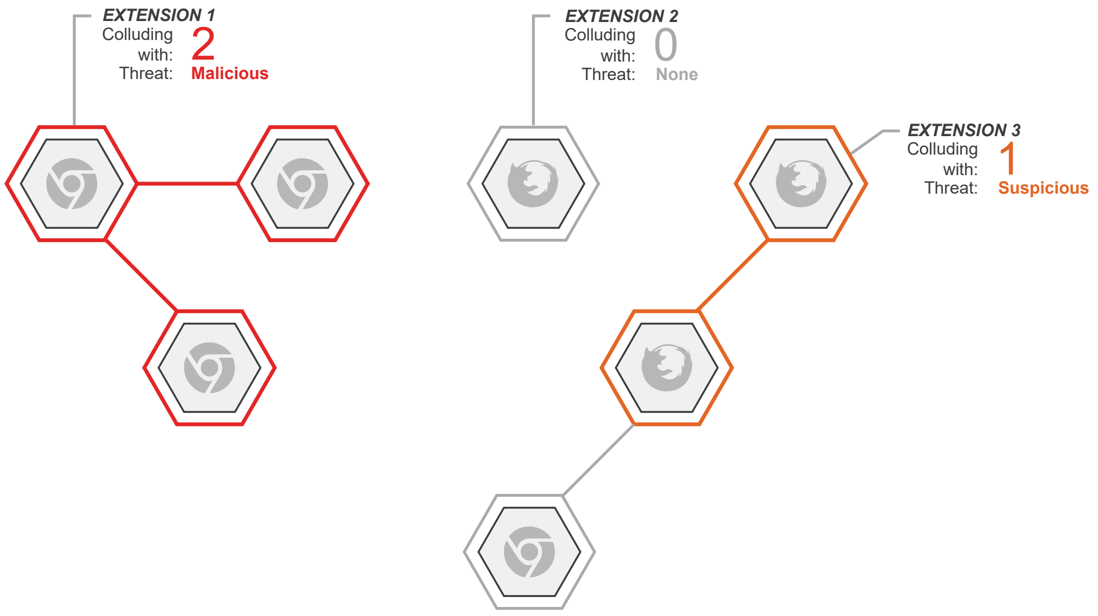
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Collusion Attacks on Browser Extensions

Revealing hidden extensions colluding against the user

Master thesis in Computer Systems and Networks

DŽENAN BAŽDAREVIĆ
MICHAEL DUBELL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017, DATX05

Collusion Attacks on Browser Extensions

Revealing hidden extensions colluding against the user

Dženan Baždarević, Michael Dubell

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Collusion Attacks on Browser Extensions
Revealing hidden extensions colluding against the user
DŽENAN BAŽDAREVIĆ
MICHAEL DUBELL

Supervisor: Pablo Picazo-Sanchez, Department of Computer Science and Engineering, University of Gothenburg
Examiner: Andreas Abel, Department of Computer Science and Engineering, University of Gothenburg

Master's Thesis 2017, DATX05
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualisation of set of extensions, some of them are colluding with each other using malicious or suspicious domains or IP addresses.

Gothenburg, Sweden 2017

Collusion Attacks on Browser Extensions
Revealing hidden extensions colluding against the user

DŽENAN BAŽDAREVIĆ, MICHAEL DUBELL

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Browser extensions have been created to extend and enhance web browsers in order to improve the user experience. Because of this, browser extensions can access a range of different resources that pose a great privacy risk for users. These sensitive resources include users' browser history, passwords and banking information. Therefore browser extensions have become a great source of interest for those with malicious intent. In order to obscure the intent behind a browser extension, a set of extensions can be created that when analysed individually does not raise any suspicion. However, by analysing the entire set of extensions, a relationship between each extension can be revealed. Namely, each extension is extracting user information under different sets of permissions, and relaying this data to a common external server. Such extensions are said to be colluding, and possibly performing a collusion attack. This form of attack is the focus of this research paper.

We propose a method for downloading and performing static analysis of the collected browser extensions. The static analysis is based on regular expressions and defined to match and extract domain names and IP addresses from the downloaded browser extensions. In order to discover domains or IP addresses that are malicious, Recorded Future's threat intelligence is used to provide classification and information behind each classification. Recorded Future collects data from technical sources, open sources and closed sources. By combining their machine learning and natural language processing, Recorded Future can identify, classify and predict events.

In this work, over 250,000 Mozilla Firefox and Google Chrome extensions have been analysed by our proposed method and as a result, 1037 browser extensions have been found to be possibly colluding. Recorded Future classified 131 domains as Malicious.

Keywords: Web Browser Extensions, Collusion Attacks, Web Security, Static Analysis.

Acknowledgements

This project was conducted as a master thesis project for Chalmers University of Technology under the spring of 2017. The scope of this course is 30 credits (hp). The master thesis was conducted at Department of Computer Science and Engineering at Chalmers University of Technology, by students of Computer Science and Engineering 300 hp and MPCSN, 120 hp.

Foremost, we would like to express our sincere gratitude to our supervisor Pablo Picazo-Sanchez for his support of our MSc. research, his motivation, guidance and enthusiasm.

Besides our supervisor, we would like to thank Steven Van Acker and Andreas Abel for their inspiration and insightful comments.

