CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Who changed my browser settings?

Silently modifying the Secure Preferences of Chrome

Master's thesis in Computer Systems and Networks

Gustav Axelsson

Joakim Sundling

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Who changed my browser settings?

Silently modifying the Secure Preferences of Chrome

Gustav Axelsson
Joakim Sundling

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Who changed my browser settings?
Silently modifying the Secure Preferences of Chrome

Gustav Axelsson
Joakim Sundling

Supervisors: Pablo Picazo-Sanchez, Department of Computer Science and Engineering and Gerardo Schneider, Department of Computer Science and Engineering

Examiner: Katerina Mitrokotsa, Department of Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Abstract

Google Chrome is as of today the most used web browser in the world. With millions of daily users the security of the browser is of high importance. When using Google Chrome each user obtains a couple of configuration files for storing information such as bookmarks, browser history, homepage and a multitude of other settings. One of these files is the Secure Preferences file in which some of the browsers most sensitive settings are stored. In order to protect these settings Chrome has added custom Hash-based Message Authentication Codes (HMACs) that are used to ensure that no settings are silently modified by third parties. This thesis describes how this security can be circumvented and implements a versatile script, for Windows, that is able to alter all the information stored in Secure Preferences without alerting the browser. This thesis also describes the steps taken in order to reproduce the hashing mechanism of Chrome as well as how different preferences can be exploited. An extension is developed which makes it possible to run the script from the Chrome browser. The script is then evaluated together with the extension by both checking the correctness of the HMAC calculation and how well it is able to perform a variety of exploits. This thesis proves that it is indeed possible to break the security of the Secure Preference file. By reproducing and replacing the HMACs which gives the user of the script the possibility to alter frequently used functions in Chrome such as homepage, new tabs, extensions and default search engine.

Keywords: Computer, science, computer science, engineering, project, thesis, Google Chrome, preferences, secure preferences

Acknowledgements

We would like to thank our supervisors Pablo Picazo-Sanchez and Gerardo Schneider. We would also like to thank the examiner of the thesis, Katerina Mitrokotsa.

Gustav Axelsson, Gothenburg, June 2018
Joakim Sundling, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1
Introduction

The Secure Preference file which determines things such as homepage or default search engine is an important part of the Google Chrome browser. This file is located in the Chrome directory and thus it is accessible for everyone. Due to the amount of sensitive information that this file handles, Chrome implements some security mechanisms to ensure that no parties apart from Chrome can modify this file. This is done by using a custom hash-based message authentication code (HMAC) algorithm [27] which produces a 256 bit Secure Hash Algorithm (SHA-256) given a key and a message. The problem with this is that this procedure is not as secure as people might think. Malware in the wild has been known to be able to modify this file in order to either annoy the user or to generate some kind of profit from the modifications [2][15]. This type of malware can perform actions such as:

- Changing your default search engine.
- Changing your starting page or homepage.
- Add any number of pinned tabs.
- Uninstalling browser extensions.
- Disabling/enabling browser extensions.
- Allow extensions to be active during incognito mode.

By changing your search engine, information about a users search queries can be gathered and ads can be tailored to spark the user's interest. The search engine could also send you to web pages containing malicious code such as key loggers. In this way the attacker could get her hands on sensitive information such as usernames and passwords. The attacker can achieve similar things as mentioned above if the starting page of the browser is altered. The page could contain malicious code with the intention of stealing your private information or it could be equipped with a malicious search bar.

Users could be cheated in order for malicious extensions to be installed, such as toolbars with pop-up ads or other extensions that aim to lower the security of your browser. In the same way, extensions that aim to protect the user's browser experience could be uninstalled. For example AdBlock could be removed in order to expose the user to unwanted ads or pop-ups.

By changing values in the Secure Preference file the attackers could also disable or enable extensions in the browser. The effect is reminiscent of the previous example since this could be used to disable protective browser extensions such as AdBlock. It

could also be used in order to re enable malicious extensions that has been disabled by the user.

1.1 Motivation

With the expansion of the Internet and the big role it plays in today's society, browser security is a field which affects almost everyone. Over the years we have seen a steady increase in malicious software (malware) which aim to affect your computer and also your browser [24]. Because of the increasing number of malware which is circulating the Internet, browser developers need to be able to protect their users against such programs. In order to do so, information about how different kinds of malware operates on the browser is needed.

Browser hijacking is a common issue which affect browser users daily [13]. Many of these hijackers targets the preferences of a browser [15][2] which are located in the Secure Preference file. By recreating some of these hijacking attacks, a deeper knowledge about how they function is obtained and this information could be used in order to create protection mechanisms. These mechanisms can then be shared with browser developers in order to make more hijack-resistant browsers in the future.

1.2 Aims and Challenges

The main aim of this thesis is to research and explore the security of the Google Chrome browser. More specifically, if it is possible to change the preferences of the Secure Preference file without the browser noticing. The thesis will also reproduce attacks on the browser in order to get greater understanding of how these attacks are performed in hopes that it could make for a more secure browsing experience in the future. This will be made possible by implementing a script that by reproducing and replacing the HMACs of the Secure Preference file makes these attacks possible.

1.2.1 Research questions

Google Chrome is widely used around the world and millions of users depend on the built in security functions that it possesses. What if you could break one of these? In a blog post about the exploitation of the Secure Preference file [25] it is stated that malware in the wild has been known to exploit the Secure Preferences file for a while now. But is it really possible to bypass a security mechanism created by a large company such as Google?

The hashing mechanism of the Secure Preference file is implemented in order to protect the user from malicious software changing the content of the file. But how secure is the mechanism? Is there a way to change the content of Secure Preference without the user or browser noticing?

Is it possible to figure out how these types of malware that is mentioned in [25] alters the file? The creation of a script that can perform these actions might lead to a better understanding of how they work and could provide knowledge for further research about how they could be stopped.

This subject has been discussed on different forums and in a few blog posts such as [25]. To the best of our knowledge, security concerns regarding the Secure Preference file of Chrome web browser has not been the subject of any research papers; thereby motivating the work of this thesis.

1.2.2 Challenges

To the best of our knowledge, there has not been any academic research conducted on the subject of this master thesis. That is why the biggest challenge that we are facing is the lack of information. As mentioned before there exists some blog posts and threads on forums discussing the matter but that is pretty much all of it. Since there is not a lot to base the project on, most of the information has to be gathered on our own and what says that the information on these blogs is accurate? These blog posts were written a couple of years ago and Chrome is constantly being improved and updated so can we be sure that the functions that worked a couple of years ago are still viable in the most recent update of Chrome?

Chrome has some security mechanisms to detect modifications of Secure Preferences and alert the user that a third party tried to change values in the file. But when the HMAC is recreated this security metric is bypassed. The problem with the security of the Chrome Secure Preferences file has been known for some time and has been the base of multiple malicious applications [15][2]. However, this does not mean that the task of breaking the security of the file is an easy one. A common challenge that you face when working with hashing, is if one character of the input is different from the expected one the hash will become completely different and there is no way you could try to reverse it in order to find the sign that was wrong.

1.3 Limitations

Given the time constraints of this thesis, we will only look at the Secure Preferences problem in the Google Chrome web browser. We will not cover the possibility of similar problems in other web browsers since security is handled differently depending on the browser. Investigating and understanding the security mechanisms of multiple browsers would require a lot of time and effort and could possibly result in a very wide scope.

The thesis will only look into solutions on the Windows operating system families. Since Windows is the most popular family of operating system [26], it feels most relevant to focus the research on this area.

Countermeasures are not a part of the scope of this thesis. However a short section with some suggestions and tips concerning the protection against this attack is provided in Chapter 7.

1.4 Contributions

In this thesis we will show that it is possible to exploit the security of the Secure Preference file by reproducing the HMAC algorithm that Chrome executes. This will provide an insight into the way the hashing and protection of a user's settings is done in Chrome. This could help developers that are trying to create smart and safe extensions and applications for Chrome. By figuring out, documenting and recreating how malware is exploiting the Secure Preference file it is possible to create ways of protecting your browser. This thesis can act as a guide for further research in the area and motivate the work towards eliminating ways of browser hijacking. It might also alert Google about the flawed security in the Secure Preference file, which could lead to a better and more secure version of Chrome in the future.

1.5 Report Outline

This Chapter gives a brief introduction to the thesis by providing motivation, aim, challenges, contributions and limitations. Chapter 2 contains the necessary technical background related to the project. In Chapter 3 we present work that is related to this thesis together with some comparison. Chapter 4 presents the design of the script that silently modifies the Secure Preferences and the extension that runs this script. Chapter 5 breaks down the implementation of the project. In Chapter 6, an evaluation of both the HMAC creation and some common exploits is performed. The last two chapters provides a discussion and the conclusion for the thesis.

2
Background

This chapter will introduce some technical background that will be necessary for the understanding of this thesis. The chapter will start by introducing how browser settings are handled by Chrome and then give more details on how the Secure Preferences file is structured. This will be followed by a short description of browser extensions and the role they play in browser exploits. The following section will then give a description of browser hijacking and present some common exploits and the malware that makes use of these. Lastly a comparison of Chrome and Chromium will be presented.

