

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

On the Lookout for Malware in Browser Extensions

Bernardo Manuel Costa Barbosa



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Bispo

Second Supervisor: Luís Reis

November 4, 2021

On the Lookout for Malware in Browser Extensions

Bernardo Manuel Costa Barbosa

Mestrado Integrado em Engenharia Informática e Computação

November 4, 2021

Abstract

Web browsers have become a fundamental platform for everyday computation, allowing not only access to websites but also to interact with multiple web applications and their rich set of functionalities.

JavaScript-based browser extensions enhance the core functionality of web browsers and allow users to customize their look and feel, improve convenience, efficiency, privacy and security. However, these extensions usually need additional access permissions to operate, including browsing history, website content, login credentials, among others.

Malicious browser extensions pose an emergent security risk and have become one of Web Browsers' most common attack vectors due to their wide popularity and high privileges. Even if the extension does not have any malicious intent, it may contain vulnerabilities that can allow a remote attacker to compromise the browser's security.

A compromised extension can allow the stealing of cookies and saved passwords or the injection of malware through commercial ads - *Malvertising* - even in well-reputed websites.

Multiple state-of-the-art approaches exist to prevent and classify the malicious behaviors and vulnerabilities of browser extensions, from static code analysis to dynamic behavior analysis, such as monitoring the execution of the extension and the corresponding network activity.

In this dissertation, we present SafeWare, a hybrid approach to quantify the risk of a browser extension by applying static and dynamic techniques to analyze an extension for extracting features from the source codes (JavaScript, HTML, CSS), online review platforms and the activities of an extension with the browser's extension API.

Combined with our hybrid approach, an experimental risk scoring system was also implemented to obtain a risk score that aims to present the amount of risk existing in an extension for the end-user.

Keywords: Software Engineering, Computational Systems, Browser Extension, Web Security, Information Flow Control

Resumo

Os navegadores de Internet tornaram-se uma plataforma fundamental para a computação do dia-a-dia, permitindo não apenas o acesso a websites, como também a interacção com múltiplas aplicações Web e o seu vasto conjunto de funcionalidades.

As extensões de navegador baseadas em JavaScript melhoram o funcionamento dos navegadores Web e permitem que os utilizadores personalizem a sua aparência, aumentando também a sua conveniência, eficiência, privacidade e segurança. No entanto, estas extensões geralmente necessitam de permissões de acesso adicionais para funcionarem, incluindo o histórico de navegação, o conteúdo do website, as credenciais de login, entre outras.

Extensões de navegador maliciosas representam um risco de segurança emergente e tornaram-se um dos vetores de ataque mais comuns nos navegadores de Internet devido à sua grande popularidade e altos privilégios. Ainda que uma extensão não tenha qualquer intenção maliciosa, pode conter vulnerabilidades que permitam a um atacante comprometer remotamente a segurança do navegador. x

Uma extensão comprometida pode permitir o roubo de cookies e passwords guardadas, ou injetar malware através de anúncios - *Malvertising* - mesmo em websites bem reputados.

Para prevenir e classificar os comportamentos maliciosos e vulnerabilidades de extensões de navegador, existem já várias abordagens, desde a análise estática do código-fonte até à análise dinâmica do comportamento, tal como a monitorização da execução da extensão e a atividade de rede correspondente.

Nesta dissertação, apresentamos o SafeWare, uma abordagem híbrida para quantificar o risco de uma extensão de um navegador aplicando técnicas estáticas e dinâmicas para analisar uma extensão ao extrair características do código-fonte (JavaScript, HTML, CSS), de plataformas online e das actividades entre a extensão e a API de extensões do navegador.

Juntamente com a nossa abordagem híbrida, foi também implementado um sistema de pontuação de risco experimental para obter um valor de risco que visa apresentar a quantidade de risco existente numa extensão para o utilizador final.

Keywords: Engenharia de Software, Sistemas Computacionais, Extensão de Browser, Segurança na Web, Controlo de Fluxo de Informação

Acknowledgements

First, I would like to thank my parents, António and Eugénia, my brothers, Gonçalo and Rodrigo, as well as my grandparents and family for their outstanding and unconditional support, for always believing in me. They are my biggest supporters, and without them, I would not be where I am today. A special thanks to my girlfriend, Alexandra, for keeping me always motivated to be the best version of myself, for being consistently by my side and for listening to me when I struggled to keep pushing this work. To all my close friends, thank you for all the support throughout these years, I feel blessed for having you and I cannot wait to show you how bright the future is ahead of us.

With great regard, a very special thank you to my supervisors Dr. João Bispo and Dr. Luís Reis. Thank you for making this dissertation possible, your knowledge, patience and perseverance were vital in this project. Also, thank you for never letting me give up and always guiding me through the last year.

Finally, I would like to thank Jscrambler for this fantastic opportunity and for treating me so nicely from the very beginning until my last day, and with so much pride, a very heartwarming thank you to the FEUP community for giving me all the possible tools to grow up not only professionally but personally as well.

Bernardo Manuel Costa Barbosa

*“I didn’t come this far to only come this far,
I came this far so I could be strong enough to go further!”*

William Hollis

Contents

1	Introduction	1
1.1	Context and Motivation	2
1.2	Objectives	2
1.3	Contributions	3
1.4	Document Structure	3
2	Background and Literature Review	5
2.1	Concepts	5
2.1.1	Browser Extension	5
2.1.2	Google Chrome Extension	6
2.2	Vulnerabilities	12
2.2.1	Password and Cookie Stealing	13
2.2.2	Malvertising	14
2.2.3	Browsing Tracking	15
2.2.4	Supply Chain Attack	16
2.3	Related Work	17
2.3.1	Sabre	17
2.3.2	Hulk	18
2.3.3	Expector	19
2.3.4	BrowsingFog	20
2.3.5	XHOUND	22
2.3.6	CRXCavator	24
2.3.7	Yao Wang et al.'s Combined Static and Dynamic Analysis Approach . . .	26
2.4	Comparison with Related Work	28
2.5	Summary	29
3	Hybrid Approach to Quantify Extension Risk	31
3.1	Methodology	32
3.2	Static Approach	32
3.3	Dynamic Approach	33
3.4	Risk Scoring System	37
3.4.1	Static Analysis	38
3.4.2	Dynamic Analysis	40
3.5	Implementation	42
3.6	Summary	43

4	Experimental Results	45
4.1	Dataset	45
4.2	Experimental Setup	48
4.3	Testing	49
4.3.1	Benign Extensions	50
4.3.2	Malicious Extensions	54
4.3.3	Benign and Malicious Extensions	57
4.4	Risk Scoring Evaluation	58
4.4.1	Fake vs Original Extensions	64
4.4.2	Conclusions	66
4.5	Execution Time Evaluation	68
4.6	Summary	69
5	Conclusions and Future Work	71
5.1	Conclusions	71
5.2	Future Work	72
A	Dataset of Extensions	75
A.1	Benign Extensions	75
A.2	Malicious Extensions	89
B	Code Snippets	103
B.1	Original ID and Public Key Extraction	103
B.2	Converting CRX2/ZIP to CRX3	104
C	Dataset of Websites	105
	References	107

List of Figures

2.1	CRX Package Format [3]	7
2.2	CRX3 Package Format [4]	8
2.3	Session Hijacking using XSS [47]	14
2.4	"Drive-By Download" Attack [22]	15
2.5	Example of Web-based Supply Chain Attack [48]	17
2.6	BrowsingFog Demo [17]	22
2.7	XHOUND Data Sources and Architecture [38]	24
2.8	Workflow of Machine-Learning-based detection [44]	27
3.1	Chrome Apps & Extensions Developer Tool GUI	34
3.2	Chrome Apps & Extensions Developer Tool – Extension Activities	35
3.3	Modification to download Behaviour History	43

List of Tables

3.1	Critical Permissions	39
3.2	High-Risk Permissions	39
3.3	Low-Risk Activities – 5 points	40
3.4	Mid-Risk Activities – 10 points	40
3.5	High-Risk Activities – 15 points	41
3.6	Activities – Web Requests	41
4.1	Top 25 Most Popular Extensions on Chrome Web Store	46
4.2	Dataset of Malicious Extensions	47
4.3	Original and Fake Extensions	47
4.4	Benign Extensions not installed during Benign Extensions Automation	50
4.5	250 Benign Extensions – Static Analysis Results	52
4.6	250 Benign Extensions – Dynamic Analysis Results	53
4.7	250 Benign Extensions – <i>webRequest</i> API Results	54
4.8	250 Malicious Extensions – Static Analysis Results	55
4.9	250 Malicious Extensions – Dynamic Analysis Results	56
4.10	250 Malicious Extensions – <i>webRequest</i> API Results	57
4.11	Extensions – New Activities Captured	58
4.12	250 Benign Extensions – Top 10 Highest Scores	59
4.13	250 Benign Extensions – Permissions	60
4.14	250 Benign Extensions – Top 10 Lowest Scores	61
4.15	250 Malicious Extensions – Top 10 Highest Scores	62
4.16	250 Malicious Extensions – Permissions	63
4.17	250 Malicious Extensions – Top 10 Lowest Scores	64
4.18	Hybrid Approach – Sum of Risk Scores	66
4.19	Static Approach – Risk Scores	67
4.20	Dynamic Approach – Risk Scores	67
4.21	250 Benign Automation - Iteration Times	68
4.22	250 Malicious Automation - Iteration Times	68
4.23	500 Benign + Malicious Automation - Iteration Times	69
4.24	No-Extensions Automation - 5 Iterations Times	69
A.1	Dataset of 250 Benign Extensions	88
A.2	Dataset of 250 Malicious Extensions	102
C.1	Top 100 Most Visited Websites by Search Traffic (2021) [27]	106

Abbreviations

API	Application Programming Interface
CRX	Chrome Extension
CSP	Content Security Policy
CSS	Cascading Style Sheets
CWS	Chrome Web Store
DNS	Domain Name System
DOM	Document Object Model
FEUP	Faculty of Engineering of the University of Porto
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
JS	JavaScript
JSE	JavaScript-based Browser Extension
JSON	JavaScript Object Notation
PBM	Private Browsing Mode
RSA	Rivest-Shamir-Adleman
SHA	Secure Hash Algorithm
URL	Uniform Resource Locator
VM	Virtual Machine
XSS	Cross-Site Scripting

Chapter 1

Introduction

Nowadays, modern web browsers are designed to allow *JavaScript-based Browser Extensions (JSEs)* [18], programmed and generally used to enhance the core functionality of the browser and to give a better user experience than the one natively provided by web applications and the browser itself.

This dissertation explores a possible solution to classify and prevent a malicious extension from exploiting a user, using the previously granted permissions, and causing damage to a web application by controlling the interaction between the user and the servers.

JavaScript code inside an extension can freely access private and sensitive data, such as cookies and browsing history [18] [5], as well as sending requests to any web server/domain or even reading our file system. Although browsers define a set of permissions in their extension API, the permissions informative prompts when installing an extension are not fully explained, allowing programmers to exploit the limitations of specific permissions, as most users tend to install extensions without being conscious of the full extent of the consequences of installing their extensions [36].

Even if a *JSE* has no malicious intentions, a malicious website can exploit an extension through its vulnerabilities, e.g. if an extension uses *eval* or *document.write* without sanitizing the input, the attacker might be able to inject a script into the extension [6].

Although it is impossible to be completely safe, there are already some tools to prevent and detect suspicious activity on a browser extension, some of which will be briefly described in this paper. State-of-the-art approaches vary, ranging from static code analysis to dynamic behaviors, such as execution monitoring and related network activity.

For this dissertation's problem, we worked on a hybrid approach as a possible solution, since we consider important what the application does when it is running and what we can get out of it as a static application.

Concerning the goal of this paper, it is essential to understand the structure of a browser extension and how it can interact with the browser's API given the permissions (and consequently

privileges) that are specified in their manifest, as well as their architecture and life cycle, so we can better set up our hybrid analysis.

We used a Virtual Machine (VM) to work on a safe environment and to test our tool, SafeWare, against possible/known malicious extensions, not only to protect our network but to save time when rebooting or reinstalling any software needed to test our solution.

Related work used extensions from the Chrome Web Store¹ and the Firefox Store² to test the efficacy of their tools. To simplify the work, we decided to focus on a single web browser, and for that, we chose the browser that currently has the largest market share worldwide³, Google Chrome⁴, with about 65%, followed by Safari⁵ with approximately 19%, while both Microsoft Edge⁶ and Mozilla Firefox⁷ are at 3.5%.

1.1 Context and Motivation

Web applications are getting more sophisticated as frameworks evolve and companies look for the best user experience possible for everyone that regularly uses their product.

Browser Extensions are commonly used to improve the core functionality of web browsers by offering a high level of content customization and changing the way we interact with websites and their features. Having this in mind, there is an opportunity for security companies to develop an efficient tool to protect their customers' services from being attacked, exploited and manipulated.

This work aims to avoid these attacks, as Internet users should browse the web safely while being protected from malicious attempts or vulnerabilities that may be an entry point to steal their personal information. Moreover, websites should also preserve their security system, image and, more importantly, their customers.

Considering this, we decided to work together with Jscrambler and their knowledge in JavaScript and Web Security to develop a better solution using multiple approaches to identify malicious extensions and improve the security level of the extensions that currently exist.

1.2 Objectives

The goal of this dissertation is to use static and dynamic code analysis to implement a hybrid tool capable of quantifying the risk associated with a browser extension.

When we mention the risk of an extension, we are mainly targeting its trustworthiness and how much we can trust in a certain browser extension. In this case, our risk score represents how much we can not trust a browser extension, since the higher the value, the higher the risk.

¹Chrome Web Store, <https://chrome.google.com/webstore/>

²Add-ons for Firefox, <https://addons.mozilla.org>

³Browser Market Share Worldwide – August 2021, <https://gs.statcounter.com/browser-market-share>

⁴Google Chrome, <https://www.google.com/intl/pt-PT/chrome/>

⁵Safari - Apple, <https://www.apple.com/safari/>

⁶Microsoft Edge, <https://www.microsoft.com/pt-pt/edge>

⁷Mozilla Firefox, <https://www.mozilla.org/pt-PT/firefox/>

The final value of each extension's report should essentially represent a level of trust in which the user can doubt not only the requested permissions but also the interests of the extension. Despite this trustworthiness value, we cannot prove or classify an extension as malicious at any time, as we can only understand that a specific extension has a set of risky properties and activities.

From analyzing static source code to monitoring page modifications and network activity, we will study some valid approaches and combine what we think can add efficacy and accuracy to our tool to correctly quantify whose extensions are more likely to present more risk to the end-user.

1.3 Contributions

Above everything, this work intends to solve a real problem regarding web browsing and extensions inside web browsers. These extensions can enhance the user experience but, at the same time, are executed on an architecture and an environment where they can be malicious and cause damage if not properly designed and programmed.

The risk quantification presented by our solution is targeting to raise awareness of the risk that extensions, malicious or not, pose to end-users. We believe we can make a difference by not only making extensions more secure but also preventing extensions that pose a high risk from continuing to use high-risk permissions and performing suspicious activities that do not comply with their privacy policies or the goals stated by them.

1.4 Document Structure

- **Chapter 2: Background and Literature Review**

This chapter presents background references around the topic of this dissertation and reviews the current state-of-the-art.

- **Chapter 3: Hybrid Approach to Quantify Extension Risk**

In this chapter, the main problem and a proposed hybrid solution approach are presented.

- **Chapter 4: Experimental Results**

This chapter describes our experimental setup, results and their analysis.

- **Chapter 5: Conclusions and Future Work**

The final chapter states conclusions on this dissertation and future work on this topic.

Chapter 2

Background and Literature Review

Web browsers are more than just a bridge to the World Wide Web, services like browsing history, bookmarks, tab management and password management can enhance the user experience and change the way we interact with web applications.

Browser extensions are small software modules that run inside web browsers and allow users to customize them according to their demands [43] and taste, giving users a personalized and more sophisticated experience beyond what web applications already allow them to do.

From ad blocking (AdBlock)¹, grammar writing assistance (Grammarly)² or password vault (LastPass)³, thousands of extensions [34] are available to increase our productivity to add more functionalities to the already powerful web browsers.

In this document, we will mainly focus on Google Chrome properties and functionalities. Still, our conclusions can be generalized to other web browsers, such as Mozilla Firefox, Microsoft Edge and Safari.

2.1 Concepts

Before we approach the related work, it is important to understand what a browser extension is, how they are built, how they work and which vulnerabilities they can have, and examples of attacks that can represent a threat for both user machines and their personal data.

2.1.1 Browser Extension

High-end web browsers (e.g., Google Chrome, Mozilla Firefox) allow people to interact with information on the Internet by enhancing their functionalities and personalizing the user's browsing

¹AdBlock, <https://getadblock.com/en/>

²Grammarly, <https://www.grammarly.com>

³LastPass, <https://www.lastpass.com/>

experience. Built using JavaScript, CSS and HTML, browser extensions are meant to extend or modify the default behavior of a browser and are targeted towards end-users.

Using extensions, users can modify the interface of web browsers, add new features that are not supported by default and modify web pages according to their preferences.

Taking advantage of the capability of browser extensions and browsers extension APIs, users use extensions to block unwanted ads, download videos, capture screenshots, manage/save their passwords and much more.

However, the implementation of these extensions in the browsers is not perfect and their architecture allows them to be exploited in an unethical way when developers present malicious intent.

2.1.2 Google Chrome Extension

Since we will handle and work with Google Chrome extensions, it is fundamental to understand its architecture and security model, since we will have to know what to extract and how to obtain that information from each extension.

In Google Chrome, extensions run in a separate, sandboxed execution environment and interact with the Chrome browser through the Chrome Extension API – interface to access multiple features of the browser itself: activating tabs, modifying net requests, reading cookies, etc [14].

An extension is created by assembling some of its resources – a manifest, source code files and images.

The manifest is a JavaScript Object Notation (JSON) formatted file that provides important information about the extension, such as the name of the extension and the code files that the extension might use, as well as some APIs the extension needs to access, which are listed in the permissions field. (Listing 2.1) These properties are enforced by Google Chrome, as the extension will only access a particular API if the permission for that API is stated in the manifest file.

2.1.2.1 CRX Package File

Essentially, an extension is a compressed Chrome Extension (CRX) package that includes the extension content and a header to ensure its security across the Web. This package consists of a *ZIP* file – an archive file format that supports lossless data compression – signed by the author to maintain the original integrity of the file when it is installed or downloaded, protecting the extension against tampering, e.g., by adding a Bitcoin miner to an extension script before installing it.

In order to sign the extension, a pair of public and private keys are generated: the extension is signed with the author's private key and users check the downloaded *ZIP* with the author's public key. Since there is no standard way to sign a *ZIP* file and its metadata does not support an embedded signature, a **.crx** file was designed to contain a copy of an RSA public key and an encrypted SHA1 of the **.zip** archive [3]. (Figure 2.1)

```

1 {
2   "manifest_version": 2,
3   "permissions": [ "activeTab", "contextMenus", "storage"],
4   "content_security_policy": "script-src 'self' 'unsafe-eval' https://translate.
   googleapis.com; object-src 'self'",
5   "content_scripts": [
6     { "matches": ["<all_urls>"],
7       "js": ["bubble_compiled.js"],
8       "css": ["bubble_gss.css"],
9       "all_frames": true } ],
10  "background": {
11    "scripts": ["injection.js", "main_compiled.js"],
12    "persistent": false },
13  "version": "2.0.9",
14  "browser_action": {
15    "default_icon": { "19": "icons/19.png", "38": "icons/38.png"},
16    "default_popup": "popup.html" },
17  "web_accessible_resources": ["popup_css_compiled.css", "options.html"],
18  "options_page": "options.html",
19  "icons": { "16": "icons/16.png", "32": "icons/32.png", "48": "icons/48.png",
20    "128": "icons/128.png" },
21  "update_url": "https://clients2.google.com/service/update2/crx",
22  "author": { "email": "thom@google.com" }
23 }

```

Listing 2.1: Example of a Manifest file [21]

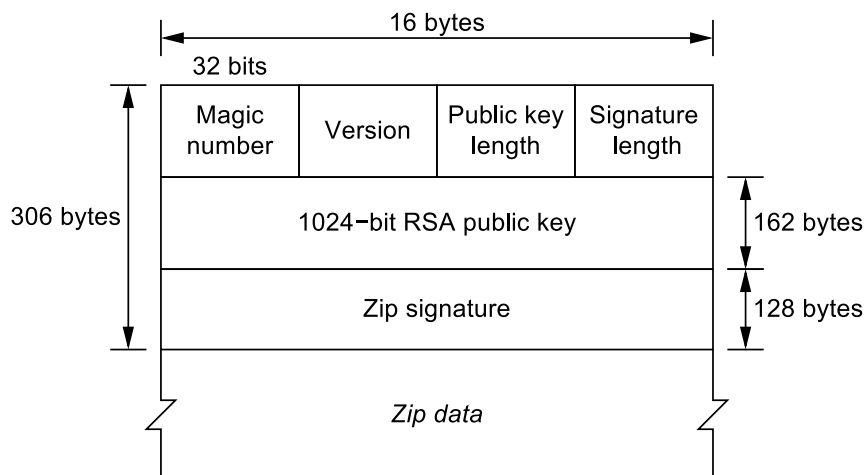
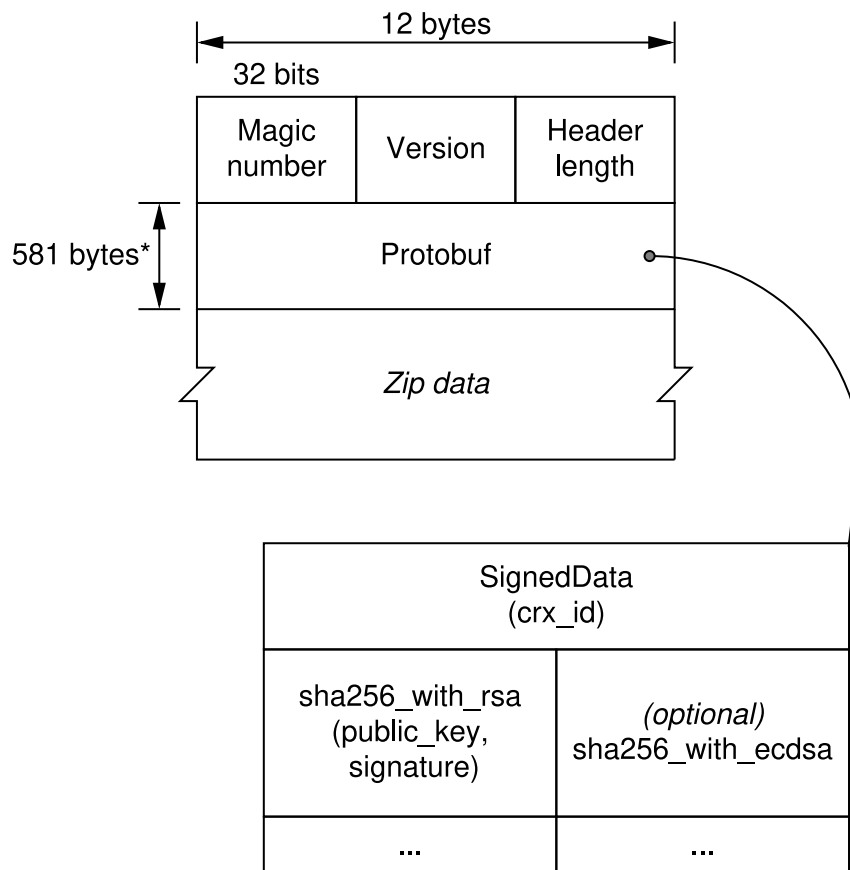


Figure 2.1: CRX Package Format [3]

Since breaking SHA1 is technically possible, which would allow attackers to intercept an extension update and inject arbitrary code into it, Google Chrome introduced CRX3 – a new file format that uses a more robust algorithm to avoid this risk – forcing all extensions to be packaged with CRX3 on Chrome version 76⁴.

⁴Archived release notes – Google Chrome Enterprise Help, <https://support.google.com/chrome/a/answer/10314655>

CRX3 prefixes a protobuf⁵ that can contain an unlimited number of public keys and signature tuples (proofs). (Figure 2.2) A created **.crx** file for a Linux version of Chrome requires only one proof, however, extensions from the Chrome Web Store incorporate multiple proofs [4].



* For a single sha256_with_rsa proof with a 2048-bit key

Figure 2.2: CRX3 Package Format [4]

This is important, as later we will need to know which public key was used to generate the extension ID – a unique identifier to ensure that a specific extension was signed by the original identity – in order to maintain the same extension ID across all tests, since Chrome checks for the key entry **key**⁶ on each manifest file before importing an unpacked extension.

2.1.2.2 Permissions

An extension only has access to APIs proclaimed by its developers in the permissions field of the manifest.

⁵CRX3 Chromium Protobuf, https://github.com/chromium/chromium/blob/e15cfb65a092b26224d6bdc685ed2aa9f41f7eb6/components/crx_file/crx3.proto

⁶Manifest – Key, <https://developer.chrome.com/docs/extensions/mv3/manifest/key/>

From a list of known strings or a match pattern that gives access to one or more hosts, the programmer must declare all the permissions required for the extension to work correctly, e.g., *tabs* permission is required to interact with the browser's tab system (create, modify, and rearrange tabs). (Listing 2.2) From that list, some permissions might also be displayed to users before the installation, as detailed in Permission Warnings [20].

```
1 "permissions": [  
2   "<all_urls>",  
3   "webRequest",  
4   "tabs",  
5   "bookmarks",  
6   "http://www.blogger.com/",  
7   "http://*.google.com/",  
8   "unlimitedStorage",  
9   "https://*/*" ]  
10 ]
```

Listing 2.2: Example – Permissions from a Manifest file [19]

Using the example above, the permission *https://*/** allows an extension to run on all HTTPS websites, while *webRequest* permission enables an extension to observe and analyze network traffic, as well as intercept, block or modify web requests. Finally, *unlimitedStorage* provides to the extension an unlimited quota for storing client-side data, such as databases and local storage files.

Although there is this layer of security between an extension and the user, at installation time, only a small alert is shown mentioning very briefly what the extension can or cannot do and the user might not be correctly aware of the risks entailed by each permission [36].

Later on, we discuss the potential threats caused by abusing some permissions that we consider to present an elevated risk to the end-user, addressing the danger they can pose when used with malicious intentions and how they can help limit the damage if an extension or app is compromised by malware.

2.1.2.3 Content Scripts

Content scripts are JavaScript files that run in the context of web pages, which means that they are meant to be executed every time the web page is refreshed.

By using the standard Document Object Model (DOM), content scripts can read details of the web page the browser visits or modify the webpage's DOM. However, content scripts have some limitations, since they cannot use Chrome APIs or access the variables and functions in the background pages directly [10]. To access Chrome APIs, these scripts exchange messages with the background scripts of the extensions.

Since content scripts live in an isolated world, they can make changes to their JavaScript environment without conflicting with the page or additional content scripts. These scripts can be declared statically or programmatically injected.

Statically declared scripts are mentioned in the manifest under the **content_scripts** field, including JavaScript and CSS files, that should be automatically run on a well-known set of pages. (Listing 2.3)

```

1 "content_scripts": [
2   {
3     "matches": ["http://*.nytimes.com/*"],
4     "exclude_matches": ["*://*/business*"],
5     "include_globs": ["*nytimes.com/???s/*"],
6     "exclude_globs": ["*science*"],
7     "run_at": "document_idle",
8     "css": ["myStyles.css"],
9     "js": ["contentScript.js"]
10  }
11 ]

```

Listing 2.3: Example – Content Scripts of a Manifest file [10]

In the example above, the content script will be injected into a web page if its URL matches any **matches** pattern, as long as the URL respects the other patterns. Since the **matches** property is required, **exclude_matches**, **include_globs** and **exclude_globs** properties can only be used to limit which pages will be affected. For example, the extension above would inject the content script into *https://history.nytimes.com* and *https://nytimes.com/history* but not into *https://science.nytimes.com* and *https://www.nytimes.com/science*.

