Hunting Unsigned DLLs Using KQL

Akhere Sonny-Egbeahie · Dec 17, 2022 · 3 min read

Unsigned DLLs, or dynamic-link libraries, are executable files that contain code that can be used by multiple programs on a Windows system. While DLLs can be a useful and efficient way to share code between applications, they can also be a security risk if they are unsigned or have been modified by an attacker.

One common attack trend involving unsigned DLLs is the use of DLL injection. In this type of attack, an attacker injects malicious code into a legitimate DLL file, and

then uses a tool such as rund[illegible] to execute the code. This can allow the attacker to exe[illegible] victim's system and potentially gain unauthorized access to sensitive data or systems.

To mitigate the risk of unsigned DLL attacks, it is important to ensure that all DLLs on your system are properly signed and have not been tampered with. One way to do this is by using a KQL (Kusto Query Language) query to hunt for unsigned DLLs on your system.

To hunt for unsigned DLLs using KQL, we can use a query like the following:

COPY

```
// Find unsigned DLLs in the System32 folder
let signedDLLs = Folder('C:\Windows\System32')
| where IsFile
| where Extension == '.dll'
| extend fileinfo = GetFileInfo()
| where fileinfo.SignatureStatus == 'Unsigned';

signedDLLs
```

This query uses the `Folder` operator to search the `C:\Windows\System32` folder for files with the `.dll` extension. It then uses the `GetFileInfo` operator to retrieve information about each DLL, and filters the results to include only those DLLs that have an `Unsigned` signature status.

We can modify this query to search for unsigned DLLs in a different folder, or to include additional filtering criteria, such as a specific file name or date range. For

example, we can search for u... rundll32.exe or regsvr32.exe from uncommon folders like

```
// Find unsigned DLLs loaded by rundll32.exe or regsvr32.exe from uncommon fo
let commonFolders = ['C:\Windows\System32', 'C:\Windows\SysWOW64'];

SecurityEvent
| where EventID == '1'
| where ProcessName in ('rundll32.exe', 'regsvr32.exe')
| extend Arguments = extractall(@"(?<=file:)[^ ]*", CommandLine)
| where Arguments !in (commonFolders)
| extend fileinfo = GetFileInfo(Arguments)
| where fileinfo.SignatureStatus == 'Unsigned'
| project TimeGenerated, Computer, UserName, ProcessName, Arguments, fileinfo
```

This query uses the `SecurityEvent` table to filter for events with the `EventID` of `1`, which corresponds to process creation events. It then filters the results to include only processes with the name `rundll32.exe` or `regsvr32.exe`.

Next, the `extractall` function is used to extract the file path of the DLL being loaded from the `CommandLine` field. The `where` clause is then used to filter out DLLs that are located in common folders, such as `C:\Windows\System32` and `C:\Windows\SysWOW64`.

Finally, the `GetFileInfo` operator is used to retrieve information about each DLL, and the results are filtered to include only DLLs with an `Unsigned` signature status. The resulting events are then projected to include the time the event was generated,

the computer name, the user [illegible] the file path of the DLL, and the signature status.

Conclusion

In conclusion, unsigned DLLs can be a potent tool for attackers looking to compromise a system. By using KQL, we can create queries that help us to identify and track these DLLs on a Windows system. By regularly running these queries and monitoring the results, we can stay vigilant against attacks involving unsigned DLLs and take steps to protect our systems.

I hope you found this helpful. Thank you and see you on the next one.

KQL Security blueteam #cybersecurity cyber attack

Written by

Akhere Sonny-Egbeahie

I am a young and grow[illegible] interested in Blue teaming, Threat Intelligence and Cloud Security.