow low-privileged users write access.

### Detection Evasion

When a process loads a DLL, it is executed in the context of that process. Using DLL hijacking, attackers can trick signed and trusted processes into loading their malicious DLL, making it appear more legitimate. Security solutions are less likely to identify the activity.

### DLL Proxying

If any DLL hijacking subvariant is discovered, an attacker could create a malicious DLL containing export functions with the same name as the legitimate DLL's export functions. When the application calls a library function, the malicious DLL can proxy the call to the legitimate DLL and execute attacker-controlled code. This has an added stealth advantage—it reduces the likelihood of application crashes or misbehavior, which could alert the user. DLL proxying is often described as being a vulnerability sub-variant of DLL hijacking. However, this term describes how an attacker's malicious DLL functions for added stealth behavior rather than an inherent difference in the vulnerability itself.

5 **LICSA-22-326-01** – cisa.gov

# DLL Abuse Targeting Industrial Organizations

Adversaries commonly abuse DLLs due to their versatility during cyber operations. While useful against any target, this technique has often been leveraged against industrial organizations. A similar technique was used in the A

## 2010

**Stuxnet** leveraged CVE-2012-3015 to trick Siemen's SIMATIC manager software into loading and executing a malicious DLL masquerading as **S7hkimdb.dll.** This DLL then decrypts, loads, and executes the main Stuxnet DLL payload with administrator privileges.[6] Stuxnet's ultimate goal was to subtly disrupt centrifuges used to enrich uranium at a facility in Natanz, Iran.[7] This exploit was present in multiple versions of Stuxnet as the malware was updated over time and helped the attackers install Stuxnet covertly.[8]

## 2022

**FEB**

**MuddyWater** has also been seen using DLL side-loading as one component of the POWGOOP malware which targeted government, oil and gas, telecommunications, and more.[9] POWGOOP is a remote access trojan, allowing the threat actor to stealthily execute commands on the host machine.

**AUG**

**Cotx/CotSam/DNSep** malware targeted industrial organizations in Eastern Europe and Afghanistan and used DLL hijacking in Windows binaries and security software to decrypt backdoors. These backdoors could execute arbitrary commands and collect host-based information.[10] This technique helped the attackers evade detection from security software solutions.

## 2023

**JUL**

As reported by Kaspersky, the **Meatball** and **FourteenH**i malware families targeted Eastern European industrial organizations as well as the Russian government and used DLL hijacking to install their payload by injecting a remote access trojan into a system process.[11] This implant helped facilitate data exfiltration, likely for the purpose of espionage.

**SEP**

**MuddyWater** used DLL hijacking to escalate privileges in a campaign against Israeli airlines and airports between September and October 2023.[12]

A similar technique was used in the **Dragonfly** espionage campaign to execute the Havex malware. The attackers compromised legitimate industrial vendors' websites and trojanized software that was available for download on the site.[13] The trojanized software, once executed, drops the malware to disk in a file named **mbcheck.dll** and invokes rundll32 with the malicious DLL:

6 **Stuxnet Dossier** – docs.broadcom.com
7 **Stuxnet Explained** – kaspersky.com
8 **Stuxnet 0.5: The Missing Link** – nsarchive2.gwu.edu
9 **MAR-10369127, MuddyWater** – cisa.gov
10 **Targeted attack on industrial enterprises and public institutions** – ics-cert.kaspersky.com
11 **Common TTPs of attacks against industrial organizations** – ics-cert.kaspersky.com
12 **Iranian Nation-State APT Groups 'Black Box' Leak** – clearskysec.com
13 **Havex Hunts For ICS/SCADA Systems** – archive.f-secure.com