Dženan Baždarević and Michael Dubell, Gothenburg, June 2017

Contents

List of Figures

List of Tables

1 Introduction

For many users, web browsers are the gate to the Internet. We use web browsers to access services such as social media, e-mail or bank accounts. To increase the browsing experience and to extend the functionality of the browser, almost all major web browser vendors allow browser extensions to be installed. A browser extension is a component that can extend the browser's functionality. Normally, extensions are not provided by the browser vendors, but by third parties. When an extension is installed, it has access to some resources, such as open tabs, bookmarks and cookies. These privileges are specified in a file called `manifest.json` [1, 2]. Some extensions may even share the same privileges as the browser itself. A few examples of privileges are allowing modifications of the HTTP headers or modifying the web page by changing the Document Object Model (*DOM*). Initially, the goal of the browser extensions was to increase the browser experience for users, however, new security and privacy issues have arisen since extensions are overprivileged and gain access to information that they do not need [3]. When users browse the Internet, sensitive information such as bank information and passwords are sent through the browser. In other words the information is exposed to the browser and its extensions. This type of information may be of interest to the get hold of. As a result, browser extensions have been created to serve a malicious purpose [4]. These kind of browser extensions can be created to, for instance, provide certain advertisements to the user, steal data and track user's behaviour. Often, users are not aware that their installed extensions can execute malicious code [5].
There exist several different ways for browser extensions to attack its users, this thesis will focus on attacks performed by multiple browser extensions that are colluding with external servers.

1.1 Purpose

The purpose of the project is to research whether there exist browser extensions that are colluding with other extensions when performing malicious activity, meaning, if they cooperate in some way by exchanging private information about users to an external server. This form of attack is called *Collusion attack*.

1.2 Problem definition

Several browser extension systems provide an interface that allows an extension to collude with other extensions installed on the system, in order to share objects

[4, 6]. An example of this might be the following. Consider two extensions A and B. Extension A has the privilege to collect anything from a web page while extension B can use the network channel in order to communicate. If we analyse them separately we can say that they are considered to be safe to use. However, if extension B acquires the information collected by A, the extensions may be colluding with each other. In this research we will focus on extensions that communicate with the same external servers, to reveal collusion attacks. For instance, extensions A and B may use the same external server(s) to store collected information or to exchange information between each other.

We will analyse all extensions in the Google Chrome Web Store and in the Mozilla Firefox Add-ons in order to determine whether there exist extensions that perform collusion attacks. At the time of this writing there exist 18,461 extensions in Firefox Add-ons and 135,909 in Chrome Web Store with almost 229,000 versions in total. The reason why extensions for Google Chrome Store and Mozilla Firefox Add-ons were chosen was because they are the most popular web browsers, as can be seen in Figure 1.1. Also, extensions from Google Chrome Store might also be installed on other web browsers such as Opera since it uses the same web engine as Google Chrome. Furthermore, Mozilla is working on to provide support for extensions from Google Chrome Store in Mozilla Firefox [7].

Web browser	**Google Chrome**	**Internet Explorer**	**Mozilla Firefox**	**Safari**
NetMarketShare	56.43%	20.84%	12.22%	3.47%
StatCounter	62.66%	9.89%	14.95%	5.10%

Table 1.1: Usage of web browsers in December, 2016 [8, 9].

We intend to detect collusion attacks by finding extensions that communicate with the same servers. We will use *Recorded Future's* Application Interface (API) to determine if the external server(s) used by the extensions are malicious. Recorded Future is described in more detail in section 2.4.

In the continuation of this paper, we will refer domain names and IP addresses as *addresses*, unless specified otherwise.

1.3 Methodology

Due to the amount of information that needs to be checked, in the beginning we focus on automatic detection of collusion attacks for the 50 most downloaded extensions. The first step that was performed was static analysis of all files for all extensions and storing the results in a database. The database contains the extensions' name, version, browser, author and domains. Once we deploy and check whether the analysis has the expected behaviour, the process is automated and extended to include all extensions.

Finally, with the data we collected in the previous step, we used Recorded Future's API to investigate whether the servers used by the extensions are somehow linked to other well-known attacks. Recorded Future is a company that focuses on threat intelligence using machine learning and natural language processing. Recorded Future collects data from a wide range of sources which allows them to predict certain events [10] using their AI algorithms. With the help of their API which gives us access to their intelligence platform, we can detect domains classified as threats.

1.4 Limitations

A major part of the analysis is to identify whether there exist extensions that use the same external server for communication. The mere existence of a set of extensions that share the same domains or IP addresses does not mean that the extensions are necessarily colluding. Extensions may be using popular Internet services for their communication, however the possibility that such services are being used by malicious extensions is still present.

Some limitations regarding the process for extracting domains and IP addresses from the extensions exist. There are techniques that malicious extensions can use in order to hide their true intention. Domains may be computed from a function which means that it is impossible to detect and extract the domain statically by regular expression matching. Also the malicious domain may be absent from the code completely and retrieved only by querying a ”*good*” server. Another technique that may be used is obfuscation. Obfuscated code is used to hide or to obscure the true intent of a piece of code. Obfuscated JavaScript code can be created with custom algorithms or with the help of popular tools available on the Internet.

With these limitations in mind, we cannot claim that an extension is safe or not. There are many other low-level attacks that colluding extensions can make use of. These attacks are not covered in this research. We can however, produce a risk value that can give an indication to whether an extension is performing any malicious activity, similar to what Recorded Future provides.

1.5 Related Work

Saini et al. [4] point out some weaknesses regarding the extension system for Mozilla Firefox when handling JavaScript objects. They show that two legitimate extensions can cooperate in order to achieve malicious goals. The current research for detecting malicious behaviour in extensions focuses primarily on single extension as the main source for attack. Because of this, they managed to perform attacks by using multiple extensions that cooperate. The result was that the attacks were undetectable by existing popular client side methods used to detect malicious behaviour in extensions.

Similar research was performed by Bauer et al. [6], they could demonstrate that extensions can become less suspicious by spreading their required permissions over several extensions. Each extension performs its specific task and communicates with the other extensions over covert channels. These covert channels include the number of open tabs, browser CPU utilisation, browser history or even the clipboard. Extensions can send private information such as passwords or banking information collected by one extension, to another extension with different permissions, that will send the data to third-party servers.

Both of the previous mentioned works focuses on Collusion attacks that are performed inside the web browser, this thesis however aims to detect a Collusion attack where external servers act as a middle man. Also, Saini et al. [4] and Bauer et al. [6] only prove that Collusion attacks can in principle be performed but they have not found extensions that are actually performing this type of attack.

