Intruders in the Library: Exploring DLL Hijacking

By Tom Fakterman, Chen Erlich, Assaf Dahan

Published: 2024-02-23 · Archived: 2026-04-05 15:12:23 UTC

Executive Summary

Dynamic-link library (DLL) hijacking is one of the oldest techniques that both threat actors and offensive security professionals continue to use today. DLL hijacking is popular because it grants threat actors a stealthy way to run malware that can be very effective at evading detection. At its core, DLL hijacking tricks an operating system into running a malicious binary instead of a legitimate DLL.

This article explains how threat actors use DLL hijacking in malware attacks, and it should help readers by providing:

- Theoretical background necessary to understand DLL hijacking
- Explanations that demystify some of the concepts around this technique
- Common variations seen in the wild
- Real-world examples from both advanced persistent threat (APT) and cybercrime threat actors

This article also provides ideas for how to better detect DLL hijacking, and we share best practices on how to reduce the risk of attack.

Palo Alto Networks customers are better protected from the threats discussed in this article through our Next-Generation Firewall, as well as Advanced WildFire, DNS Security, and Advanced URL Filtering. Cortex XDR and XSIAM detect known and novel DLL hijacking attacks. The Prisma Cloud Defender agent can assist in identifying malware that uses DLL hijacking techniques. If you think you might have been compromised or have an urgent matter, contact the Unit 42 Incident Response team.

What Is DLL Hijacking?

DLL files are programs that are meant to be run by other programs in Microsoft Windows. DLL hijacking allows attackers to trick a legitimate Windows program into loading and running a malicious DLL. Adversaries leverage DLL hijacking for multiple purposes, including defense evasion, privilege escalation and persistence.

DLL hijacking has evolved, with many variations over the past several years. To understand DLL hijacking, we must first understand the DLL search order mechanism, which is a crucial function in Microsoft Windows.

Windows DLL Search Order

DLL hijacking relies on the DLL search order that Windows uses when loading DLL files. This search order is a sequence of locations a program checks when loading a DLL. The sequence can be divided into two parts: special search locations and standard search locations. You can find the search order comprising both parts in Figure 1.

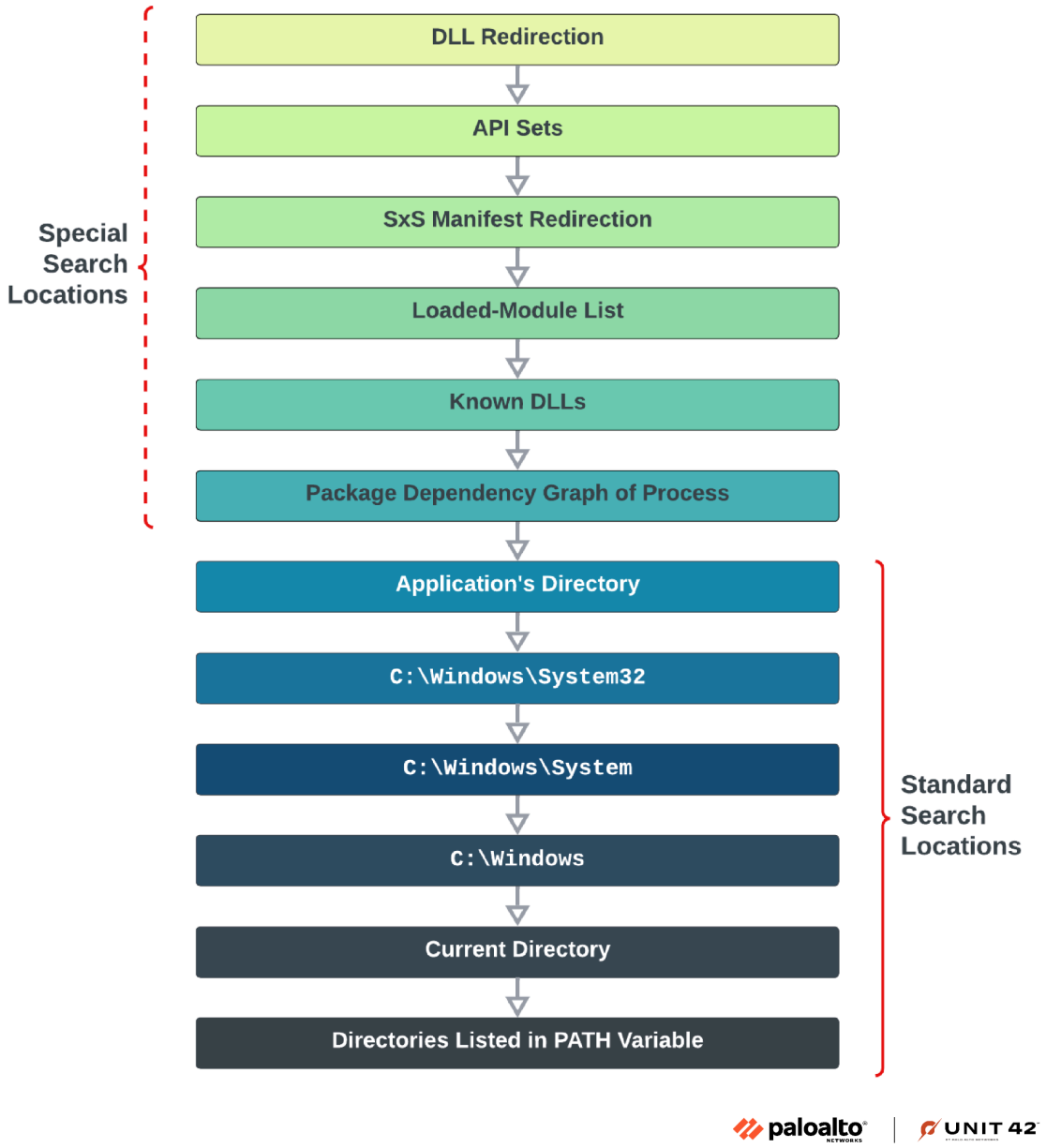


Figure 1. Flow chart of the Windows DLL search order.