2.1 Browser Settings

When controlling the behavior and functionality of a browser, most focus is on two different methods, namely policies and preferences. With policies, an administrator have an easy way of centrally managing the behavior of all browsers in an organization. There is, however, different user preferences that do not need to be managed centrally and preferences are used instead. Even if both of the methods control the behavior of the browser they have different purposes:

Policies
- Are rules that the browser must abide by.
- Usually applied to groups of users instead of being unique to a single user.
- Are typically not writable by the user.
- Do not keep track of a users browser experience.

Preferences
- Keep state of a users browsing experience.
- Usually unique for each user.
- Are writable by the user, since they are contained in text files on the users computer.
- Are often technical settings that do not necessarily make sense to pre-set per user or lock in with policies.

There are also several preferences that are also policies. *Homepage* is the most common such preference. If two user preferences are defined by both methods, policies will always take precedence. If *homepage* is specified by policies and in the "Preferences" file, the policy will always override. On Windows all policies are stored

in the Windows registry. In Chrome, users can view the policies on a the web page by using the URL chrome://policy/.

2.1.1 Preferences in Google Chrome

To allow for an easy customization of a web-browser, there are usually many configuration parameters that can be modified to fit the needs of the user. Setting the homepage to a site that a user frequently visits or changing the default search engine are just two changes that can be performed to make the users experience a little more pleasant. Most of these changes can be performed very easily by going into the settings page of the browser.

In the Chrome architecture, these parameters are stored in what is called preferences. Each preference is identified by a key which in turn points to the value of the preference. These values can be represented by a number of different data-types such as booleans, integers, strings, dictionaries or lists depending on the information that is stored. Preferences can be associated with either a specific browser profile or with local state. Local state contains everything that is not directly associated with a user profile but instead represent values that are associated with the host computer on which Chrome is running.

Most preferences in Chrome are stored locally in a JSON file with nested dictionaries called "Preferences". This file can be found in the folder of every Chrome profile. If there is only one profile the file can be found in the "Default" folder.

2.1.2 Secure Preferences in Google Chrome

To improve on the security regarding user preferences, Chrome has added signatures and validation on a number of preferences. These preferences are stored in a JSON file called "Secure Preferences". This file is similar to the "Preferences" file but it also contains HMACs of every entry in the file. In addition to this, the file also has an HMAC which it calls "super_mac" which is there to check the validity of all the other HMACs.

Both the "super_mac" and all the other HMAC values are validated when the browser is started. If Chrome does not detect any anomalies in the HMACs, the value of each preference stored in Secure Preferences is applied to the browser. In the event of a faulty HMAC, Chrome will reset the value of that preference to a default value. In order to alert the user of the unexpected change, Chrome will also display a pop-up window when the user enters the "settings" menu in the browser. As can be seen in Figure 2.1, this message notifies the user of the change as well as giving them the opportunity to reset all settings back to a default state.

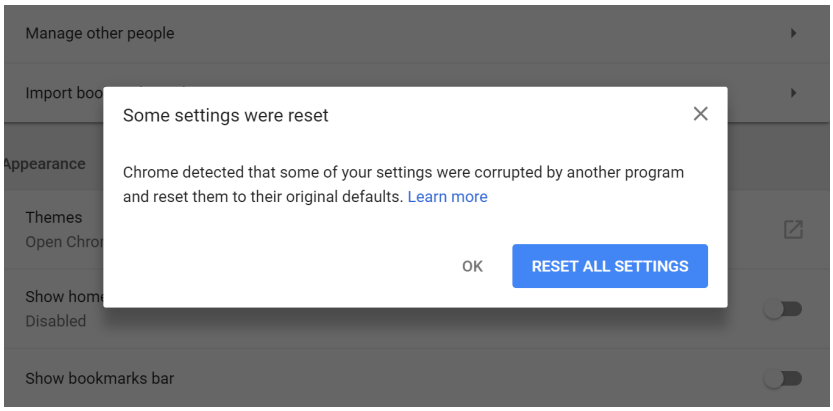

Figure 2.1: Chrome discovering a faulty change in Secure Preferences when a value is altered.

2.2 JSON

JSON [5] is a lightweight open-standard data-interchange format based on a subset of the JavaScript language, an example of an entry can be seen in Figure 2.2. It is easy for humans to read and write while also being easy for machines to parse and generate. Even if JSON is completely language independent it uses conventions that are similar to those from languages in the C-family. JSON is built on two structures:

- A collection of name/value pairs. In different languages, this is realized as an object, record, dictionary, hash table, keyed list, or associative array.
- An ordered list of values which is, in most languages, realized as an array, vector list, or sequence.

```
{
  "firstName": "Jane",
  "lastName": "Doe",
  "isAlive": true,
  "age": 30,
  "address": {
    "streetAddress": "14 3rd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10023-3300"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "123 456-789"
    }
  ],
  "children": [],
  "spouse": null
}
```

Figure 2.2: Example of a JSON entry.

2.3 SHA-256 Hashing

A cryptographic hash function is designed to make it easy for the user to compute the hash but very hard for an attacker to invert it. The hash function maps a message of arbitrary length into a fixed length and also satisfies the property for collision resistance. SHA-256 [27] belongs to the SHA-2 cryptographic family which is an improvement from the older SHA-1. SHA-2 is widely used in security applications, examples of such applications are TLS [23], SSL [23] and IPsec [16].

2.4 Browser Extensions

In order to improve users browsing experience, many major browsers support a type of software called *extensions* which are often created using HTML, JavaScript and CSS. Extensions are installed in the browser to extend its functionality and there are thousands of different extensions available [6]. Each extension has its own purpose and performs different actions depending on what that purpose is. Some extensions help to integrate the browser with other services that are used by the user such as *Mailtrack* [11] which is an extension that informs if the email which the user sent has been read or not. While others add additional features to the browser such as *LastPass* [9] which handles password management. These are just a few examples of what extensions can do and since there exists such a vast amount of extensions there is a lot of extra functionality that can be added to the browser.

Browser extensions have been the target of many attacks due to the tight relation they have to the browser environment [22]. Attackers use the extensions for malicious intent such as private information gathering, password theft or browser history retrieval. Browsers in the past had little to no security when it came to extensions but that has changed with the development and improvement of browsers [22]. As of today all browsers implement defensive countermeasures that, in theory, protects both the extensions and their resources from third party's trying to access them [22].

2.4.1 Native Messaging

An extension in Google Chrome is able to communicate with native desktop applications using a specific API called *native messaging* [4]. For this message passing to work the application needs to be registered as a *native messaging host*. This means that the application needs to have a manifest file that specifies that it has native messaging capabilities as well as which extensions that are allowed to communicate with it. Each native messaging host is started as a separate process and the only way of communicating with it is through the standard input/output pipes, *stdin* and *stdout*.

2.5 Browser Hijacking

Browser hijacking is an attack where the attacker tries to change the behavior of the web browser without the user noticing [13]. This can be done by altering the preferences of the web browser, common preference targets are for example search engine, homepage and new tab page URLs.

A common intention of the attack is either to redirect the user to certain web pages containing malicious software such as key loggers that will collect sensitive user data such as passwords, user names or email addresses [13]. Another is forcing the user to use your services generating more ad revenue and directing the user towards specific web pages that pay for the service [13].

There exists various ways for the malicious software to enter your system but the majority of the hijackers are installed by hazardous free applications, ad-supporting programs or shareware. These often include a variety of toolbars and plug-ins. These extensions will then access your preferences- and secure preference file in order to change the values of the desired targets mentioned above [13].

2.5.1 Examples of Browser Hijacking Software

Trotux [15] - Trotux.com is a fishy website that claims to be a legit Internet search engine. *Trotux* falsely claims that it generates the most relevant search results and often tricks user into believing that it is legitimate. It often enters your system through dubious installations or freeware and targets the most popular web browsers Internet Explorer, Google Chrome and Mozilla Firefox. Trotux.com is then assigned as the new tab URL, homepage and default search engine which forces the user to encounter unwanted browser redirects upon doing things like opening the browser, opening a new tab or simply trying to search the Internet. *Trotux* also makes it hard for the user to roll back these changes, during the installation *Trotux* adds a number of "helper objects" which are designed to automatically reassign settings when the user attempts to change them. Websites visited, search queries, Internet Protocol addresses and other similar information about the user's web browsing activity is recorded by *Trotux*. This information might be shared with third parties who can use the private information to generate revenue. The data tracking that is performed can lead to serious privacy issues or even identity theft.

WinYahoo [2] - This malware is not related to the legit company that is Yahoo even though it tries to fool the user with its name and it also sets Yahoo as the default search engine and homepage in the affected browsers. *WinYahoo* is bundled in with a patent installer for Adobe Photoshop Album Starer Edition which fits the trend for a lot of unwanted software. *WinYahoo* then recreates the Message Authentication Code (MAC) for the Google Chrome browser which allows it to change the Secure Preferences thus hijacking the browser. With the help of an extension called Sale Charger the user's browser is cluttered with annoying features such as new tabs with advertisement or tech support scams.

