A Single Sub-Technique for DLL Hijacking

A proposal for MITRE® ATT&CK®

Wietze Beukema (@wietze)

February 2022

Contents

1 Introduction **1**
2 The problem **1**
2.1 Definitions: History 1
2.2 Definitions: Analysis 3
2.3 Gaps 4
2.4 Community usage 5
2.5 Summary 7
3 Solution **8**
3.1 Considerations 8
3.2 Sample definition 10
4 Summary and conclusion **12**

1 Introduction

This article explores two sub-techniques part of version 10 of MITRE ATT&CK: DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002). Issues with the current state of these techniques are examined and discussed, after which a solution that addresses the identified problems is presented.

2 The problem

This section introduces the different sub-techniques, analyses their definition and usage, and tries to articulate the problem with this status quo.

2.1 Definitions: History

MITRE ATT&CK is a knowledge base of adversary tactics and techniques based on real-world observations; therefore, it is to be expected that names, definition and alignment may change over time. After all, it is a reflection of what attackers do and have done, which constantly evolves. Furthermore, MITRE ATT&CK is a means to standardise terminology in the cyber security community; and as such, it is both pro-actively giving names to techniques so the community can use them, as well as adopting terms already in use by the community. ATT&CK is thus comparable to a natural language; it has a clear structure, it allows users to communicate with a shared terminology and understanding, but it also evolves over time, heavily influenced by its users.

With this in mind, consider two (sub-)techniques that have been present in ATT&CK from the very start. From the initial release of ATT&CK in 2018 until early 2021, the definition of both DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002) have remained unchanged. In version 7, launched in July 2020, they were assigned their current sub-technique IDs under Hijack Execution Flow (T1574), with their definitions staying the same. However, adjustments to both definitions were introduced in April 2021 when version 9 of ATT&CK was launched. Whereas DLL Search Order Hijacking (T1574.001) mostly remained the same, the definition of DLL Side-Loading (T1574.002) changed substantially. The different definitions as found on the MITRE ATT&CK website can be found in table 1.

Table 1: Definitions of *DLL Search Order Hijacking* and *DLL Side-loading* according to the MITRE ATT&CK website.

	Version 1 – Version 8 (between January 2018 and March 2021)	**Version 9 – Version 10** (between April 2021 and now)
T1574.001: DLL Search Order Hijacking	Adversaries may execute their own malicious payloads by hijacking the search order used to load DLLs. Windows systems use a common method to look for required DLLs to load into a program. [1] Hijacking DLL loads may be for the purpose of establishing persistence as well as elevating privileges and/or evading restrictions on file execution. There are many ways an adversary can hijack DLL loads. Adversaries may plant trojan dynamic-link library files (DLLs) in a directory that will be searched before the location of a legitimate library that will be requested by a program, causing Windows to load their malicious library when it is called for by the victim program. Adversaries may also perform DLL preloading, also called binary planting attacks, [2] by placing a malicious DLL with the same name as an ambiguously specified DLL in a location that Windows searches before the legitimate DLL. Often this location is the current working directory of the program. Remote DLL preloading attacks occur when a program sets its current directory to a remote location such as a Web share before loading a DLL. [3] Adversaries may also directly modify the way a program loads DLLs by replacing an existing DLL or modifying a .manifest or .local redirection file, directory, or junction to cause the program to load a different DLL. [4] [5] [6] If a search order-vulnerable program is configured to run at a higher privilege level, then the adversary-controlled DLL that is loaded will also be executed at the higher level. In this case, the technique could be used for privilege escalation from user to administrator or SYSTEM or from administrator to SYSTEM, depending on the program. Programs that fall victim to path hijacking may appear to behave normally because malicious DLLs may be configured to also load the legitimate DLLs they were meant to replace.	Adversaries may execute their own malicious payloads by hijacking the search order used to load DLLs. Windows systems use a common method to look for required DLLs to load into a program. [1][2] Hijacking DLL loads may be for the purpose of establishing persistence as well as elevating privileges and/or evading restrictions on file execution. There are many ways an adversary can hijack DLL loads. Adversaries may plant trojan dynamic-link library files (DLLs) in a directory that will be searched before the location of a legitimate library that will be requested by a program, causing Windows to load their malicious library when it is called for by the victim program. Adversaries may also perform DLL preloading, also called binary planting attacks, [3] by placing a malicious DLL with the same name as an ambiguously specified DLL in a location that Windows searches before the legitimate DLL. Often this location is the current working directory of the program.[4] Remote DLL preloading attacks occur when a program sets its current directory to a remote location such as a Web share before loading a DLL. [5] Adversaries may also directly modify the search order via DLL redirection, which after being enabled (in the Registry and creation of a redirection file) may cause a program to load a different DLL.[6][7][8] If a search order-vulnerable program is configured to run at a higher privilege level, then the adversary-controlled DLL that is loaded will also be executed at the higher level. In this case, the technique could be used for privilege escalation from user to administrator or SYSTEM or from administrator to SYSTEM, depending on the program. Programs that fall victim to path hijacking may appear to behave normally because malicious DLLs may be configured to also load the legitimate DLLs they were meant to replace.
T1574.002: DLL Side-Loading	Adversaries may execute their own malicious payloads by hijacking the library manifest used to load DLLs. Adversaries may take advantage of vague references in the library manifest of a program by replacing a legitimate library with a malicious one, causing the operating system to load their malicious library when it is called for by the victim program. Programs may specify DLLs that are loaded at runtime. Programs that improperly or vaguely specify a required DLL may be open to a vulnerability in which an unintended DLL is loaded. Side-loading vulnerabilities specifically occur when Windows Side-by-Side (WinSxS) manifests [1] are not explicit enough about characteristics of the DLL to be loaded. Adversaries may take advantage of a legitimate program that is vulnerable by replacing the legitimate DLL with a malicious one. Adversaries likely use this technique as a means of masking actions they perform under a legitimate, trusted system or software process.	Adversaries may execute their own malicious payloads by side-loading DLLs. Similar to DLL Search Order Hijacking, side-loading involves hijacking which DLL a program loads. But rather than just planting the DLL within the search order of a program then waiting for the victim application to be invoked, adversaries may directly side-load their payloads by planting then invoking a legitimate application that executes their payload(s). Side-loading takes advantage of the DLL search order used by the loader by positioning both the victim application and malicious payload(s) alongside each other. Adversaries likely use side-loading as a means of masking actions they perform under a legitimate, trusted, and potentially elevated system or software process. Benign executables used to side-load payloads may not be flagged during delivery and/or execution. Adversary payloads may also be encrypted/packed or otherwise obfuscated until loaded into the memory of the trusted process.[1]