Special Search Locations

Special search locations are taken into account before the standard search locations, and they contain different factors that can control the locations to be searched and used to load a DLL. These locations are based on the application and the system configurations.

1. DLL redirection allows specifying which DLL should be loaded by the DLL loader
2. API sets allows dynamically routing function calls to the appropriate DLL based on the version of Windows and the availability of different features
3. SxS manifest redirection redirects DLL loading by using application manifests
4. Loaded-module list verifies whether the DLL is already loaded into memory
5. Known DLLs checks whether the DLL name and path match the Windows list of known DLLs. This list resides in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session

Manager\KnownDLLs
6. The package dependency graph of the process, in case it was executed as part of a packaged app

Standard Search Locations

The standard search locations are the ones most associated with the DLL hijacking technique, and they will usually be used by adversaries. Windows will use the following order to search for the desired DLL.

1. The application's directory (the directory containing the executable)
2. C:\Windows\System32
3. C:\Windows\System
4. C:\Windows
5. The current directory (the directory from which we execute the executable)
6. Directories listed in the PATH environment variable

Hijacking this whole DLL search order will grant an adversary the option to load their malicious DLL within the context of a legitimate application and achieve stealthy execution. They can do this by triggering a malicious DLL to load before the valid one, replacing the DLL or by altering the order (specifically the PATH environment variable).

The prevalence of DLL hijacking has been on the rise in recent years, and DLL hijacking continues to gain popularity. This is because discovering and exploiting the vulnerability in legitimate executables isn't considered to be particularly difficult. However, detecting an attacker loading malicious, camouflaged DLLs within legitimate executables remains a complex undertaking.

Common DLL Hijacking Implementations

As the concept of DLL hijacking continues to evolve over time, threat actors have evolved as well, using different approaches to perform this kind of attack. The three most common techniques we have observed are DLL side-loading, DLL search order hijacking and phantom DLL loading. The most common technique is DLL side-loading.

DLL Side-Loading

In this most commonly used DLL-hijacking technique, an attacker obtains a legitimate executable that loads a specifically named DLL without specifying the DLL file's full directory path. DLL side-loading uses a malicious DLL renamed to the same filename of a legitimate DLL, one normally used by a legitimate executable. Attackers drop the legitimate executable and a malicious, renamed DLL within a directory they have access to.

In DLL side-loading, the attackers rely on the fact that the executable's directory is one of the first locations Windows searches for.

We have studied examples of attackers employing this technique in recent Unit 42 posts, including an instance by the APT Cloaked Ursa (aka APT29), and as part of our Threat Hunting series.

DLL Search Order Hijacking

This implementation exemplifies the core abuse of the entire Windows DLL search order. It is used by adversaries, red teamers and security validation solutions.

This technique simply leverages the Windows DLL search order to drop a malicious DLL in any of its searched locations that would cause a vulnerable, legitimate program to execute a malicious DLL. An attacker can place a malicious DLL in a location prioritized by the DLL search order before the location of a valid DLL. This can happen at any point in the DLL search order, including the PATH environment variable, which attackers can modify by adding a path directory with a malicious DLL.

An example of this type of attack is to drop a malicious DLL in a Python installation directory to hijack the DLL search order. This is an implementation that different security practitioners have already demonstrated.

When Python is installed on a Windows machine, it often adds its installation directory to the PATH environment variable, usually in one of the first searched locations, as shown in Figure 2.

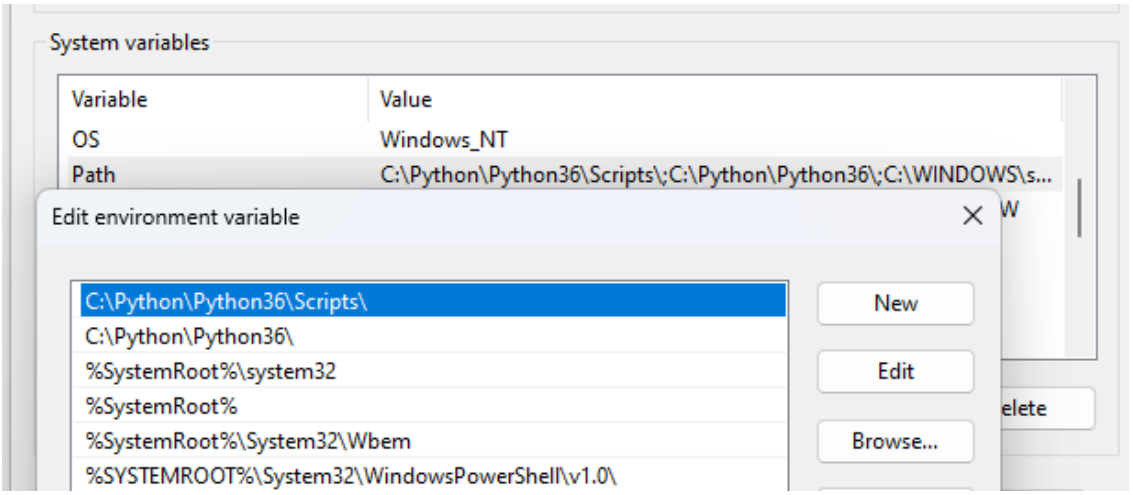


Figure 2. Python folders in the PATH environment variable.