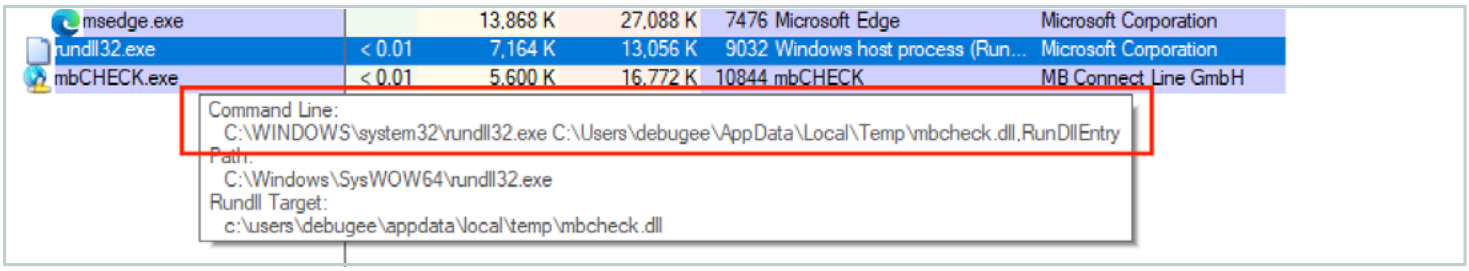



Figure 4: Havex malware invoked via rundll32.exe

Once executed, the **Havex** payload scans the network for OPC-speaking devices to map out and enumerate industrial equipment.[14] The initial infection vector is not an example of DLL hijacking. However, similar infection vectors exist, such as drive-by-download DLL hijacking, as discussed above.

## Dragos Vulnerability Database Statistics

Dragos maintains a vulnerability database of ICS and OT-related vulnerability data. Dragos performs vulnerability research, discloses the findings to the vendor, and publishes the data in our database for our **Dragos WorldView** customers. Additionally, Dragos collects public and private vulnerability data so that each security advisory published by an OT-related vendor or independent researcher can be analyzed and assessed for Dragos WorldView and Platform customers.

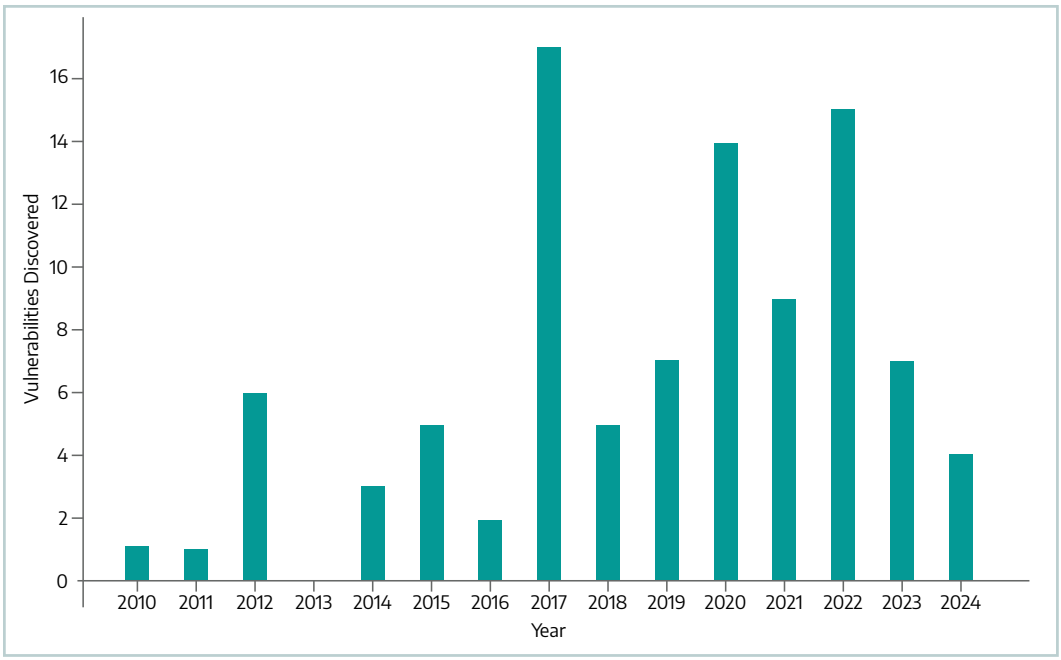



Figure 3: Malicious DLL planted in location searched prior to where legitimate DLL resides.

Dragos is aware of at least 98 vulnerabilities that impact ICS/OT-related software. The list includes almost all major industrial vendors and many commonly used software. Please see Appendix B for a complete list of OT-related DLL hijacking CVEs on page 15.

During the analysis of each vulnerability, Dragos classifies the affected software into Purdue Levels as defined in the Purdue Model.[15] Understanding where the vulnerability resides in OT networks and physical processes is critical to proper defense. The following statistics were generated from Dragos's vulnerability database and only include DLL hijacking vulnerabilities affecting ICS/OT/IIoT-related software.

