

Detecting DLL hijacking with machine learning: real-world cases

Gleb Ivanov : : 10/6/2025



Authors

-  [Gleb Ivanov](#)
-  [Andrey Gunkin](#)

Introduction

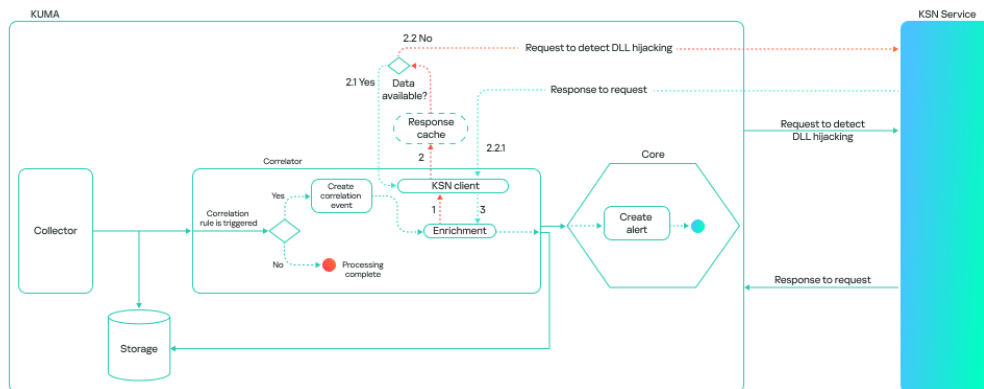
Our colleagues from the AI expertise center recently developed a machine-learning model that detects DLL-hijacking attacks. We then integrated this model into the [Kaspersky Unified Monitoring and Analysis Platform](#) SIEM system. In [a separate article](#), our colleagues shared how the model had been created and what success they had achieved in lab environments. Here, we focus on how it operates within Kaspersky SIEM, the preparation steps taken before its release, and some real-world incidents it has already helped us uncover.

How the model works in Kaspersky SIEM

The model's operation generally boils down to a step-by-step check of all DLL libraries loaded by processes in the system, followed by validation in the Kaspersky Security Network (KSN) cloud. This approach allows local attributes (path, process name, and file hashes) to be combined with a global knowledge base and behavioral indicators, which significantly improves detection quality and reduces the probability of false positives.

The model can run in one of two modes: on a correlator or on a collector. A correlator is a SIEM component that performs event analysis and correlation based on predefined rules or algorithms. If detection is configured on a correlator, the model checks events that have already triggered a rule. This reduces the volume of KSN queries and the model's response time.

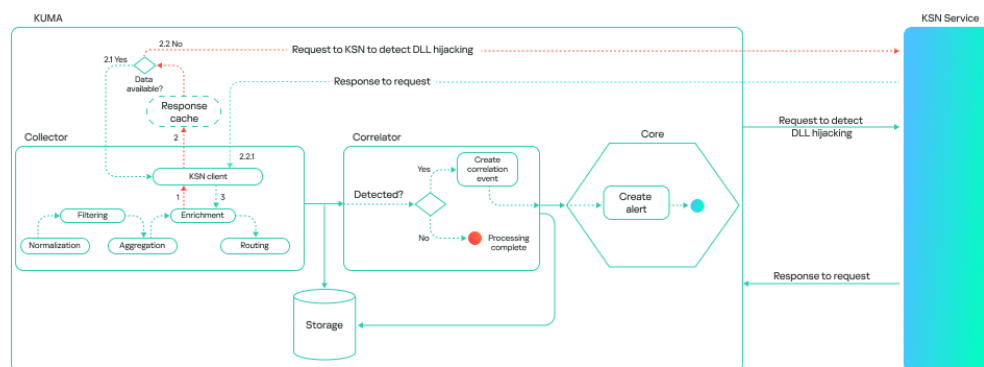
This is how it looks:



A collector is a software or hardware component of a SIEM platform that collects and normalizes events from various sources, and then delivers these events to the platform's core. If detection is configured on a collector, the model processes all events associated with various processes loading libraries, provided these events meet the following conditions:

- The path to the process file is known.
- The path to the library is known.
- The hashes of the file and the library are available.

This method consumes more resources, and the model's response takes longer than it does on a correlator. However, it can be useful for retrospective threat hunting because it allows you to check all events logged by Kaspersky SIEM. The model's workflow on a collector looks like this:



It is important to note that the model is not limited to a binary "malicious/non-malicious" assessment; it ranks its responses by confidence level. This allows it to be used as a flexible tool in SOC practice. Examples of possible verdicts:

- 0: data is being processed.
- 1: maliciousness not confirmed. This means the model currently does not consider the library malicious.
- 2: suspicious library.
- 3: maliciousness confirmed.

