2017

Settings Protection Add-on: A User-Interactive Browser Extension to Prevent the Exploitation of Preferences

Venkata Naga Siva Seelam
Wright State University

SETTINGS PROTECTION ADD-ON: A USER-INTERACTIVE BROWSER EXTENSION TO PREVENT THE EXPLOITATION OF PREFERENCES

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

VENKATA NAGA SIVA SEELAM
B.Tech., V. R. Siddhartha Engineering College, 2015

2017
Wright State University

WRIGHT STATE UNIVERSITY

GRADUATE SCHOOL

April 20, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Venkata Naga Siva Seelam ENTITLED Settings Protection Add-on: A User-Interactive Browser Extension to Prevent the Exploitation of Preferences BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQIREMENTS FOR THE DEGREE OF Master of Science.

Adam R. Bryant, Ph.D.
Thesis Director

Mateen M. Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on
Final Examination

Adam R. Bryant, Ph.D.

Mateen M. Rizki, Ph.D.

Michelle A. Cheatham, Ph.D.

Robert E.W. Fyffe, Ph.D.
Vice President for Research and
Dean of the Graduate School

ABSTRACT

Seelam, Venkata Naga Siva. M.S., Department of Computer Science and Engineering, Wright State University, 2017. Settings Protection Add-on: A User-Interactive Browser Extension to Prevent the Exploitation of Preferences.

The abuse of browser preferences is a significant application security issue, despite numerous protections against automated software changing these preferences. Browser hijackers modify user's desired preferences by injecting malicious software into the browser. Users are not aware of these modifications, and the unwanted changes can annoy the user and circumvent security preferences. Reverting these changes is not easy, and users often have to go through complicated sequences of steps to restore their preferences to the previous values. Tasks to resolve this issue include uninstalling and re-installing the browser, resetting browser preferences, and installing malware removal tools. This thesis describes a solution to this problem by developing a user-interactive browser add-on named "Settings Protection."

This thesis describes the various means of exploiting preferences in the Google Chrome and Mozilla Firefox browsers and discusses preferences that are frequently exploited by browser hijackers. The Settings Protection add-on observes and detects changes in preferences that users may be unaware of. After detecting these unknown changes, the add-on reverts the modified preferences to old values and saves the new changes upon the user's confirmation. If the user is not expecting these changes in the browser, the add-on discards them. A proof of concept add-on for the Mozilla Firefox browser implements this research and is tested in a real-time environment. Lastly, this thesis evaluates the performance of the Settings Protection add-on using the Mozilla Firefox performance tools.

Table of Contents

List of Figures

Acknowledgement

I would like to thank my advisor, Dr. Adam Bryant for his guidance and support to this thesis development. Besides, I would like to thank the rest of my thesis committee members: Dr. Michelle Cheatham and Dr. Mateen Rizki for their valuable time and insightful comments to this thesis.

I would like to extend my gratitude to Dr. Ramana Grandhi, Executive Director of International Collaborations and Graduate Programs and the Wright State University Department of College of Engineering and Computer Science for the financial support under Assistantship Award Scholarship. I would also like to thank Dr. Mateen Rizki, Chair of Department of Computer Science and Engineering for extending the financial support.

Finally, I would like to thank my family and friends for their support and encouragement. I would especially thank my parents for their support, love, and care towards me.

1. Introduction

1.1. Browser Hijacking

Browser hijacking is a type of attack in which the attacker tries to modify the normal operation of a web browser. The attackers change the behavior of the web browser through modifying the preferences. Browser hijackers mainly target the preferences such as default search engine, homepage, and new tab page URL. They also modify the browser security preferences to a lower level of security. By disabling the security preferences, it would be easier for them to take the user to websites of malicious content and advertisement pages.

1.2. Various ways of Exploiting Browser Preferences

The malicious software exploiting preferences enter the browser in many ways including installing an add-on, tricking users to click on pop-ups, malicious software bundled with free software, and Drive-by Download. In the following sections, this thesis explains each of them in detail.

1.2.1. Browser Add-ons

A browser add-on is a piece of software, which extends the functionality of the browser. The second chapter provides a detailed explanation of the concepts about the add-on system and add-on architecture. Consider a scenario where a user tries to install a browser add-on. When the user installs a browser add-on for a specific purpose, the add-on may change the preferences of the browser without the user's notice apart from its normal desired operation.

For example, the user is looking for an add-on to block advertisements. The user will look for different add-ons available in the official Mozilla add-on store. Based on the add-on description, the user will find an add-on from the available list of add-ons suitable to his requirements. The user installs the add-on in the browser hoping that it will serve a specific purpose as per his needs. However, the installed add-on may perform additional operations on the browser preferences where a developer may not disclose these additional functionalities in the add-on description. If the user comes across such type of add-on, some browser preferences modification happens without the user's notice. Users unknowingly install these types of add-ons, which make undesired changes to preferences. Browser hijackers make use of this path to deploy malicious add-ons available in the market.

1.2.2. Advertisement Pop-ups

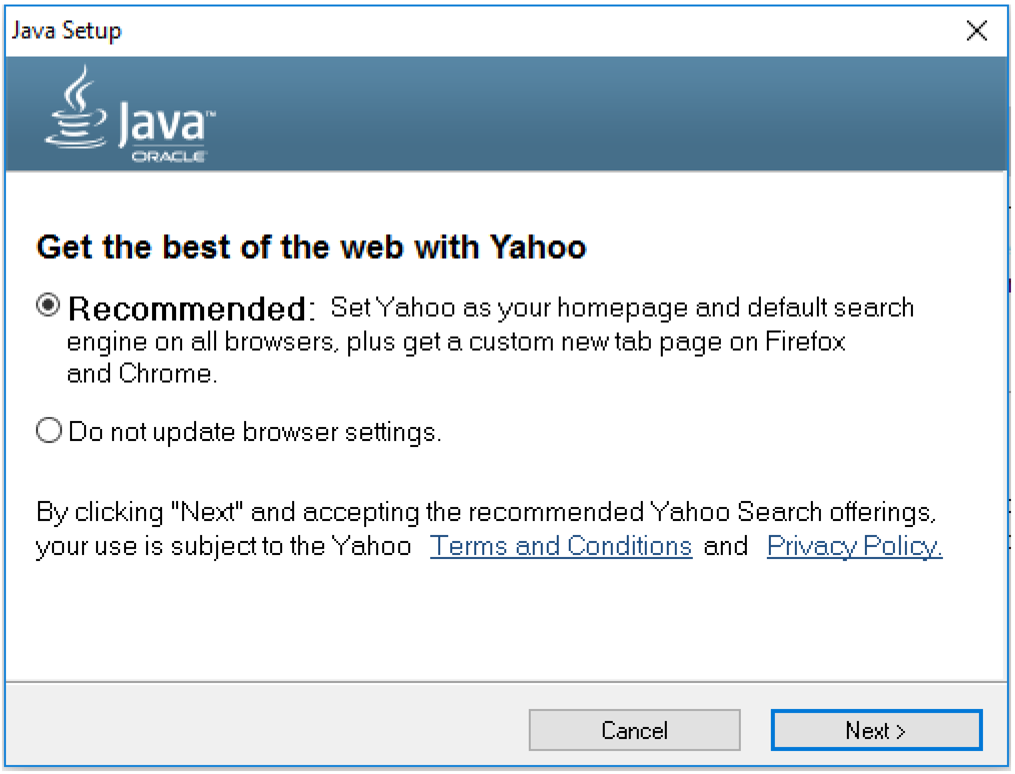
An advertisement pop-up is a browser window displaying random advertisements. Some websites create a lot of pop-ups with advertisements and trick users to click them. These pop-ups include redirecting to random advertisement pages, the opening of new advertisement windows, and fake anti-virus software advertisements. In these pop-ups and advertisement pages, browser hijackers will open add-on installation pop-ups. In the process of closing these pop-ups and advertisement pages, users may agree to installation of the malicious add-ons. These undesired add-ons in the browser might modify preferences without the user's permission.

1.2.3. Drive-by Download

A drive-by download is a process, in which clicking on advertisements pages leads to download of malignant codes into the browser. Some websites open new advertisement windows and occupy the whole desktop screen. These advertisements may contain malware that is harmful to the user's system. In some cases, when the user clicks on such malicious advertisements, the browser window downloads malignant codes. These codes change browser behavior causing redirection to malicious websites and modifying the browser preferences. These redirecting websites include some fake antivirus pages which prompt the user to install antivirus software. In other cases, the hijackers share malicious website links through email and make users click them.

1.2.4. Freeware Software

Malicious code bundled with free software is one of the primary sources for browser hijacking. While installing free software, users may unknowingly accept to additional downloads, which include malicious add-ons and software. Hijackers will trick average users intentionally to install the malicious codes along with the main software. If the user skips on unselecting the additional downloads, then it leads to installation of other unwanted malware programs. Moreover, browser hijackers do not mention enough details or description about the extra software attached. This additional software disturbs the user's surfing experience by modifying the browser preferences and creating a lot of advertisements.



Get the best of the web with Yahoo

[✓ **Recommended:** Set Yahoo as your homepage and default search engine on all browsers, plus get a custom new tab page on Firefox and Chrome.]

Do not update browser settings.

By clicking "Next" and accepting the recommended Yahoo Search offerings, your use is subject to the Yahoo Terms and Conditions and Privacy Policy.

Figure 1. Recommendations Provided by Java Software

For instance, when installing the Java software downloaded from the official ORACLE website, the installation process recommends some suggestions. These recommendations include a change of the search engine and homepage URL to the Yahoo search engine and homepage. Figure 1 shows these recommendations. The Java installation process provides an option to de-select the installation of unwanted pieces of software. While clicking on the "next" button continuously in the installation process, there is a chance that an average user misses the de-selection of undesired pieces of software. It leads to modifications in the browser preferences such as changes in the homepage URL and changes in the default search engine.

1.3. Significance of the Browser Hijacking

Browser hijackers exploit browser preferences mainly for the benefits gained indirectly. The benefits may include obtaining revenue through advertisements and increasing the number of downloads for a specific software. The modified values of the preferences search engine and homepage directs the user browsing window to an organization’s web page thus, increasing the page visits to that organization’s web page. This process benefits both the browser hijackers and the third parties at the cost of the end users surfing experience.

Another important consequence of the browser hijacking is that the user needs to spend additional time in browsing. For instance, consider a case where hijackers change the preference search engine. Now, the user searches for something in the browser URL bar assuming that the default search engine in the browser is his desired value. Whereas, the user obtains the unexpected results from the modified search engine. To obtain the desired results, the user again should search on the web page of the old search engine. This rework in navigating across search engines would affect the user’s browsing time.

1.4. Scope

This thesis seeks to provide a user interactive solution to prevent browser preference exploitation. This research focuses only on the browser preferences and does not consider other issues in browser hijacking such as the redirection to new web pages and random advertising pages. This research implements the solution by developing a browser add-on called "Settings Protection," in the range of add-on capabilities to handle the modifications in preferences. Considering all the above aspects, the intent of this research is to discover answers to the below questions:

Q1: In what ways, can browser preferences be exploited through browser hijacking? What preferences are targeted in browser hijacking?

Q2: Can a browser add-on be effective in defending the browser preferences exploitation?

Q3: How does our browser add-on performs and analyzes its functionality using browser performance tools provided by Mozilla Firefox.

1.5. Thesis Overview

Throughout this document, the word "hijacker" refers to a person who makes the attacks browser hijacking. In web browser terminology "settings" and "preferences" are synonyms and this document uses the word "preference". Chapter one briefly discusses the concept of browser hijacking and the various ways of exploiting preferences using browser hijacking. This chapter further provides the significance of the browser hijacking issue and its impact on the user. Lastly, this chapter defines the scope and research questions of this research work.

Chapter two is about the background information of browser hijacking. This chapter discusses the related work and preference architecture of Google Chrome and Mozilla Firefox. Further, this chapter addresses the preferences targeted in browser hijacking. This chapter presents the existing solutions of the browsers Google Chrome and Mozilla Firefox against an unwanted modification. Later, this chapter continues with a study of the add-on system and add-on architecture. This chapter completes the discussion of the methods of accessing preferences in both the browsers.

Chapter three is all about the approach and the design of our proposed solution. It discusses all of the design steps of the add-on and presents the features provided by the add-on. Chapter four presents the user interaction and performance measure of the implemented add-on with corresponding screenshots. The final chapter discusses the limitations of the add-on and presents the future recommendations, followed by the summary of the thesis.

2. Background & Literature Review

2.1. Background Information on Web Browser Attacks

A browser is one of the primary and necessary applications on any computer that has a network connection. The browser is the platform for a user to access many different activities, including social networking, online shopping, financial services, and educational resources. (Ter Louw et al., 2009).

Paul Lonescu (2015) gives a brief report on different kinds of attacks in a web browser. URL redirecting is one of those attacks. In a URL redirect attack, attackers try to forward URLs of frequently visited websites to a web location that hots malware, which is undesired by the end user. Attackers exploit the additional software running in the browser like plug-ins and add-ons. Poor maintenance and lack of proper updating of old software are among the targets to attackers that exploit. Through attacking old software in the system, it is possible to steal sensitive information of the user that the browser sends and receive, which are unsecured with old security mechanisms (Zuccato et al., 2015). Attackers try to inject malware JavaScript into trusted websites using Cross-site scripting (XSS), which might inject advertisements or do some malicious activity in the browser. Another important type of web attacks is browser hijacking, which is the modification of preferences without user notice, and this thesis is about preventing browser hijacking.

Removal of some malicious codes is easy by uninstalling the corresponding malware using the computer's control panel application. The user has a chance to avoid these types of software, but in some cases, the process of removing malware is tougher. A more difficult removal method involves installing a malware removal tool to remove unwanted programs in the system. Other solutions include deleting hidden add-ons in the browser, removing browser registry files, and removing all of the browser's locally stored data. These methods might be painful for the user to go through the total process of removal.

The solution in this thesis is simple compared to these types of solutions. The add-on will confirm with the user before saving new changes and provides information regarding the source of any change. With the add-on, it is very easy for the user to protect browser preferences.

2.1.1. Spyware

Spyware is one category of web malware programs in which the attackers interrupt, taking limited or full control of a system operation without the knowledge of the authorized system user (Gribble et al, 2016). Several types of attacks in the web browser fall under spyware. AM Payton (2006) gave a report discussing different spyware tools and techniques. The various kinds of attacks discussed by him are:

- Random advertisements which show up automatically,
- Automatic download of malware known as drive-by download,
- Session hijacking,

- Bundled packages download,
- Fake anti-spyware pop-ups appear with the description "Your browser is infected and you need to install anti-virus tools," and
- Browser hijacking.

Regarding preferences, Payton discussed that the browser hijackers change the browser preferences like homepage URL to advertising pages, drive-by download pages, and other sites for commercial purposes. Hijackers change browser security preferences allowing silent installation of third party software, which leads to modification of the normal browser operation.

Steven Gribble (2016) proposed a detection technique for spyware, which is one of the latest research efforts on detecting spyware. They used a virtual machine (VM) to detect and stop spyware entering the computer. They implemented a tool to detect the activities of spyware and drive-by downloads in the browser. They used a VM to stop and analyze malware programs in their tool. When downloading and installing any program executable files in a virtual machine environment, there are many tools available to detect if the executable file is malicious or not. In this way when the user is opening a web page in a VM environment controlled browser, their tool alerts the user if there are any attacks of malicious program downloading or drive-by downloads.

2.2. Related Work

A vast amount of research work is available for detection and classification of an URL to categorize it as malicious or non-malicious. Niels Provos and his team (2010) from

Google did one such type of research analyzing malware attacking browsers. They mainly concentrated on detection of malware web applications and creating a database. They analyzed several millions of URLs and successfully found 450k URLs, which download malware into the browser and several thousands of malicious URLs. MoshChuk and colleagues (2005) did a similar type of research. He studied about web applications which are downloading malware binaries into the browser. Both of them concentrated on detecting malware websites rather than defending.

This thesis is entirely different from this research. This thesis focused on only preferences exploitation which is one part of web malware, and the proposed solution is a user interactive solution rather than an analysis and the detection of malware.