14 **The Evolution of Cyber Attacks on Electric Operations** – dragos.com
15 **What Is the Purdue Model for ICS Security?** – zscaler.com

As demonstrated in Figure 6, most software vulnerable to DLL hijacking is in Levels 2 and 3 of the Purdue Model. These are where most Engineering Workstation Systems (EWS) or HMI/SCADA software are situated and where the configuration and programming of industrial devices often take place. Unfortunately, systems at Levels 2 and 3 can be quite sensitive, as these EWS often communicate directly with critical devices such as PLCs or HMIs and, therefore, have direct network or serial access to them.

Dragos analysts grouped all OT-related software vulnerable to DLL hijacking into software types and use cases commonly found in OT environments. This contributes to our understanding of what types of systems are most commonly vulnerable to this threat and where it may reside in the network.

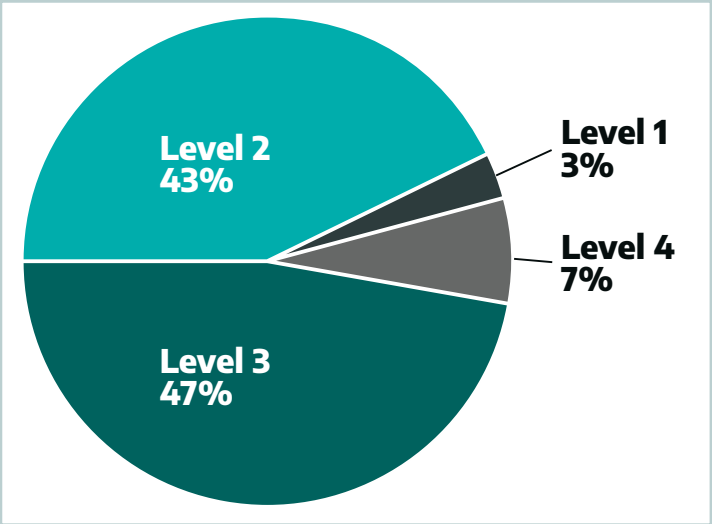



Figure 6: Where Does Vulnerable Software to DLL Hijacking Reside in the Purdue Model?

**Analyst Note:**

These are systems researchers chose to target, likely due to their importance or the availability of free software trials. This does not mean that these types of software assets are inherently more vulnerable.

## Finding 0-Day DLL Hijacking Vulnerabilities

Dragos encourages asset owners and security practitioners to attempt the discovery of 0-day DLL hijacking vulnerabilities when Windows hosts are connected to sensitive industrial systems. These vulnerabilities are incredibly low-hanging fruit for adversaries – they take a few minutes to discover and verify. Compared to more technically challenging vulnerabilities, DLL hijacking requires minimal technical know-how.

Microsoft provides a powerful tool, Process Monitor, as a Windows process, file system, and registry monitoring tool.[16] Process Monitor can be quickly and easily used to find DLL hijacking vulnerabilities, as shown in Microsoft guidance.[17]

Dragos discovered and disclosed a vulnerability in the Siemens Software Center (CVE-2021-41544[18]), which will be used to demonstrate how to identify DLL hijacking vulnerabilities. This software is used to "*download, install, and activate purchased software.*"[19]