A Kaspersky SIEM rule for detecting DLL hijacking would look like this:

1 N.KL_AI_DLLHijackingCheckResult > 1

Embedding the model into the Kaspersky SIEM correlator automates the process of finding DLL-hijacking attacks, making it possible to detect them at scale without having to manually analyze hundreds or thousands of loaded libraries. Furthermore, when combined with correlation rules and telemetry sources, the model can be used not just as a standalone module but as part of a comprehensive defense against infrastructure attacks.

Incidents detected during the pilot testing of the model in the MDR service

Before being released, the model (as part of the Kaspersky SIEM platform) was tested in the [MDR](#) service, where it was trained to identify attacks on large datasets supplied by our telemetry. This step was necessary to ensure that detection works not only in lab settings but also in real client infrastructures.

During the pilot testing, we verified the model's resilience to false positives and its ability to correctly classify behavior even in non-typical DLL-loading scenarios. As a result, several real-world incidents were successfully detected where attackers used one type of DLL hijacking — the DLL Sideload technique — to gain persistence and execute their code in the system.

Let us take a closer look at the three most interesting of these.

Incident 1. ToddyCat trying to launch Cobalt Strike disguised as a system library

In one incident, the attackers successfully leveraged the vulnerability [CVE-2021-27076](#) to exploit a SharePoint service that used IIS as a web server. They ran the following command:

```
c:\windows\system32\inetsrv\w3wp.exe -ap "SharePoint - 80" -v "v4.0" -l "webengine4.dll" -a
1 \\.\pipe\iisipmd32ded38-e45b-423f-804d-34471928538b -h "C:\inetpub\temp\appools\SharePoint -
80\SharePoint - 80.config" -w "" -m 0
```

After the exploitation, the IIS process created files that were later used to run malicious code via the DLL sideloading technique ([T1574.001 Hijack Execution Flow: DLL](#)):

1 C:\ProgramData\SystemSettings.exe

2 C:\ProgramData\SystemSettings.dll

SystemSettings.dll is the name of a library associated with the Windows Settings application (SystemSettings.exe). The original library contains code and data that the Settings application uses to manage and configure various system parameters. However, the library created by the attackers has malicious functionality and is only pretending to be a system library.

Later, to establish persistence in the system and launch a DLL sideloading attack, a scheduled task was created, disguised as a Microsoft Edge browser update. It launches a SystemSettings.exe file, which is located in the same directory as the malicious library:

```
1 Schtasks /create /ru "SYSTEM" /tn "\Microsoft\Windows\Edge\Edgeupdates" /sc DAILY /tr
"C:\ProgramData\SystemSettings.exe" /F
```

The task is set to run daily.

When the SystemSettings.exe process is launched, it loads the malicious DLL. As this happened, the process and library data were sent to our model for analysis and detection of a potential attack.

Correlation event details

×

 Copy

 Detailed view

TenantName	Main
Timestamp	2024-09-03 19:17:21 :544
Name	Dll_hijacking_ml_detect
StartTime	2024-09-03 19:17:21 :870
EndTime	2024-09-03 19:17:21 :870
DeviceEventCategory	module
DeviceHostName	
DeviceProduct	KUMA
DeviceTimeZone	+03:00
DeviceVendor	Kaspersky
CorrelationRule	Dll hijacking_ml_detect
Service	Correlator
BaseEventCount	1
FileHash	E0E092D4EFC15E25FD9C0923C52C33D6
FileName	SystemSettings.exe
FilePath	C:\ProgramData
OldFileHash	686EDEE95099B6A727Q38J6AE3F28272
OldFileName	SystemSettings.dll
OldFilePath	C:\ProgramData
Priority	High
Type	Correlated

Extended event schema fields

N.KL_AI_DLLHijackingCheckR 2 (Suspicious)
result

Example of a SystemSettings.dll load event with a DLL Hijacking module verdict in Kaspersky SIEM

The resulting data helped our analysts highlight a suspicious DLL and analyze it in detail. The library was found to be a [Cobalt Strike](#) implant. After loading it, the SystemSettings.exe process attempted to connect to the attackers' command-and-control server.

1 DNS query: connect-microsoft[.]com

2 DNS query type: AAAA

3 DNS response: ::ffff:8.219.1[.]155;

4 8.219.1[.]155:8443

After establishing a connection, the attackers began host reconnaissance to gather various data to develop their attack.

