The Goot cause: Detecting Gootloader and its follow-on activity

By Anna Seitz

Published: 2022-05-12 · Archived: 2026-04-10 03:03:28 UTC

The Goot cause: Detecting Gootloader and its follow-on activity

Gootloader is a pervasive and enduring threat affecting enterprise organizations. Read on for context on recent iterations of this threat and high-fidelity opportunities to detect known behaviors.

Originally published May 12, 2022. Last modified May 8, 2025.

EDITOR'S NOTE: On November 18, 2022, we made substantial updates to the Execution section of this article (and minor changes to the Persistence and Detection sections) to reflect operational changes we've observed in Gootloader campaigns in recent days and weeks. We used a strikethrough to call out the no-longer relevant TTPs in the Execution section and noted where the net new content starts thereafter. New detection opportunities are noted as such.

Over the past several years, Red Canary has routinely detected activity involving a threat known as Gootloader: malware that can deliver additional payloads, siphon data from victims, and stealthily persist in a compromised environment. Gootloader was originally delivered via spam campaigns and older exploit kits. We've increasingly observed Gootloader operators using search engine optimization (SEO) poisoning tactics to gain access to victims' environments and initiate multi-pronged intrusions involving follow-on payloads such as Cobalt Strike and Gootkit.

Gootloader specifically represents a significant threat to enterprise environments because it is designed to deliver additional malware. We included it as a highlighted threat in our 2022 Threat Detection Report, and it boasts a regular presence in our monthly top 10 rankings for Intelligence Insights. Despite the high volume of Gootloader infections, there is relatively little reporting available on complete intrusion chains.

Distinguishing Gootloader from Gootkit

We historically referred to both Gootloader and Gootkit under the same name of "Gootkit," but after realizing others in the community tracked these as different threats, we decided to do the same. Separating the initial delivery and loader distinctly from the payload also allows us to better track variations in follow-on activity. We decided to impose an analytic boundary (see below for the full intrusion chain) between Gootloader and Gootkit after the execution of the .NET DLL module commonly observed in intrusion chains involving these threats. This division is consistent with parameters outlined by researchers from Kaspersky and allows us to account for variations in activity observed after Gootloader more precisely. Though Gootkit may be a follow-on payload, we've also observed other activity following Gootloader infections, including Cobalt Strike beacons and the Osiris banking trojan.

Below you'll find our analysis on Gootloader, the most prolific piece of the intrusion chain, and its capabilities. We're also including a graphic to highlight the boundary between Gootloader and its following payloads, along with supplemental malware analysis.

Initial access

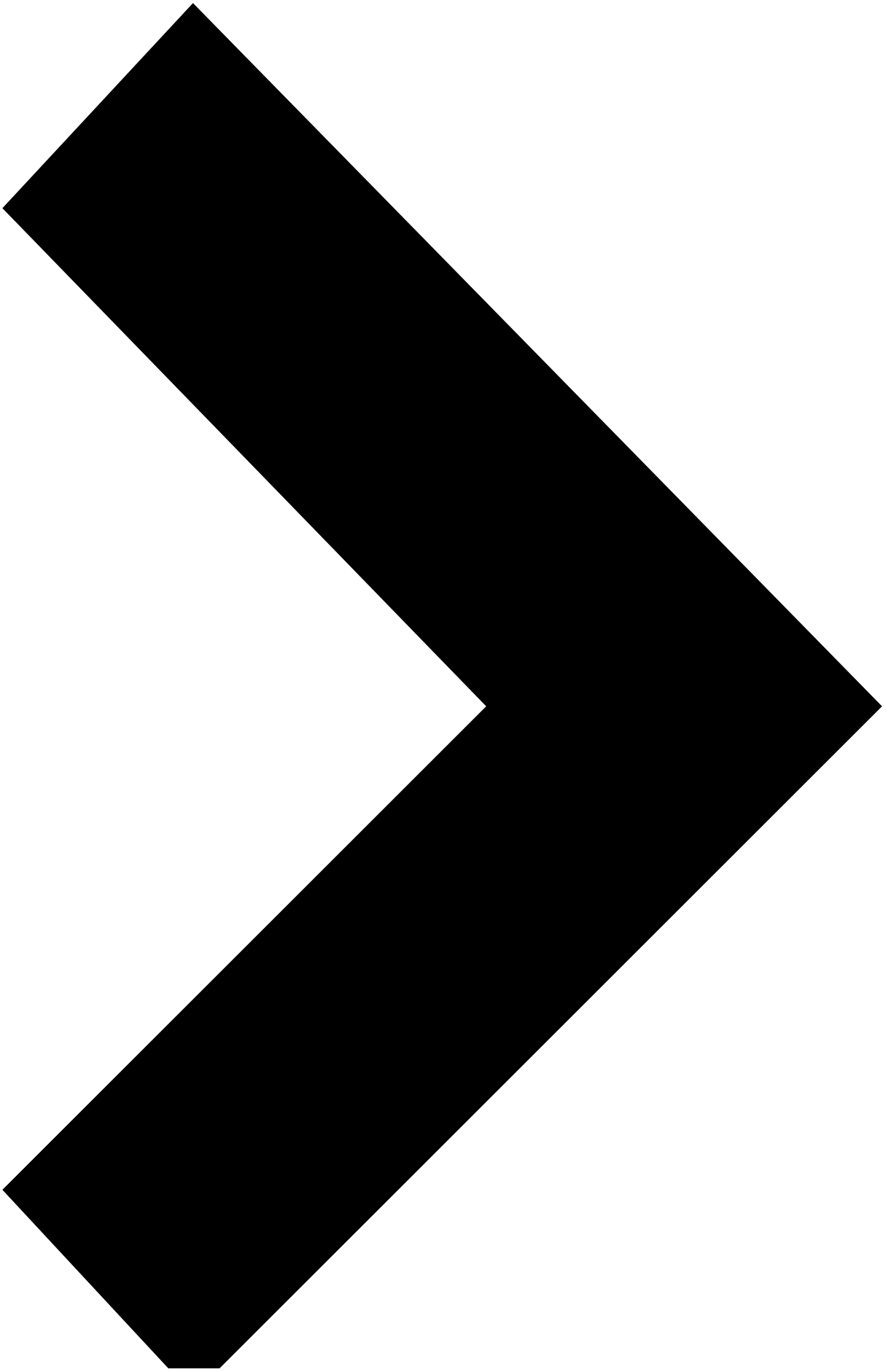
Gootloader operators compromise legitimate infrastructure, such as WordPress blogs, and seed those sites with common keywords. Operators then use SEO techniques in an attempt to direct anyone entering those keywords into a search engine to a site that lures users to download a ZIP file containing the initial Gootloader script. Previous work from Sophos and other organizations covers this aspect of Gootloader in great detail. From our own investigations, we've discovered trends relating to possible search terms that lead users to Gootloader. It's not clear if operators are specifically targeting individuals in specific functions across specific organizations, or if they cast a more opportunistic net, using common terms that potential victims may be more likely to seek out on their own.

