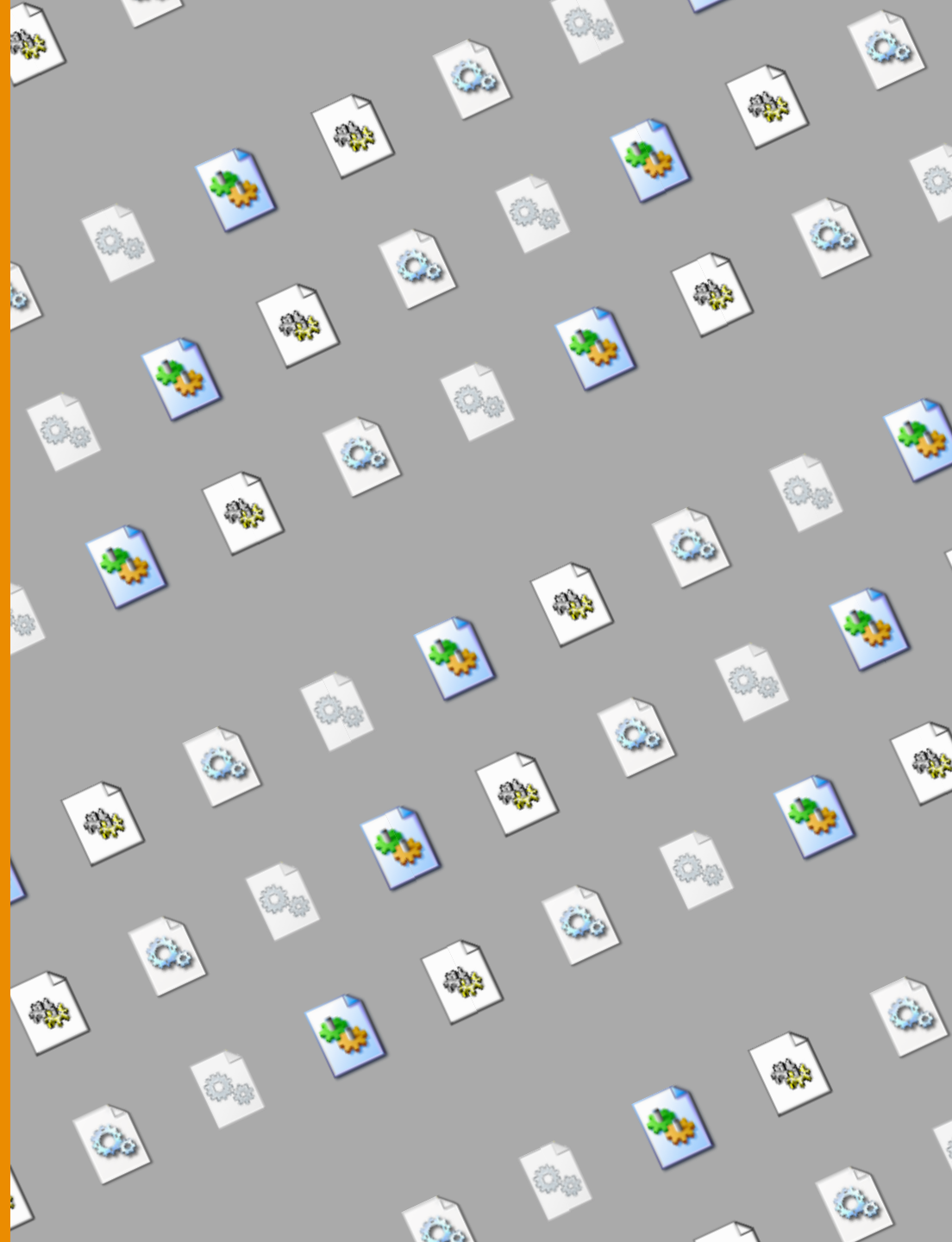
Hijacking DLLs in Windows

Wietze Beukema
October 2020



Hello, who dis?

Wietze Beukema

- Threat Hunting
- Detection Engineering & Research
- PwC UK, London

Keep in touch:

@Wietze

github.com/wietze

wietze.beukema@pwc.com

DLL Hijacking

DLL Hijacking

“Tricking a legitimate/trusted application into loading an arbitrary DLL”

What's in a name?

