









mscorsvc.dll

Part of the  [HijackLibs](#) project.

Types	<div> DLL Sideloading (1 EXE)</div> <p>By copying (and optionally renaming) a vulnerable application to a user-writeable folder, alongside a malicious <code>mscorsvc.dll</code>, arbitrary code can be executed through the legitimate application. See also MITRE ATT&CK® technique T1574.002: Hijack Execution Flow: DLL Side-Loading.</p> <div> Phantom DLL Hijacking (1 EXE)</div> <p>By copying a malicious <code>mscorsvc.dll</code> to a specific location, a vulnerable application will execute the malicious DLL's code upon normal execution.</p> <p>See also MITRE ATT&CK® technique T1574.001: Hijack Execution Flow: DLL Search Order Hijacking.</p>
Vendor	Microsoft
Resources	 <code>decoded.avast.io</code>  <code>www.securityjoes.com</code>
Acknowledgements	Thanks to Michał Kucharski (@Kucharskov).
Last updated	about 5 months ago

Expected Locations

The file `mscorsvc.dll` is normally found in the following paths:

-  `%WINDIR%\Microsoft.NET\Framework\v%VERSION%`
-  `%WINDIR%\Microsoft.NET\Framework64\v%VERSION%`

Vulnerable Executables

The following executables attempt to load `mscorsvc.dll`:

DLL Sideloading

 [%WINDIR%\Microsoft.NET\Framework\v%VERSION%\mscorsvw.exe](#)

Phantom DLL Hijacking

 [%WINDIR%\WinSxS\amd64_netfx4-ngentask_exe_%VERSION%\ngentask.exe](#)

Detection

Below a sample Sigma rule that will find processes that loaded `mscorsvc.dll` located in a folder that is not one of the expected locations (see above).

```
title: Possible DLL Hijacking of mscorsvc.dll
id: 4602001b-4150-48a3-8413-5b9ff8132122
status: experimental
description: Detects possible DLL hijacking of mscorsvc.dll by looking for suspicious image loads, loading this DLL from unexpected locations.
references:
  - https://hijacklibs.net/entries/microsoft/built-in/mscorsvc.html
author: "Wietze Beukema"
date: 2023-04-04
tags:
  - attack.defense_evasion
  - attack.T1574.001
  - attack.T1574.002
logsource:
  product: windows
  category: image_load
detection:
  selection:
    ImageLoaded: '*\mscorsvc.dll'
  filter:
    ImageLoaded:
```

 [Download YAML](#)

Note that this rule is also included in the [Sigma feed](#) that comprises all DLL Hijacking entries part of this project.

Why should I care about this?

DLL Hijacking enables the execution of malicious code through a signed and/or trusted executable. Defensive measures such as AV and EDR solutions may not pick up on this activity out of the box, and allow-list applications such as AppLocker may not block the execution of the untrusted code. There are numerous examples of threat actors that have been observed to leverage DLL Hijacking to achieve their objectives. As such, this project wants to encourage you to monitor for unusual activity involving `mscorsvc.dll`.

How do I abuse this vulnerability?

As a red teamer, you will have to compile your own version of `mscorsvc.dll`. There are [various guides](#) on how this can be achieved.

How could the vendor have prevented this vulnerability?

Phantom DLL Hijacking vulnerabilities are typically introduced due to human error: for example, a typo may cause an executable to attempt to load a non-existing DLL; similarly, the removal of a DLL without removing the reference in the depending application will result in the same. Better (code coverage) testing and unused dependency detection may aid in catching potential Phantom DLL vulnerabilities from being introduced. Most DLL Hijacking vulnerabilities are introduced by the 'lazy' loading of DLL files, which relies on Windows' default `DLL search order`. Explicitly specifying where a required DLL is located is easy and often already helps a lot. This doesn't have to hurt portability if Windows API calls are used to obtain paths, e.g. `GetSystemDirectory` to get the path of the System32 folder. Even better is to check the signature of required DLLs prior to loading them; most platforms, frameworks and/or runtimes offer means to verify DLL signatures with minimal performance impact.

This DLL Hijack doesn't seem to work (anymore), why is it still included?

Luckily, vendors regularly patch vulnerable applications in order to prevent DLL Hijacking from taking place. Nevertheless, older versions will remain vulnerable; for that reason, the entry won't be deleted from this project. To help others, you may want to open a pull request updating the 'precondition' tag on this entry to make the community aware of the reduced scope.

[Homepage](#) | [API](#) | [Contributors](#)