2.2 Definitions: Analysis

This section walks through the different definitions and their versions as presented in table 1, interpreting their meaning and usage.

2.2.1 DLL Search Order Hijacking (T1574.001)

At the start of the definition, it is explained Windows loads DLLs in a 'common order', and introduces the concept of hijacking, in which an attacker executes their own code by taking over this search order.

It then states that 'DLL loads' can be hijacked in 'many ways', including:

1. by placing a DLL *in a directory that will be searched before the location of a legitimate library*;
2. by 'DLL Preloading'/'DLL Planting' attacks, which is defined as placing a DLL *in a location that Windows searches before the legitimate DLL*. It specifically mentions that this 'often' includes the current directory, and that this can be combined with e.g. setting the current directory to a web share to enable remote DLL preloading; and,
3. by 'DLL Redirection', which requires registry changes, the creation of a special redirect file, and effectively overrides the path from which a DLL should be loaded.

In it's final paragraph, it is mentioned that 'search order-vulnerable programs' running under higher privileges may lead to privilege escalation, and that hijacked programs not necessarily will malfunction as the legitimate DLL may be loaded as well.

$\Longrightarrow$ With this definition, there seems to be a heavy emphasis on the search order, which given the name of the technique shouldn't come as a surprise. However, the somewhat ambiguous start of the second paragraph that talks about a more broad 'hijacking of DLL loads', asserts that T1574.001 can be achieved in a variety of ways. It then provides three examples of which the first two seem identical (as can be seen by comparing the italic texts in example 1 and 2 above) and the last one doesn't quite seem to be in line with the introductory paragraph, as the search order isn't hijacked, but rather 'overruled'. Furthermore, due to the usage of the rather broad statement "There are many ways an adversary can hijack DLL loads.", users may think anything involving DLL hijacking can be aligned to this sub-technique.

2.2.2 DLL Side-Loading (T1574.002)

The old version (pre April 2021) tells us that hijacking 'library manifests' can lead to the loading of arbitrary DLLs by other programs. It states that attackers may 'replace a legitimate library with a malicious one' to achieve this. It is then suggested that T1574.002 'specifically occurs' when a Windows feature called Windows Side-by-Side (WinSxS) relies on manifests that are 'not explicit enough' about the DLL that should be loaded. Attackers can abuse this by 'replacing the legitimate DLL with a malicious one'. Whether 'specifically occurs' means that the description that follows is the only instance of DLL Side-Loading or whether it is simply an example, remains unclear. Whether the 'Side' in 'DLL Side-Loading' comes from Windows Side-by-Side, is not explained either.