DLL Injection
DLL Load Order Hijacking
DLL Load Order Hijacking
DLL Search Order Hijacking
DLL Spoofing
DLL Spoofing
DLL Side-Loading
DLL Spoofing

DLL Load Order Hijacking
DLL Search Order Hijacking
DLL Hijacking
DLL Injection
DLL Side-Loading
DLL Injection

DLL Imitation
DLL Search Order Hijacking
DLL Imitation
DLL Side-Loading

DLL Hijacking (2)

- **T1574**: Hijack Execution Flow
  - ***T1574.001***: *DLL Search Order Hijacking*
  - ***T1574.002***: *DLL Side-Loading*
- **T1218**: Signed Binary Proxy Execution (?)
- **T1036**: Masquerading

DLL Hijacking (3)

But why?

- Your evil code, executed by a trusted executable
- Great for Execution, Persistence, Privilege Escalation
- The threat is real

Types
A few common ones

Type	Method
1: **DLL replacement**	Replace a legitimate DLL with the evil DLL.
2: **DLL search order hijacking**	Place the evil DLL in a location that is ched for before the legitimate DLL.
3: **Phantom DLL Hijacking**	Drop the evil DLL in place of a missing/existing DLL that a legitimate application tries to load.
4: **DLL redirection**	Change the location in which the DLL is searched for, by editing the %PATH% environment variable or .exe.manifest / .exe.local.
5: **WinSxS DLL replacement** (*'DLL side-loading'*)	Replace the legitimate with the evil DLL in the relevant WinSxS folder of the targeted DLL.
6: **Relative path DLL Hijacking**	Copy (and optionally rename) the legitimate application to a user-writeable folder, alongside the evil DLL.

