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 was 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, allowing for free modification of the Secure Preferences file. Last but not least, we analyze the security of the four 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 [42]. 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 [27]. Given its widespread use, around 75% of the desktop users on Internet [38], the security of Chromium is paramount.

To allow easy customization of the web browser to fit the needs of the users, many configuration parameters may be modified. Setting the homepage to a custom webpage 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 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 [18], 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 [18, 19, 34, 36], vulnerabilities still abound [2, 3, 35], being Potentially Unwanted Programs (PUPs) one popular and challenging example because they are not usually marked as malware by antivirus vendors [21, 40].

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 add 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 Section 3).

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 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 its browser's security 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 now allowed.

Concretely, from its version 25 Chromium implemented a security 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 [22] 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. In previous work, it has been proposed to use WhiteBox-Cryptography [10] 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 [17]. 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 [17]
58.0.2988.0	2017-01	Attack [17]
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 [31, 43], fingerprinting attacks [2, 23, 34], remote code execution [35], as well as silent browser extensions (un)installation (something Google has in principle banned years ago [11]). 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, 25]; 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 [15]—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 has not been altered by any other party different than the browser (Section 2.2). We implement a changeware that impersonates the browser and (un)install extensions, perform phishing attacks, hijack the user's browser, fingerprint users through the extensions the browser has, among other things (Section 3).

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

Section 2 presents background information concerning the Secure Preferences file and how Chromium uses it. Section 4 exposes some countermeasures to avoid the attack as well as a brief discussion about how this vulnerability can be used by the research community for analyzing browser extensions. Finally, Section 5 presents the related work and Section 6 concludes the paper.

2 Background

In this section, we explain the role of the Secure Preferences file and how the HMAC is generated in Chromium-based browsers.

2.1 Chromium Preferences

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.

We show how Secure Preferences file works via an example. Let Alice be a user who wants to manually modify any of the preferences stored in the Secure Preferences file. She accesses 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 wants to. Once she has modified the file, she saves it and launches her Chromium instance to check whether the changes have been applied or not. 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. Chromium then 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 expecting her changes to take effect. That, however, 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 rationale behind Chromium's behavior is to avoid external modifications to the Secure Preferences file for privacy reasons. 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 [22] 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 is usually used for both data verification and 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 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.

2.2 HMAC in 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 re-computes 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 it computes both the HMACs of the settings, and a final HMAC called *super_mac*.

Acquiring the seed The seed is stored in the `resource.pak` file. We explain in what follows how we get the seeds of the latest versions as of June 2020 of the four browsers being considered.

Chrome The seed that Chrome uses to compute the HMAC is a 64-long character hexadecimal string that can be found in the `resource.pak` file. Concretely, the first resource that has a length of 256 binary bits in the `resource.pak` file is the seed Chromium uses. Roughly speaking, we obtain this resource 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\O\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 as for Chrome 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 these four vendors: 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.

Obtaining the Message 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 four 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.

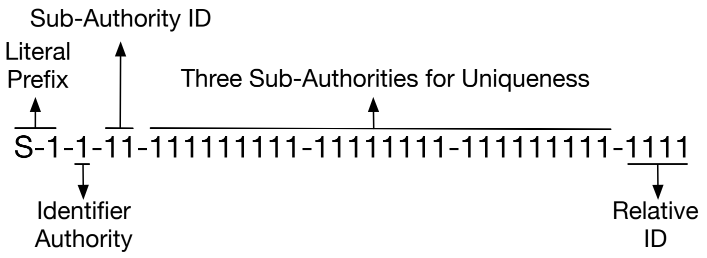

Sub-Authority ID
Literal Prefix
Three Sub-Authorities for Uniqueness
S-1-1-11-11111111-1111111-11111111-1111
Identifier Authority
Relative ID

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 obtained 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 Mac OS but there are no rules for Linux. 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 manually analyzing the message in Chromium, we realized that it is composed of key of the Secure Preferences file value it wants to modify together with either the SID (or the UUID) of the current user (or computer). 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*, 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 where the Secure Preferences file is in the computer. It has a concrete format that uses dots (".") as delimiters. For example, the preference that handles if the home button is visible or not is `show_home_button`, being the path `browser.show_home_button` and it contains a Boolean value.