As for injecting programmatically, an extension needs host permissions for the page that it is trying to inject scripts into. Host permissions can be granted either by requesting them from the extension's manifest or temporarily via the **activeTab** permission.

Listing 2.4 contains examples of possible content scripts injections by an extension: as code, an entire file or a function body can be injected and executed as a content script.

Not only that, but an extension can also dynamically register a new content script using *chrome.scripting.registerContentScript* method.

```

1 # Code
2 document.body.style.backgroundColor = 'orange';
3
4 # Entire File
5 chrome.scripting.executeScript({ target: { tabId: tab.id }, files: ['content-script
   .js'] });
6
7 # Function Body
8 function injectedFunction() { document.body.style.backgroundColor = 'orange'; }
9 chrome.scripting.executeScript({ target: { tabId: tab.id }, function:
   injectedFunction });

```

Listing 2.4: Injecting Script as Code and Entire File

Also, the developer can define when these scripts are injected into the web page through the **run_at** property:

- **document_start**

Injected after any CSS but before any other DOM is constructed or any other script is run.

- **document_end**

Injected immediately after the DOM is complete, but before sub-resources such as images and frames have loaded.

- **document_idle** (default)

Browser chooses a time to inject between **document_end** and immediately after the *window.onload* event fires – depends on how complex the document is and how long it is taking to load.

2.1.2.4 Service Workers

Essentially, extensions are event-based programs used to modify or enhance the Chrome browsing experience. Those events are browser triggers that can be listened to, such as navigating to a new page, removing a bookmark, or even closing a tab. In their background service worker, extensions monitor a defined set of events, reacting with specific instructions.

Known as background scripts before the introduction of *Manifest V3*⁷, a background service worker is loaded when it is needed and unloaded when it goes idle, e.g., installation or update; message received from any content-script or extension; listened event is dispatched. Just like content scripts, service workers are declared in the extension's manifest using a field **service_worker** under the **background** field, specifying only one JavaScript file [12].

Service workers can maintain the state and control the logical behavior of the extension. For example, an extension can create a new tab by using the *chrome.tabs.create()* method in the background script.

After being initialized, this script registers all the listeners synchronously, as well as their reaction structures to run when an event is fired. More importantly, these are the responsible scripts for interacting with Chrome APIs, being considered the brains of an extension.

2.1.2.5 Content Security Policy

Nowadays, there are multiple important concepts in the web application security model, one being the same-origin policy – a critical security mechanism that restricts how a document or script loaded by one origin can interact with a resource from another origin. For instance, it can prevent a malicious website on the Internet from running JavaScript in a browser to read data from a third-party webmail service (which the user is signed into) and relaying that data to the attacker. Two URLs have the same origin if the protocol, port (if specified), and host are the same for both [25].

⁷Welcome to Manifest V3, <https://developer.chrome.com/docs/extensions/mv3/intro/>

In theory, code from *https://mybank.com* should only have access to *https://mybank.com*'s data, and *https://example.com* should certainly never be allowed access, keeping each origin isolated from the rest of the web. In practice, attackers have found ways to subvert the system, such as Cross-Site Scripting (XSS) attacks, that bypass the same-origin policy by tricking a site into delivering malicious code along with the intended content, taking advantage from the fact that browsers trust all the code that shows up on a page as being legitimately part of that page's security origin.

Aiming to fight this and other possible attacks, Content Security Policy⁸ (CSP) is an HTTP response header that modern browsers use to enhance the security of the web page, allowing web developers to restrict how the browser loads resources (JavaScript, CSS, etc.). This is possible by creating an allow-list of trusted content sources where the browser will only execute or render resources from those sources. Even if an attacker can find a door through which to inject script, if the script does not match the allow-list, it will not be executed [33].

Unlike JavaScript code in a web application, JavaScript code in *JavaScript-based Browser Extensions (JSEs)* execute with the privileges of the browser, not being constrained by the same-origin policy and accessing sensitive data through Chrome APIs (cookies, history, etc.) [18].

Although not mandatory, Chrome recommends extension developers to make their extensions CSP-compliant by following a set of restrictions [11]:

- No inline scripting – banning `<script>` blocks and event handlers (`<button onclick="...">`);
- No reference to any external resources in any file (except for video and audio resources);
- No external resources inside an `<iframe>`;
- No string-to-JavaScript methods such as `eval()` and `new Function()`;
- All JavaScript files and resources should be local (package).

In the latest Manifest version, it is possible to declare which pages inside the extension are covered by this policy, including HTML files and service workers, as well as which pages will be sandboxed – allowing these pages to not subject to the CSP used by the rest of the extension (having its own separate CSP), being able to use, for example, inline scripting and `eval()` methods [13].

2.2 Vulnerabilities

As mentioned before, browser extensions run through the browser's JavaScript event listener system by subscribing to a number of events associated with the browser (e.g., a new tab is opened or a new bookmark is added) or with the content (e.g., a user clicks on an HTML element or when the page is loaded). When a JavaScript event is triggered, the event is captured by the browser engine and all extensions subscribed to this event are executed [35].

⁸Content Security Policy Reference, <https://content-security-policy.com>

Since browser extensions run with high privileges, they have access to sensitive data. That needs to be protected and well monitored, as developers can use them as open doors to various malicious intentions. Even with some security requirements, official stores like Google's Chrome Web Store cannot completely certify if an extension is malicious or has vulnerabilities, hosting extensions that contain malware [16].

However, just because an extension is not malicious does not necessarily mean that it is safe. Since the majority of extensions authors are not security-related specialists [44], extensions can be a gold target for hackers that are looking for breaches to exploit [34].

2.2.1 Password and Cookie Stealing

The impact of malicious browser extensions can be huge when we are talking about passwords that are the key to our bank accounts, exchange accounts (cryptocurrency), social media accounts, e-mail clients and streaming services [29] [44].

Cookies are built to give websites the ability to implement session management (recognize login information and preferences), personalization (customized advertising is used to personalize user's sessions) and track users (track previously viewed websites, suggest others that might fulfill the user's needs) [30].

The main threat relies on Permanent (also know as Persistent [30]) Cookies – cookies that remain in operation even after the web browser has closed and are destroyed once its expiration date is reached – since they are mostly used for Authentication and Tracking purposes, as proof of our identity (which, if compromised, can expose the user to session hijacking) and profiling what we visited in a website (profile building) [32].

2.2.1.1 Session Hijacking

The act of stealing a user's session cookie after generated is known as session hijacking. Among security attacks through browser extensions [26], session hijacking is an attack where an attacker can steal an authentication session ID using brute-force (trying multiple IDs until one is successful), capture (acquiring the session ID using different techniques) or reverse-engineering (finding out how IDs are being generated), while a valid session is still in progress [23]. Once extensions have access to cookies, they can easily send the stolen session ID to third-party domains.

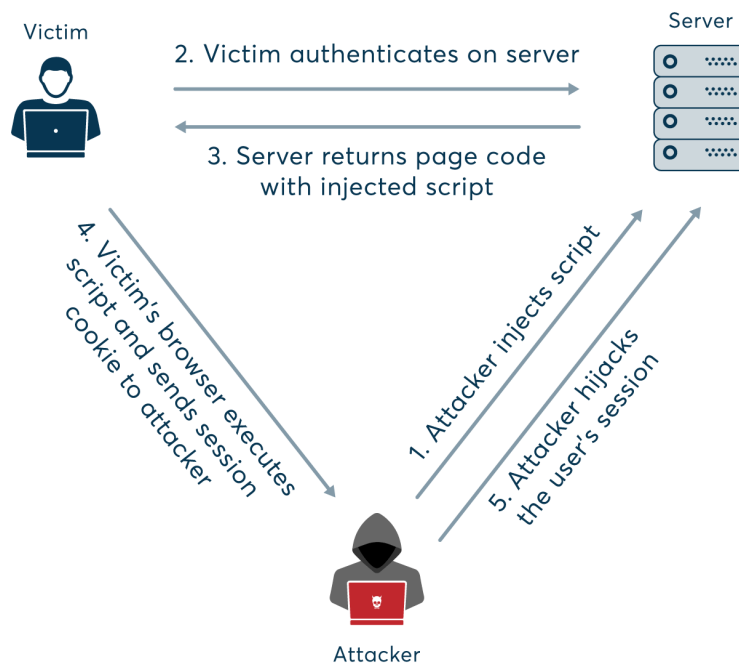


Figure 2.3: Session Hijacking using XSS [47]

In Figure 2.3, there is an example of a Session Hijacking Attack using XSS – manipulating a vulnerable website to return malicious JavaScript to a user – where the server accepts and executes unsafe input from the HTTP request.

2.2.2 Malvertising

There are innumerable ways to inject malware into the victim's device. Even with highly sophisticated antivirus software, browser extensions can be a genuine threat in websites that use ads to support themselves.

Advertisements – a powerful way to deliver brand messages to a potential customer – allow websites to monetize their work by giving a fraction of their space to ad networks, that act as brokers between advertisers and publishers [45].

Extensions have enough privileges to change the content of visited websites, resulting in severe consequences, since an attacker can inject malware ads on many popular websites. In addition, users may be unaware that they could encounter malicious content while browsing on highly-reputable websites.

Here, it is essential to mention that Malvertising is not the same as adware – a program running on a user's computer, usually used to display unwanted advertising, redirecting search requests to advertising websites and mining data about the user to help target or serve advertisements – as it only affects users viewing an infected website, while adware once installed, keeps operating on a user's computer.

Malvertising can perform various types of attacks on users without even clicking on the targeted ads, but just by viewing the ad. That is the case of a "Drive-by Download" Attack [22] (Figure 2.4) – consists of the installation of malware or adware on the computer of a user, mostly possible due to browser vulnerabilities.

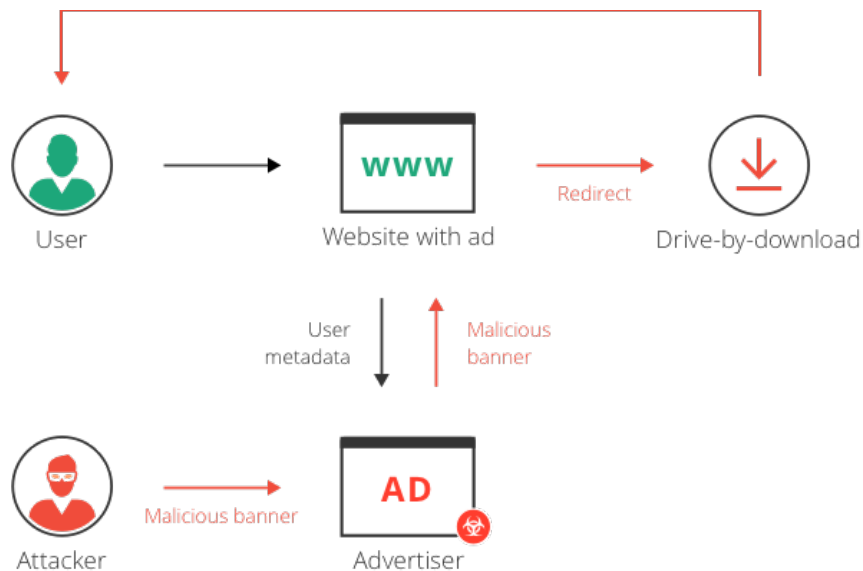


Figure 2.4: "Drive-By Download" Attack [22]

For example, extensions that use ad-injection to monetize their functionalities can participate in Malvertising by using advertising networks. Ad networks are extending their business by connecting advertisers to browser extension developers. A study on the Top 1000 Alexa websites concluded that around 4% of extension-injected ads were malicious. With this, a user installing an ad-injecting extension is more likely to be exposed to Malvertising, even if visiting only top-rated websites [45].

Due to the increasing efforts of large ad networks to mitigate Malvertising from their networks, browser extensions became a new route for malicious Malvertising activities as they provide more privileges to attackers.

Besides that, using JavaScript, it is possible to display unwanted advertising and malicious content beyond the ads legitimately shown by the ad network.

2.2.3 Browsing Tracking

When we accept all the permissions required by a specific extension, we are allowing it to use not only what is being displayed but other information that may or may not be relevant to find out who we are, what we want and what are we doing in that web browser, if those permissions allow gathering that same information. Here, it is important to reinforce that the main risk is not accepting all permissions that a specific extension asks but what permissions are being accepted.

Not all permissions have the possibility to become harmful, but a large percentage of extensions request permissions to all possible URLs, or at least to all non-HTTPS pages [38].

This can lead, for example, to spying extensions – using users browsing history for more purposes other than its core functionality, such as sending it to other domains or selling it to data mining companies [1]. Spying extensions usually steal our public IP Address (and geographic location), browsing activity, social media access tokens (cookies) and passwords that may be saved in our browser’s vault [2].

However, extensions can still not respect Private Browsing Mode (PBM) guidelines when handling private browsing data [49]. PBM (as known as Incognito Mode) is a feature supported by web browsers where no private browsing data is stored on disk [49], such as browsing history, passwords, cache, making life harder for attackers to steal personal information, as well as avoiding being tracked.

2.2.4 Supply Chain Attack

A Supply Chain Attack is an attack that aims to inflict damage to a certain company by exploiting vulnerabilities in its supply chain network [24]. Frequently targeted by hackers, supply chain network attacks can expose a vulnerability in a company’s supply network, which shows that an organization’s security is only as strong as the weakest link on the chain.

Supply Chain Networks have the advantage of reducing costs by using tools already developed and constantly updated by others, e.g., jQuery library [28] – open-sourced JavaScript library that allows web developers to add extra functionality to their websites.

In this type of attack, multiple entities can be affected since the security of the entire chain can be compromised by anyone that relies on it. In 2013 [37], Target was a victim of a supply chain attack when one of its third-party’s security credentials was compromised, allowing the attackers to gain access to login credentials and the entire Target’s network, resulting in a loss of millions of dollars and more than 70 million customers information.

In Figure 2.5, an ad network is compromised by injected skimming code that will then compromise the JavaScript library that is being loaded to multiple e-commerce websites. Following a purchase, the attackers will get all the payment information that could be used to steal bank accounts or even sell to other entities.

In browser extensions, an extension can be compromised by a vulnerability existing in one of the external libraries on which it is dependent or even if any malware attacks a server that is essential to run correctly.

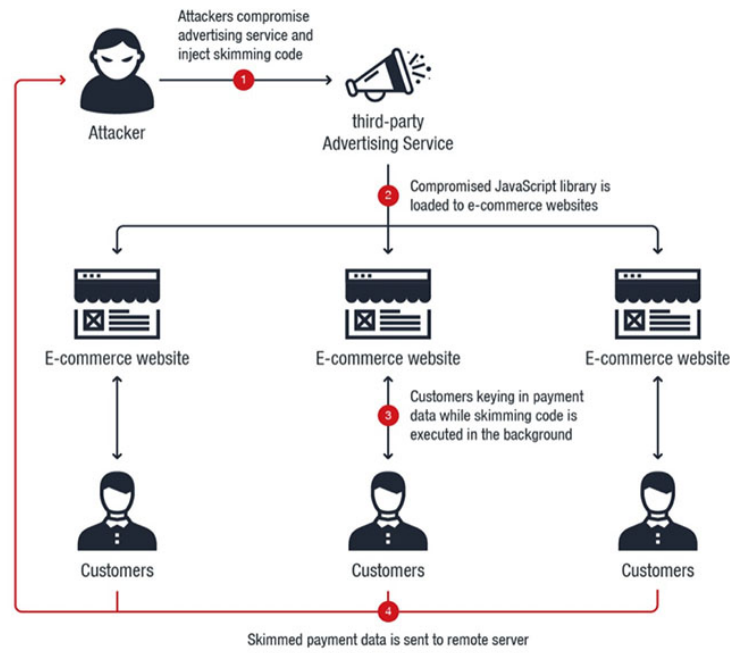


Figure 2.5: Example of Web-based Supply Chain Attack [48]

2.3 Related Work

Vulnerabilities on browser extensions, and even on browsers themselves have been known since 2006 [42], which motivated a lot of software companies and studies to create solutions, not only to detect but also to protect both clients and servers against the most frequent and simple attacks.

In this section, we present relevant tools to this dissertation’s topic, of which some are still in development.

2.3.1 Sabre

Sabre [18], Security Architecture for Browser Extensions, is a system that analyzes extensions by tracking their information flow, monitoring what data is being processed as well as which information is being sent and to which domain.

Built specifically for Mozilla Firefox extensions, this tool was implemented by modifying SpiderMonkey’s – Firefox’s JavaScript interpreter – data structures that represent JavaScript objects, by including security labels, thereby allowing information flow tracking across browsers subsystems.

Associating a pair of security labels with each in-memory JavaScript object in the browser and tracking them within the interpreter, **Sabre** raises an alert when a *JSE* accesses an important object unsafely, sends over the network a DOM node containing sensitive data (e.g., *form* or *script* element containing a password or cookie) or tries to write a new file containing the information

on a valuable object. Here, one label is responsible for tracking the flow of sensitive information, while the other tracks the flow of low-integrity information to detect possible violations of confidentiality and integrity.

Each security label consists of three information components:

- **Sensitivity Level** – determines whether the object is associated with sensitive data;
- **Boolean Flag** – determines whether the object was modified by a *JSE*'s JavaScript code;
- **Name(s)** of the *JSEs* that have modified the object.

Since **Sabre** tracks the execution of all JavaScript code (including code in web applications and browser core), it only raises an alert when an object is modified by a *JSE* in order to avoid false positives.

However, browser objects from *document* and *window* interfaces, used to interact with the DOM, are neither stored and managed by the JavaScript interpreter, requiring *Sabre* also to modify the browser's DOM subsystem, assigning a security label to each DOM node that will later be transmitted by the browser to the JavaScript interpreter when accessed by any *JSE*. Adding to these features, **Sabre** also detects when an extension is trying to execute a script with high-level privileges from an unreliable domain.

Twenty-four different extensions were collected for testing, 4 with known malicious flows and 20 popular Firefox extensions. This tool was tested and suited in 5 categories:

- Interaction with HTML forms;
- Sending/Receiving data over an HTTP channel;
- Interaction with the File System;
- Loading URLs;
- JavaScript events.

In the end, it was able to detect confidentiality and integrity violations, caused either due to malicious functionalities or exploitable vulnerabilities in the source code.

2.3.2 Hulk

Aiming to detect malicious behavior in browser extensions, **Hulk** [29] is a dynamic analysis system that monitors their execution and corresponding network activity by dynamically loading extensions in a monitored environment and observing the interaction of extensions with the loaded web pages. Using a set of heuristics to identify potentially dangerous behaviors, it labels extensions as malicious, suspicious, or benign.

First, it extracts URLs from three sources – manifest, source code and popular websites – to be able to trigger the extension's functionalities, and then proceeds to leverage **HoneyPages**, which

are dynamic pages created to satisfy and trigger a specific extension by fulfilling its structural conditions by checking for new *iframe* and *div* elements, which could be considered as malicious behavior, e.g., if an extension queries an *iframe* DOM element to modify it, a HoneyPage will create an *iframe* tag element, inject it in the DOM tree and return it to the extension. Essentially, these pages create an environment for the extension to perform its actions while all its interactions within a HoneyPage are being recorded.

Complementing this approach, an event handler fuzzer attempts to execute event handlers that are being listened to by a specific extension. At the same time, **Hulk** monitors the changes that the extension is attempting to make on a loaded HoneyPage. During its execution, it also uses a transparent proxy to intercept all the HTTP requests and DNS traffic, while identifying extensions that contact outdated and suspicious domains.

As for labeling them, **Hulk** considers an extension as malicious when it identifies harmful behaviors, suspicious when there is a presence of potentially dangerous actions and finally, benign when none of these are found.

Using only dynamic analysis, it relies only on upon triggered malicious activities, not providing a complete view of the extension behavior. Also, a determined adversary with knowledge of the system could try to evade **Hulk** by querying for random elements in the DOM tree first and, if found, avoid malicious activity.

Using a dataset of nearly 48,000 extensions from the Chrome Web Store, Hulk labeled 130 as being malicious, 4712 being suspicious and the rest as benign.

2.3.3 Expectator

We already described one of the most used malicious activities, Malvertising (2.2.2), and to identify ad-injecting extensions, **Expectator** [45] is a measurement framework that automatically identifies browser extensions that inject ads on web pages.

According to their authors, there are three main types of ad-injection:

- **Search Result Pages** – replacing search engine ad results;
- **Retail Websites** – overlaying existing online retailers content (e.g. Amazon, Ebay) with third-party ads with similar or related products;
- **Unrelated Websites** – pop-ups, alerts or just advertisements on every web page.

First, **Expectator** performs a static analysis on the extension's source code to identify the websites where the extension is triggered. After that, it installs a virtual proxy between the extension and the browser's JavaScript engine to intercept third-party scripts with ad-injection code that could be loaded at run-time, using remote debugging protocols supported by browsers (e.g., Chrome⁹).

Extensions can inject ads not only on visited websites but also on specific events (e.g., extension code may register an event listener on a product image of a retail site and listen to the

⁹Chrome DevTools Protocol, <https://chromedevtools.github.io/devtools-protocol/>

MouseOver event). Fighting this, **Expector** tries to trigger as many as possible potential events that the extension might be interested in by instructing the browser to execute the corresponding JavaScript function for each DOM element that has a JavaScript event handler to process user events. Since some extensions may only inject ads after some time being installed (attempting to hide from extensions reviewers), **Expector** sets a random foretime before installing an extension and setting the time correctly after launching the browser.

As for detecting extension-injected elements, this tool tries to identify the suspicious DOM elements by comparing the HTML source of two different pages – one with the extension loaded and another unloaded. This approach, however, can result in a large number of false positives in situations where each reload of a single website results in ads served from different ad providers. To address this problem, **Expector** uses both DOM structure and content of its elements to identify potential injected elements.

To identify extension-injected ads, it instructs the browser to visit the element's URL, if there is any associated, and if not, to trigger a click event in order to emulate a user clicking inside that element. If an element contains an ad, the advertiser's website will be loaded through a series of redirects, first to one or more ad networks and finally to the ad landing page. Detecting these redirection patterns is a task for a lightweight implemented Chrome extension that logs all the HTTP requests made by the browser. When an HTTP request matches one of the above cases, it is sent for further analysis. It is considered a redirection when the last domain is different from the original domain where the ad is located.

After downloading the entire list of available extensions from Chrome Web Store (around 18,000 by March 2014), **Expector** reported 303 extensions as adware. After manually installing and inspecting their source code, only 292 of them were effectively adware. Using those 292 extensions against the Top 1000 Alexa websites, none of the websites had any malicious ads and about 4% of the extension-injected ads on these websites were indeed malicious, implying that by installing an ad-injecting extension, a user is more likely to be exposed to Malvertising than visiting highly popular websites.

So far, **Expector** is only designed to inspect Google Chrome extensions, but there are plans of extending it to be able to process extensions of other popular browsers.

2.3.4 BrowsingFog

When dealing with malicious intentions, it is not always about malware, as extensions can violate privacy limits with the proper permissions, since they have access to your browsing history and cookie storage.

Helping to identify extensions that can be leaking our browsing record, **BrowsingFog** [39] is a browser extension that aims to protect the user's privacy by simulating visits to a dataset of webpages of random categories while hiding the user's real areas of interest inside a "fog" of irrelevant pages.

Through dynamic analysis and user interactions, this extension was tested against the 10,000 most popular extensions on Chrome Web Store, while trying to detect any leaking that might be happening due to extensions behaviors.

Installing each extension in an instance of Google Chrome browser, using core automation, **BrowsingFog** works by opening a new tab every minute, amounting to four-page visits per category during one hour. These preferences values can be changed to match the amount of traffic that the user usually produces in that amount of time. In this process, all the traffic originating from the monitored browser is channeled through an instance of mitmproxy¹⁰ which captures packet traces on all ports greater than 22, capturing standard HTTP/HTTPS requests as well as outgoing traffic using non-standard ports, e.g., enabled by extensions with proxying capabilities.

Privacy leakage is considered to be an event where any private data is transmitted to a third-party server, and to measure extension-based leakage, **BrowsingFog** focus on 4 different types of leakage of private information:

- **History Leakage**

When visiting a certain amount of URLs and subdomains, the resulting number of requests allows the detection of accidental leakage of browsing history via the HTTP Referer header (e.g., when an extension embeds a remote object on every visited webpage);

- **Search Query Leakage**

Automating Google searches and looking for the queries in outgoing traffic towards unrelated third parties;

- **Form Data Leakage**

By submitting a mock HTML form with previously defined values on several URLs and looking for these values being sent to a third party in the outgoing traffic;

- **Information about other Extensions**

Looking for the presence of extension identifiers in outgoing traffic when installing multiple extensions in a single instance of Chrome;

It is also important to mention that, to be considered an "authorized" leaking, if an extension that requires the capture of private information to achieve its stated goals, it should provide an acceptable privacy policy in which describes precisely what data can be collected and how that data is going to be handled and properly stored.

In Figure 2.6, **BrowsingFog** was running while an Instant Translator extension was enabled and it is possible to observe all the websites that were leaked by that same extension.

Overall, around 38% of the extensions used to test this approach were making new requests towards third parties, averaging six different third-party hosts by each extension.

¹⁰mitmproxy, <https://mitmproxy.org>

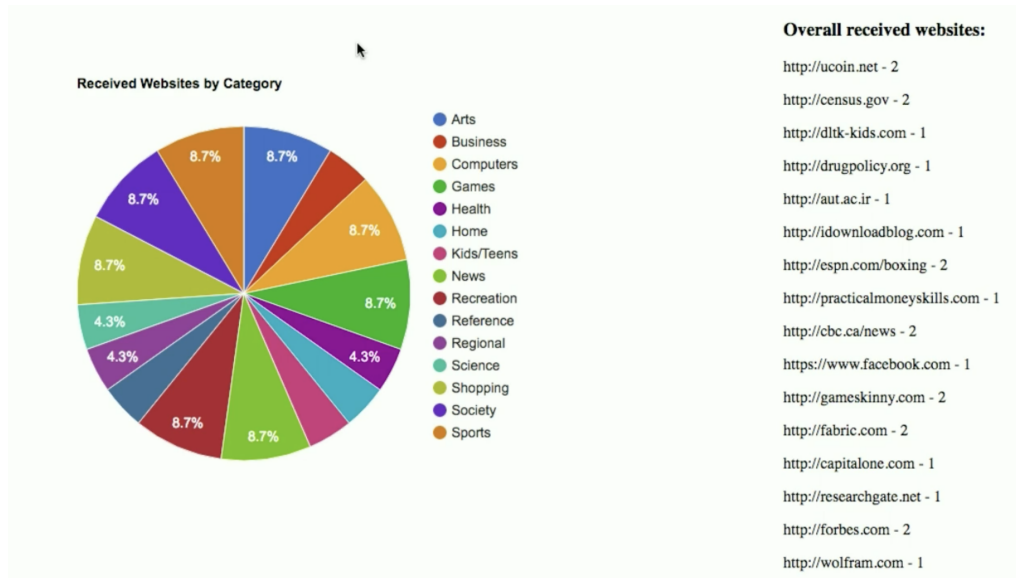


Figure 2.6: BrowsingFog Demo [17]

2.3.5 XHOUND

As previously mentioned, web tracking is becoming an increasingly intrusive and sophisticated technique since most advertisers are trying to capitalize on users' online activity. For this, browser fingerprinting is a common technique as it allows trackers to identify users uniquely, even if they navigate on a private mode or clear their web browser cookies [38].