The new version (April 2021 onward) paints a rather different picture: it starts off by saying this technique is rather similar to DLL Search Order Hijacking (T1574.001), but instead of planting only a DLL and waiting for the vulnerable program to be executed, the attacker may 'directly side-load their payloads' by 'planting' both the program and the DLL. It is then explained that it is thanks to the DLL search order putting both together will cause the malicious DLL to be loaded. Although not explicitly stated, it may be possible that 'Side' in 'DLL Side-Loading' comes from the fact that executable and library should be put next to one another ('side by side').

$\Longrightarrow$ There seems to be a big discrepancy between the version 1 and version 2 of this sub-technique definition. The former talks about replacing DLLs and changing WinSxS manifests, whilst the latter says legitimate programs should be planted next to malicious DLLs. It is hard to come up with any examples that would align to both definitions. Other than the shared name, it seems like we are looking at two completely different techniques.

Taking the most recent version, the comparison that the definition makes between DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002) is telling. Although a clear distinction between the two sub-techniques is made, it is also made clear they are closely related due to their shared reliance on the DLL search order. T1574.001 is hijacking the execution flow by hijacking *the DLL search order* itself; T1574.002 on the other hand is hijacking the execution flow by relocating the original executable and then either placing the DLL anywhere in the search order, as long as it is loaded first. Due to relocation, the original DLL may not be found at all, and therefore the DLL search order is not hijacked as such.

2.3 Gaps

Having analysed the current definitions of both DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002), this section discusses other, related attacker techniques that involve taking over ('hijacking') DLLs in a way that does not seem to be covered by either definition:

2.3.1 Phantom DLL Hijacking

This technique abuses executables attempting to load DLLs that under normal conditions do not exist. Such 'Phantom DLLs' are often introduced when dependencies are removed but the reference to it is not, but there are also instances where typos or programming errors cause the unexpected DLL load.

Examples of the former kind include instances documented by Hexacorn [6], examples of the latter include a Kaspersky executable accidentally appending the drive letter to a DLL load [9].

Due to the fact that the DLL doesn't normally exist, technically the search order isn't hijacked. After all, to hijack (or: take over) something, it should normally load a DLL in the first place; T1574.001 therefore doesn't apply. Nor is putting the DLL next to the vulnerable executable a hard requirement (as demonstrated by the Kaspersky example [9]); T1574.002 doesn't apply either.

2.3.2 DLL Substitution

In a rather straight-forward approach, this technique simply replaces a pre-existing, legitimate DLL with a malicious one. This is often combined with 'DLL Proxying' [2], a technique in which the malicious DLL proxies all calls to the legitimate DLL, before adding the malicious payload(s) to them.

This technique was deployed by the threat actor responsible for Stuxnet, who renamed a legitimate DLL present in the `System32` folder, dropped a DLL with the original name next to it, proxing most calls but intercepting a specific one in an effort to manipulate the SCADA code executed on the PLC [7].

The DLL Search Order remains unchanged, and therefore T1574.001 does not apply. In some cases DLL Side-Loading (T1574.002) may technically apply (when executable and DLL were already placed alongside each other), but this will not always be the case: even when the full path of the DLL is specified (and thus the DLL Search Order won't be relied on), this attack may still work. Some may argue Create or Modify System Process (T1543) could apply, although this technique specifically focuses on processes rather than libraries.

2.3.3 WinSxS Hijacking

With this technique, a legitimate DLL is replaced with a malicious one in the relevant WinSxS (Windows Side-by-Side) folder of the targeted DLL.

The best documented case (and possibly the only one) is provided in a 2014 report by FireEye [3].

Although featuring prominently in the early version of T1574.002, the latest version doesn't seem to cover this technique any longer. Because for this technique to work the DLL Search Order wouldn't be involved, T1574.001 doesn't seem a good fit either.

2.3.4 Environment Variable-based DLL Hijacking

Some programs rely on environment variables (either set globally or on process level) to load DLLs. By changing the environment variable, it is possible to load a different DLL than intended.

A specific example of this is the `COR_PROFILER` environment variable, which can be used to load arbitrary DLLs into .NET executables. Presumably because of its sizable attack surface, a dedicated sub-technique T1574.012 was assigned. This is however not the only known instance of environment variable-based DLL hijacking; for example, Fortinet [4] describes how changing `%SystemRoot%` can trick various processes into loading arbitrary DLLs.

Because environment variables usually resolve to full paths, by altering the environment variable, the search order is typically not involved, and as such T1574.001 does not apply. Because the vulnerable program does not need to be relocated and since the DLL does not necessarily has to be placed next to the vulnerable program, T1574.002 does not apply either.