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

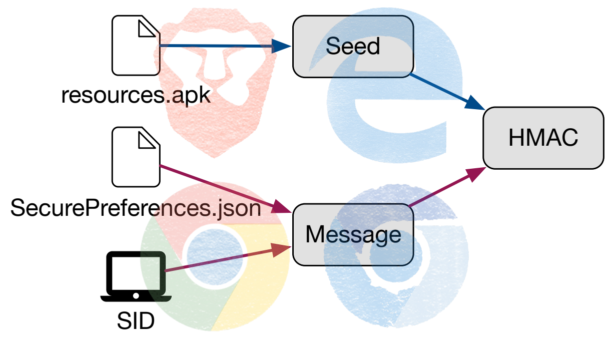


Fig. 2. HMAC protocol in Chromium based browsers

HMAC Reproduction 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 and the message are as described above. 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) they are then 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 (a string), a message (generated from the function `GetMessage` on `pref_hash_calculator.cc`), and a digest string (the HMAC). After being verified by the function `Verify` located on `hmac.cc`, a SHA256 string is returned.

3 Security Analysis

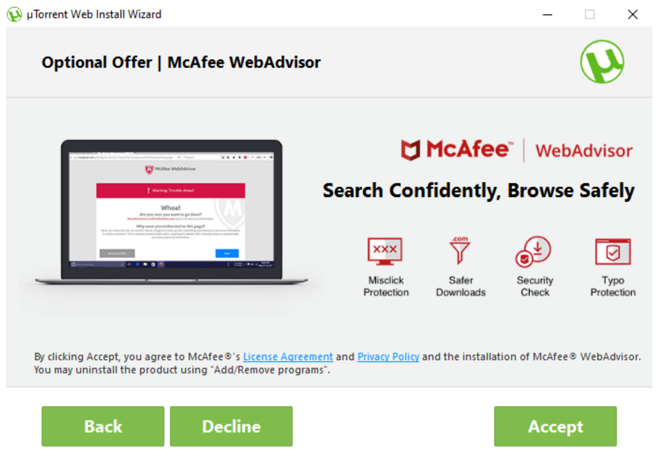
In what follows, we introduce the attacker model and provide some examples that exploit the HMAC detailed in the previous section to modify the Secure Preferences file. We present a proof-of-concept whose source code we released for future research on the field[2]. Finally, we analyze the main differences between the installed-by-default extensions in Brave, Chrome and Edge and Opera browsers, and demonstrate how an external server can execute some parts of the code of the installed-by-default extensions creating a big security threat.

3.1 Attacker Model

Our attacker model is composed of any software application that specifically alters the Secure Preferences file of Chromium. This attacker model is known in the literature as Potentially Unwanted Programs (PUPs) which are executable files that apart from the desired program installation also install other software that might not be related to the legitimate one, typically *adware* [21, 40]. What makes PUPs different from malware is that users are tricked to approve the installation of this third party application. Typically, during the installation process the PUPs shows a message that the user has to (un)check before the process continues.

More specifically, there is a subset of PUPs called *changeware* whose aim is to modify the settings of the browser [6], usually for malicious purposes as confused deputy. Let us give an illustrative example. Years ago, Oracle used to include

[2] `https://github.com/Pica4x6/SecurePreferencesFile`


µTorrent Web Install Wizard

Optional Offer | McAfee WebAdvisor

Search Confidently, Browse Safely

By clicking Accept, you agree to McAfee®'s License Agreement and Privacy Policy and the installation of McAfee® WebAdvisor. You may uninstall the product using "Add/Remove programs".

Fig. 3. uTorrent Installation Process.

in the Java installation file one selected-by-default checkbox by which a Yahoo toolbar was automatically installed unless the user did not manually uncheck it during the installation process [39]. Similar cases were seen with WinYahoo which was installed as part of the Adobe Photoshop Album Starter Edition software [26]. In all the aforementioned cases, the attackers are the binaries that modify the Secure Preferences file. In both cases, unwanted browser extensions are installed in the user's browser. Note that browser extensions can usually have access to any website (sensitive or not) that the user visits.

Even though the issue made apparent in the examples above was identified and marked as PUPs some time after its detection, there still are many up-to-date examples of applications where browser extensions are piggyback programs. One such example is *uTorrent Web* binary installation. Concretely, during the installation process, the user has to accept or decline the installation of *McAfee WebAdvisor* software (see Figure 3). If the user accepts, that "extra" software is installed together with a browser extension which is automatically installed in Chrome (see Figure 4). However, when the user manually uninstalls the McAfee WebAdvisor application, the extension might also be uninstalled from Chrome. This clearly indicates that there still are applications that can install browser extensions without requiring the user to use Google WebStore as it is claimed.