Being the first fully automated system for fingerprinting browser extensions, **XHOUND** (Extension Hound) [38] was developed to quantify the fingerprintability of browser extensions by detecting and analyzing their side effects on a page's DOM.

Based on a combination of static and dynamic analysis, **XHOUND** fingerprints the organic activity of extensions in a web page's DOM, such as the addition of new DOM elements and removal of existing ones, being thus robust against incremental patching of extensions.

At the time of their article [38], an examination of the Top 10,000 Chrome Web Store extensions was able to show that roughly 9,2% of them introduce detectable changes on any domain, growing to 13,2% if considering just the Top 1,000.

When considering the fingerprintability of extensions, it is possible to distinguish two possible types of tracking scripts by the domain in which they operate:

Tracking Scripts on an Arbitrary Domain – these tracking scripts can be loaded inside any web page, regardless of its URL. Here, the detectable functionality of these fingerprintable extensions must be content-dependent instead of URL-dependent. One common content-dependent example is the changes made by LastPass¹¹ in Listing 2.5. This extension will make these visible changes on any password-type input field, making it fingerprintable.

¹¹LastPass, <https://www.lastpass.com/>

```

1 <input type="password" style="cursor: pointer; background- image: url(&quot;data:
   image/png;base64, iVBORwOKGgoAAAANSUheUgAAABAAAAA...yeBAAAAAE1FTkSuQmCC& quot;);
   ; background-attachment: scroll; background-size:
2 16px 18px; background-position: 98% 50%; background- repeat: no-repeat;">

```

Listing 2.5: DOM changes introduced by LastPass [38]

Tracking Scripts on a Popular Domain – some extensions are only targeted to work on specific popular websites such as social networks and streaming platforms, where their URL matters, making them URL-dependent. A popular extension called SaveToPocket¹² is known for the ability to save web pages (or parts from them) for later reading, while adding a personalized and distinctive button to the element that is designed to save. As seen in Listing 2.6, these changes could be fingerprintable by any script loaded, whether from Twitter’s domain or third-party scripts loaded from an arbitrary remote domain.

```

1 <div class="ProfileTweet-action action-pocket-container"> <a class="js-tooltip"
   href="#" data-original-title="
2 Save to Pocket">
3 <span class="icon icon-pocket"></span> </a></div>
4 <div class="ProfileTweet-action...

```

Listing 2.6: DOM changes introduced by SaveToPocket [38]

The possibility of extension-based fingerprinting relies on a tracker’s ability to distinguish which modification was introduced by what extension.

In Figure 2.7, it is possible to visualize **XHOUND**’s architecture, as well as its data sources. Following a similar approach to Hulk’s Honey Pages (2.3.2), **XHOUND** unpacks each extension and patches automatically with JavaScript code to intercept most of the queries that an extension may use to locate DOM elements. For example, it patches the *document.getElementById* method so when an extension is querying for a specific ID, the library developed for this purpose – **On-TheFlyDOM** – will create it on-the-fly, record that same creation and return it to the extension script that queried it. Since an extension’s logic may search for elements on elements subtree, all the DOM elements created on the fly are also patched with intercepted functions.

¹²SaveToPocket, <https://chrome.google.com/webstore/detail/empty-title-niloccemoadcdkdjlinkgdfekeahmflj>

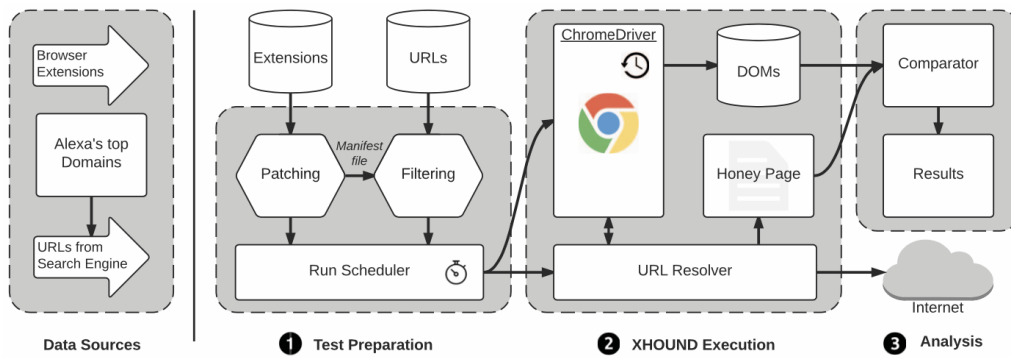


Figure 2.7: XHOUND Data Sources and Architecture [38]

These dynamic honey pages are not sufficient for revealing all possible cases, as a significant number of extensions are only content-based, modifying only the content that matches their internal logic. Facing this problem, static honey pages were also created with various types of content, including tracking scripts, audio/video tags, login forms, etc.

Using Selenium¹³ ChromeDriver¹⁴ and a set of scheduled navigations, XHOUND proceeds to visit the Top 50 Alexa domains¹⁵ and up to 20 URLs for each popular domain. At the same time, all initial DOM elements on the honey pages are indexed for further comparison of the potentially modified DOM page against the original one. Also, each queried element (created on the fly) is recorded to know later if an extension modified it.

Currently, **XHOUND** supports both Google Chrome and Mozilla Firefox extensions, as their **OnTheFlyDOM** library is able to patch the most popular DOM-querying methods (*getElementById*, *getElementsByName*, *getElementsByClassName*, etc).

2.3.6 CRXCavator

CRXCavator¹⁶ was first created as an internal tool by *Duo's Corporate Security Engineering* team to solve a real problem and later made publicly available to anyone who wanted to take advantage of its main purpose: scanning an extension and get a risk score value based on several factors.

This tool automatically scans the entire Google Chrome, Mozilla Firefox and Microsoft Edge Web Stores every 3 hours, producing, for each version of an extension, a quantified risk score based on a set of properties¹⁷:

- **Extension Metadata**

Metadata information about the extension retrieved from the Chrome Web Store (name, description, rating, active users, privacy policy, developer e-mail/address, etc).

¹³Selenium WebDriver, <https://selenium-python.readthedocs.io/api.html>

¹⁴Chrome WebDriver, <https://chromedriver.chromium.org/>

¹⁵Alexa, <https://www.alexa.com/topsites>

¹⁶CRXCavator, <https://crxcavator.io/>

¹⁷CRXCavator - Report Breakdown, https://crxcavator.io/docs.html#/report_breakdown

- **Risk over Time**

Plots the risk score between different versions of the extension to show how the risk of the extension has changed over time.

- **Permissions & Optional Permissions**

All the permissions listed in the extension's manifest.

- ***RetireJS* Vulnerability Scan**

Results of a *RetireJS* vulnerability scan, showing the extension's files where the vulnerabilities were found in, as well as the severity of them.

- **Content Security Policy**

The Content Security Policy allows analysts to determine what the extension is talking to, which is vital to the security of an extension.

- **Potential External Communication**

A guess at possible outgoing domains where an extension is making connections.

- **Manifest**

Contains the manifest details of the extension.

- **External JavaScript Files**

All the external JavaScript files loaded into the extension – a change in the JavaScript file could result in a behavior change without publishing an update to Chrome Web Store.

- **Entry Points & Dangerous Functions**

Any entry points and dangerous functions found within the extension, where an attacker could potentially inject code into the extension or cause an unexpected behavior change.

- **Related Extensions**

Related extensions can be helpful by potentially find a more secure alternative.

For each section, **CRXCavator** assigns a risk score depending on the severity of each subsection, obtaining a risk value summing the risk value of each section, giving the possibility to see in which section the extension presents a greater risk and what are the reasons for this higher risk.

Besides that, **CRXCavator** offers a well-documented API¹⁸, allowing anyone to submit an extension ID to be scanned, and most importantly, to obtain a report for a specific extension ID. Also, providing the source code of each analyzed extension, this tool has a web application where it is possible to easily search for any extension through its name or ID and view the report of any version that has been analyzed.

Since October 2018, the company behind **CRXCavator**, *Duo's Corporate*, is owned by Cisco [15].

¹⁸CRXCavator - API Docs, <https://crxcavator.io/apidocs>

2.3.7 Yao Wang et al.'s Combined Static and Dynamic Analysis Approach

Recently, a 2016 Symantec's Internet security threat report [40] manifested that the presence of potential vulnerabilities and malicious extensions is on a sustained rise by evidencing the flawed vetting process that malicious extensions pass through before being available on browsers' official stores.

In May 2018, Yao Wang et al. [44] published a research article entitled "A Combined Static and Dynamic Analysis Approach to Detect Malicious Browser Extensions", proposing a machine-learning-based approach to detect malicious extensions by extracting features from static and dynamic techniques that on a later stage are used to feed several detection models.

Before explaining their approach, it is essential to mention that they focused on Google Chrome only and were able to gather a dataset consisting of 3,187 benign extensions and 1,490 malicious extensions, taking them about 14 months to collect the extension samples and, to ensure that benign extensions were indeed harmless, they downloaded 5-star extensions from Chrome Web Store (claiming that extensions with high reputation are less likely to contain malicious intentions) and made a further validation by manually reviewing them and using detection tools. As for malicious extensions, they collected from Chinese forums¹⁹ and drew from third-party extension sites for low-score extensions. Once again, they validated them by using detection tools and manual review.

Starting by their static analysis, a scan is performed to an extension's package, searching specifically for these features:

- **Manifest Permissions**

Using the Top 10 most used permissions required by their malicious extensions, they set 1 or 0 to present whether a permission exists in an extension's manifest file, making use of each one of these ten permissions as the permission feature in their feature set;

- **Content Scripts and Background Pages**

A total of 12 features are extracted from JavaScript code files, of which 5 were proposed by previous works [7] and [9]: white-space percentage; average line length; occurrence frequency of specific characters; word size; entropy of strings. Adding to these, 7 features were also elaborated based on the number of occurrences: dynamic code generation functions (e.g. *eval*, *setTimeout* and *setInterval*), HTML DOM change methods (e.g. *document.write*, *innerHTML* and *outerHTML*), event handlers (e.g. *addEventListener*), HTTPS scripts (external scripts that are loaded over HTTPS), number of HTTP Header modification callbacks (*webRequest* API callbacks), XMLHttpRequests (used to access the network) and finally, keywords density.

- **HTML Features**

Analyzes 4 features from extension's HTML pages, capturing the number of certain HTML tags (i.e. *script*, *body*, *iframe*, *img*, *form*).

¹⁹Cardfan - Technology News - Computer Security, <https://www.kafan.cn>

- **CSS Features**

Analyzes 3 features from extension's CSS files, capturing the amount of occurrences of CSS properties (i.e. *background-image*, *behaviour*, *@import*).

When it comes to dynamic analysis methods, they made use of Chrome's built-in activity logger²⁰ and its developer tools to record Chrome Extension API calls, DOM operations and network requests.

After running each extension of their dataset, they monitor the usage occurrences of multiple APIs, such as *management.uninstall*, *tabs.remove*, *webRequest* Callbacks and *tabs.query*. Prior to that monitoring, an inspection is done to 8 DOM operations (i.e. *createElement*, *getElementsByClassName*, *getElementById*, *getElementsByName*, *createElementNS*, *appendChild*, *navigator* and *location*). From network logs, they extract nine features (mostly the number of occurrences) that they consider to characterize HTTP sessions: GET/POST requests, other methods requests (e.g., PUT, DELETE), requests towards script files, requests with (1-5)XX status codes, average request substring length and query string parameters.

On Feature Selection Phase (Figure 2.8), they run Random Forest Algorithm to obtain the importance score of each one of their 51 features, where the ten permissions previously featured topped the list.

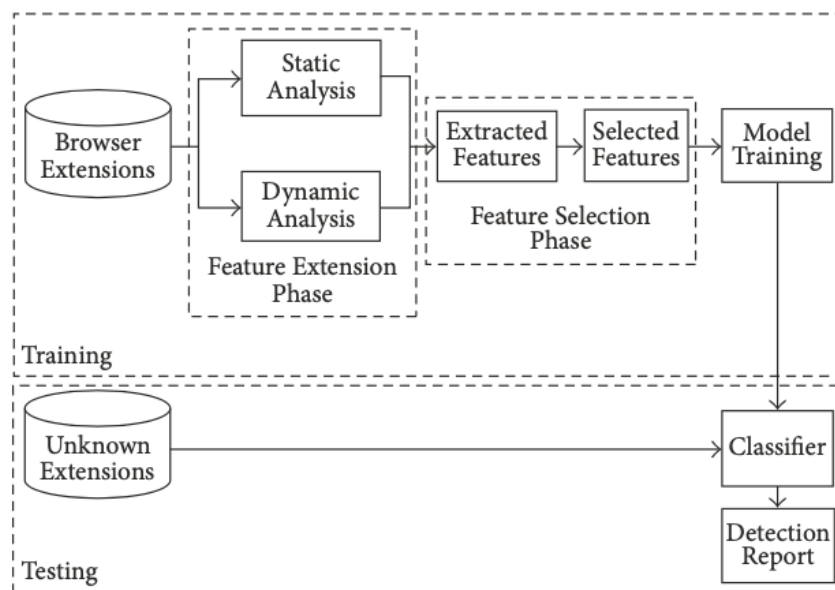


Figure 2.8: Workflow of Machine-Learning-based detection [44]

After static and dynamic analysis being completed, 4 detection models were implemented using scikit-learn machine-learning framework²¹: Support Vector Machine (SVM), Bayesian Network (BN), Logistic Regression (LR) and Multilayer Perceptron (MLP).

²⁰Chrome Apps & Extensions Developer Tool, <https://chrome.google.com/webstore/detail/chrome-apps-extensions-de/ohmmkhmmmpcnpijkeljgnaobkaalbgc>

²¹Scikit-learn: machine learning in Python, <http://scikit-learn.org>

Finally, 5-fold cross-validation was adopted to evaluate all four models, with SVM having the most accuracy (percentage of correctly classified samples), precision (quantity of True Positives divided by total True Positives and False Positives), recall (proportion of TP and sum of TP and FN samples) and F1-score (harmonic mean of precision value and recall value) with LR the least performance of all.

2.4 Comparison with Related Work

Related work revealed a good variety of approaches, from dynamic information-flow analysis, Machine-Learning-based models and fingerprintability quantification of browser extensions.

Since we plan to use a different dataset, with different metrics, we cannot directly compare our results with any related work. Still, we can highlight similarities and where we think our solution (3) stands out from other solutions.

SafeWare shares the risk score component of CRXCavator (2.3.6), but the latter does not contain any dynamic factor, being only static-based. On the other hand, CRXCavator tracks and analyzes extensions from multiple official stores over time, presenting a Risk Over Time graphic where it is possible to see the evolution of an extension's risk score across multiple versions, which is an excellent feature since it allows to detect any significant increase/decrease from one version to another. Through our dynamic analysis, we can identify more accurately which interactions with the browser's API are occurring and which permissions are actually being used and how they are being used.

As for the Machine-Learning-based model solution (2.3.7), our work is similar in some aspects on both analysis (static and dynamic), with their analysis being slightly more complex by extracting more features from JavaScript code files (e.g., dynamic code generation functions, HTML DOM change methods, event handlers, etc.) and performing a network logs analysis (counting the number of GET/POST methods and filtering requests by their status code), while sharing with us the same capture of interactions between extensions and browser's API. Here, our static approach focuses more on the extension's available metadata (Chrome Web Store, and if not available, CRXCavator database) and its manifest file properties, while taking advantage of CRXCavator's *RetireJS* vulnerabilities report and Google SafeBrowsing API for checking if Google already banned a specific extension from its official store. Adding to that, we also search for all URLs and IPs in the Google SafeBrowsing API database, attempting to check if any is related to known malware.

Having a manually reviewed dataset, they were able to implement a Machine-Learning-based model using a scikit-learn machine-learning framework²² for implementing their detection models, something that we were not able to implement due to having a very limited dataset of verified malicious extensions during most of the time this work was carried out.

Nevertheless, we complemented our approach with a system that aspires to present a representative value of an extension's risk based on all the properties extracted and on demonstrative weights for each property, according to its severity and potential risk to the end-user.

²²Scikit-Learn: machine learning in Python, <http://scikit-learn.org>

Other tools, such as Expector (2.3.3) and XHOUND (2.3.5), also use Selenium for dynamic automation, with the first using with Node.js²³ and the latter with ChromeDriver, similar to our method.

2.5 Summary

Malicious browser extensions and vulnerabilities on browsers have been known for a long time [42], and there has been an effort in the past decade to improve web security and to understand the potential damage that users are exposed to when they install and enable such extensions.

Both static and dynamic approaches can achieve good results but are far from perfect, as browsers and web protocols evolve. There are always new ways and techniques to attack client and server sides, but it is tough to automate a process that assesses the true intent of an extension and uses it on a large scale without human interaction.

State-of-the-art approaches showed that this is a real problem. Several approaches attempt to combat the existence of malicious extensions in the different official stores. However, none was fully able to classify with certainty an extension as malicious, but it was possible to identify vulnerabilities and suspicious behaviors that can tell us a little bit more about an extension, and here, the most important thing is to be aware of the risk we run as active users of these same extensions.

Each of the reviewed tools has its own approaches, some even having similarities, from dynamically monitoring each in-memory JavaScript object to just statically analyzing source code. There is potential to obtain good results using a combination of some of these methods. While most of them are designed to detect malicious extensions, BrowsingFog (2.3.4) aims to detect leakages and XHOUND (2.3.5) attempts to quantify the fingerprintability of browser extensions.

It is impossible to compare them directly, as even those with a similar objective used different datasets at different date times. However, we highlight the Machine-Learning-based approach (2.3.7) as the most advanced, not just because it gathers a more complete and manually revised dataset but also for presenting a comprehensive hybrid approach linked to a machine learning detection model.

Finally, a mention to CRXCavator (2.3.6) for being a multi-browser extension public database and, in addition to allowing the submission of extensions for analysis, it also has an API to obtain reports in JSON format, which, together with its static analysis, are useful to complement any work.

²³Node.js, <http://nodejs.org/>

Chapter 3

Hybrid Approach to Quantify Extension Risk

Unlike JavaScript code in Web Applications, JavaScript-based browser extensions are not constrained by the same-origin policy. That allows some extensions to misuse their privileges and compromise the application integrity and confidentiality by stealing sensitive information or allowing the execution of third-party code on the host system. Facing this problem, it is relevant to identify browser extensions and profile their behavior on any web page. Our main goal was to create a solution using static and dynamic approaches to identify/classify an extension as malicious/suspicious or benign. Still, we developed a risk scoring system to quantify an extension's risk to the user.

Initially, the main plan was to develop an extension for the Google Chrome web browser, as it seemed a good place to run our tool since we could interact with multiple running extensions while analyzing both their behavior and source code. However, after spending some time to learn the basics and life cycles of an extension, we realized that it was not efficient to be constantly monitoring the source code of each extension and we had no guarantee that our extension would capture all the injected scripts if executed after other extensions, as others could be executed first, which would lead us to miss some possible important scripts and behaviors to capture.

The implementation of an extension did not look promising, and given our basic knowledge of the Chrome API, we chose to focus on the static analysis methods to advance in work and realize which static factors can influence or signal an extension as being malicious.

We discarded very early the possibility of having extensions with obfuscated code, given that since the beginning of 2019, Google only accepts extensions in its store whose code is not obfuscated [8]. This helped our work, since it would make some static analysis tasks more difficult, e.g., searching for source code patterns.

3.1 Methodology

Supporting the fundamentals and results of this work, a dataset built by us will be present in all tests, based on benign extensions and on extensions that we know to be malicious, in order to be able to compare the results and evaluate our solution.

Our strategy consists mainly of three stages supported by a dataset of extensions and a connection to the Internet.

During the first stage, we run our dataset of both benign and malicious extensions inside a controlled environment while we monitor and record every Chrome Extension API call and Network activity between our extensions and a set of given websites (see Section 4.1). Although this stage is primarily automated, it also contains some manual human interaction (browsing through social media, login/logout, searching, etc.) to gather more and better information to be later used on the last stage.

In the second stage, we analyze all the static information presented in the extension's source code, extracting relevant features from the Manifest file, searching for patterns in all the files available and checking if any have malware associated with them.

Finally, the last stage is responsible for merging both static and dynamic analysis data by applying a risk scoring value to each extension and generating a report on the dataset used. Each extension gets a score corresponding to an accumulation of multiple factors that can be considered high/low suspicious, which should indicate or represent a level of suspicion on possible malicious intentions or even a level of access that we are granting to an extension.

Since initially our dataset only contained ten verified malicious extensions, we could not build a Machine-Learning-Based model trained with the features provided by both approaches and possibly identify with a better confidence level if an extension is indeed malicious or not. Instead, we came up with a risk scoring system that intends to quantify the amount of risk present in an extension through the factors extracted from our static and dynamic analysis, thus combining the two approaches by assigning a risk score to each extracted feature – a value that we consider to be representative not only of the risk to the end-user but also the severity of this feature concerning other factors of greater or lesser risk.

3.2 Static Approach

Starting with the static analysis, we wrote a script to unpack all the Chrome Extension (CRX) files into a folder containing all the extensions unpacked and unzipped. Each extension has its directory identified by its Chrome Web Store (CWS) ID, which will later allow us to merge all the information collected from both dynamic and static analysis.

After having all the unpacked extensions in a single directory, the main script, SafeWare, starts by reading and indexing all the extensions inside our dataset, then extracts their IDs and compares them with Google Safe Browsing API¹ list of known malicious extensions IDs, already banned

¹Google SafeBrowsing API, <https://developers.google.com/safe-browsing/v4>

from Chrome Web Store. If we have a match, that extension is automatically marked as banned. This public list is updated every day as Google keeps adding and banning more extensions from their store, but for this work, we will download the list right before starting this stage.

Next, if the extension is available on the Chrome Web Store, we scrap from there all the relevant properties, such as name, version, developer (e-mail, address and privacy policy), last updated date, website, number of users and rating. This step is important as not all extensions have all these properties and we will take that into account later in our risk scoring system.

We then extract the local version of each extension through its manifest and we request a CRXCavator² report if there is one in their database for that specific version. This is important when we are analyzing an extension that is no longer available on Chrome Web Store as it may have been reviewed by them before it was banned. If there is an available report for a banned extension, we will use its *webstore* metadata as its latest Chrome Web Store available metadata.

In that report, we check for the *Content Security Policy* [11] (CSP) – a policy to mitigate against cross-site scripting issues – and *Retire.js*³ – an open-source project that helps detecting the use of JavaScript-library versions with known vulnerabilities – values as both should help to determine if the extension is complying with the CSP for Chrome Apps (which is vital to the security of an extension) and how many known vulnerabilities were found, according to a degree of risk established by CRXCavator Team.

Since we have access to every single line of code that makes an extension, except scripts that can be downloaded while running, we extract all URLs and IPs from every file inside an extension source code and we look them up through Google SafeBrowsing API database to check if they were used in previous malware attacks or even in malicious extensions that ended up being banned.

Several APIs were tested, such as IPQualityScore⁴. However, most of them are APIs with a limited number of requests, and given the high number of requests necessary to send, we do not have the resources to support these APIs. Hence we have relied on Google SafeBrowsing's API, which despite not being so advanced, serves for the demonstration and intention of this work.

Last but not least, we look for potential keylogging and screen capturing extensions by searching for JavaScript patterns (event listeners, methods, etc.) to check if an extension is listening for any pressed key or capturing any visible tab, regardless of what might be doing with it.

3.3 Dynamic Approach

Although we can predict some behaviors on a purely static analysis, it is essential to run these extensions against a set of websites and multiple tasks while watching what they are doing in real-time behind our screen.

²CRXCavator, <https://crxcavator.io/>

³Retire.js, <https://retirejs.github.io/retire.js/>

⁴IPQualityScore, <https://www.ipqualityscore.com/>

The end-users of extensions associate extensions to pop-ups, graphic modifications, hiding our real public IP address and ad-blockers, but there are a lot of actions that these extensions take to achieve their advertised purposes that might be compromising our information, privacy and, in some cases, our machine's processing power [46].

Moving to the dynamic analysis, we started by detecting possible mutations on the website and monitoring the access to important variables such as authentication tokens and session IDs. Still, shortly afterward, we discovered an app, *Chrome Apps & Extensions Developer Tool*⁵, made by Google and last updated on June 2014, that provides information about the connection between extensions and Chrome Extension API by monitoring all the calls made during their execution. (Figure 3.1)

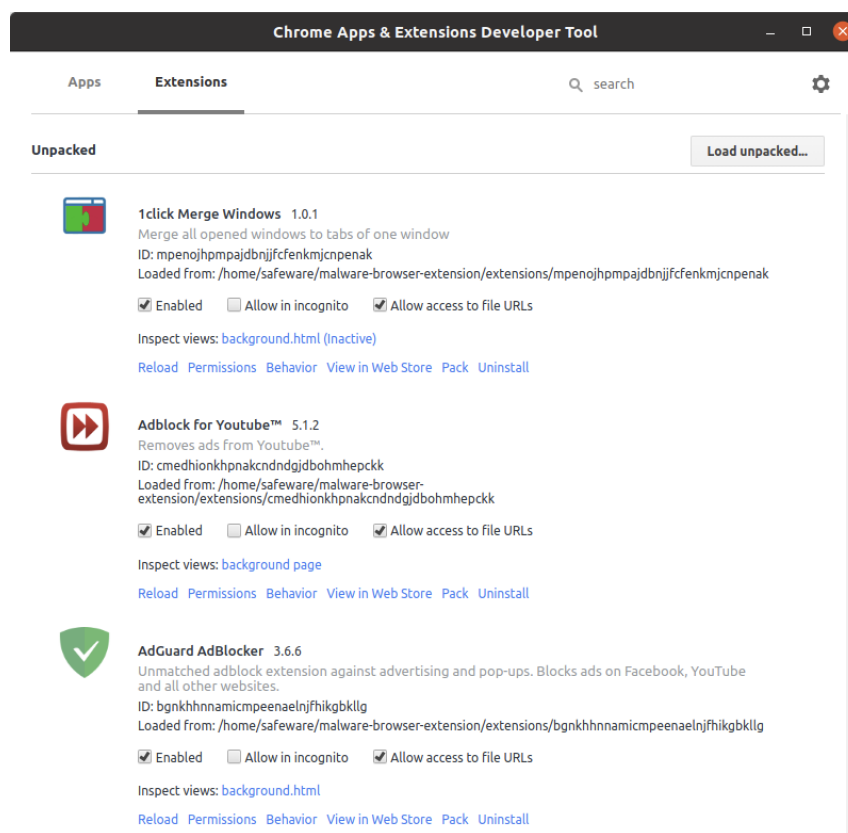


Figure 3.1: Chrome Apps & Extensions Developer Tool GUI

After testing it with a small set of extensions from our dataset, we decided that it was an effective and promising tool to register each extension's activities in our later automated simulation. (Figure 3.2)

⁵Chrome Apps & Extensions Developer Tool, <https://chrome.google.com/webstore/detail/chrome-apps-extensions-de/ohmmkhmmmpcnpikjeljgnaobkaalbgc>

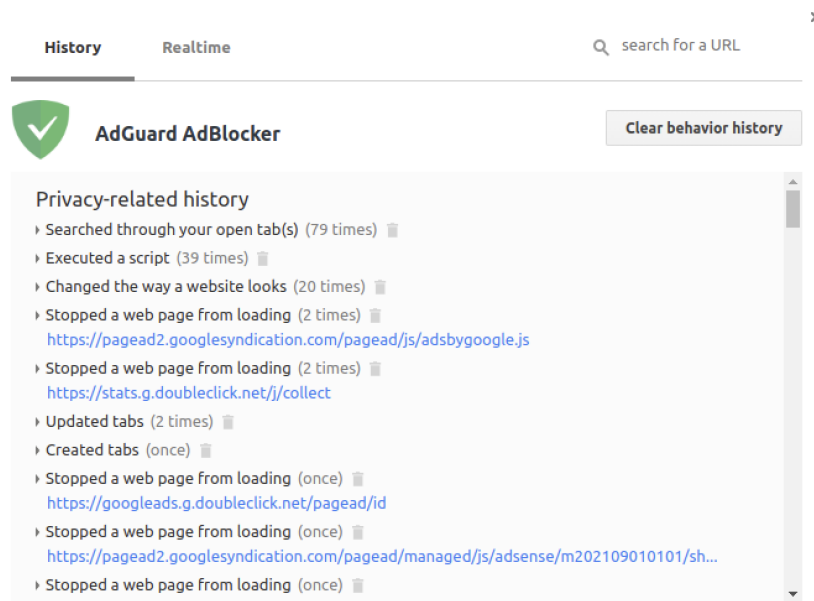


Figure 3.2: Chrome Apps & Extensions Developer Tool – Extension Activities

Since the mentioned app was made public on Google Archives GitHub⁶, we extracted the source code to add a new functionality – download all the registered activities filtered by each extension ID – to read this information in our main script when merging both static and dynamic analysis.

