HMAC and "Secure Preferences": Revisiting Chromium-based Browsers Security*

Pablo Picazo-Sanchez, Gerardo Schneider, and Andrei Sabelfeld

Chalmers University of Technology
Gothenburg, Sweden,

Abstract. Google disabled years ago the possibility to freely modify some internal configuration parameters, so options like silently (un)install browser extensions, changing the home page or the search engine were banned. This capability was as simple as adding/removing some lines from a plain text file called Secure Preferences file automatically created by Chromium the first time it is launched. Concretely, Google introduced a security mechanism based on a cryptographic algorithm named Hash-based Message Authentication Code (HMAC) to avoid users and applications other than the browser modifying the Secure Preferences file. This paper demonstrates that it is possible to perform browser hijacking, browser extension fingerprinting and remote code execution attacks as well as silent browser extensions (un)installation by coding a platform-independent proof-of-concept changeware that exploits the HMAC, making possible the free modification of the Secure Preferences file. Last but no least, we analyze the security of the three most important Chromium-based browsers: Brave, Chrome, Microsoft Edge and Opera, concluding that all of them suffer from the same security pitfall.

Keywords: HMAC · Changeware · Chromium · Web Security

1 Introduction

Chrome is as of today the most used web browser in the world [29]. Chrome, as well as many other browser vendors like Opera, Brave and Vivaldi are based on Chromium, an open-sourced web browser developed by Google. Recently, Microsoft moved to adopt Chromium as the basis for the new Microsoft Edge browser [21]. Given its widespread use, around 75% of the desktop users on Internet [27], the security of Chromium is paramount.

In order to allow the web browser for an easy customization, there exist many configuration parameters that might be modified to fit the needs of the users. Setting the homepage to a custom webpage that a user frequently visits, changing the default search engine, "pinning" some URLs to tabs and browser extensions management are just a few examples of the huge list of actions that can be performed to make the user experience more pleasant. One of the most

* Version modified for responsible disclosure

promising tools for enriching the browser experience of the user is *browser extensions*. Extensions are installed from the Chrome Web Store, which is a central repository managed by Google.

As recently claimed [14], approximately 10% of the browser extensions stored between 2012 and 2015 in the Web Store were classified as malware and deleted from the repository. Despite many attempts done to improve the security and privacy of extensions [14, 15, 24, 26], vulnerabilities still abound [2, 3, 25], being Potentially Unwanted Programs (PUPs) one popular and challenging example because they are not usually marked as malware by antivirus vendors [16, 28].

PUPs are installation executable files that, apart from installing the application the user wants, they also execute other software that might not be related to the legitimate one. *Adware* and *changeware* are two types of PUPs that adds advertisement to the webpages the user visits and changes the configuration properties of the browser silently, respectively. Recently, a cybersecurity firm discussed the thin line between espionage-level malware and PUPs and detected more than 111 browser extensions considered to be PUP whose goal was to spy users [4]. In this paper, we consider PUPs and pay special attention to how changeware works, providing a concrete example of how the installation of uTorrent application modifies the configuration of the browser (see **??**).

In the particular case of Chromium-based browsers, each user obtains a couple of configuration files for storing information such as bookmarks, history, homepage and some other preferences. One of these files is the *Secure Preference file* which is automatically loaded when the browser is launched and it is updated each time the browser is closed. In 2012 Google improved the security of browser to protect users from silently installing extensions since these were causing more and more problems. Before that, it was possible to silently install extensions into Chrome by directly modifying the Secure Preferences file or by using the Windows registry mechanism. Extensions that were installed by third party programs through external extension deployment options were disabled by default and only extensions installed from Google Web Store are allowed.

Concretely, Chromium implemented from version 25 a secure mechanism to ensure that no external applications apart from the browser can modify the Secure Preferences file. This mechanism is a custom Hash-based Message Authentication Code (HMAC) algorithm [17] which produces a SHA-256 hash given both a seed and a message. However, as the original authors claimed, the security of HMAC relies on the seed generation, thus being secure as long as the seed is.