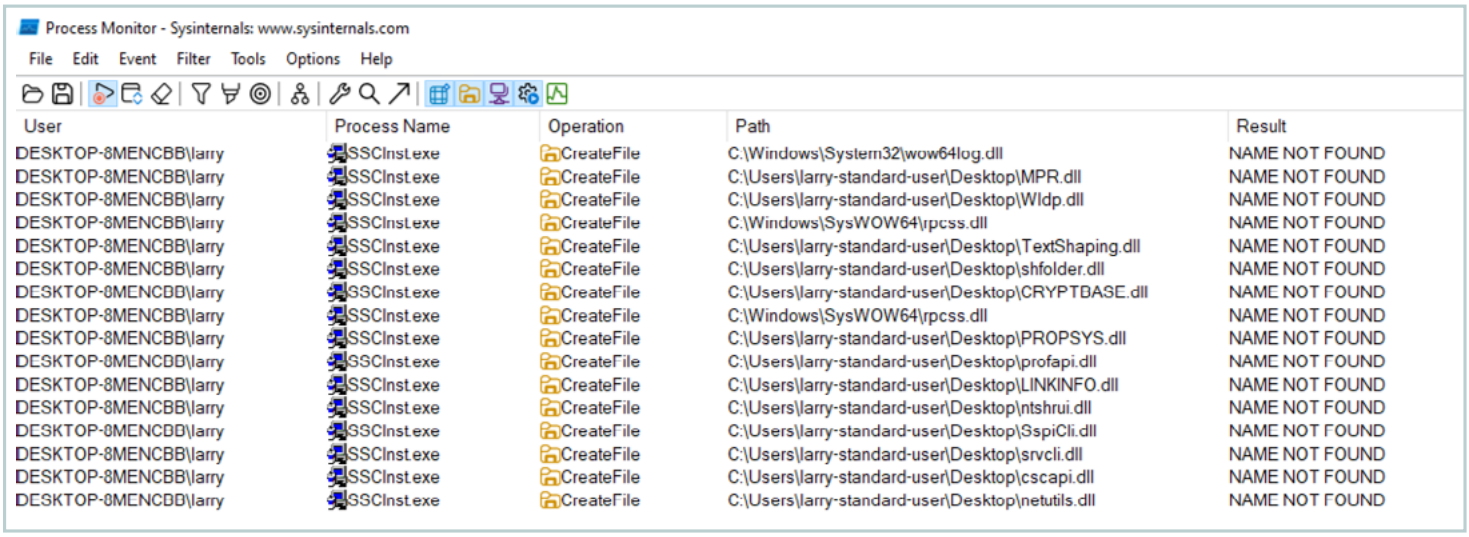
16 **Process Monitor** – learn.microsoft.com
17 **Secure loading of libraries to prevent DLL preloading attacks** – support.microsoft.com

Start by running Process Monitor with a few filter fields set to "Include":

| Column | Relation | Value | Action |
|---|---|---|---|
| Process Name | contains | SSCInst | Include |
| Result | is | NAME NOT FOUND | Include |
| Path | ends with | .dll | Include |

Figure 8: Process Monitor filters

These three filters, shown in Figure 8, are required to reduce the noise of irrelevant events. Once enabled, run the installer application. If the **"NAME NOT FOUND"** result appears for any **CreateFile** event, you may have found a DLL hijacking vulnerability (shown in Figure 9):




| User | Process Name | Operation | Path | Result |
|---|---|---|---|---|
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Windows\System32\wow64log.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\MPR.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\Wldp.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Windows\SysWOW64\rpcss.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\TextShaping.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\shfolder.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\CRYPTBASE.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Windows\SysWOW64\rpcss.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\PROPSYS.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\profapi.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\LINKINFO.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\ntshrui.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\SspiCli.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\srvcli.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\cscapi.dll | NAME NOT FOUND |
| DESKTOP-8MENCBB\larry | SSCInst.exe | CreateFile | C:\Users\larry-standard-user\Desktop\netutils.dll | NAME NOT FOUND |

Figure 9: DLL hijacking opportunities found!

In Windows, **CreateFile** is the API call used to create new files. Counterintuitively, it's also used to open and read existing files. When a process starts execution, it's normal behavior to see **CreateFile** events with DLLs in the path, as the process needs to open each DLL to load them. A **"NAME NOT FOUND"** result indicates that the process could not find the specified DLL. Therefore, a hijacking opportunity may exist.

In the Figure 9 example, there is an administrator user, *"larry,"* and a standard user, *"larry-standard-user."* Notice how the administrator is executing each **CreateFile** event and that several paths to the DLL are under *"larry-standard-user"* control. That is, *"larry-standard-user"* can write to the directory where the DLL file is stored since it's located on their Desktop. To exploit, the adversary must drop a malicious DLL file with the same name as one of the DLL files listed above.

Depending on the specific variant of DLL hijacking, different requirements must be satisfied. Since this is a DLL hijacking vulnerability in an installer application, the exploitation vector would likely be a drive-by-download scenario. In that case, execution by any privilege level is desired as the attacker would gain initial access to an environment. Luckily for the attackers, this installer application required administrator privileges to run, meaning the attackers would gain administrative code execution and take full control of the Windows workstation.