This way, we can later filter each activity inside the extension by their activity type (i.e., *API Call*, *API Event*, *DOM Access* or *Web Request*) and inside each type, filter them by their Callbacks, so that we can later have an idea on what exactly is the extension doing and if that activity can be harmful to the user's privacy.

Last but not least, we decided to implement a network activity logger to register all the network requests that each extension of our dataset does. Here, the goal is to filter them by extension ID and then inspect each activity element, which server IP address is being communicated, as well as its content and methods. This technique can help detect any suspicious behavior if an extension uploads any information to a non-trusted server or downloads any malware into our device.

Even with automated and manual interaction in our simulation, we do not have any guarantee that all the actions, malicious or not, will be triggered, as more complex extensions might need a more profound and focused interaction to activate a specific set of actions.

For the purpose of this work, we are targeting only simple navigation tasks such as opening a new empty tab, entering a website, scrolling up and down, saving network logs and closing a tab. This process is executed in each automated visited website, with hard-coded intervals between each action. After that, we manually access a list of websites to simulate a more human behavior such as logging in on social networks, searching for results on search engines, shopping

⁶Chrome Apps & Extensions Developer Tool – GitHub Archive, <https://github.com/googlearchive/apps-developer-tools>

on online stores, browsing video platforms, etc. Although it can all be automated, we are aware of JavaScript-based Bot Detection scripts, which can be present not only on websites but also on extensions, so we also opted for a manual process, hoping to capture more behaviors from the installed extensions.

At the end of this simulation, we obtain a JSON file with all activities between extensions and Chrome Extension API (Listing 3.1) and a network log for each website visited.

```
1 "aamjlohoaogjmfddgdjojghjaamgmhdhj": [  
2   {  
3     "activityId": "2",  
4     "activityType": "api_call",  
5     "apiCall": "tabs.create",  
6     "args": "null",  
7     "count": 1,  
8     "extensionId": "aamjlohoaogjmfddgdjojghjaamgmhdhj",  
9     "time": 1629376291048.796  
10  },  
11  {  
12    "activityId": "3",  
13    "activityType": "api_event",  
14    "apiCall": "runtime.onInstalled",  
15    "args": "null",  
16    "count": 1,  
17    "extensionId": "aamjlohoaogjmfddgdjojghjaamgmhdhj",  
18    "time": 1629376289820.012  
19  },  
20  {  
21    "activityId": "1",  
22    "activityType": "api_call",  
23    "apiCall": "runtime.setUninstallURL",  
24    "args": "null",  
25    "count": 1,  
26    "extensionId": "aamjlohoaogjmfddgdjojghjaamgmhdhj",  
27    "time": 1629376289814.344  
28  }  
29 ]
```

Listing 3.1: Example of a Chrome Extension Activities Log

Later on, this information will be used to confirm if an extension is acting suspicious, not only for what it might be doing but also if it is actually using all the permissions that it requests.

3.4 Risk Scoring System

Facing the quantity and quality of our dataset, we discussed the best way to classify an extension as malicious or benign. The maliciousness of an extension is and will always be partially subjective, as there may or may not be a direct malicious intent on a particular user or website, i.e., we cannot declare that an extension has malicious intentions if, by any means, the extension is being exploited by a third-party identity, we cannot simply blame the developer that may also be a victim.

With that in mind, we would have to have a sufficiently large dataset, both malicious and benign extensions, and through our approaches, try to predict whether a given extension is malicious by comparing its static and dynamic footprint to some previously analyzed extension or even a group of malicious extensions also already analyzed.

Looking into our dataset, we had to think about how to evaluate the collected data. After some discussion, we chose to develop a risk scoring system that assesses the activities registered and, according to their severity, assigns a value that can be interpreted as a risk value, e.g., an Extension A score greater than an Extension B score indicates that the Extension A represents a higher risk of maliciousness than Extension B.

Our risk scoring system assigns points to the presence or absence of a set of static and dynamic features of an extension, defined by each of our approaches. Each feature has a value corresponding to a degree of risk that we consider it to represent. The risk of each feature will always be a debatable value, since it will be related to the value of the other factors, as the risk comparison between them is what allows us to distinguish the different degrees of risk.

If an extension does not present any risk, its score will be 0, with the maximum possible limit being 1535 points – 1190 from static analysis and 345 from dynamic analysis – where an extension fills all the boxes of potential risk to the end-user, according to our solution. This system aims to penalize extensions for using powerful permissions, lack of credibility, and the several behaviors captured when installed and activated.

Although this system penalizes the behaviors and factors that represent a higher risk, it still has its pros and cons – an extension that requires several high-risk permissions and actually uses them without malicious intentions will always be a high-risk extension. When dealing with malicious extensions, sometimes perceiving the intention of the extension is something that cannot always be understood in a simple and enlightening way, as attackers also develop new ways to implement their maliciousness.

As discussed above, an extension that depends on and trusts a third-party for its operation could become malicious due to the intentions of that same third-party. For that reason, it makes a lot more sense to approach risk over intention, because if you cannot understand the intention of the extension, you have to perceive something measurable, and the risk might be.

On the other hand, a malicious extension that, for example, only does keylogging, will have a lower score value than an extension that only has access to our cookies, whether it is malicious or not. Here, the distinction between two different actions and their respective risks may be debatable, as it is not something we can substantiate for all cases based on published articles or official

documents, so it will always be a distinction based on personal opinion and difference in risk that a given activity presents when compared to another one with a lower risk. With this, the weights of each attribute/group of attributes are used only for demonstrative purposes, being an experimental system that can be modified based on further investigation.

It is also important to note that this risk scoring system is not only intended to assess the risk of an extension's behavior but also its security, as an extension that is not malicious but contains high-risk permissions can still represent a threat for the user if it is not properly protected against cyber attacks. Nevertheless, the user will have a better idea of the currently installed extensions and the risk they may represent for their privacy and the safety of their own machine.

3.4.1 Static Analysis

As for the risk scoring system itself, we first look at the data extracted from the Chrome Web Store page, mainly looking for possible missing information and if there is a later version than the one locally installed.

Starting with Chrome Web Store metadata, we applied 5 and 10 points to these factors:

- **Missing Developer Address** - 5 Points
- **Missing Privacy Policy** - 5 Points
- **Missing Developer Email Address** - 10 Points
- **Less than 1,000 Active Users** - 10 Points

If a banned extension does not have an available CRXCavator report, 30 points will be added, since we do not have any of the metadata properties mentioned above.

These points were decided based on what we believe to be the risk and the importance of having it against not having it. For example, although not a massive risk, an extension with a developer e-mail associated is believed to be safer than one that does not have, since the developer is, or at least is pretending to be, willing to be contacted for possible bugs, improvements or even collaborations. Although this example can be easily exploited in the future (if malicious developers learn of this missing e-mail property), a brief analysis of the e-mail domain is something that could be done to strengthen this case, e.g., an institutional e-mail is more valuable than an e-mail from a free e-mail service (e.g., Gmail).

Approaching Chrome Extension permissions, we chose to assign two permission categories, **Critical** and **High-Risk**, worth 40 and 25 points respectively. (Tables 3.1 and 3.2)

Critical
<i>cookies</i>
<i>debugger</i>
<i>declarativeWebRequest</i>
<i>webRequest</i>
<i><all_urls></i>
<i>*://*/*</i>
<i>*://*/</i>

Table 3.1: Critical Permissions

High-Risk			
<i>clipboardRead</i>	<i>experimental</i>	<i>pageCapture</i>	<i>vpnProvider</i>
<i>clipboardWrite</i>	<i>geolocation</i>	<i>privacy</i>	<i>webNavigation</i>
<i>contentSettings</i>	<i>history</i>	<i>processes</i>	<i>http://*/*</i>
<i>declarativeNetRequest</i>	<i>idlltest</i>	<i>proxy</i>	<i>https://*/*</i>
<i>desktopCapture</i>	<i>management</i>	<i>tabCapture</i>	<i>file:///*</i>
<i>displaySource</i>	<i>mdns</i>	<i>tabs</i>	<i>http://*/</i>
<i>dns</i>	<i>nativeMessaging</i>	<i>topSites</i>	<i>https://*/</i>

Table 3.2: High-Risk Permissions

In addition, 10 points will be added if the extension has not been updated during the last six months, as it may pose a threat to the user by not following the latest security methods available in recent Chrome versions. Having been updated in that period can be a good indicator since updated extensions have a higher chance of being protected against recent attacks or vulnerabilities.

If any static mention of a possible keylogging or screen capturing is found, 10 points will be added for each, even if they are not caught doing any of these activities. Also, 50 points will be added if any URL/IP is reported as malicious, although there is no guarantee that it has been accessed.

As for the CSP and *RetireJS* values that we extract from all the available CRXCavator reports, we decided to ignore the value of the CSP as the vast majority of extensions had a very similar value, not only keeping the risk scores proportional to each other but also favoring the extensions that do not have a report over those that do, and the results of this work cannot depend largely on the availability of those reports. On the other hand, we add half the value of *RetireJS* to the final

risk score, as we believe it results in a fairer and more balanced value compared to other properties present in this scoring system.

Last but not least, if an extension ID is found in the Google SafeBrowsing API list, that means that Google banned the extension. In that case, 100 points are added to its risk score.

3.4.2 Dynamic Analysis

Regarding the recorded activities, groups of 5, 10 and 15 points were established, according to the risk that we consider that each represents, as can be seen in the following tables.

API	API Call	Description
bookmarks	<i>getTree</i>	Examined browser bookmarks
management	<i>onInstalled</i> <i>onUninstalled</i>	Monitoring changes to the install state of extensions and apps
tabs	<i>get</i>	Got a list of browser's open tabs
tabs	<i>insertCSS</i>	Changed the way a website looks
storage	<i>Window.localStorage</i>	Read a website's local storage

Table 3.3: Low-Risk Activities – 5 points

In Table 3.3 are mentioned activities that, despite presenting a reduced risk, are still suspicious, especially if they are registered in extensions that, supposedly, do not need them for what they promise to deliver.

API	API Call	Description
cookies	<i>remove</i>	Deleted browser's cookies
history	<i>getVisits</i>	Read browser's history
nativeMessaging	<i>runtime.sendNativeMessage</i>	Sent a message to another application on local computer
tabs	<i>onActivated</i> <i>onCreated</i> <i>onMoved</i> <i>onRemoved</i> <i>onUpdated</i>	Monitoring changes to browser's tabs
tabs	<i>executeScript</i>	Executed a script
tabs	<i>move</i>	Moved open tabs
tabs	<i>query</i>	Searched through open tabs
tabs	<i>update</i>	Modifies the properties of a tab
topSites	<i>get</i>	Read browser's history

Table 3.4: Mid-Risk Activities – 10 points

As for mid-risk activities, Table 3.4 displays a set of activities, and their respective API, that already represent a considerable level of risk, requiring special care to verify if the extension has

sufficient credibility to call them safely. Finally, high-risk activities are listed in Table 3.5, which are activities that already put the user's privacy and security at risk. For this reason, the presence of each one of them is penalized with 15 points.

API	API Call	Description
chrome.windows	<i>create</i>	Created a new browser window
chrome.windows	<i>remove</i>	Closed a browser window
cookies	<i>get</i> <i>getAll</i> <i>getAllCookieStore</i>	Read browser's cookies
cookies	<i>onChanged</i>	Monitoring changes to browser's cookies
cookies	<i>set</i>	Changed browser's cookies
geolocation	<i>getCurrentPosition</i>	Read current physical location
history	<i>search</i>	Searched browser's history
management	<i>get</i> <i>getAll</i> <i>getPermissionsById</i> <i>getPermissionsByManifest</i>	Accessed list of installed extensions
management	<i>setEnabled</i>	Enabled or disabled another Chrome Extension or App
management	<i>uninstall</i>	Uninstalled another Chrome Extension or App
tabCapture	<i>capture</i> <i>getCapturedTabs</i>	Took a screenshot of a tab
tabs	<i>captureVisibleTab</i>	Took a screenshot of a tab
tabs	<i>remove</i>	Closed browser's tabs

Table 3.5: High-Risk Activities – 15 points

When it comes to web requests, malicious extensions can remove the security-related option which is typically used to avoid cross-site scripting attacks from HTTP Request/Response headers by using the callback functions of the *webRequest* API [44]. (Table 3.6) For example, a malicious extension can delete the security option from an HTTP header through a *onBeforeSendHeaders* callback, so that it could inject a remote malicious JavaScript that does not belong to that extension.

API	API Call	Description
webRequest	<i>onBeforeSendHeaders</i>	Intercept HTTP Request headers
webRequest	<i>onHeadersReceived</i>	Intercept HTTP Response headers
webRequest	<i>onBeforeRequest</i>	Intercept/Block all outgoing HTTP Requests

Table 3.6: Activities – Web Requests

Having this in mind, we also penalize with 15 points any occurrence of *onHeadersReceived* and *onBeforeSendHeaders*, since these are often used to modify response and request headers, as well as *onBeforeRequest*, a callback used to intercept/block all outgoing HTTP requests.

With this system, our goal is to have a measure to represent the risk that an extension can present based on its metadata, source code and on activities that perform on a real-world simulation, having the advantage of being possible to add new behaviors and tune the points system.

3.5 Implementation

In this section, we will cover in more detail what was used to implement our solution across all three stages.

For the first stage, a Python3 script is responsible for the entire automation and everything involved. Before the automation starts, every manifest file is opened to add the public key of the original CRX file – this guarantees us that Google Chrome will import the extension (already unpacked) with the same ID as the one used for static analysis.

Then, we decided to go with *Selenium WebDriver*⁷ – a browser automation library used for testing web-applications, mostly for tasks that require automating interaction with the browser – together with *ChromeDriver*⁸ – a separate executable that Selenium WebDriver uses to control Google Chrome – as we load all the extensions before scheduling the tasks to be performed. Also, extensions needed to be loaded unpacked since it was crashing while loading them packed.

Besides that, the Google Chrome window needs to be launched with a set of arguments to work as we intend to: start a maximized window; loading all the unpacked extensions in a single argument; allowing performance logging (off by default) to get timeline, network, and page events (we are only interested in network traffic) by adjusting its *desired_capabilities*. (Listing 3.2)

```

1 # Initialize the driver
2 options = Options()
3
4 # load modified Chrome Apps & Extensions Developer Tool extension
5 extensionsArgs = "load-extension=" + chrome_activities_tool
6
7 for extensionID in testDataset:
8     if(os.path.isdir(os.path.join(unpack_extensions_path, extensionID))):
9         extensionsArgs += ',' + os.path.join(unpack_extensions_path,
10             extensionID)
11 options.add_argument(extensionsArgs)
12 options.add_experimental_option("excludeSwitches", ["enable-automation"])
13 options.add_argument("--start-maximized")
14
15 capabilities = DesiredCapabilities.CHROME
16 capabilities["goog:loggingPrefs"] = {"performance": "ALL"}
17 driver = webdriver.Chrome(executable_path='./chromedriver', options=options,
18     desired_capabilities=capabilities)

```

Listing 3.2: Source Code of Selenium + ChromeDriver Setup

⁷Selenium WebDriver, <https://selenium-python.readthedocs.io/api.html>

⁸Chrome WebDriver, <https://chromedriver.chromium.org/>

Aiming to avoid any detection by any page or extension, we modified⁹ the original ChromeDriver to remove any Selenium fingerprint.

After launching the web browser window, we programmed a set of actions to occur and hard-coded time intervals between them to allow the browser to successfully load any website before moving to the next one.

Although almost everything is automated, the process of saving extensions interactions with the Chrome Extension API is carried out manually by accessing the previously discussed application and clicking on the option to download all the activity history. (Figure 3.3)

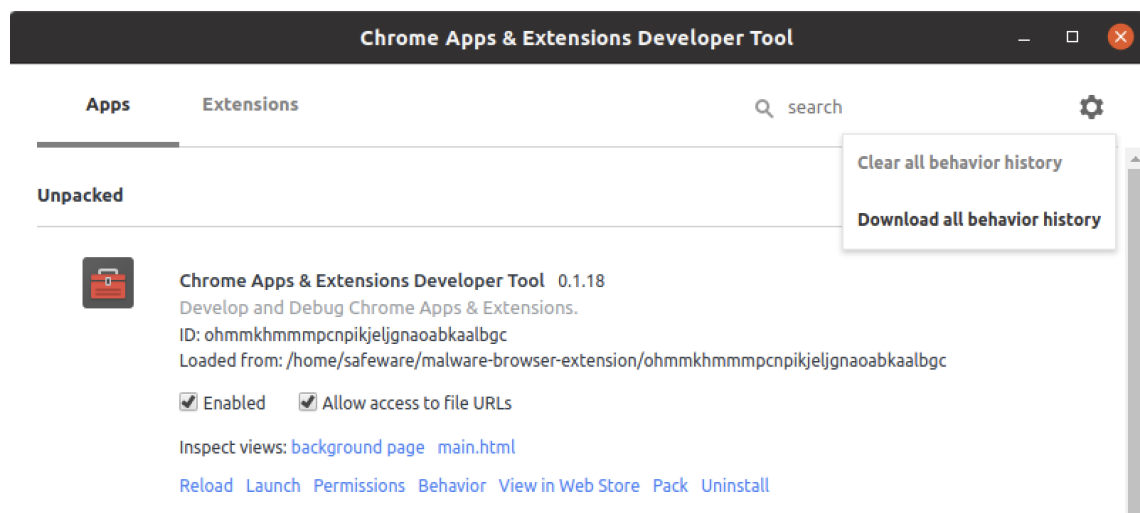


Figure 3.3: Modification to download Behaviour History

As for the second and third stages, we also use Python3 to make the necessary requests and to merge the data from both analyses, generating a JSON report in the end, based on the results obtained.

3.6 Summary

In this chapter, we explained our goals and our hybrid approach in detail, combining static and dynamic analysis, using a risk scoring system to obtain a numbered value that corresponds to the risk presented by an extension.

Our implementation consists of extracting multiple factors from different sources inside an extension, recording its behaviors on a simulated automation and outsourcing information from multiple APIs (CRXCavator API and Google SafeBrowsing API).

⁹StackOverflow – Can a website detect when you are using Selenium with chromedriver?, <https://stackoverflow.com/questions/33225947/can-a-website-detect-when-you-are-using-selenium-with-chromedriver/>

After all the properties have been extracted, an experimental risk scoring system is applied to those properties (or lack of them) according to a set of predefined risk values, resulting in a final risk score that aims to present the quantity of risk that an extension has to the end-user.

Once implemented, we proceed to test our solution and analyze the results obtained from experiments with our dataset.

Chapter 4

Experimental Results

For this work, we performed several tests for each of the three stages.

Before carrying out the tests, it was important to gather a dataset of extensions, both malicious and benign, and create an experimental setup that allows us to quickly test the changes made during the development of this work.

4.1 Dataset

Since we are testing and working only with the Google Chrome web browser, our dataset is only based on Google Chrome-based (CRX) extensions. When it comes to benign extensions, we used the Chrome Web Store¹ as a reference to extract the extensions to be used in testing. For the sake of this dissertation, we assume that any and all extensions available in the Chrome Web Store are benign until considered malicious by the responsible authorities and banned from it.

Through the official Chrome Web Store Sitemap², we were able to extract the list of all existing extensions available for installation. Using a Chrome Extensions Archive tool³, it was possible to get a list of 191,466 extensions, from which we chose to download a set of 220 random extensions. Due to regional restrictions, some extensions were not available for download, but this was not an issue given the extreme abundance of extensions.

To make the tests more realistic, we have also downloaded the 25 most popular extensions to understand what risk each of them represents for the user according to our approach. Looking into Table 4.1, it is possible to observe the power of Advertisement Blockers in the extension market, being the most wanted and installed by the world population.

¹Chrome Web Store, <https://chrome.google.com/webstore/>

²Chrome Web Store – Sitemap, <https://chrome.google.com/webstore/sitemap>

³Chrome Extensions Archive – GitHub Archive, <https://github.com/mdamien/chrome-extensions-archive>

Top 25 Most Popular Extensions	
AdBlock — best ad blocker	Skype
Honey	Cisco Webex Extension
Adblock Plus - free ad blocker	Chrome Remote Desktop
Tampermonkey	迅雷下载支持
AdGuard AdBlocker	Zoom
Google Translate	Windows 10 Accounts
Grammarly for Chrome	Adblock for Youtube™
LastPass: Free Password Manager	Just Black
Avast Online Security	Pinterest Save Button
uBlock Origin	Hola Free VPN Proxy Unblocker - Best VPN
Avast SafePrice Comparison, deals, coupons	TouchEn PC
Adobe Acrobat	Malwarebytes Browser Guard
Netflix Party is now Teleparty	

Table 4.1: Top 25 Most Popular Extensions on Chrome Web Store

As for malicious extensions, that is where we started noticing the lack of shared databases of previously banned extensions and their documentation, so initially, we could not collect a good dataset in terms of both quantity and quality. We were able to find some old high risk extensions: 3 malicious copies of the extension **Unblock-Youku**⁴⁵ (these extensions downloaded arbitrary JavaScript code from their server *5ihaitao.com* and executed it locally), **GumShoe**⁶⁷ (a discreet password logging) and about 44 extensions from a public spreadsheet⁸ that lists fake extensions that used to make people believe that it were authentic ones, but were actually extensions that, once installed without requiring any special permission, invariably prompted the user to provide personal and financial data [31].

These 44 extensions spoofed a range of consumer brands, including Adobe, Amazon, Facebook, HBO, Microsoft, Roku and Verizon. Scouring the manifests for each of these extensions revealed that many of the same developers were tied to multiple apps promoted by the same phony Google accounts. Some of them had only a handful of downloads, but most had hundreds or thousands: e.g., a fake Microsoft Teams extension attracted 16,200 downloads in the roughly two months it was available on Chrome Web Store and a counterfeit version of CapCut – a professional

⁴Malicious Chrome Extensions – GitHub Archive, <https://github.com/uku/malicious-chrome-extensions>

⁵Unblock-Youku, <https://chrome.google.com/webstore/detail/unblock-youku/pdnfnkhpgegcpcingjbfihlkjeighnddk>

⁶Gumshoe CRX 2.14 for Chrome, <https://www.crx4chrome.com/crx/8037/>

⁷GumShoe – GitHub Archive, <https://github.com/ajartille/gumshoe>

⁸FakeReviewFakeApp Spreadsheet, <https://docs.google.com/spreadsheets/d/1CcFc4mgGA9Ping8RZ1h6MH5XUUFXYdJlvO-xRMda1Q/>

video editing software suite – claimed nearly 24,000 downloads over a similar period [31]. Since these 44 extensions are very similar, mostly changing only the icon, name and author, we chose to randomly add 5 of them to our dataset, assuming that all adopt the same type of behavior, only the destination where the stolen information is sent can vary from one to another.

During our initial tests, we were using an allegedly benign extension – **Video Downloader Proficient**⁹ – that was removed from Chrome Web Store, so we decided to include it in our malicious extensions dataset, attempting to understand if it was removed due to malicious intentions (banned) and how much risk does it present when compared to other malicious extensions. (Table 4.2)

Malicious Extensions	
Video Downloader Proficient	GumShoe
Microsoft Authenticator	优酷土豆海外播放
Microsoft Teams	优酷海外版(1)
Remini - AI Photo Enhancer	优酷海外版(2)
SnapCalc - Math Problem Solver	Amino: Communities and Chats

Table 4.2: Dataset of Malicious Extensions

As some of our malicious extensions are modified copies of the original extensions, we decided to download the corresponding original extensions to later compare our solution results in both versions. (Table 4.3) In order to be able to mention and compare extensions with the same name, we will distinguish them using numbers after their name, being properly identified in the appendixes along with this document. Throughout the chapter, extensions are only referenced by their name.

Real Extensions	Fake Extensions
Microsoft Teams Screen sharing	Microsoft Teams
Remini - AI Photo Enhancer	Remini - AI Photo Enhancer
SnapCalc - Math Problem Solver	SnapCalc - Math Problem Solver
Unblock Youku	优酷土豆海外播放
	优酷海外版(1)
	优酷海外版(2)

Table 4.3: Original and Fake Extensions

⁹Video Downloader Proficient, <https://chrome.google.com/webstore/detail/video-downloader-proficient/pboidikkjjoedgpcndgmcalcpcpfdoia>

A quick look at these extensions revealed that two of them – **Remini - AI Photo Enhancer**¹⁰ and **SnapCalc - Math Problem Solver**¹¹ – are precisely the same as their versions that were previously banned from Chrome Web Store. Both are still available on Chrome Web Store under the same phony Google Account – *oliviasmith6582@gmail.com* – and currently have 0 users.

The lack of variety and quantity of malicious extensions influenced the methodology and the initial idea of this work, as it would take more time to investigate extensions with a high-risk index, both active extensions in the Chrome Web Store and banned extensions.

In a later stage of the work, we were able to cross the IDs provided by the list of banned extensions from Google SafeBrowsing API with *Chrome-Stats*¹² – an interactive analytics platform that provides insights into what is going on within Chrome Web Store – and download the extensions that were available in their database. Out of 33,508 extensions that we extracted from the Google SafeBrowsing API list of banned extensions – we were able to find 3621 extensions in *Chrome-Stats* database, and only 2633 were effectively available for download.

Again, due to storage space, time and performance reasons, we randomly downloaded and added only 240 malicious extensions to our dataset. Together with the ten malicious extensions already collected, we can match our dataset length of benign extensions. Going into testing, we are using 500 extensions, being 250 of them benign and 250 malicious. (see Appendix A)

Besides being necessary to classify extensions, it is also important to understand the maliciousness behind each of them to predict this behavior when analyzing future extensions, something that would require more time.