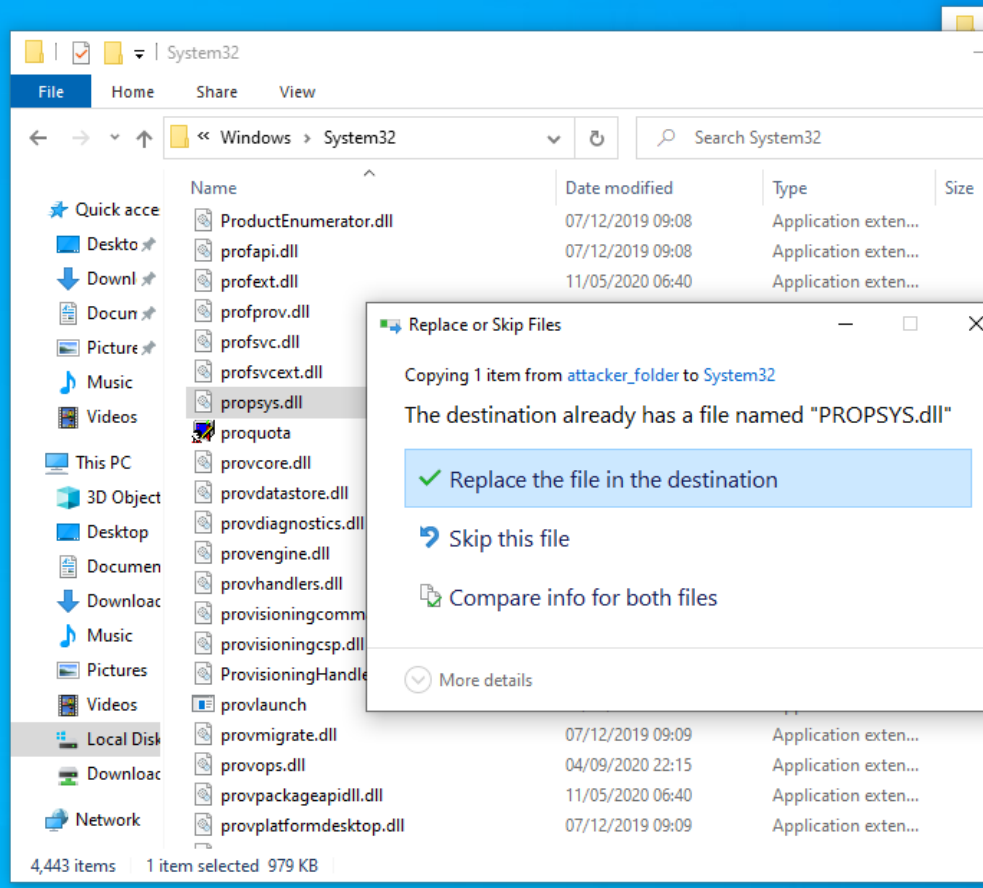
Types
1: DLL replacement

> *"Replace a legitimate DLL with the evil DLL."*

- Most basic type
- Could run DLL with high integrity (although replacement usually requires this too)
- Often used together with DLL Proxying [1].

Example

Stuxnet: renamed legitimate Siemens DLL `S7OTBXDX.DLL` in `c:\windows\` and added its own malicious version, implementing all exports but with modified logic when interacting with the PLC [2]. *Execution + Persistence*!



Types

2: DLL search order hijacking (T1574.001)

> *"Place the evil DLL in a location that is searched for before the legitimate DLL."*

- Look at the *DLL search order*: (1) Directory of application → (2) System32 directory → (3) System directory → (4) Windows directory → (5) Current directory → (6) `%PATH%`
- CVE potential: especially executables running with high integrity
- Good way to find these: look for `FILE NOT FOUND` in Procmon

Example

Vault7 Leaks: portable Chrome tries to load `DWrite.dll`, located in `System32\`. Because no absolute path is specified, the directory of the application is tried first [3]. *Execution* (+ *Persistence*)!

Types
3: Phantom DLL Hijacking

> *"Drop the evil DLL in place of a missing/non-existing DLL that a legitimate application tries to load."*

- More rare, yet not uncommon
- Less likely your DLL will break the running application
- CVE potential

Example

Vault7 leaks: Kaspersky process `avp.exe` intended to load `WHEAPGRD.DLL`. Due to a bug, it would prepend the drive letter, meaning it would try to load non-existing file `CWHEAPGRD.DLL` [4]. *Execution* + *Persistence* (+ *Elevation?*)

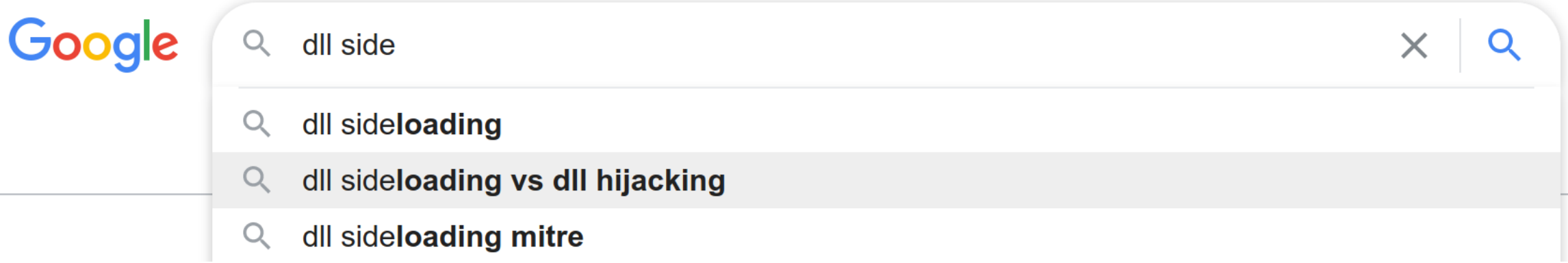
Types
4: DLL redirection

“Change the location in which the DLL is searched for, by editing %PATH% or .exe.manifest / .exe.local.”

Types
5: WinSxS DLL replacement (T1574.002)

"Replace the legitimate with the evil DLL in the relevant WinSxS folder of the targeted DLL."