Our findings reveal that the seed needed to generate the HMAC, stored in a public file named `resources.pak`, is not randomly generated. Moreover, for each Chromium-based browser, the seed is the same for all the Operating Systems (OSs). Nevertheless, if the seed were randomly generated the problem of where to securely store either the seed or the key used to encrypt the seed, still persists. Because of that, some authors proposed to use WhiteBox-Cryptography [8] to secure this seed on Chromium [6]; however, this solution is platform dependent and only works under certain circumstances and on a concrete OS. As we show in this paper the problem remains unsolved. Once a malicious party gets such a

seed, it may impersonate the browser and modify any parameter of the Secure Preferences file.

To the best of our knowledge, the attack against the Secure Preferences file has never been published with the exception of a partial description in (at least) one Internet forum—whose moderator claimed that this attack no longer works [13]. To confirm this, we downloaded and installed multiple versions of Chromium in computers with Windows 10 and MacOS. We implemented the attack described in that forum and confirmed that it did stop working from Chromium versions up to 58.0.2999.0. In this paper we present a proof-of-concept PUP that modifies the Secure Preferences file of any Chromium version from 58.0.2999.0 until the latest one at the time of writing (85.0.4172.0). Additionally, if used together with the attack presented in that forum, any Chromium version can be easily editable (see Table 1).

Table 1. Chromium versions exploitable via HMAC.

Chromium Version	Released	SPF
(prior to) 25.0.1313.0	2012	Free modification
25.0.1313.0	2012	Attack [13]
58.0.2988.0	2017-01	Attack [13]
58.0.2999.0	2017-02	This paper
85.0.4172.0 (latest)	2020	This paper

This poses serious security and privacy issues. For instance, it is possible to perform browser hijacking attacks [22, 30], fingerprinting attacks [2, 18, 24], remote code execution [25], as well as silent browser extensions (un)installation (something Google has in principle banned years ago [9]). In many cases, the way of proceeding is the same: changing the browser search provider to generate advertising revenue by using well known search providers like Yahoo Search or Softonic Web Search among others [1, 20]; retrieving information about that uniquely identifies the user, and; exploiting other extensions to gain privileges or to remotely execute source code.

Contributions This paper analyzes how four of the most important Chromium-based browsers [11]—Chrome (70% of market share), Microsoft Edge (5% of market share), Opera (2.4%), and Brave[1]—manage the security and privacy of the users through a configuration file named Secure Preferences file. We discover that all of them use fixed seeds to generate the HMACs to secure the Secure Preferences file. These HMACs are used to guarantee that the content of the users' privacy settings have not been altered by any other party different than the browser (Section 3). We implement a changeware that performs a Chromium impersonation attack to (un)install extensions, perform phishing attacks, hijack

[1] Brave uses Chrome user-agent (desktop and Android) and Firefox user-agent (iOS).

user's browser, fingerprint users through the extensions the browser has as well as remote code execution from installed-by-default extensions among other things (**??**).

Section 2 presents the background about the Secure Preferences file and how Chromium uses it. **??** exposes some countermeasures to avoid the attack as well as brief discussion about how this vulnerability can be used by the research community for analyzing browser extensions. Finally, Section 4 presents the related work and Section 5 concludes the paper.

2 Chromium Preferences

In order to manage and enforce configurable settings, Chromium implements a mechanism called *preferences* to modify the settings of the browser per user instead of doing this centrally. Using preferences it is possible to configure, for instance, the homepage, which extensions are enabled/disabled and the default search engine.

To understand how Secure Preferences file works, we provide an example in what follows. Let Alice be a user who wants to manually modify any of the preferences stored in the Secure Preferences file. She accesses to her profile's folder, opens the JSON file—all the preferences are stored in plain text so anyone can access that file—and manually alters the preferences she would want to. Once she has modified them, then she saves the file and launches her Chromium instance to check whether the changes have been applied or not. The problem is that when Chromium loads, it automatically checks the integrity of the Secure Preferences file, warning Alice that the file has been externally modified and the browser marks the file as corrupted. Finally, Chromium automatically restores the Secure Preferences file to either a default or to a previous safe state.