There is a lot of research done and published on security and exploitations in browser add-ons, and a few of them are discussed here. Mike Ter Louw and his team (2009) proposed a security solution in the browser regarding add-ons. For analyzing how could an add-on damage the browser, they developed a malicious add-on that attacks the browser. They observed two major problems; code integrity and attacking the user-sensitive information in the browser. They proposed user authentication to the add-ons and granting permission only for authorized add-ons. They tested their solution using many Firefox add-ons and were successful in achieving desired solutions.

Felt (2010) studied the permissions given to Firefox add-ons and states 88% of the add-ons do not require the permissions that they have. He proposed a new add-on system that gives very limited privileges to add-ons to more concisely meet their needs.

Karsten Knuth (2012) analyzed the add-on system of the following browsers: Internet Explorer, Google Chrome, Mozilla Firefox, and Safari. He listed all of the vulnerabilities in each browser add-on architecture. He developed a Chrome add-on for testing and proposed new security mechanisms in the Chrome add-on system.

Rezwana Karim (2015) studied the present Mozilla Firefox add-on system's security and proposed enhanced security to the architecture. They presented Beacon, which detects unsafe practices in JavaScript programs and detected 36 unsafe cases in 68000 lines of JavaScript code. Mozilla Firefox implemented a new add-on system and required porting of legacy add-ons to the latest add-on system. For this, Karim provided a technique called Morpheus for porting purpose while showing vulnerabilities in legacy add-ons.

Nicolas Golubovic (2016) studied attacks through add-ons in the browser. He manually tested add-ons, which is different to previous research in this area and provided case studies on attacks through vulnerable add-ons. He implemented a test suite for add-on system testing and states that current security mechanisms are not sufficient to prevent attacks in a browser.

There are few studies on broad browser hijacking, which means the research is not just about browser preferences but includes other malicious activities in the browser also. Diogo Monica and Carlos Ribeiro’s (2015) studies were one of the latest research efforts. They developed a tool in the form of an add-on to detect browser hijacking activity in the browser. Their tool detects any suspicious activity in the browser and automatically reports the malicious URLs. Their primary intention is to detect hijacking activity and report it to the user. They used a certain number of factors for detecting hijacking such as JavaScript code running in a potentially harmful manner, botnet commands in the browser, and linear detector used to separate browser hijacking based on some factors. Their tool is successful in detecting malicious activity. However, this tool needs validation with larger datasets.

Diogo Monica et al., (2015) developed a tool that has some specific features such as not needing a network connection and remote data. All the factors for detection are available in the browser add-on only, and the add-on operates without the need for any additional components. The add-on developed in this thesis also does not require any additional resources.

There is minimal research conducted on the hijacking of browser preferences. One of them is "Software protection against changeware" by Sebastian Banescu and colleagues (2015), but it is for general software assets, and in the case of a browser, the assets are preferences. They proposed a solution to protect software data from changeware. Changeware is a piece of code or software that attacks and changes software application assets. These assets are the important files of the software such as preferences in a browser.

Changeware is different from other types of malicious programs. It does not need deep permission in an operating system as the developers of the changeware target only the changeable resources of software. These changeable resources are not under operating system security mechanisms, and this makes it easy for the changeware developers to attack the assets of software. Banescu and colleagues (2015) explain that various anti-malware software available are unable to stop changeware attacks as these attacks look like an authentic operation in the operating system.

The solution proposed by Banescu and colleagues (2015) is to provide security in the target application which has important assets. For instance, if the Google Chrome browser is the target application, and their proposed solution suggests modifications to the Chrome architecture. They had implemented the solution in the Chromium browser to protect preferences from hijacking and evaluated implementation as a case study. They had implemented the solution only in the Windows operating system, but the proposed solution in this thesis is independent of the operating system. Their protection mechanisms include software diversity and white box cryptography. They detect asynchronous calls made in the software application and verify them. They verified these calls against a set of genuine calls, which is a list of possible calls in that particular software. Their solution also consists of run-time checking, and it includes injection of code in calls. This code injection is only possible in the Windows operating system as the Linux-based operating systems does not allow this.

Another research effort discussed here is about exploitation of browser preferences. Thomas Wespel and Thomas Salomon (2015) filed a patent on technology to stop changes to preferences like homepage, search engine, and new tab URL from freeware software installations. They discussed different methods of preferences exploitation and proposed a cloud-based solution.

They discussed that changes in preferences are primarily due to advertising and marketing in the browser. Some major companies like Java also provide these types of recommendations for changes in browser preferences. When installing Java, there is an opt-out option for not changing the browser preferences. If users miss that opt-out option or overlook the recommendation, the additional software will modify the preferences.

In the case of hijacked preferences, browsers do not allow users to easily revert the hijacked values to previous preference values (Wespel & Salomon, 2015). The user needs to revert all the preferences to default preferences values or revert every single preference to their choice of value, which is somewhat time-consuming. Wespel & Salomon (2015) describes that reverting the preferences to previous values is possible through their proposed solution, which relieves users from this time-consuming problem. In the solution described in this thesis, whenever there is a change and if the add-on detects that the user did not do the change, then the add-on reverts the changes preferences to old values. If the user wants to save the new changes, then the add-on saves the new values. Otherwise, the add-on will discard the new changes and save the old values.

Wespel and Salomon (2015) proposed a database of blacklisted preferences based on some analysis. The analysis includes the events of malware, new software installation, and drive-by downloads. After detecting a preference change in a suspicious manner, their solution will verify whether the preference is a blacklisted preference. If it is a blacklisted preference, then they will report to the user about the change and provide recommendations accordingly.

While in this thesis case, preferences changed without user notice are immediately reported to the user and asked whether to save or discard new changes. In the solution described in this thesis, the add-on need not maintain any database nor does statistical analysis unlike what Thomas & Salomon(2015) claimed to do.

2.3. Preference System

Preferences, options, and settings are synonyms in browser terminology. The user can configure the browser through preferences. Users can change the browser's behavior by modifying the preferences to their desired values. For instance, users can set the value of preference homepage to a frequently visited web page, so that they can easily access that web page by simply clicking the home button. In this way, by configuring the browser with desired preference values, users can save a little time.

In the Google Chrome architecture, each preference has a value and key, and the key is the means of accessing preferences in add-ons. The preference values are of different data types, which are Boolean, string, integer, or other specific type based on a particular preference. There are different kinds of preferences in Google Chrome; some are runtime editable preferences, preferences modified at start time and shutdown time, and preferences stored by add-ons (Battre & Pamg, 2013). Users configure only the runtime editable preferences, which are the focus of this thesis.

Battre and Pamg (2013) describe that the preferences are stored locally in a file called "preferences.json" and the user's profile folder contains this file. The user's profile folder is available in the corresponding browser locally stored data. Chrome maintains separate profiles if more than one user is using the browser. At the time of browser installation, there will be no profile folder. Opening the browser for the first time will retrieve the preferences from the master preferences file, which contains default preferences values. While using the browser for the first time, the browser will create a profile folder. From the second time of opening the browser application, the browser loads the preferences from the preferences file, which is available in the user's profile folder.

In Mozilla Firefox, the preference system is somewhat different. Preferences in Mozilla Firefox are available through XPCOM services, and "nsIPrefService" and "nsIPrefBranch" are the interfaces when accessing the preferences. Mozilla designed the preferences in a tree branches method, and every category of preferences lies under the root branch of the tree. For example, the accessibility group of preferences is available under the Accessibility child branch of the preferences root branch. For accessing preferences, the add-on developer uses nsIPrefService and the name used in accessing includes its ancestors also. For instance, to access the preference, "enablesound," the word "accessibility.typeaheadfind.enablesound" is used. The preference "enablesound" is a child branch of typeaheadfind, which is a child node of accessibility. The preferences names available in the page "about:config" are according to this tree branch.

In Google Chrome, users can see the configurable preferences on the web page with the URL, "chrome://preferences," and in Mozilla Firefox, "about: preferences" is the URL. Users can open these web pages by clicking the "Options/Settings" button available on the toolbar of the corresponding browser. In this document options/settings page refers to these web pages. Some of the runtime editable preferences and the user configurable preferences are only visible on the preferences page. In Mozilla Firefox, users can see all the preferences by using the URL, "about:config," and only some of them are available on the options page with the URL, "about:preferences".

2.3.1. Preferences targeted in Browser Hijacking

This research analyzed the key targeted preferences in browser hijacking in designing a robust solution. The default search engine is one of the preferences which is one of the hijacker's prime targets. Examples of the major search engines are Google, Yahoo, Bing, Baidu, Ask.com, AOL, and Yandex. Browser hijackers exploit this preference in several ways, and users might face a tough time in getting the preference back to their desired value. On a modified search engine, the users get unexpected results in the browser and need to search again for the desired search engine results.

Another significant browser preference that the hijacker targets is the exploitation of browser homepage. Users set their homepage to their desired URL which they frequently visit and save their time by just clicking the home button on the toolbar of the browser. Unexpected changes in the homepage URL due to the modifications of browser preferences may annoy the user.

Some hijackers target preferences such as "Block pop-up windows" and "Block Dangerous & deceptive content". They modify these two preferences values to false, which are true by default. By altering Block pop-up windows to false, any web page can open random and unwanted advertisement pop-ups without the interaction of the user. These pop-ups trick the user to click them, which cause the download of malicious content or forward to an irrelevant website. By enabling this preference to true, the browser will block pop-ups, which cause malicious attacks. By disabling "Block dangerous and deceptive content," the user will be at risk from dangerous and malware websites. The user might unknowingly expose valuable information to suspicious third party websites.

Another significant preference targeted is new tab URL, the hijackers modify this preference value to an organization website for publicity and commerciality purposes. Users cannot change this preference directly in options page "about:preferences," and if they want to customize a new tab page, then they need to install any add-on like "New tab override" which customizes new tab URL. Users should go through this long process for reverting this preference value to their desired page URL.

In the design of solution, this thesis considered the above preferences more important than other runtime editable preferences. This thesis researched all the possible ways a hijacker might try to exploit these preferences and designed the add-on accordingly.

2.4. Comparing Browser Actions against Unwanted Changes

As previously mentioned, this thesis studied security measures on preferences taken by two browsers, Google Chrome and Mozilla Firefox. Considering Google Chrome, silent installation of an add-on, which is installing an add-on into the browser without user permission, is not possible. The user must agree to installing an add-on into the browser. Another useful measure Chrome gives its users is that if an add-on modifies any preferences, then it will display the name of the add-on above that preference in the settings page. In this way, the user knows which third party tool is controlling a particular preference.

When an add-on modifies a preference value, Chrome does not warn the user about modification of a corresponding preference immediately. However, it will give information about modifications when the user performs an activity in the browser related to that particular preference. For instance, an add-on or malicious code modified the preference default search engine in the browser. At the time of modification, Chrome does not warn the user about search engine modification. When the user searches for something in the Omnibox, the browser gives results from the new search engine. Now Chrome gives a warning pop-up showing "Is this the change in the search engine you are expecting?" or "Are these the results you are expecting" and gives the user an option whether they want to revert the search engine to a previous value. The solution in this thesis is such that the add-on will generate a pop-up immediately at the moment of a change in preference without users notice.

Coming to Firefox browser, silent installation of add-ons is possible. There is a preference called "warn before installing of add-on silently." Disabling this preference, the hijacker can easily install add-ons without user permission. Unlike Chrome, Firefox does not warn about modifications in preferences at any time. If any add-on modifies preferences, then the user should manually revert those preferences or uninstall the particular add-on that modified the preferences. The user does not have any option to know which add-on changed the particular preference value. The Google Chrome browser has the upper hand compared to Mozilla Firefox in preferences protection with providing users some warning pop-ups.

2.5. Add-on System

Add-ons became very powerful, and popular nowadays and will be in the future. (Felt, 2010). Add-on is a synonym for extension in the browser field. An add-on is a piece of software which extends the functionality of the browser. Plug-in is different from add-on, in that a plug-in also extends a browser’s functional operations, but the functionality of the plug-in is specific to a page. An add-on is browser specific and functionality is all over the browser and not specific to one web page. For instance, Adobe Reader or any media player is a plug-in, which is web page specific. "Search and Newtab by Yahoo" is an add-on, which changes preferences default search engine and new tab URL to Yahoo.

Add-on developers keep all of the files needed for the add-on in one file, and the add-on development tool will export that single file into a browser-supported format. The browser-supported format is the format of an add-on file in a particular browser. It is a "crx" type of file in the case of Google Chrome and "xpi" type of file in the case of Mozilla Firefox. Users can simply download these types of files and install them into the browser.

The process of installing an add-on will be less than one minute, and the user does not need to follow any steps for installing. The user can find different add-ons available in the official add-on store of the browser. After finding a suitable add-on, with a single click users can simply install the add-on in the browser. The simple installation process is one of the reasons to choose an add-on for implementing the proposed solution.

Another reason to select an add-on as our implementation is, in any research implemented, the solution needs feedback and suggestions from users. To get feedback, many users should use the developed solution. In the browser, an add-on is the quick path to reach users. The Firefox official add-on store AMO (addons.mozilla.org) statistics show that the total count of add-ons used daily is 200 million to 300 million in only the Firefox Browser. These statistics adds support to implement the proposed solution in the form of an add-on. With the add-on, it is possible to get feedback from users and suggestions easily. With the feedback from users, the developers can modify the add-on, and release a new version.

The performance degraded by installing an add-on is also minimal, and performance of the add-on is also good. The fourth chapter discusses the performance of the add-on in detail. The add-on will not cause any disturbance to the user's browsing experience unless there is an unwanted modification.

2.6. Add-on Architecture

An add-on is an exported version of a group of files. These files are of different formats such as JSON, JavaScript, HTML and CSS. Add-on developers have access to several APIs provided by the browser. Add-ons can perform different types of actions related to the browser, including content scripts changing, adding buttons to toolbars, and web content blocking. Developers can add a user interface if required to the add-on and provide customization to the browser. The following sections discuss the add-on architecture of the browsers Google Chrome and Mozilla Firefox.

2.6.1. Google Chrome

In any add-on, there must be one file which provides essential information and the starting point of execution. "Manifest.json" is the most important file in a Google Chrome add-on. It contains the all of the necessary information regarding the add-on. Any typical Google Chrome add-on consists of following files:

1. Manifest.json file
2. HTML files (zero or more as per need)
3. JavaScript files (one or more as per need)
4. Data folder

The “manifest.json” file is an essential file for every Google Chrome add-on. It contains the basic information regarding the add-on such as developer name, version number, add-on name, and minimum chrome version required for installing. It contains the name of the JavaScript file that is the starting point of execution at the time of add-on installation and every further browser start-up. If the add-on requires its own preferences/options, then the developer should mention them in this file. This file contains all of the details about the self-preferences including name, description, title, the data type of the value, and the default value. The "manifest.json" file initiates the execution of all of the other files, whether HTML or JavaScript. Following is an example of, how the "manifest.json" file looks (Brandon H. Baker. 2013)

```
{
  "manifest_version": 2,
  "name": "Preferences protection",
  "description": "This add-on protects the browser preferences from unwanted changes",
  "version": "1.0",
  "browser_action": {
   "default_icon": "icon.png",
   "default_popup": "popup.html"
  },
  "permissions": [
   "activeTab"
   ]
}
```

The above code is a very basic manifest file. In the above code, popup.html file is the HTML file used for displaying the content of a pop-up window upon clicking the add-on icon on the toolbar. Similarly, developers can use several other HTML and JavaScript files for add-on development. A manifest file does not contain any logic code; instead it contains all of the information about the add-on. It is like metadata about the add-on. This file contains a description of extra features and restrictions in the add-on.

The data folder contains all the required images for the add-on such as icons and pictures needed as part of the user-interface. The developer has the flexibility to include many files and different file formats.

Add-on developers can use standard Google Chrome add-on APIs for interacting with browser actions. There are many APIs available, and each serves a specific operation. For instance, they can use the button API for adding an action listener to any buttons. They can write on-change listeners for preferences values using the preferences API. They can use the tabs API to read the active tab URL of the browser.

In Google Chrome, developers must mention the permissions needed for the add-on. Google Chrome designed their latest add-on policy in such a way that an add-on can only request the services it needs. By this restriction, it eliminates security issues from buggy add-ons.