Installing Python on a Windows host creates a directory with relaxed permissions, allowing any authenticated user (including unprivileged ones) to write to this location. This gives attackers the best conditions to execute their DLL search order hijack attack and infect the targeted machine.

Phantom DLL Loading

In this technique, adversaries look for a vulnerable executable that attempts to load a DLL that simply doesn't exist (or is missing) due to an implementation bug. Then, attackers will plant a malicious DLL with the non-existent DLL's filename in its expected location.

A familiar example of this technique is the abuse of the Windows Search (WSearch) Service. This service is responsible for search operations and it launches with SYSTEM privileges upon system startup.

When this service starts, it executes SearchIndexer.exe and SearchProtocolHost.exe, which both attempt to load msfte.dll from System32. In default Windows installations, the file does not exist in this location.

An adversary can plant their malicious DLL if they can write to the System32 folder or an alternate DLL search order location, or insert another attacker-controlled location into the PATH environment variable. This allows

them to gain a stealthy pathway for execution with SYSTEM privileges, and a means to maintain persistence on the machine.

Uncovering Threat Actors and Campaigns

Using our telemetry, we set out to hunt for DLL hijacking attacks, which revealed a large volume of attempted DLL hijacking attacks – including their variations. The following section provides real-world examples of how various threat actors, both cybercrime and nation-state APT groups, use DLL hijacking.

Examples of DLL Hijacking by Nation-State APT Threat Actors

ToneShell's Triple DLL Side-Loading

In September 2023, Unit 42 researchers discovered attackers using DLL side-loading to install the ToneShell backdoor. Attacks using a ToneShell variant were linked to Stately Taurus, in a campaign that built upon three DLL components working in tandem as shown in Figure 3. In the image, each component has been paired with its associated Image Load event in Cortex. The action type shows that the malicious DLLs were loaded to each legitimate process.

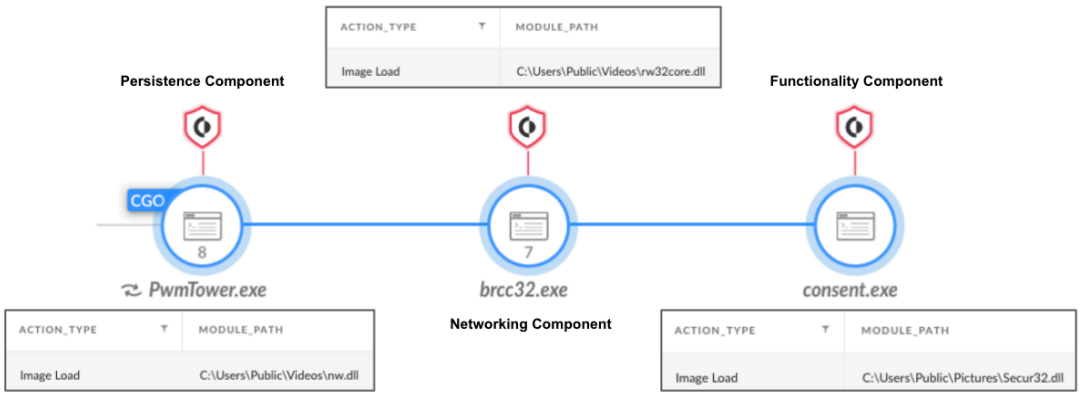


Figure 3. ToneShell process tree in Cortex XDR.

Each DLL component holds a different purpose:

- **Persistence components (nw.dll, nw_elf.dll):** These DLLs are in charge of persistence for the backdoor, as well as dropping the other components to disk.
- **Networking component (rw32core.dll):** This DLL is in charge of command and control (C2) communication.
- **Functionality component (secur32.dll):** This DLL is in charge of executing the different commands of the backdoor.

The persistence components (nw.dll, nw_elf.dll) are side-loaded by PwmTower.exe, a component of a password manager, which is a legitimate security tool.

The networking component (rw32core.dll) is side-loaded by Brcc32.exe, the resource compiler of Embarcadero, an app development tool.

The functionality component (secur32.dll) is side-loaded by Consent.exe, which is a Windows binary described as "Consent UI for administrative applications."

PlugX RAT Leverages DLL Side-Loading to Remain Undetected

Another recent example of a DLL side-loading alert that caught our attention was an attack using the infamous PlugX backdoor.

PlugX is a modular backdoor that is predominantly used by various Chinese APT groups like PKPLUG. PlugX developers circulate in underground hacking communities, and the malware binaries can be found online, so non-Chinese threat actors can also use PlugX.

In the following example, PlugX infected a machine via a compromised USB device. Figure 4 shows the contents of the USB device. This device contained a directory named History and a Windows Shortcut (LNK) file. The History folder's name and icon were disguised as the Windows History folder, and the LNK file uses an icon to appear as a removable disk.

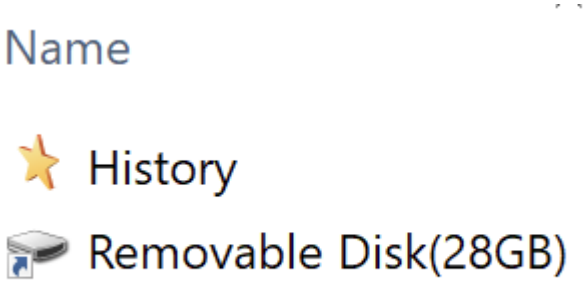


Figure 4. Fake History folder and malicious link file.

The fake History folder contains three files:

- 3.exe
  - A renamed Acrobat.exe file (a legitimate component of Adobe Acrobat)
- Acrobat.dll
  - The PlugX loader, renamed to appear to be a legitimate Adobe Acrobat file
- AcrobatDC.dat
  - A malicious payload that the PlugX loader decrypts in memory

Once the victim clicks the removable disk LNK, it launches the 3.exe process. Then 3.exe loads the PlugX component named Acrobat.dll via DLL side-loading.