Alice, who is an advanced user, tries to cheat Chromium by launching the web browser, and manually modifying the Secure Preferences file when the browser is running to check whether the changes can take effect. That will not work since Chromium loads the Secure Preferences file when it is launched the first time and overrides the whole Secure Preferences file when Alice closes the browser.

The reason to avoid external modifications to the Secure Preferences file is to improve on the privacy of the Chromium's users. In particular, what makes the Secure Preferences file secure is that Google added a Hash-based Message Authentication Code (HMAC) signature of every entry (settings/preference) in the file. In addition to this, the file also has a global-HMAC called *super_mac* to check the integrity of all the other HMACs.

HMAC [17] is a particular case of Message Authentication Code (MAC) which involves a hash function in combination with a shared secret key—also known as *seed* in these schemes. This algorithm was created in the 90's and has been usually used for both data verification and also for message authentication. As stated in the original proposal, the security of the HMAC protocols rely on the security of the underlying hash function, as well as both the size and quality of the seed.

Finally, if all the HMACs of the Secure Preferences file are correct, the browser will set up the settings according to what is stated in that file. In the case the validation procedure fails, the browser will use the default values for those ones where the HMAC validation failed. This recovery process is the same for all the Chromium-based browser but Brave. In this particular browser, instead of restoring the file to a previous state, it keeps a copy in the file system of the "corrupted" preferences file (using `.old` extension) and creates a new one. We included in Appendixes A.2, A.3 and A.1 the path where the Secure Preferences file are usually placed according to the platform and the browser.

3 Background: HMAC and Chromium

From version 25.0.1212.0 released in 2012, Google decided to not allow other parties different than the browser to modify the user's settings by including an HMAC per setting stored in the Secure Preferences file. When the user closes the browser, it computes the HMAC whereas when the user opens it, the browser recomputes all the HMACs and checks whether they were created by the browser. In particular, to modify the Secure Preferences file, the browser needs to: a) acquire the seed, and; b) obtain the message. Once the browser has these data, then it computes both the HMACs of the settings, and a final HMAC called *super_mac*.

3.1 Acquiring the seed

The seed is stored in the `resource.pak` file. See Appendix B.2 for more information about the paths of the browsers on different OSs. We explain in what follows how we got the seeds of the latest versions as of June 2020 of the three browsers being considered.

Chrome The seed that Chrome uses to compute the HMAC is a 64-long characters hexadecimal string that can be found in the `resource.pak` file. Additionally, we included the source code we run to retrieve the seed which can be seen in Figure 3 (Appendix C). Concretely, the first resource that has a length of 256 binary bits in the `resource.pak` file is the seed Chromium uses. Roughly speaking, the way we obtained this resource, is by loading the file and seeking for the first line (resource) with 64 characters.

OS	#PC	Same Seed
Linux	48	✓
Windows	44	✓
MacOS	8	✓

Table 2. Seed calculation on different OS

We executed the script on 100 different computers with different OSs (48 Linux, 44 Windows and 8 MacOS) and the results can be seen in Table 2. Concluding that the seed is not randomly computed as claimed. Concretely, the

seed is: `b'\xe7H\xf36\xd8^\xa5\xf9\xdc\xdf%\xd8\xf3G\xa6[L\xdffv\x00\xf0-\xf6rJ*\xf1\x8a!-&\xb7\x88\xa2P\x86\x91\x0c\xf3\xa9\x03\x13ihq\xf3\xdc\x05\x8270\xc9\x1d\xf8\xba\0\xd9\xc8\x84\xb5\x05\xa8'`. We run this experiment on Chrome version 85.0.4172.0.

Brave, Microsoft Edge and Opera We executed the same script to extract the seed on Brave, Edge and Opera but we could not change the user's settings. We had then to perform a brute force attack to extract the seed because the file was different than in Chrome. We got an alarming result concerning the seed used by these three vendors. We realized that the seed is the blank string, i.e., `seed = b''` in both Windows and MacOS. The version of Microsoft Edge we used was 85.0.564.51, for Brave we used version 1.14.81 (based on Chromium: 85.0.4183.102) whereas for Opera we used version 71.0.3770.148.