3.2 Changeware Proof-of-Concept

The challenging part of PUPs is that they are not usually marked by antivirus vendors as malicious software [21, 40]. Windows claimed to stop PUPs and they even added such an option as part of the recently released Microsoft Edge [12]. We created a changeware and confirmed that it is not classified as malware by the Microsoft detection mechanism [12, 29]. Additionally, we run a set of popular online antivirus tests like Virustotal[3], MetaDefender[4] and VirScan[5] and our changeware passed all the security checks. As a conclusion, we can effectivelly alter the Secure Preferences file with no restrictions at all. All files to reproduce the attacks above are publicly available in `https://github.com/Pica4x6/SecurePreferencesFile`.

3.3 Practical Attacks

We analyze now the main attacks a changeware can exploit. In particular, we classify the attacks into browser hijacking and browser extensions. Most of these attacks are interconnected since the goal of the attacks is to get the private information of the user. Some years ago Kotzias *et al.* [21] analyzed almost 4 million hosts and conclude that half of them have some kind of PUPs installed. More recently, Urban *et al.* [40] analyzed the communication carried out by 16k PUPs and 5.5k Firefox extensions and got that almost 40% and 45% include personal information of the user respectively.

Browser Hijacking The goal of this attack is to increase the advertising revenue by forcing the user to access concrete webpages. To redirect users to such sites, changeware may modify up to five main values of the Secure Preferences file, namely: i) `homepage`; ii) `pinned tabs`; iii) `import_bookmarks_from_file`; iv) `search engine`, and; v) `sessions` keys. Antivirus vendors usually identify this attack by parsing the Secure Preferences file and analyzing the URLs defined in it. If they belong to a *blacklist* the antivirus constantly keeps updated, then an unwanted change might be detected and the antivirus analyzes the disk looking for malicious software. However, this method can be bypassed by modifying the `import_bookmarks_from_file`. This is a special option in the manifest which states the path where Chrome silently and automatically imports bookmarks from the HTML stated in the path of such key.

Phishing The goal of this type of attack is to steal user's private information. This is done by loading a fake webpage that looks similar to the legitimate one. If not aware of the URL, the user will interact with the site as usual. To trick users, browser extensions can implement some strategies to redirect them to fake pages and perform phishing attacks [41], analyze the most visited web pages and generate bookmarks, change the pinned tabs they already have or even generate new ones.

Browser Extensions: Execution Order, Paths and Fingerprinting Recently, Picazo-Sanchez *et al.* demonstrated that the order in which browser extensions are executed may alter the content of the DOM and the behavior of the browser in general [30]. The attack was implemented corroborating that the changeware could

[3] `https://www.virustotal.com`
[4] `https://metadefender.opswat.com`
[5] `https://www.virscan.org`

modify the installation time of extensions altering the execution order. Moreover, it can also modify any of the paths the extensions define in the manifest, being possible to include new paths in the user's file system loading different extension files. Different techniques have been proposed so far to fingerprint browser extensions, i.e., using Web Accessible Resources (WARs) [34], using behavioral-based enumeration [36, 37] or because inter or extra communication messages [20, 35]. Any of these methods can be easily exploited by the changeware. This could be done for instance by defining and including new resources as WARs, including JavaScripts into the extensions files that automatically inserts content into the DOM, deleting the `externally_connectable` key of the extensions so that other webpages can send messages to the background pages of the extensions (we show an example of this attack in Section 3.4 by analyzing the installed-by-default extensions) or a combination of them.

Browser Extensions: Permissions Chromium offers a set of APIs that extensions can use, being some of them accessible by defining the corresponding permission in the manifest of the extensions. Once installed, the manifest is parsed and stored as part of the Secure Preferences file under the extension id key, therefore the original manifest is no longer checked. This poses serious security issues since any changeware might alter the permissions the user agreed upon and either provide it with more or fewer permissions than initially. Let us give a concrete example, the browser extension whose id is `mgpdmkkhjffhfkbpeigghejkngiaaike` and more than 400,000 downloads, includes in the background.js file `return chrome.webRequest.onCompleted` but it does not include the `webRequest` permission needed to execute such statement. The changeware can easily add such permission into the Secure Preferences file giving access to that API to the extensions and thus, executing that line without any errors.