GoSave [19] - Is an add-triggering software that displays pop-up ads and unwanted advertisements on web pages that is visited. GoSave injects advertisement banners into the web pages that are being visited, turns web page text into hyperlinks, displays pop-ups with recommended fake updates or other software and it might install other unwanted adware programs without the user's knowledge. As many other browser hijackers GoSave is commonly bundeled with other freeware that can be downloaded.

2.6 Chromium

Chromium is an open-source web browser project that was created by Google in order to provide the source code for the Chrome browser. The Chromium and the Chrome browsers share the majority of code but there are some differences mainly in licensing but there are also a couple of different features.

2.6.1 Chrome vs Chromium

- *Automatic updates* - Chrome uses Google Update on Windows in order to update the browser to the latest version. Chromium has to be manually updated.
- *Crash reporting and usage tracking* - Chromium does not report crashes nor does it send usage statistics. Chrome on the other hand has both of these features, it includes general data such as Chrome settings, visited websites containing malware, information about your device and OS, search queries, etc.
- *Chrome Web Store* - Web extensions can be installed on Chrome and found in the Web Store. However the functionality to add extensions outside of the Web Store is disabled on all Windows Channels. If such extensions are to be installed it has to be added via developer mode.
- *Media support* - When it comes to media, Chromium's is a bit limited. It lacks some media codecs to support the patent-encumbered H.264 and ACC formats.
- *Adobe Flash Plugin* - Chrome supports a Pepper API version of Adobe Flash which has an automatic update function. Since this is not open source it is not supported by Chromium.
- *Sandbox support* - Both Chromium and Chrome have Sandbox support.

2.7 Silently Installing Extensions

In 2012 Google updated the Chrome browser to protect the users from silently installed extensions since these were causing more and more problems [10]. It was no longer possible to silently install extensions into Chrome using the Windows registry mechanism since this feature was widely abused by third parties. Extensions that were installed by third party programs through external extension deployment options were disabled by default and only extensions installed from the Google

Chrome Store were allowed. This however did not mean that Google managed to eliminate all the malicious extensions from Chrome. There are still a lot of malicious extensions out there and also ways to detect and protect the user from these [8].

3
Related Work

Research concerning the Secure Preferences file is very limited. However, there are a lot of papers about browser security in general as well as the security of extensions. This chapter will start off with some work on browser security and extensions and then present the work that relates the most to this thesis: Preferences.

3.1 Browser Security

In a paper by Ries et al. [21], the security of the Chrome web browser is discussed in a more general sense. They summarize how to minimize the danger posed to the users in three main cases.

- *The severity of vulnerabilities.* If the browser puts its rendering engine in a sandbox it is able to limit the damage caused by an attacker.
- *The window of vulnerability.* One of the biggest threats is users having older versions of the browser. Attackers constantly create new types of malware, which leads to browsers having to launch new updates and patches in order to protect itself. Therefore, it is very important to keep your browser up to date. One way to achieve this is providing automated updates which will lead to many users running the up to date version of the browser.
- *The frequency exposure.* When a user enters a site that is known to contain malware a warning is displayed before allowing the user to enter. This will reduce the frequency with which users are exposed to malicious content.

3.2 Browser Extensions

Gaur et al. [3] present the vulnerabilities of browser extensions are described and discussed. Browser extensions sometimes have the same privileges as the browser itself and the usage of a malicious extension might expose the browser and system resources to attacks. An attacker could spy on web applications, launch arbitrary processes and also access files from the host file system. An extension can together with already installed extensions cooperate in order to share objects and change preferences. The paper presents new ways to perform attacks via browser extensions and divides the vulnerable points in extension development into:

- Object reference sharing.
- Preference overriding.

In their paper, Rana et al. [20] studied which permissions browser extensions asks for and how these permissions can be used by an attacker in order to perform attacks against the user. They also conduct a study on ten thousand extensions where they compare how many permissions the extension asks for compared to the ones it actually uses. Lastly the paper addresses silent installation of browser extensions in both Google Chrome and Firefox. In Chrome they access the *Preferences* file in the user data directory. The file is in JSON format and by creating a fake entry in this file they are able to show that upon restart, they are able to install an extention without the browser alerting the user.

Hulk is a dynamic analysis system that detects malicious behavior in browser extensions created by Kapravelos et al. [8]. By monitoring the execution and network activity of extensions Hulk is able to detect maliciousness using both *HoneyPages* which are dynamic pages that will adapt to an extension's expectations in web page structure and content. Hulk also uses a *fuzzer* in order to drive numerous event handlers which modern extensions depends upon. Kapravelos et al. discusses a couple of exploits which is performed by extensions:

- *Extension Management* - Hulk found several extensions that completely replaces the Chrome's extension management page with a page that hinder the users ability to uninstall these extensions.
- *Code Injection* - The most commonly detected "maliciousness" within the extensions was code injection. These extensions used remotely-retrieved code either through script injections or by evoking eval.
- *Ad Manipulation* - Attackers use extensions that will replace the advertisement on websites, robbing the website owner of the revenue. These extensions can also insert banners and text advertisement into well-known websites such as Wikipedia. These extensions works as leeches which profits from other peoples content.
- *Information Theft* - These extensions has the intention to access the user's sensitive information such as: keypresses, passwords, private in-page content and authentication tokens such as cookies.
- *Online Social Network (OSN) Abuse* - These extensions typically targets Facebook in order to spam users with malicious links that can generate profit to the attacker.

3.3 Preferences

One paper discussing browser preferences is a Master thesis by Venkata Seelam [18]. This thesis explains different ways of exploiting preferences in Google Chrome and Mozilla Firefox that are usually used in browser hijacking. The thesis also proposes and describes an extension that can detect the modification of preferences in Mozilla

Firefox by third parties and inform the user. The user can then chose to discard or approve the changes. The thesis by Naga Siva Seelam puts most of its focus on Firefox since this browser is claimed to be less secure by the author. The focus is also only preferences that are very easily modified, in other words they can be seen as being stored in Preferences. This differs from our thesis since our focus is on Google Chrome and the protected preferences stored in the Secure Preferences file.

Banescu et al. [1] has researched *changeware*, which is software that modifies resources of software applications without the user noticing. Comparing this with our thesis the resource that gets modified is the Secure Preference file. The concept of changeware is to gain profit from a large amount of end users executing the malicious software. One popular example is to change the web browser settings such as default search engine and homepage using Browser hijacking malware. The paper proposes a solution which combines several protection mechanisms such as white-box cryptography, software diversity and run-time process memory integrity.

Research concerning how browser preferences can be exploited by third parties is described in a patent application by Thomas Wespel and Thomas Salomon [12]. They discuss different ways of exploiting browser preferences and then present a method and the technology for automatic classification and resetting of browser preferences. According to Wespel and Salomon, most external changes to browser preferences occur because of the huge role that browsers play in advertisement and marketing. One way in which a company can reach a lot of consumers is through the homepage and search provider of a browser. These are some preferences that a user comes into contact with very often when using the browser. This has led to a lot companies using different means to change a users browser to display the pages that will benefit that specific company.

A very common way in which companies try to modify the preferences is through slightly hidden offers presented as "opt-out". These offers are usually bundled with software that has nothing to do with browsers and if a user leaves the boxes checked, the software will modify the browser settings. The biggest problem with these modifications is that most browsers do not keep track of any history of browser preferences. This means that if a user wants to reset the preferences the only option offered by the browser is to reset to certain default values. Another way of resetting the values is to do it manually but since different browsers handle preferences differently this could be quite difficult and very time consuming. In the solution presented by Wespel and Salomon, they propose a Cloud based approach that uses black- or whitelists to determine if a value is suspicious and then recommends setting that can be used in the event of a reset.

4
Design

This chapter starts by gives an overview of the script that performs the modification of preferences created for this thesis. The implementation shows how a third party other than Chrome can make changes to certain preferences of the browser without the user being notified of the changes. The overall design of the script presented in this thesis can be seen in Figure 4.1 with more in-depth descriptions of its different phases presented later in this chapter. Lastly, the extension created for executing the script from the browser is described.

4.1 Secure Preferences

The Secure Preferences file is a file that keeps track of certain preferences as well as the signing and validation of these preferences in the Chrome browser. The preferences are read and validated when the browser is started. Chrome calculates the HMACs of every preference stored in Secure Preferences and compares them to the HMACs that are also stored in the file. In the event of two values being different, the browser will notice that modifications have been performed in the file by a third party and revert the preferences to their default values.

4.2 The Script

The attack that is implemented consists of a script that goes through four different phases in order to make changes to the preferences in the Secure Preferences file, see Figure 4.1. These phases are: 1) Initialization; 2) Modification; 3) Calculation and 4) Update. In order to modify this file without alerting the browser or user, the script also calculates the corresponding HMACs. The script we created for this thesis is written entirely in the programming language Python. The design of the different phases of the attack and their purpose are described below.