2.3.5 Other input-based DLL Hijacking

This technique, somewhat similar to the previous, can be used when a program relies on any other form of user input (such as command-line arguments, registry locations, configuration files, etc.) for determining where one or more DLLs are loaded from.

One example of this technique is `extexport.exe`, which takes a path as command-line argument from which three DLLs are loaded [8].

Again, it depends on the vulnerable program, but if the value taken normally resolves to a full path, altering it will not affect the search order, and as such T1574.001 does not apply. Because the vulnerable program does not need to be relocated and since the DLL does not necessarily has to be placed next to the vulnerable program, T1574.002 does not apply either.

Further techniques

Finally, it should be noted that there may be more techniques that involve hijacking DLLs in some shape or form that are not listed above; including techniques that may surface for some time to come.

2.4 Community usage

As set out in section section 2.1, one of the reasons MITRE ATT&CK 'works', is due to the fact that it is widely adopted in the community, as well as its reflection of the cyber security community's usage of certain terminology. Whilst not necessarily a goal in itself, it is therefore at least desirable that community usage is in line with ATT&CK's definitions.

Misalignment

Practice however shows that this is not (always) the case for the two sub-techniques that have been considered so far. Table 2 lists every example that is listed on the MITRE ATT&CK page for DLL Search Order Hijacking (T1574.001), as of version 10 [1]. The fact that these publications are listed as examples implies that MITRE ATT&CK considers the observed technique to be in line with the definition of T1574.001. For each entry, Table 2 additionally lists related terms that were used in the publication, and evaluates whether it meets the definition of T1574.001 as assumed based on section 2.2.

Table 2's analysis suggest that whilst for 5 entries (∼20%) no verdict could be given, more than half (13 entries, ∼54%) do not seem to be in line with T1574.001 as currently defined. Although 3 entries (∼12.5%) do seem to be actual examples of DLL Search Order Hijacking, only one of them seems to be using this term; the other two refer to it as 'DLL Hijacking'. Another 3 entries (∼12.5%) are likely to align to T1574.001 but have some caveats; for example, two of them are technically hijacking the search order, but due to the fact that the executable is dropped by the attacker, show a very strong similarity with DLL Side-Loading (T1574.002).

Term confusion

The second column in table 2 also demonstrates that a variety of different terms are used; both when in line with T1574.001 and when not. This is possibly a result of the confusion that DLL Side-Loading (T1574.002) may have caused prior to version 9. As discussed in section 2.2, the original definition seemed to describe to a rather specific technique, users may have chosen to align to DLL Search Order Hijacking (T1574.001) instead, even when with the current definition, it would have been in line with DLL Side-Loading (T1574.002). This is reflected in the fact that some reports describe DLL Side-loading, but still align to T1574.001 in the MITRE ATT&CK tables/sections.

When checking search engine results for the various observed terms in the analysis, figure 1 shows that although 'DLL Search Order Hijacking' and 'DLL Side-loading' are used in roughly equal measure, they