Regarding the dataset of websites to visit in our dynamic approach, we decided to use a public list of the 100 most visited websites in the United States of America [27], as the list referring to world traffic included several subdomains of the same domain (e.g., *Wikipedia*). (see Appendix C)

4.2 Experimental Setup

Since we are dealing with malicious extensions, it is essential to have a safe and controlled working environment as these are applications that can cause damage when installed and ran on any machine. For that, we used a MacBook Pro 2016 with an Intel Core i7 6700HQ @ 2.6GHz with 16GB of RAM running a Virtual Machine with Ubuntu¹³ 20.04 on VMware Fusion¹⁴ 11.5.7. Furthermore, to make testing on multiple machines easier, a virtual environment in Python (version 3.8.10) was also created with the requirements included in the workspace.

Complementing that, a setup shell script was also written to ensure that all the files and folders are created and organized before the testing phase, as well as extracting each extension ID and the public key to be later added to its manifest, ensuring that both approaches are using the same ID

¹⁰Remini - AI Photo Enhancer, <https://chrome.google.com/webstore/detail/remini-ai-photo-enhancer/hdlhdkchhhieipniibccdefnoobiaapf>

¹¹SnapCalc - Math Problem Solver, <https://chrome.google.com/webstore/detail/snapcalc-math-problem-sol/mcnjlbnncmfaephaoaknienbohggbnq>

¹²Chrome-Stats, <https://chrome-stats.com/>

¹³Ubuntu 20.04 LTS, <https://releases.ubuntu.com/20.04/>

¹⁴VMware Fusion, <https://docs.vmware.com/en/VMware-Fusion/index.html>

for the same extension. This process is critical since all CRX files are decompressed to carry out the static analysis and later be imported to Chrome. When importing, we need to add the public key inside the manifest to maintain the original ID of the CRX, so we extract it before unzipping each one of them.

Alongside some shell commands, we also used *crx2zip*¹⁵ to extract the zip archive payload from a Chrome Extension, and *crx3-utils*¹⁶ to extract the original ID and the public key from a CRX3 file. (see Appendix B)

Once all of our malicious extensions – except **Video Downloader Proficient** since it was removed while we were testing as a benign extension – were downloaded in *.zip* or CRX2 – an older version of CRX3 that was deprecated in Chrome 76 and support completely removed in Chrome 78¹⁷ – we had to build a new CRX3 file for each malicious extension, automatically creating a new extension ID.

All the CRX2 files had to be renamed as ZIP files to obtain the zip archive inside it, and once we had all in the ZIP format file, we unzipped and used Google Chrome to pack all the extensions one by one before running our setup shell script. (see Appendix B) It was necessary to map the new IDs to the originals, since the original ID, while not a behavioral factor, is important as it may already belong to the Google SafeBrowsing API list and can immediately reveal if an extension has already been reported as malicious and banned from the official store.

4.3 Testing

After installing all the requirements and running the auxiliary shell script without any incident, we started testing our work in a safe and isolated environment. Several tests were carried out during the development of the solution, not only to look for errors in the code but also to tune our risk scoring system to achieve a score that we consider fair.

Going into testing our hybrid approach, we chose to run three experiments – one dedicated to benign extensions, other to malicious extensions and finally, a mix of both benign and malicious extensions.

Due to the high number of extensions and the limitations of our machine, although we tried it at an early stage, it was not possible to install and run all extensions simultaneously in a single simulation, as the machine becomes too slow and crashes even before the simulation starts. To solve this problem, the simulation is done and repeated in groups of 50 extensions to be executed without any delay. Here, it is relevant to mention the importance of simulating browsing with several extensions installed as we also want to record the behavior that certain extensions might have with/against others.

Additionally, we also had to deal with the tabs created by each extension and ensure that they were all closed before automated browsing began.

¹⁵CRX2ZIP – GitHub Archive, <https://github.com/mikhirev/crx2zip>

¹⁶CRX3-UTILS – GitHub Archive, <https://github.com/gromnitsky/crx3-utils>

¹⁷CRX Format Checker, <https://crx-checker.appspot.com/>

4.3.1 Benign Extensions

First, we tested with our 250 benign extensions, of which 6 of them were not able to be installed due to manifest anomalies (missing manifest version, missing icons, etc.) and corrupted JavaScript files. Since these extensions were directly taken from the Chrome Web Store and cannot be installed, we considered a certain risk attributed to the author's negligence in publishing a version of the extension that cannot even be installed. (Table 4.4)

Name	Rating	Users
Font Changer Online - #1 IG Font Changer	0/5	0
PaidBefore	0/5	0
Diceware passwords generator	1/5	12
Night Shift GEMS	5/5	9
Ascent Internet	0/5	4
Remini - AI Photo Enhancer	0/5	0

Table 4.4: Benign Extensions not installed during Benign Extensions Automation

Analyzing these extensions by just looking into the number of users, rating and amount of reviews, it is possible to see that they are poor extensions and that, given their popularity and the credibility of their developer, these extensions represent a very high risk, even if we cannot test them dynamically. Therefore, our automation consists of visiting 100 different websites in 5 open tabs, with a maximum of 50 installed extensions and a set of hard-coded intervals, such as the browsing time of a single tab, opening a new tab and time between scroll up and down.

Following each automation, which averaged 36m54s, there is a manual interaction with 4 popular hard-coded websites (*Facebook*¹⁸, *YouTube*¹⁹, *Google*²⁰ and *Amazon*²¹), in which a more human interaction is performed, such as opening a new logging in/out, searching for results and a more responsive scroll through out the entire web-page.

During one of five automations performed, it was noticeable that one extension – **auto-pin-tab** – was pinning specific tabs. Still, it ended up not affecting the procedure of our experiment.

On the third manual interaction, an extension was blocking us from logging in on Facebook and watching a video on Youtube. After some brief investigation, we discovered that **NoCors** was affecting our navigation, so it was immediately flagged as a high-risk extension.

After saving all the activities logs in the respective directory, we proceed to the second stage, which is integrated at the beginning of the third and last stage. At this stage, SafeWare analyzes all 250 extensions without any limitation as no extension is being executed but only locally examined.

¹⁸Facebook, <https://www.facebook.com/>

¹⁹YouTube, <https://www.youtube.com/>

²⁰Google, <https://www.google.com/>

²¹Amazon, <https://www.amazon.com/>

As mentioned above, the third stage ends by creating a report for each extension, in which all analyzed and evaluated properties are represented, as well as their risk value. (Listing 4.1)

```

1  "ihcjiegdanjaechkgeegckofjjedodee": {
2      "apiCalls": {
3          "storage.get": true ,
4          "storage.set": true ,
5          "tabs.get": true ,
6          "tabs.query": true ,
7          "webRequestInternal.eventHandled": true
8      },
9      "apiEvents": {
10         "tabs.onRemoved": true ,
11         "tabs.onUpdated": true ,
12     },
13     "banned": false ,
14     "crxCavator": {
15         "csp": 377 ,
16         "retireJS": 40
17     },
18     "crxReport": true ,
19     "cws": {
20         "email": "support@malwarebytes.com",
21         "name": "Malwarebytes Browser Guard",
22         "privacyPolicy": "https://www.malwarebytes.com/privacy",
23         "updated": "11-08-2021",
24         "users": 7000000,
25         "version": "2.3.6"
26     },
27     "domAccess": {
28         "blinkAddEventListener": true
29     },
30     "keylogging": false ,
31     "malwareURL": false ,
32     "name": "Malwarebytes Browser Guard",
33     "permissions": [
34         "alarms",
35         "downloads",
36         "storage",
37         "tabs",
38         "webRequest",
39         "webRequestBlocking",
40         "unlimitedStorage",
41         "<all_urls>"
42     ],
43     "riskScore": 150 ,
44     "screenshot": false ,
45     "updated": true ,
46     "version": "2.3.6",
47     "webRequests": {
48         "webRequest.onBeforeRequest": true
49     }
50 }

```

Listing 4.1: Example of a Chrome Extension Report

After getting the report for each of the 250 benign extensions, it is interesting to filter the results by the property to get a more clear idea of the most and least common behaviors of extensions, out of the ones that we are capturing, that are currently available in the Chrome Web Store.

First, we will have a look into details extracted from our static analysis, as shown in Table 4.5:

Properties	Number of Extensions
CRXCavator Report	221
Developer E-mail	156
Developer Address	28
Privacy Policy	85
Users (≥ 1000)	69
Keylogger	8
Tab Capturing	10
Updated (< 6 months)	47
Malware URL/IP	0

Table 4.5: 250 Benign Extensions – Static Analysis Results

As expected, no extensions were found on the Google SafeBrowsing API list. Only 221 extensions revealed a CRXCavator report, as only 47 extensions were updated in the last six months, representing only a quarter of our dataset for this experiment.

Although no extension has been classified as banned, at the time of this test, four extensions were not available any more on Chrome Web Store – **Waterfall Popular HD New Tabs Theme**, **Easy Color Grabber**, **Lil Mosey Search** and **Well actually I’m an idiot**. As all four extensions have a CRXCavator report, we are using their *webstore* metadata contained inside it.

A total of 5540 IP addresses and 47047 URLs were found, yet, none were flagged as malicious by Google SafeBrowsing, the API responsible for performing the individual lookup of each feature. While a third of the 250 extensions have a privacy policy, only 69 are used by 1000 or more users.

Stepping into the dynamic analysis, in Table 4.6, we count the number of extensions that called a specific API call, ordered by their risk level, according to the levels previously defined.

Activities (5 points)	Number of Extensions
<i>tabs.get</i>	18
<i>tabs.insertCSS</i>	4
<i>Window.localStorage</i>	4
<i>management.onInstalled</i> <i>management.onUninstalled</i>	3
<i>bookmarks.getTree</i>	1
Activities (10 points)	Number of Extensions
<i>tabs.onActivated</i> <i>tabs.onCreated</i> <i>tabs.onMoved</i> <i>tabs.onRemoved</i> <i>tabs.onUpdated</i>	49
<i>tabs.query</i>	35
<i>tabs.create</i>	22
<i>tabs.executeScript</i>	12
<i>tabs.update</i>	2
<i>runtime.sendNativeMessage</i>	1
Activities (15 points)	Number of Extensions
<i>cookies.get</i> <i>cookies.getAll</i> <i>cookies.getAllCookieStore</i>	6
<i>cookies.onChanged</i>	3
<i>cookies.set</i>	3
<i>management.get</i> <i>management.getAll</i> <i>management.getPermissionsById</i> <i>management.getPermissionsByManifest</i>	3
<i>history.search</i>	1
<i>tabs.captureVisibleTab</i>	1
<i>tabs.remove</i>	1

Table 4.6: 250 Benign Extensions – Dynamic Analysis Results

Curiously, eight extensions were flagged as keylogging and even though ten extensions were flagged as capturing visible tabs, only one was actually caught doing it – **History Search**. By meticulously investigating the source code of this extension, we were able to find that it was

capturing the visible tab every time the user changed tabs when the window resolution was equal or higher than 960x600. The script that was responsible for this activity was also collecting our location via *geoPlugin*²² – a geolocation Web service API, showing how easy it is to get our location even without having the *geolocation* permission referenced in its manifest.

Starting by the most high-risk activities, three extensions were caught accessing the list of installed extensions, one extension removing tabs – **KeepTabs** – and also another extension searching our browser’s history – **Suspicious Site Reporter** – with the latter being an official Google Extension that helps to protect web users by reporting suspicious sites in Chrome to Google SafeBrowsing.

Furthermore, a total of 6 extensions were reading our browser’s cookies, while three were changing them and three were monitoring any changes to them as well.

Regarding the *tabs* API, 49 extensions were monitoring changes to browser’s tabs, 35 were searching through them and only 18 were getting a list of open tabs.

Last but not least, a single extension – **AssetView** – was caught sending a message to another native application on our computer.

Finally, we can also have a look at *webRequest* API Callbacks, attempting to understand how many of these extensions are intercepting HTTP Requests and Responses.

Callbacks	Number of Extensions
<i>webRequest.onBeforeSendHeaders</i>	5
<i>webRequest.onHeadersReceived</i>	4
<i>webRequest.onBeforeRequest</i>	7

Table 4.7: 250 Benign Extensions – *webRequest* API Results

Analyzing Table 4.7 results individually, we conclude that the majority of them are ad-blocking and anti-virus extensions, all of them containing *webRequest* and *webRequestBlocking* permissions.

4.3.2 Malicious Extensions

Repeating the same procedure for the 250 malicious extensions, all were successfully imported and installed on our Google Chrome testing browser.

In this test, we faced an extension – **IDTrust Signer** – that, once installed and running, downloads a *.msi* file – an installation package file format used by Microsoft Windows – that we assume is highly likely to contain malware.

Besides that, another extension – **Acortador URL** – was hijacking some websites by changing their URL right before their execution, redirecting to the desired site by first going to another one, unnoticed. This malware is known as the *MoviesDimension.com* browser hijacker²³ and it often

²²IP Geolocation and Currency Converter APIs by Geoplugin, <https://www.geoplugin.com/>

²³How to get rid of Moviesdimension.com browser hijacker?, <https://mypc.guru/remove-moviesdimension-com-hijacker/>

switches the search engine settings in the victim's browser to one which it is paid for, as well as makes some undesirable modifications that can provoke personal information leaks. (Listing 4.2)

```
1 https://moviesdimension.com/loading/?p=9192&v=400&url=https%3A%2F%2Fwww.amazon.com%2Fref%3Dnav_logo&guid=
```

Listing 4.2: *MoviesDimension.com* hijacking *Amazon.com*

Interestingly, one extension – **PBlock+** – was even blocking access to the extensions installed. If we tried to access via Chrome Settings or even typing *chrome://extensions*, it would not allow us to uninstall any extension. Whenever we tried to access the installed extensions, we were redirected to Google Chrome's installed applications page, without any access to any extension. In addition, we also noticed that this extension was redirecting all Google searches to its search engine – *search1.me* – being another very suspicious behavior and undoubtedly high risk, as we do not know this search engine or where our searches are going.

Once the final stage was finished, again, a report for each one of the 250 malicious extensions was generated. This time, we will focus more on the most common behaviors and, if a CRXCavator report exists, trying to understand their Chrome Web Store properties at the time of their last available version.

Below, Table 4.8 shows the principal details extracted from our static analysis:

Properties	Number of Extensions
Google SafeBrowsing	246
CRXCavator Report	243
Developer E-mail	138
Developer Address	19
Privacy Policy	103
Users (≥ 1000)	140
Keylogger	9
Tab Capturing	7
Updated (< 6 months)	19
Malware URL/IP	1

Table 4.8: 250 Malicious Extensions – Static Analysis Results

98.4% of them were effectively found on the Google SafeBrowsing's list and surprisingly, 243 extensions revealed to have a report on CRXCavator, as 19 of them were updated in the last six months (when last available on Chrome Web Store).

A total of 4900 IP addresses and 36019 URLs were found, and only one was flagged as malicious by Google SafeBrowsing – *http://e-days.cc/music/feature/tunein/200911/29006.php?page=2*

– coming from **Better Pagerize**. Here, we believed that we would get different results using multiple APIs, but as previously discussed, this was not possible given the required and available resources.

Focusing only on the ones that happen to have a CRXCavator Report, it is interesting that about 140 extensions had 1000 or more users, showing the power these malicious extensions had in tricking users into installing them.

Moving into the dynamic analysis, we applied the same method as in the previous experience.

Activities (5 points)	Number of Extensions
<i>tabs.get</i>	12
<i>Window.localStorage</i>	12
<i>tabs.insertCSS</i>	4
Activities (10 points)	Number of Extensions
<i>tabs.create</i>	49
<i>onActivated</i> <i>tabs.onCreated</i> <i>tabs.onMoved</i> <i>tabs.onRemoved</i> <i>tabs.onUpdated</i>	44
<i>tabs.query</i>	21
<i>tabs.executeScript</i>	11
<i>cookies.remove</i>	1
<i>tabs.update</i>	1
<i>topSites.get</i>	1
Activities (15 points)	Number of Extensions
<i>cookies.get</i> <i>cookies.getAll</i> <i>cookies.getAllCookieStore</i>	9
<i>windows.create</i>	6
<i>cookies.set</i>	2
<i>history.search</i>	2
<i>cookies.onChanged</i>	1
<i>Geolocation.getCurrentPosition</i>	1
<i>tabs.captureVisibleTab</i>	1
<i>tabs.remove</i>	1
<i>windows.remove</i>	1

Table 4.9: 250 Malicious Extensions – Dynamic Analysis Results

In Table 4.9, the number of extensions reading the browser’s cookies is noticeable, as well as reading websites’ local storage and creating new windows.

Despite seven extensions being flagged as screen captors, again, only one was caught – **Eye-dropper Pro** – while nine were marked as listening every *keydown* press on the keyboard. This time, one extension was reading the browser’s history, as two extensions were searching through it and another extension – **Pixlr Today** – was even reading our current physical location using the *geolocation* API.

Curiously, not a single extension was recorded accessing the list of installed extensions, let alone disabling others. Still, we can also highlight an extension that was removing tabs, **PBlock+**, and a couple of them changing cookies – **Roblox New Tab** and **Change Cursor**.

Callbacks	Number of Extensions
<i>webRequest.onBeforeSendHeaders</i>	6
<i>webRequest.onHeadersReceived</i>	6
<i>webRequest.onBeforeRequest</i>	11

Table 4.10: 250 Malicious Extensions – *webRequest* API Results

Concerning the *webRequest* API, this only-malicious dataset has about 11 extensions intercepting outgoing HTTP requests and six extensions intercepting both HTTP request and response headers. Although not enough to prove, the number of occurrences of these callbacks suggests that malicious extensions tend to use them more often than benign ones.

4.3.3 Benign and Malicious Extensions

In an attempt to further explore the behavior of malicious extensions, we also chose to test with both benign and malicious extensions by testing using the entire dataset available, 250 extensions from each group.

Like previous tests, the simulation was done and repeated in groups of 50 extensions due to performance reasons. We tried to mix both categories inside those groups, attempting to capture new behaviors, since static analysis produces the same results.

In this test, the main focus is to observe the existence of new behaviors by each extension and not to verify if the previous behaviors were also captured or their risk scores. Since this test was not done right after the previous two, we will ignore changes in static factors, more precisely Chrome Web Store metadata as some extensions may have been updated since then.

Comparing the results of this experiment with the union of the two previous reports, we captured 19 new activities over ten extensions, as it is possible to verify in the table below. (Table 4.11)

Name	Activities
American Gods Themes & New Tab	<i>bookmarks.getTree</i> <i>topSites.get</i>
My Email Tab	<i>tabs.create</i> <i>tabs.update</i>
ReachEdge Website Compatibility Checker	<i>cookies.get</i> <i>identity.getProfileUserInfo</i>
Hola Free VPN Proxy Unblocker - Best VPN	<i>cookies.get</i> <i>cookies.set</i> <i>tabs.create</i> <i>webRequest.onBeforeSendHeaders</i>
The 100 HD Wallpapers New Tab	<i>bookmarks.getTree</i> <i>topSites.get</i> <i>Geolocation.getCurrentPosition</i>
Appstation Video Downloader Professional	<i>tabs.query</i>
Lion & Tiger New Tab & Wallpapers Collection	<i>tabs.create</i>
AliHelper - shopping assistant	<i>cookies.getAll</i> <i>tabs.create</i>
Rung Rinh: Free Cash Back Shopping Assistant	<i>Window.localStorage</i>
Louis Vuitton Bags HD Wallpapers New Tab	<i>topSites.get</i>

Table 4.11: Extensions – New Activities Captured

It is somewhat uncertain why these activities were not performed before, but nonetheless, we consider that most of them are medium-high risk and capturing these behaviors is important in the final risk score for each of these extensions.

4.4 Risk Scoring Evaluation

Having the report of each of the three experiments performed, we need to understand if our hybrid approach in conjunction with the risk scoring system is a viable solution and whether it can help classify malicious extensions.

Starting with the first experiment results, the highest risk score belongs to **smarterPod: Simple and Smart Podcast Player**, scoring a total of 1290 points. Counting more than 4000 users, it has only two high-risk permissions, not performing any low, mid or high-risk activity. Despite not being updated, its high score is mostly penalized by its *RetireJS* score, revealing a high amount of vulnerabilities on *bootstrap*, *jquery* and *angularjs* libraries.

Outscored by just 140 points, **AdBlock - best ad blocker** asks for two critical and two high-risk permissions while monitoring tabs and searching through them. Also, it was flagged by possible screen tab capturing and intercepting/blocking outgoing HTTP requests. Its significantly higher risk score is also heavily penalized by its *RetireJS* score, having vulnerabilities on its *jquery* library. According to Chrome-Stats²⁴, the latter is currently the #1 popular extension on Chrome Web Store.

Name	Risk Score	CRXCavator Report
smarterPod: Simple and Smart Podcast Player	1290	Y
AdBlock - best ad blocker	1150	Y
TopRecipeFinder	335	Y
Marketo Sales Connect	300	Y
CatchYourTicket	300	Y
Tripadvisor Browser Button	285	Y
Adobe Acrobat	275	Y
Unblock Youku	270	Y
Gladdress.com	265	Y
Tampermonkey	265	Y

Table 4.12: 250 Benign Extensions – Top 10 Highest Scores

Looking into the Table 4.12 and analyzing each extension report, we can see that their *RetireJS* score has a significant impact, aside from their high-risk permissions and activities, where searching through the open tabs reveal to be the most captured in this Top, as well as monitoring them, which helps to explain a higher risk score.

Among the 250 benign extensions, the average score was 93,8 points, and if we do not consider the Top 2, heavily influenced by *RetireJS*, the average drops to 84,7 points.

²⁴Chrome-Stats, <https://chrome-stats.com/>

If we take a look at the most requested permissions, in Table 4.13, we can understand the weight of their risk in that average risk score against the captured suspicious behaviors.

Permissions (Critical)	Number of Extensions
<i>webRequest</i>	38
<i><all_urls></i>	23
<i>cookies</i>	17
<i>*://*/*</i>	8
<i>debugger</i>	1
Permissions (High-Risk)	Number of Extensions
<i>tabs</i>	77
<i>webNavigation</i>	19
<i>http://*/*</i>	18
<i>https://*/*</i>	18
<i>clipboardWrite</i>	8
<i>management</i>	8
<i>nativeMessaging</i>	7
<i>topSites</i>	6
<i>desktopCapture</i>	5
<i>history</i>	5
<i>proxy</i>	5
<i>http://*/</i>	3
<i>https://*/</i>	3
<i>privacy</i>	3
<i>clipboardRead</i>	2
<i>contentSettings</i>	1
<i>declarativeNetRequest</i>	1
<i>pageCapture</i>	1

Table 4.13: 250 Benign Extensions – Permissions

When analyzing the permissions, two extensions were found repeating the same permission in their manifest, therefore being counted as only one. Out of 38 extensions with *webRequest* permission, only 13 extensions are using its API, which raises some questions, not only on the intention of the captured API calls but also on the need to demand from the user a permission that is not being used, at least in our simulation. The same is verified for more permissions, as, for example, only nine extensions out of 17 are effectively using the *cookies* API, which by

itself is already worrying since it is where our sessions are stored. On the other hand, it is worth mentioning the number of extensions that require the *tabs* permission, 78, being by far the most required and used API.

Name	Risk Score	CRXCavator Report
Aurora - Multi Color Tabs	0	N
MP3Juice - Search MP3 Downloads	5	Y
Hikaru and GothamChess Tell You to MOVE	10	Y
AdGuard Extra	10	Y
Just Black	15	N
Tăng Like Instagram, Tăng Follow Instagram	15	Y
Classroom Tools	15	Y
cwcom Sites	15	Y
LeetPlug	15	Y
Far Cry 6 Animated New Tab	15	Y

Table 4.14: 250 Benign Extensions – Top 10 Lowest Scores

Reversing the order of risk scores, in Table 4.14, we were able to find an extension that achieved zero points – **Aurora - Multi Color Tabs** – filling all the Chrome Web Store acceptable requirements, not having a single permission and being updated. Although we aim for a risk score proportional to the risk level, an extension that reveals a low-risk score and has no permissions is not entirely risk-free. In this specific case, this extension is just a theme for Google Chrome, hence the lack of permissions or interaction with the Chrome Extension API.

As for the other extensions in this Top 10, they all have none to harmless permissions. A developer address and a Privacy Policy missing are the reasons for such a low-risk score, except for the latter, which is also outdated, hence the small increment. Other activities were captured (e.g., *storage* API), but we do not consider them dangerous.

Before moving to the next experience, it is worth mentioning the risk score of extensions with unique activities:

- **EnvTab** – *bookmarks.getTree* – 140
- **AssetView** – *runtime.sendNativeMessage* – 250
- **Suspicious Site Reporter** – *history.search* – 115
- **History Search** – *tabs.captureVisibleTab* – 200
- **KeepTabs** – *tabs.remove* – 100

Inspecting the report of the second test, with 250 malicious extensions, it is already noticeable the higher risk scores, as visible in Table 4.15:

Name	Risk Score	CRXCavator Report
Video Downloader PLUS	460	Y
Tests-moi - Offres shopping	450	Y
AdsFuzion	415	Y
shopee-fun	370	Y
Savings Alerts	365	Y
Nano Adblocker	360	Y
uBlocker Pro - Free AdBlock for Chrome Plus	360	Y
ProxyFlow - Free VPN Proxy	360	Y
NativeDoc	350	Y
Terrific Shopper-B	340	Y

Table 4.15: 250 Malicious Extensions – Top 10 Highest Scores

Scoring about 460 points, **Video Downloader PLUS** tops the score sheet by accessing, searching and monitoring browser's tabs, being outdated, having listed in its manifest about three critical and five high-risk permissions, intercepting HTTP Request headers and, of course, is already banned from Chrome Web Store.

From this Top 10, we highlight **Nano Adblocker** for being the only extension across all 500 extensions to use all the three callbacks from the *webRequest* API, scoring 45 points just on that alone.

Comparing Tables 4.13 and 4.16, it is remarkable the difference in some permissions, especially those involving the URL patterns, both critical and high-risk ones. Still, 77 extensions do not have any permission, leading to an average risk score of 192,84 points across all the malicious extensions used to perform this test.

Permissions (Critical)	Number of Extensions
<i>webRequest</i>	45
<i><all_urls></i>	41
<i>cookies</i>	25
<i>*://*/*</i>	14
<i>*://*/</i>	1
Permissions (High-Risk)	Number of Extensions
<i>tabs</i>	115
<i>http://*/*</i>	32
<i>https://*/*</i>	26
<i>webNavigation</i>	16
<i>proxy</i>	10
<i>clipboardWrite</i>	7
<i>history</i>	5
<i>management</i>	5
<i>topSites</i>	5
<i>http://*/</i>	4
<i>clipboardRead</i>	3
<i>https://*/</i>	3
<i>contentSettings</i>	2
<i>geolocation</i>	2
<i>nativeMessaging</i>	2
<i>privacy</i>	2
<i>desktopCapture</i>	1
<i>pageCapture</i>	1

Table 4.16: 250 Malicious Extensions – Permissions

Since we had Google SafeBrowsing API matching 246 extension IDs, that same number of extensions automatically has a minimum of 100 points, regardless of other properties. As shown in Table 4.17, this Top 10 is occupied by extensions that have been banned by Google and do not have some properties in their metadata, being unknown and uncertain why they were banned.