- Sometimes referred to as 'DLL Side-Loading' [5]



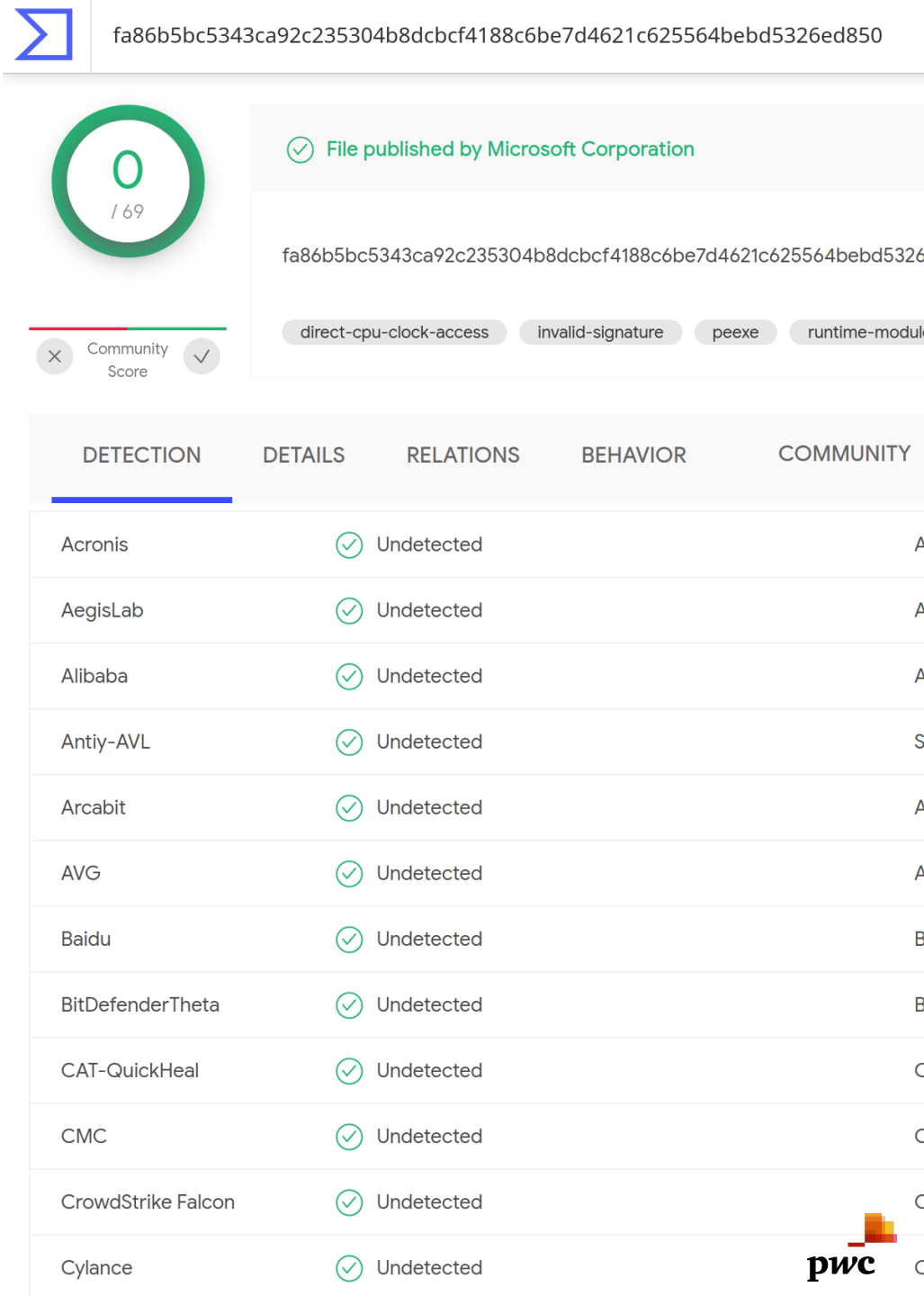
Types
6: Relative path DLL Hijacking

> *"Copy (and optionally rename) the legitimate application to a user-writeable folder, alongside the evil DLL."*

- Simple: DLL Hijacking with least preconditions
- No CVE/LOLbin potential, yet very powerful
- Combine with other techniques for more impact, e.g. UAC bypass