4.2.1 Initialization

The first phase that is entered when running the script is the initialization phase. It is during this phase that most of the values which are needed for later phases are gathered or calculated. These include both values that are specific to the user running the script as well as those that are supplied as arguments to the script. The purpose of these arguments are to specify which preferences to modify and the desired values. It is also here that all necessary files are located and read. This

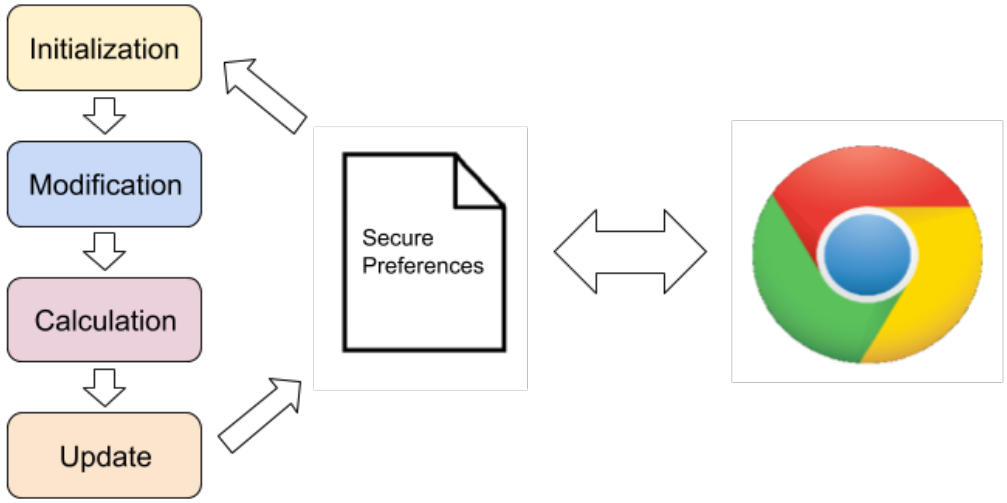

Figure 4.1: The four phases of the script and how they interact with Secure Preferences and each other

includes the Secure Preferences file, from which all preferences are read and stored for later use.

4.2.2 Modification

When the initialization phase is complete the script moves on to the modification phase. During this phase the supplied arguments are used to determine which preferences to modify as well as how. These modifications can be of three different types, namely add, remove or modify. The script then performs the desired actions on the preferences and moves on to the next phase.

4.2.3 Calculation

The third phase of the script handles all the calculations relating to the preference values and the signing and validation of preferences. It is here that the HMACs and the super_mac are calculated.

In order to calculate a corresponding HMAC to a desired preference the hashing algorithm needs to be supplied with the correct input. If the input is not correct the result will always be wrong and Chrome will be able to notice the modifications. The input that is used in the algorithm is called message and can be seen as a simple text string that has a certain structure. This message is calculated by combining information about the user running the script with information regarding the preference that is to have its HMAC calculated. All this information is computed during this phase.

After the script is done calculating the HMACs of all the desired preferences, there is only one value left to calculate. This value is the super_mac, which is the HMAC

used to validate all the other HMACs.

4.2.4 Update

When everything is modified and calculated, the script enters the final phase. Here the script makes some small modifications to the entire set of preferences before updating Secure Preferences with the new values. When this is complete the Secure Preferences file will be updated with the new preference values and their corresponding HMACs. The updated Secure Preferences file will then by read by Chrome the next time it is started and a silent modification of Chromes behavior will have been performed.

4.3 The Extension

In order to execute the script from the browser, an extension is created. The extension has as its main purpose to run the script with a certain set of arguments and thereby modifying Chrome, see Figure 4.2. For the purpose of this thesis the look of the extension is not important, as it only serves as a proof of concept. The extension therefore consists of only a button that when pressed executes the script. For the extension to to run the script it has to be located on the users computer. If this is the case the extension makes use of a Chrome API that allows it to run the script and alter the Secure Preferences file.

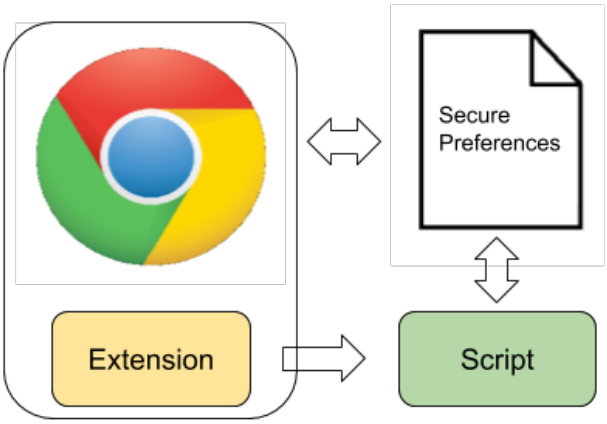

Figure 4.2: Overview of the extension calling the script

4.4 Design Discussion

When deciding on the which programming language to use for the creation of the script, the choice fell on Python. Python supports a lot of different tools and libraries to help with a multitude of tasks such as hashing, JSON operations and process calls. Since the script created for this attack needs to find values that are specific to the computer on which Chrome is running there would be a need for certain system

calls. Because of this, languages such as C or C++ could also be used for such calls. However, Python is able to perform and find all these value on its own so using different languages would only complicate the implementation of the script.

5
Implementation

The following chapter will present the implementation of the script created in this thesis. The chapter starts off by giving a full description of how the analysis of Chromium is performed and what this information is used for. It will then proceed to give detailed descriptions on how to acquire or calculate all the necessary values needed to calculate the HMACs. The chapter ends by giving a full description of all the steps performed by the script in order to silently modify the preferences in Chrome.

5.1 Analyzing Chromium

Manually changing different Chrome preferences that are stored in the "Preferences" file is a very easy task. The file is a simple text file with a JSON markup so a preference can be altered by replacing the old value with a new. When it comes to the preferences stored in "Secure Preferences" this is not the case. In this file every key/value pair is signed with a custom HMAC so every change in values will result in different HMACs. This makes the alteration of preferences much harder since the HMAC belonging to the altered preferences also has to be calculated.

In order for us to reproduce the HMACs used in Secure Preferences, knowledge about the exact way that Chrome performs its hashing is required. This information is hidden deep within the source code of the browser. Since the source code of Chrome is not open source this information is hard to find. Luckily, Chromium, which is the project behind the Chrome browser, is open source. The source code of Chromium contains most of the functions that also seem to exist in Chrome, we use Chromium version 67.0.3369.0. When looking for information about the HMAC calculations we use two different methods:

- Exploring the source code and searching for functions relating to HMAC calculations.
- Debugging the code using breakpoints.

5.1.1 Exploring the Source Code

The source code of Chromium contains a lot of files and functions but there are some functions that are a little more interesting than others when it comes to HMAC calculations. These functions are located in a file called pref_hash_calculator.cc and

handles most of the calculations needed.

```
// Calculates an HMAC of |message| using |key|, encoded as a hexadecimal string.
std::string GetDigestString(const std::string& key,
                            const std::string& message) {
  crypto::HMAC hmac(crypto::HMAC::SHA256);
  std::vector<uint8_t> digest(hmac.DigestLength());
  if (!hmac.Init(key) || !hmac.Sign(message, &digest[0], digest.size())) {
    NOTREACHED();
    return std::string();
  }
  return base::HexEncode(&digest[0], digest.size());
}
```

Figure 5.1: Chromium function GetDigestString which calculates an HMAC given a message and a key.

When searching the source code of Chromium for interesting functions, we discover that the function responsible for calculating the HMACs is called GetDigestString. As can be seen in Figure 5.1 the hashing algorithm used for the HMAC calculation is a SHA256 algorithm that takes a message and a key as input and returns a hexadecimal string.

Another important function that is also located in this file is the function GetMessage. The purpose of this function is to create the message that is used in the HMAC calculations. It does this by concatenating three values; *device_id*, *path* and *value_as_string*. These values and how we are able to obtain them will be explained further in section 5.2.2.

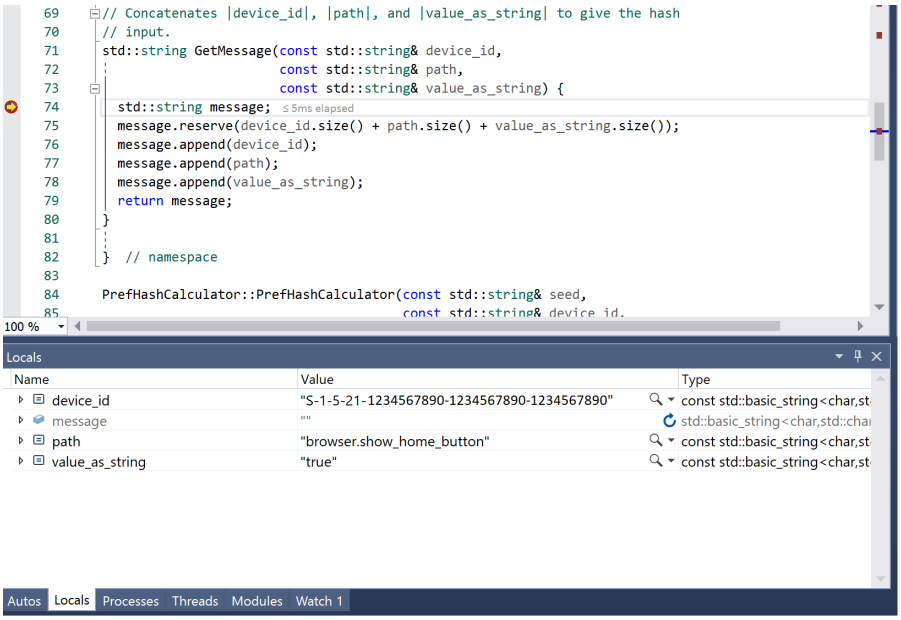

Figure 5.2: Debugging of GetMessage for the preference show_home_button.

