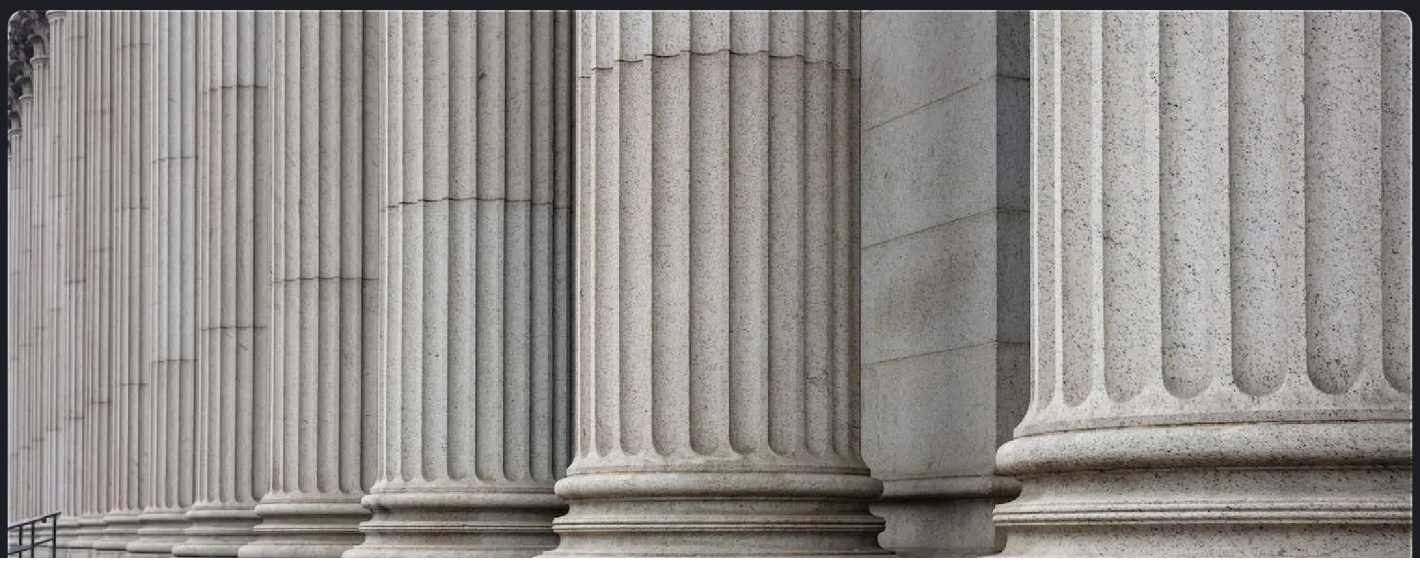
elastic security labs

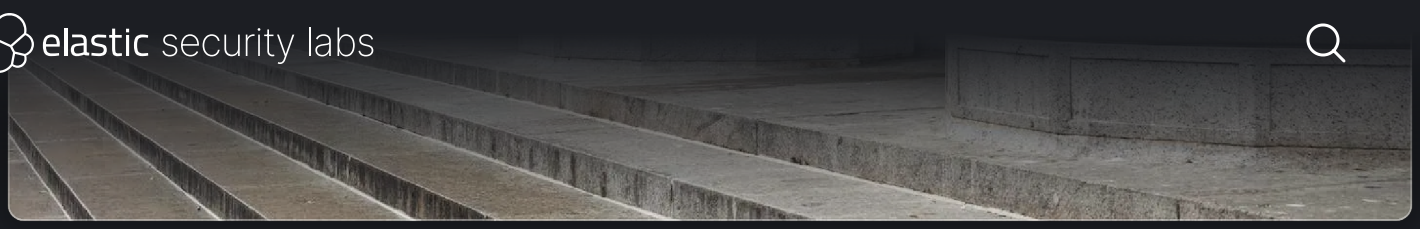
1 MARCH 2023 • SAMIR BOUSSEADEN

Hunting for Suspicious Windows Libraries for Execution and Defense Evasion

Learn more about discovering threats by hunting through DLL load events, one way to reveal the presence of known and unknown malware in noisy process event data.

11 min read Security operations, Security research, Detection science





Dynamic-link library (DLL) image loads is one of the noisiest types of event in Windows, which may discourage defenders from using it for detection engineering or threat hunting. Even if logged in some environments, it's often limited to function-specific DLLs such as scheduled tasks (taskschd.dll), Windows Management Instrumentation (wmiutil.dll) and potentially DLLs loading from a few suspicious folders. In addition to the data volume issue, the false positive (FP) rate of the detection rules using DLL events also tend to be proportional to the data volume.

Unfortunately, both advanced adversaries and also commodity malwares are taking advantage of those limitations to increase the chances of their attack success, especially during the delivery phase via diverse spear phishing procedures.