Name	Risk Score	CRXCavator Report
Trust Wallet: Sim CX	105	Y
T-Rex Dino Chrome	110	Y
Mobdro	110	Y
Cowin slot Booking BOT	110	Y
YouTube SEO Checklist (SEO Ranking Tool)	110	Y
EduPageTestHack	110	Y
Imo For Pc Download – Windows XP/7/8/Mac	115	Y
ResultsExpander	115	Y
Color Dino Game	120	Y
FiLMiC Pro - Video Camera	120	Y

Table 4.17: 250 Malicious Extensions – Top 10 Lowest Scores

Analyzing **Video Downloader Proficient** – an extension that went unavailable during initial testing – was not banned from Chrome Web Store and its score risk is supported by *webRequest* and *tabs* API calls, as well as two high-risk and one critical permission, scoring 155 points in our test. While we are not classifying an extension as malicious, it is impossible to look at its report and conclude that it does not have anything suspicious.

Also, it is worth highlighting the risk scores of extensions that were captured with unique behaviors:

- **Savings Alerts** – *cookies.remove* – 365
- **Thop TV PC** – *tabs.update* – 130
- **Terrific Shopper-B** – *topSites.get* – 340
- **Roblox New Tab** – *cookies.onChanged* – 210
- **Pixlr Today** – *Geolocation.getCurrentPosition* – 295
- **Eyedropper Pro** – *tabs.captureVisibleTab* – 200
- **PBlock+** – *tabs.remove* – 215
- **PBlock+** – *windows.remove* – 215

4.4.1 Fake vs Original Extensions

Since we have some extensions that pretend to be others, we can now compare their reports and check if their differences are noticeable through our analysis. Sharing the same identity, some fake

extensions were slightly modified with malicious intentions, while others are just stealing a name, an icon and its description.

Starting with the most popular, **Microsoft Teams Screen sharing**, did not register any activity during our simulation, scoring 40 points for being outdated, not having a developer address and having a high-risk permission (*desktopCapture*). Despite its 300,000 users, this extension was last updated in December 2018 and it is harmless, at least without interacting with it. As for the fake extension, entitled **Microsoft Teams**, scored a total of 120 points according to our solution, being penalized for already being banned from CWS, not being updated, missing a developer address and a Privacy Policy. Counting with 10,000 users at the time, it does not have a single permission and when inspecting its CRX package, we can only find a simple *install.html* file containing a *iframe* embedding a shortened URL that was redirecting to a sketchy website. Unfortunately, all these URLs were down and we were not able to explore them to understand exactly what their objective was. (Listing 4.3)

```
1 http://przekierowanie1-chrome.augustow.pl/?123-Microsoft58525836954
2 http://przekierowanie1-chrome.augustow.pl/?123-Remini5246987312
3 http://przekierowanie1-chrome.augustow.pl/?123-SnapCalc4569875
```

Listing 4.3: Shortened URLs inside `<iframe>` Tag

Moving to the following two extensions, **Remini - AI Photo Enhancer** and **SnapCalc - Math Problem Solver**, both share the same structure, similar to **Microsoft Teams**, only with different shortened URLs redirecting to suspicious websites. Although these extensions are still available on Chrome Web Store, which is dangerous, currently both do not have any user and one of them could not even be installed, as mentioned in Table 4.4.

Finally, we have three modified versions of **Unblock Youku**, an available extension with over a million users and scoring 180 points, mainly for its critical and high-risk permissions, alongside not being updated in the last six months and creating tabs. As already mentioned, these three versions are modified to download JavaScript code from their server *5ihaitao.com* and execute it locally, scoring 190, 140 and 140 points, respectively. None of the three extensions were flagged as previously banned from Chrome Web Store, with the last two being precisely the same, having fewer two high-risk permissions than the first one.

A *log.js* is downloaded and programmed to manipulate browser's cookies, stealing online identities and leaking private information.²⁵ In our tests, this script was not downloaded due to the website being unreachable. However, our static analysis would also not parse the malicious URL due to a string reversal trick, which is represented in Listing 4.4.

This small but effective trick shows how complex and challenging it is to cover all possible cases, as more developers find new ways to hide their true intentions from users and malware detection and prevention systems.

²⁵Malicious Chrome Extensions – GitHub Archive, <https://github.com/uku/malicious-chrome-extensions>

```
1 a.open("GET", "sj.gol/gol/moc.oatiah5.emorhc//:ptth".split("").reverse().join(""),
    !0);
```

Listing 4.4: String Reversal Trick

Although we cannot draw any concrete conclusions from this small test of original and modified extensions, it remains one of the principal methods of distributing malware as the Chrome Web Store takes a lot of time manually reviewing extensions updates while approving harmful extensions from multiple different accounts instantly [41].

4.4.2 Conclusions

After extracting the main results from both reports, we need to check whether our solution is efficient and viable to use at a larger scale against a bigger and better dataset of real and available extensions. For this, we calculated the weights of each approach in each total sum of risk scores from both tests. (Table 4.18)

	250 Benign	250 Malicious
Static Analysis	21580	46085
Dynamic Analysis	1870	2125
Total	23450	48210

Table 4.18: Hybrid Approach – Sum of Risk Scores

Comparing the total sum of risk scores of both tests, the difference between both results is notable, with 24790 points splitting the benign test from the malicious one. While promising, we cannot forget the weight of a known banned extension, which in this case represents 24600 points on **250 Malicious** static analysis, about 55% of its total score. If we disregard if an extension was already banned from Chrome Web Store, **250 Malicious** still totals 23610 points, about 160 points above **250 Benign**.

Although the difference is not exceptional, if we also do not consider any *RetireJS* score, **250 Malicious** totals 19820 points against 16995 points of **250 Benign**, since we know that at least two benign extensions have a significantly high *RetireJS* score, enough to influence our results.

Considering just our analysis and excluding Google SafeBrowsing API and *RetireJS*, our risk scoring system was able to recognize more risk in malicious extensions.

	250 Benign	250 Malicious
Web Store Metadata	6715	6420
Critical Permissions	3480	5040
High-Risk Permissions	4750	6025
Google SafeBrowsing API	0	24600
RetireJS Vulnerabilities	6455	3790
Others	180	210
Total	21580	46085

Table 4.19: Static Approach – Risk Scores

Focusing only on our static analysis, in Table 4.19, in **250 Malicious** there is a higher use of permissions, both critical and high risk, revealing less risk only on Web Store Metadata and *RetireJS* Vulnerabilities, where **250 Benign** seems to have more risk associated. The difference in risk in Web Store Metadata can be explained by the number of extensions from each dataset with 1000 or more users, where **250 Malicious** has 140 against 69 from **250 Benign**, since extensions with a higher number of users tend to have more information on their metadata, producing a lower risk score. If we only take into account Web Store Metadata on extensions with ≥ 1000 users from both test datasets, **250 Malicious** totals 2915 points, averaging 20.8 points per extension where **250 Benign** sums only 965 points, averaging just 13.9 points per extension.

Here, the value corresponding to *RetireJS* does not tell us whether an extension or a group of extensions is malicious, as it is just an assessment of the severity of vulnerabilities found in libraries that an extension relies on to work as expected. With this, we consider it a risk property since an extension with a high value of *RetireJS* is an extension that is more likely to be exploited by hackers, not undermining the intentions of the extension itself.

Regarding our dynamic analysis, we can verify, once again, a superior risk score from **250 Malicious**, being 255 points ahead. (Table 4.20)

	250 Benign	250 Malicious
Low-Risk	150	140
Mid-Risk	1210	1280
High-Risk	270	360
webRequest API	240	345
Total	1870	2125

Table 4.20: Dynamic Approach – Risk Scores

Concerning extension activities and their risk, **250 Malicious** outscores **250 Benign** from mid-risk up, showing in general how these extensions can be distinguished by their behavior and the

danger that each one implies for the security and privacy of the system.

Since not a single extension was individually explored, we cannot assume that all behaviors were captured from an extension, as some sometimes require a few steps before working as intended, e.g., *Grammarly* requires a licensed account logged in. Although the probability of capturing more and new behaviors is higher, interacting with each one of 500 extensions would consume much more time, as we are aiming for a more practical and capable solution.

In general, we managed to achieve results that we consider to be good given the difficulties we encountered, such as finding a good malicious extensions dataset and properly evaluating the risk. However, the risk value of each extension also depends on the existence of a CRXCavator report, for both the value assigned to *RetireJS* and the metadata assigned to extensions that are no longer in the Chrome Web Store.

Although we have collected network activity records, we could not integrate them in time in our dynamic analysis and, consequently, in the risk scoring system.

4.5 Execution Time Evaluation

Although not initially addressed, we wanted to check if the run-time of the automation performed in tests can indicate something else about what an extension is doing.

Since the intervals between tabs opening and closing are hard-coded, an analysis of the run times of different tests may reveal whether or not the interactions between the browser and the extensions are overloaded or taking more time than usual.

This comparison can be interesting if a group of extensions takes longer than the average to complete all the tasks. However, we always have to consider factors such as processor, random memory and internet speed.

Iteration	1	2	3	4	5
Time	40:16	37:44	33:23	36:16	36:53

Table 4.21: 250 Benign Automation - Iteration Times

Iteration	1	2	3	4	5
Time	38:08	33:33	38:05	37:18	38:38

Table 4.22: 250 Malicious Automation - Iteration Times

Looking into iteration times on **250 Benign** and **250 Malicious** (Tables 4.21 and 4.22), both present a very close average time of 36m54s and 37m08s, respectively. Even if we consider only 4 of 5 iterations (excluding the furthest iteration from the average), the average times are 37m47s and 38m02s, showing no relevant difference between both tests.

Iteration	1	2	3	4	5
Time	38:39	38:09	32:47	32:36	32:53
Iteration	6	7	8	9	10
Time	37:49	34:28	36:00	37:14	35:48

Table 4.23: 500 Benign + Malicious Automation - Iteration Times

Running a test with our entire dataset, containing about 500 extensions (10 iterations needed), each iteration lasted on average 35m38s, a couple of minutes faster than both previous tests. (Table 4.23)

Iteration	1	2	3	4	5
Time	36:18	36:10	36:23	36:03	33:52

Table 4.24: No-Extensions Automation - 5 Iterations Times

In order to make a fair comparison, we decided to run five similar iterations to those performed in previous tests, but this time, with an instance of Chrome with no extensions installed. (Table 4.24) Averaging 35m45s per iteration, a value very identical to the mixed test but still lower than the two main tests.

Since there was no significant variation between **250 Benign** and **250 Malicious**, we cannot draw any conclusion, as other tests were also inconclusive.

4.6 Summary

In this chapter, we explained the composition of our dataset and how we obtained each extension, our experimental setup, the tests performed, the extensions used and suspicious behaviors that occurred in each one of them. Then, we proceeded to analyze the results from each performed test, where we presented both static and dynamic results, trying to understand which factors stood out more through the number of occurrences of each one.

Before drawing our conclusions, we analyzed each final report (derived from both tests). We listed the higher/lower Top 10's of risk score and the most permissions requested (divided by Critical and High-Risk categories).

Finally, we compared the results from each test and their approaches' score to check whether or not our solution was viable and which approaches (and properties) were more effective to the final score of each extension. Additionally, we decided to compare the run-times of 4 distinct experiments, attempting to find any relevant contrast between iteration times.

Chapter 5

Conclusions and Future Work

There are many ways to make and execute a malicious extension and many known, or yet to be discovered, ways to use vulnerabilities inside an extension against its intentions. Despite the difficulties and demands that this work required, we were able to build a tool that, based on our results, was able to identify, on average, a higher degree of risk on malicious extensions compared to harmless extensions.

5.1 Conclusions

Malware in browser extensions is a severe threat to users that use them to improve their web browsing experience and add new features to a specific website/web application. Browsers' stores will continue to grow as more official and unofficial extensions keep being developed and are continuously updated.

This dissertation intended to improve web browsing safety and raise awareness of everything behind what the end-user can see. We explored the problem by understanding the main threats and the best way to identify them in a single extension, while trying to catch other possible behaviors that may represent a threat to users' privacy and security.

State-of-the-art revealed good solutions, some supported by static analysis, others based on only dynamic analysis and even one similar to our hybrid approach and a Machine-Learning-Based Model to distinguish malicious Chrome extensions from benign ones.

Overall, we are satisfied with the results obtained by our solution, as it manages to assign a risk value regarding having a certain installed and active extension, which should illustrate whether the extension represents danger and whether the user needs to pay more attention to its navigation when they are running.

The average risk score was a good indicator that allowed us to show that, on average, there is more risk (points) inside a group of malicious extensions comparing to a benign one. Still, when

approaching an extension individually, its risk score alone can only point that there is, in fact, a certain amount of risk to have it up and running in our browser.

As we have mentioned several times throughout the document, classifying the intention of an extension is not a binary problem, since sometimes it requires an intense individual study to determine if there really is a bad intention and what exactly this intention is targeted to do. Based on this, the decision to assess the risk, even when it does not necessarily imply maliciousness, seemed to be the best option, since we were able to present a value that combines several factors that can represent danger or demonstrate the risk that the actions of an extension may have for the end-user.

With that in mind, we seek possible improvements to our current solution, as both static and dynamic approaches can be improved with state-of-the-art techniques. Our risk scoring system can have new relevant properties that might force a tweak in all currently assigned points.

5.2 Future Work

Although we had good results, some improvements can be made to enhance our risk scoring system and our hybrid approach to have more valuable and important data from each extension.

Browser extensions are also being developed to run on mobile browsers, where we store a considerable amount, if not all, of our data, so it would be interesting to implement an approach to work with mobile versions as well. In addition to adding detailed analysis to network logs, we suggest other proposals and approaches that could be integrated or implemented separately to make the system more robust and accurate in the risk assessment of extensions:

- Combine sets of suspicious extension behaviors using both analyses to form easily identifiable patterns;
- Directly compare code from multiple extensions and identify copies of original extensions by the similarity's percentage between source codes (e.g., through the institution of the associated e-mail and the number of active users, raise a fake extension alert when comparing a certain extension against an entire dataset and another extension has a degree of similarity above 80%);
- Analyze each extension's developer e-mail domain, if it exists, and check if the domain belongs to a well-reputed institution;
- Integrate HoneyPages [29] concept, or similar, in our automation to capture which DOM nodes are being queried and determine what exactly the extension is doing with them;
- Build a database to save the risk score of different versions of an extension and raise an alert if a new version shows a significant difference, that may show whether the developer has started to take security more seriously or is acting more suspicious than the previous version;

- Implement a data flow analysis to give us an in-depth analysis of malicious behaviors, as it would allow us to identify the potential sources and sinks of tainted variables automatically.

Last but not least, it is crucial that the identities and organizations behind each web browser can have access to the results obtained and take advantage of the calculated risks to improve the validation process of extensions that are uploaded or updated in official stores. We think that sharing our knowledge and feedback with these identities can improve web security for all users.

Appendix A

Dataset of Extensions

This appendix presents the full dataset of extensions for this work, divided into two categories, benign and malicious extensions.

Each extension is specified with its name, extension ID, its version, risk score and if there was an existent CRXCavator report for it.

Some extension names could not be rendered due to Russian character library conflicts.

A.1 Benign Extensions

Name	Extension ID	Version	CRXCavator Report	Risk Score
Nstools	aacnmgkanldpddhhhiogdgbaoeedjdma	1.3	Y	105
Eiffel Tower Pop Building HD New Tabs Theme	aamjlohaogjmfddgdjogjhaamgmhdhj	1.1.1.0	Y	50
Google Translate	aapbdbdomjkkjaonfhkkikfgjllcleb	2.0.9	Y	20
Linkedin Filter Companies	acpcfckgpcacinbdlmbfdmhonfolihgo	1.0	Y	30
	addncdcjbaalekeonckkdmddccfbpiap	1.31	N	30
Just Black	aghfnjkcakhmadgdomlmhlhaocbkloab	3	N	15
AZ Parcel	agmkjabiiimbmecphdaaeajmcgnaloejo	1.0.0	Y	45
reKnowledge DIB	ahiojgkkgogepeoaeajcmppppilaobk	1.2.7	Y	35
Currency converter +	aailnohijenlijndnmnmhjceapkfca	1.2	Y	30
American Gods Themes & New Tab	aiddhbgkicempileffbmfdhofcokmink	2.0	Y	175
Surfshark VPN Extension	ailoabdmglmfmdhagdmlhohpjlbpffblp	3.5.1	Y	155
PaidBefore	ajlmnbhkhikmbdpefencjglalhcepbjo	1.2	Y	65
Batman: The Killing Joke Batman	aledhmpjccgmldfcgiineckmcjbabhf	1.0	N	30
Tăng Like Instagram, Tăng Follow Instagram	alhpdcmejaghdpoeeenlcobkneanenbm	6.8	Y	15
umbrella	anodajlkmobkbpogcogceeeajaf	1.3	Y	40
Mute All for Google Meet™	aphligncnahhmlmcndcpmadohdngngjf	1.0.2	Y	30
GitHub Repository Size	apnjnioapinblneaedefcnopcjepgkci	0.6.0	Y	20
Logr	baeanbiibajakocmgmnoadgljmnknop	0.1	Y	40
FB2 to EPUB Converter	baoohcoldcfpkmcdfghobbjaniifinl	2.0	Y	50
Ascent Internet	bbmpkplgienogedlljhgpmmmmkikdmgib	1.0	Y	40

Name	Extension ID	Version	CRXCavator Report	Risk Score
VK UA	bchfldglhelihmdgfdjfhfiaglhehifc	1.2	Y	30
Plugin_URL	bejaegihcbgnakbfgphpfhameiheenjib	1.0	Y	90
Bitmoji	bfgdeiadckcfbkeigkoncpdieiiefpig	10.31.1224	Y	170
Webclipper	bgbdknfmkdkcnjfkobghlfblijdlnhd	1.1	Y	190
Casinobonusesfinder Canada	bghkcfpnfhmldhohmnkienimppcpmced	1.0	N	40
AdGuard AdBlocker	bgnkhnnamiecmpeenaenljfhikgkllg	3.6.6	Y	230
Organize Downloads by Month	bjlgllbbnbbpbepkhdhbjopdgdcijhcc	0.1	Y	40
The Super Bat and Ball Game	blhppglopaiphpcodmehacnbgmdhaal	1.1	Y	25
HostingSpell	bmaojeflpminkhjgabmdmpajmahcjdag	1.0.0	Y	40
Honey	bmnlcjabgnpnenekpadlanbbkooimhnj	13.9.0	Y	185
QR creator	bnapnikdipejhnceiohjeldjopimjefc	0.3	Y	35
Noty	bocobeipokdapflcoalkhhoaegeccgc	0.0.1	Y	40
Cardano Hoje	cadlepmndhondfchdeadfophgaeedke	0.0.2	Y	30
Rakim HD Wallpapers Music Theme	ccdicpegfiaaajpdobciffiboeikk	1.0.2	Y	95
chrome baidu ad beautify	cdgjnpgcochdjgepbllbphndibppbgcmp	1.0.1	Y	100
IceBucket Away	cdncngodachdenbflhnghegnijlppsgdm	1.2	Y	150
Diceware passwords generator	cfepfnlehmjojjiijpfjapgmcldbjgbj	1.1.1	Y	40
Adblock Plus - free ad blocker	cfhdobjbkjhnklbpkdaibdcddiifiddb	3.11.1	Y	210
tibow statement posting plugin	chnlpfmikkdicmhofhmmllnhleihdeadd	1.0	Y	85
uBlock Origin	cjpalhdlmbpafiaamejdnhcphjybkeiagm	1.37.2	Y	240

Name	Extension ID	Version	CRXCavator Report	Risk Score
Adblock for Youtube™	cmedhionkhpnaekndndgjdbohnhpckk	5.1.2	Y	135
WhatRuns	cmkdbmfndkfgebldhmkbfhneefdaaip	1.7.7	Y	165
Angel Anime Konan Naruto	cmlapjldpjpeljpkoldicbiebdcoopo	1	N	30
Seo from Kenny	cnoffkehapncbdfcnmlaljmmekegdb	0.1	Y	40
Ky Nang The Ky 21 Nén Có- Swinburne-vn.edu.vn	cpdebkhpahgkjnlldkjjbbbmngkibbpeb	0.0.1	Y	20
Classroom Tools	dacldcglacbdodmjfbchncfnndhininn	0.0.2	Y	15
LockPad by Cohort Extension	dbejekhpifgfhbjmfncoobbaffaic	2.13.6	Y	135
ITU Remote Screenshare Extension	dbfojkkkkfbjffimkdekncgnahjeidbh	1.0	Y	65
Amazon Storage Fees Calculator	dbglamehnlckoibnodikjmcfdokehepi	1.2	Y	95
Hikaru and GothamChess Tell You to MOVE	dbooaamkljifogllbgookpiliaphkabb	0.0.0.7	Y	10
Gladdress.com	dcdbmndmdbaddnpjlelohkiiidneifb	1.2.2	Y	265
Push HKGolden	dfhabplllejifdoomlcfo kocnbiphmd	1.0	Y	65
Tampermonkey	dhdgffkkehbmkfjojejmpbldmpobfkfo	4.13	Y	265
Microsoft Teams Screen sharing	dhheiegalgcabbcobinipgmhepkkeidk	1.1.0	Y	40
See Star	djlmbimgikacpcnkjpdkki gjfjfnom	1.6	N	40
Kashback - Cash Back Rewards	dkbpnofijnjdcabfbihddfhchlmidcg	3.0.4	Y	90
Livecoaches.com screenshare	dkedngmhfgpgghfjmahfdkdmbdagfhmmb	1.0.1	Y	90
Iron Man HD Wallpaper New Tab Theme	dkgkjphadlmccngfnmoefjldndekpjma	2.19.85	Y	185
Superman Blue - Reboot by toxic	dkifhieopbafgcfcdijdfphbnbamhmkk	2.1	N	30
Better Lounge by JMPH	dmeliinkeblbjlbecmbifplcbpfdgg	0.7	Y	90

Name	Extension ID	Version	CRXCavator Report	Risk Score
TouchEn PC	dncepekefegjijlfbihljgogepdhph	1.0.1.15	Y	110
Custom Hotkeys	dogblfnfkmjolfcbpgfkhpaaahocjhkbk	1.1	Y	90
Edit in Zed	dpkaficlkfnjemltheobmkabnoafeepg	0.1.3	Y	40
Ghostery Insights Beta	eafjgffiocmnmiplemahnlhgibaanhch	0.7.4	Y	245
Hyperdimension Neptunia 11 - 1366x768	ecglcoagacjikjdpfnjillbdialcmodh	1	N	30
ETH Verify	edbpnigdhhocjocgelbijbnhlphhabm	0.3	Y	95
Fastmail favicon count	edphkeodanpjklfbfopplbdfpofglfpdm	3.0.4	Y	30
Seishun Buta Yarou Wallpapers&Themes	edpmlidioeioekjkcnegicloklddefg	2	Y	65
Adobe Acrobat	efaidnbmnmnibpcajpcgicdefindmkaj	15.1.3.2	Y	275
smarterPod: Simple and Smart Podcast Player	ehlmmfeohfahemdjnknackiaheggkjojchf	0.3.0.2	Y	1290
BlockSite: Block Websites & Stay Focused	eiimmioipafcokbfkbljfdcojpcgbh	5.3.1.0	Y	250
Lil Mosey Search	ekmmnmkndonabckclhghlecaaplomino	0.3	Y	35
Gambling Time Monitor	enlbfghahcpbbjlfioblmbbmjeajmalf	0.1.5	Y	90
FaceBoycotter	enpggepcmcncpempgelhmmndcoeiden	1.0	Y	30
Learn Cantonese	eodeegbgphcmbfhlkcomcnofpeopnpj	1	Y	40
Avast SafePrice Comparison, deals, coupons	eofcbnmajmjmplflapaojjinihckigck	20.5.1662	Y	175
血型属相匹配	eojjjecnphmbhlgaeffffkccdbimcd	2.0.0.1	Y	40
TopRecipeFinder	fdnjiondibmefmjdbokflohocecepoa	1.2	Y	335
My Little Pony G4 10 - 1920x1080	ffaeilonemehncnlpmflkcmldgmjc	10.1	N	20
Made in Italia Style Theme by Spinelli	ffakobimphicifdpjmhbcdcoejoibgak	1.0	N	20

Name	Extension ID	Version	CRXCavator Report	Risk Score
Codingame puzzles accordions	ffkfmp[pojkbkbh]hembjapbalcjchjdjp	1.0	Y	80
Well actually I'm an idiot	fpgghemfneilglikcaklohncjplmhdo	0.1	Y	40
Apistop Pro	fhkioimlijfmbckmdikkadobdmlgn	0.9.1	Y	150
File Icons for GitHub and GitLab	ficfmibkjinpogdcfhfokmihanoldbfe	1.1.2	Y	40
Microsoft Autofill	fiedbfgeceddlbcmgdjgdfeggcjcion	1.2.2	Y	190
	fiekffjkdplmniioinfgkdoacpnefibeb	2.1.2	Y	30
Practran	fignonfnbbdfoopnocnkpjinfhamaab	0.1	Y	150
新亚PS4游戏监控	fjcoeinnhicfnbdpddalbiecjodablkf	1.4.2	Y	80
tabzen	fkc[ljmgnc]jkkcdandfdodbgdefgbjep	0.2.8	Y	75
Discord PGP	flbbfckkblcgfkbbnoalkmpfampmdcc	0.3	Y	80
cwcom Sites	fmlooiimfhjfcfdehlnoiimedklldonkop	8.2	Y	15
GDQFebreze	fmmfcjnbldmfjmplfgfkompfppeemljm	1.1	Y	40
EditThisCookie	fngmhnpilhp[aeedifh]cceeomclgfbg	1.6.3	Y	235
getId.in	fnnhnbaalimobpabecjpeaiblnldakjod	3.0	Y	100
Super shorturl generator	gcbblkepelpgiggpncfdlnfdaaafdh	1.0.1	Y	40
AdBlock - best ad blocker	gighmmppiobklfepjocnamgkkgbiglidom	4.35.0	Y	1150
Style Jukebox - Hi-Res Cloud Music Player	ginjgbgljgcpmbmcnilpgaeecod[kjnp	1.5.0.0	Y	30
Good old time	gijjpmiekdiefime[anafc]pkedgmkamjg	1.0	Y	40
AdGuard Extra	gkeojjjcdcojpkbelgbcpcplegclfeg	1.0.286	Y	10
r/Twitch Flair Linkifier	gkgobdhmlglipgkijfooplgbiembkagh	0.1	Y	30

Name	Extension ID	Version	CRXCavator Report	Risk Score
Xing Autopilot	gkndpenbfbffdecgjhjfpdceohfkmfa	1.12	Y	105
Hola Free VPN Proxy Unblocker - Best VPN	gkojfkhlaki ghikafcpjkiklfbnlmeio	1.186.727	Y	235
Marketo Sales Connect	gllmkcahdekdbapmdffclacbpnicaj	8.6.34	Y	300
Internautas Cristãos	glpmckacaglcpihaadanoloaaamlngdn	2.1	Y	25
AssetView	gmmjlpalijddliofagfpmmeapeojhje	9.2.0.1001	Y	250
auto-pin-tab	gnmpgcbidokemiciojjogpnfkncbkam	0.0.50	Y	210
Classic Chrome	goefhnepjncdbcefgjokbjmafjlhpgij	1	N	40
sardar's downloader	goijnaccefdgccjlmoccekpmnojnkk	1.0	Y	90
Avast Online Security	gomekmidlodglbmalcneegieacbdmki	20.3.10	Y	195
Pinterest Save Button	gpdjojdkbbmdfjfahjcgigfpmkopogic	4.1.111	Y	225
Quick Search for YouTube™	haepcjchonagnpkeaijnbfeldaikijm	1.1	Y	40
betteroptions	hbmoafpbooblnmfpcoondojbopghidjh	2.3	Y	20
Viasoft Sistemas Internos	hcaptibijjlppegiekkgecbggodmoadfb	7.5	Y	30
urltoqrcode	hclhmdgpfbmmlmebdiplibijpfneglofc	0.0.0.3	Y	50
Remini - AI Photo Enhancer	hdlldkchhhieipniibccdefnoobiaapf	1.0.3	Y	40
LastPass: Free Password Manager	hdokiejnpimakedhajdlcegeplioahd	4.78.0.2	Y	190
Trello Tools	hfjpnjlpniibgeeijjeekoknhbbnbj	1	Y	40
Study Mode: YouTube	hhjnoabnlcipahfibejfhankpfahnd	1.0	Y	40
微信链接可点	hiajdanihnbnmhjfoapdlhilfplmpdgb	1.0	Y	30
Scylla	hidglpghoipondigbeekpbpbnehcmbmf	1.1	Y	65