Next, the malware creates a directory at C:\ProgramData\AcroBat\AcrobatAey and copies the three files to this location as Acrobat.exe, Acrobat.dll and AcrobatDC.dat, respectively.

To achieve persistence, this PlugX sample creates a scheduled task named InternetUpdateTask, which it sets to run every 30 minutes.

Figure 5 shows the initial process tree of the infection in Cortex XDR.

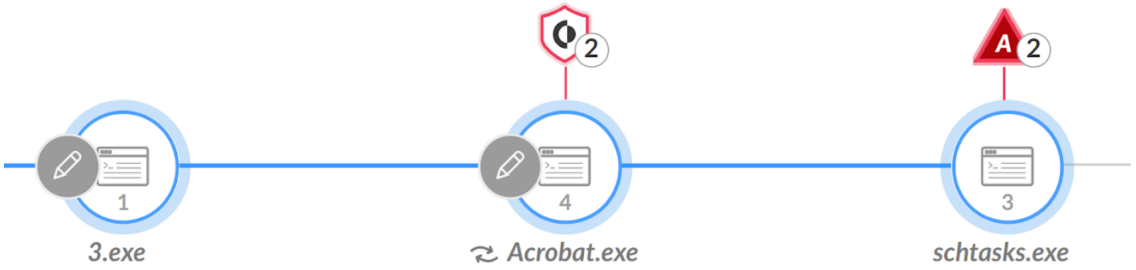


Figure 5. Process tree of initial execution of PlugX.

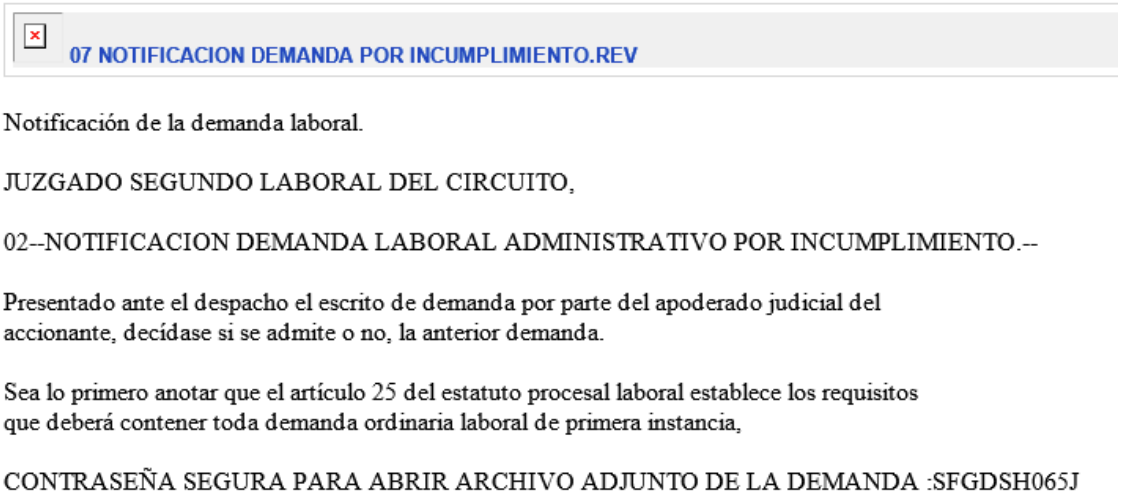
Examples of DLL Hijacking by Cybercrime Threat Actors

Uncovering AsyncRAT Phishing Campaign Targeting South American Organizations

By hunting for DLL side-loading alerts in Cortex XDR Analysis, we discovered a phishing campaign targeting victims mainly in Colombia and Argentina, aiming to deliver AsyncRAT.

AsyncRAT is open-source malware that is very popular among cybercriminals. It gives attackers a range of capabilities such as executing commands, screen capturing and key logging.

The infection starts with phishing emails written in Spanish that contain descriptions of required legal actions, as shown in Figure 6.



07 NOTIFICACION DEMANDA POR INCUMPLIMIENTO.REV

Notificación de la demanda laboral.

JUZGADO SEGUNDO LABORAL DEL CIRCUITO,

02--NOTIFICACION DEMANDA LABORAL ADMINISTRATIVO POR INCUMPLIMIENTO.--

Presentado ante el despacho el escrito de demanda por parte del apoderado judicial del accionante, decídase si se admite o no, la anterior demanda.

Sea lo primero anotar que el artículo 25 del estatuto procesal laboral establece los requisitos que deberá contener toda demanda ordinaria laboral de primera instancia,

CONTRASEÑA SEGURA PARA ABRIR ARCHIVO ADJUNTO DE LA DEMANDA :SFGDSH065J

Figure 6. Text of a phishing mail to deliver AsyncRAT.

The emails also contain links to a Google Drive URL hosting a malicious ZIP archive.

These archive files contain an executable with the same name as the ZIP filename and a malicious DLL file named http_dll.dll.

The executable is actually a renamed legitimate component of the ESET HTTP Server service process, originally named EHttpSrv.exe. When the victim executes the renamed EHttpSrv.exe, it loads the malicious http_dll.dll file

from the same directory via DLL side-loading. After the executable loads http_dll.dll, the DLL unpacks in memory and loads the AsyncRAT malware.

Figure 7 shows the infection chain as seen in Cortex XDR. The malicious ZIP archive is downloaded, extracted with 7-Zip (7zG.exe) and the renamed EHttpSrv.exe is executed.