2.6.2. Mozilla Firefox

In Firefox, three different types of add-on implementation methods are available: bootstrap add-ons, web add-ons, and SDK add-ons. Web add-ons are the newest add-on system out of the three, and the bootstrap add-ons system is the oldest one. The problem with bootstrap add-ons is that developers need to learn XUL programming, which takes a lot of time. Another issue is that the bootstrap add-ons are restart add-ons, which means to complete the installation of the add-on, the user must close and reopen the browser. The web add-ons system is in the development stage, and there are no APIs available to develop the solution proposed in this thesis. (Wbamberg, 2016).

Based on the above information, this thesis selected the SDK add-ons system for developing, so that users can complete the installation process of the add-on without restarting the browser. Developing an SDK add-on is easy compared to other add-on systems. Documentation for SDK add-ons is also good and sufficient for low-level technical user’s understanding. It is possible to develop the proposed solution using the resources available for SDK add-ons.

Following is the basic architecture of an SDK add-on. Similar to the "manifest.json" file in Google Chrome, "package.json" is the most important file in a Mozilla Firefox add-on. It contains all of the essential information about the add-on. A typical Mozilla Firefox SDK add-on contains the following files:

1. “package.json”
2. “main.js”
3. Other HTML and JavaScript files as required

4. Data Folder

5. Test folder

"Package.json" is an essential file and the add-on implementation starts with writing this file. It contains all of the basic information about the add-on such as the title of the add-on, description of the add-on, developer of the add-on, version number, minimum Firefox version number required to install the add-on, main JavaScript file name, and keywords. The file, "main.js," is the starting execution point of the add-on. The file "package.json" does not contain any information regarding other JavaScript or HTML files used for pop-up windows. It is like metadata of the add-on. Following is an example of how the "package.json" file looks:

```
{
  "title": "Settings Protection",
  "name": "Settings-protection",
  "version": "1.0.1",
  "description": "To prevent the unwanted preferences
changes in the browser",
  "main": "main.js",
  "author": "ACSL WSU",
  "engines": {
    "firefox": ">=38.0a1",
    "fennec": ">=38.0a1"
  },
  "license": "ACSL",
  "keywords": [
    "preferences"
  ]
}
```

The "main.js" file is the next important file of the add-on. It contains all of the code for the add-on execution. It can access other files in the add-on folder including data and test folders. The above code will be the same for any Firefox add-on. The main field in the "package.json" file describes the name of the file, where execution starts. Developers need to write all of the other code in "main.js" and supporting JavaScript or HTML files.

The data folder contains the images required for add-on icons and other data files. The test folder is optional, and developers can use this file to store documentation and testing files for things like exporting the add-on. Developers will write code in "main.js" and for simplicity of coding, can add other HTML or JavaScript files. The data folder contains these additional files for quick accessing.

If the add-on functionality requires its own preferences/options, the developers should mention them in the "package.json" file only. Developers can also hide these preferences in add-on options. The title and description for each preference will be visible to the user. A checkbox will be visible for toggling the preference value if the datatype of the preference is a Boolean datatype. If it is a number or string, then a text area will appear accordingly. Following is an example of a self-preference declaration in the "package.json" file:

```
{
"description": "Never change preferences/options from
outside of preferences page. When you enable this
feature, then the add-on discards any change in
preference/option value which is not from preferences
page",
        "name": "NeverChange",
        "type": "bool",
        "value": false,
        "title": "Never change preferences/options from
outside of preferences page"
 },
```

Mozilla Firefox has standard APIs, which are helpful to add-on developers. For instance, developers can use the preference system API for modifying and getting preference values, and writing on-change event listeners for preferences. They can use the SDK/tab API for reading the active tab URL of the browser. Panel and button APIs are available for the user-interface part of the add-on.

An add-on interacts with the user through pop-up windows, and the panel API in Mozilla Firefox provides these windows. Developers need to include HTML files for displaying content in a panel. Developers can add JavaScript files for Document Object Model (DOM) manipulation and data passing. DOM manipulation is updating the data of the panel window, and it is possible to pass data dynamically to the HTML elements from the JavaScript file. Developers should mention the information about these panels such as the supporting file name, dimension properties, and focus properties of a panel in the main file, that is "main.js." The following is example for an SDK panel declaration:

```
var textEntryPanel = require("sdk/panel").Panel({
        width:530,
        height:390,
        contentURL: data.url("text-entry.html"),
        contentScriptFile:data.url("get-text.js")
});
```

In the above code, contentURL is the URL of an HTML file, and contentScriptFile is the URL of a supporting script file in the current directory. It is possible to set a panel focus property, such that the pop-up will not fade away if users click outside of the panel dimensions. However, the user can close the browser when there is an active pop-up, and this is a limitation of these panels. As part of the content in the HTML file, the developer can add any framework for extra functionality as per the design requirements.

2.7. Accessing Preferences

In Google Chrome, add-on developers can access the preferences easily using stable APIs provided by Chrome for add-on developers. An add-on has permissions to get the current value of preference or set a new value to preference. Add-ons are able to detect the changes in preferences by using an event observer for that particular preference. Upon notification, add-ons can do anything with the particular preference. If two add-ons simultaneously want to set a particular preference, then the value set by the add-on installed latest dominates (Battre and Pamg, 2013).

Google Chrome allows the add-on to configure only the preferences it requested and it will not allow the add-on to change or get the other preferences. If the add-on should be effective in the incognito window also, then developers should specify this in the "manifest.json" file. Preferences set by an add-on dominates those set by the user and default values of the preferences. In Google Chrome, all preferences are either Boolean, string, or integer, and never an object nor array.

Considering Mozilla Firefox, developers can access the preferences using the preference service API in an SDK add-on. However, it is not possible to access some preferences using this API. Some preferences like new tab page and search engine are not accessible with this API. Florian Queze (2015) reported the add-on developers misuse new Tab URL preference, and Mozilla considered this complaint and removed that preference from the options page. By removing that New tab preference, it is no longer available in "about:config" page for modification or reading. Add-on developers should use the "newtabURL.jsm" service for accessing preference new tab URL.

Developers cannot access another significant preference, that is search engine using the reference service API. Firefox stores this preference values as a plug-in object with a set of properties. For accessing the search engine, developers should use the service "nsIBrowserSearchService." With this service, an add-on can retrieve the current search engine or replace it with a new engine. Before changing the search engine to a new value, developers should add the new engine to the group of search engine plugins available in the browser. Developers need to provide properties like name, URL, icon URL, and data type to the new search engine object.

3. Add-on Design and Implementation

3.1. Approach

The main aim of this thesis is to stop random changes in browser preferences in which the user is not aware of. The initial idea of this thesis is to stop changes in preferences completely, that is to stop any modifications the user does not know about and then ask the user whether to save or discard the changes to the preferences. The user should go through authentication for saving new changes in preferences. During and before the authentication, the old values of preferences are only in action and configured in the browser. However, a few problems in preventing unwanted modifications completely.

The first issue is the browser preference architecture. In some applications, users need to click the "okay" button after configuring preferences and only after this are the new changes placed into action. In the case of browser preferences, they are of a different type compared to the above applications. In the browser, the preferences available to the user for modification are runtime editable preferences, and the user does not need to click any button after making modifications. The moment there is a change in preference, the browser configures and operates accordingly with the new changes. For example, if any add-on sets a different URL as the value of preference homepage, then from that moment, clicking the home button will load the new homepage URL.

In this architecture, it is not possible to stop the changes from happening. Stopping random changes completely requires modifying the architecture of the browser should, which means changing the source code of the browser. For a complete prevention implementation, modifying the source code of the browser is the only solution. While studying about modifying the browser source code, there are few problems. Browser source code is large and takes a huge amount of time to understand. Another issue is that building the source code requires computers with decent performance, which is expensive. Based on these factors, this thesis avoided implementation in this way and searched for alternatives.

This thesis considered the approach of observing and restricting modifications to the file "preference.json." The browser saves the preferences in a file called "preference.json" available in the user’s profile folder. Whenever there is a change in this file, the user should confirm the changes. Keeping a barrier to this file modification is difficult as this file contains all other preferences along with the runtime editable preferences. The other type of preferences included in this file are preferences modified without the user interaction and the preferences modified at the time of browser start-up and shut down. It is not possible to know the type of changed preference upon this file modification and implementation in this way is also not possible.

This thesis researched what an add-on can do regarding preferences exploitation. This research studied the add-on architecture to know what permissions an add-on has on preferences. An add-on can retrieve and set the value of preferences to a different value, and it can also observe changes in preferences. The add-on installing process is very straightforward and user-friendly. An add-on does not require any building costs and users can simply download the add-on and install it in their browser from the official browser add-on store. With these positive factors, this thesis decided to implement the solution in the form of an add-on.

3.2. Overview of the Proposed Solution

This thesis provided the proposed solution in the form of an add-on. The following points give a brief outline of the solution:

- Upon installation of the add-on, the user needs to select an authentication: master password or separate (used for the add-on only) password.
- If the user selects separate authentication, to handle the case of a forgotten password the user needs to enter security questions and answers.
- The add-on stores the old values of the preferences and listens for preferences modifications.
- The add-on authenticates every change in preferences not made by the user and modifications in the add-on self-preferences.
- Upon a change in preference, determine whether the user initiated the change or not.
- If the change is not from the user, then a pop-up window will appear with the user selected type of authentication and the user should choose whether to save or discard changes.

- The add-on provides users with some specific features in the form of self-preferences, which include blocking preferences page, never changing preference values, and exempting certain preferences from monitoring.

3.3. Add-on Implementation

Upon deciding to develop an add-on to implement the proposed solution, choosing the browser (Google Chrome or Mozilla Firefox) is the next task. For selecting the browser, this thesis studied the add-on architecture of the two browsers and came to know that there are some differences in the permissions available to an add-on over preferences. The proposed solution is possible in both of the browsers and initially planned to implement the proposed solution in both browsers. Realizing that developing the same solution in both browsers is meaningless and decided to develop only for one browser. It helps to concentrate more on adding additional features and handling multiple cases if developed for a single browser rather than developing for both of the browsers.

At this point, the browser selection for implementing the proposed solution is still undecided. For selecting the browser, this thesis considered the factors such as which browser has given more importance to the preferences exploitation issue in their security mechanisms and the browser action against unwanted changes currently. Google Chrome gives warning pop-ups when there is change happened without the user notice. Further, Chrome gives information about the software controlling the modified preference. In the case of Mozilla Firefox, there is no such type of warning pop-ups about unwanted modifications.

Based on this information, this thesis selected the browser with lower security mechanisms, so that it would reach a large number of users and help in getting feedback. This thesis selected the browser Mozilla Firefox as it is lacking compared to Google Chrome concerning the above factors.

Development started with studying the basics of add-on development for Firefox. Firefox has a unique add-on architecture which helped in designing the add-on, and many useful APIs are available for add-on developers. After studying the APIs available, an add-on cannot stop changes in preferences. An add-on can only detect the change of any preference, and any further action is not possible to take until a change happens. The first goal in implementing the add-on is handling preference values.

3.3.1. Handling Preferences Values

The add-on needs to have control over preferences values such as reading the present value and modifying the value to a different one. The add-on requires these operations to revert preferences values to the old values upon unwanted changes and set the new values of preferences if the user confirms the new changes. With the preference/service API the add-on can easily access and modify the present values of preferences. Each preference has a name used with the preference API for accessing.

A list of preferences with their values is available on the page with the URL "about:config". On this page, all of the preferences are available with their current values, and it is possible to modify any preference values on the same page. For instance, the name of the preference homepage URL used for accessing is "browser.startup.homepage". The statement used to get the value is

```
     require("sdk/preferences/service").get("browser.startu
p.homepage")
```

and to set the value of the homepage URL to the Google search engine website is

```
     "require("sdk/preferences/service").set("browser.start
up.homepage","http://www.google.com")".
```

Only some of the preferences listed on the page with the URL "about:config" are available in the options page with the URL, "about:preferences". The preferences available on the options page are more understandable as they are visually displayed, and this makes it easy for the user to set the desired values. While considering the page with the URL "about:config" it would be difficult for users to modify the preferences values. Users should have technical knowledge of preferences and preference names to modify on the page "about:config". Even some technical people rarely modify the remaining preferences on the page "about:config" so the add-on will not monitor these preferences.

The add-on does not monitor all of the preferences on the "about:config" page because browser hijackers generally target only the preferences on the options page. The add-on will store all of the current values of monitoring preferences because upon unwanted changes the add-on uses the current values for reverting modified preferences to their previous values.

There is a problem in saving the current values as normal variables. The browser deletes the memory of these variables at the time of shutdown as these are part of temporary storage. The add-on should store these values in permanent memory as it needs these values at the time of browser restarting. In Mozilla Firefox, add-ons can save memory persistently by using the “simple storage API”. The add-on used this API to save the current values of preferences so that they do not vanish after a browser shutdown.

The next step is to write action listeners for each preference to trigger after a change. Add-on developers can use the Preference/event-target API write on-change event listeners for preference values. Developers uses the preferences names from the page "about:config" for writing listeners.

The preference/event-target API cannot monitor the preference search engine For the current search engine, the add-on used a normal event observer to observe changes in this preference. After a modification trigger, the add-on needs to know whether there is a need to authenticate or simply save the new change. The following sections discusses briefly the authentication method used in this thesis.

3.3.2. Authentication Methods

Throughout this document, this thesis mentions that the add-on will save the unwanted modifications only with the user confirmation. User confirmation means the user should give some input to the add-on that they agree to the new changes. One such type of input is to click the button “okay” in the pop-ups that appear. If the user needs to click only the "okay" button for confirmation, then they may simply click the "okay" button without reading the description in the pop-up window. Users may also click the "okay" button unintentionally while doing some work. Some users might also click the "okay" button in the thought of general security measure. Considering the above factors, this thesis added authentication in the add-on for the user confirmation.

The user needs to enter a password for saving changes or click cancel for discarding new changes. In the authentication process, the user cannot click cancel in the thought of a general pop-up. There will be no mistaking the of pop-up as a general security windows since the user needs to enter a password. In this process, the user will read the pop-up window description to know what is the purpose of the authentication. The following paragraph discusses the authentication method used in this add-on.

For authentication, this thesis considered a few different methods. One way is to authenticate by using the operating system password. In this authentication method, the operating system password pop-up window appears for the user’s confirmation. There are some problems with this authentication method such as that the process of integrating the operating system password is difficult, and the add-on must deal with a lot of security barriers which could affect add-on performance severely. This method is also operating system specific, and the developers must develop the add-on for each operating system and each browser separately, which requires a lot of time. With these problems, this thesis did not consider this authentication method.

This thesis searched for an authentication method with factors such as built into the browser, the add-on has accessibility to the authentication method, and the method is not unique to an operating system. The master password in Mozilla Firefox is one of the authentication methods satisfying the above conditions. The master password is built into Firefox and provides authentication for viewing saved logins. If the user uses the master password in the browser, then for viewing saved logins they should enter the password. Master Password is very secure, ease to access and not operating system specific. These are advantages of using the master password authentication. However, if the user does not want to use the master password for any reasons then the add-on cannot use this authentication method.

This thesis also considered an alternative method, that is separate authentication used in this add-on only. For ease of use and implementation, this thesis designed a simple authentication method and called it separate authentication. However, there are a few security issues with implementing the add-on's own authentication. The problem with this approach is storage of the password on the local system. This thesis considered Firefox password manager for storing the password in this method, which is very secure and protected. The problem of storing a password in the password manager is, the password is visible to the browser user in saved logins. If the user is not using the master password, then there is no barrier to viewing saved logins and the password of this add-on is also viewable in the password manager. For better security, this method requires hashing of the password.

Hashing is converting a string of characters into a fixed length of characters. For password hashing, this thesis considered different hashing techniques. This thesis considered only hashing techniques rather than encryption and decryption because encryption techniques require a private key. It is not possible to store the key securely in the add-on so this thesis considered keyless hashing techniques only. This thesis considered few popular hashing techniques: MD5 (Message Digest), SHA-1 (Secure Hash Algorithm), and SHA-2. Any of these hashing techniques would serve the purpose. Gupta (2014) describes that SHA algorithms should be given importance than MD5 in terms of security and performance. Among different SHA algorithms with various digests, this thesis uses SHA-2 with a digest of 256 because a higher digest like 512 would take more time to calculate the hash and lower digest are less secure and high chances of collisions occurring.