Name	Extension ID	Version	CRXCavator Report	Risk Score
www.smartclimate.kiev.ua	hipjkdjeahcealdhdbdecgdkfpmbgmgf	2.1	Y	40
Marble Fade	hjfopemajhjiacdnhnkaplacbefpanch	1	N	20
Crypto Prices (Binance)	hjlndpiclhlaoemebmfiffocgabipje	1.2	Y	95
Download Traffic Racer Apk	hkadmhebdccbhciacdpcchfkjkgbechk	2.0	Y	30
Kick the Buddy New Tab	hkkjehmcokegjimhipeamifkdkpbemep	0.0.1	Y	70
Zoom	hmbjbjdpkobdjplfobhljndfdidipjhg	5.0.4290.0808	Y	140
Waterfall Popular HD New Tabs Theme	hmignnknkfabfkjioimbibboionefo	1.1.1.0	Y	50
Bot++	hoblebaoppbhplejgelaciapadpkilbk	1.0.6	Y	80
Digital Forensics	hokdnhnmhjhdncajpopbklifablhcem	2	Y	35
SimilarWeb - Traffic Rank & Website Analysis	hoklmmgfnppagjgscpechhaamimifchmp	5.6.4	Y	195
SG++	hopjciplochnbaaeidhcaebhdmdcdia	0.1.0	Y	90
EnvTab	iahdcckhdhmdhdfghjdehalncebffeffe	1.0.3	Y	140
Tweet page	iapceionbjmliaakiopifeppghnnamei	1.1	Y	135
Aurora - Multi Color Tabs	ibaojndojembmeodhedfkacclckgfiog	1.3	N	0
Mighty Fred	ibmriadpogijlkadpealnbdobfabkgph	1.95	Y	40
Sayfay1 Sosyal Medyalarda Paylaş	ifabnedcpjghhlbjjeenhkcekbddjdhpggh	1.0	Y	125
Minneapolis Homes For Sale	igdpoiigibddmgmkbmjmdjnfggbofijj	7.0.0	Y	40
YouTube Minimizer	igjfhbgmolfphjeiknbmpipijljdmdic	1.4	Y	30
Malwarebytes Browser Guard	ihcjiegdanjaechkgeegckofijjedodee	2.3.6	Y	180
Non Surgical Liposuction Finder	iihegdiepenjolinbcbjeldefmomkogk	0.0.1	Y	40

Name	Extension ID	Version	CRXCavator Report	Risk Score
DashOwl	iiienjbjkaohmlpfgpfhijmnebddjm	0.1	Y	40
Github Inline Memes Auto-complete	iiifphamaoidpopebinklenjbiknejm	0.1.2	Y	80
Pancakulator	iiipgkdnelelbbhicddgehjponkhhmbo	1.0	Y	90
QR Code Chrome Extension	iiapagkpdgncnfnhifoemlnodfcmlikb	1.0.1	Y	30
Air Gear.	ilmallgblmoibfbjdpccoigfddbdojhjo	1	N	40
Chrome Remote Desktop	inomeogfingihgfjlpeplalecfajhgai	1.5	Y	105
Wikipedia URL Shortener	ioekneidlojahdoiddhikknahbbkhan	1.0.3	Y	55
Taobao Tmall Data Scraper For Lazada	iphcpginpgdplghfmbgpkmcemoadigk	1.1	Y	90
Cowin Quick Book	iplngloelmhigjffhfhjinpjmcldfahd	1.4	Y	215
Social Media for hotels and restaurants	jdhpoagfkddihkeclliadpdobmjkiabb	0.1	Y	40
Ice Cream	jedfgcpjiegghliidmnllehnbnkcglijf	1.6	N	40
Sea Turtle - Marine Tortoise HD Wallpapers	jennjncdonfkmahfeclafolblkpaabo	2.19.71	Y	200
ZapFixed	jgiflbpkoefoepgmeifoglafkomgjbge	0.0.7	Y	90
Swimming Pool Maintenance Tips	jgnnpdchmgghgkckcdkihmbjddipokhko	0.3	Y	40
MP3Juice - Search MP3 Downloads	jhaaffdflnhnenohfboieepggejcb	1.0.1	Y	5
Can_Senegal	jidhapdjmggkphhecelgcehbdgkdkdhe	1	N	40
Vindictus	jigblgdgikipphecpfldhhabhaamgpco	2	Y	40
MoonFaze	jkaammnejammacclalmefmhkpcjcdcfim	1.1	Y	20
Suspicious Site Reporter	jknembkdbdhdcpllfgbfekkdciegfboi	1.23	Y	115
Cisco Webex Extension	jilhmfgmfgeifomenelgieghnjghma	1.16.0	Y	40

Name	Extension ID	Version	CRXCavator Report	Risk Score
Grammarly for Chrome	kbnfbcaepblcioakpcpgfkobkghlhen	14.1026.0	Y	180
SCS EdTech	kbkmebhbndngpnmcffgkngkglajfmpg	3.0	Y	40
知识星球-评论助手	kcbickhdcnhckabhlbdfhjpdnjijpdmipo	1.0	Y	100
Final Fantasy Wallpaper HD New Tab Theme	kdiiajgmckplhgkklhlopgeagopcmmnp	1.4	Y	40
EtherID	kkinkecbgenlipcookacgpnmahhhiccdh	0.0.2	Y	120
Auto Repeat for HTML5 Youtube Videos	khkihoopolaklepkckddoohipmnnejii	0.0.1.2	Y	90
Forest: stay focused, be present	kjacjdnodndnpbbbcjilcajfhbhdhkgpgk	4.15.0	Y	115
NotNews!	kkohmdcmgddjhcmdeojibcldfponodod	1.3	Y	165
Shadows in the Dark	klmjfmfcjmjaihehccgjoecagnmigpb	1	N	40
Tardis Loader	kl0bipibcaodcdpmiapialidbpcmpffc	1.0.9	Y	85
	knaddecekpkgcfgaefcdkjbkifneeaof	1.0	Y	40
Yen São Yen Bac	kociejnedejijijmionejgcmhbkjgob	1.0.0	Y	20
Chennai Systems	lbdahlmafahjggmlnaehcgcncnhjadcnf	1	N	25
Lifestreet	lfbgmebapjnockfokmplklmcikmiccme	1.0	Y	25
Tampa Bay Pirates Popular Sports HD Themes	lffdpohnedkegnhnplobnjjamgmhckmk	1.1.0.0	Y	40
Quick Links	lgdefdcffekannfhikahgncilmogfci	0.2	Y	40
Huntflow Magic Button	lhhehchebjnjaeklbpoanpogkggnhadk	3.18.21	Y	25
Skype	lifbcibllhkdhaoafpjfnlhfpfgnpldfl	10.2.0.9950	Y	70
SaveTik - Tik Tok Downloader	liilbblpakjjohhllknkgcgfiabehbiai	1.3	Y	75
Ad Extractor	lklhghpmbgfeacoemajbggoimcajp	2.6.9	Y	50

Name	Extension ID	Version	CRXCavator Report	Risk Score
The 100 HD Wallpapers New Tab	lnlpehdoofgijlkffnfknoeedeckck	12	Y	75
QuickAlerts	loafcmllacjldigeifbmbogkbihmblij	2.11.14	Y	150
Gazelle Snatched	lobkjdblepleddkoepdajiaognkpnigc	2.4.2	Y	90
Cashback service LetyShops	lphicbbhtfmlmgomkkhjfkpbdlncafb	2.10.47	Y	145
New Eden Word Bank (NEWB)	maaojdhpfbfbfhogccckalfhhiagian	0.0.8	Y	25
Mold Busters	mceoogjglfbmaamdeaenafmmcojmkieb	0.1	Y	40
SnapCalc - Math Problem Solver	mcnjlbnncmfaeephaoaknienbohggng	1.0.3	Y	40
LeetPlug	mcpaohfldejneeahigainahjenkgmef	1.1.1	Y	15
Excited Replay	mddmkjmppecbdnpepgfeliideiljhbjaaj	1.1	Y	80
History Search	mgacnbmocekcfjdbfdpnbjnfhaomffhk	1.5.40	Y	200
England Cricket Worldeup 2011	mgmebyebeedmiieinbokhdlepmijjock	4	N	40
Daenerys Targaryen House Targaryen Desktop Wa	mijlebnlfpeppodnegpkodfajkdmaajmc	1.0	N	30
Cristiano Ronaldo HD Wallpapers CR7 Theme	mkdmmimfjnitfdpmfhhheonclhkgbhhfg	3.7	Y	175
Font Changer Online - #1 IG Font Changer	mlffdfbokfecagejhmfcpjigonammdh	1.4	Y	40
Night Shift GEMS	mmocigemagdmpaiaadgopbihcfjoeicci	0.1.1	Y	40
Madison Bumgarner Themes & New Tab	mmfmnoocldbjfdbibohppbkhkhdckhbp	2.0	Y	175
eTelecom - Tong dài CSKH	mnpcebklghjmaeekcpihikigpoiejdnipg	3.3.0	N	45
Social Media Marketing	modeokidabhbgbgaebbfhkgcglaimekko	0.0.1	Y	40
Angel the Mudskipper Easters' theme 1920x1080	mpgmioipinogplbceoaitemjmlobelhnn	1.1	N	40
Câncer, vírgula	mpkjnojdbdeamappdfbjpjeoghmpgpnha	1.1	Y	40

Name	Extension ID	Version	CRXCavator Report	Risk Score
Vimeo AutoPlay	nbnmlmfmedpoanpenhfjmcblplajakje	1.0.1	Y	30
迅雷下载支持	ncennffkjdiampmcbaikmaiiddgioo	3.29	Y	255
Netflix Titles Rater	ncoinhpcdpqfimgogkjcobdbkglnffbl	0.0.1	Y	70
Gamers Tab	ndfpbeikamamnfnekcffpmdmldmlgahcmel	1.3	Y	90
FG pause/resume	ndgdcjdneftopopekldanijfafnkodel	0.2	Y	60
Add URL To Window Title	ndiaggkadcioihmhghipjmgfeamgeoi	2.2.1	Y	30
Liam Foxinator	neighkpnbcgljifgdmijhabhgifdhavn	1.1	Y	30
NoCors	nfakhfincbmkmomdkmknpebftpalkkcm	1.4.1	Y	200
No cache refresh	njacldedajpdoofkhpfcckdkgcahjlfkj	0.0.6	Y	95
Rei Ayanami Fan art Anime Art	njmokijpajmfjcaegfkckmdhoogpjij	1.0	N	30
Free VPN	njpmiufchgidinihmijhcfpbdmglecdlb	1.1.1	Y	135
KOVE CastCombo	nkdhaoohlkdimmibbpakmeppadhpei	1.3.2	Y	135
Servers Room	nkfbckddfcckjnpnmickbheeijikghego	1.0	N	40
Chalmfood	nlmggbgdbnkbnpneedonlphbkfjkknpc	1.0	Y	90
Weeronline actueel Nederland	npgafdngocgnaincgfbpeblbeaadkpfa	3.3.1	Y	30
Alone Again	npmidhdaebjjcilhdgdimmihgendhgc	1.1	N	40
CatchYourTicket	obbmigdelgnhapmkfkihcfhokbmelhnb	0.4.21	Y	300
Real Reality	obkcimklomeknmfjmgfkggenjijioenj	0.12.6	Y	170
Heimat Concierge	odkfagmdcjipgmopoikbapdagjmlopl	1.0.0	Y	20
Arkansas Razorbacks Small	oebehebaecmdelhijhaigjcompmgac	1.0	N	40

Name	Extension ID	Version	CRXCavator Report	Risk Score
EğitimPanel Ekran Paylaşımı	oenmbmljdbmknkgpmnmjkbmehlobkbmdo	0.1.0	Y	65
Kantai Theme 20 - 1600x900	ofdbhhfhfoelkefkalimfojofghocm	1	N	30
Custom Cursor for Chrome™	ogdplmhglpejoiomcodnnpjfgecpmgale	3.0.1	Y	80
	ogmjkaljmcaenpkkgdghendiijjnmepfb	2.1.2	Y	105
PasswordViewer- Password Revealer	ohahnaoiiemjagndndajfmjnfbbpdomoi	2.1.0	Y	25
Tripadvisor Browser Button	oiekdm labennjdp gimlc pmp h d jph lcha	1.1.2.4	Y	285
StyleFinder	oinajfdbbnippmbfdbffhcajekoamed	1.15	Y	120
AliHelper - shopping assistant	ojclfkinnapkabameogippmeedicean	2.24.6	Y	195
KeepTabs	omipmlalmaccpahnjbnoipcpkncjaoci	1.2	Y	100
Thomas	onhplkgodhmpejojnlnpcfebmdacpmhm	1.0	N	40
Easy Color Grabber	onopoladlnicjogiadokelbhioobloa	1.0.0	Y	20
Netflix Party is now Teleparty	oocalimmgaihdkbhlhfgmpkcpnmlaoa	3.1.1	Y	35
To-do-list	oonnhgbnbdanmicdkncfmkgclplohbln	1.0	Y	80
Far Cry 6 Animated New Tab	pepeofhffocmdnmfkcpeddfpipklakh	0.0.1	Y	15
Fabrikatör Chrome Eklentisi	pdfanecjioplmnnjfgfdeicobigfali	0.0.2	Y	45
Unblock Youku	pdnfnkhpgegcpcingjbflhkljeighnddk	3.8.12	Y	270
Shooterz	peafahgkhhkflbkapndmpfbblkojeomm	1.0	Y	40
Black Rock Shooter	pglfjajnmkfibnppihlfjlboddiiceen	1.0	N	40
NUTabs	phcefmjlbehmneoegeokgmaboiklbnnf	0.1.3	Y	60
Location Tap	phkknleboeohiodcdcolkc djppbhahop	3.0.5	Y	60

Name	Extension ID	Version	CRXCavator Report	Risk Score
を切り替えたがっているんだ	pkajpcdbkgndgbokblbgjfcjodffn	0.0.3	Y	150
	pkdhofhdmfoncknhldjhbibddimibf	1.1.2	Y	30
FeHelper(前端助手)	pkgecpejnmalmmdimnhkfkafefagiiaad	2020.05.2810	Y	150
Cancel Authentication Prompts	pkjgpnjlaeejfpddahbfhcmnfhbjbl	1.0.0	Y	110
Louis Vuitton Bags HD Wallpapers New Tab	plbakgdpghkinkkjhbfgkanngeeie	2.19.78	Y	215
Christmas Chronicles Wallpapers New Tab	pmafondefkdoenakgogppnbegpobpmi	2.6	Y	55
Vietnam Salary Calculator	pnmnpealgaokichjfdfpikfbfgaeghc	1.2.2	Y	45
Columbus Blue Jackets New Tab	pogjhlonomamjpfkmlimlnbegfmaelb	1.1.1.0	Y	40
Tribal Wars Train	ppajhpdaingbnajnkoidegmkbldgpflok	1.7	Y	30
Windows 10 Accounts	ppnbneolgkicgegkbbjmhlideopiji	1.0.4	Y	40

Table A.1: Dataset of 250 Benign Extensions

A.2 Malicious Extensions

Name	Extension ID	Version	CRXCavator Report	Risk Score
Night mode	aanohaaifnggigindonkplcalbamddg	1.7.0	Y	175
Color Dino Game	abodebeifaichgdiojkluebhaknpbjoj	2.0.1	Y	120
Natural Sounds - Mood Relax Focus	adoagmhahleicgmjdjhdghoagadoak	1.2	Y	305
优酷土豆海外播放	aeocankoflecfaiabdkdjkenofcpiagb	3.1	N	190
T-Rex Dino Chrome	afdpaiahlpgichpphnpdodjphbddelf	0.2	Y	110
Brainly - The Homework App	agdjemejfihopjkabopbnadjiifbijokh	1.0.3	Y	140
Gacha Life New Tab & Wallpapers Collection	agjiifbkagnaljhlgmaogompbnaaleoa	0.0.5	Y	150
Close Right Tabs	ahdmifnjcpekllsobmgkkidkaedkodii	1.0.0	Y	165
Super Mechs	ajoeplmoggaookhlcnpiciabboelflei	1	Y	140
Change Cursor	akchkohnglcihfggdbhfegmpkjcdckllhb	2.1.7	Y	245
MathHook	akellaenjhgmejppedgbogginblehmg	0.1.0	Y	285
迅捷画图	amggiokkmhmmjinnndgfjoeflhlhejog	1.0.1	Y	205
IHS Highlighting	ankngheammfdckeigggcmibipfobpkmfco	9.5.280.0	Y	260
Day night switch	aokhljcmhafilftbbinhjheldjekio	1.2.1	Y	170
Screen Video Recorder	apiggaccmejkhhjjdmbldgoanbilnmfjb	1.2.3	Y	170
Fortnite Wallpapers & New Tab	apphicobkoechiljpbpplafghapnihah	0.3.0	Y	160
FiLMiC Pro - Video Camera	balfbjfoomdpodkibnonmecjgiofmle	1.1.0	Y	120
Online Converter	baonhnbkpcocnnlmejhnmchodfcjkilo	1.0.1	Y	215
My Email Tab	bbkekfpacakcmokjdajdmndmcaknbbcn	1.3.0	Y	255
Sonic The Hedgehog Game	bcgaiaoohedkokgnicpaoifefpnohda	0.1.8	Y	270

Name	Extension ID	Version	CRXCavator Report	Risk Score
Tests-moi - Offres shopping	befplhfnaknjdgnogagnlpkkljljod	4.7.0	Y	450
Missile defense system	bgdlnemflfkacblcpjgijgakmfkkgloge	2.0	Y	120
CheckChrome	bgjhmmigalkldoefglpgfamapoconfo	1.1	Y	140
Girl Wallpaper HD Custom New Tab	bhnehmkbaemgblhbdleonkpfieibjgb	7.0.0.0	Y	135
AutoProxy	bjbdlacnaicmclhmppiejhmbphcblc	0.0.1	Y	280
Candy Crush Soda Saga New Tab	bjcmmbjclmapcbpmhnlckfhlbhhnpimd	0.0.3	Y	140
Email Finder	bngmobitapgiphgfeiegibbmaoffdoaa	2.0.2	Y	255
SnapCalc - Math Problem Solver	bnkfindcaablkaikpghncjfpfpohlehl	1.0.3	Y	150
	bogeiijdedckmglbpndenepbcmoijedi	2.0	Y	195
Google Meet Grid View	boknfbmocmmiiplogmkhgkbknhmldia	2.40.0	Y	135
My Verizon	cbeheemkdnigelpcgdjabhhfleabpnci	1.0.3	Y	150
Youtube Music Party	cdmmbgbbolflmmfecknfkcfnadaocg	1.0.2	Y	155
Winter Snow Wallpaper HD New Tab	ceedopkemdpegdolakjlhefcelfeedkk	2.0	Y	155
Microsoft Teams	cfpojningikehbalpifbfnofalkmeikm	1.0.3	Y	140
Video Downloader Plus	cgmprnfilbdlmohglaobhngamnglfkgeb	1.1	Y	265
Roku TV	cjfmjhhakeknhldlciafffkpfjjajaca	1.0.7	Y	140
FVDTab speed dial	cklafjchoobldkimicddpobdpfkckfp	1.1.4	Y	260
Brawl Stars Wallpapers and New Tab	cnlfpomadabhegoddjkdoojbhehcgenn	0.0.1	Y	125
Draksy	cofaagajkokfibgfiobmapegecbgpkli	1.5	Y	200
YouControl	cokemedmlgcmpatdijlogdincddpjbgb	1.7	Y	185

Name	Extension ID	Version	CRXCavator Report	Risk Score
HBO Max: Stream HBO, TV, Movies & More	cpmpipfpcgoiedkehiobjidambkldfi	1.0.3	Y	150
ProfileMate	cpobppcoiojpficphicellijjeogcpf	1.6.0	Y	175
Material Design Color Palette - Click to Copy	daejiicekmhkbpgahnfhfmpmjhojhpane	1.0.7	Y	145
Happy Wheels New Tab & Wallpapers Collection	dafhaonnaogomjfamecoiikojkenaka	0.0.5	Y	145
NativeDoc	dbfmbgmjomgkleanjebbonjhukigimgh	1.1	Y	350
Crazy Street Racing	dbmki bokicddahoklkcejnmlhgomfjpa	2.1	Y	135
Movie notifier	dcbcgjecgjbphkjhiodlobklcpookf	1.0.7	Y	225
Free VPN ULTRA - the fastest VPN in the house	dchimebmiinkoaeopfijcdmpjhhdcof	1.0.10.2	N	280
科学上网	dcknjolmcfdoafcfipdkmnofdiogma	0.2.8.3	Y	310
RSS Feed Reader	ddbkebecdkpljhlkehkmbiapohccgl	1.1	Y	155
Free TikTok Fans, Followers \$ Likes - BOOST	ddegmaianhjgfdjnfijpjlmgcpolnb	1.0.2	Y	150
Trust Wallet: Sim CX	dddpeopljcbcaofofjenhncdcmloob	1.1.0	Y	105
Gta San Andreas New Tab	ddflcofeonemhhjhgcdmcnibkopjcho	0.0.5	Y	145
eye guard	ddjgapnkmndalaeffojlnificfein	1.4	Y	245
Scratch Notifier	ddjlhbkkjhngiaghjibekchheccnbf	1.0.2	Y	225
Alt CSS	deaodobjfcolfhkecnghdclnlmfdje	0.9.4.3	Y	160
Ads Feed	deciloopcooglpjhombllbjeenohbpg	2.10.1445	Y	220
Inked: Tattoo My Photo	demiaanijbajdmkfpengamcgfklgige	1.0.3	Y	215
Flash-Radio	dffhheolfefjddjhkojcmcdmfmgdabpa	1.0	Y	155
Vans Wallpaper HD Custom New Tab	dkdnpoifaochhlgooalahkegcicpnbhj	7.0.0.0	Y	125

Name	Extension ID	Version	CRXCavator Report	Risk Score
Simply Reader	dlbfminlfoedlkcblpmnbfjommaebc	0.3	Y	155
Custom Search	dlljemmfdaoilbnjhpcpnekjhdohmdn	0.52	Y	215
Milkshake Cafe	dmjjjjakdmnfpebakpaobjbfnmjndndmel	0.0.1	Y	175
Network Activator (HP)	dnoojilbfcdboiempbolfbmfmlfalbi	0.0.0.4	Y	155
Adstopper adblock software	eaaidjlapgmppcenpgkeaonekgahfemcb	0.9.0	Y	280
Silber Bird	eaehbdhmohpmebonhmlahopbinkgdkml	1.9.11.1	Y	180
WhatsApp Web	ebhebkkicbgkandemdkjkjchndjcmje	3.1	Y	130
VKStats	ebifhijhgljalnmjnonafaihefgehmfc	0.3	Y	230
Dyslexia Unscrambled	ebobafleaeaaandaofchakjljjikdfg	0.6.4	Y	305
WebNotes	eiaadlbidoeoplecpogdailomdnahhich	1.4.3	Y	155
Photomath	ekjpgaiepbcajanmlakjlblacmmagli	1.0.3	Y	140
优酷海外版(1)	emmoddikhgncnaikamapbk ggedoafomi	2.8	N	140
gem	enmjddbghmojhllbblbblolfljndkkjn	0.1.0	Y	250
Eyedropper Pro	eodpplilocijojgocbefbklnmgileppo	4.23	Y	200
Hack Instagram Account 2020 V1.0	fagndphbcoffpcfcffhffgpmofdeeaaj	1.0	Y	125
Cave Wallpaper HD Custom New Tab	fcimmmfdaldldifhknhtfbikcfpcffla	7.0.0.0	Y	125
Recover closed tab	fcjlmbbejgnfpegiipgpfacipecopdkf	1.30	Y	165
Brawl Stars New Tab & Wallpapers Collection	fdempplfnopcafknkjgoajkgkcphelegg	0.0.5	Y	150
Terrific Shopper-B	fdgadolaoapiibejmhagmecjaniteon	6.0.7	Y	340
news.bazza.in	feieglghjamjchknhdnhmpbemooaifbc	1.1	Y	140

Name	Extension ID	Version	CRXCavator Report	Risk Score
Mobdro	ffcghdnemnodcjlpmjajieecigcndl	2.6.2	Y	110
Kotnet Login	ffdmhfbpjdldhcnjndjohjoiaindlog	1.9.0	Y	175
Helix Jump	fffhmgahcphihomlmmoafbdogddbdbo	5.1.9	Y	130
Nucleus VPN	ffhhkmlgedgcliajaedapkdfgdbocif	0.8.7	Y	200
Pin It All	fgkahiidpmgagcfiodnkecmecddgldk	1.0	Y	215
Angela And Tom Dream Wedding Online	fgmkffcdhcgbbllkbjcbekcanaofdi	1.0.2	Y	135
Minecraft Classic Game	fhedmlbajbeiledgmgbnjdaelmccclknm	11.12.20.1	Y	140
Video Downloader PLUS	fhplmmlnpjllncfjpbpbjaoeijkogc	6.0.2	Y	460
Zoom on image	fjknppfcgokhfkmachnkfbahdlhgiafj	1.0	Y	210
优酷海外版(2)	fmakoabecagdgdfhohcdjiaifnkmpldj	2.8	N	140
ÉireVPN Beta - Free VPN & Ad Blocker	fmhlhjefkgemampnpomkajhdphjnhmie	0.1.7	Y	210
Gta V Wallpapers New Tab	fnnaolmjooajipoiaackbmjpmppgidngc	0.1.4	Y	125
Reduce RAM Usage	fnppmllgcobhkildjldlclcgcllfaaffpo	1.0	Y	200
Nano Adblocker	gabbboaceombphkmmnoamkioajlkfo	1.0.0.154	Y	360
Zoom +	gadflkeikojdjckcjckjaenohmfmaokd	1.0.0	Y	255
Mint Search	gcppnihgpfjckijmccfamonnfjocgmelh	1.0	Y	225
ReachEdge Website Compatibility Checker	gdpmjnadhcfnheighnfnkiojobcfak	2.5.0	Y	305
Roblox New Tab	geoahbjdobejhoncclpbljdjaglakaoa	1.2	Y	210
Fenetre Mailto	gepijnnkhniemlhacebnhendgogkame	1.2	Y	195
Binary text converter	gfabkfmgcaffpcepocajidfbicgnepif	0.0.0.3	Y	140