5.1.2 Debugging the Source Code

While the source code can tell a lot about the different values that are used in the calculations, their exact structure is hard to figure out without actually seeing the values. This is where debugging is a very helpful tool. To debug the code of Chromium, we download the entire browser source code and install it. The code is then debugged using Microsoft Visual Studio. By putting breakpoints on critical functions such as GetMessage we are able to figure out the exact structure of many necessary values as illustrated in Figure 5.2.

5.2 Reproducing the HMAC

This section will describe in detail how our implementation is able to find or calculate the values necessary for correctly reproducing the HMAC used by Chrome to validate the preferences.

5.2.1 Finding the Seed

As mentioned in Section 5.1.1 the SHA256 algorithm used by Chromium needs a key to accurately calculate the HMAC. This key is usually called a seed and can be found in a data pack file called "resources.pak", which is located in the installation path of Chrome. This file contains packed binary data which makes the content very hard to read and understand without knowing the exact format of the file.

In order to figure out the format, we start by looking at the code and tools of Chromium. Since the browser is able to read this file there is most likely a tool that is able to perform this action. In Chromium there exists a tool called GRIT which is the internationalization tool used by all the Chromium projects. This tool contains a python script called "data_pack.py". The purpose of this script is to provide support for formatting data pack files and therefore contains the necessary information on the format of these files.

By looking at the code of this script we find that the first part of the file consists of a header that contains information about the file. The first value in this header is a 4 byte long version number, which is the current version of the data pack file. The version number is important since this value determines the structure of the header and its values.

By reading and unpacking the binary data into values that can easily be understood, we find that the version used for this file is version 5. This version uses a header with a length of 12 bytes and its structure is illustrated in Figure 5.3. The second value in the header that is important for finding the seed is the number of resources in the file. This 2 bytes long value is used to determine where the resource information ends and the actual resources begin.

Figure 5.3: Structure of the header information in resources.pak.

The values that are located after the header are information about the resources in the file. As can be seen in figure 5.4 this information is made up of a resource ID and the location of the resource, stated as an offset from the start of the file.

Figure 5.4: Structure of resource information in resources.pak.

The seed used in Chrome is a 64 character long hexadecimal value. In order to find the exact seed which is used by Chrome we combine the information about the number of resources and their offsets to determine the length of each resource. We then locate the seed by finding the first resource that has a length of 64.

5.2.2 Creating the Message

As mentioned in Section 5.1.1 the message used in the HMAC calculation of Chromium is created by the function GetMessage, which performs this task by concatenating three values. These values are sent to the function as string arguments and the resulting value is also a string. In order for us to recreate the message we have to know the exact structure of all the strings used in this calculation. We discover what these strings look like by debugging the code and putting breakpoints around GetMessage, as can be seen in Figure 5.2.

By evaluating the value of *device_id* we discover that this value remains the same during all HMAC calculations. A deeper investigation into the value leads us to its origin. The value used as *device_id* in the calculation is the same as the Security IDentifier (SID) of the computer on which Chrome is running. In order to acquire the SID of a user we make use of the Windows infrastructure for management data and operations called Windows Management Instrumentation (WMI) [17]. By creating a subprocess that runs the command-line utility of WMI we are able to send a query for the SID of the specific user running the script. The result of this query will include the Relative ID (RID) part of the users SID so the RID is removed to get the correct value of *device_id*.

The second value that is used for the message calculation is *path*. This value consists of the JSON keys that are traversed in Secure Preferences to reach the preference

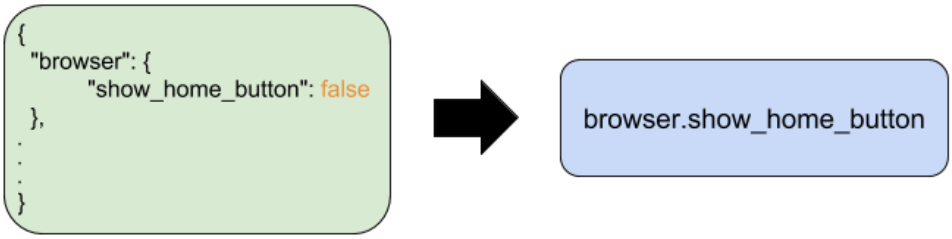

Figure 5.5: A JSON object and its corresponding path used in the message creation for the show_home_button preference.

for which the HMAC is to be calculated. As can be seen in figure 5.5 the path used in the calculation uses the (.) notation to separate the JSON keys that are traversed. Since we know the exact structure of Secure Preferences the path for a specific preference is very easily constructed.

Value_as_string is the third and last value that is used in the message creation. The content of this string is the value of the preference for which the HMAC calculation will be performed. Since preference values can be of many different data-types the exact structure of *value_as_string* will differ between preferences. By looking at the message creation of different data-types we find two modifications that are performed on the preference values before they are used as *value_as_string*. To correctly create a string to be used in the message creation we first find the value of the desired preference and then remove all of its empty arrays and objects. We then replace all "<" characters with their unicode representation, "\u003C".

When all the values have been collected or created we combine them to create our own message that is used in the HMAC calculation. By taking the values presented in figure 5.2 the resulting message would be:

$S-1-5-21-1234567890-1234567890-1234567890browser.show\_home\_buttontrue$

5.2.3 Calculating HMAC

Having both the seed and a the messages created for a set of certain preference we are able to correctly calculate their corresponding HMACs. We perform these calculations by using a SHA256 HMAC function which takes the seed and message as input. The result of this function is a 32 bytes hexadecimal value which is used to validate the preference.

5.2.4 Calculating super_mac

The final step in correctly modifying the preferences in Secure Preferences is calculating the HMAC called *super_mac*, which is there to validate all the other HMACs. The *super_mac* is calculated in the same way as all other HMACs with some slight differences in the values used. Since the seed is used as the key for all calculations

this value remains the same. This is also true for *device_id* since the SID of the computer will not change. The differences appear in path and *value_as_string*. The value of path is an empty string instead of the path to where in Secure Preferences the *the super_mac* is located. The value of *value_as_string* is a JSON entry called macs which contains all of the HMACs.

5.3 Script Implementation

The script we implement in this thesis is a versatile python script that when supplied with the desired preferences and their values can silently change the behavior of Chrome. In order to correctly modify the preferences in Secure Preferences the script only needs to be given arguments in the form of the preferences that are to be modified. All other values are automatically gathered or calculated by the script. The step by step process performed by the script when supplied with the desired preferences is described below. A graphical illustration of this process can be found in Figure 5.6.

1. Locates the Secure Preferences file and reads all the preferences stored in the file.
2. Locates the resources.pak file and uses it to acquire the seed.
3. Modifies the preferences that are specified in the arguments with the desired values.
4. Calculates the message of all the modified preferences.
5. Calculates the HMAC of all modified preferences.
6. Calculates the super_mac to validate all the other HMACs.
7. Updates the Secure Preferences file with the new preferences.

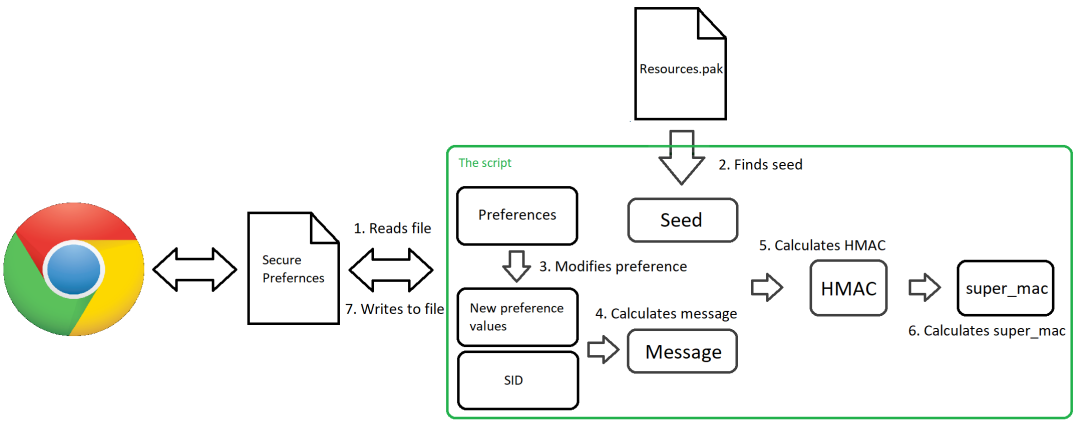

Figure 5.6: A detailed description of the step by step process performed by the script when modifying a preference.

5.4 Extension Implementation

In order to show how the script can be executed from the browser it is trying to hijack, we also create a proof of concept extension which performs this action. The extension in itself is a button that when clicked executes the script created for this thesis and thereby changing the behavior of Chrome. For the extension to be able to do this it makes use of the native messaging API that is available in Chrome. This API allows the extension to communicate with applications that are considered native messaging hosts. In this thesis, the native messaging host consists of the script together with a Windows batch file as illustated in Figure 5.7. The batch file sets the desired preferences as arguments and then runs the script.