It is possible to decode the hash obtained with the SHA – 2 hashing technique using tools available in online. To overcome this problem, this thesis added salt to the password string before hashing. Salt hashing is adding random text to the password before hashing to make decoding of the hash more difficult. By adding salt to input, it will be difficult to decode the hash generated with available online tools. In the implementation, the add-on adds the salt text to the password text before hashing the password. Online tools available to decode the hash are unable to decode the hash obtained after adding salt. This thesis's main intention is not authentication, so this thesis did not concentrate on hashing and authentication in depth. This thesis focused on preferences modifications and different ways of exploiting preferences rather than authentication mechanisms.

Master password authentication and separate authentication of this add-on meet this thesis's design requirements. Each authentication method has some advantages and disadvantages. Without compromising, this thesis implemented both of the authentication methods. Users can select the method of authentication per their need and ease of use.

The flow chart in Figure 2 explains the installation steps performed in the add-on. Based on whether the user is using Master password or not, the first pop-up will appear. It

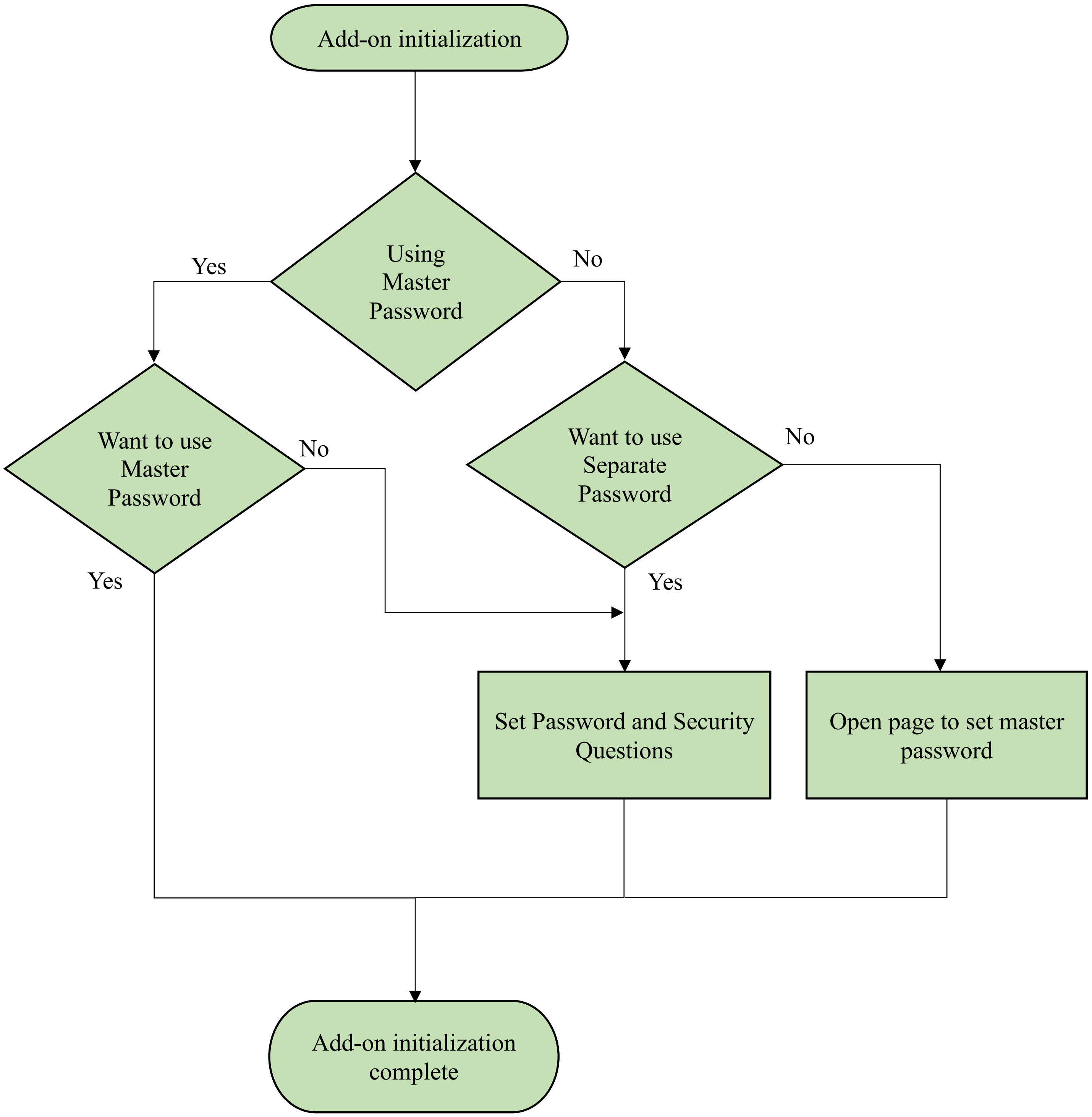

Add-on initialization
Using Master Password
Yes
No
Want to use Master Password
No
Yes
Want to use Separate Password
No
Yes
Set Password and Security Questions
Open page to set master password
Add-on initialization complete

Figure 2. Add-on Installation Flowchart

is their choice to select the type of authentication. For separate authentication, users need to enter the password and select & answer security questions. For the master password, they need to set the master password in the page with the URL "about:preferences#security" which will open automatically upon their selection. The add-on will save the current values of the preferences, and there is no need of user interaction for this step. With this step, the add-on installation process is complete, and the add-on listens for new changes.

3.3.3. Detecting a Non-user Change

To completely prevent changes, the add-on needs to authenticate users, even if they changed preferences on their own. If the add-on authenticates every change, the user needs to enter their password every time they try to change preferences, and it is not user-friendly. The goal of this thesis is to stop changes that will occur without the user’s notice and the solution provided should not interrupt changes made by the user.

After detecting a preference modification, the add-on needs to classify it as a user change or a change that occurred without the user’s notice. A user change means the user changed the preference value and a non-user change means the user did not change the preference value. For this classification, this thesis gathered information about different ways the user can change preferences. The following are the ways that the user can change preferences:

1. Changes in preferences/options page,
2. An add-on installed can modify the preferences,

3. Changes on the page with the URL "about:config," and
4. Changes in preferences through the locally stored preference file “preference.json.”

This thesis had considered only the first factor that is “Changes in preferences/options page” and discarded the other factors for classification. The reason for this is an add-on can install silently without the user’s permission and modify preferences. In some cases, the developer intentionally does not mention changes in the add-on description. If the user installs these type of add-ons, these add-ons modify the preferences without the user’s permission.

For the next factor, changes in page "about:config" and to modify preferences on this page, the user should have some technical knowledge. They need to have knowledge of the preferences name to search and modify on this page. It is difficult for non-expert users to change the preferences on this page, so this thesis did not consider this factor.

Browser hijackers can also manipulate the final factor as they can modify the contents of the locally stored preference file. However, this thesis did not consider the final factor for classification. In designing the add-on, this thesis considered only the first factor for differentiating between user changes and non-user changes.

When the add-on detects a change in a preference value, then it will check URL of the browser active tab. If the page is the options tab, that is the URL starting with "about:preferences", then the add-on considers the change a user change. Otherwise, the add-on considers the change a non-user change.

3.3.4. After Detecting a Change

Whenever there is a change, first the add-on should classify it as either a user change or a non-user change. If it is a user change, then the add-on needs to consider some situations based on the time of change. If there was any other add-on installed recently before the modification, then the add-on will authenticate the changes. The Settings protection add-on will observe three seconds of time from other add-on installations. If there is a preference change within that three seconds of time, then the add-on will also authenticate that change irrespective of the browser's active tab page. It is not often that user will modify the preferences within three seconds of installing any add-on themselves.

Clearly explaining, let us assume that the user installed an add-on not related to preferences. After installing the add-on, the user modified some preferences. There would be an average time gap between the two operations which would be at least three seconds. This research estimated this time based on experiments and careful observations. If there any changes in preferences within this three seconds of time, then the add-on will consider these changes as non-user changes irrespective of the browser's active tab page.

If the user enables the self-preference advanced monitoring, then the add-on will authenticate the change. If there are no such situations at the time of the preference modification, then the add-on will simply saves the new value of the particular preference for future references. Figure 3 illustrates this flow of this process.

After detecting a non-user change, the add-on will check its self-preferences to determine whether to discard or save the new changes and whether to authenticate or not. If the user enables related "exempt preference monitoring", then the add-on will save the

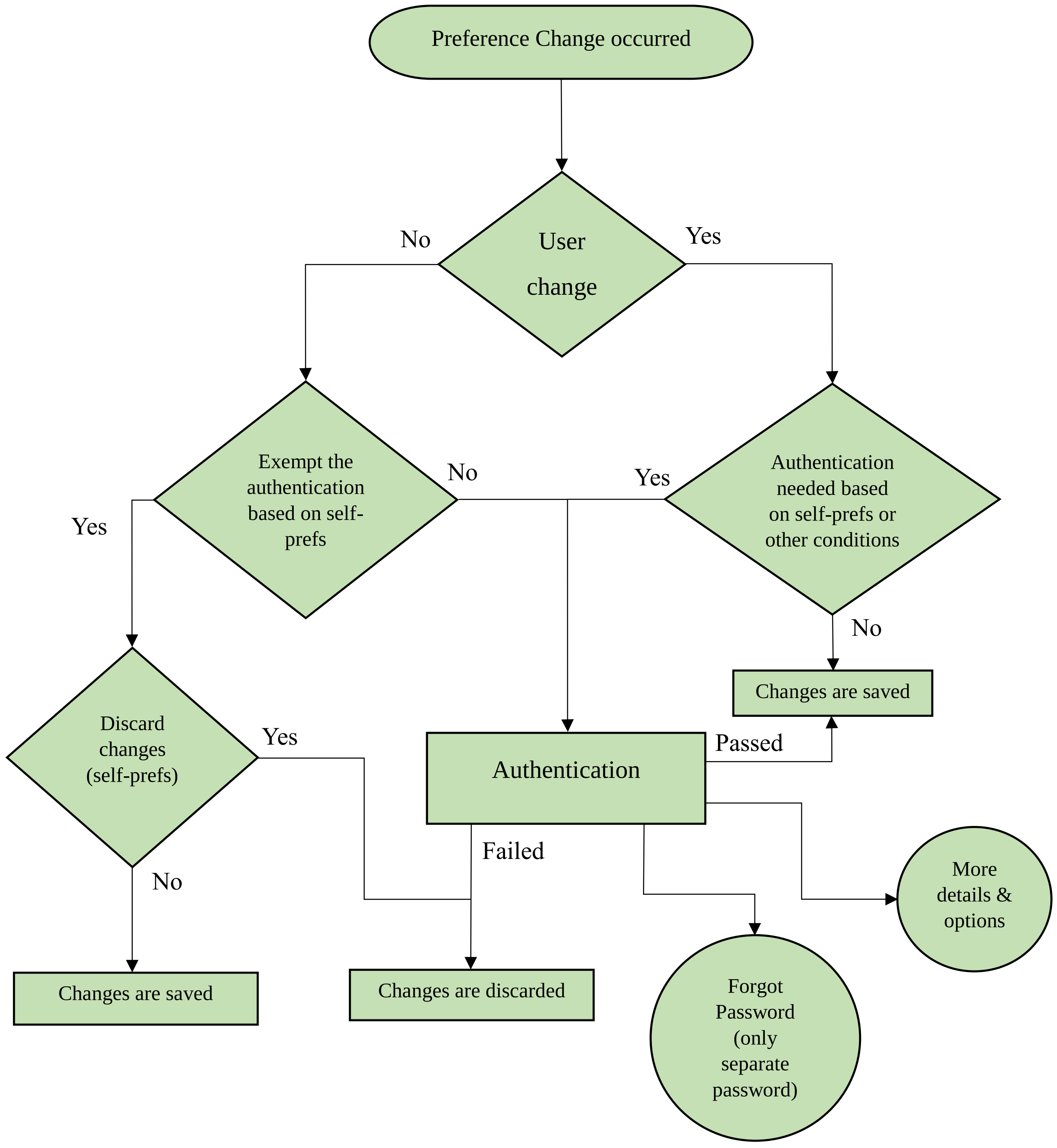

Preference Change occurred
User change
No
Yes
Exempt the authentication based on self-prefs
No
Yes
Authentication needed based on self-prefs or other conditions
Yes
No
Changes are saved
Discard changes (self-prefs)
Yes
Authentication
Passed
Failed
No
More details & options
Changes are saved
Changes are discarded
Forgot Password (only separate password)

Figure 3. Flowchart After a Change Occurred.

preferences without the authentication. If the user enables the preference never “save non-user changes”, then the add-on will discard the new changes. If there are no such related self-preferences enabled, the add-on will immediately roll back the preference value to the previously stored value. Now, the add-on will show a pop-up window with the user selected authentication.

It is the user’s choice whether to save the new changes by passing the authentication or discard the new changes by clicking the button "cancel." In the case of separate authentication, the user has three chances to pass the authentication. After failing three times, the add-on will discard the new changes automatically. If the selected authentication is master password, then there is no limit on authentication failure times. The only option the user will have to discard new changes is to click the "Cancel" button in the case of the master password. If the user wants to know more details about the changes, then they should click the button "More options & details." The flowchart in Figure 3 explains the flow of a process in the add-on after detecting a change.

The add-on handles the case where there is a group of changes that occurs at a single time. The add-on will authenticate only once for all of the preferences modified at one time. The user can simply save all preference changed at one time with a single authentication or discard all of the preferences by clicking cancel. By handling this case, the add-on will not irritate the user with multiple pop-up windows.

When the user clicks the "Cancel" button, a small pop-up will appear showing “Discarded the new changes.” The following sections describe what will happen when the user clicks “Forgot password” and “More details & options.”

3.3.5. Forgot Password

In the case of the master password, if the user forgets the master password, then they should use the master password reset process. The add-on will discard the current changes as the user needs to click the "Cancel" button in the authentication's pop-up. The add-on has nothing to do with the update of the master password as it is a built in feature controlled by the browser.

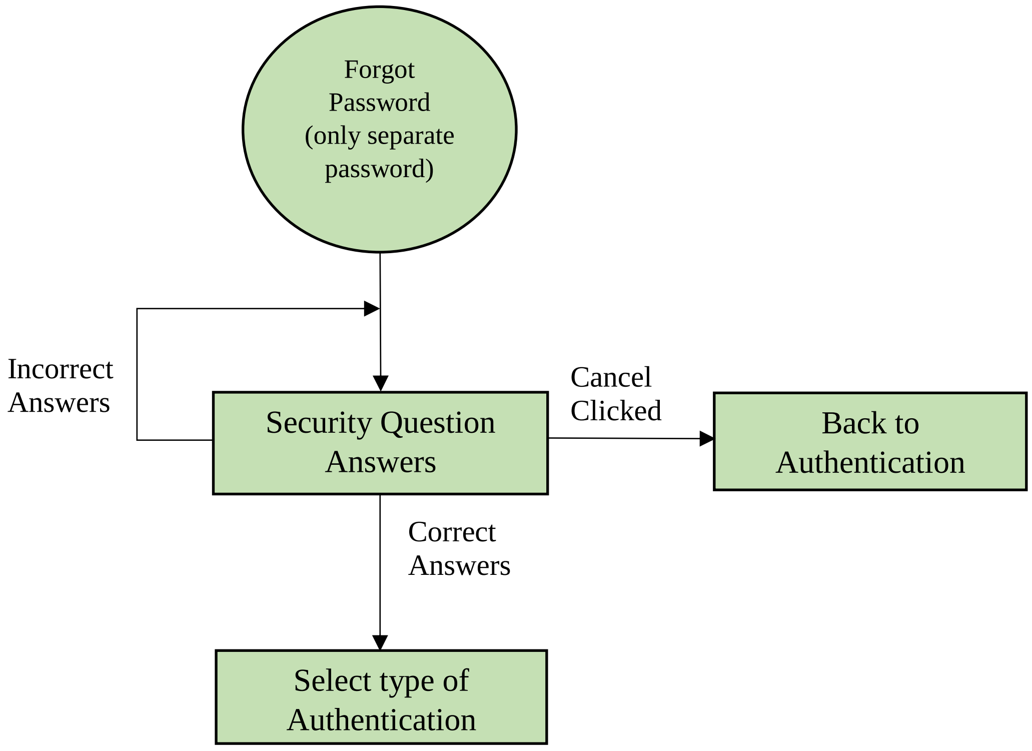

Forgot Password (only separate password)
Incorrect Answers
Security Question Answers
Cancel Clicked
Back to Authentication
Correct Answers
Select type of Authentication

Figure 4. Flowchart for Forgot Password.