1 C:\ProgramData\SystemSettings.exe

2 whoami /priv

3 hostname

4 reg query HKLM\SOFTWARE\Microsoft\Cryptography /v MachineGuid

5 powershell -c \$psversiontable

6 dotnet --version

7 systeminfo

8 reg query "HKEY_LOCAL_MACHINE\SOFTWARE\VMware, Inc.\VMware Drivers"

9 cmdkey /list

10 REG query "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server\WinStations\RDP-Tcp" /v PortNumber

11 reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Servers

12 netsh wlan show profiles

13 netsh wlan show interfaces

14 set

15 net localgroup administrators

16 net user

17 net user administrator

18 ipconfig /all

19 net config workstation

20 net view

21 arp -a

22 route print

23 netstat -ano

24 tasklist

25 schtasks /query /fo LIST /v

26 net start

27 net share

28 net use

29 netsh firewall show config

30 netsh firewall show state

31 net view /domain

32

```
33 net time /domain
34 net group "domain admins" /domain
35 net localgroup administrators /domain
36 net group "domain controllers" /domain
37 net accounts /domain
38 nltest / domain_trusts
39 reg query HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
40 reg query HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
41 reg query HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
42 reg query HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Run
    reg query
    HKEY_CURRENT_USER\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\RunOnce
```

Based on the attackers' TTPs, such as [loading Cobalt Strike as a DLL](#), using the DLL sideloading technique (1, 2), and exploiting SharePoint, we can say with a high degree of confidence that the [ToddyCat APT group](#) was behind the attack. Thanks to the prompt response of our model, we were able to respond in time and block this activity, preventing the attackers from causing damage to the organization.

Incident 2. Infostealer masquerading as a policy manager

Another example was discovered by the model after a client was connected to MDR monitoring: a legitimate system file located in an application folder attempted to load a suspicious library that was stored next to it.

```
1 C:\Program Files\Chiniks\SettingSyncHost.exe
2 C:\Program Files\Chiniks\policymanager.dll E83F331BD1EC115524EBFF7043795BBE
```

The SettingSyncHost.exe file is a system host process for synchronizing settings between one user's different devices. Its 32-bit and 64-bit versions are usually located in C:\Windows\System32\ and C:\Windows\SysWOW64\, respectively. In this incident, the file location differed from the normal one.


```

strcpy(fileName, "\\AvastAuth");
v20 |= 0x6D2Cu;
v20 /= 20128;
v20 *= -10603;
strcpy(FileExtension, ".dat");
v20 += 20720;
v20 *= -16593;
v20 |= 0x8102u;
v20 /= 57669;
v10 = 5;
v20 -= 32644;
String2 = 0;
memset(v17, 0, sizeof(v17));
v18 = 0;
v19 = 0;
v20 += 7401;
Count = strlen(fileName);
v20 += 25421;
strncpy(&String2, fileName, Count);
v20 *= -3967;
strncat(&String2, FileExtension, 4u);
v20 >>= 1;
lstrcatA(&FullFileName, &String2);
v20 *= -21067;
v13 = 0;
v20 /= 55465;
data = 0;
v20 /= 18043;
dataLength = 0;
v20 |= 0x800Du;
v20 &= 0xCE77u;
GetFileContents(&FullFileName, (int)&data, (int)&dataLength);

```

Code snippet from the malicious file

The loader opens a file named AvastAuth.dat, which contains an encrypted backdoor. The library reads the data from the file into memory, decrypts it, and executes it. After this, the backdoor attempts to connect to a remote command-and-control server.

The library file, which contains the malicious loader, is on the list of known libraries used for DLL sideloading, as presented on the HijackLibs project website.

Conclusion

Integrating the model into the product provided the means of early and accurate detection of DLL-hijacking attempts which previously might have gone unnoticed. Even during the pilot testing, the model proved its effectiveness by identifying several incidents using this technique. Going forward, its accuracy will only increase as data accumulates and algorithms are updated in KSN, making this mechanism a reliable element of proactive protection for corporate systems.

IoC

Legitimate files used for DLL hijacking

E0E092D4EFC15F25FD9C0923C52C33D6 loads SystemSettings.dll

09CD396C8F4B4989A83ED7A1F33F5503 loads policymanager.dll

A72036F635CECF0DCB1E9C6F49A8FA5B loads wsc.dll

Malicious files

[EA2882B05F8C11A285426F90859F23C6](#) SystemSettings.dll

[E83F331BD1EC115524EBFF7043795BBE](#) policymanager.dll

[831252E7FA9BD6FA174715647EBCE516](#) wsc.dll

Paths

C:\ProgramData\SystemSettings.exe

C:\ProgramData\SystemSettings.dll

C:\Program Files\Chiniks\SettingSyncHost.exe

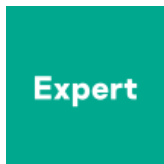
C:\Program Files\Chiniks\policymanager.dll

D:\RECYCLER.BIN\1\CEFHelper.exe

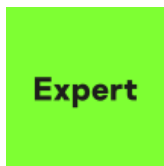
D:\RECYCLER.BIN\1\wsc.dll

- [Security technology](#)
- [Machine learning](#)
- [DLL hijacking](#)
- [Threat hunting](#)
- [DLL sideloading](#)
- [Cybersecurity](#)
- [Artificial intelligence](#)
- [DLL](#)
- [SIEM](#)

Authors



- [Gleb Ivanov](#)



- [Andrey Gunkin](#)

Detecting DLL hijacking with machine learning: real-world cases

Your email address will not be published. Required fields are marked *

[Cancel](#)

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)