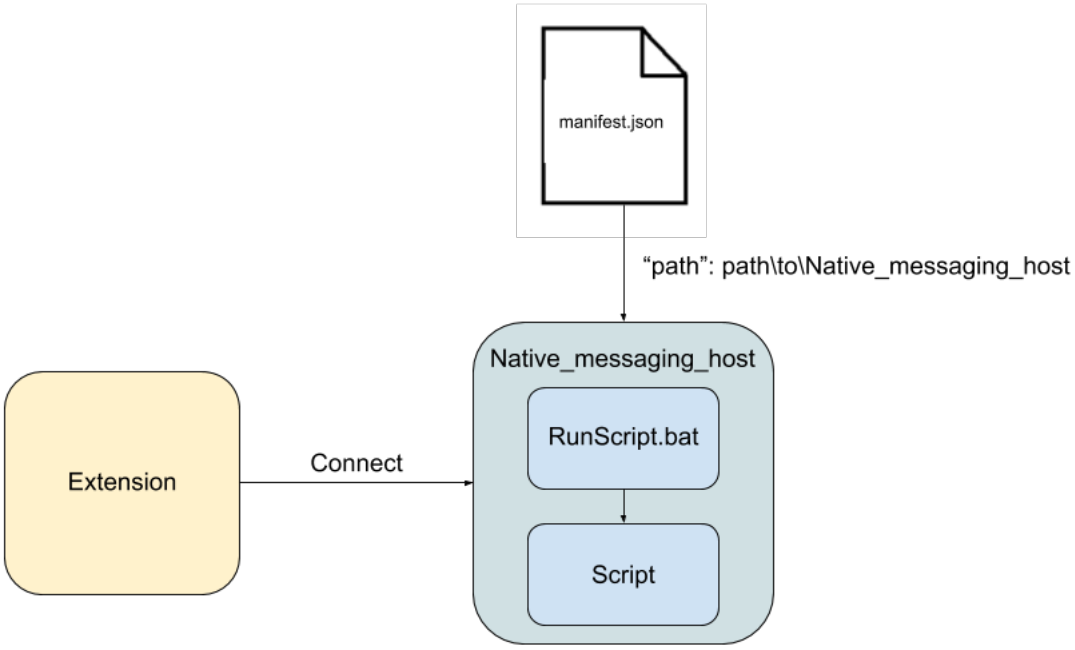

Figure 5.7: An overview of an extension connecting to the native messaging host.

5.4.1 Setting up a Native Messaging Host

An extension in Chrome can only communicate with applications that have a native messaging host. To be able to be a host there are certain requirements that have to be fulfilled by the application:

- The application needs to have a manifest file with certain values that specifies the configuration, see Figure 5.8. The location of this manifest file depends on the platform. On Windows this location can be anywhere in the file system.
- The application also has to be added as a Windows registry key *HKEY_CURRENT_USER\SOFTWARE\Google\Chrome\NativeMessaging Hosts\name_of_host*
  or

HKEY_LOCAL_MACHINE\SOFTWARE\Google\Chrome\NativeMessaging Hosts\name_of_host with a default value of the full path to the manifest file.

```
{
  "name": "calchmac.py",
  "description": "Script Host",
  "path": "runScript.bat",
  "type": "stdio",
  "allowed_origins": [
    "chrome-extension://mjglhlojpnlokfhmbnkbggnnnheaogna/"
  ]
}
```

Figure 5.8: The manifest file used for the configuration of the host.

6
Evaluation

The chapter will describe the evaluation performed to show that the script created in this thesis work as intended. The evaluation was performed in two steps with the first evaluation being performed on only the HMAC calculation part of the script and the second evaluation on the script as a whole when used to modify preferences.

6.1 HMAC Calculation

In order to silently alter the behavior of Chrome through the modification of preferences the most important part of the implementation is the HMAC calculation. Without a correct way of reproducing the HMACs created by Chrome the implementation is never going to be successful and Chrome will be able to detect that something has been altered every time the script is executed.

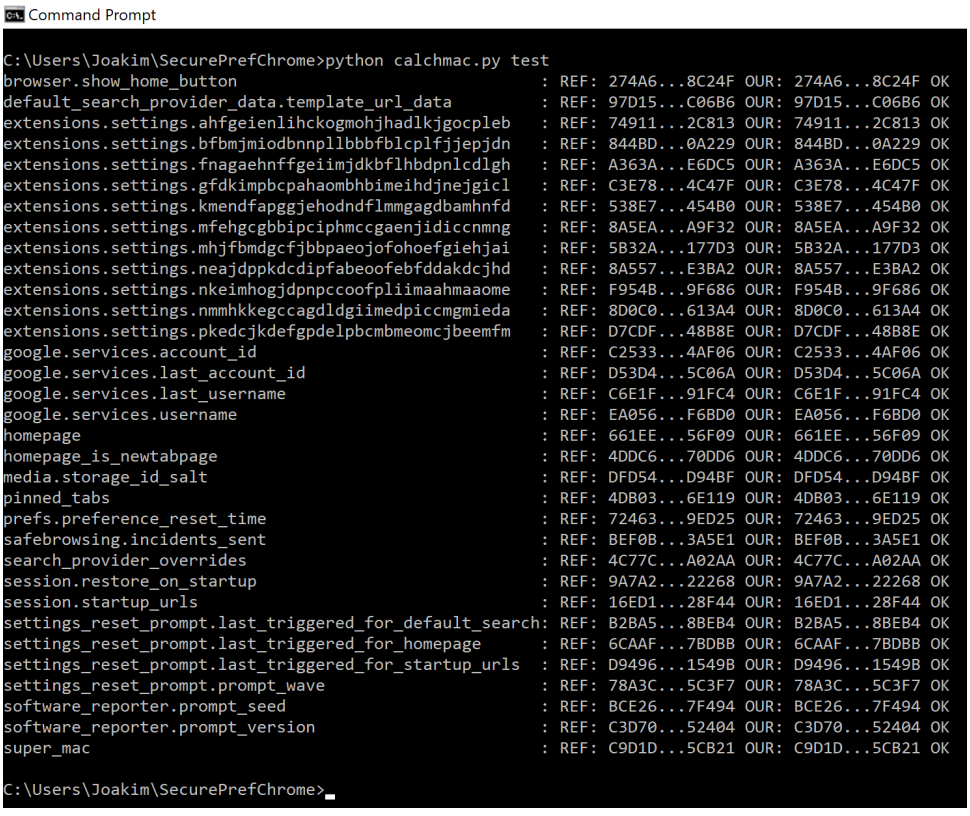

Figure 6.1: Results from the test of the HMAC calculations.

The HMAC calculation is evaluated by reproducing the HMAC of all preferences stored in Secure Preferences. Doing this allows us to test the calculation of HMACs belonging to multiple values of different data-types. This test does not modify the preference before calculating the HMAC to be able to compare the result with the HMAC created by Chrome. In other words, this test is only used to test the HMAC calculation part of the implementation and not the functionality of the entire script.

After all the reproduced HMACs have been calculated they are compared with the HMACs that are created by Chrome and stored in the Secure Preferences file. The test of an HMAC is considered successful if the two HMACs are identical.

As can be seen in Figure 6.1, the result of the evaluation is very positive with the implementation being able to correctly calculate the HMACs of all preferences as well as the super_mac. This gives our implementation of the HMAC calculation a success rate of 100% in this test.

6.1.1 Discussion

The test is performed by taking all the HMACs that are stored in the Secure Preferences file and trying to reproduce them. There are cases where there exists HMACs for preferences that are not present in the file. In the event of a preference having an HMAC but not a preference value, that value is represented by an empty string. Using an empty string will give a correct result most of the time but there are situations when this is not the case. For preferences where the value is an array of different values, the value would instead be an empty array. It is however hard to know the structure of the value if the preference is not present in the Secure Preferences file.

Because of this, the result of this test will differ depending on the Secure Preferences file that is used for the test. Even if this might result in some HMAC calculations being wrong when running this test, it will not have any impact on the overall functionality of the script. When running the script to modify a preference, only the HMAC for that preference is calculated which means that the preference will always be stored in the file and have a value.

6.2 Implementing Common Hijacking Exploits

The second evaluation performed is an evaluation of the entire functionality of the script. In other words, this test aims to show how well the script is able to successfully alter certain preferences without alerting the user or browser.