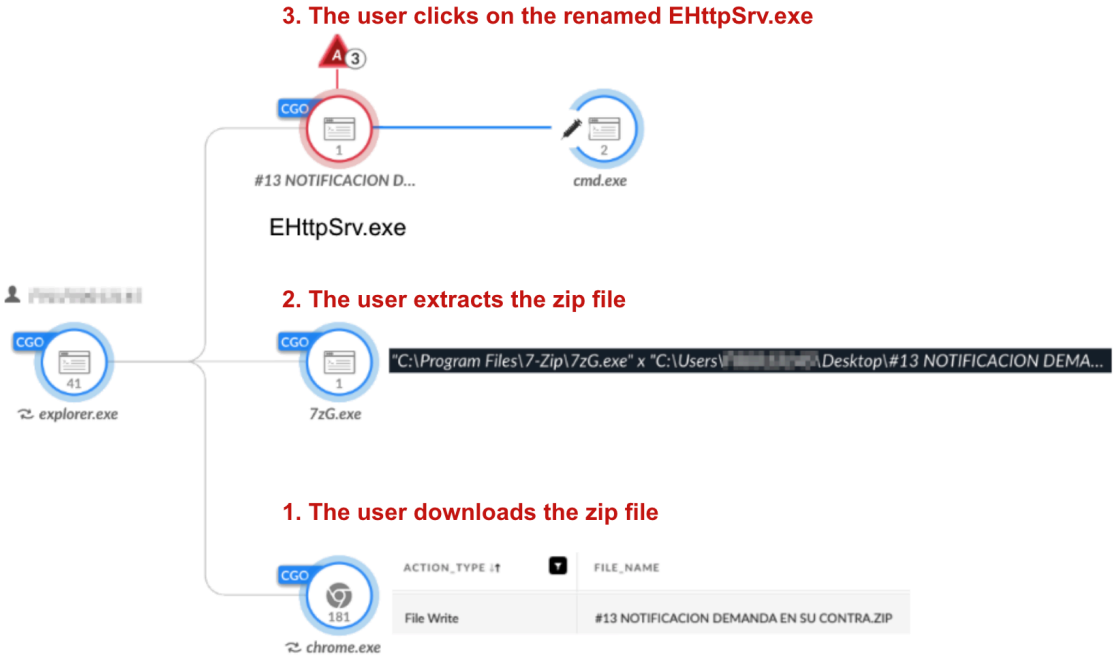


Figure 7. AsyncRAT infection in Cortex XDR.

Figure 8 shows the “Possible DLL Side-Loading” alert Cortex XDR raised for this chain of events.

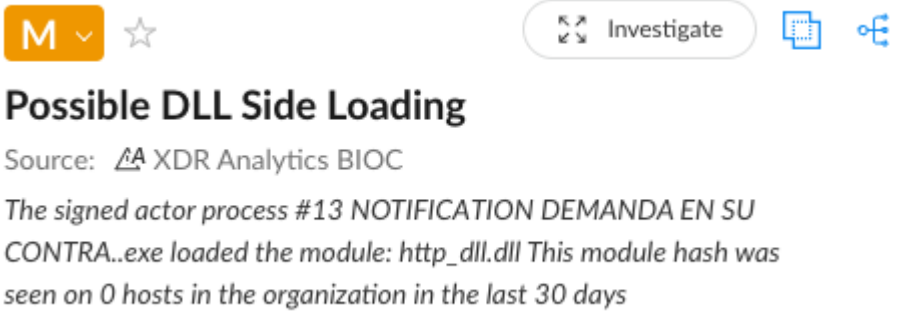


Figure 8. Possible DLL Side Loading alert in Cortex XDR.

Phantom DLL Loading for CatB Ransomware

CatB ransomware was first seen in December 2022. In at least one campaign since then, threat actors have abused the Distributed Transaction Coordinator (MSDTC) service to achieve phantom DLL loading for CatB ransomware.

The core of this CatB ransomware campaign consists of two components: a dropper DLL and a ransomware DLL. The dropper DLL performs different anti-sandbox and anti-virtual machine (VM) checks to ensure the environment is safe to drop its ransomware payload.

After the dropper DLL is satisfied the environment is clear, it writes a second DLL named oci.dll under the C:\Windows\System32 directory. Then, the dropper kills the MSDTC process by msdtc.exe as shown below in Figure 9.

Figure 9. Execution of the dropper DLL in Cortex XDR.

This is done to implement phantom DLL loading. When msdtc.exe launches, it attempts to load a DLL named oci.dll, which does not usually exist in the System32 folder. When msdtc.exe relaunches, it loads the malicious oci.dll, which is the ransomware payload, as shown in Figure 10. In the image, the process msdtc.exe is paired with the Image Load event in Cortex for the malicious oci.dll.

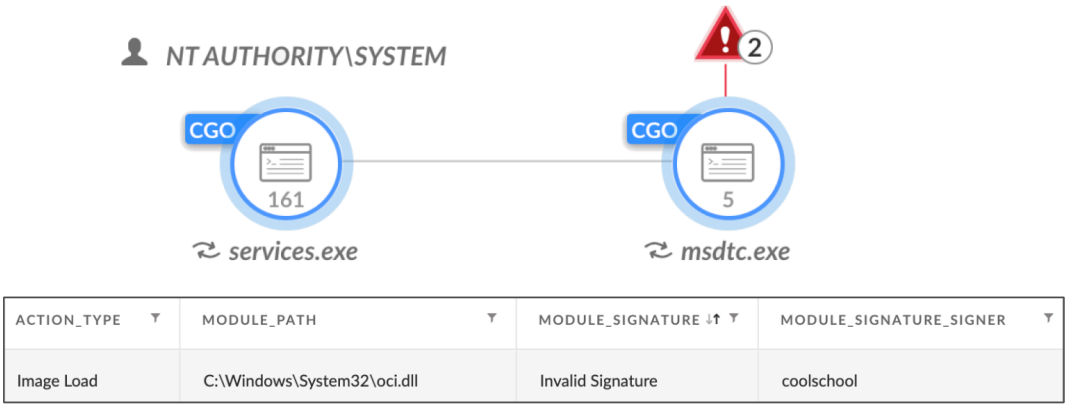


Figure 10. Msdtc.exe loads the malicious oci.dll module shown in Cortex XDR.