Browser Extensions: Silent Installation Even though Google banned silent browser installations in 2012, we managed to successfully install, delete, activate and deactivate browser extensions. See Figure 4 for a real example used today by software that includes a browser extension in the browser without using the official WebStore. In this particular case, the extension can be seen and manually removed by the user, but this might not always be the case. In the worst-case scenario, the changeware could have the source of the extension to be installed inside the binary file.

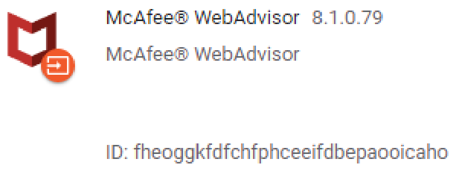

McAfee® WebAdvisor 8.1.0.79
McAfee® WebAdvisor
ID: fheoggkfdfchfphceeifdbepaooicaho

Fig. 4. McAfee Browser Extension.

Note that is then possible to install new browser extensions without the user being notified similarly to how installed-by-default extensions work (see Tables 3 to 6). Concretely, enabling and disabling extensions is determined by the `state` key of the extension in the Secure Preferences file being straightforward to modify them. Also, remark that to uninstall an extension, the changeware can simply delete the whole entry of the preferences file and computing the `super_mac` of the Secure Preferences file.

3.4 Installed-by-Default Extensions

In the following, we analyze the extensions that are installed by default with the browser, for all four browser under consideration.

Brave This browser has ten installed-by-default apps where none of them are extensions and they cannot be removed by the user (see Table 3). Brave, renames some of the default extensions despite being exactly the same as the one provided by Chromium, e.g., *Chromium PDF viewer*. However, what makes Brave different from the other browsers is that it allows the user to disable (not to uninstall) some installed-by-default apps, e.g., WebTorrent, Google Hangsout and Crypto Wallet.

Chrome We confirmed that the number of default extensions is sixteen (between browser extensions and apps) in the three OSs. In addition to that, only a subset of them might be uninstalled by the user in the usual way, i.e., going either to `chrome://extensions` or `chrome://apps` and manually removing them. We show a detailed list of the installed-by-default extensions in Table 5.

Regarding the platform, our initial hypothesis was that Linux was the most privacy compliant of the evaluated OSs. After running the first part of the experiments we confirmed that indeed Chrome does not install any single browser extension by default in Linux (contrarily to what happens in other OSs), and the file is totally empty except for the `super_mac` key. We realized that Linux does not modify such a file but `Preferences` file, so we had to adapt our tests to use that Preferences file instead.

Microsoft Edge Unlike Chrome, Microsoft Edge has ten installed-by-default applications and only one extension (see Table 4). Most, if not all, browser extensions developed for one particular Chromium-based browser can be easily exported to other Chromium-based browsers. There are cases where vendors can modify some parameters like the name of the extensions, e.g., `mhjfbmdgcfjbbpaeojofohoefgiehjai` is named here *Microsoft Edge PDF Viewer* and *Chrome PDF Viewer* on Chrome. We tested Edge on Windows and MacOS without noticing differences when the Secure Preferences file is generated for the first time. It is also interesting to mention that there is no way for the user to get rid of any default extensions on Edge.

Opera This browser is the one that has more information by default in the Secure Preferences file when it is installed for the first time. Concretely, there are more than 300 extensions hardcoded whose purpose is to ban them to be installed by the user— a disallowed list. Other than that, there are 21 extensions installed by default and none of them can actually be uninstalled nor disabled by the user. Table 6 shows a list of the installed-by-default extensions in Opera.

3.5 Google Hangsout Use Case

There is an unexplored set of extensions that are typically overlooked by the research community: installed-by-default extensions. We manually analyzed all of them and realized that *Google Docs Offline*, *Chrome Media Router*, *CryptoTokenExtension* and *Google Hangouts*, implement external message listeners and have the `externally_connectable` key defined in their manifest files. Given that only the *Google Docs Offline* extension can be deleted by the user, if a changeware modifies such key in the Secure Preferences file any website can send messages to these extensions as if they were legitimate websites.