ID / Name	**Terms used**	**Meets def?**	**Comment**
Name: G0096 / APT41 **Report**: CrowdStrike	DLL Search Order Hijacking	✓	Actually hijacks search order by putting DLL in `c:/windows/` which is searched before `c:/windows/system32`.
Name: S0373 / Astaroth **Report**: Kaspersky	DLL Search Order Hijacking DLL Hijacking	✗	Doesn't hijack search order, relies on path passed on command line.
Name: G0135 / BackdoorDiplomacy **Report**: ESET	DLL Search–Order Hijacking	✗	'the operators uploaded, via a webshell, both `ScnCfg.exe`, a legitimate McAfee executable, and `vsodscpl.dll`, a malicious DLL'; meets definition of DLL Side-loading.
Name: S0415 / BOOSTWRITE **Report**: FireEye	DLL Hijacking DLL Search Order Hijacking	?	Unclear
Name: S0631 / Chaes **Report**: CyberReason	N/A	✗	`hh.exe` is 'brought to' the machine by the attacker; meets definition of DLL Side-loading.
Name: S0538 / Crutch **Report**: ESET	DLL Hijacking	✗	`finder.exe` is 'dropped on the compromised system by the operators'; meets definition DLL Side-loading.
Name: S0134 / Downdelph **Report**: ESET	DLL Load Order Hijacking	✗	Relies on changing the environment variable `%systemroot%`.
Name: S0363 / Empire **Report**: Empire	N/A	?	Not clear from readme.
Name: G0120 / Evilnum **Report**: ESET	DLL Search Order Hijacking	✓*	Arguably closer to DLL Side-loading since the executable is dropped in `c:/users/public`.
Name: S0182 / FinFisher **Report**: Kaspersky	DLL Search Order Hijacking	✗	`AdapterTroubleshooter.exe` is copied to a subfolder in `c:/programdata/`; meets definition of DLL Side-loading.
Name: S0009 / Hikit **Report**: FireEye	N/A	?	Dead link.
Name: S0070 / HTTPBrowser **Report**: ZScaler	DLL Hijacking Load Order	✗	`VPDN_LU.exe`, a legitimate Symantec executable, is created by the dropper; meets definition of DLL Side-loading.
Name: S0260 / InvisiMole **Report**: ESET	DLL Hijacking	✓	Actually hijacks search order by putting DLL in `c:/windows/` which is searched before `c:/windows/system32`.
Name: S0530 / Melcoz **Report**: Kaspersky	N/A	-	See *S0373 / Astaroth* .
Name: G0045 / menuPass **Report**: PwC	DLL Load Order Hijacking DLL Search Order Hijacking DLL Sideloading	✗	The technical annex only describes two instances of what is referred to as 'DLL sideloading'.
Name: S0280 / MirageFox **Report**: Intezer	DLL Hijacking	✗	Relies on the 'distribution' of a legitimate McAfee binary.
Name: S0194 / PowerSploit **Report**: PowerShellMafia	DLL Hijacking	✓	Has the ability to detect actual Search Order Hijack opportunities.
Name: S0458 / Ramsay **Report**: ESET	Phantom DLL Hijacking	✗	Relies on non-existing DLLs; debatable whether this can be seen as Search Order Hijacking (see section 2.3).
Name: S0153 / RedLeaves **Report**: FireEye	DLL Search Order Hijacking	?	Unclear from description.
Name: G0048 / RTM **Report**: Group IB	DLL Search Order Hijacking	✓*	Arguably closer to DLL Side-loading since the executable is dropped in `%appdata%`.
Name: G0027 / Threat Group-3390 **Report**: NCC Group	DLL Hijacking DLL Search Order Hijacking	✗	`INISafeWebSSO.exe` was included in dropper; meets definition of DLL Side-loading.
Name: G0131 / Tonto Team **Report**: ESET	DLL Search-Order Hijacking	✗	Relies on variable legitimate executables 'also dropped by the attackers'; meets definition of DLL Side-loading.
Name: S0612 / WastedLocker **Report**: NCC Group	DLL Hijacking	✗	Requires legitimate executables to be copied to a user-writable folder; meets definition of DLL Side-loading.
Name: S0109 / WEBC2 **Report**: FireEye	DLL Search Order Hijacking	✓*	Although not clear from the report itself, from other reports it seems likely that a pre-existing, legitimate executable will hijack the seach order.
Name: G0107 / Whitefly **Report**: Symantec	DLL Search Order Hijacking	?	DLL Search Order Hijacking is described, but no examples are provided.

Table 2: Review of 'DLL Search Order Hijacking' examples listed on MITRE ATT&CK's technique page of T1574.001 (version 10) [1].

are significantly outnumbered by the term 'DLL Hijacking'. This seems to be a broader term, encapsulating multiple DLL Hijack-related terms including both Search Order Hijacking and Side-loading, that is commonly used by the community to avoid the definition differences.

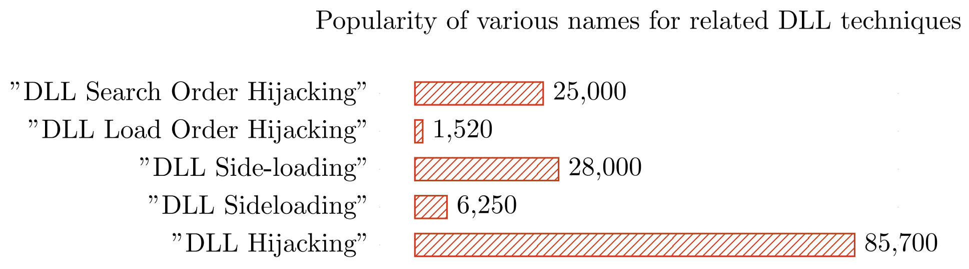

Figure 1: Popularity of different terms used for various DLL techniques according to Google Search on 30 January 2022. Every term was searched verbatim, i.e. wrapped in double quotes. The results show that 'DLL Hijacking' is not an uncommon term; in fact, it is used most often.