Cortex XDR alerts on phantom DLL loading attempts, as shown in Figure 11.

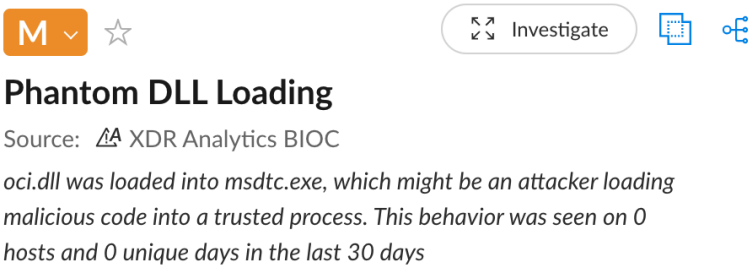


Figure 11. Phantom DLL Loading alert in Cortex XDR.

Abuse of Microsoft DLLs Leads to Dridex

Threat actors have implemented DLL side-loading for another well-known malware, the Dridex banking Trojan. The initial infection vector for Dridex has most often been malicious emails or web traffic.

When executed, the Dridex loader has used AtomBombing to inject code into the process space used by explorer.exe. Next, the injected explorer.exe process writes Dridex DLLs as .tmp files and shell scripts with random names to the user's TEMP directory. An example of these files being written to disk is shown in Figure 12.

ACTION_TYPE	FILE_PATH
File Write	C:\Users\ \AppData\Local\Temp\jJu411A.tmp
File Write	C:\Users\ AppData\Local\Temp\FFAK.cmd
File Write	C:\Users\ AppData\Local\Temp\JZ4822B.tmp
File Write	C:\Users\ AppData\Local\Temp\bopc.cmd

Figure 12. Malicious files written to disk by explorer.exe shown in Cortex XDR.

Up to three of the shell scripts can appear, and they create the persistent Dridex infection in three different locations under random directory paths on the victim's host. The persistent infection uses DLL side-loading.

The shell scripts create these randomly-named directories under random directory paths, copy legitimate Microsoft executables and rename the Dridex DLL .tmp files for the DLL side-loading. An example of two shell scripts are shown below in Figure 13.

```
md C:\Windows\system32\W9RGOv
copy C:\Windows\system32\DeviceEnroller.exe C:\Windows\system32\W9RGOv
move C:\Users\USER\AppData\Local\Temp\JZ4822B.tmp C:\Windows\system32\W9RGOv\XmlLite.dll
del %0 & exit
```

```
md C:\Users\User\AppData\Roaming\oYgxmq
copy C:\Windows\system32\dpnsvr.exe C:\Users\User\AppData\Roaming\oYgxmq
move C:\Users\User\AppData\Local\Temp\jJu411A.tmp C:\Users\User\AppData\Roaming\oYgxmq\TAPI32.dll
del %0 & exit
```

Figure 13. The shell scripts that copy Dridex.

Afterward, the injected explorer.exe process creates persistence for the copied binaries using up to three methods:

1. A registry update under HKCU\SOFWARE\Microsoft\Windows\CurrentVersion\Run (example in Figure 14)
2. A Windows shortcut under the user's AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup folder
3. A scheduled task

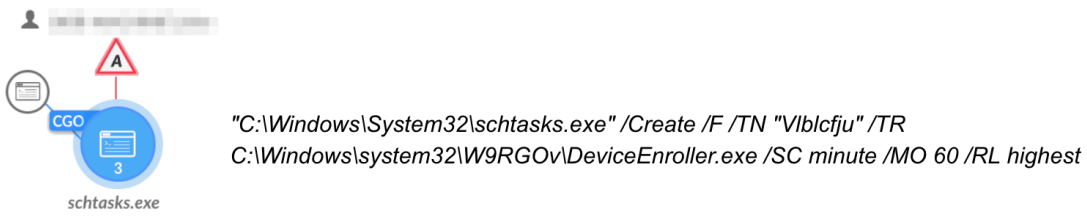


Figure 14. Cortex XDR alert on Dridex creating a scheduled task for persistence.

Figure 15 shows Cortex XDR alerting on the legitimate file DeviceEnroller.exe side-loading a malicious Dridex DLL.