For the case of separate authentication, the user needs to click the forgot password button in the authenticate pop-up window. When the user selects separate authentication at the time of the add-on installation, the add-on will ask the user to select two security questions and give answers to them along with password set-up. These security questions are useful if the user forgets the password. When the user clicks “Forgot Password,” the add-on will ask them to enter the answers to selected security questions. If the user gives correct answers, the add-on will discard the preferences changes and ask them to choose one of the authentication methods.

While in the forgot password pop-up, if the user remembers the password and wants to go back to authenticate window, then they can click cancel. The flow chart in Figure 4 gives the flow of the process in this step.

3.3.6. More Details & Options

In the authentication window, the add-on will display only the names of the preferences changed. If the user wants to know more details about the changed preferences, then they should choose “More details & options.” The first part of this window contains the name of the preference changed, along with the old and new values. With the help of these details, users having knowledge about these preferences will know about the changes happened. If more than one preference changed, this part of the window will display all of the changed preferences along with the new and old values.

The next part of this window contains the name of preferences changed with check boxes. Initially, the add-on selects these checkboxes. If the user wants to save only some of the preferences changed and ignore the remaining, then they should deselect the check boxes corresponding to the undesired preferences. This part of the window helps to select only the new changes the user is expecting and desires. This could also protect the user from an incorrect description of the add-on. For instance, the "Search and Newtab by Yahoo" add-on description says it modifies search engine and new tab only, but it also modifies the homepage URL. In this case, this feature will be handy in avoiding the part of the changes that the user is not expecting. If the number of changed preferences is one, then the add-on will not display this section in the pop-window.

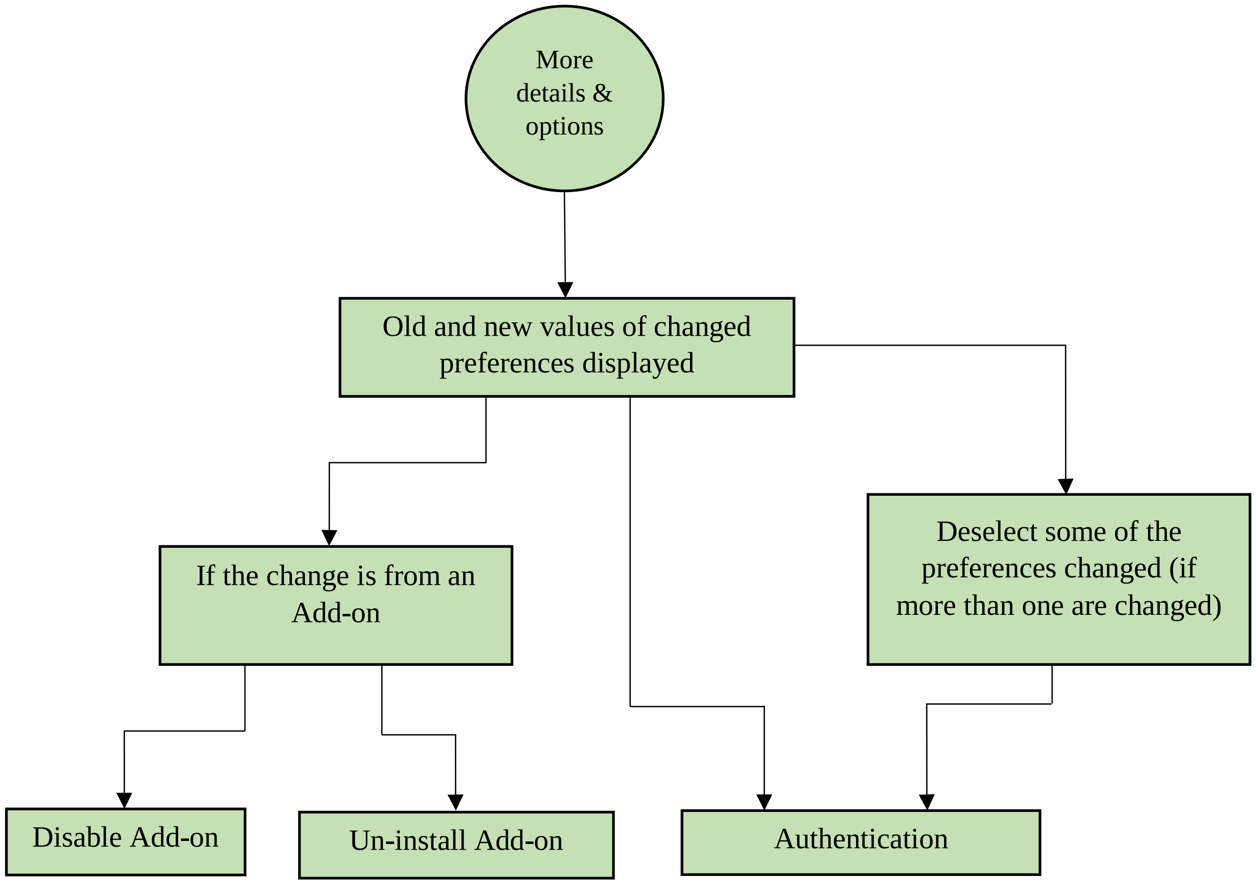

More details & options
Old and new values of changed preferences displayed
If the change is from an Add-on
Deselect some of the preferences changed (if more than one are changed)
Disable Add-on
Un-install Add-on
Authentication

Figure 5. Flowchart for More Details & Options.

The final part of this window is to show the source of modifications to the preferences. It is not possible to tell the exact source of modifications in Mozilla Firefox through an add-on. If a change has occurred immediately after an add-on installation, then there is only a *possibility* that the add-on changed the preferences. It might not be the source of modification exactly. This part of the window will display the add-on name and tell the user it might be the reason for the modification. The user can decide whether the add-on is familiar or not by its name. If the user installed that add-on, then it will be familiar to them. If it is not familiar and the user believes it is the source of the modifications, the user has the chance to disable or uninstall the add-on.

The add-on displays "Disable" and "Uninstall" buttons at the bottom of the pop-up window. If the user selects disable, then they can re-enable the add-on after knowing more information about the add-on. In this way, this part of the add-on displays the source of the modifications though the add-on cannot know the exact source of modification for sure. The flow chart of Figure 5 gives the flow of the process for this step of the add-on.

3.3.7. Extra Features

The add-on provides additional features through self-preferences. The second chapter discusses the self-preferences details of an add-on and declaration in detail. As part of the add-on, some specific extra features and flexibility are available to the user. The user can enable/disable any of these features only with the selected authentication to restrict unauthorized access. There is a self-preference to change the authentication method between master password and separate authentication.

Some users frequently change preference values, and they may change preferences on their own or through some other way. The add-on will provide users a choice to skip listening to changes on the particular preference category. For instance, if they do not want the add-on to respond to changes of the homepage preference, they have a self-preference to enable/disable accordingly. For the search engine, new tab URL and homepage URL separate self-preferences are available for exemption from monitoring. The remaining preferences fall into the category of preferences for exemptions. Suppose if the user wants to exempt “Block pop-up preference” from monitoring, then they should enable the security group of preferences exemption from monitoring.

Another feature available to the user is to lock the options page and open the options page only with authentication. Enabling this option requires the user to enter their selected authentication password every time they want to access the options page so that their friends or children cannot have access to that page. This feature is similar to a parental control over browser preferences.

Another feature is to discard non-user changes completely. By selecting this option, the user will not get any notification pop-up if the change is a non-user change. The add-on will simply roll back to the old values and discard the new changes.

Another feature is allowing changes from the add-ons downloaded from the official Mozilla Firefox add-on store. If the user enables this option, whenever there is a change from an add-on downloaded the from official Mozilla website, then the add-on saves the new changes without any pop-up.

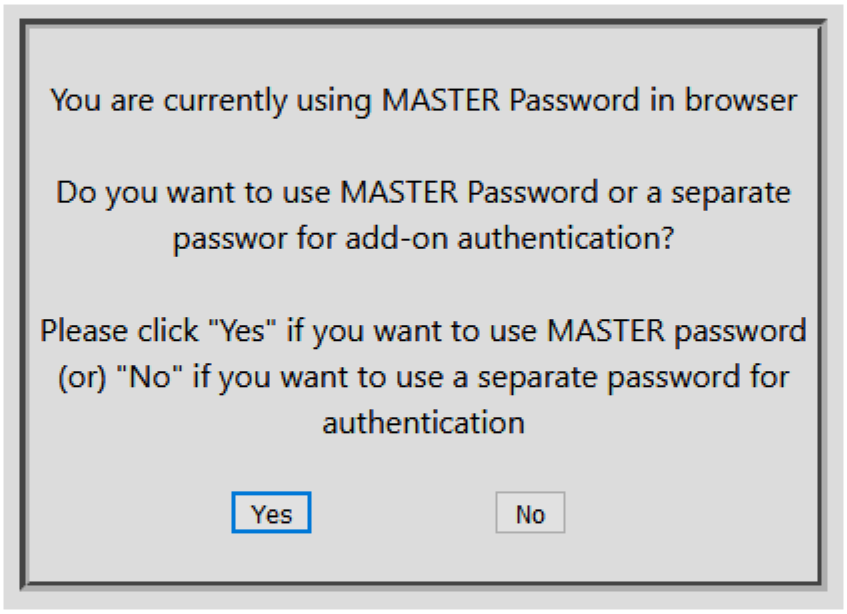
If the user wants to authenticate every change in preferences including changes made on the preferences page with the URL starting "about:preferences," then users should enable the self-preference “Advanced Monitoring.” This feature provides additional security to the preferences by authenticating every modification. This feature can be useful for a public computer where more people are using the computer such as library computers in universities.

All of these features are available as add-on self-preferences and visible below the add-on description available on the page with the URL, "about:addons". The user can open the options page simply by clicking the add-on button in the toolbar. By clicking the add-on button in the browser toolbar, the add-on opens the options page in a new tab and the user can simply access the self-preferences and make changes as per their requirement.

4. Evaluation and Analysis of the Add-on

4.1. User Interaction of the Add-on

The following pop-ups will appear immediately after the add-on is installed. These are part of the add-on initialization set-up. If the user is using the master password, then the pop-up in Figure 6 will appear.

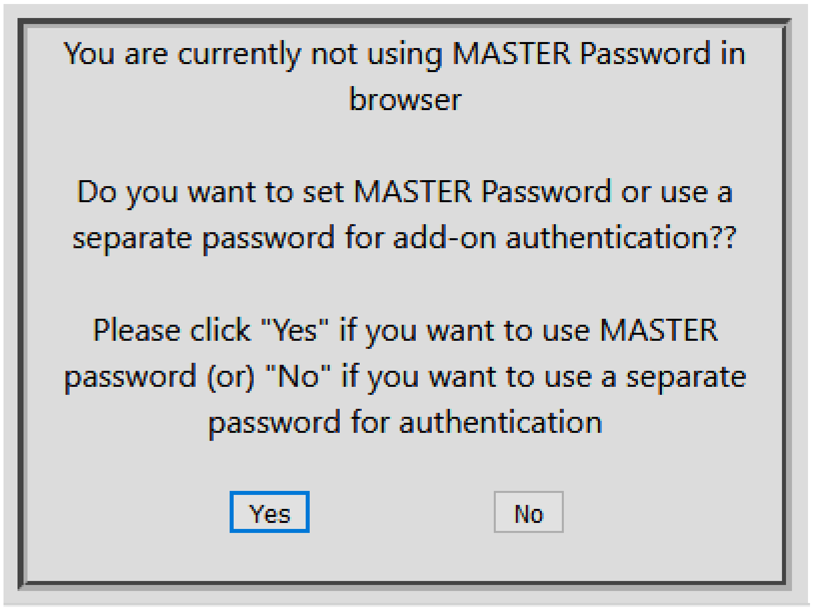

You are currently using MASTER Password in browser

Do you want to use MASTER Password or a separate passwor for add-on authentication?

Please click "Yes" if you want to use MASTER password (or) "No" if you want to use a separate password for authentication

Figure 6. Using the master password Pop-up

In the above figure, if the user clicks "Yes," then the add-on uses the master password for authentication and with this step the process of add-on initialization will be complete, and the add-on begins listening for new changes. If the user clicks "No," then the pop-up in Figure 8 will open for password entry. If the user is not using the master password, the pop-up in figure 7 will appear instead of the pop-up in Figure 6. In the Figure 7 pop-up, if the user decided to use the master password, then the "about:preferences/security" page is opened in new tab. In the opened page, the user needs to set the master password. If a change occurs now and the user still did not set the master password, then the pop-up in Figure 7 will appear again to select the type of authentication. If the user selects "No" in the pop-up of Figure 7, then the pop-up in Figure 8 will appear to set up separate authentication.

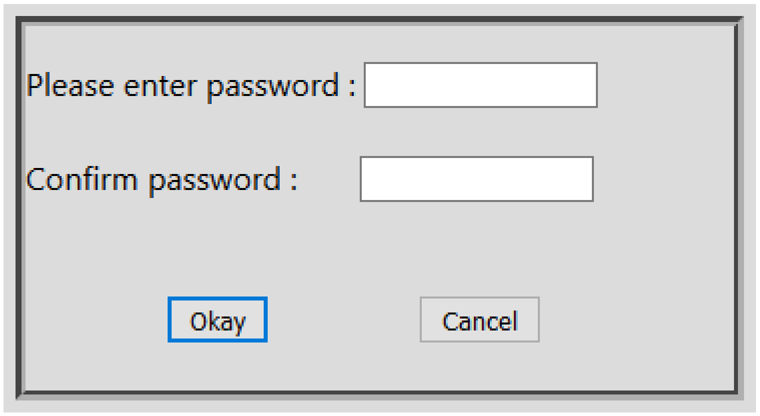

You are currently not using MASTER Password in browser

Do you want to set MASTER Password or use a separate password for add-on authentication??

Please click "Yes" if you want to use MASTER password (or) "No" if you want to use a separate password for authentication

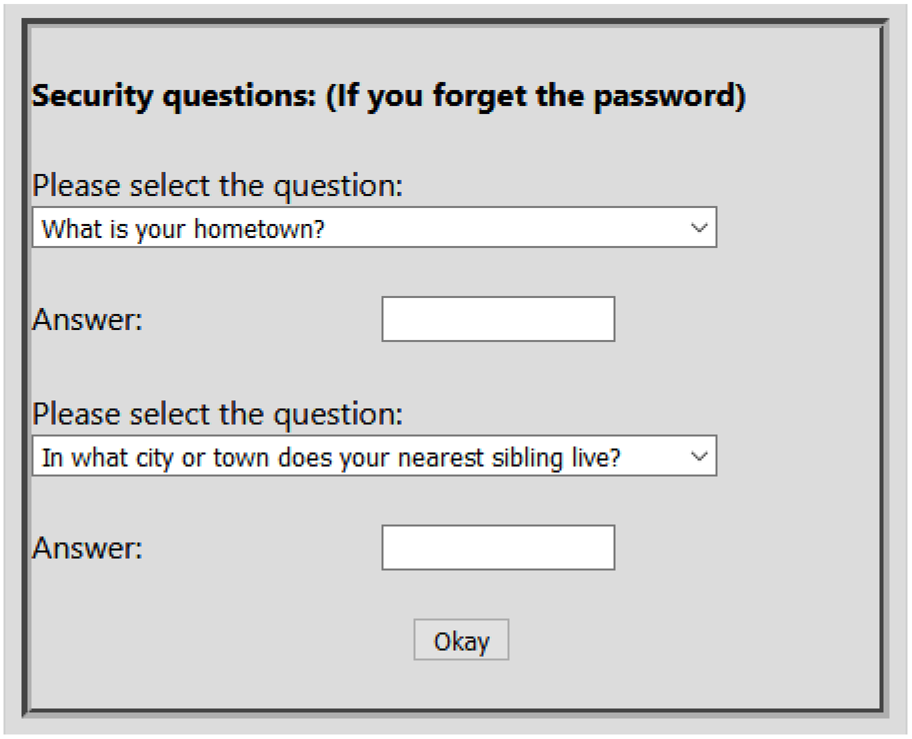
Figure 7. The User is Not Using the Master Password

The user needs to enter a password for separate authentication. In this pop-up, the user needs to enter the same password for both fields. If the users enters different passwords and clicks "okay," the same pop-up will appear again. If the text in both of the fields is non-empty and the same, then the pop-up in Figure 9 will appear. If the user clicks "cancel" in Figure 8, then the add-on initialization step will start again.