1.6 Thesis Contributions

We design and implement a platform to download, extract and analyse browser extensions as well as visualising the results. The platform downloads browser extensions from Google Chrome Store and Mozilla Firefox Add-on Store. The collected extensions are then analysed by the static analysis process which extracts domains and IP addresses that might be used for external communication. The extracted addresses are further analysed by Recorded Future's threat intelligence to identify malicious domains and IP addresses.

The platform has successfully downloaded and analysed 279,250 browser extensions while 505,322 addresses were extracted. As a result, 1,037 browser extensions were found to be potentially colluding.

1.7 Disposition of the Thesis

The thesis is organised as follows: Chapter 2 provides background knowledge to the topic of this work, for instance how a browser extension is built and the type of functionality it can provide. In Chapter 3, a description of the overall system design is given. Chapter 4 describes how the system was implemented, its different components and the obstacles found during this process. Chapter 5 focuses on evaluating the static analysis process and proving the accuracy of this process. In Chapter 6, our findings and results are presented. The final chapters, Chapter 7 and 8, provide a discussion and a conclusion, respectively.

2 Background

To understand the privacy implications of extensions and their capabilities, some background knowledge regarding the structure of browser extensions is required. This section describes the basics of browser extensions, how they are built, how they can interact with the user and finally how Recorded Future will be used to identify and classify addresses as malicious.

2.1 Browser extensions

Browser extensions are small pieces of programs that can be installed in the user's web browser. The purpose of extensions is to enhance the browser experience for users as well as personalise their browser. Extensions can modify or extend the default browser functionality, e.g., an extension may be installed to block advertisements on websites [11], add foreign language dictionaries or change the visual appearance of the browser itself. Before installing extensions, users must usually accept the permissions required by the specific extension. In order for the extension to interact with websites and customise the browser, the extension needs to specify the permissions needed. An extension can specify a range of permissions without necessarily needing them to function. By requiring unnecessary permissions, extensions pose a great privacy risk for users. This will be discussed further in Section 2.2.

Extensions have the possibility to communicate with external servers as well as with internal browser extensions. Consider an extension that blocks advertisements from websites visited by the user. Assuming the extension uses a blacklist with a set of advertisement companies, in order for the extension to update this list it would need to contact an external server for an updated version. As mentioned, extensions can communicate with other installed extensions by using an API provided by the browser. One extension could potentially collect information under a specific context, e.g., save the user's login credentials for a specific site and send the information to a second extension that may upload the data to an external server. Communication between extensions and servers are needed for advanced functionality that enhances the user's browser experience, however it can also be used for malicious intent which will be discussed in Section 2.3.

In this paper, only the extensions provided by Mozilla Firefox and Google Chrome Store will be examined. The difference between the two browsers in terms of extension structure is minimal. Both web browsers provide an API for the extension to utilise and information about the extension such as name, creator and required permissions can be found in the extension's manifest file. Both Firefox and Chrome require their extensions to be developed in JavaScript, HTML, CSS and can be downloaded from the respective extension store.

2.2 Privacy

Extensions can require a set of permissions in order to perform certain functions. These functions include reading and writing into web pages as well reading users' web history. In the case of Google Chrome extensions, developers can require access to all websites which can bee seen in listing 2.1.

```
{
    "name": "My extension",
    ...
    "content_scripts": [
    {
      "Matches": ["*://*.*"],
      "css": ["mystyles.css"],
      "js": ["myscript.js"]
    }
    ],
    ...
}
```

Listing 2.1: Chomre Content scripts, specified in the manifest.json

The above code snippet shows how a developer can specify a content script that will be injected into every website visited by the user, as specified by the wildcard *(*)* in the protocol, host and top level domain name. This allows malicious browser extensions to intercept and modify web content in-transit for any website.

Other potential privacy invasive sources which Google Chrome extensions can access can be seen in table 2.1. A complete list can be found in [12].

Sources	**Impact**
Bookmarks	Users' bookmarks
Clipboard	Read and write to users' clipboard
Cookies	Websites' cookies
History	A complete history of users' Internet browsing

Figure 2.1: Potential privacy invasive sources.

Once an extension have acquired the necessary permissions, the extension can at any point in time start behaving maliciously, e.g., stealing private credentials such as banking information or injecting advertisements in every web request as was seen in 2016. A popular Google Chrome extension called Live HTTP Headers, used for inspecting web requests, suddenly begun injecting unwanted advertisements into every page for every user [13]. The extension was eventually removed from the Google Chrome Store. Other examples of browser extensions spying and injecting code into users' browsers was shown by [14] in which a JavaScript library was used to track internet users as well as collect their browser history. Within the top 7000 Google Chrome and Mozilla Firefox extensions, the library was used 42 times with a total of eight million installations.

Extensions that require many permissions in order to function may arouse unwanted suspicion, therefore developers with malicious intent can spread out the required permissions over multiple extensions. By doing this, multiple malicious installed extensions can collude with each other by transferring data collected with different permissions. This is further discussed in Section 2.3.

2.3 Collusion attacks

A collusion attack can be defined as two or more components that cooperate with each other in order to achieve some malicious goal [4]. An example of this regarding browser extensions may be: consider two web extensions A and B installed on the same browser. Extension A's functionality is to extract information from any web page the user visits. Extension B can access the network channel and broadcast and receive information from an external server. If the two extensions are analysed individually, they may not arouse any suspicion. However, consider a scenario where the two extensions can exchange information with each other. This might lead to a problem since extension A can access sensitive information. It can for example extract passwords or banking information from the user's web page. If extension B can acquire this information from extension A and send it to some external server, both extensions are said to be preforming a collusion attack. An attack that is performed with more than one extension is harder to detect and can pass unnoticed [15]. Normally, security mechanisms that analyse malicious extensions statically or dynamically only focus on a single extension which means that they can not be used to detect colluding extensions. As previously mentioned, if an extension requests too many privileges it may draw undesired attention to it. However by splitting up the privileges among several extensions, the individual extension may look less suspicious and thus harder to detect.