Name	Extension ID	Version	CRXCavator Report	Risk Score
Savings Alerts	gflpeapppfjfecjmibidlnfggdifmic	10.1.42	Y	365
PBlock+	ghglajkejhjaonaednmdoplhjiokljhd	1.0.4	Y	215
Cookie Cruncher	gjjjomblijecmngnpdagbbmliidkhiad	1.0.8	Y	270
Gun Shoot the Apple	gjkgoneongcjgidecceapgdmitblfjfp	2.1.1	Y	125
Ouverture mail	gkckdecbiahbaeaplakdedlgchfghli	1.0	Y	195
Fall Wallpaper HD Custom New Tab	gkpkdeohcpjiepfhcmddmmphdoflbpji	7.0.0.0	Y	135
Farm Heroes Saga New Tab Theme	gncdaokgidgloebfhdmedhmfjgbapf	0.0.2	Y	140
Web WhatsApp - Download For Windows 10/8/Mac	gpfejgbhgkbeahobnfmelgdjjjnbododop	1.0	Y	125
Acortador URL	heapgkainmjfgganbefabkjdihkeieifm	1.0	Y	230
Analyze Page Performance	hemibacgndhdkfhahkjdedjdgfapmfki	1.2	Y	215
WebinarJam Screen Sharing	hgjbncindoeafnjfjggbkckjhomfacilk	2.5	Y	135
Saka Key	hhhpckekipnbloiiiiiaokibebdpdpakdp	1.25.5	Y	210
E-mails openen	hhnemjoniijbpcgmppmdkehnkckcadbki	1.0	Y	195
Kill Notifications	hibhbkljenkheombjhgkmkpkjcjkgej	0.1.0	Y	190
Blinkk	hiifcipebmikcaninmakpdihkojaamgf	1.0	Y	200
thinkery	hjftaafenmihckkohhbbehlimaldghle	1.22	Y	275
迅雷远程下载(非官方)	hjfkfpcdmfaofbgfcngcoegnflbja	1.2.2	Y	205
IMO FOR PC DOWNLOAD - IMO PC GUIDE	hjhfejdopdlcmajnefokobpcglbbkhf	0.2	Y	125
Imo For Pc Download – Windows XP/7/8/Mac	hijjiaabciononohpetioochfbfbggmnl	1.0	Y	115
uBlocker Pro - Free AdBlock for Chrome Plus	hjbpfllngbnfhehfpplpbfmgalebeo	1.2.0	Y	360

Name	Extension ID	Version	CRXCavator Report	Risk Score
One Piece Wallpapers	iaidmccllchpjkcgjmdmnikjoeiieki	0.0.2	Y	125
The Tech SEO - Quick Click Website Audit	ibclohpehkoagiennackaiphhkkolfnm	2.2	Y	140
AdsFuzion	icegedoajhdclokfmeobnnbalabmpehm	1.0.1	Y	415
Vex 4 Unblocked	idekohalmdeibamjibonpioogmloaeog	4.0	Y	195
Level Up	iebondhlmljlonlgoijjffmmndnefbpfkn	2.11.0	Y	140
Pokemon Cursors	igbbbfccjjfdgmdoiglgagppfdddjem	1.0.1	Y	195
IQ Option Robot	iglkjoibikdflfcjidabfmgodjkmkxbd	1.2	Y	185
Ellie Toddler Vaccines	iglobcklgbcoleekjhmhndhnlimgmen	0.0.1	Y	130
Unblocked Gun Mayhem	ihconhklflnbnfjbicpkglcpooiingo	1.6	Y	180
Translator - Select to Translate	iheomjndpgpejeofegebgcjalfhfmhnj	1.0.2	Y	220
Cold Search	ihfelciiegpgiioegehelegcbmgdmkcccd	0.2.4	Y	150
Bing Homepage	ihhbmhpgfjdfcfnlghlgpkmbckmhmkb	1.0.0.5	Y	190
Remini - AI Photo Enhancer	ijgahedgacnbnbdakolnhhkhbjbhjgfao	1.0.3	Y	140
Open in Moz backlinks	ilccbikdhfidjknjeafoemdnlblbc	2.0	Y	145
腾讯设计导航	imeammgbgncchlhdoghcpljkbpbnpfj	0.3	Y	305
Link shortener	imkdcbmfebkedbleekdbcimkanidpepo	2.9	Y	165
shopee-fun	innhdkcpobgnmhpcnhijmpkdedcklci	3.9.3	Y	370
Kimetsu no Yaiba Wallpapers New Tab	iopnipljgjjfdmlkcnogladfohaendhm	2	Y	165
Thop TV PC	ipdipcnbfaaccocohkokdmaddiplopbcb	1.00	Y	130
电影VIP	ipielliclaogmnmnmpdpabmgpjnfpgf	1.0.0.2	Y	305

Name	Extension ID	Version	CRXCavator Report	Risk Score
Badger Wallet	jadobjbegibiopkifknkfnohlelpocll	0.7.16	Y	185
Colorful Youtube	jaiidakeldcnpbibnaemaooahcboagh	5.12	Y	150
thanos	jbhkbbppibmpcahaodnhmokgeljbbfkk	1.0	Y	155
Star2Star Browser Extension	jdcojlpblbkaappphkpgcpmhcabcoala	1.0.8	Y	205
Screensy - screen recording	jehgkfidilkpfbodnbjddbnmjnbhccn	1.13.6	Y	190
No Notifications	jfididmdpnaajdfajhlohkejmjmjbgkh	1.0.3	Y	295
MiniRoyale 2 - Battle Royale Game	jgiienpkjeehjmfnhikggeekgnnccgjpl	1.0.1	Y	130
Green Messenger	jhcfnjahmdghhebdaoijngelknfkbn	15.1	Y	240
Backspace	jicjblfmbhkgjobmmiehiipndnimoijj	1.4.0	Y	245
Matematikhistorier	jidpijldnidfjkbffeehcdgfiifnjdko	1.0.0	Y	270
eye protection for night	jijioamppbmdiedokaffaciebdaaejik	1.0.0	Y	270
Amino: Communities and Chats	jiodaacchkgkgngmgbjoommaebhbkke	1.0.3	Y	150
JSON X	jicgokadnhecikghhckfaonfkdlmkomh	1.3.3	Y	235
1click Merge Windows	jngapkhejnmlegkjhlcfpkdmhldapikm	1.0.1	Y	130
Clash of Clans New Tab	jnilifbmnephjneeppdfajkfegnelicf	1.1.2	Y	255
XLS Editor Online	jolpcbkpljocpfcaiahbhnppmolech	1.0.6	Y	155
Garmin Connect IBP	jpbgemineckpfioeoadikachcpcfddha	1.0.5	Y	195
Wallpaper Rocket League New Tab	keahgpfkneehfleeppfncpnkifgnk	5.6.24	Y	190
Emo-sion [Emoji Extension]	kkgmokalgilgennilfapllhdcpffkbc	0.0.2	Y	245
Favorites Checker	khifdnlptaleccjbioapcmgnjngajepm	2.2	Y	205

Name	Extension ID	Version	CRXCavator Report	Risk Score
Cowin slot Booking BOT	kiagbicgfegekabhppjdpjpaflbbbd	1.5	Y	110
Form Editor	klacimjlbpfompicealiiifcdjnkbpn	1.3.3	Y	155
Sakura Miku Anime HD New Tabs Theme	kmphnbplajibkgobjkelcggoalapnkgj	1.1.1.0	Y	150
Flight Search	kobkhhbfkjdaackpjdglmlggdnaknecb	1.2	Y	160
192.168.1.1 Admin Login	kofhlhloimifpiadndnmeldlndckmnleh	2020	Y	140
Better Pagerize	kolbhnlipefgpejilgmhbbcombhphihl	1.4.0	Y	270
Crackle Activate Guide	lcpjchmncnefonhabobdkgffgdokck	1.0	Y	120
streaming film gratuit	leckhndclghnbobfeklfjcapgaeaamg	0.0.1	Y	140
Balloons Matching Deluxe	lepcalholejedahdjehfkcmllbcoban	0.0.1	Y	145
Amazing Slither	lfhiggppmmnkfhabmffglimmklpkimb	7	Y	120
Flash Video Downloader	lfkgajffoebieppaiegjlheepiecad	1.0.7	Y	285
FORBES	lgjogljbnbfjcaigalbhiaagkboajmkkj	1.8.44	Y	240
Free Robux - Roblox Free Robux Generator 2021	lhbkbmmfgaldckelefhobefneedcbdok	1.1	N	140
Roblox Robux Generator	lihjokegkdegdfddbfajdlcldbfofdbhl	1.2.8	Y	140
MKV to MP4 Converter	lihplaklhelbplnabphjkbiggonoalei	2.0	Y	140
Dino Run	liilcplcpobpmljcabioodolfnagiod	1.0	Y	185
Pingdom	lkeniemiefebmdabdllkkbhdnejhoka	2.0	Y	165
ProxyFlow - Free VPN Proxy	llbhddikeonkpbhpnncnialfbpnilcnc	0.0.1	Y	360
Flag & WHOIS SEO Domain Full Info	lmihgdonoeihjgknagpeeijlgigdhbl	1.0.7	Y	205
Twitter	lnhafpappkfaotibaimdblonlcmclladf	1.2	Y	165

Name	Extension ID	Version	CRXCavator Report	Risk Score
Vex 3 UNBLOCKED!	lpnmijebjaopgallojdmaahlocfamdj	1.0	Y	150
Microsoft Authenticator	mabdjppmcjppjlojggpbonahnijlgkf	1.1.0	Y	120
To Geniusweb	mdkhkfeigkfojmobjknkhidiedjndnie	2.3.1	Y	205
YouTube SEO Checklist (SEO Ranking Tool)	meajddijjmeaibppajnajjiloknomijjp	1.12.0	Y	110
f(x) Traducteur pour Spreadsheet	mehikbdjadagnngdnfngopmgepdgknak	1.0	Y	255
GumShoe	mekpfnngodchodemgmkhinohkfejfeea	2.14	N	175
Download Facebook videos	mfojiahihccminapjhdbpilcakhpmsg	3.0.2	Y	150
eBook Reader for Google Drive™	mfpbhmcmafaeajfpehaoijecamlehpl	1.0.9	Y	200
Mailöffnung	mkcpefgkfjaghdmkncckfooolcbfale	1.0	Y	195
Parental Control	mljibhffjkpchemfjdmaikdclemfcgkn	1.0	Y	290
The Nootanator	mlkhnccglcdainbppoolfpgnenfinplp	0.1.3	Y	165
URL Shortener (support QRCode)	mmghgajjciodkdocdbkeplfnppomfjpp	0.0.4	Y	270
Nativ Send Report!	mnnhhkfnokpbmejlkpnimkcijgeaeml	1.472	Y	305
PCM Tools Persian	mpagfchelimlopofdi ginnmffaidigckh	2.0.5	Y	175
SocialMonkee Submitter	mphmcdpcokgeknmaineflodmagodpmdd	1.1	Y	205
EduPageTestHack	mpjlmnnbpcfaemadngfdjhpoikkblell	4.7	Y	110
Burn the Web	mppbcmjglgknkmiamicioakfkikohib	2.0	Y	205
IsItUP	nabhcfbaakelgjpolfnjmobaplnlhadf	0.30.0	Y	125
Copy Paste Clipboard	ndnflkjaammnmjbgncecbacbidpjhcbpe	1.0	Y	180
Discord Popup-Window	nfhjhclki cfjfcjdpdkapbkmmmbnbpf	1.3	Y	120

Name	Extension ID	Version	CRXCavator Report	Risk Score
Top Bottom Scroll button	nhbojecnoedpeldiaipohpbbpljeldpee	0.16.0	Y	175
Appstation Video Downloader Professional	nhmjejmgcodjfoapdolfhafdgbgghpl	2.21	Y	325
Relay: Chat on Every Website	niifmlaakldieiifdlnhjkghaiaipjdmh	0.0.103	Y	145
Lion & Tiger New Tab & Wallpapers Collection	njenadbcclafnholffpnkhnjnikbkmaa	0.0.5	Y	140
InflueNex - YouTube Analytics Tool	njmhejnammknjdmefflmfblbdcckocla	1.0.8	Y	215
	nkeddgniojfkbihtkooahkjpaigogldmn	1.0.11	Y	165
Pixlr Today	nkgmhpdhaijoaeaoakmhdpfegfmbmfjg	1.0.15	Y	295
My New Purchase	nkhdblldgnapmagaemenbebdblcooepo	2.1	Y	150
SoundCloud Likes to Playlist	nknddcbocfchmjogahmikkapkaoglkoh	2.0.1	Y	195
Typing Speed Test	nlpgbjanddbdbbalmofkobbhgciaghn	1.0.1	Y	120
Right-click On Any Image	nmnigllmkknngbolhijaacgekmdbbopk	1.1	Y	250
	oabmfniijbopjmenkibdpdcdpdlgoup	0.0.2.7	Y	255
Video Speed Controller	oadfmbaggknipdhbhbaomghbkgfdppmme	0.0.1	Y	140
The Sims 4 New Tab & Wallpapers Collection	oao khcnfpehnffpblncgjelgipcdllo	0.0.3	Y	125
MKV to MP4 Converter	odccmejckbebmomgagjffcmndhkcebah	1.0.6	Y	120
Triangles Dark & Black Themes 4K	ojpephllplnlpalpnkhnmbdmnjpfaehai	1.0	N	140
TicketNow	okdjbiaomkeiepljpabgihakjpcmmml	1.34	Y	325
Goomarks (Browser for Google Bookmarks)	oknllbfkmapccinochelgmcdjhlgfnik	1.0.3.2	Y	155
Upload photo to Instagram™	oknpgmaeedlbdichgaghebhiknmghffa	1.4.8	Y	260
PDF Merger	okodhidcjjllmkgnjpmkcecajbbgdbbllf	1.0	Y	165

Name	Extension ID	Version	CRXCavator Report	Risk Score
BBC iPlayer subscription manager	omconeoffommngjnkafldpjmndbmopelh	0.0.2	Y	190
ResultsExpander	onalkmkmknbndikdbddsgoclhnknjifalk	1.0.0	Y	115
Sign On Express Extension	onoakpekjdnamofckefambjgdcdbbnddb	6.8	Y	290
Baby Elsa Puppy Surgery	ooblennhepfkjlpmofnlaajpgjkgahf	0.0.1	Y	130
Photomath	opmdomafhdphidaghneldjkdimpccopj	1.0.3	Y	150
Notifications for Instagram	opnbnmdkdfhlhjclaoiifmheknpcal	8.1.15	Y	295
Adresist adblocker	padockkdmabpildideccompngnkoifdim	3.3.0	Y	280
Rung Rinh: Free Cash Back Shopping Assistant	paenbjckelooppieetecheehogoha	1.6.00.6	Y	335
Christians Wallpaper HD Custom New Tab	pbghjoeefklijmmbmoomgmihdmgmccak	7.0.0.0	Y	135
Dandelion Wallpaper HD Custom New Tab	pbjgffkfljbmamdbpnkfmgomnapjahh	6.0.0.0	Y	135
Video Downloader Proficient	pboidikkgoedgccndmgcalcpofdoia	1.6.0	Y	155
Apex Legends Mirage New Tab	pcmnbcicpmniffiehgkglmdimkehcibk	0.0.2	Y	135
Ambulance Mission 3D	pepojgkomlaigcalkfoidcgbencado	0.0.3	Y	125
Make Facebook™ The Classic Design Again	peftoififeahhoomlbnegeneppnmleokp	2020.10.26	Y	160
Doodle Trump	penbafigcbhjchdcapmohnmcapjajicc	50.10.2020.1	Y	120
Alarm Clock	pikejdabngkioijkbfeejoeikpcpcckmn	1.3	Y	155
Screenshotting - Full Page Capture	piljhklcpelbkpfjgkfajeghhghcnhjc	7.0.2	Y	310
Pixel Gun 3D New Tab & Wallpapers Collection	pjogimemedhnnkghgcbpmiicjaodjeme	0.0.5	Y	145
Netflix	plinjajenepheahigdamlgaheieibep	0.0.1	Y	130
JioTV	plmacanepmojfhghgobobokjjiojamenpi	0.1	Y	140

Name	Extension ID	Version	CRXCavator Report	Risk Score
NoMoreAds- Privacy & Ad Blocker	p\phcnkkekhlhcagdpbpfmngommepdj	1.0.4	Y	330
411Screen	pmcklmglhkgldghapidekehecolnfi	1.0.1	Y	270
Among Us Mod Menu and Hack	pmidcdminpbjjeiekieicamjibbckflh	1.0	Y	130
Screenshot Tool	pnggpmgkgoilgheafmlfhincbeajelej	1.2	Y	205
8tracks controller	pofoeceigcjlglkfpkeicgmpafmjihmh	0.1.0	Y	280
Website Editor.	polnjpnlfdbajjabjnojopfoddneobm	1.0	Y	205
Yup7 Games - Free Fun Unblocked Games	pommdiihkhbjpggindbngkbceljninbdkj	1.1	Y	145
Watcher for Gmail™	ponjigdcpcfepepjcapljjkaaecnhgjm	2.30	Y	185
iDtrust Signer	ppnhjogokaemcoppokpcojdfpppkndk	0.1.2	Y	145
Plants vs Zombies 2 New Tab Theme	ppojkeheaooklhdakchcgckmjcdehed	0.0.5	Y	135

Table A.2: Dataset of 250 Malicious Extensions

Appendix B

Code Snippets

This appendix presents some code snippets mentioned in the document.

B.1 Original ID and Public Key Extraction

```
1 DIR=../malicious/
2 if [ "$(ls -A $DIR)" ]
3 then
4   for file in ../malicious/*
5   do
6     filename=$(basename -s .crx "$file")
7     crxID=$(crx3-info < "$file" | grep 'id ' | awk '{print $2}')
8
9     if [ "$filename" != "$crxID" ]; then
10      echo "$crxID=$filename" >> ../src/maliciousMap.txt
11    fi
12
13    ../crx2zip "$file"
14    crxMainIDX=$(crx3-info < "$file" | grep 'main_idx=' | cut -f2 -d "=")
15
16    crx3-info rsa $crxMainIDX < "$file" > "../extensions/$crxID.txt"
17  done
18 fi
```

Listing B.1: Extract Original ID and Public Key from a Malicious Extension

B.2 Converting CRX2/ZIP to CRX3

```
1 # unzip all the extensions
2 for i in *.zip; do unzip "$i" -d "${i%.zip}"; done
3
4 #pack all the extensions
5 for i in */; do google-chrome --pack-extension="$i"; done
```

Listing B.2: Bash Commands to convert CRX2/ZIP into CRX3

Appendix C

Dataset of Websites

This appendix presents the full dataset of websites used in our dynamic approach automation.

Top 100 Most Visited Websites by Search Traffic (2021)			
en.wikipedia.org	yahoo.com	play.google.com	weather.com
youtube.com	indeed.com	cnet.com	ups.com
amazon.com	target.com	roblox.com	verizon.com
facebook.com	microsoft.com	usnews.com	forbes.com
twitter.com	nytimes.com	zillow.com	wowhead.com
fandom.com	mayoclinic.org	businessinsider.com	expedia.com
pinterest.com	espn.com	bulbagarden.net	urbandictionary.com
imdb.com	usps.com	paypal.com	foodnetwork.com
reddit.com	quizlet.com	finance.yahoo.com	nbcnews.com
yelp.com	gamepedia.com	genius.com	macys.com
instagram.com	lowes.com	usatoday.com	apartments.com
ebay.com	irs.gov	realtor.com	ign.com
walmart.com	nih.gov	medicalnewstoday.com	capitalone.com
craigslist.org	merriam-webster.com	fedex.com	costco.com
healthline.com	steampowered.com	bankofamerica.com	theguardian.com
tripadvisor.com	mapquest.com	washingtonpost.com	cnn.com
linkedin.com	foxnews.com	investopedia.com	glassdoor.com
webmd.com	allrecipes.com	speedtest.net	yellowpages.com
netflix.com	quora.com	spotify.com	att.com
apple.com	aol.com	cdc.gov	bbc.com
homedepot.com	britannica.com	chase.com	khanacademy.org
mail.yahoo.com	live.com	hulu.com	ny.gov
cnn.com	bestbuy.com	xfinity.com	twitch.tv
etsy.com	rottentomatoes.com	msn.com	adobe.com
google.com	ca.gov	dictionary.com	cbssports.com

Table C.1: Top 100 Most Visited Websites by Search Traffic (2021) [27]

References

- [1] A. Aggarwal, B. Viswanath, L. Zhang, S. Kumar, A. Shah, and P. Kumaraguru. I spy with my little eye: Analysis and detection of spying browser extensions. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 47–61, 2018.
- [2] Anupama Aggarwal, Saravana Kumar, Ayush Shah, Bimal Viswanath, Liang Zhang, and Ponnurangam Kumaraguru. Spying browser extensions: Analysis and detection. 2016.
- [3] Alex Gromnitsky. Bloatware comes when nobody’s lookin’. <http://gromnitsky.blogspot.com/2017/03/bloatware-comes-when-nobodys-lookin.html>.
- [4] Alex Gromnitsky. Crx3. <http://gromnitsky.blogspot.com/2019/04/crx3.html>.
- [5] Sruthi Bandhakavi, Nandit Tiku, Wyatt Pittman, Samuel King, P. Madhusudan, and Marianne Winslett. Vex: Vetting browser extensions for security vulnerabilities. *Commun. ACM*, 54:91–99, 2011.
- [6] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting Browsers from Extension Vulnerabilities. *Ndss*, 147(June):1315–1329, 2010.
- [7] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: A fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International Conference on World Wide Web, WWW ’11*, page 197–206, New York, NY, USA, 2011. Association for Computing Machinery.
- [8] Catalin Cimpanu. Google to no longer allow chrome extensions that use obfuscated code. <https://www.zdnet.com/article/google-to-no-longer-allow-chrome-extensions-that-use-obfuscated-code/> Last accessed on 2021-08-02.
- [9] YoungHan Choi, TaeGhyoon Kim, SeokJin Choi, and CheolWon Lee. Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis. pages 160–172, 2009.
- [10] Chrome Developers. Content scripts. https://developer.chrome.com/docs/extensions/mv3/content_scripts/.
- [11] Chrome Developers. Content security policy. <https://developer.chrome.com/docs/apps/contentSecurityPolicy/>.
- [12] Chrome Developers. Manage events with service workers. https://developer.chrome.com/docs/extensions/mv3/service_workers/.

- [13] Chrome Developers. Migrating to manifest v3. <https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/>.
- [14] Chrome Developers. What are extensions? <https://developer.chrome.com/docs/extensions/mv3/overview/>.
- [15] Cisco. Cisco completes acquisition of duo security. <https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1945408>.
- [16] Dan Goodin. Malicious chrome extensions infect 100,000-plus users, again. <https://arstechnica.com/information-technology/2018/05/malicious-chrome-extensions-infect-more-than-100000-users-again/> Last accessed on 2021-02-10.
- [17] Demo. Browsingfog's demo. <https://vimeo.com/188468225> Last accessed on 2021-01-30.
- [18] M. Dhawan and V. Ganapathy. Analyzing information flow in javascript-based browser extensions. In *2009 Annual Computer Security Applications Conference*, pages 382–391, 2009.
- [19] Google. Declare permissions. https://developer.chrome.com/docs/extensions/mv2/declare_permissions/ Last accessed on 2021-01-31.
- [20] Google. Declare permissions and warn users. https://developer.chrome.com/docs/extensions/mv2/permission_warnings/ Last accessed on 2021-01-30.
- [21] Google. Google translate. <https://chrome.google.com/webstore/detail/google-translate/aapbdbdomjkkjkaonfhkkikfgjllcleb>.
- [22] Imperva. Malvertising. <https://www.imperva.com/learn/application-security/malvertising/> Last accessed on 2021-01-25.
- [23] Imperva. Session hijacking. <https://www.imperva.com/learn/application-security/session-hijacking/> Last accessed on 2021-02-10.
- [24] Jake Frankenfield. Supply chain attack. <https://www.investopedia.com/terms/s/supply-chain-attack.asp> Last accessed on 2021-01-30.
- [25] Jesse Ruderman. Same-origin policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.
- [26] J. Joseph and S. Bhadauria. Cookie based protocol to defend malicious browser extensions. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6, 2019.
- [27] Joshua Hardwick. Top 100 most visited websites by search traffic (2021). <https://ahrefs.com/blog/most-visited-websites/>.
- [28] jQuery. jquery. <https://jquery.com> Last accessed on 2021-01-30.
- [29] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. 2014.

- [30] Kaspersky. What are cookies? <https://www.kaspersky.com/resource-center/definitions/cookies> Last accessed on 2021-01-30.
- [31] Krebs on Security. Using fake reviews to find dangerous extensions. <https://krebsonsecurity.com/2021/05/using-fake-reviews-to-find-dangerous-extensions/>.
- [32] MDN Web Docs. Using http cookies. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies> Last accessed on 2021-02-11.
- [33] Joseph Medley Mike West. Content security policy. <https://developers.google.com/web/fundamentals/security/csp>.
- [34] C. Obimbo, Y. Zhou, and R. Nguyen. Analysis of vulnerabilities of web browser extensions. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 116–119, 2018.
- [35] Pablo Picazo-Sanchez, Juan Tapiador, and Gerardo Schneider. After you, please: browser extensions order attacks and countermeasures. *International Journal of Information Security*, 19(6):623–638, 2020.
- [36] Pieter Arntz. Chrome extensions that lie about their permissions. <https://blog.malwarebytes.com/puppum/2020/08/chrome-extensions-that-lie-about-their-permissions/> Last accessed on 2021-02-10.
- [37] Xiaokui Shu, Ke Tian, Andrew Ciambrone, and Danfeng Yao. Breaking the target: An analysis of target data breach and lessons learned. *ArXiv*, abs/1701.04940, 2017.
- [38] O. Starov and N. Nikiforakis. Xhound: Quantifying the fingerprintability of browser extensions. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 941–956, 2017.
- [39] Oleksii Starov and Nick Nikiforakis. Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. pages 1481–1490, 2017.
- [40] Symantec. Internet security threat report. *Network Security*, 21(2), 2016.
- [41] Thomas Claburn. Fake crypto-wallet extensions appear in chrome web store once again, siphoning off victims’ passwords. https://www.theregister.com/2020/05/06/chrome_malicious_extensions/.
- [42] Zero Day Initiative TippingPoint. Mozilla foundation security advisory 2006-45 - javascript navigator object vulnerability. <https://www.mozilla.org/en-US/security/advisories/mfsa2006-45/>.
- [43] G. Varshney, S. Bagade, and S. Sinha. Malicious browser extensions: A growing threat: A case study on google chrome: Ongoing work in progress. In *2018 International Conference on Information Networking (ICOIN)*, pages 188–193, 2018.
- [44] Yao Wang, Wandong Cai, Pin Lyu, and Wei Shao. A Combined Static and Dynamic Analysis Approach to Detect Malicious Browser Extensions. *Security and Communication Networks*, 2018:7087239, 2018.

- [45] Xinyu Xing, Wei Meng, Byoungyoung Lee, Udi Weinsberg, Anmol Sheth, Roberto Perdisci, and Wenke Lee. Understanding malvertising through ad-injecting browser extensions. pages 1286–1295, 2015.
- [46] Tommy Dong Yuanjing Guo. Cryptojacking extensions found on google chrome web store. <https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/cryptojacking-extensions-chrome-web-store>.
- [47] Zbigniew Banach. What is session hijacking: Your quick guide to session hijacking attacks. <https://www.netsparker.com/blog/web-security/session-hijacking/> Last accessed on 2021-01-30.
- [48] Zeljka Zorz. Compromised ad company serves magecart skimming code to hundreds of websites. <https://www.helpnetsecurity.com/2019/01/17/magecart-supply-chain-attack/> Last accessed on 2021-01-30.
- [49] B. Zhao and P. Liu. Private browsing mode not really that private: Dealing with privacy breach caused by browser extensions. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 184–195, 2015.