2.5 Summary

In summary, we have identified the following problems:

1. DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002) are often confused and/or used interchangeably. This may have been caused by:
   (a) The community seems to have assumed a different definition than ATT&CK, in particular for T1574.002 (see section 2.4);
   (b) Possibly in response to this, the definition of T1574.002 was changed quite radically (see section 2.2), adding to the confusion;
   (c) Despite the fact that T1574.001 and T1574.002 have considerable overlap (especially since the release of ATT&CK v9), they were separated into two distinct techniques, making misalignment more likely.
2. Various attacks of a similar nature are not covered or covered insufficiently by T1574.001 or T1574.002, or any other technique (see section 2.3).

3 Solution

Trying to address the two problems identified in the previous section, this section discusses various options that would improve the current situation, and presents what the author reckons is the preferred solution in more detail.

3.1 Considerations

To address the issues identified and summarised in section 2.5, the following solutions may help solve them:

3.1.1 Improving the existing sub-techniques

By updating the definitions of the existing techniques T1574.001 and T1574.002, it may be possible to take away the confusion (problem 1) and cover all techniques (problem 2). The main advantage of this solution is that no technique/technique IDs will require to be abolished, ensuring the highest degree of backwards compatibility.

To accomplish this, the examples on both pages should be updated to accurately reflect the definitions for each technique. Although this does not immediately solve problem 1, it should make the distinction between the two techniques clearer; and hopefully over time, the community will adopt and apply the techniques in an appropriate manner.

To address problem 2, it should either be encouraged to use the more generic Hijack Execution Flow (T1574) for such techniques, or the definitions of T1574.001 and T1574.002 should be updated to ensure more techniques are covered. The former option would be undesirable as related techniques should be put together as much as possible. The latter option will, as seen in the analysis of section 2.2, however prove rather difficult; most identified cases are likely to end up under DLL Search Order Hijacking (T1574.001), as is currently the case. Doing so is destined to introduce more confusion: T1574.001 will be a 'catch-all' technique whereas T1574.002 will remain more specific. This will conflict with solving problem 1.

In summary, this solution does reduce the problems somewhat, but fails to address them completely.

✓ **Pros** Better backwards compatibility, no need to demote/supersede existing techniques;

✗ **Cons** Does not fully solve the terminology confusion; additional techniques cannot be aligned appropriately.

3.1.2 Adding more sub-techniques under T1574

Given that the additional techniques seem to cause the most trouble with the previous solution, it could be argued more sub-techniques should be introduced under Hijack Execution Flow (T1574) so they can be put alongside T1574.001 and T1574.002. This would show they are clearly part of T1574 but also reaffirm their equality with techniques DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002).

A big advantage of this approach is that it allows to give more specific and narrow definition of the various types of DLL Hijacking; as shown in section 2.3, there is quite a range of possible techniques that fall into this category. Since not only the 'attacker approach' but also the detection and preventative measures differ for each, there is a strong case for separate pages for each with tailored descriptions. On the surface, this seems to solve problems 1 and 2 entirely. It should be noted though that by introducing new techniques some initial confusion will always remain, especially for adversarial behaviour that historically has been aligned to a different technique. However, by creating specific and narrowly-defined sub-techniques for each, there should really be *less room* for confusion – if defined properly, each observed behaviour should really only have one ATT&CK technique candidate.

There is however a major disadvantage: the number of DLL Hijack sub-techniques would, if adopting all 5 suggestions in section 2.3, be at least 7; since T1574 already has 11 sub-techniques (including the two pre-existing techniques), it would increase to 16. It would also mean nearly half the sub-techniques are of a single class of Hijack Execution Flow techniques, which is undesirable. Because they are mixed with other non-DLL-related techniques, it makes their relation also rather obscure, possibly introducing new confusion. Finally, the fact that new sub-techniques of a similar nature may emerge, means this disadvantage is likely to grow even bigger.

Altogether, it makes this solution feasible, yet not desirable.

✓ **Pros** Covers all DLL Hijack techniques; eliminates most confusion; allows for detailed descriptions and recommendations;

✗ **Cons** Suboptimal backwards compatibility; makes T1574 rather unclear; less future-proof solution.

3.1.3 Creating a new, single technique with sub-techniques

The confounding situation that might emerge by implementing the previous solution could be averted by creating a new ATT&CK technique that brings together all different kinds of DLL Hijacking techniques.

This new technique, which could be called 'DLL Hijacking', would have multiple sub-techniques, including the pre-existing DLL Search Order Hijacking and DLL-Sideloading, alongside techniques such as identified in section 2.3. This still takes advantage of the fact that every technique can be described in detail on its own page, while keeping this class of techniques clustered together under a single, dedicated technique. Although this would require the pre-existing DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002) to be 'renumbered', it ensures that the related techniques stay together, and that the number of sub-techniques doesn't grow absurdly big as it would in the previous solution. As such, it solves problem 1 and 2 even better.