There exist multiple ways how extensions can cooperate when performing collusion attacks. Web browsers have a feature rich API that extensions can utilise in order to make network requests or access the local file system. In Google Chrome every extension can send direct messages to other installed extensions. This feature is available by default without needing any extra privileges [6]. This works by letting extension B create a listener which is an event handler that waits for messages sent to it. When extension A has collected information from the browser, it can send the information directly to the event handler created by B. The outcome is that extension B have acquired information that it does not have access to with its current privileges. This can be visualised in figure 2.2.

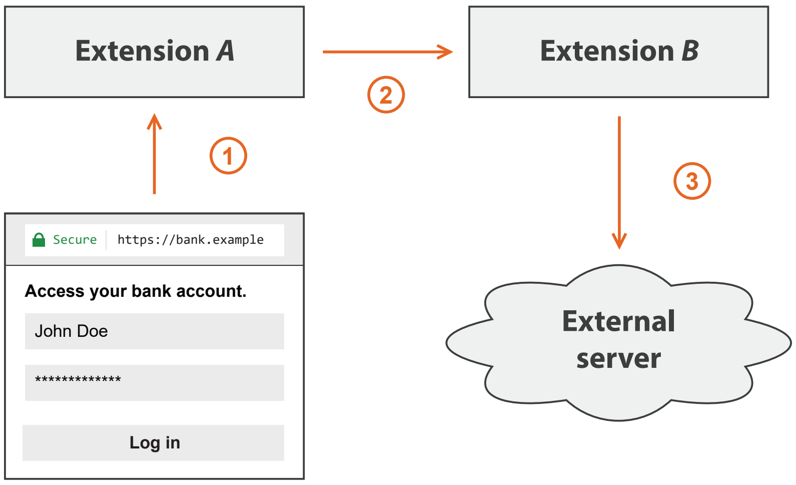


Figure 2.2: A scenario showing a collusion attack using message passing between two extensions. In step 1 the user has accessed a page where she enters her log in details. Since extension A can read the page it can extract the information, in this case the log in details. In the second step, extension A passes the information to extension B through the listener. Finally in the third step extension B can send the information to an external server.

A different method is to share the information via shared states, for instance the history or bookmarks. In order to access any shared states, the colluding extensions need of course to have the necessary privileges.

Extensions can also use external servers to exchange information. Consider a scenario where two extensions A and B are colluding. Both extensions gather different sensitive information independently from each other and send it to an external server. An example of this can be seen in Figure 2.3.

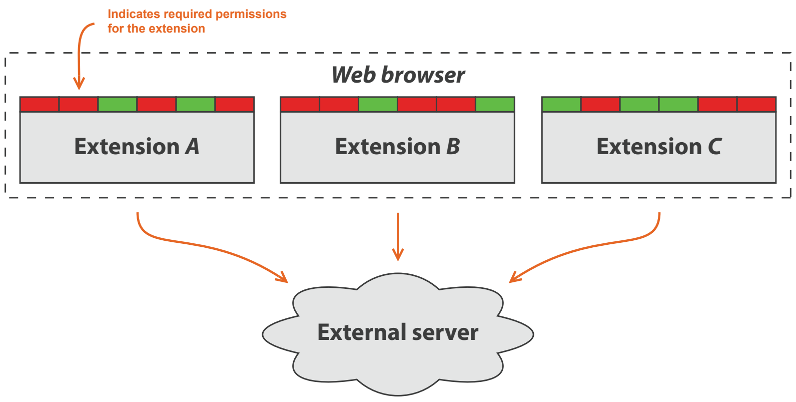


Figure 2.3: A scenario where three extensions with different privileges perform a collusion attack. All three extensions communicates with the same external server.

Extensions created by the same developer may cooperate more easily with each other. A study [6] shows that 14.9% of the developers in the Google Chrome Store publish multiple extensions. 70% of these developers have created extensions that requires different permissions. The developer that required the most permissions, i.e., the sum of all permissions from its extensions had published 6 extensions. The amount of required privileges from each extension varied between 0 and 11.

2.4 Recorded Future

Recorded Future is a company that focuses on threat intelligence using machine learning and natural language processing. Recorded Future collects data from a wide range of technical, open, and closed sources which allows them to deliver threat intelligence in real time. Figure 2.4 shows different types of sources Recorded Future collects information from.

Technical	**Open**	**Closed**
Malware Infrastructure and Files	Social Media	Dark Web
Vulnerabilities	Security Reporting	Special Access
Phishing and Spam	Black Hat Sites	Actor Engagement
Abuse and Infections		Criminal Forums
Web Infrastructure		
Vulnerabilities		

Figure 2.4: Examples of different sources Recorded Future collects information from.

Recorded Future's machine learning technology allows them to predict certain events [10] such as if an IP will become malicious in the future, even though such activity has not been observed in the past. A visual presentation of Recorded Future's intelligence platform can been seen in Figure 2.5.