18 **ICSA-23-222-04: Siemens Software Center** – cisa.gov
19 **Siemens Software Center** – sw.siemens.com

## OT Cyber Defense Against DLL Hijacking

The root cause of DLL hijacking is insecure programming practices by the application's creator. However, if an organization is willing to audit host-by-host, there are a few different mechanisms to make successful exploitation of the vulnerability harder.

## CWDIllegalInDllSearch Registry Key

The registry key **CWDIllegalInDllSearch** can be set to remove the Current Working Directory (CWD) from the default DLL search order. There are three values that **CWDIllegalInDllSearch** can be set to:

| | |
|---|---|
| 0xFFFFFFFF | removes CWD from the search order. |
| -1 | blocks loading DLL files if CWD is set to a WebDAV folder. |
| -2 | blocks loading DLL files if CWD is set to a remote folder. |

The registry keys can be found in the following locations:

- **System-wide:** HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\CWDIllegalInDllSearch
- **Application specific:** HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\<app_name>.exe

To enable this key, open Registry Editor (regedit.exe) with administrator privileges, browse to the registry key, add a new DWORD value, name it CWDIllegalInDllSearch, and give it one of the three values listed above.

Before setting the registry key, the DLL is searched for in the CWD (highlighted in blue). Note that our CWD is on **C:\Users\larry\Desktop\dll-hijacking-example\dll-load-insecure\dllDoesNotExist.dll**:

| Process Name | Operation | Path | Result |
|---|---|---|---|
| dll-load-insecure.exe | CreateFile | C:\Windows\System32\wow64log.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Users\larry\Desktop\dll-hijacking-example\dll-load-insecure\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Windows\SysWOW64\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Windows\System\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Windows\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Users\larry\Desktop\dll-hijacking-example\dll-load-insecure\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\VC\Tools\MSVC\14.40.33807\bin\Hostx86\x86\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\Common7\IDE\VC\vcpackages\dllDoesNotExist.dll | NAME NOT FOUND |

Figure 10: Highlighted CreateFile event shows attempted load in CWD

After setting the registry key with value **0xFFFFFFFF,** the DLL is not searched for in the CWD:

| Process Name | Operation | Path | Result |
|---|---|---|---|
| dll-load-insecure.exe | CreateFile | C:\Windows\System32\wow64log.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Users\larry\Desktop\dll-hijacking-example\dll-load-insecure\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Windows\SysWOW64\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Windows\System\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Windows\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\VC\Tools\MSVC\14.40.33807\bin\Hostx86\x86\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\Common7\IDE\VC\vcpackages\dllDoesNotExist.dll | NAME NOT FOUND |
| dll-load-insecure.exe | CreateFile | C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\Common7\IDE\CommonExtensions\Microsoft\TestWindow\dllDoesNotExist.dll | NAME NOT FOUND |

Figure 11: CWD is no longer searched in the search algorithm

Analyst Note:

Microsoft recommends thorough testing that important applications are working correctly before production deployment and states that setting CWDIllegalInDLLSearch to 0xFFFFFFFF may have *"...unintended consequences for applications that require this behavior."*[20] Dragos recommends configuring specific applications individually to override the system-wide setting.

## Principle of Least Privileges

Always follow the Principle of Least Privileges for users and directories.[21]

- Only run applications as a low-privileged user (Standard User) unless the application specifically requires administrator access.
- Audit application directories to ensure Everyone and Standard Users user groups do not have write access. This will reduce the chances of successful privilege escalation. If implemented, ensure the application is working normally, as restricting access could break functionality if the application is programmed poorly.

## KnownDLLs Registry Key

The registry key **KnownDLLs** stores a list of DLL file names. During the loading of a library, if the file name of the library is stored in **KnownDLLs,** the Windows loader will forego the search path and assume the file is in **C:\Windows\System32** or **C:\Windows\SysWOW64\,** depending on whether the application is 32-bit or 64-bit. If the file is not found, the load library call will fail. This registry key prevents hijacking/preloading attempts for common shared libraries that are expected to be stored in the **System32/SysWOW64 directory.**

The registry key can be found in the following location:

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

Analyst Note:

It is not recommended users add or modify the KnownDLLs registry key.