Concretely, Google Hangsout is one of these installed-by-default extensions present in all the Chromium-based browsers. It cannot be uninstalled by the user with the only exception of Brave which can be disabled. In the following, we analyze it to demonstrate the information that an attacker can get by exploiting the Secure Preferences file.

A recent study performed by Somé demonstrates how browser extensions allow web applications to bypass the Same Origin Policy and have access to sensitive information of the user [35]. Concretely, extensions that listen for external messages should be defined in advance in the manifest similar to what the `externally_connectable` does. If, for instance, such a key is not defined in the manifest file and extensions implement any of the external message listeners (i.e., `onMessageExternal` and `onConnectExternal`), they will listen and execute the code defined in any of these functions. Moreover, if any dangerous function like `eval()` is defined in these listeners, the consequences might be catastrophic since the attacker can take control of the extension and run arbitrary code on it.

```
var editorExtensionId="nkeimhogjdpnpccoofpliimaahmaaome";
var port_a = chrome.runtime.connect(editorExtensionId,{name: "
    processCpu"});
var port_b = chrome.runtime.connect(editorExtensionId,{name: "
    chooseDesktopMedia"});
port_b.postMessage({
    method: 'chooseDesktopMedia',
    // sources: ['screen','window','tab','audio']
    sources: ["window"]
});

port_a.onMessage.addListener(
    function(msg) {console.log(msg)});
port_b.onMessage.addListener(
    function(msg) {console.log(msg)});
```

Fig. 5. Script that the attacker injects to extract information from the user.

Google Hangsout defines the pattern `https://*.google.com/*` as trusted webpages, making thus possible for any (sub)domain of `google.com` send messages to the extension. We created a dummy server (*http://www.attacker.com*) and added it to the `externally_connectable` list by using the attack described in Section 3. We manually analyzed such extension and realized that the attacker can obtain information like `{"browserCpuUsage":2.2,"gpuCpuUsage":3.0," tabCpuUsage":0.0,"tabJsMemoryAllocated":3133440,"tabJsMemoryUsed":1743032, "tabNetworkUsage":0}` by using a port named "processCpu" (line 3 of Figure 5).

Apart from that, by setting the right parameters (lines 4 and 5 of Figure 5) the attacker may execute the `chrome.desktopCapture.chooseDesktopMedia(array of DesktopCaptureSourceType sources, tabs.Tab targetTab, function callback)` function where the array of sources can be either "screen", "window", "tab", or "audio" according to the official documentation provided by Google. The attacker can then get a screenshot of: 1) the current screen of the user's computer; 2) any software the user is running; 3) the tab of the browser, and; 4) the audio of user.

Finally, with our changeware we can remove the entire `externally_connectable` entry of the Hangsout extension from the Secure Preferences file. In fact, any browser extension can execute those functions and retrieve such information. In addition, by including a list of allowed sites in the `matches` list of the `externally_connectable` key, any website can also execute and get these data.

4 Discussion

Here we discuss countermeasures and proposals to prevent the Secure Preferences file attack as well as the potential benefits that our attacks have for future analysis of the extensions.

Coordinated Disclosure. We contacted the four vendors: Brave, Google, Microsoft and Opera to report our findings. Brave is in progress of fixing the problem.Google acknowledged that "defeating the HMAC is a signal that the software is in violation of the Unwanted Software Policy". Both Microsoft and Opera deferred to the Chromium project.

Preventing the SPF Attack. In our approach, we first generate a nonce—a random values of 64-character long string—from a uniform distribution, and use it to replace the seed already stored in the `resources.pak` file. When we launched Chromium for the first time after that change, it showed an alert pop-up saying that something went wrong and the configurations were to be restored. After that, we did not notice any difference when working with Chromium while surfing the web, installing/uninstalling browser extensions, adding plugins, modifying the homepage, adding bookmarks or adding/deleting pinned tabs.

However, even if we generate random seeds to mitigate the attack, the problem remains if the nonce is still stored in the file system. We thus implement a script to generate the random seed each time Chromium is about to open. The problem with this solution is that Chromium became impractical since it was always trying to restore the file from external changes (remember that the Secure Preferences file is analyzed and loaded each time Chromium is launched). If the seed is changed, the HMAC protection mechanism implemented by the Secure Preferences file should also be updated, generating thus new values for all the `macs` as well as for the `super_mac`. The conclusion is that the seed generation is not secure as long as it is stored somewhere in the file system of the user: performing a reverse engineering process is enough to reveal where the seed is stored, being therefore easy to get it.