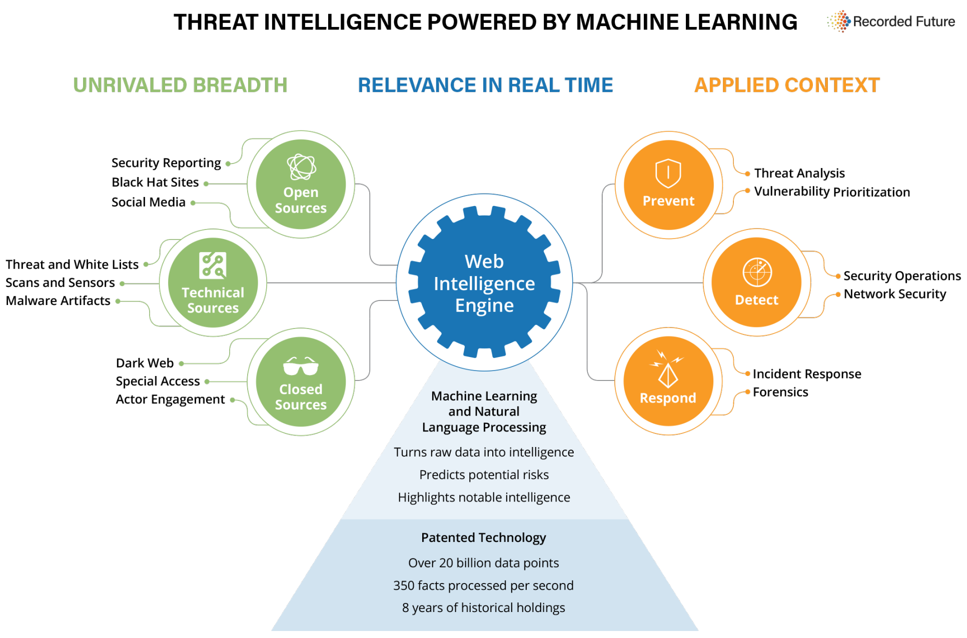


Figure 2.5: Recorded Future's intelligence platform visualised.

By using Recorded Future's API that provides threat intelligence, we can detect addresses classified as threats which can give an indication on which extensions that may be colluding. Figure 2.2 shows the result from Recorded Future when querying for *google.com*.

```
{
    "data": {
        "risk": {
            "criticalityLabel":"None",
            "rules":0,
            "evidenceDetails":[],
            "riskSummary":"No Risk Rules are currently observed.",
            "criticality":0,
            "score":0.0
        }
    }
}
```

Listing 2.2: Risk analysis from Recorded Future when quering domain `google.com`.

The important fields for this work which can be found in figure 2.2 are `evidenceDetails` and `score`. The field `evidenceDetails` provides the data that explains why a given address has been classified as such. The `score` field gives an overall score for the address between 0 - 100, where 100 is very malicious.

3

System Design

In order to download all the extensions provided by the Mozilla Firefox and Google Chrome extension store, perform static analysis and visualising data, we built a platform that includes the necessary components needed to achieve this goal. This section presents the overall system design and describes its individual components as well as how the results of the analysis are visualised.

The system has been built with the popular programming language Python [16] *(version 3.6)*. We chose Python because of its ease of development and its large number of third-party libraries. Our system was run on a virtual machine with the specifications shown in figure 3.1.

Operating System	CentOS 7
Memory	4 GB
Disk	HDD 930GB
Processor	Intel Xeon E3-1245 Quad-Core 3.4 Ghz
Internet bandwidth	100mbit/s download, 10mbit/s upload

Table 3.1: Virtual Machine specifications

The virtual machine was hosted on a computer running VMware vSphere Hypervisor 6.5 [17].

The system design can be described in three different phases, the first phase involves downloading all the extensions (and their previous versions) provided by the Mozilla Firefox and Google Chrome extension store. This phase consists of two separate download components, one that downloads extensions from Mozilla and one that downloads extensions from Google. These separate components are needed because Mozilla does not provide an API or a way to enumerate all their extensions easily. Therefore it was necessary to build a program around Mozilla's Add-on store in order to correctly read and parse extensions provided by the store. Google however provides a sitemap that makes it possible to easily enumerate extensions provided by their store.

The download components are built using the Python multiprocessing [18] library in order to achieve parallelism. This is preferable over using a single process to run the software because of time. By using multiple processes the program can complete more work in less time. Both components use a first in, first out *FIFO* queue to store work.

The second phase is split into two parts, the first part performs static analysis of the downloaded extensions to extract addresses found in the extension's code. The result of the static analysis is saved into a MongoDB server [19]. The second part collects the extracted addresses from the database and sends them to Recorded Future for threat analysis. Recorded Future performs their threat analysis using machine learning to classify each address as well as to provide evidence for their classification. The results are returned to our platform and saved to the database.

The third and final phase involves visualising the data stored in the database. The visualisation is provided by a custom made web interface built on Flask [20] that displays the data with numbers and graphs. By using the web interface it is very easy to browse the results to find possible colluding extensions, domains/extensions with highest threat score or find all domains used by any extension.

The entire flow of the system can be visualised in Figure 3.1.

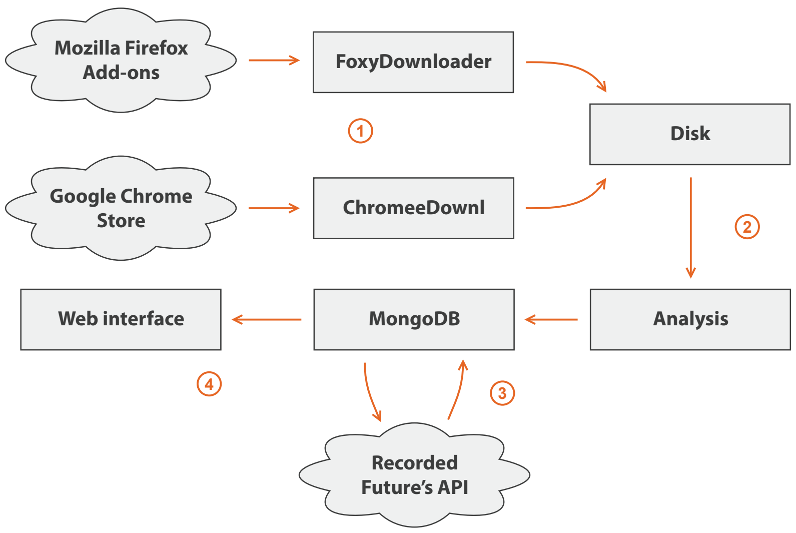


Figure 3.1: Illustrates an overview of the system design. In step one, we execute scripts that downloads extensions from Mozilla Firefox Add-ons and Google Chrome Store. In step two, all extensions found on disk are analysed by our analysis process. In the third step all extracted domains are sent to Recorded Future's API for threat analysis. In the fourth and final step, the collected data is presented and visualised in the custom made web interface.