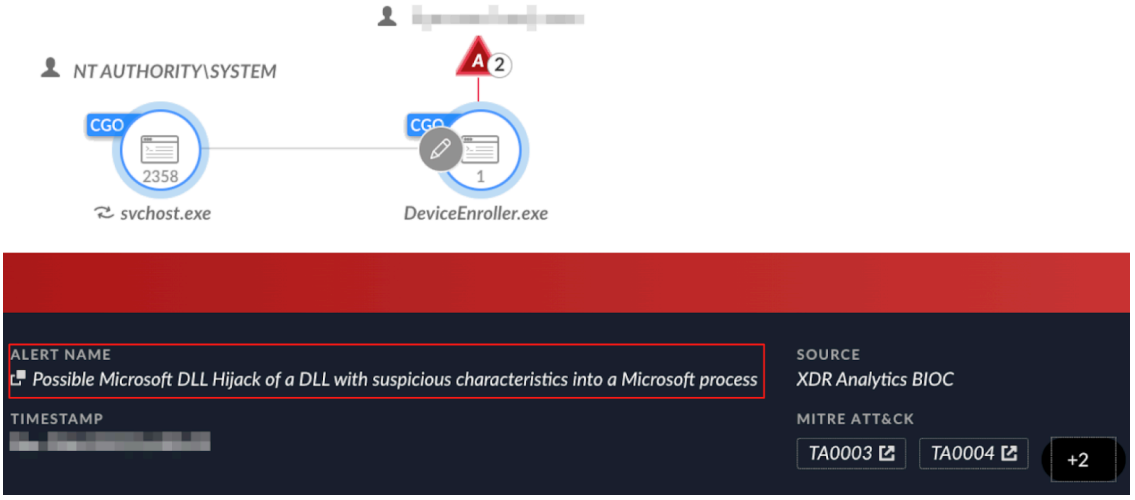


Figure 15. Alert for the legitimate DeviceEnroller.exe side-loads the malicious Dridex DLL.

Principles for Efficient DLL Hijacking Detection

Pinpointing instances where an executable unexpectedly loads a malicious DLL with an identical name, but that is otherwise different in its content, is a rather challenging task. This challenge significantly increases when attempting to detect these behavioral anomalies at scale.

In this section we provide several principles for effective detection of DLL hijacking, including its variations. The principles will focus on the malicious DLL, the vulnerable application and the loading event, where a vulnerable application loads the malicious DLL.

Malicious DLL

Since the malicious DLL has the same name as a legitimate DLL, we look for abnormalities. For example:

- No digital signature or a stolen signature
- An unusual file size
- Unusually high or low entropy
- A rare file hash (compared to baseline) in the organization
- DLL compilation time significantly newer than the loading application
- A DLL placed in a path it doesn’t usually reside in

Vulnerable Application

The vulnerable application is usually a legitimate one to allow better disguise for the malicious DLL execution. Given that, we proceed to seek out distinct traits:

- Usually a valid digital signature
- Trusted vendors (antivirus, browsers, VPNs, Microsoft applications) are a common target
- Commonly abused application (by hash or version)
- In DLL side-loading
    - It will usually be an uncommon application in the organization
    - It will usually use an uncommon directory (e.g., C:\Users\<Username>\AppData, C:\ProgramData)

Loading Event

We can find different abnormalities also within the loading event. For example:

- The first time the application loads a suspected DLL name and/or its hash
- The application usually loads several DLLs, but now it loads only one

Mitigating the DLL Hijacking Attack Surface

To secure applications from possible DLL hijacking attacks, developers need to be cognizant of this attack technique and integrate diverse protective measures.

Microsoft has published a DLL security article covering several best practices to support developers in this effort, including the following:

- Wherever possible, specify a fully qualified path when loading DLLs or triggering new process executions.
- Gain more control of your application behavior by utilizing DLL redirection and manifests.
- Do not assume the operating system version when users execute an application. Develop your application to be handled as intended in all OSes.

Conclusion

This article covers DLL hijacking, providing the technical background needed to understand how threat actors weaponize it, along with an explanation of popular variations in its implementation.

In addition, we provide examples that demonstrate how various threat actors – both APT nation-state and cybercrime groups – rely on this technique to achieve stealth, persistence and privilege escalation in their operations.

Lastly, we discuss possible approaches for detecting and mitigating DLL hijacking in enterprise environments.

Protections and Mitigations

For Palo Alto Networks customers, our products and services provide the following coverage associated with the threats described above:

- Next-Generation Firewall and Advanced WildFire accurately identifies known samples as malicious.
- Advanced URL Filtering and DNS Security identify domains associated with this group as malicious.
- Prisma Cloud
    - When paired with the WildFire integration, the Prisma Cloud Defender agent will identify malicious binaries and make verdict determinations when analyzing executing processes.
    - When paired with XSIAM, the Prisma Cloud Defender is enabled to block malicious processes from operating within the cloud environment.
    - Prevents the execution of known malicious malware, and also prevents the execution of unknown malware using Behavioral Threat Protection and machine learning based on the Local Analysis module.
- Cortex XDR and XSIAM
    - Detects known and novel DLL hijacking attacks, using the new generic Analytics DLL Hijacking tag.
    - Prevents the execution of known malicious malware, and also prevents the execution of unknown malware using Behavioral Threat Protection and machine learning based on the Local Analysis module.
    - Protects against credential gathering tools and techniques using the new Credential Gathering Protection available from Cortex XDR 3.4.
    - Protects from threat actors dropping and executing commands from web shells using Anti-Webshell Protection, newly released in Cortex XDR 3.4.
    - Protects against exploitation of different vulnerabilities including ProxyShell and ProxyLogon using the Anti-Exploitation modules as well as Behavioral Threat Protection.
    - Cortex XDR Pro detects post exploit activity, including credential-based attacks, with behavioral analytics.

If you think you might have been impacted or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America Toll-Free: 866.486.4842 (866.4.UNIT42)
- EMEA: +31.20.299.3130
- APAC: +65.6983.8730
- Japan: +81.50.1790.0200

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

The following are SHA256 hashes of files from the examples used in this article.