Please enter password :

Confirm password :

Figure 8. Separate Password Entry


Security questions: (If you forget the password)

Please select the question: What is your hometown?

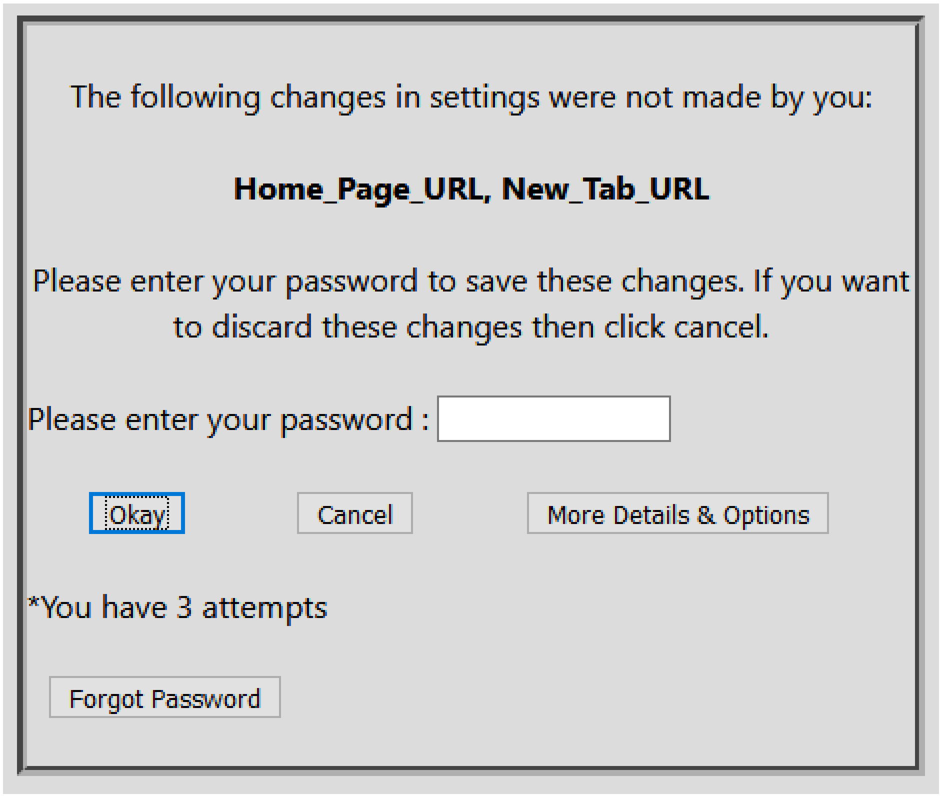
Answer:

Please select the question: In what city or town does your nearest sibling live?

Answer:

Figure 9. Security Questions

In the case of separate authentication only, the pop-up in Figure 9 appears, and the user needs to select the questions and give answers to them. If the user forgets their password, then the user must provide correct answers to these questions to change the password or select a new type of authentication. The user should choose different questions and give different answers. Otherwise, the same pop-up will repeat to select questions and give answers. If the user clicks "okay" after selecting different questions and giving different answers, then the process of add-on initialization is complete and the add-on begins listens for new changes.


The following changes in settings were not made by you:

Home_Page_URL, New_Tab_URL

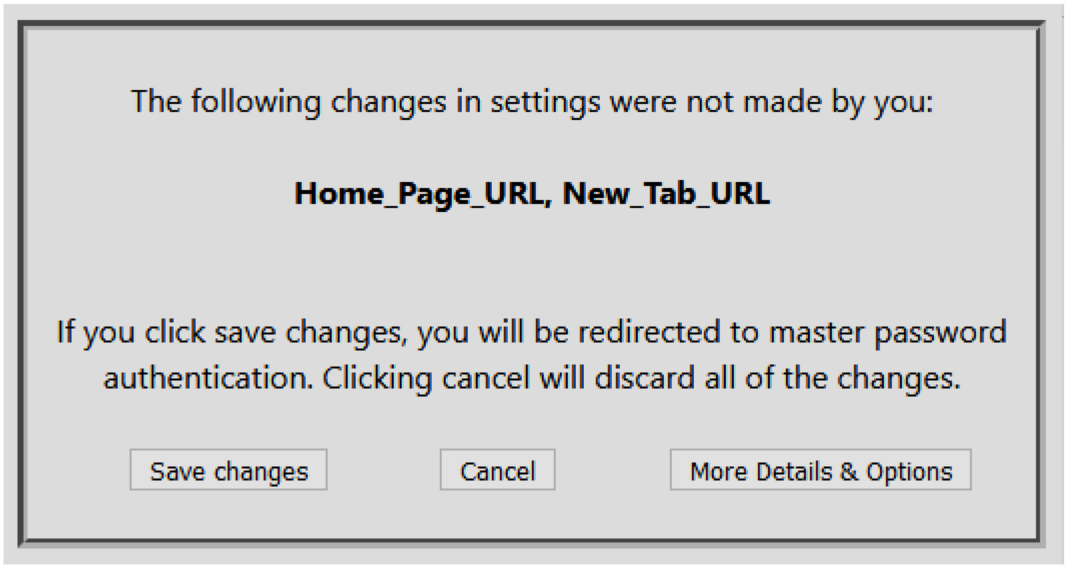
Please enter your password to save these changes. If you want to discard these changes then click cancel.

Please enter your password :

*You have 3 attempts

Figure 10. Separate Password Authentication

The pop-up in Figure 10 will appear for separate password authentication when a change occurred that requires authentication. Installing the add-on "Search and Newtab by Yahoo" modifies two preferences in the browser. The pop-up contains the names of the preferences changed. In this pop-up, if the user clicks "okay" with an incorrect password, then this pop-up will appear again. The user has three chances to enter the correct password, and after the third incorrect attempt the add-on discards the new changes automatically. If the user clicks "cancel," the add-on discards the changes. If the user wants to know more about the changes and other options, then they should click the "More Details and Options" button. If the user forgets their password, then they should click the "Forgot Password" button.



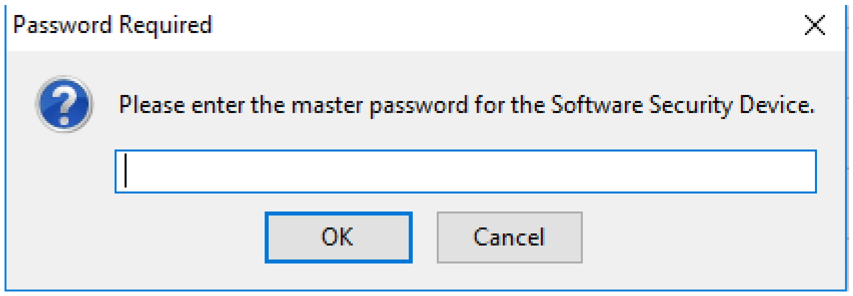
The following changes in settings were not made by you:

Home_Page_URL, New_Tab_URL

If you click save changes, you will be redirected to master password authentication. Clicking cancel will discard all of the changes.

Figure 11. Details of Modifications (Master Password Only)

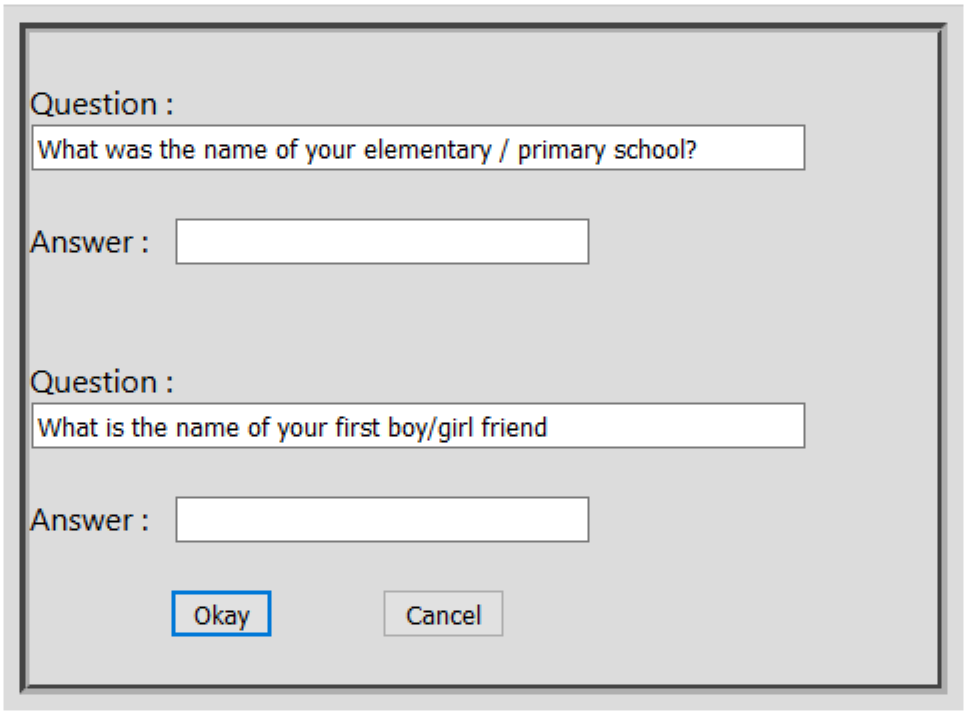
The pop-up in Figure 10 is for separate authentication only. If the user selected master password as the authentication method, then the pop-up in Figure 11 will appear instead of Figure 10. Similar options are available in both Figures 10 and 11. The only difference is, in the case of master password authentication, the user should click "save changes" to save the changes. The "Cancel" and "More details & options" buttons will do the same thing, in both types of authentication. In Figure 11, upon the user clicking "Save Changes" button, the pop-up in Figure 12 will appear for authentication. After clicking the "Save changes" button, the user should enter their master password to save the new changes. If the user clicks "Cancel," then the add-on will discard the new changes. The process of master password authentication is in two steps rather than one step in the separate authentication process.


Password Required

Please enter the master password for the Software Security Device.

Figure 12. Master Password Authentication

In the case of separate authentication only, if the user clicks "Forgot password" then the pop-up in Figure 13 will appear. If the answers are incorrect, then the same pop-up repeats. If the answers are correct, then either the one pop-up in Figure 6 or Figure 7 will be shown according to whether the user is using the master password or not. If the user clicks "Cancel," then the add-on displays the pop-up in Figure 10 which is the separate password authentication panel.


Question :

What was the name of your elementary / primary school?

Answer :

Question :

What is the name of your first boy/girl friend

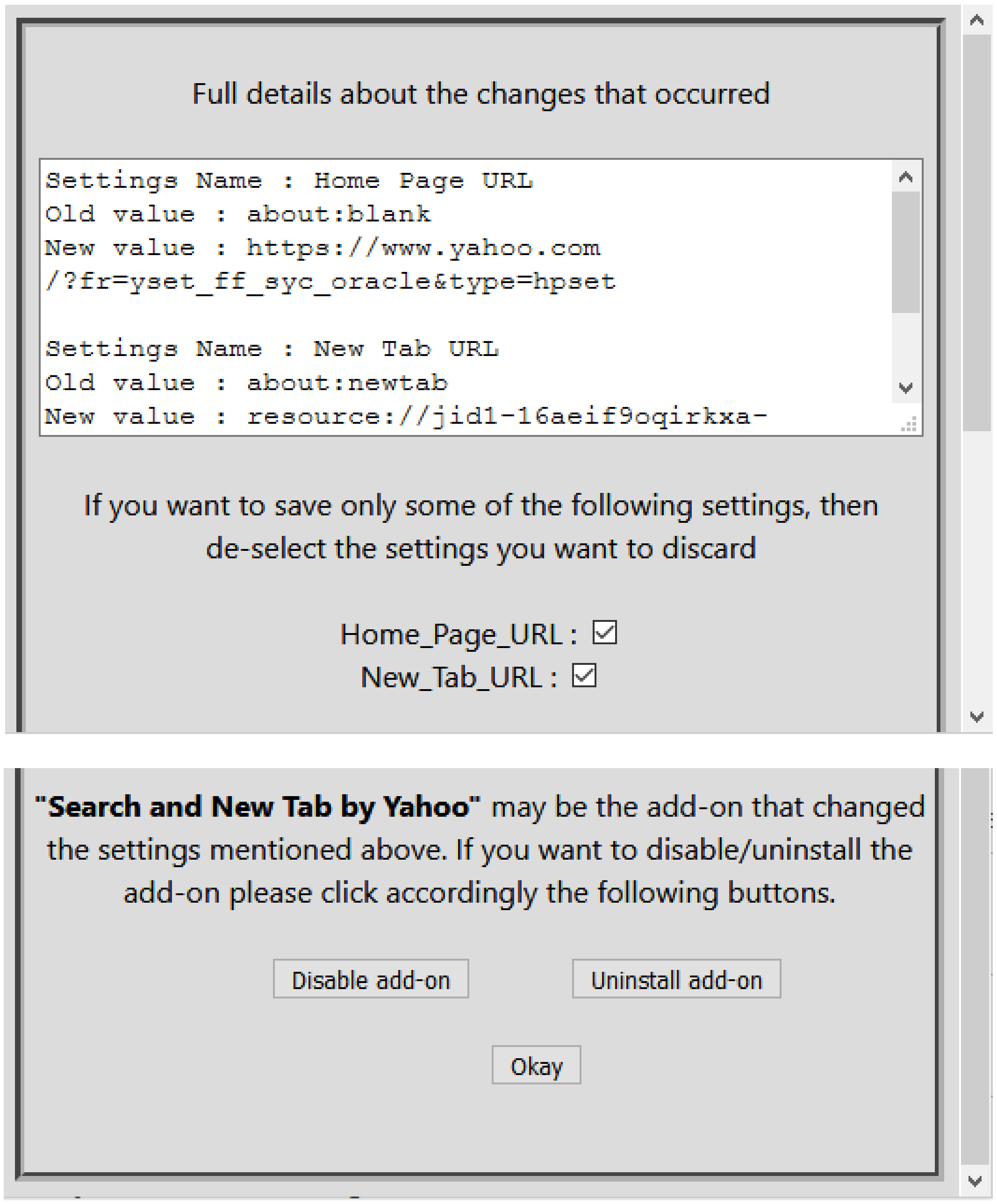
Answer :

Figure 13. Security Question Answers

Upon clicking "More options and details," then the pop-up in Figure 14 will appear. It contains the details of the changes such as preference name and the old and new values of the changed preferences. If the number of modified preferences is more than one, this part of the window contains the list of preferences changed with check boxes.

If the user want to discard some of the modified preferences, then they should de-select them and click okay. The add-on then goes back to the authentication panels in either Figure 10 "Separate Password Authentication" or Figure 11 "Master Password Authentication."

The remaining part of the pop-up in Figure 14 contains the suspected source of the preferences modification. In the above pop-up, it mentions “Search and New tab by Yahoo” might be the add-on that changed the preferences. If the user clicks "Disable add-on" or "uninstall add-on" then the add-on performs the corresponding action. Clicking any of these buttons also discards the new changes.