4
Implementation

This section describes how the system is implemented in detail and how each component works.

4.1 Downloading extensions

Two separate programs for downloading extensions were implemented, one used for Mozilla Firefox extensions and the other for Google Chrome extensions. The programs are described in the following subsections.

4.1.1 Mozilla Firefox extensions

To download extensions provided by the Mozilla extension store, we developed a program called *FoxyDownloader*. The implementation design follows a producer-consumer paradigm, where one processor exists with the purpose of generating work while there are multiple consumer processes that consumes work. When the program is executed, the producer will begin by visiting the *Most Popular Extensions* page at the Mozilla extension store [21]. This page will be referred to as the start page for the remainder of this section. The start page contains all extensions ordered by popularity. As of this writing, there exists 941 sub-pages with 20 extensions on each page (the last page contains between 1-20 extensions), a total of 18,804 extensions (not counting each extension's previous versions). In order to extract download links from each extension, FoxyDownloader uses Beautiful Soup [22] to parse the HTML code for each page it visits.

The producer parses an extension by visiting the *previous versions* page, which contains all the released versions of an extension. The link to this page can be found on the start page for each listed extension, this can be seen in listing 4.1.

```
<div data-version-supported="true" class="install featuredaddon
   clickHijack" data-addon="1865" data-icon="https://addons.cdn.
   mozilla.net/user-media/addon_icons/1/1865-32.png?modified
   =1493199227" data-developers="/en-US/firefox/addon/adblock-plus/
   developers" data-versions="/en-US/firefox/addon/1865/versions/"
   data-name="Adblock Plus" data-min="38.0" data-max="*" data-is-
   compatible="true" data-is-compatible-app="true" data-compat-
   overrides="[]">
```

```
    <p class="install-button">
        <a class="button  add installer" data-hash="sha256:2
           de5616ae0ad17fd6345c6f8f8ec5bd8a8db1bdf8d8dad9995857ad2d
            6454306" href="/firefox/downloads/latest/adblock-plus/
           addon-1865-latest.xpi?src=cb-btn-users">
  <b></b>
  <span>Add to Firefox</span>
</a></p></div>
```

Listing 4.1: Finding the previous versions page of an extension.

The page that contains all the versions of an extension can contain multiple subpages depending on how may versions there exist. Each version contains a download link which the producer will find and place into a queue (to generate work) for the consumers to consume. Once the producer has found all the download links for every version of the current extension, it will move to the next extension found in the start page. The producer will sequentially generate download links according to the order on the start page.

Once the producer has begun producing work (download links for extensions), consumers will begin pulling items from the queue and start downloading each extension to disk. When the producer has generated all work it can, it will push the value *None* × *(amount of consumers)* into the queue meaning that once each consumer reaches the end of the queue they will receive the value *None* and terminate. The entire flow of the program can be visualised in figure 4.1.

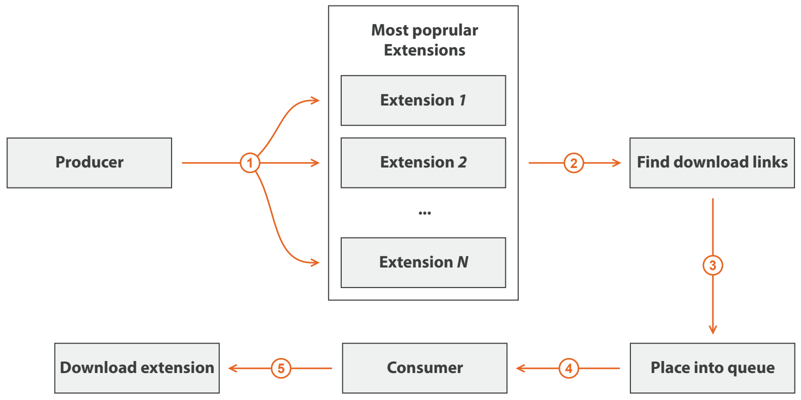


Figure 4.1: Visualisation of FoxyDownloader's execution flow

4.1.2 Google Chrome extensions

In order to download all extensions from Google Chrome store we developed a program to ease this process. The program is called "`ChromeeDownloader.py`" and an example of its execution can be seen in Listing 4.2.

```
(venv) $ python chromeedownloder.py run

  ____ _                                    ____                      _
 / ___| |__  _ __ ___  _ __ ___   ___  ___|  _ \  _____      ___ __ | |
| |   | '_ \| '__/ _ \| '_ ` _ \ / _ \/ _ \ | | |/ _ \ \ /\ / / '_ \| |
| |___| | | | | | (_) | | | | | |  __/  __/ |_| | (_) \ V  V /| | | | |
 \____|_| |_|_|  \___/|_| |_| |_|\___|\___|____/ \___/ \_/\_/ |_| |_|_|


[!] 113548 extension(s) were found in file.
Progress [####################################] 113548/113548 Done!
```

Listing 4.2: Execution of `ChromeeDownloader.py`. 113 548 extensions were found and successfully processed.

The program requires the help of a JSON file in order to download extensions. The JSON file, named `extensions.json`, contains URLs to an extension's detail page on Google Chrome store. The JSON file is generated with help of a script [23] which parses and collects information from the sitemap of Google Chrome store. The sitemap is a XML file and contains URLs to every extension detail page on Google Chrome store.

Once `extensions.json` has been generated, *ChromeeDownloader* can use this file to begin downloading extensions. Since there exist several thousands of extensions and downloading all of them with one process may take time, therefore multiprocessing has been implemented in the program. The number of processes can be chosen by the user and each process is downloading extensions independently from others. Before a process starts downloading an extension, it needs to extract the id and version number, this is done by collecting metadata from the extension's detail page on Google Chrome Store. The version of an extension is needed for identification since there might exist a duplicate extension but with different version number. Once this has been done the program can start downloading the extension. The file type of the extension is CRX Package Format which is a ZIP archive but with an extended header [24]. When the ZIP archive has been extracted, a file is created that contains some information about the extension and the date it was downloaded. This file can help to speed up the process and make the script more efficient if same extension URL is found multiple times. In the beginning when the id and version are extracted, the process reads the special file to check whether the extension is already downloaded, if true it skips this extension and starts with next one. This file gives and indication that the whole extension is present on the disk. In other words, it can differentiate between a fully downloaded extension and a partially downloaded extension caused by some error or by termination of the process by the user.