AsyncRAT ZIP

- 26fc0efa8458326086266aae32ec31b512adddd1405f4dd4e1deed3f55f7b30d
- 0709e3958f343346406c5a26029748f5d15101d3b7d8b8c1119f7642754ae64e
- 5e50329c4bcb67a1220f157744e30203727f5a55e08081d1ae65c0db635ce59d
- af8baffceafeda320eab814847dee4df74020cc4b96a4907816335ad9b03c889
- c3ec461e8f3d386a8c49228a21767ff785840bc9ae53377f07ff52d0ccba1ccf
- e00918a579ced5783cefc27b1e1f9f0bc5b0f93a32d4a7170c7466b34cc360df
- e41f58d82394853fc49f2cccae07c06504cc1d1f3d49ba6bfd8f8762948b7c16
- eadd74bbb7df21e45abc07c065876ba831978185c9e0845f19e86c151439020a

AsyncRAT

- 54fc9f4699d8fb59ce1635df5aaa2994b5d924d7b4d7626e1b5d9a406bef899d
- 10fec9bf8d695ab14b1329cc6ca6d303d87617ffa76e3e4cc46f8f542e062d70
- 69985edc2510803cfd862bdf87c59cc963be1bde5e08a0f10c0fd109c2134eab
- 2ea71c9cbb949e96da71716d8a431952632b954c7fc5ba87e6f84684957f07ef
- a8b7aaede89c587525906fa24f392b1ce0b4a73c6193eb6db95b586ae378649c
- 27b8bfe997400a956cd7ec9a3f68e198fe690562d909185b7d41b1e9ce31c53f
- e4fdc02f196cedb98d2098b6993f6e28976abe9b5c8e9f9752dea493b9d1dcb9

PlugX ZIP

- 86a5ce23cf54d75d9c8d9402e233d00f8f84a31324ae8e52da6172e987d9a87b
- dca39474220575004159ecff70054bcf6239803fcf8d30f4e2e3907b5b97129c

PlugX loader

- 12c584a685d9dffbee767d7ad867d5f3793518fb7d96ab11e3636edcc490e1bd

PlugX dat file

- 95205b92d597489b33854e70d86f16d46201803a1a9cb5379c0d6b7c0784dbc7

PlugX LNK file

- 515fd058af3dfd2d33d49b7c89c11c6ef04c6251190536ca735a27e5388aa7e7

Dridex

- f101cc7885e44eee63713a71bba85baa7c135a9b1fe49480e05fc872f84993e7
- 3f98a3e8ea69daf06e6da6e8d495bba42e575dbd0ba26f5e6035efb017545be1
- 2f043922d42fbef8d1a08395bf0928d6181863c44b53bccc8c3806796db1c50e
- 25085c4f707583052d7070ddb5473bb0684e588694279c7f85e4c17e36837074
- 8a1c5858440a3eaa91f7442b7453127432f240637d22793dca6bfe5406776fbe
- fbc4421f8454139f4e2ebd808ebb224c0d773b0d62f69ef2270da386a4aab3e7

- 0d4a7b43b5dbe8b8492c51a3f7595c8e188d558390ee1ab0586d1315b98619c9
- 98ebb3e797e19e0e6aeffc6d03e7ad5ce76f941a175c3cacc3a7f0056d224f95
- 0989a9be27bdc8827008f1837e62d88a077f8541a7b080e367b08facd9382962
- 4a6ebd82b30063c73283b5364e34fc735ad05b5dd62bfa77f38617e9b2937444

ToneShell Persistence Component

- 2f5cf595ac4d6a59be78a781c5ba126c2ff6d6e5956dc0a7602e6ba8e6665694
- 0f2f0458d2f1ac4233883e96fe1f4cc6db1551cdcfdd49c43311429af03a1cd5
- 011fe9974f07cb12ba30e69e7a84e5cb489ce14a81bced59a11031fc0c3681b7
- 3fc4d023d96f339945683f6dc7d9e19a9a62b901bef6dc26c5918ce9508be273
- 3a429b8457ad611b7c3528e4b41e8923dd2aee32ccd2cc5cf5ff83e69c1253c2
- f58d3d376c8e26b4ae3c2bbaa4ae76ca183f32823276e6432a945bcbc63266d9
- 46c6ee9195f3bd30f51eb6611623aad1ba17f5e0cde0b5523ab51e0c5b641dbf
- 86140e6770fbd0cc6988f025d52bb4f59c0d78213c75451b42c9f812fe1a9354

ToneShell Networking Component

- a08e0d1839b86d0d56a52d07123719211a3c3d43a6aa05aa34531a72ed1207dc
- 19d07dbc58b8e076cafd98c25cae5d7ac6f007db1c8ec0fae4ce6c7254b8f073
- 8e801d3a36decc5e4ce6fd3e8e45b098966aef8cbe7535ed0a789575775a68b6
- df4ba449f30f3ed31a344931dc77233b27e06623355ece23855ee4fe8a75c267
- 345ef3fb73aa75538fdcf780d2136642755a9f20dbd22d93bee26e93fb6ab8fd
- 3a5e69786ac1c458e27d38a966425abb6fb493a41110393a4878c811557a3b5b

ToneShell Functionality Component

- 66b7983831cbb952ceeb1ffff608880f1805f1df0b062cef4c17b258b7f478ce
- f2a6a326fb8937bbc32868965f7475f4af0f42f3792e80156cc57108fc09c034
- dafa952aacf18beeb1ebf47620589639223a2e99fb2fa5ce2de1e7ef7a56caa0
- 52cd066f498a66823107aed7eaa4635eee6b7914acded926864f1aae59571991

CatB Loader

- 3661ff2a050ad47fdc451aed18b88444646bb3eb6387b07f4e47d0306aac6642

CatB Payload

- c8e0aa3b859ac505c2811eaa7e2004d6e3b351d004739e2a00a7a96f3d12430c
- 83129ed45151a706dff8f4e7a3b0736557f7284769016c2fb00018d0d3932cfa
- 35a273df61f4506cdb286ecc40415efaa5797379b16d44c240e3ca44714f945b
- 9990388776daa57d2b06488f9e2209e35ef738fd0be1253be4c22a3ab7c3e1e2