The test was performed by implementing a number of common exploits and documenting the behavior of Chrome before and after the implementation. For this evaluation we used both exploits that are well known to be used by malware in the wild such as changing the startup page and default search engine as well as exploits

Exploits	Path	Value
Toggle visibility of the home button	browser.show_home_button	True/False
Set the default homepage	homepage homepage_is_newtabpage	http://www.example.com True/False
Add pinned tabs	pinned_tabs	Array[http://www.example1.com]
Disabling extension	extensions.settings.extensionID.disable_reasons extensions.settings.extensionID.last_activated_ime_engine extensions.settings.extensionID.state	1 False 0
Enabling extension	extensions.settings.extensionID.disable_reasons (remove) extensions.settings.extensionID.last_activated_ime_engine extensions.settings.extensionID.state	None False 1
Removing extension	extensions.settings.extensionID (remove)	None
Allowing extension in incognito mode	extensions.settings.extension.incognito	True/False
Startup page	session.restore_on_startup session.startup_urls	4 http://www.example.com
Default search engine	default_search_provider_data	list{search provider data}

Table 6.1: The test cases together with the path and value of the preferences that are to be modified.

that to the best of our knowledge are new and creative such as enabling or disabling extensions.

In order to make sure that Chrome does not interfere with the outcome, all tests are performed when Chrome is not running. After running the script, the browser is started in order to validate the results. In order for a test to be considered successful the intended preferences and their HMACs should be modified and alter the behavior of the browser without Chrome noticing and alerting the user.

The different test cases that were used during this test can be found in Table 6.1 with more in depth descriptions of each exploit being presented in Sections 6.2.1-6.2.7.

6.2.1 Toggle visibility of the home button

In the default setting of Chrome the Secure Preference file lacks the field "show_home_button" so the first step is to check if it exists. If it does not exist it is added and activated by setting its value to True. It is then sorted to in order to end up in the correct place. If it already exists it is just a matter of choosing whether to set it to True or False.

6.2.2 Set the default homepage

In order to set a default homepage the two fields in the Secure Preference that have to be changed is "homepage" and "homepage_is_newtabpage". The "homepage" field is the url of the website and "homepage_is_newtabpage" simply decides if your homepage also should be the default url for when a new tab is opened in the browser. If it does not exist in it is added and sorted in order to end up in the correct spot of the Secure Preference file.

6.2.3 Add pinned tabs

When adding pinned tabs to the browser the field that has to be changed is conveniently called "pinned_tabs". It consists of an array with the urls to the websites

that should be pinned upon starting the browser. If the browser does not have any previous pinned tabs this field has to be added and sorted in order to end up in the correct spot.

6.2.4 Disabling, enabling and removing an extension

These three functions are performed in the "extension" dictionary inside Secure Preference. Here, all the extensions installed on the browser can be accessed and altered. When disabling an extension the three fields "disable_reasons", "last_activated_ime _engine" and "state" have to be added. The "disable_reasons" is set to 1, "last _activated_ime_engine" is set to False and "state" is set to 0. When enabled again you simply remove the "disable_reasons" field entirely and set the "state" field to 1. In the matter of removing an extension completely two things needs to be done. First you need to remove the extension from the "extension" dictionary and then the HMAC belonging to it from the dictionary "protection".

6.2.5 Allowing an extension in incognito mode

By changing the field "incognito" inside of an extension to True, that extension is allowed to still be active when the user is browsing in incognito mode in Chrome.

6.2.6 Startup page

Two fields have to be altered here as well, "restore_on_startup" and "startup_urls". They are located in the "session" dictionary where if you want a specific website to be your startup page you need to set "restore_on_startup" to 4 and "startup_urls" to your desired url.

6.2.7 Default search engine

The field that has to be added and altered in order for this to function is called "default_search_provider_data". A list is added to the field containing the data for the chosen search provider. The dict is then sorted in order for the field to end up in the correct position.

7
Discussion

This chapter will start off by explaining how the possible exploits mentioned in this thesis can be used by an attacker and what impact that can have on the user. Then we discuss what was the inspiration of the thesis and how our work has evolved from a couple of blog posts. Future work is then discussed as well as countermeasures, ethics and sustainability.

7.1 Possible Exploits

The attack makes it possible to alter every field of the Secure Preference file, seen below we discuss some interesting exploits and how an attacker can use these for malicious intent.

7.1.1 Toggle visibility of the home button

This is more like a proof of concept than an actual attack, by recreating the HMAC we are able to either have the value of the home button as True or False. This single exploit is pretty harmless in the hands of an attacker but the fact that the HMAC has been recreated poses a big threat towards the user. With the home button activated the attacker could together with setting the default homepage of the browser send the user to a web page with malicious intent.

7.1.2 Set the default homepage

An attacker could set the default homepage to a malicious one and lure the user into using it, for example it could look like that user's online banking page. Upon using the malicious banking page the attacker can obtain sensitive information and even gain access to the user's bank accounts. The attacker could also set the default homepage any advertisement web page or other pages with financial gain as intention. If the page contains advertisement revenue would be generated, the site could also contain fake offers with the intention to trick to user into making a purchase.

7.1.3 Add pinned tabs

The previously mentioned attack can also be applied when adding pinned tabs, an attacker can fill up the browser with ads and other malicious web sites. The attacker could also fill up the browser completely with a huge number of pinned tabs which overwhelms the browser and consumes a lot RAM from the computer.

7.1.4 Disabling, enabling and removing an extension

Disabling/removing extensions can be used by an attacker in order to weaken the security of the browser. By disabling/removing an extension such as AdBlock the user will be exposed to unwanted advertisement and pop ups. Other extensions preventing monitoring of the browser activity can also be disabled/removed. If the user disables an unwanted extension the attacker can enable it again. Even if extensions have to go through the Chrome Web Store there still exists a lot of malicious extensions. These can have different malicious intentions such as code injection, ad manipulation, affiliate fraud, information theft and OSN (Online Social Network) abuse [8].

7.1.5 Allowing an extension in incognito mode

An attacker could abuse this in order to still monitor the activity of a user through an extension even when the user clearly does not want that since it is using the incognito mode which is designed to not create any browser history. The icon of the extension is however still visible which means that if the user is aware he/she can disable the extension but most users will not bother to make sure that no extension is active in incognito mode which plays in favor for the attacker.

7.1.6 Startup page

Changing of the startup page or pages can be used by an attacker in the same way as setting the default home page and adding pinned tabs which is discussed in the sections above.

7.1.7 Default search engine

An attacker could replace the default search engine with a malicious one such as *Trotux* [15] which will display popup ads, hijack the browser, infect the user's desktop shortcuts and insert ads to the web pages. Changing the search engine can be very profitable for an attacker since the user's search results can be tailored in such a way that she is sent to websites that can generate ad revenue or malicious websites with the intention to steal the user's personal information such as credit card information etc.

7.2 HMAC Creation

As we mentioned before in this report, information about how malware in the wild such as *WinYahoo* [2] that can alter values of the Secure Preference file is very limited. They main inspiration for the attack is the blog post which a user named Tigzy [25] has posted on the *www.adlice.com* web site. In this post Tigzy gives a somewhat detailed description of how how these malware performs the recreation of the HMAC and at the end Tigzy suggests a scan that he/she claims deals with this

problem.

In the references of Tigzy's blog post we also found another interesting blog post [7] by the user named Kaimi on the web site *kaimi.io* which also addresses the HMAC creation in Secure Preferences. This post is in Russian so some serious Google Translating had to be performed. Kaimi goes into more details of how the HMAC creation is performed and also provides some code which gave a better insight into what exact steps that has to be performed in order to get the attack to function properly.

A third post that touches on the Secure Preference attack is one by an author at Malewarebytes LABS [14] named Joshua Cannell. He does not go into any details regarding how the attack is performed, instead he gives an example of malware that is out there and can alter your Secure Preference file. The malware he talks about is *WinYahoo* and has some of the same abilities as our attack such as changing the default home and start page it can also change the default search provider.

With the information provided by these blog posts we started to build our script but we realized pretty quickly that the way the HMAC was created was not fully viable anymore and adjustments had to be made in order to get it to work. The first thing we encountered that was different was the seed, in Tigzy's post it states: "The seed is unique to a machine". But when comparing the seed of different machines we found out that the value of the seed was the same on every machine. This poses a threat to the security of Secure Preferences since the attacker does not need to go the extra mile and obtain the seed if it always is the same. However in our attack we made sure to fetch the seed in the proper way from the file "resources.pak" which leaves no room for error.

When creating the HMAC, Tigzy uses the following steps:

1. Obtain machine SID.
2. Obtain volume ID.
3. Create machine ID.
4. Obtain the message.
5. Create the HMAC.
6. Replace the old HMAC with the new.

But when using Chromium to debug we noticed that all of these values was not necessary or outdated. The HMAC of the current version of Chrome is created using three values namely: *device_id*, *path* and *Value_as_string* the process and values are described in detail in the Implementation chapter.

After debugging and researching in order to fully understand the way which HMAC was created in the Secure Preference file we managed to create an attack that not only manages to recreate the HMAC but also replace it in order to perform numerous

exploits in Chrome. Our attack is fully dynamic both in the way which we gather the necessary data and in the way that every field in the Secure Preference file can be altered to the users liking. It was known that exploits like changing the start page, home page and default search engine was possible. But in our thesis we prove that there is so much more that can be altered, for example extensions both in regular and incognito mode. To the best of our knowledge this thesis is the only one giving a complete and detailed look on the way which Secure Preference file works and which ways it can be attacked and exploited.

7.3 Future Work

The work presented in this thesis will hopefully raise awareness towards the flaws in security concerning the Secure Preference file which is being exploited by malware in the wild. It could provide information that helps eliminating malware such as *Trotux* [15] and *WinYahoo* [2] which targets users browsers by changing settings like start page and default search engine.