3.2 Obtaining the Message

In order to correctly generate the HMAC, a message should be passed as input. This message is composed of a *MachineIdStatus* and a string message. Such a variable is platform dependent, i.e., the *MachineIdStatus* is a different value in Windows, Linux and MacOS. That said, all of the three browsers have similar procedures to create the message used to generate the HMAC. In what follows we detail how the three different platforms obtain that *MachineIdStatus* value.

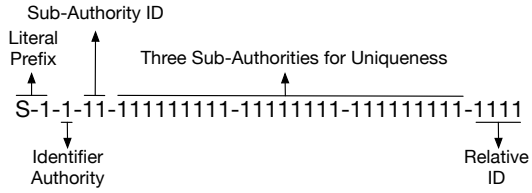

Fig. 1. Security IDentifier (SID)

Windows Users are provided with a unique identifier named SID. This identifier is usually used to control the access to resources like files, registry keys and network shares, among others. An example of the SID can be seen in Figure 1 and it might be easily retrieved by executing either the `wmic` or the `whoami` commands on Windows. After retrieving the SID, the last characters (Relative ID in Figure 1) are deleted for the final usage.

MacOS Instead of using the SID, MacOS uses the hardware Universally Unique IDentifier (UUID) which is a 128-bits number got by using the command `system_profiler SPHardwareDataType`. It outputs an hexadecimal number split in five groups by a "-", e.g., 1098AB78-6BF1-517E-905A-F018AABC4B26. In particular, in the `device_id_mac.cc` we can find how Chromium retrieves that UUID which is used afterwards as part of the message.

Linux Both Windows and MacOS have their own files under `chromium/src/services/preferences/tracked/` directory but there is no references about Linux. We corroborate that by checking the `device_is_unittest.cc` file where we found an if-then-else statement to differentiate how the SID should be computed depending whether the OS is either Windows or MacOS but there are no rules for Linux OS. Consequently, when the browser is running on Linux, the else statement is executed where there is a `MachineIdStatus::NOT_IMPLEMENTED;`. As a consequence, the *MachineIdStatus* variable has an empty string.

By analyzing how the message looks like in Chromium, we realized that it uses the key of the Secure Preferences file value it wants to modify together with either the SID or the UUID of the current user/computer in order to create such a message. More concretely, Chromium implements a function named `GetMessage` in the `pref_hash_calculator.cc` file, whose purpose is to concatenate three parameters given as inputs: *Device_ID*, *path* and *value*.

Device_ID corresponds to the *MachineIdStatus* variable, i.e., UUID on MacOS or the SID of the user without the relative ID information on Windows or the empty string on Linux. In other words, *Device_ID* is the identifier of the machine where Chromium is installed. Since every machine has its own unique SID no two HMACs will be the same when computed on different machines. However, on MacOS, since that the UUID is linked to the machine instead of being associated to the user, different profiles in the same machine will have the same UUID value.

Path is the path where the Secure Preferences file is in the computer (path of the JSON file). This path has a concrete format that uses dots (".") as delimiters. For example, the preference that handles if the home button should be visible or not is `show_home_button`, and the path of this preference is `browser.show_home_button` and it contains a Boolean value.

The final HMAC is a string with, some peculiarities. For example, all empty arrays and objects are removed, or; the character "!" is replaced by its Unicode representation ("\u003C"). In the example, the value of the home button would be `"show_home_button":true`.

3.3 HMAC Reproduction

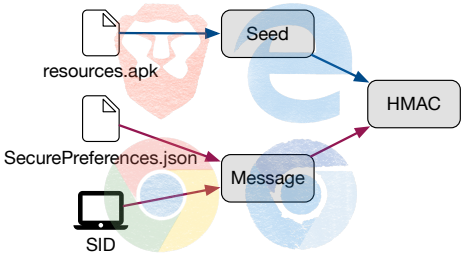

Fig. 2. HMAC protocol in Chromium based browsers

The function `GetDigestString` located in `pref_hash_calculator.cc` file is the one that generates an HMAC given a message and a key as inputs. The key has already been described in Section 3.1 whereas the message was described in Section 3.2. Therefore, we can impersonate the browser and generate HMACs to change any of the values of the Secure Preferences file as if we were the

browser. An illustrative summary of the HMAC protocol in Chromium-based browsers can be seen in Figure 2. Once the HMACs are computed (one per modified value in the Secure Preferences file) then they are combined to create a new message that is used as input of the hash algorithm to calculate the final HMAC called *super_mac*. The Secure Preferences file is then updated with the result of these calculations together with the modified preference values.