20 **An update on the DLL-preloading remote attack vector** – msrc.microsoft.com
21 **Principle of least privileges** – en.wikipedia.org
22 **Secure loading of libraries to prevent DLL preloading attacks** – support.microsoft.com
23 **SetDllDirectoryA function** – learn.microsoft.com

## Vendors Need More Secure Programming Practices

The ultimate root of DLL hijacking is the mismatch in expectations between Microsoft and programmers. Microsoft expects vendors to fully qualify their specified search path, while programmers expect the operating system to know exactly where the file is by name. Threat actors exploit this gap to escalate privileges, hide their activities, or even gain initial access through trojanized applications.

Vendors can reduce opportunities for bad actors by following safe programming practices. Developers must review Microsoft's **Dynamic-Link Library Security** webpage to understand best practices. More secure Windows programming practices when loading DLLs include:[22]

- Call SetDllDirectory("") function followed by a call SetDllDirectory("<path_to_correct_dll>"). This will remove the Current Working Directory (CWD) from the search order. See Microsoft API documentation for best SetDllDirectory practices.[23]
- Do not use the SearchPath API to find and then load DLL files. Only use the LoadLibrary function to find and load DLL files. SearchPath function will first search CWD, creating a hijacking opportunity.
- Use fully qualified paths when the library location is static for the following functions:
  - LoadLibrary(...)
  - CreateProcess(...)
  - ShellExecute(...)
- Ensure that the application verifies the digital signature of external DLLs being loaded.

# Appendix A: Sub-Variants of DLL Hijacking

## DLL Search Order Hijacking / DLL Preloading

A locally authenticated adversary plants a DLL file in a directory that is **searched prior** to where the legitimate DLL file is stored. Most DLL hijacking terms can be used interchangeably with DLL search order hijacking.

## DLL Replacement

The directory where the application is stored can be written to by Standard Users and contains a DLL that is loaded by the application. Therefore, a low-privileged attacker can **replace** the DLL file with a malicious DLL of the same name. The key difference is the attackers are **replacing** a legitimate DLL with a malicious one.

## Phantom DLL Hijacking

Applications may attempt to load a DLL file that does not exist on the host system and is not bundled with the application. This may be a leftover library that was required in a previous version but forgot to be removed from the application's imports. If an attacker places a DLL file anywhere in the search order, it will be loaded and executed.

## DLL Side-Loading

Online sources have multiple definitions of DLL side-loading. MITRE and Palo Alto define side-loading as attackers placing a program known to be vulnerable in a directory they have write access to alongside a malicious DLL.[24, 25] This definition does not provide a privilege escalation advantage; the sole benefit would be detection evasion. However, Mandiant defines side-loading as an attacker planting a malicious DLL in the Window side-by-side (WinSxS) directory and tricking a vulnerable WinSxS assembly into loading the malicious DLL.[26]

## DLL Redirection Hijacking

DLL redirection is a Windows feature that allows developers to modify the search order.[27] It forces the local directory where the application exists to always be searched first, even if a DLL is loaded with a full path. This feature must be turned on via the registry settings to work. Once enabled, an adversary can copy their malicious DLL to the application directory, create a dot local or manifest file in the application's directory, and force the loading and execution of the DLL upon running the application.

While there are many variants of DLL hijacking, all refer to the same outcome—tricking the loader into executing an attacker-created DLL file. Many variants can be used interchangeably, such as search order hijacking, preloading, replacement, and phantom hijacking. The side-loading, replacement, and redirection techniques are a bit more unique, but still rely on the operating system finding the malicious DLL before the legitimate one.

24 **DLL Side-Loading** – attack.mitre.org
25 **Intruders in the Library: Exploring DLL Hijacking** – unit42.paloaltonetworks.com
26 **DLL Side-loading: A Thorn in the Side of the Anti-Virus Industry** – mandiant.com
27 **Dynamic-link library redirection** – learn.microsoft.com