This thesis has provided an insight into a number of exploits which has been tested and explained. But due to the lack of information regarding the Secure Preference file we have not been able to cover all possible exploits. There are a lot of fields that are not visible by default in Secure Preferences and research regarding what possible exploits can be performed is something we see a lot of potential in.

The intention of the thesis is not to provide a guide that can enable attackers to be able to exploit the Secure Preferences file. The intention is that by being able to reproduce the HMAC and show that Secure Preferences does not really live up to the "Secure" in its name and thereby show the developers that some changes to the file and HMAC procedure is in order. Therefore Google is being alerted and given the chance to fix this issue before the publication of this thesis.

7.4 Countermeasures

We find it hard to completely erase the attack since as long as the Secure Preference file is accessible to the user, with the right information, the attacker will probably be able to alter the file and carry out the exploits. One can however make it more difficult for the attacker with for example the handling of the seed for the HMAC. We found out that the seed is not a random string, it is in fact the same for every computer we tested our script on. If the seed were random and unique for each individual machine it could complicate things for an attacker who expects that the seed should be the same. This would not stop our attack since we retrieve the seed from the file "resources.pak" where it is located. If this retrieval was to be negated the seed's location in "resources.pak would have to be random as well.

Venkata Naga Seelam Siva has created a proof of concept add-on which can detect unexpected changes of the user's preferences and reverts them to their old values

[18]. This only works when the browser is running but if such an add-on is created for Chrome it would be very usefull for users targeted by preference-changing malware.

Since it might be hard to find the ultimate countermeasure to the attack it might be a good idea to instead focus on protecting your computer against the intrusion of these types of malware that targets the preferences of a browser. A common theme is that the malware enters through the downloading of free software from suspicious web sites. Being careful and selective of what software you download will help you against these types of malware and also malware in general.

7.5 Ethics and Sustainability

In order to show that the preferences in Secure Preferences are not as secure as it seems, a script that reproduces the actions of many different kinds of malware have been implemented. Because of this, this thesis can be used as a guide for creating more malware. This was not the intention of this thesis, as we only set out to prove that the modification of Secure Preferences is possible and not in any way wanted to help people with ill intent to exploit others.

Because of this risk and since this thesis concerns the security of one of the most used browsers in the world this company is alerted and given time to update the browser before our finding are made public, this will hopefully prevent the scenario where our script can be used as an real world attack. We do not aim to frame a company in order to make it look bad, the aim of this thesis is simply to show and explain that the security of the Secure Preference file is flawed and that is being abused by malware in the wild. Our hopes is that the security becomes flawless and malware which is exploiting this file are disarmed for good.

8

Conclusion

Google Chrome's Secure Preferences has proven not to be as secure as one might think. In this thesis we explore different ways of abusing the preferences stored in Secure Preferences and silently modify the behavior of Chrome. In chapter 4 we present the design of a script that is able to successfully recreate and replace the HMACs for the different preferences in the Secure Preferences file. With the help of this script we show how different attacks can be performed in real life scenarios. We also give a detailed description of the implementation of the script in chapter 5. In this chapter we present the different parts of the script and how all necessary values are obtained. With these values we show how to recreate the HMACs and by doing so allowing us to modify Chrome without alerting the user or browser. To show how attacks could be performed from the browser we also create an extension that is able to execute the script and perform attacks. We then evaluate the correctness and functionality of the implementations by testing both the HMAC calculation as well as how well the script is able to reproduce common exploits. These tests show that the script is able to successfully perform all the common exploits that were tested. This proves that the security mechanisms of Chrome preferences are fairly easy to bypass and can therefore be exploited by attackers.

Bibliography

[1] Sebastian Banescu et al. "Software-Based Protection against Changeware". In: *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy - CODASPY '15* (2015), pp. 231–242. DOI: 10.1145/2699026.2699099.

[2] Joshua Cannell. *"WinYahoo" PUP Modifies Chrome Secure Preferences*. 2015. URL: https://blog.malwarebytes.com/threat-analysis/2015/05/winyahoo-pup-modifies-chrome-secure-preferences/ (visited on 05/23/2018).

[3] Hiten Choudhury, Basav Roychoudhury, and Dilip Kr Saikia. "Security extension for relaxed trust requirement in non3GPP access to the EPS". In: *International Journal of Network Security* 18.6 (2016), pp. 1041–1053. ISSN: 18163548. DOI: 10.1007/978-3-319-13841-1.

[4] Chrome. *Native Messaging*. URL: https://developer.chrome.com/extensions/nativeMessaging (visited on 06/05/2018).

[5] Douglas Crockford. *Official JSON website*. URL: http://www.json.org/ (visited on 05/08/2018).

[6] Google. *Google Chrome Web Store*. URL: https://chrome.google.com/webstore/category/extensions?hl=sv (visited on 06/05/2018).

[7] Kaimi. *Google Chrome and Secure Preferences*. 2015. URL: https://kaimi.io/2015/04/google-chrome-and-secure-preferences/ (visited on 05/23/2018).

[8] Alexandros Kapravelos, Chris Grier, and Neha Chachra. "Hulk: eliciting malicious behavior in browser extensions". In: *Proceedings of the 23rd ...* (2014), pp. 641–654. DOI: 10.13140/2.1.1324.2249.

[9] *LastPass: Free Password Manager*. URL: https://chrome.google.com/webstore/detail/lastpass-free-password-ma/hdokiejnpimakedhajhdlcegeplioahd?hl=sv (visited on 06/05/2018).

[10] Peter Ludwig. *No more silent extension installs*. 2012. URL: https://blog.chromium.org/2012/12/no-more-silent-extension-installs.html (visited on 04/26/2018).

[11] *Mailtrack for Gmail & Inbox: Emailtracking.* URL: https://chrome.google.com/webstore/detail/email-tracking-for-gmail/ndnaehgpjlnokgebbaldlmgkapkpjkkb?hl=sv (visited on 06/05/2018).

[12] Masahiro MAKINO. "(12) Patent Application Publication (10) Pub . No .: US 2006 / 0222585 A1 Figure 1". In: 002.15 (2017), p. 7. ISSN: 13871811. DOI: 10.1037/t24245-000. arXiv: 0403007 [arXiv:physics].

[13] Malwarebytes. *Malewarebytes: Browser Hijacking.* URL: https://blog.malwarebytes.com/threats/browser-hijacker/.

[14] *Malwarebytes LABS.* 2018. URL: https://blog.malwarebytes.com/ (visited on 05/23/2018).

[15] Tomas Meskauskas. *How to eliminate browser redirects to trotux.com?* 2017. URL: https://www.pcrisk.com/removal-guides/10142-trotux-com-redirect (visited on 05/28/2018).

[16] Microsoft. *IP Security (IPSec).* URL: https://technet.microsoft.com/en-us/library/cc179879.aspx (visited on 06/04/2018).

[17] Microsoft. *Windows Management Instrumentation.* 2018. URL: https://msdn.microsoft.com/en-us/library/aa394582(v=vs.85).aspx (visited on 05/30/2018).

[18] Venkata Naga Seelam Siva. "Settings Protection Add-on : A User-Interactive Browser Extension to Prevent the Exploitation of Preferences settings protection add-on : a user-interactive browser extension to prevent the". PhD thesis. 2017.

[19] Stelian Pilici. *How to remove "Ads by GoSave" adware (Virus Removal Guide).* 2017. URL: https://malwaretips.com/blogs/remove-gosave-virus/ (visited on 05/29/2018).

[20] Abhay Rana and Rushil Nagda. "A Security Analysis of Browser Extensions". In: *CoRR* abs/1403.3235 (2014). arXiv: 1403.3235.

[21] Charles Reis, Adam Barth, and Carlos Pizano. "Browser Security: Lessons from Google Chrome". In: *Queue* 7.5 (2009), p. 3. ISSN: 00010782. DOI: 10.1145/1536616.1536634.

[22] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. "Extension breakdown: Security analysis of browsers extension resources control policies". In: *26th USENIX Security Symposium (USENIX Security 17)* (2017), pp. 679–694.

[23] Symantec. *What is SSL, TLS and HTTPS?* URL: https://www.websecurity.symantec.com/security-topics/what-is-ssl-tls-https (visited on 06/04/2018).

[24] AV-TEST - The Independent IT-Security Institute. *Malware.* 2018. URL: https://www.av-test.org/en/statistics/malware/ (visited on 06/19/2018).

[25] Tigzy. *Google Chrome: Bypassing Secure Preferences*. https://www.adlice.com/google-chrome-secure-preferences/ [Accessed: 2017-12-19]. 2016. URL: https://www.adlice.com/google-chrome-secure-preferences/.

[26] W3schools. *OS Platform Statistics*. 2018. URL: https://www.w3schools.com/browsers/browsers%7B%5C_%7Dos.asp (visited on 05/28/2018).

[27] Wikipedia. *Hash-based message authentication code*. https://en.wikipedia.org/wiki/Hash-based_message_authentication_code [Accessed: 2017-12-19]. URL: https://en.wikipedia.org/wiki/Hash-based_message_authentication_code.