Chromium has a validation mechanism to check the integrity of the HMACs which is also calculated in the `Validate` function of the `pref_hash_calculator.cc` file. Such a function takes three parameters as input: a path of the JSON file, a value of the JSON file and a digest string which is the current HMAC of that value. Inside that function, another function called `VerifyDigestString` (which is also located in the same file, i.e., `pref_hash_calculator.cc`) takes as inputs a key—which is a string, a message—generated from the function `GetMessage` on `pref_hash_calculator.cc`, and a digest string—which is the HMAC. After being verified by the function `Verify` located on `hmac.cc`, a SHA256 string is returned.

4 Related Work

Many researchers have analyzed browser extensions from the security and privacy point of view (e.g., [5, 7, 10, 12, 15, 19, 23, 25, 26, 31]) but very little research has been conducted about how browser preferences and the Secure Preferences file can be used by malicious software to attack user's privacy or security.

The first attack against the Secure Preferences file, on Chrome for Windows, was described in one Internet forum in 2015, where it was shown how this file could be silently modified [13]. We confirmed this and developed a new attack based on that one that combined can be used to modify the Secure Preferences file of any version of any Chromiun-based browser. Indeed, we turned a less known narrowly-targeted attack (that only worked for Chrome and only on Windows) into a powerful platform-independent attack that exploits the most important Chromium-based browsers. Furthermore, we presented a systematic study of this class of attack and investigated its hefty consequences for browser hijacking and browser extensions.

Far from being part of Chrome security model, the same year, Banescu *et al.* [6] assumed the existence of a type malware called changeware with no root privileges. This malware is typically installed by Internet toolbars, banners or the execution of executable files like installers whose goal is to change user's configuration files. However, no more information was provided about how the attack could be performed.

In most, if not all, the referenced papers try to find security solutions for browser extensions without being concerned about the entry point of these preferences in the browser. Active extensions, web accessible resources, permissions they have, silent (un)installations or the path of installation where all the files

and extra files are located in the OS are a few examples of topics covered in the literature. We went one step forward and described an attack to the Secure Preferences file where all the preferences of the user are stored. We can actually modify any of these settings and thus by pass most of the proposed solutions in the literature, originating new security and privacy issues.

5 Conclusions

In this paper, we revisited the security and privacy of Chromium's mechanism to access the Secure Preferences file. Google introduced a security mechanism based on a cryptographic algorithm named HMAC to avoid users and applications other than the browser modifying the Secure Preferences file. We found that the seed used for the HMAC is fixed making Chromium vulnerable to PUP. We analyzed the three most important Chromium-based browsers, i.e., Brave, Chrome, Edge and Opera. Last but not least, this paper demonstrates that it is possible to perform browser hijacking, browser extension fingerprinting and remote code execution attacks as well as silent browser extensions (un)installation by coding a platform-independent proof-of-concept changeware that exploits the HMAC, freely modifying the Secure Preferences file. Our changeware, in combination with the one proposed years ago [13], can be used to modify such preferences file of any Chromium version from 25 to the latest one (85.0).

Acknowledgments This work was partially supported by the Swedish Foundation for Strategic Research (SSF) and the Swedish Research Council (Vetenskapsrådet) under grant Nr. 2015-04154 (PolUser: Rich User-Controlled Privacy Policies).