The downside of taking this approach is that the link with Hijack Execution Flow (T1574) disappears; which is undesirable, given that DLL Hijacking is clearly a subclass of hijacking execution flows. Since sub-sub-techniques do not exist, it will not be possible to place the new 'DLL Hijacking' technique under T1574. Creating another technique in the already crowded Defence Evasion tactic is, whilst not the end of the world, not ideal. There is no straightforward solution of this problem; if going ahead with this solution, the pre-existing T1574 would require an overhaul of sorts - for example creating a clearer distinction between it and the new technique, or putting the remaining sub-techniques in clearer-defined techniques. Either way, it affects backwards compatibility more than with the other solutions, whilst also introducing new causes of confusion.

Therefore, this solution is promising, but has practical implications that make execution rather challenging.

✓ **Pros** Covers all DLL Hijack techniques; eliminates most confusion; allows for detailed descriptions and recommendations; keeps all related techniques together; avoids the need of sub-sub-techniques; more future-proof solution;

✗ **Cons** Suboptimal backwards compatibility; removes the link with T1574 despite being clearly related; possibly requires overhaul of T1574 as a whole.

3.1.4 Creating a single sub-technique

Taking into account all observations made in the previous solutions, a more sustainable approach would be to create a single sub-technique under T1574, capturing all DLL Hijacking techniques.

There is a strong case to make for the similarity in the discussed hijacking techniques (sections 2.2 and 2.3); even though historically at least two different techniques have existed, the interchangeable use of them demonstrates the fact that to many they are considered one and the same already. It was also shown in section 2.4 that 'DLL Hijacking' is already in use, and in fact seems to be the most popular term. In the end, what all discussed techniques have in common is that they trick an executable into executing code originating from a DLL that was not intended to be loaded; even though the way of achieving this may differ, the 'result' is ultimately similar in all cases.

This single sub-technique, which could be called 'DLL Hijacking', has several advantages. Having a single technique could eliminate discussions about what kind of DLL Hijacking applies entirely. As observed in table 2, there are cases where it could be argued that a behaviour can be aligned to either of two techniques. Having a single sub-technique allows them to be aligned to the same kind of technique, while leaving it to the author to elaborate on the specific type of DLL Hijacking, if desired. Additionally, there is the option to re-use one of the pre-existing techniques to ensure better backwards compatibility.

It could be argued that a downside of this approach is the fact that it loses detail; it lacks dedicated pages for each DLL Hijacking 'subtype', although that is not really the case in the current version of ATT&CK either. Furthermore, by discussing the different kinds of DLL Hijacking and providing good examples, it could still be captured comprehensively by the single sub-technique page. It should be noted that DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002) will effectively be merged; it may therefore still cause some backward compatibility issues.

Whilst not a silver bullet either, this solution seems the most sensible when taking into account all different aspects that come with the usage, adoption, and maintainability of ATT&CK and its (sub)techniques.

✓ **Pros** Covers all DLL Hijack techniques; eliminates most confusion; keeps all related techniques together; avoids the need of sub-sub-techniques; more future-proof solution; uses industry-recognised term;

✗ **Cons** Suboptimal backwards compatibility; fewer options for discussing different types in detail.

3.2 Sample definition

When considering the adoption of a single sub-technique for DLL Hijacking (section 3.1.4), the below is an example of what the definition might look like:

Hijack Execution Flow: DLL Hijacking

Definition

Adversaries may execute their own malicious payloads through hijacking the process flow of vulnerable programs, by tricking them into loading a malicious Dynamic Link Library (DLL). As a result, adversary-controlled code will be executed through a legitimate (and often trusted) program.

There are a number of ways in which an adversary can perform DLL Hijacking, including but not limited to:

- **DLL Search Order Hijacking**: Windows systems use a common method to look for required DLLs to load into a program [I]. Adversaries may plant DLL files in a directory that will be searched before the location of a legitimate library, causing a vulnerable program to load their malicious library when it is called for [II].

- **DLL Side-Loading**: Adversaries may plant a vulnerable program alongside an attacker-controlled DLL, causing the program to load the malicious library [III]. Adversaries have been seen to bring a legitimate/trusted yet vulnerable application themselves in some cases.

- **Phantom DLL Hijacking**: Programs vulnerable to this type of DLL Hijacking attempt to load libraries that do not exist under normal conditions. By creating a malicious DLL in the expected location, the vulnerable program will inadvertently load the malicious library [IV].

- **DLL Substitution**: An adversary might hijack a DLL altogether by replacing it with a malicious version [V]. In some cases, the attacker might proxy certain functions within the DLL to the legitimate version, to ensure the malicious DLL doesn't break the executable that loads it.