Full details about the changes that occurred

```
Settings Name : Home Page URL
Old value : about:blank
New value : https://www.yahoo.com
/?fr=yset_ff_syc_oracle&type=hpset

Settings Name : New Tab URL
Old value : about:newtab
New value : resource://jid1-16aeif9oqirkxa-
```

If you want to save only some of the following settings, then de-select the settings you want to discard

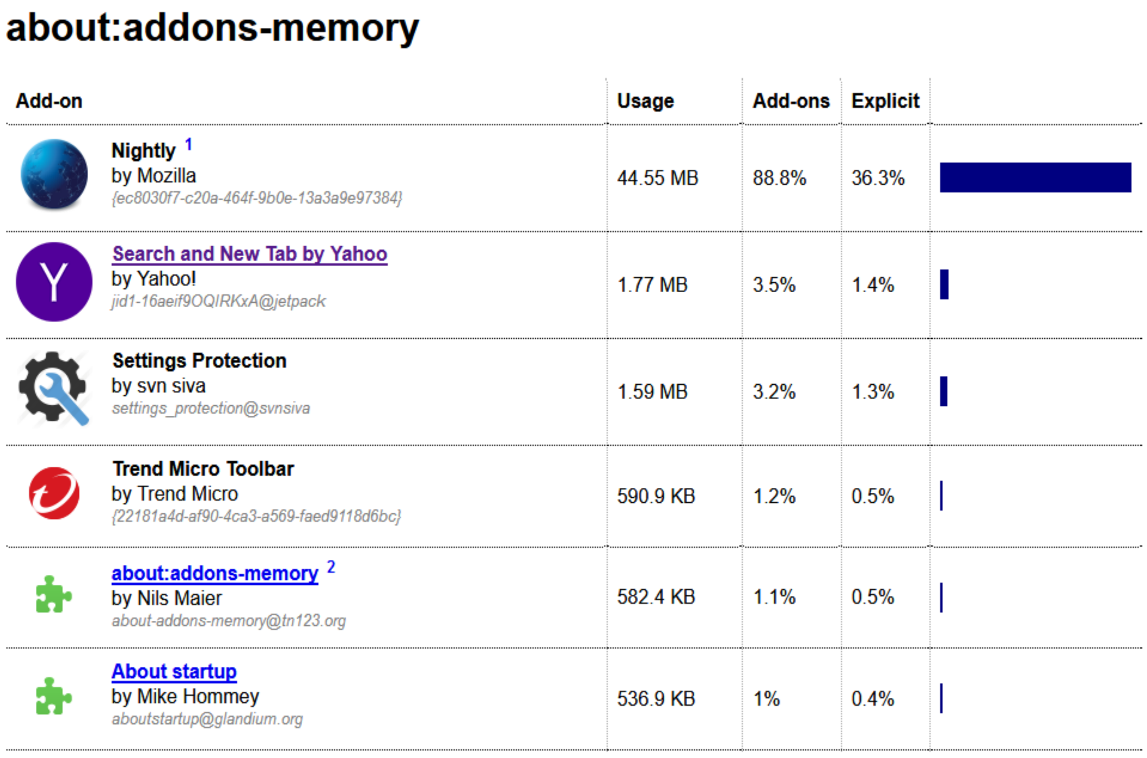
Home_Page_URL : ☑
New_Tab_URL : ☑

"Search and New Tab by Yahoo" may be the add-on that changed the settings mentioned above. If you want to disable/uninstall the add-on please click accordingly the following buttons.

Figure 14.More Options and Details

4.2. Performance of the Add-on

An add-on means providing additional features and extending the functionality of the browser. This customization and additional features come at the cost of degrading the performance of the browser. Clearly, by installing any add-on, the browser performance is degraded, and the amount of impact depends on the particular add-on. If the add-on is a heavy add-on with more operations, then the browser performance is more degraded compared to an add-on which performs fewer operations. Figure 15 displays the memory used by the add-on. This thesis used the software "about:addons-memory" for obtaining the results.



about:addons-memory

Add-on	Usage	Add-ons	Explicit
Nightly [1] by Mozilla {ec8030f7-c20a-464f-9b0e-13a3a9e97384}	44.55 MB	88.8%	36.3%
Search and New Tab by Yahoo by Yahoo! jid1-16aeif9OQIRKxA@jetpack	1.77 MB	3.5%	1.4%
Settings Protection by svn siva settings_protection@svnsiva	1.59 MB	3.2%	1.3%
Trend Micro Toolbar by Trend Micro {22181a4d-af90-4ca3-a569-faed9118d6bc}	590.9 KB	1.2%	0.5%
about:addons-memory [2] by Nils Maier about-addons-memory@tn123.org	582.4 KB	1.1%	0.5%
About startup by Mike Hommey aboutstartup@glandium.org	536.9 KB	1%	0.4%

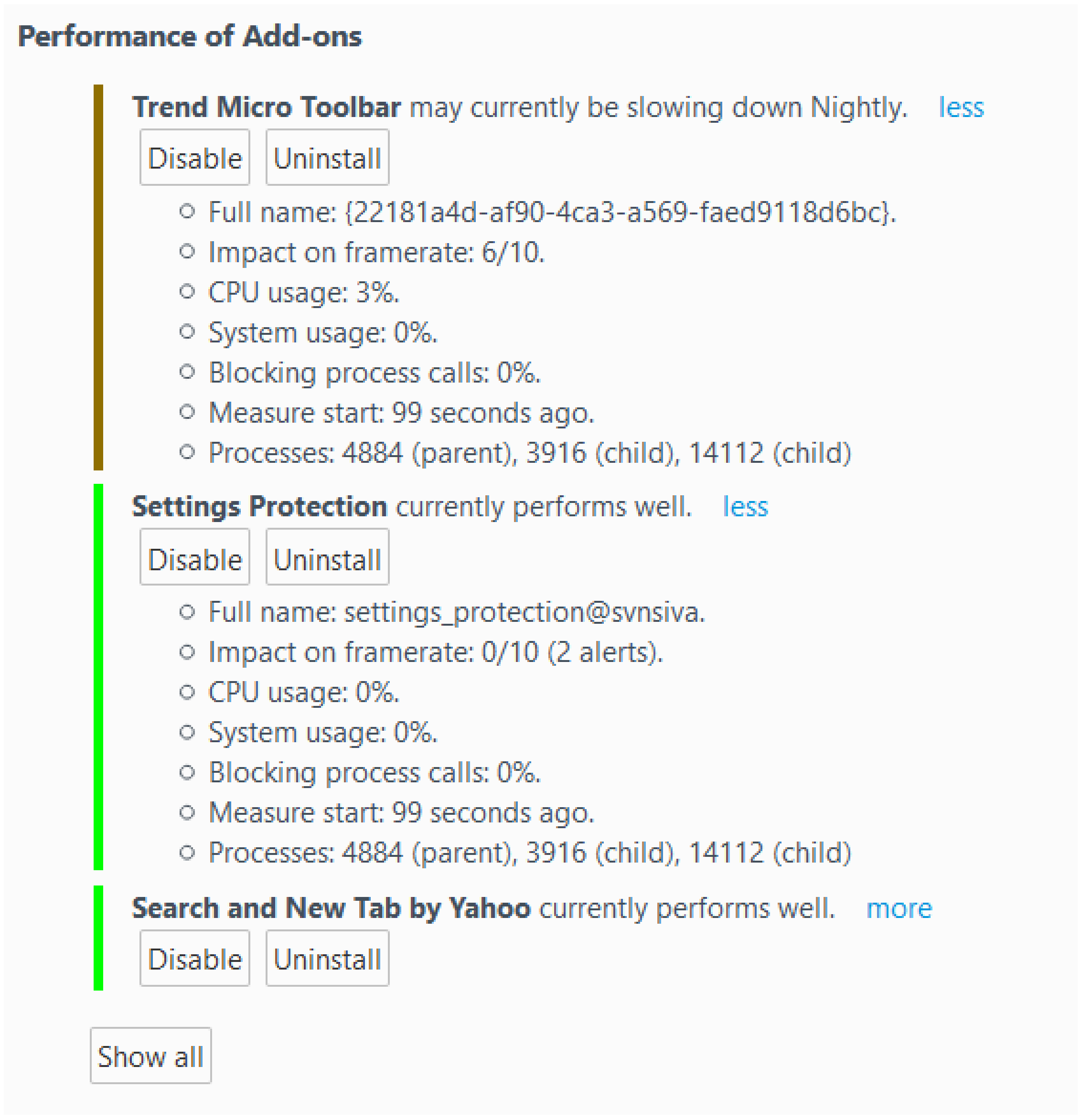
Figure 15. Memory Used by the Add-on

If there are any memory leaks in the add-on, then add-on memory will increase constantly. The main intention of this analysis is not about how much memory the add-on is using but for testing memory leaks in the add-on. Using this software, this thesis checked the add-on for memory leaks and the results show that the memory occupied by the add-on is constant without increasing randomly. The memory used by the add-on might change based on browser start-up or user profile, but it is does not constantly grow. This analysis demonstrates there are no memory leaks in the add-on.

As part of the add-on performance analysis, this thesis observed the startup time of the browser several times before and after installing the add-on. There is not much difference between the results with and without the add-on. The impact on the performance of the browser due to the add-on is very low and is not very noticeable. This thesis cannot report the exact results of the startup time because Mozilla Firefox health report only provides a "startup time graph." This graph plots the average start-up time, and there is no way to get exact startup time. If exact startup times were available, then it would be easier to analyze the add-on’s impact on browser start-up time.

This research tested the add-on in the browser nightly for obtaining the performance results. Mozilla provides the browser nightly for testing new features in Firefox. Another type of analysis given by Mozilla Firefox over add-ons is the performance of the add-on. This tool gives the performance of add-ons by showing whether a particular add-on is working well or slowing down the browser. Figures 16 and 17 are screenshots related to the performance of the add-on.

Figure 16 is the screenshot while the add-on is idle, which means the add-on is not performing any operation in the browser. In Figure 16, it shows that the add-on is working well with 0% CPU usage and 0% system usage. When there are no changes in preferences, the add-on will not affect browser performance. If the add-on occupies more CPU cycles, then it will show "The add-on is slowing down the browser."



Performance of Add-ons

Trend Micro Toolbar may currently be slowing down Nightly. less

Disable Uninstall

- Full name: {22181a4d-af90-4ca3-a569-faed9118d6bc}.
- Impact on framerate: 6/10.
- CPU usage: 3%.
- System usage: 0%.
- Blocking process calls: 0%.
- Measure start: 99 seconds ago.
- Processes: 4884 (parent), 3916 (child), 14112 (child)

Settings Protection currently performs well. less

Disable Uninstall

- Full name: settings_protection@svnsiva.
- Impact on framerate: 0/10 (2 alerts).
- CPU usage: 0%.
- System usage: 0%.
- Blocking process calls: 0%.
- Measure start: 99 seconds ago.
- Processes: 4884 (parent), 3916 (child), 14112 (child)

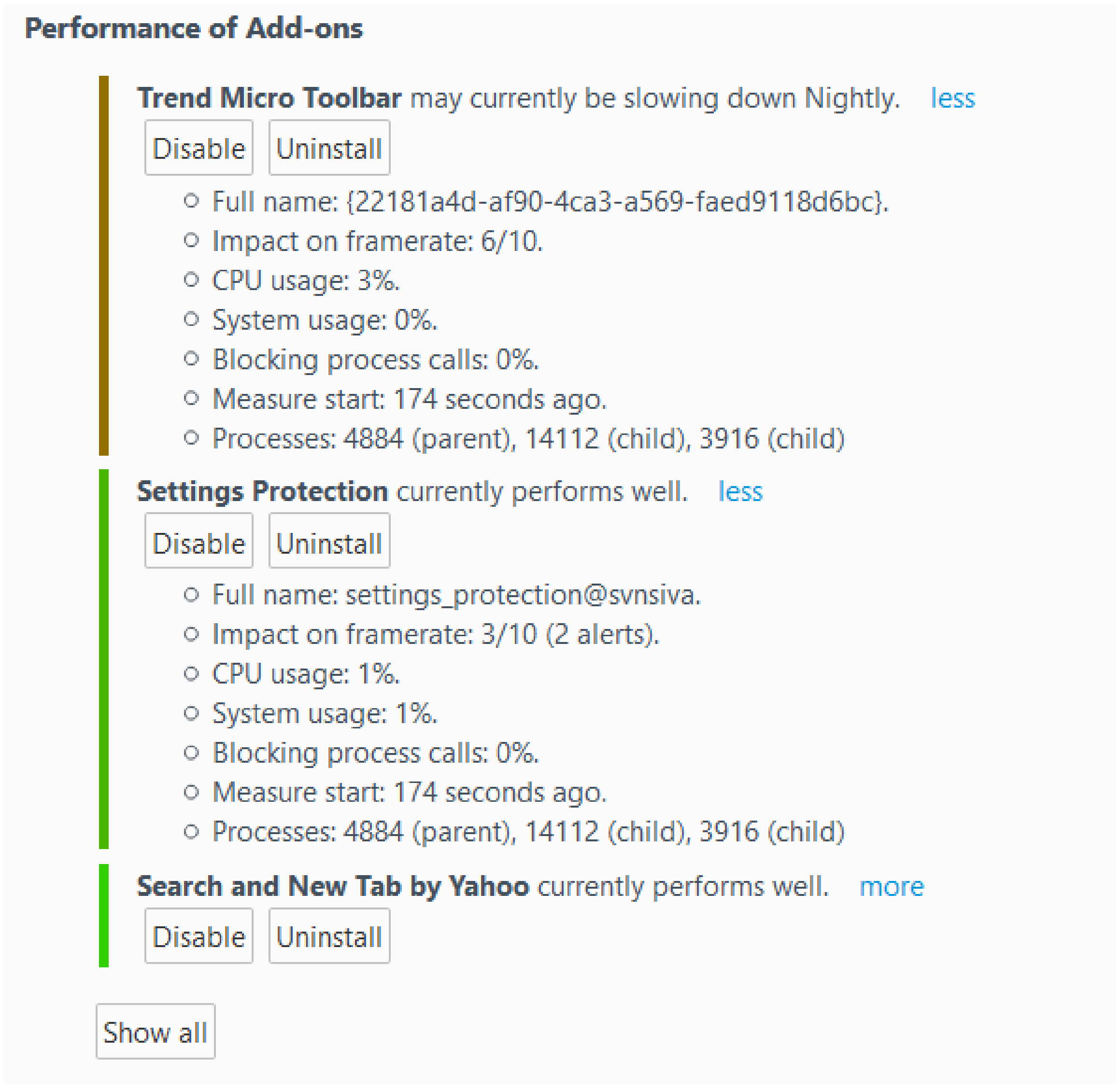
Search and New Tab by Yahoo currently performs well. more

Disable Uninstall

Show all

Figure 16. Performance of Idle Add-on

Figure 17 is the screenshot which shows the performance of the add-on in a case of a changed preference. In this figure, the results show that both the CPU and system usage are 1% although it performs well currently. The add-on works when there is a change, at that time also the impact on the performance of the browser is minimal.


Performance of Add-ons

Trend Micro Toolbar may currently be slowing down Nightly. less

- Full name: {22181a4d-af90-4ca3-a569-faed9118d6bc}.
- Impact on framerate: 6/10.
- CPU usage: 3%.
- System usage: 0%.
- Blocking process calls: 0%.
- Measure start: 174 seconds ago.
- Processes: 4884 (parent), 14112 (child), 3916 (child)

Settings Protection currently performs well. less

- Full name: settings_protection@svnsiva.
- Impact on framerate: 3/10 (2 alerts).
- CPU usage: 1%.
- System usage: 1%.
- Blocking process calls: 0%.
- Measure start: 174 seconds ago.
- Processes: 4884 (parent), 14112 (child), 3916 (child)

Search and New Tab by Yahoo currently performs well. more

Figure 17. Performance of the Add-on on Preference Change

With all of these evaluations and test results, the add-on works as expected. The overall performance is good, and impact on the performance of the browser is low. The following section discusses different exception cases handled in the add-on.