References

1. 2-spyware: Remove Softonic. `https://www.2-spyware.com/remove-softonic.html` (2019)
2. Aggarwal, A., Viswanath, B., Zhang, L., Kumar, S., Shah, A., Kumaraguru, P.: I spy with my little eye: Analysis and detection of spying browser extensions. In: EuroS&P. pp. 47–61 (April 2018)
3. Arshad, S., Kharraz, A., Robertson, W.: Identifying extension-based ad injection via fine-grained web content provenance. In: RAID. vol. 9854, pp. 415–436 (2016)
4. Awakesecurity: Discovery of a massive, criminal surveillance campaign. https://awakesecurity.com/blog/the-internets-new-arms-dealers-malicious-domain-registrars/ (2020)
5. Bandhakavi, S., Tiku, N., Pittman, W., King, S.T., Madhusudan, P., Winslett, M.: Vetting browser extensions for security vulnerabilities with VEX. Commun. ACM **54**(9), 91–99 (Sep 2011)
6. Banescu, S., Pretschner, A., Battré, D., Cazzulani, S., Shield, R., Thompson, G.: Software-Based Protection against Changeware. In: CODASPY. pp. 231–242 (2015)

7. Carlini, N., Felt, A.P., Wagner, D.: An evaluation of the google chrome extension security architecture. In: USENIX. pp. 97–111 (2012)
8. Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Selected Areas in Cryptography. pp. 250–270 (2003)
9. Chromium: No more silent extension installs. http://blog.chromium.org (2019)
10. Dhawan, M., Ganapathy, V.: Analyzing information flow in javascript-based browser extensions. In: ACSAC. pp. 382–391 (2009)
11. gs.statcounter: Browser market share. `https://gs.statcounter.com/browser-market-share` (2020)
12. Guha, A., Fredrikson, M., Livshits, B., Swamy, N.: Verified security for browser extensions. In: S&P. pp. 115–130 (2011)
13. HMAC: Chromium Secure Preferences. `https://kaimi.io/2015/04/google-chrome-and-secure-preferences/` (2019)
14. Jagpal, N., Dingle, E., Gravel, J.P., Mavrommatis, P., Provos, N., Rajab, M.A., Thomas, K.: Trends and lessons from three years fighting malicious extensions. In: USENIX. pp. 579–593 (2015)
15. Kapravelos, A., Grier, C., Chachra, N., Kruegel, C., Vigna, G., Paxson, V.: Hulk: Eliciting malicious behavior in browser extensions. In: USENIX. pp. 641–654 (2014)
16. Kotzias, P., Matic, S., Rivera, R., Caballero, J.: Certified pup: Abuse in authenticode code signing. In: CCS. pp. 465–478 (2015)
17. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication. Internet Engineering Task Force (IETF) (1997)
18. Laperdrix, P., Bielova, N., Baudry, B., Avoine, G.: Browser fingerprinting: A survey. CoRR **abs/1905.01051** (2019), `http://arxiv.org/abs/1905.01051`
19. Lerner, B.S., Elberty, L., Poole, N., Krishnamurthi, S.: Verifying web browser extensions' compliance with private-browsing mode. In: ESORICS. pp. 57–74 (2013)
20. Malwarebytes: Billion-dollar search engine industry attracts vultures, shady advertisers, and cybercriminals. https://blog.malwarebytes.com (2020)
21. Microsoft: Microsoft edge: Making the web better through more open source collaboration. `https://bit.ly/2QeZFwm` (2019)
22. Rogowski, R., Morton, M., Li, F., Monrose, F., Snow, K.Z., Polychronakis, M.: Revisiting browser security in the modern era: New data-only attacks and defenses. In: EuroS&P. pp. 366–381 (April 2017)
23. Sánchez-Rola, I., Santos, I., Balzarotti, D.: Extension breakdown: Security analysis of browsers extension resources control policies. In: USENIX. pp. 679–694 (2017)
24. Sjösten, A., Van Acker, S., Picazo-Sanchez, P., Sabelfeld, A.: LATEX GLOVES: Protecting browser extensions from probing and revelation attacks. In: NDSS. p. 57 (2018)
25. Somé, D.F.: Empoweb: Empowering web applications with browser extensions. In: S&P. pp. 227–245 (May 2019)
26. Starov, O., Nikiforakis, N.: Xhound: Quantifying the fingerprintability of browser extensions. In: S&P. pp. 941–956 (2017)
27. Statcounter: Desktop Browser Market Share Worldwide. https://gs.statcounter.com (2019)
28. Urban, T., Tatang, D., Holz, T., Pohlmann, N.: Towards understanding privacy implications of adware and potentially unwanted programs. In: ESORICS. pp. 449–469 (2018)
29. w3schools: Browser Statistics. `https://www.w3schools.com/browsers/` (2019)
30. Xing, X., Meng, W., Lee, B., Weinsberg, U., Sheth, A., Perdisci, R., Lee, W.: Understanding malvertising through ad-injecting browser extensions. In: WWW. pp. 1286–1295 (2015)