The majority of the Gootloader campaigns we've observed involved initial malicious ZIP files containing the word "agreement" in the file name. We can identify victims' search queries based on the name of the malicious ZIP file that contains the Gootloader script. The malicious ZIP name is usually the user's search query terms joined by dashes. For example, if a user searched for "mortgage subordination agreement," the downloaded ZIP would be named `mortgage_subordination_agreement.zip`.

Figure 1: Gootloader intrusion chain

Executive Summary: 2024 Threat Detection Report

Learn more



Execution

The first stage of Gootloader on the endpoint is a JScript file extracted from a ZIP file and executed via `wscript.exe`. ~~While these JScript files have been a common Gootloader entry point since December 2020, the scripts changed around October 2021 to masquerade as legitimate jQuery JavaScript library files. To achieve this masquerading, the adversary creates scripts by mixing malicious Gootloader code with benign jQuery library code, producing a file around 296KB in size.~~ [*Note: the following two sentences were added on November 18, 2020 to reflect recent operational changes we've observed in recent Gootloader campaigns*] While these JavaScript files have been a common entry point since December 2020, the adversary has changed them slightly to masquerade as legitimate code. First we observed malicious Gootloader code spliced together with code from the jQuery JavaScript library to inflate file size and hinder analysis. Starting in November 2022, we began observing Gootloader code getting spliced together with code from another JavaScript library named Underscore.JS. In both iterations, the malicious code and harmless library code were combined together into the malicious "agreement" files that executed on victim systems.

Gootloader queries the value of the `USERDNSDOMAIN` environment variable, which is a simple check to determine if the affected host is part of an Active Directory domain. This means that the malware specifically targets business or enterprise victims that use Active Directory. ~~On systems where the check passes, Gootloader pulls down an additional JScript stage that executes in the same `wscript.exe` process. That stage contains two embedded payloads: a .NET DLL component and a Cobalt Strike beacon or other malware component. During execution, these two payloads are written into Windows Registry keys to enable persistence.~~

NOTE: We added the following behavioral descriptions and images to the Execution section of this article on November 18, 2022 to reflect recent operational changes we've observed in recent Gootloader campaigns.

Lately we've observed Gootloader writing a randomly named JavaScript file (initially with the extension `.log`) to `appdata\roaming` using variations of random naming conventions. Interestingly, part of the file path almost always seems to include the name of a legitimate software or security product. We've changed most of the filenames included here to protect the innocent, but we found numerous examples—like the following—in a public malware sandbox.