Google Chrome Store does not store previous versions of an extension, in order to get the previous version they need to be downloaded from somewhere else. To overcome this problem we used a repository archive [25] that contains previous extension versions from Google Chrome Store, however the repository was started in 2016 meaning extensions before that date are not available. We implemented another script called `generate_links.py` that collects every extension on the disk that was downloaded from Google Chrome Store and then queries the archive for older versions. In the end the script will produce another JSON file called `old_extensions.json` that contains download URLs for older versions. By changing the command to `archive` when executing `ChromeeDownloader.py`, the program can use `old_extensions.json` and download extensions from the archive to the disk. A visualisation of the complete execution flow can be seen in Figure 4.2.

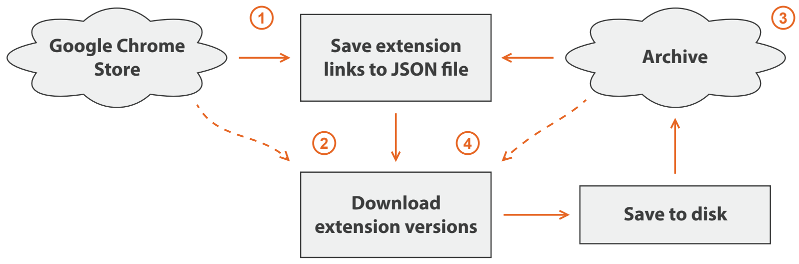

Google Chrome Store
1
Save extension links to JSON file
Archive
3
2
4
Download extension versions
Save to disk

Figure 4.2: Visualisation of ChromeeDownloader's execution flow. In step 1, extension links from the sitemap of Google Chrome store are saved to a JSON file. In step 2, `ChromeeDownloader` reads the JSON file and downloads all extensions to disk. In step 3, all extensions from disk are read and a query is sent to the archive to see whether older versions exists. If found, a download link of the version is saved to a JSON file. The last step, i.e., step 4, `ChromeeDownloader` uses the new JSON file to download extensions from the archive.

4.2 Designing the analysis process

Once all of the extensions have been downloaded, the system will begin analysing the extensions. The analysis process can be described in three steps, *generating a list of extensions*, *static analysis* and *threat analysis*. A visualisation of the analysis process can be seen in Figure 4.3.

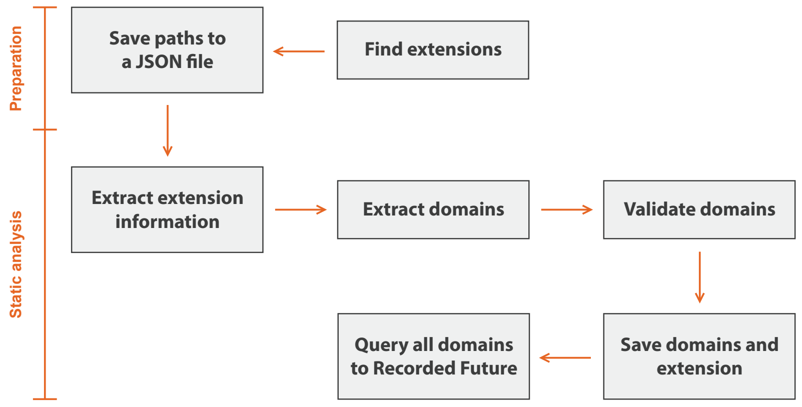

Preparation
Static analysis
Save paths to a JSON file
Find extensions
Extract extension information
Extract domains
Validate domains
Query all domains to Recorded Future
Save domains and extension

Figure 4.3: Visualisation of the static analysis process.

4.2.1 Generate extensions

The program begins from step 1, by generating a list of extensions to be analysed from a folder containing all the extensions. It starts by letting the user enter a path which gives the program a starting point on where to look for extensions. Every extension contains either a `manifest.json` or `install.rdf` file, by searching for these files we can identify an extension and store it in a list. Once the program is completed, every extension that was found will be stored in a JSON file.

4.2.2 Static analysis

The analysis process makes use of Python's multiprocessing library in order to make the program run faster. The program follows a producer-consumer design similar to the previous components. In the analysis process the producer is responsible for reading the paths from the previously generated JSON file and place them into a queue. The consumers fetch the paths of the extensions from the queue and begin extracting information such as name, version and author from its manifest file.

The process of extracting information from a manifest file has been heavily adapted to the developers of the extensions. We noticed that many developers does not follow the recommendation on how to write a manifest file, this problem occurred mostly for extensions with `install.rdf` files. Some of the problems were comments in JSON files, many developers wrote C/C++ styled comments in their JSON files even though it is not supported in JSON [26], this led to the JSON parser failing. Another problem was the syntax style in `install.rdf` files, we found that the XML tags were written differently, for instance some had uppercase letters where they do not belong and vice versa. According to the official documentation [27] information in `install.rdf` should be written inside of XML elements (as can be seen in Listing 4.3) however, we have found multiple occurrence where the information were written in the attributes of the element (Listing 4.4), in some cases developers used both attributes and elements, i.e., some information was written in the element and the other ones in the attribute. Still, there are occurrences where our parser fails due to syntax errors. When this happens, information will be extracted from the directory path since it contains a unique id and version number.

```
<Description about="urn:mozilla
    :install-manifest">
    <em:id>{id}</em:id>
    <em:name>Ext name</em:name>
    <em:version>1.0.0</em:
        version>
</Description>
```

Listing 4.3: Information specified inside XML elements.

```
<Description about="urn:mozilla
    :install-manifest">
    em:id="{id}"
    em:name="Ext name"
    em:version="1.0.0"
</Description>
```

Listing 4.4: Information specified inside XML attributes.