Briefly, either the seed generation, the Secure Preferences file or the seed storage have to be protected from unauthorized parties. To achieve that we propose a solution based on Trusted Platform Module (TPM), in which case the seed should automatically be generated and stored in a secure memory so only the browser can access it (as Windows 10 currently does [28]). A limitation of the usage of TPMs, despite being widely extended, is that not all computers have one.

Alternative solutions to TPM, e.g., Intel SGX, ARM Trusted Zone or MAC secure enclave, could be considered. In such cases, the browser can either partially or totally run the seed generation procedure in the enclave and securely store the generated seed. Moreover, the browser could also store the Secure Preferences file in the enclave so no other parties different than the browser can access it.

Potential Benefits. Creating a controlled environment to execute browser extensions and analyze them is difficult. Honey pages have been widely used in the literature to fire the execution of browser extensions [9,19,36]. Our released source code can easily be used to modify extensions in such a way that the main functionality is not modified. Therefore, applying techniques like fuzzing, improving static analysis strategies or making dynamic analysis less demanding are some of the examples where our changeware can be helpful.

5 Related Work

Many researchers have analyzed browser extensions from the security and privacy point of view (e.g., [5,8,13,16,19,24,32,35,36,44]) 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 [17]. 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 Chromium-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.

In the same year, Banescu *et al.* [6] assumed the existence of a type of 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. In this paper, Banescu *et al.* proposed a solution based on White-Box Cryptography. As this type of cryptographic tool is insecure [7, 33], they include software diversity [14] to mitigate attacks against this cryptographic scheme [6]. Since the attackers are not aware of the used obfuscation transformation, they need to explore all the possible generated binaries to run cryptanalysis. Despite being a promising technique, the proposed solution is only deployable in Windows since they do not modify the kernel of the operating system. A modification of the kernel would be mandatory in Linux and Mac. In comparison to our paper, we focused on how to perform the attack against the Secure Preferences file and the consequences for the user.

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 bypassing most of the proposed solutions in the literature, originating new security and privacy issues.

6 Conclusions

We have 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. Our analysis was carried out on Brave, Chrome, Edge and Opera.

We have also demonstrated that it is possible to perform browser hijacking, browser extension fingerprinting and remote code execution attacks as well as silent browser extensions (un)installation. We did so 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 in [17], can be used to modify such a preferences file of any Chromium version later than v.25 (including the latest one, v.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: 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. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: Hiding your white-box designs is not enough. In: CHES. pp. 215–236 (2016)
8. Carlini, N., Felt, A.P., Wagner, D.: An evaluation of the google chrome extension security architecture. In: USENIX. pp. 97–111 (2012)
9. Chen, Q., Kapravelos, A.: Mystique: Uncovering information leakage from browser extensions. In: CCS. p. 1687–1700 (2018)
10. 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)
11. Chromium: No more silent extension installs. http://blog.chromium.org (2019)
12. Cimpanu, C.: Windows 10 to get pua/pup protection feature. `https://www.zdnet.com/article/windows-10-to-get-puapup-protection-feature/` (2020)
13. Dhawan, M., Ganapathy, V.: Analyzing information flow in javascript-based browser extensions. In: ACSAC. pp. 382–391 (2009)
14. Forrest, S., Somayaji, A., Ackley, D.H.: Building diverse computer systems. In: Workshop on Hot Topics in Operating Systems. pp. 67–72 (May 1997)
15. gs.statcounter: Browser market share. `https://gs.statcounter.com/browser-market-share` (2020)
16. Guha, A., Fredrikson, M., Livshits, B., Swamy, N.: Verified security for browser extensions. In: S&P. pp. 115–130 (2011)
17. HMAC: Chromium Secure Preferences. `https://kaimi.io/2015/04/google-chrome-and-secure-preferences/` (2019)
18. 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)
19. 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)
20. Karami, S., Ilia, P., Solomos, K., Polakis, J.: Carnus: Exploring the privacy threats of browser extension fingerprinting. In: NDSS (2020)
21. Kotzias, P., Matic, S., Rivera, R., Caballero, J.: Certified pup: Abuse in authenticode code signing. In: CCS. pp. 465–478 (2015)
22. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication. Internet Engineering Task Force (IETF) (1997)