4.3. Evaluation and Results

This thesis tested the add-on in various cases, and it handled most of the exception cases. This section explains some of the exception cases handled. This thesis tested the add-on using many other add-ons, which modify one or more preferences and the add-on detected all of them. The add-on is tested with add-ons that modify preferences search engine, new tab URL, and homepage URL.

The add-on will work only after detecting changes, so it needs to detect all of the changes that occur to preferences. The preferences service API will listen for changes to preferences, irrespective of the reason for the modification, and the reason may be an add-on, malware, and adware. Thus, the add-on detects all of the changes to preferences irrespective of software type.

Some add-ons descriptions do not mention regarding modifications to preferences. After installing the add-on, a few days later the developer of the add-on may release a new update in which the new version modifies preferences. In the process of getting approval of the add-on by the official Mozilla add-on store, the developer will mention the modifications in the description and then after officially signed, the developer will change the description such that they no longer mention the modifications that the add-on performs to preferences. The add-on defended against these types of modifications also.

For testing purposes, this research developed a dummy add-on and got it signed by the official add-on store. The researchers of this thesis updated the add-on and released a new version of the dummy add-on, such that it modifies the preference homepage URL. The official add-on store signed the new version of the add-on. Later, the researchers changed the description such that it no longer mentions the homepage preference modification. The add-on stopped the modification of preference homepage upon updating the dummy add-on to new version.

Some users observe changes in preferences every time the browser restarts. These changes may be due to an add-on or malware. In some cases, the add-ons installed may try to modify preferences every time the browser starts up. Every time the user will modify the preferences to their desired values, and upon restarting the browser, the user will notice modifications to the preferences. In this case, if the modified values are different from old values stored, then the add-on developed in this thesis will discard the new changes for a small period of time at the browser start-up. The time is three seconds, and it is not possible for users to change preferences or install a new add-on in such a small period of time. Simply saying, the proposed solution does not allow non-user changes in the first three seconds of browser start-up.

The add-on handled situations like unauthorized disabling and uninstalling of the add-on. Users should go through authentication for disabling and uninstalling the add-on. In the case of failed authentication, the add-on stops the process of disabling or uninstalling. This feature will stop unauthorized access to disable and uninstall. In Mozilla Firefox, an add-on can uninstall other add-ons by searching in the list of add-ons available. The add-on avoids these types of actions by authenticating the disable/uninstall process.

This thesis handled runtime exceptions related to authentication. If the user selects master password as authentication and after the initialization process if the user stops using the master password, then the add-on will pop-up again asking the user to choose the type of authentication. If the user closes the browser without completing the add-on installation, then the add-on will initiate the installation process again at the next time of browser restarting.

This thesis handles the cases of unexpected browser shutdown while there are active pop-ups of the add-on. The add-on takes further steps based on the user response through the pop-ups. If the user closes the browser without responding to the active authentication pop-up, then the add-on discards the new changes. At the next time of the browser restarting, the add-on configures the browser with old preference values only. However, in the case of advanced monitoring if the user closes the browser while authentication pop-up is active, then the add-on authenticates these modifications at the next time of browser restarting. The reason is, in this case, the add-on stores changes in the permanent storage as these modifications might include the user changes.

In the browser, unwanted modifications can occur through any software. The add-on classifies these changes as a non-user change based on the active page of the browser. The add-on will respond to all non-user changes regardless of the kind of software that caused the changes. The browser modifies the preferences to previous values while uninstalling or removing the malware, and in this case, the add-on will not respond. If the user is the one who configured the previous values, there is no need to authenticate the changes due to uninstalling or removing malware.

For instance, consider a case in which the user sets Google as the current search engine in the browser. Now, software X is installed and modifies the search engine to a different search engine. The browser will revert the value to Google after uninstalling or removing of the software X. In this case, the user set the value to Google, and there is no need to authentication for this change.

4.4. Approving the Add-on

The developers of this thesis sent the add-on for official Mozilla approval. The AMO (addons.mozilla.org) team suggested some changes to make the add-on suitable to

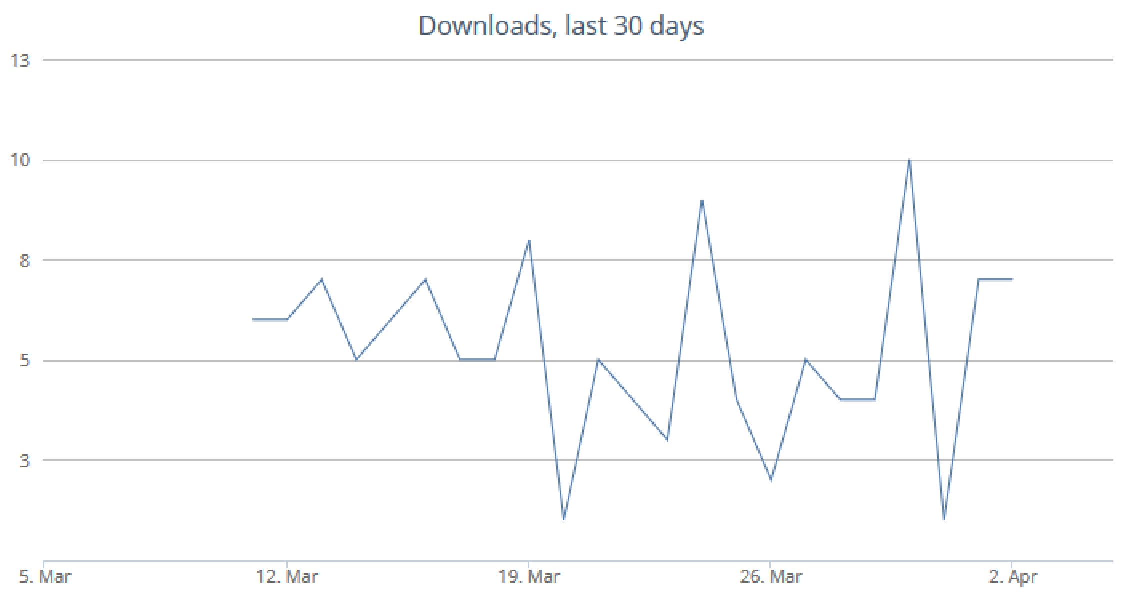


Figure 18. Downloads of the Add-on in March, 2017

Adopted from AMO statistics. Image is from:
https://addons.mozilla.org/en-US/firefox/addon/settings-protection/statistics/downloads/?last=30

their policy. The developers updated the add-on with suggested changes and sent the add-on again for the approval. The AMO team approved the add-on and released it for public use. Users of the Firefox browser can download the add-on from the web page with the URL,

`“https://addons.mozilla.org/en-US/firefox/addon/settings-protection/?src=userprofile,”`

and the name of the add-on is “Settings protection.”

Figure 18 presents the number of downloads in last 30 days. This thesis obtains this image from the AMO statistics of the add-on “Settings Protection.” AMO statistics reveal that 131 users download the add-on. AMO approved the add-on recently, and feedback about the add-on is very minimal. The developers will update and release new versions of the add-on based on user’s feedback in the future.

5. Future Recommendations and Summary

5.1. Limitations and Future Work

Every solution has some drawbacks and problems it cannot overcome. Similarly, there are a few limitations in the proposed solution found at the time of analysis. One limitation is when the user sets a preference to their desired value after an unwanted modification, and then the malware again modifies the preference. These changes are a repetitive process, and the user will be unable to set the desired value. With the solution proposed, users need to interact with the pop-up each time modifications occur and this leads to multiple pop-ups. Users may be unable to set preferences to their desired values with these unwanted modifications. The user should reset the total preferences or go through the difficult process of removing the malware.

The possible solution with an add-on is, consider a case in which a preference is modified to the same value more than once in a small period of time through non-user changes. Assume that the user discards these modifications two times. If the malware modifies the preference to the same value again, then the add-on will discard the modification automatically without informing the user.

This solution is not an ideal one to this problem because it has some drawbacks. At one time, the user is not interested in the modification, and later the user may require the modifications. The add-on can resolve this problem if it can have the information about the exact reason for the change and show it to the user. Based on this information, the user has a choice between saving new changes and dealing with the source of modification.

There are other problems such as redirecting of web pages and adware pop-ups. This thesis did not deal with these problems, as the proposed solution concentrated on the hijacking of browser preferences only. Protection against these issues can be part of the future work to the proposed solution.

Another drawback is exploitation of the proposed solution implemented through the add-on. The add-on depends on the preferences/options page for classifying a modification as a user or non-user change. Browser hijackers could try to exploit this feature and may modify the preferences. The proposed solution can overcome this drawback if the add-on has information about the exact reason for the change. By knowing the exact source of a modification, the add-on can easily classify between a user change and a non-user change and then authenticate if it is a non-user change.

If the browser hijacker knows the design methodology of the add-on, there is a chance of manipulation and exploitation of the add-on. The major limitation in developing the proposed solution as an add-on is that it is unable to know the exact reason for the modification. An add-on does not have any information about the source of modification in Mozilla Firefox. If this information were available, then it is would be difficult for the hijackers to exploit or manipulate the proposed solution.

This thesis recommends the browsers Google Firefox and Mozilla Firefox to make this solution available as a built-in feature. Browsers have exact information about the source of a modification so that they can take proper action. This information, along with the proposed solution, will be an ideal one for the browser hijacking problem. Browsers can make this feature available and leave the choice to users whether to enable or disable this feature.

5.2. Summary

The main goal of this thesis is to describe a technology to prevent unwanted changes to browser preferences and provide security to the user internet surfing. The add-on provides a user interactive solution to this problem. The add-on can help non-experts, as it does not require any technical knowledge to use. This thesis focused on protecting the mainly targeted preferences in browser hijacking such as search engine, homepage URL, new tab URL, and security preferences. The add-on successfully detected those undesired modifications of the preferences in a browser that the requirements considered during the design phase.

The "Settings Protection" add-on is readily available for use in the Mozilla Firefox browser's official add-on store and possess key features such as 1) an option to view more details about new changes, 2) ability to classify user changes and non-user changes accurately 3) a selection to accept the desired changes from a list of all of the changed preferences. 4) a description in the pop-up provided specifically when another add-on caused the modifications to the preference. Also, in the add-on, the user has the ability to exempt some preferences from monitoring, authenticate of opening preferences page, and always discard non-user changes.

The add-on development made use of APIs provided by Mozilla Firefox for add-on developers and coded in Javascript, HTML and CSS programming languages. Chapter 4 discussed the performance of the add-on using Mozilla performance tools and the outcomes proved to be satisfactory. The results in Chapter 4 show that the “Settings Protection” add-on had no memory leaks, with constat memory storage space occupied. The total number of downloads of this add-on in the month of March 2017 is 121 which demonstrates the significance of the “Settings Protection” add-on.

Bibliography

Abrams, L. (2006). Understanding Spyware, Browser Hijackers, and Dialers. Retrieved from https://www.bleepingcomputer.com/tutorials/understanding-spywarebrowser-hijackers-and-dialers/.

Baker, B. H. (2013). *High Fidelity Website Research: Using a Browser Extension to Provide a Natural Environment.* (Doctoral dissertation, Massachusetts Institute of Technology).

Banescu, S., Pretschner, A., Battré, D., Cazzulani, S., Shield, R., & Thompson, G. (2015, March). "Software-based protection against changeware." In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy* (pp. 231-242). ACM.

Battre & Pamg. (2013). Google Chromium preferences design documents. Retrieved from http://www.chromium.org/developers/design-documents/preferences.

Browser hijacking. (2017, February 18). In Wikipedia, The Free Encyclopedia. Retrieved 22:42, February 26, 2017, from https://en.wikipedia.org/w/index.php?title=Browser_hijacking&oldid=766180565.

Drake, J., Mehta, P., Miller, C., Moyer, S., Smith, R., & Valasek, C. (2011). Browser Security Comparison – A Quantitative Approach. Accuvant Labs. Retrieved from http://files.accuvant.com/web/files/AccuvantBrowserSecCompar_FINAL.pdf.

Felt, A. P. (2010). *Least Privilege for Browser Extensions* (Doctoral dissertation. University of California, Berkeley).

Google Chrome add-on development basics and APIs. (n.d.). Retrieved from https://developer.chrome.com/add-ons

Golubovic, N. (2016). *Attacking Browser Add-ons*. Master Thesis. Ruhr University Bochum. Retrieved from https://golubovic.net/thesis/master.pdf

Gribble, S., Levy, H., Moshchuk, A., & Bragin, T. (2012). Detection of spyware threats within virtual machine. U.S. Patent No. 8,196,205. Washington, DC: U.S. Patent and Trademark Office.

Gupta, S., Goyal, N., & Aggarwal, K. (2014). A Review of Comparative Study of MD5 and SSH Security Algorithm. *International Journal of Computer Applications*, 104(14).

Knuth, K. (2012). *Security Analysis of Browser Extension Concepts* (Doctoral dissertation, Saarland University). Retrieved from https://www.infsec.cs.uni-saarland.de/ uploads/ 2016/04/SecurityAnalysisOfBrowserAdd-onConcepts.pdf.

Karim, R. (2015). *Techniques and Tools for Secure Web Browser Extension Development* (Doctoral dissertation, Rutgers, The State University of New Jersey). Retrieved from http://dx.doi.org/doi:10.7282/T3000433.

Liverani, R. S. (2009). Exploting Firefox add-on. OWASP New Zealand day, Auckland. Retrieved from http://docslide.us/documents/exploiting-firefox-add-ons-owasp-new-zealand-day-2009-auckland-roberto.html.

Looksmart, Technorti, Genei Knows Media, Anchor Intelligence. “The uninvited guest - A browser hijacking experience dissected.” Retrieved from http://www.looksmart.com/assets/Uploads/browserhijackingreport.pdf.

Lonescu, P & Zuccato, J. (2015). “The 10 Most Common Application Attacks in Action. Security Intelligence.” Retreived from https://securityintelligence.com/the-10-most-common-application-attacks-in-action/.

Maier, N. (2016). about:addons-memory Mozilla Firefox add-on. Retrieved from https://addons.mozilla.org/en-US/firefox/addon/about-addons-memory/.

Mónica, D., & Ribeiro, C. (2015) An IDS for Browser Hijacking. *The Ninth International Conference on Emerging Security Information*. Systems and Technologies.

Moshchuk, A., Bragin, T., Gribble, S. D., & Levy, H. M. (2006, February). A Crawler-based Study of Spyware in the Web. In NDSS (Vol. 1, p. 2).

Mozilla Firefox SDK add-on add-on development details and APIs. Retreived from https://developer.mozilla.org/en-US/Add-ons/SDK.

Payton, A. M. (2006, September). A review of spyware campaigns and strategies to combat them. *In Proceedings of the 3rd Annual Conference on Information Security Curriculum Development* (pp. 136-141). ACM.

Provos, N., McNamee, D., Mavrommatis, P., Wang, K., & Modadugu, N. (2007). The Ghost in the Browser: Analysis of Web-based Malware. HotBots, 7, 4-4.

Schlarb, A. Manage search engines add-on in Mozilla Firefox. Retrieved from https://addons.mozilla.org/en-US/firefox/addon/manage-search-engines-button/?src=api.

Serrhini, M., & Moussa, A. A. (2013). “Home Users Security and The Web Browser Inbuilt Settings, Framework to Setup It Automatically.” *Journal of Computer Science*, 9(2), 159.

Ter Louw, M., Lim, J. S., & Venkatakrishnan, V. N. (2008). Enhancing web browser security against malware extensions. *Journal in Computer Virology*, 4(3), 179-195.

Topolski, R. M. (2008). NebuAd and Partner ISPs: Wiretapping, Forgery and Browser Hijacking. Retrieved from https://www.savetheinternet.com/sites/default/files/resources/NebuAd_Report.pdf.

Wbamberg. (2016, December). Mozilla Firefox Add-on Development. Retrieved from https://developer.mozilla.org/en-US/Add-ons/SDK/Tools/jpm.

Wespel, T., & Salomon, T. (2015). Cloud based reputation system for browser settings. U.S. Patent Application No. 14/879,892.

Wüest, C. & Florio, E. (2009). Firefox and Malware:When Browsers Attack. Symantec Corporation. Retrieved from https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/firefox_and_malware.pdf.