# Appendix B: DLL Hijacking Vulnerabilities in OT Devices

| CVE ID | Affected Product |
|---|---|
| CVE-2024-1182 | Iconics AlarmWorX Multimedia |
| CVE-2024-5650 | Yokigawa CENTUM |
| CVE-2024-2637 | B&R Scene Viewer\|mapp Vision\|mapp Safety\|mapp View\|mapp Cockpit |
| CVE-2022-24767 | Bosch DIVAP IP |
| CVE-2024-1595 | CNCSoft-B DOPSoft |
| CVE-2023-6132 | AVEVA Edge |
| CVE-2023-0898 | GE MiCOM S1 Agile |
| CVE-2021-41544 | Siemens Software Center |
| CVE-2022-25634 | Siemens Software Center |
| CVE-2022-43722 | Siemens SICAM PAS\|Siemens SICAM PQS |
| CVE-2022-2333 | Honeywell SoftMaster |
| CVE-2023-29444 | Kepware KepServerEX\|ThingWorx Kepware Server\|ThingWorx Industrial Connectivity |
| CVE-2023-29445 | Kepware KepServerEX |
| CVE-2023-30897 | Siemens SIMATIC WinCC |
| CVE-2022-28686 | AVEVA Edge |
| CVE-2022-28687 | AVEVA Edge |
| CVE-2022-28688 | AVEVA Edge |
| CVE-2022-2334 | Softing Secure Integration Server |
| CVE-2022-2006 | Automation Direct DirectSOFT 6 |
| CVE-2022-23449 | Siemens SIMATIC Energy Manager |
| CVE-2022-23401 | Yokogawa Centum\|Yokogawa Exaopc |
| CVE-2022-23104 | WIN-911 |
| CVE-2022-23922 | WIN-911 |
| CVE-2021-44463 | Emerson DeltaV |
| CVE-2021-34803 | VISAM VBASE Editor |
| CVE-2021-38416 | Delta Electronics DIALink |
| CVE-2021-38469 | AUVESY Versiondog |
| CVE-2021-38410 | AVEVA Platform Common Services |
| CVE-2021-22775 | Schneider Electric Pro-face GP-Pro EX |
| CVE-2020-25182 | Rockwell Automation's ISaGRAF5 Runtime |
| CVE-2021-31219 | VIPA WinPLC7 |
| CVE-2020-25244 | Siemens LOGO! Soft Comfort |
| N/A | Yokogawa CENTUM |

| CVE ID | Affected Product |
|---|---|
| CVE-2021-22665 | Rockwell Automation DriveTools\|Rockwell Automation Drives AOP |
| CVE-2020-25245 | Siemens DIGSI 4 |
| CVE-2020-10616 | Opto 22 SoftPAC |
| CVE-2020-10610 | OSIsoft PI System |
| CVE-2020-10626 | Schneider Electric EcoStruxure IT Gateway |
| N/A | Rockwell Automation Connected Components Workbench |
| CVE-2019-6825 | Schneider Electric ProClima |
| CVE-2019-6564 | GE Communicator |
| CVE-2019-6546 | GE Communicator |
| CVE-2019-6534 | AVEVA Indusoft Web Studio\|AVEVA Intouch Edge HMI |
| CVE-2018-7799 | Schneider Electric Software Update (SESU)\|AVEVA Vijeo Citect\|AVEVA CitectSCADA |
| CVE-2018-14812 | Fuji Electric Energy Savings Estimator |
| CVE-2018-13806 | Siemens TD Keypad Designer |
| CVE-2018-14797 | Emerson DeltaV DCS Workstation |
| CVE-2017-14020 | Automation Direct CLICK Software |
| CVE-2017-14029 | Trihedral Engineering VTScada |
| CVE-2017-14010 | SpiderControl MicroBrowser |
| CVE-2017-14017 | Progea Movicon SCADA/HMI |
| CVE-2017-5147 | AzeoTech DAQFactory |
| CVE-2017-12717 | Advantech WebAccess |
| CVE-2017-9661 | SIMPlight SCADA |
| CVE-2017-9646 | Solar Controls Heating Control Downloader |
| CVE-2017-9648 | Solar Controls WATTConfig M |
| CVE-2017-5170 | Moxa SoftNVR-IA Live Viewer |
| CVE-2017-9961 | Schneider Electric Pro-face GP-Pro EX |
| CVE-2017-7966 | Schneider Electric SoMachine HVAC |
| CVE-2017-6033 | Schneider Electric Interactive Graphical SCADA System Software |
| CVE-2017-5176 | Rockwell Automation Connected Components Workbench |
| CVE-2017-5175 | Advantech WebAccess |
| CVE-2017-5161 | Sielco Sistemi Winlog SCADA |
| CVE-2016-4526 | ABB DataManagerPro |
| CVE-2016-2281 | ABB Panel Builder 800 |
| CVE-2015-7917 | Open Automation OPC Systems.NET |
| CVE-2015-3940 | Schneider Electric Wonderware Systems Platform |
| CVE-2015-1014 | Schneider Electric OPC Factory Server |
| CVE-2015-0990 | Ecava IntegraXor |