23. Laperdrix, P., Bielova, N., Baudry, B., Avoine, G.: Browser fingerprinting: A survey. CoRR **abs/1905.01051** (2019), `http://arxiv.org/abs/1905.01051`
24. Lerner, B.S., Elberty, L., Poole, N., Krishnamurthi, S.: Verifying web browser extensions' compliance with private-browsing mode. In: ESORICS. pp. 57–74 (2013)
25. Malwarebytes: Billion-dollar search engine industry attracts vultures, shady advertisers, and cybercriminals. https://blog.malwarebytes.com (2020)
26. Malwarebytes: WinYahoo. https://blog.malwarebytes.com (2020)
27. Microsoft: Microsoft edge: Making the web better through more open source collaboration. `https://bit.ly/2QeZFwm` (2019)
28. Microsoft: How windows 10 uses the trusted platform module. (2020)
29. Microsoft: Windows defender and secure preferences file. https://answers.microsoft.com (2020)
30. Picazo-Sanchez, P., Tapiador, J., Schneider, G.: After you, please: Browser extensions order attacks and countermeasures. International Journal of Information Security pp. 1–16 (2019)
31. 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)
32. Sánchez-Rola, I., Santos, I., Balzarotti, D.: Extension breakdown: Security analysis of browsers extension resources control policies. In: USENIX. pp. 679–694 (2017)
33. Sanfelix, E., Mune, C., de Haas, J.: Unboxing the white-box. In: Black Hat EU 2015 (2015)
34. Sjösten, A., Van Acker, S., Picazo-Sanchez, P., Sabelfeld, A.: LATEX GLOVES: Protecting browser extensions from probing and revelation attacks. In: NDSS (2018)
35. Somé, D.F.: Empoweb: Empowering web applications with browser extensions. In: S&P. pp. 227–245 (May 2019)
36. Starov, O., Nikiforakis, N.: Xhound: Quantifying the fingerprintability of browser extensions. In: S&P. pp. 941–956 (2017)
37. Starov, O., Laperdrix, P., Kapravelos, A., Nikiforakis, N.: Unnecessarily identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In: WWW. p. 3244–3250 (2019)
38. Statcounter: Desktop Browser Market Share Worldwide. https://gs.statcounter.com (2019)
39. UK, P.: Update java, get yahoo as your default search engine. https://uk.pcmag.com (2019)
40. Urban, T., Tatang, D., Holz, T., Pohlmann, N.: Towards understanding privacy implications of adware and potentially unwanted programs. In: ESORICS. pp. 449–469 (2018)
41. Varshney, G., Misra, M., Atrey, P.K.: Detecting spying and fraud browser extensions: Short paper. In: MPS. p. 45–52 (2017)
42. w3schools: Browser Statistics. `https://www.w3schools.com/browsers/` (2019)
43. 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)
44. Zhao, R., Yue, C., Yi, Q.: Automatic detection of information leakage vulnerabilities in browser extensions. In: WWW. pp. 1384–1394 (2015)

A Installed-by-default extensions

Table 3. Brave installed-by-default extensions.

ExensionID	Name	Uninstallable
ahfgeienlihckogmohjhadlkjgocpleb	Web Store	✗
jidkidbbcafjabdphckchenhfomhnfma	Brave Rewards	✗
kmendfapggjehodndflmmgagdbamhnfd	CryptoTokenExtension	✗
lgjmpdmojkpocjcopdikifhejkkjglho	Brave Webtorrent	can be disabled
mfehgcgbbipciphmccgaenjidiccnmng	Cloud Print	✗
mhjfbmdgcfjbbpaeojofohoefgiehjai	Chromium PDF Viewer	✗
mnojpmjdmbbfmejpflffifhffcmidifd	Brave	✗
nkeimhogjdpnpccoofpliimaahmaaome	Google Hangouts	can be disabled
odbfpeeihdkbihmopkbjmoonfanlbfcl	Crypto Wallets	can be disabled
oemmndcbldboiebfnladdacbdfmadadm	PDF Viewer	✗

Table 4. Microsoft Edge installed-by-default extensions.