Log

```
C:\Users\Admin\AppData\Roaming\Sun\Financial Support.log
C:\Users\Admin\AppData\Roaming\Sun\Virtual Currency.log
```

JS

```
Sun\Broker Price Opinion.js
Sun\Progress Billing.js
```

`wscript.exe` executes the initial JavaScript dropper (`how do I withdraw funds (epl).js`), unpacks a second .js file (`choice showing.js`), and then creates a scheduled task. This initiates a chain of execution where `cscript.exe` executes the second .js file along with an instance of PowerShell without any command line, which, in turn, passes an encoded command into a second PowerShell instance. The scheduled task is a persistence mechanism intended to run these commands again the next time the user logs in (more on this in the Persistence section).

It's worth noting that .js file referenced in the `cscript.exe` command line (as shown in the images included in this article) does not match the actual file name (i.e., `CHOICE~1.JS` != `Choice Showing.js`). However, they are in fact the same file, but one is leveraging a Windows shortname).

Persistence

The first PowerShell command referenced above retrieves the .NET DLL from the Windows Registry, reflectively loads it, and executes a function within the DLL named `Test()` .

```
sleep -s 83;$opj=Get-ItemProperty -path ("hk"+"cu:\sof"+"tw"+"are\mic"+"ros"+"oft\Phone\"+[Environment]::("use
```

~~The second PowerShell command establishes persistence via a scheduled task using a combination of cmdlets. The execution of the .NET DLL module is one of the main differentiators between traditional Gootkit and the~~

~~initial Gootloader.~~ [*Note: the following sentences were added on November 18, 2022*]. The initial instance of `wscript.exe` establishes persistence by creating a scheduled task to execute when the affected user logs in. It picks up execution at the `cscript.exe` stage.

```
$a="[Base64 code]...";$u=$env:USERNAME;Register-ScheduledTask $u -In (New-ScheduledTask -Ac (New-ScheduledTask/
```

Follow-on payload

In the .NET DLL module, the adversary implements code to pull an obfuscated payload (such as Cobalt Strike) from a Windows Registry key, remove the obfuscation, and then execute its contents. The decoding part is fairly straightforward, using text replacement to shield the malware from cursory inspection. Follow-on payloads vary and have included Cobalt Strike, Gootkit, and Osiris. In the event Cobalt Strike is the follow-on payload, see our malware analysis for more details.

Red Canary recommends detecting Gootloader activity to catch this threat early in the intrusion chain. One potential detection idea is to look for the execution of PowerShell with the encoded command switch (`-enc`), which you can find here. See below for additional opportunities to identify Gootloader or follow-on activity in your environment.

New detection opportunity: `wscript.exe` spawning `cscript.exe` and PowerShell

This detection opportunity identifies the chain of process executions—whereby `wscript.exe` spawns `cscript.exe` and `cscript.exe` spawns `powershell.exe` —described in the Execution section that we updated on November 18, 2022.

parent_process == (`wscript.exe`)

&&

process == (`cscript.exe`)

&&

child_process == (`powershell.exe`)

Detection opportunity: Windows Script Host (`wscript.exe`) executing content from a user's AppData folder

This detection opportunity identifies the Windows Script Host, `wscript.exe` , executing a JScript file from the user's AppData folder. This works well to detect instances where a user has double-clicked into a Gootloader ZIP file and then double-clicked on the JScript script to execute it.

process == (`wscript.exe`)
&&

process_command_line_includes == `appdata\*.js`

Detection opportunity: PowerShell (`powershell.exe`) performing a reflective load of a .NET assembly

This detection opportunity identifies PowerShell loading a .NET assembly into memory for execution using the `System.Reflection` capabilities of the .NET Framework. This detects PowerShell loading the .NET component of Gootloader and multiple additional threats in the wild.

process == (`powershell.exe`)
&&
process_command_line_includes == `Reflection.Assembly` AND `Load` AND `byte[]`

Detection opportunity: Rundll32 (`rundll32.exe`) with no command-line arguments

This detection opportunity identifies `rundll32.exe` executing with no command-line arguments as an injection target like we usually see for Cobalt Strike beacon injection. The beacon distributed by Gootloader in this instance used `rundll32.exe` , as do many other beacons found in the wild.

process == `rundll32.exe`
&&
command_line_includes (“”)*
&&
has_network_connection
||
has_child_process

**Note: “” indicates a blank command line.*

Mitigation advice

You can prevent Gootloader from executing by changing the default file association for JScript files on your Windows systems. Consider using a Group Policy Object to associate JScript files with `notepad.exe` instead of `wscript.exe` . This will make the malicious scripts open in Notepad when a victim double-clicks on the file. For successful execution, the victim would have to manually issue a `wscript.exe` command instead.

Additionally, a little bit of education can help mitigate Gootloader. In all of the instances we observed, victims were seeking legal agreement documents via Google searches. Documenting safe places to obtain legal documents can help prevent users from downloading potentially malicious files.

You'll receive a weekly email with our new blog posts.