Example

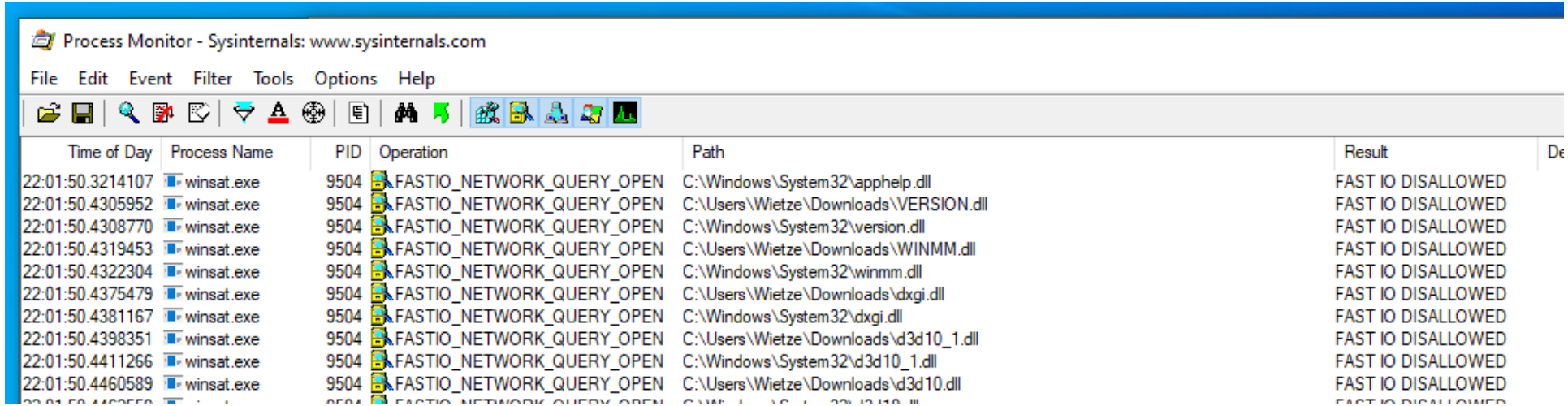
PwC Threat Intelligence observed a dropper ('xStart') used by a threat actor targetting Chinese organisations to deliver Cobalt Strike payloads. Through phishing, a user is tricked into opening a legitimate (but renamed) `winword.exe` that is put alongside a malicious alongside a malicious `wwlib.dll`. When the renamed `winword.exe` is executed, `wwlib.dll` is triggered, starting phase 2 of the attack [6]. *Execution*!



Types
6: Relative path DLL Hijacking (2)

"Copy (and optionally rename) the legitimate application to a user-writeable folder, alongside the evil DLL."

- Nearly 300 (!) `system32\` executables are vulnerable to this [7]
- Many, many more outside this dir, as well as non-Microsoft (see Hexacorn et al.)
- Can be combined with UAC bypass techniques



Time of Day	Process Name	PID	Operation	Path	Result
22:01:50.3214107	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Windows\System32\apphelp.dll	FAST IO DISALLOWED
22:01:50.4305952	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Users\Wietze\Downloads\VERSION.dll	FAST IO DISALLOWED
22:01:50.4308770	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Windows\System32\version.dll	FAST IO DISALLOWED
22:01:50.4319453	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Users\Wietze\Downloads\WINMM.dll	FAST IO DISALLOWED
22:01:50.4322304	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Windows\System32\winmm.dll	FAST IO DISALLOWED
22:01:50.4375479	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Users\Wietze\Downloads\dxgi.dll	FAST IO DISALLOWED
22:01:50.4381167	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Windows\System32\dxgi.dll	FAST IO DISALLOWED
22:01:50.4398351	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Users\Wietze\Downloads\d3d10_1.dll	FAST IO DISALLOWED
22:01:50.4411266	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Windows\System32\d3d10_1.dll	FAST IO DISALLOWED
22:01:50.4460589	winsat.exe	9504	FASTIO_NETWORK_QUERY_OPEN	C:\Users\Wietze\Downloads\d3d10.dll	FAST IO DISALLOWED

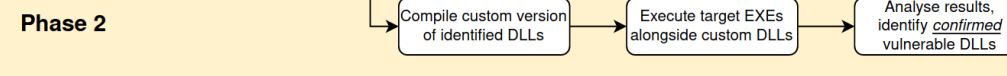
Types
6: Relative path DLL Hijacking (3)

> *"Copy (and optionally rename) the legitimate application to a user-writeable folder, alongside the evil DLL."*

This poses a problem:

- Relies on legitimate executables
- Many possible candidates
- Actively used in the wild
- Impact is medium to high

So how to prevent / detect this?