Other techniques include WinSxS hijacking [VI] and input-based DLL Hijacking [VII].

If a DLL Hijack-vulnerable program is configured to run at a higher privilege level, then the adversary-controlled DLL that is loaded will also be executed at the higher level. In this case, the technique could be used for privilege escalation from user to administrator or SYSTEM or from administrator to SYSTEM, depending on the program. Programs that fall victim to DLL hijacking may appear to behave normally because malicious DLLs may be configured to also load the legitimate DLLs they were meant to replace.

Sources

[I]: Microsoft. (2018, May 31). Dynamic-Link Library Search Order. Retrieved November 30, 2014.
[II]: Harbour, N. (2010, July 15). Malware Persistence without the Windows Registry. Retrieved November 17, 2020.
[III]: Grunzweig, J., Lee, B. (2016, January 22). New Attacks Linked to C0d0so0 Group. Retrieved August 2, 2018.
[IV]: Hexacorn. (2013, December 8). Beyond Good Ol' Run Key, Part 5. Retrieved January 20, 2022.
[V]: Trend Micro. (2010, October 1). STUXNET Malware Targets SCADA Systems. Retrieved January 20, 2022.
[VI]: Amanda Steward. (2014). FireEye DLL Side-Loading: A Thorn in the Side of the Anti-Virus Industry. Retrieved March 13, 2020.
[VII]: Yotam Gottesman. (2016, August 18). Elastic boundaries – elevating privileges with environment variables expansion. Retrieved January 20, 2021.

4 Summary and conclusion

This article has explored two problems with two separate yet closely related sub-techniques that are currently found in MITRE ATT&CK: DLL Search Order Hijacking (T1574.001) and DLL Side-Loading (T1574.002). It is argued that the terms are often confused, used incorrectly and have gaps with regards to the coverage of similar techniques.

Various options for solving this problem were presented, discussing the advantages and disadvantages of each approach. Having compared the impact of each solution, a case is made for a single sub-technique for 'DLL Hijacking' that should replace the existing techniques T1574.001 and T1574.002. A sample definition page for this new sub-technique is provided, aiming to demonstrate how it solves the problems identified.

Cyber security is a very dynamic discipline that changes constantly, as a result of constant security research, new insights, as well as the ongoing cat-and-mouse game between adversaries and defenders. MITRE have accepted the difficult challenge of structuring adversarial techniques and standardising terminology, and have delivered the now industry-wide [5] accepted ATT&CK framework for this. A key factor to its success is the synergy between ATT&CK and the community: one cannot do without the other. The author of this article hopes that the suggestions in this article may help improve a next version of ATT&CK; a small contribution in the much bigger goal of bringing the community together in defending against adversaries.

References

[1] MITRE ATT&CK. Hijack execution flow: DLL Search Order Hijacking, sub-technique T1574.001 - enterprise — MITRE ATT&CK (version 10), April 2021. URL https://attack.mitre.org/versions/v10/techniques/T1574/001/.

[2] Red Teaming Experiments. Dll proxying for persistence, 2020. URL https://www.ired.team/offensive-security/persistence/dll-proxying-for-persistence.

[3] FireEye. DLL Side-Loading: A Thorn in the Side of the Anti-Virus Industry, 2014. URL https://www.mandiant.com/resources/dll-side-loading-a-thorn-in-the-side-of-the-anti-virus-industry.

[4] Fortinet. Elastic Boundaries – Elevating Privileges with Environment Variables Expansion, August 2016. URL https://www.fortinet.com/blog/threat-research/elastic-boundaries-elevating-privileges-with-environment-variables-expansion.

[5] Anna Georgiadou, Spiros Mouzakitis, and Dimitris Askounis. Assessing MITRE ATT&CK risk using a cyber-security culture framework. *Sensors*, 21(9), 2021. ISSN 1424-8220. doi: 10.3390/s21093267. URL https://www.mdpi.com/1424-8220/21/9/3267.

[6] Hexacorn. Beyond good ol' run key, part 5, December 2013. URL https://www.hexacorn.com/blog/2013/12/08/beyond-good-ol-run-key-part-5/.

[7] Trend Micro. STUXNET Malware Targets SCADA Systems, October 2010. URL https://www.trendmicro.com/vinfo/de/threat-encyclopedia/web-attack/54/stuxnet-malware-targets-scada-systems.

[8] Oddvar Moe. Extexport - LOLBAS, May 2018. URL https://lolbas-project.github.io/lolbas/Binaries/Extexport/.

[9] Wikileaks. Kaspersky "heapgrd" DLL Inject, March 2017. URL https://wikileaks.org/ciav7p1/cms/page_3375327.html.