31. Zhao, R., Yue, C., Yi, Q.: Automatic detection of information leakage vulnerabilities in browser extensions. In: WWW. pp. 1384–1394 (2015)

A Secure Preferences File

A.1 Brave

The *Default* directory where the Secure Preferences file is usually placed is `C:\Users\<user_name>\AppData\Local\BraveSoftware\Brave-Browser\User Data\Default` in Windows, and `~/Library/Application Support/BraveSoftware/Brave-Browser/Default` in MacOS.

A.2 Chrome

Preferences are stored in a plain text JSON file that can be found in a directory of every user's profile. If there is only one profile in the system, the Secure Preferences file is in the `Default` directory which is located in Windows under `C:\Users\<user_name>\AppData\Local\Google\Chrome\User Data\Default`. In MacOS can be found at `~/Library/Application Support/Google/Chrome/Default/` and in Linux at `~/.config/google-chrome/Default/`.

A.3 Microsoft Edge

The *Default* directory can be found under `C:\Users\<user_name>\AppData\Local\Microsoft\Edge Dev\User Data\Default` path in Windows and `~/Library/Application Support/Microsoft Edge/Default` in MacOS.

A.4 Opera

The *Default* directory where the Secure Preferences file is usually placed is `C:\Users\<user_name>\AppData\Local\BraveSoftware\Brave-Browser\User Data\Default` in Windows, and `~/Library/Application Support/com.operasoftware.Opera` in MacOS.

B Resources.pak File

B.1 Brave

In Brave, the `resources.pak` file can be found in `C:\Program Files\BraveSoftware\Brave-Browser\Application\<version>` in Windows, and in `/Applications/Brave Browser.app/Contents/Frameworks/Brave Browser Framework.framework/Versions/Current/Resources` in MacOS.

B.2 Chrome

In Chrome, `resource.pak` file can be usually found in `C:\Program Files\Google\Chrome\Application` (Windows), `/Applications/Google Chrome.app/Contents/Frameworks/Google Chrome Framework.framework/Versions/Current/Resources` (MacOS), and `/opt/google/chrome` (Linux).

B.3 Microsoft Edge

In Edge, `resource.pak` is usually at `C:\Program Files\Microsoft\Edge\Application\<version>` in Windows, and `/Applications/Edge Dev.app/Contents/Frameworks/Microsoft Edge Framework.framework/Versions/Current/Resources` in MacOS

B.4 Opera

In Opera, the `resources.pak` file is actually called `opera.pak` and it can be found in `C:\Program Files\BraveSoftware\Brave-Browser\Application\<version>` in Windows, and in `/Applications/Opera.app/Contents/Frameworks/Opera Framework.framework/Versions/71.0.3770.148/Resources` in MacOS.

C Source Code

```python
def calculate_seed(data):
    def entry_at_index(idx):
        header_size = 12
        resourceSize = 2 + 4
        offset = header_size + idx * resourceSize
        return struct.unpack('<HI', data[offset:offset +
            resourceSize])[1]

    encoding, resource_count, alias_count = struct.unpack
        ('<BxxxHH', data[4:12])
    pre_offset = entry_at_index(0)

    for i in range(1, resource_count + 1):
        offset = entry_at_index(i)
        if (offset - pre_offset == 64):
            seed=data[pre_offset:offset]
            break
        pre_offset = offset

    return seed
```

Fig. 3. Script to extract the seed used to generate the HMAC on Chrome.

```
{
  "name": "name",
  "description": "description",
  "path": "<path_to_host_app>",
  "type": "stdio",
  "allowed_origins": ["chrome-extension://<ext_id>/"]
}
```

Fig. 4. Host manifest file.