The full code with a more thorough, technical explanation can be found on GitHub [14]

Confirmed DLL Hijack candidates

The following table lists all executables in `c:\windows\system32` on Windows 10 v1909 are vulnerable to the 'relative path DLL Hijack' variant of DLL Hijacking. Next to executable is one or more DLLs that can be hijacked, together with the procedures o DLL that are called. As explained in the previous section, these are not mere theor targets, **these are tested and confirmed to be working**. The list comprises 287 execut and 263 unique DLLs.

Showing **1,566** entries Search:

Auto-elevated ▲	Executable	DLL	Procedure
✓	bthudtask.exe	DEVOBJ.dll	DllMain
✓	computerdefaults.exe	CRYPTBASE.DLL	DllMain
✓		edputil.dll	DllMain
✓			EdpGetIsManaged
✓		MLANG.dll	ConvertINetUnicodeToMultiBy e
✓			DllMain
✓		PROPSYS.dll	DllMain
✓			PSCreateMemoryPropertyStore
✓			PSPropertyBag_WriteDWORD
✓		Secur32.dll	DllMain
✓		SSPICLI.DLL	DllMain
✓			GetUserNameExW
✓		WININET.dll	DllMain
✓			GetUrlCacheEntryBinaryBlob
✓		ColorAdapterClient.dll	DllMain

Some caveats:

- The test was performed by simply running each executable, without specifying a

☐ Prevention & Detection

Preventing DLL Hijacking

🔧 **Developers**	👩‍💻 👨‍💻 **System admins**
Use absolute DLL paths where possible *e.g.* `system32\` *DLLs, environment variables if needed*	Use `PreferSystem32Images` and `MicrosoftSignedOnly` /`StoreSignedOnly` process mitigations? [8]
Verify validity of DLL *i.e. check expected signature*	

Nevertheless: the problem remains!

Detecting DLL Hijacking

It’s not straightforward…

A few (flawed) ideas:

- Look for known DLL hijack targets (DLL names, executables) - see [9]
- Look for creation of DLLs by unexpected processes
- Look for common targets (e.g. Microsoft-signed executables) in unexpected locations
- Look for common targets loading DLLs not on VT
- …

290 lines (290 sloc) | 8.65 KB

```yaml
title: Possible Windows DLL Hijacking
id: 980b0288-2174-418f-b64b-c9c8fb9a74fb
status: experimental
description: Detects possible DLL hijacking of vulnerable Windows DL
references:
    - https://wietze.github.io/blog/hijacking-dlls-in-windows
author: "@Wietze"
date: 2020/06/22
tags:
    - attack.defense_evasion
    - attack.T1038
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 7
        ImageLoaded:
            - "*\\activeds.dll"
            - "*\\adsldpc.dll"
            - "*\\aepic.dll"
            - "*\\apphelp.dll"
            - "*\\appxalluserstore.dll"
            - "*\\appxdeploymentclient.dll"
            - "*\\archiveint.dll"
            - "*\\atl.dll"
            - "*\\audioses.dll"
            - "*\\auditpolcore.dll"
            - "*\\authfwcfg.dll"
            - "*\\authz.dll"
            - "*\\avrt.dll"
            - "*\\bcd.dll"
            - "*\\bcp47langs.dll"
            - "*\\bcp47mrm.dll"
            - "*\\bderepair.dll"
            - "*\\bootmenuux.dll"
            - "*\\cabinet.dll"
```

Detecting DLL Hijacking (2)

… but not impossible

- 🔎 Instead of just looking for the DLL Hijacking, look for the **behaviour that follows**
- 📚 **Layer your defences**!
- 🛡️ Ensure a **broad base** of behavioural rules, hunting techniques, anomaly detection
  - *You're never going to detect everything*
  - *The more you monitor, the less likely an intrusion will go undetected*

Thank you