The most commonly observed delivery techniques are the following :

- Loading malicious DLLs using binary execution proxies Rundll32 and Regsvr32
- Sideloading a malicious DLL from a virtual disk image (ISO/VHD files) into a convenient signed benign binary
- Extracting a DLL from a malicious Microsoft Office document (i.e. Word, Excel) and immediately loading it via Visual Basic for Applications (VBA)
- Downloading or extracting a DLL using a **lolbin** and loading it by another program
- Sideloading a malicious DLL extracted from a compressed archive (zip, rar, etc) into a signed benign binary
- Dropping a malicious DLL in the current directory of an existing program vulnerable to DLL sideloading (e.g. OneDrive, Slack, Teams) via one of several means
- Less common but also very effective is the use of Windows Installer MSIEXEC to load a malicious DLL

elastic security labs

What DLL events do we log with Elastic Endpoint ?

With the exception of the following Microsoft DLLs, Elastic endpoint since version 7.16 records all non-Microsoft signed DLLs:

We also added some enrichments to both DLL and process events that records the following metadata:

Below is an example of device information for DLL and Process execution from mounted ISO and VHD files, two file objects increasingly used to deliver malware:

Here is an example of process execution relative file creation and modification times for svchost.exe :

The relative execution time enrichment will help us create less noisy detection rules (we can match our rules against the first or few image load or process execution instances), and the device information will allow us to better target suspicious use of ISO/VHD files for malicious purposes.

Detection

In this section we share some detection ideas that are both reliable signals and effectively match the most common scenarios we mentioned earlier.

DLL via Rundll32 / Regsvr32

As captured in our own **Global Threat Report**, Rundll32 and Regsvr32 lolbins are two of the most abused binary execution proxies. These utilities can load malicious DLLs and are a commonly seen component of many phishing attacks (malicious shortcuts, ISO file, macro enabled documents):

where the malicious file was a DLL loaded by either regsvr32 or to a lesser degree rundll32.

The following two endpoint behavior protection rules are effective against about 80% of those samples (~17k out of ~21k) leveraging rundll32 or regsvr32 to execute malicious modules: - **Unusual DLL Extension Loaded by Rundll32 or Regsvr32**- **RunDLL32/Regsvr32 Loads Dropped Executable**

Rundll32 or Regsvr32 Executing an oversized File

The following EQL query correlates creation of an executable file event with file size equal or greater than 100MB (this threshold can be adjusted to your environment) subsequently followed by being loaded as a DLL via rundll32 or regsvr32:

Below are examples of malicious control panel (CPL) files with sizes over 700MB, a technique used to bypass AV file scanning and reputation-based cloud services that implement a maximum file size for uploaded files:

Rundll32 or Regsvr32 loading a DLL with a suspicious original file name

Some malicious DLLs have a suspicious original file name, such as ending with .EXE extension or with a great mismatch between the length of the original file name and the actual DLL name. This kind of defense evasion is less common and is employed by a good number of known malware families:

A few examples:

DLL via Disk Images

gained in popularity among commodity malware families.

Suspicious ImageLoad from an ISO Mounted Device

The following rule looks for the execution of commonly-abused Windows binaries to load a DLL from a mounted virtual disk image:

Below are some example of the technique:

Suspicious Microsoft Image Loaded from a Disk Image

The following rule is triggered when an executable, running from a mounted virtual disk image (.vhd, .iso), loads a suspicious Microsoft-signed DLL such as the taskschd, bitsproxy or vaultclient modules that are associated with some common malware capabilities like persistence, credential access, and evasion.

This query identifies many commodity malware families delivered via ISO files:

Potential DLL SideLoad via a Renamed Signed Binary

The following query identifies attempts to load an unsigned DLL from a mounted virtual disk (.iso, .vhd) and using a renamed signed binary (original file name is different than the process name).

This depicts some examples of matches where a signed and renamed program is loading a DLL from a mounted disk image:

Potential DLL SideLoad via a Microsoft Signed Binary

This detection identifies attempts to load unsigned DLLs from a mounted virtual disk (.iso, .vhd) and using a signed Microsoft binary:

sideload malicious modules for initial access and execution.

DLL from Archive Files

Similarly to virtual disk images, attackers can also use ZIP/RAR archive files with embedded malicious DLL paired with a trusted binary or a shortcut (LNK) file to gain access.

The following screen capture shows how this query identifies a malicious file from a RAR archive which was auto-extracted into a temporary user directory. This scenario is moderately common.

DLL via Malicious Documents

Microsoft Office documents can be also used to deploy and load a malicious DLL to avoid spawning a suspicious child process. The following query correlates an executable (PE) file creation event with a DLL load event.

Below are some examples of malicious Word and Excel documents using this delivery technique.