ExensionID	Name	Uninstallable
dgiklkfkllikcanfonkcabmbdfmgleag	Edge Clipboard	✗
fikbjbembnmfhppjfnmfkahdhfohhjmg	Media Internals Services Extension	✗
fogppepbgmgkpdkinbojbibkhoffpief	Edge Collections	✗
iglcjdemknebjbklcgkfaebgojjphkec	Microsoft Store	✗
ihmafllikibpmigkcoadcmckbfhibefp	Edge Feedback	✗
jdiccldimpdaibmpdkjnbmckianbfold	Microsoft Voices	✗
kmendfapggjehodndflmmgagdbamhnfd	CryptoToken	✗
mhjfbmdgcfjbbpaeojofohoefgiehjai	Microsoft Edge PDF Viewer	✗
ncbjelpjchkpbikbpkcchkhkblodoama	WebRTC Internals Extension	✗
nkeimhogjdpnpccoofpliimaahmaaome	Google Hangouts	✗
pkedcjkdefgpdelpbcmbmeomcjbeemfm	Chrome Media Router	✗

Table 5. Chrome installed-by-default extensions.

ExensionID	Name	Uninstallable
aapocclcgogkmnckokdopfmhonfmgoek	Slides	✓
ahfgeienlihckogmohjhadlkjgocpleb	Web Store	✗
aohghmighlieiainnegkcijnfilokake	Docs	✓
apdfllckaahabafndbhieahigkjlhalf	Google Drive	✓
blpcfgokakmgnkcojhhkbfbldkacnbeo	Youtube	✓
felcaaldnbdncclmgdcncolpebgiejap	Sheets	✓
gfdkimpbcpahaombhbimeihdjnejgicl	Feedback	✗
ghbmnnjooekpmoecnnnilnnbdlolhkhi	Google Docs Offline	✓
kmendfapggjehodndflmmgagdbamhnfd	CryptoTokenExtension	✗
mfehgcgbbipciphmccgaenjidiccnmng	Cloud Print	✗
mhjfbmdgcfjbbpaeojofohoefgiehjai	Chrome PDF Viewer	✗
neajdppkdcdipfabeoofebfddakdcjhd	Google Network Speech	✗
nkeimhogjdpnpccoofpliimaahmaaome	Google Hangouts	✗
nmmhkkegccagdldgiimedpiccmgmieda	Google Wallet	✗
pjkljhegncpnkpknbcohdijeoejaedia	Gmail	✓
pkedcjkdefgpdelpbcmbmeomcjbeemfm	Chrome Media Router	✗

Table 6. Opera installed-by-default extensions.

ExensionID	Name	Uninstallable
apkgpnbdglipaagpckkbdbigfmmomobn	Onboarding popup	✗
bcibcaaakpeekhbnddgnajbmjdcemfkf	Opera Addons Portal	can be disabled
bennllbledkboeijomefbhpidmhfkoih	News feeds popup	✗
cgloclgndbkhmjcaddholfcgghcgmmig	Opera Welcome Page	✗
eeiccfifdclpgnnaagpkjfpkaabgcbne	SD suggestions list	✗
efpeldimhbhjejgcdcbhmjllaafhjmge	VKontakte Notifications	✗
enmlgamfkfdemjmlfjeeipglcfpomikn	News feed handler	✗
gfobfmjpcnapngbghpcbodncehngmdln	Opera Crypto Wallet	✗
hhckidpbkbmoeejbddojbdgidalionif	Video handler	✗
ibgcfekaaejggoajjnmknjcoieffdnod	Google Drive/Docs clipboard and notifications support	✗
ionkhgehfolinkdpgdbinmgbfaonpcnk	Amazon promotion	✗
jaocpokicpmlhbchlodlkiochdkmophj	Aliexpress observer	✗
kmendfapggjehodndflmmgagdbamhnfd	CryptoTokenExtension	✗
knohfebhibeknbfioecpdmdkjkjdnjnl	Bookmarks	✗
mfglbjdihkhhnimlecioccjbjiepicip	Opera Sync Auth Flow	✗
mhjfbmdgcfjbbpaeojofohoefgiehjai	Chromium PDF Viewer	✗
midfadfpkkakgcmbgpngfnfekghligek	Rate Opera	✗
nkeimhogjdpnpccoofpliimaahmaaome	Google Hangouts	✗
obhaigpnhcioanniiaepcgkdilopflbb	Background worker	✗
odndjkngipngdmdlfodecoelobjbidna	Opera In-App Notification Portal	✗
onigllbobbpllnfcjanphobocbkcdghh	Discord Notifications	✗