| CVE ID | Affected Product |
|---|---|
| CVE-2014-9209 | Rockwell Automation FactoryTalk |
| CVE-2015-0978 | Elipse E3 |
| CVE-2014-9207 | Cimon CmnView |
| CVE-2014-5430 | ABB RobotStudio\|ABB Test Signal Viewer |
| CVE-2012-3004 | RealFlex RealWin |
| CVE-2012-3015 | Siemens SIMATIC STEP 7 |
| CVE-2012-3005 | Invensys Wonderware InTouch 10 |
| CVE-2012-1824 | Measuresoft ScadaPro |
| CVE-2012-1819 | WellinTech KingView |
| CVE-2012-0223 | 7-Technologies TERMIS |
| CVE-2011-4053 | 7-Technologies IGSS |
| CVE-2010-4599 | Ecava IntegraXor |
| CVE-2017-6051 | BLF-Tech VisualView |
| CVE-2020-7585 | Siemens Step 7\|SIMATIC PCS\|SIMATIC PDM\|SINAMICS Starter |
| CVE-2019-10971 | Omron Network Configurator |
| CVE-2023-3662 | CODESYS Development System |
| CVE-2020-25238 | Siemens PCS neo\|Siemens TIA Portal |
| CVE-2022-35868 | Siemens TIA Project-Server |
| CVE-2022-1098 | Delta Electronics DIAEnergie |
| CVE-2018-7239 | Schneider Electric's SoMove |
| CVE-2019-6858 | Schneider Electric's MSX Configurator |
| CVE-2023-6061 | ICONICS SCADA Suite |
| CVE-2020-7490 | Schneider Electric's Vijeo Designer Basic |
| CVE-2020-6771 | Bosch IP Helper |
| CVE-2020-6786 | Bosch Video Recording Manager |
| CVE-2020-6787 | Bosch Video Client Installer |
| CVE-2020-6790 | Bosch Video Streaming Gateway |
| CVE-2020-6788 | Bosch Configuration Manager |
| CVE-2019-6826 | Schneider Electric SoMachine HVAC |

# Appendix C: CWE ID, CAPEC IDs, and MITRE ATT&CK Techniques

Common Weakness Enumeration (CWE) is a "list of software and hardware weaknesses that can become vulnerabilities."[28] DLL hijacking is most often associated with CWE-427: Uncontrolled Search Path Element.

Common Attack Pattern Enumeration and Classification is a "community resource for identifying and understanding attacks."[29] DLL hijacking is most often associated with CAPEC-471: Search Order Hijacking and CAPEC-641: DLL Side-Loading.

MITRE has documented DLL hijacking as an ATT&CK Technique, Hijack Execution Flow [T1574], with various sub-techniques:

- DLL Search Order Hijacking [T1574.001][30]
- DLL Side-Loading [T1574.002][31]
- Executable Installer File Permissions Weakness [T1574.005][32]
- Path Interception by PATH Environment Variable [T1574.007][33]
- Path Interception by Search Order Hijacking [T1574.008][34]

28 **About CWE** – cwe.mitre.org
29 **About CAPEC** – capec.mitre.org
30 **Hijack Execution Flow: DLL Search Order Hijacking** – attack.mitre.org
31 **Hijack Execution Flow: DLL Side-Loading** – attack.mitre.org
32 **Hijack Execution Flow: Executable Installer File Permissions Weakness** – attack.mitre.org
33 **Hijack Execution Flow: Path Interception by PATH Environment Variable** – attack.mitre.org
34 **Hijack Execution Flow: Path Interception by Search Order Hijacking** – attack.mitre.org


## About Dragos, Inc.

Dragos, Inc. has a global mission to safeguard civilization from those trying to disrupt the industrial infrastructure we depend on every day. Dragos is privately held and headquartered in the Washington, DC area with regional presence around the world, including Canada, Australia, New Zealand, Europe, and the Middle East.

Learn more about our technology, services, and threat intelligence offerings:

Request a Demo | Contact Us