DLL via MSIEXEC

MsiExec is another great option when you need to execute malicious DLLs because this activity blends in well with legitimate software installers. Two observed delivery methods are:

- Calling the DLLRegisterServer export from a random DLL using the command-line arguments /y or /z as documented here

The following query can be used to identify the execution of the built-in Windows Installer, MSIEXEC, to call the exported function and run code:

Examples where MSI is used to load malicious DLLs:

DLLs delivered via Windows Installer custom actions can be detected by correlating a DLL file creation event where the calling process is MsiExec and where that DLL is subsequently loaded by the same MsiExec process.

It's worth noting that there are some legitimate uses of Windows Installer custom actions and this query may require some filtering in environments where those are used.

The following query matches the Gwisin Ransomware documented by **AhnLab** and for which a **PoC** has been created.

DLL delivery via lolbins

Some malware relies on trusted Microsoft binaries to download, decode or extract DLLs. This query correlates PE file creation or modification by common built-in tools, followed by an image load.

Examples of malware identified using this detection approach:

DLL sideload into existing program

The following detection identifies attempts to load a recently-created and unsigned DLL file by an already existing signed process within the same current directory. Comparing the difference between the creation time of the existing program and the DLL creation time we can spot these kinds of anomalies.

the current OneDrivedirectory and automatically loaded by OneDrive.exe (created 2.5 years ago):

DLL loading from suspicious directories

Dropping a DLL to a user-writable directories and side loading that with a trusted binary is also a common pattern. The following query looks for this behavior and, by leveraging relative creation and modification times, it can reduce the alerts volume while limiting those to a time window following initial execution.

The most commonly-targeted user-writable directories are `?:\Users\Public` and `?:\ProgramData`. The full query containing more than 70 suspicious folders can be found **here**.

Below see a example depicting malicious matches where various trusted binaries were abused to load malicious DLLs:

DLL load with a abnormal creation time

Another interesting scenario is identifying a DLL load event where the DLL has a suspicious creation time, and which could be a result of timestomping. This query compares inconsistencies between the creation time and filename modification time using dll.Ext.relative_file_name_modify_time and dll.Ext.relative_file_creation_time immediately followed by an image load:

The following is an example where malware drop DLLs in trusted directories and then use timestomping to ensure those DLLs blend in with existing files in those directories:

DLL from removable device

networks. An example was recently shared by Mandiant involving an espionage-oriented threat. The following EQL query can be used to find similar behavior:

Here is an example with several matches:

Protection Rules

Elastic provides significant capabilities for identifying unusual or malicious library load events with existing behavior protection rules that take advantage of Windows Libraries events:

- NTDLL Loaded from an Unusual Path
- Suspicious NTDLL Image Load
- DLL Loaded from an Archive File
- Microsoft Office Loaded a Dropped Executable File
- Suspicious ImageLoad from an ISO Mounted Device
- Potential Evasion via Oversized Image Load
- Suspicious ImageLoad via Windows Update Auto Update Client
- Privilege Escalation via Microsoft Exchange DLL Hijacking
- Potential DLL SideLoad via a Microsoft Signed Binary
- Potential DLL SideLoad via a Renamed Signed Binary
- Library Load of a File Written by a Signed Binary Proxy
- Potential DLL Search Order Hijacking of an Existing Program
- Suspicious DLLRegisterServer Execution via MSIEXEC
- ImageLoad of a File dropped via SMB
- RunDLL32/Regsvr32 Loads Dropped Executable

- RunDLL32/Regsvr32 Loads a DLL Downloaded via BITS
- Potential Initial Access via DLL Search Order Hijacking
- Suspicious Control Panel DLL Loaded by Explorer
- Protected Process Light Bypass via DLL Tampering
- Potential Privilege Escalation via DLL Redirection
- Potential Privilege Escalation via Missing DLL
- Potential Privilege Escalation via Elevated IFileOperation
- Suspicious DLL Loaded by Svchost
- Suspicious DLL Loaded from a Removable Media
- Suspicious Control Panel DLL Loaded by Explorer
- Dynwrapx Image Load via Windows Scripts
- Suspicious Image Load via Windows Scripts
- Potential Image Load with a Spoofed Creation Time

Conclusion

Compared to detections that rely on process execution events and where adversaries expose more detection opportunities via command-line flags and parent process relationships, designing detections based on DLL events requires more enrichment and correlation to decrease noise rate and increase confidence.

In this publication we shared numerous examples of how we're using DLL events to identify threats. You can use the different capabilities Elastic endpoint offers to produce higher signal alerts, too. Given the multitude of methods of delivering malicious code as DLLs, though, relying on behavioral detections alone is not enough. Combining this logic with malware file classification, shellcode detection features, and user-entity based analytics (UEBA) improves the fidelity of this metadata for detection purposes.