4.2.2.1 Regular Expressions

The static analysis is performed by searching for addresses in the source code of all files stored in an extension. In order match and extract addresses, regular expressions *(regex)* are used. Regular expressions are used to create search patterns for finding a specific sequence of characters such as a phone number, URL or an IPv4 address. Figure 4.4 shows how searching for a sequence of characters can be expressed with regular expressions.

Regular Expression	**String**	**Result**
`(\d{4}-\d{6})`	My number is 0750-346781	0750-346781
`([A-Z])`	Cozy Hawk Apple Logan Merry Ed Rogue Sun	CHALMERS

Figure 4.4: Regular Expression examples.

Our initial regex pattern searched for domains and IP addresses in the following format `http(s)://www.domain.com, www.domain.com, domain.com`, but this pattern resulted in a large amount of false positives. Addresses classified as false positives are addresses found in contexts that are not used in any form of communication or computation. An example of this could be a block of text presenting some information which contains several addresses. While these addresses may be of interest, it is impossible through static analysis to understand the purpose of these addresses without manually analysing or performing dynamic analysis. This is discussed further in Chapter 7.

To reduce the amount of false positives we constructed a regex pattern that searched for addresses in the context of variables and functions, e.g.,
`var domain = "www.dubell.io"` or `xhr.open("GET", "https://dubell.io");`.
The updated regex yielded better results and reduced the amount of false positives.

With the updated regex, we noticed some problems. It performed well most of the time but sometimes it lead to serious problems and affected the process's performance. The problem was the code format in some files. We found that many developers uglifyed or minified their code which makes the code appear in one line. When our regex found a line containing e.g., 50,000 characters, the process time increased. In some cases, it took hours to find a match in a string with more than 100,000 characters. The reason why the process time increases with larger strings is due to the fact that Python's regular expression library called `Re`, is based on recursive backtracking. The algorithm used by `Re` is simple but can be slow because it can read a string multiple times before it finds a match [28]. If no match is found, the regex must try all possible patterns before it gives up, which means that the algorithm has a worst-case exponential complexity. An example of this problem can be explained as follows: consider a regex $(a?)^n a^n$ ($(a?)^2 a^2$ is shorthand for $a?a?aa$) which is used on string a^n. The performance of the regex on the string is illustrated in Figure 4.5. As can be seen the time increases exponentially the longer the string is. To match a string with the length of 29 characters takes little more than 60 seconds.

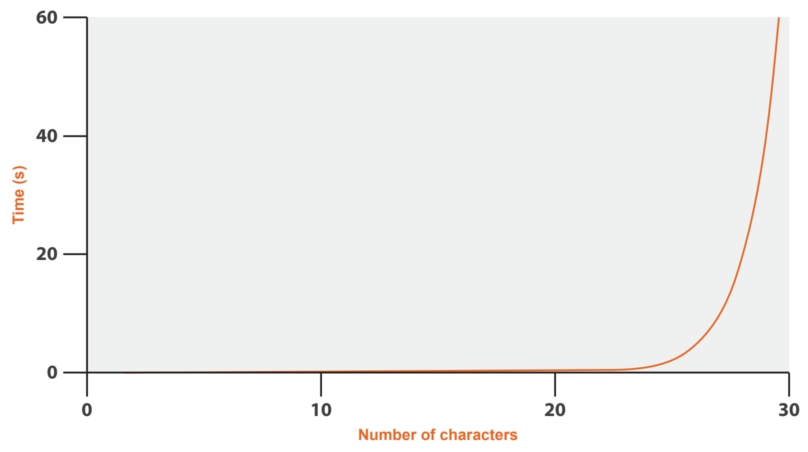


Figure 4.5: Processing time of regex $(a?)^n a^n$ on string a^n. [28].

To solve this problem we tried to find a regex algorithm with better time complexity. We did find algorithms that are using finite automaton, however they were not fully implemented and could not replace Python's `Re` library completely. To overcome this obstacle we used a beautifier to reformat the problematic files, however it did create a new problem for us. The beatify scripts loads the file into memory during its process, and may produce some issues if the beautifier is used together with multiprocessing. We found that if multiple large files were beatified during the same time, the memory would be filled up leading to a deadlock state in the program.

4.2.2.2 Validating extracted addresses

Performing static analysis by using regular expressions to extract addresses proved to be difficult. This is because there are many patterns in the source code of the extensions that resemble the appearance of a domain name or an IP address. Therefore it is necessary to validate extracted addresses and to remove any false positives.

Validating extracted domains and IP addresses are done in two steps. The first step when validating domain names is to check whether its top-level-domain (TLD) is valid. The last step is to query DNS resolvers for any resource records. If the DNS returns any resource records such as *A*, *AAAA*, *TXT* or *NS*, then the domain has been successfully validated. IP addresses are validated by first checking if the IP follows the specification of a valid IPv4 address. Next step is to check if the IP belongs to any private or reserved IP address spaces, if not then the IP is considered validated.

4.2.2.3 Saving data

Once all domains and IP addresses have been extracted from the extension's source files, the results are stored in our MongoDB database. We use two collections, one that keeps information about extensions and the other one that keeps information about domains and IP addresses. By using this we prevent duplication of domains in our database. Before any domain is saved, a database query is sent to check whether the domain already exist in our database, if not the domain will be inserted inside the domain collection. An example of domain object in the database can be seen in Listing 4.5.

```
{
    "_id" : ObjectId("5919bc7f0aff0066fe98736f"),
    "name" : "google.com"
}
```

Listing 4.5: An example of an object an object in domain collection that contains information about a domain.

When all addresses found for a given extension have been saved to the database, we collect each domain's *ObjectId* and save them to the extension's address list. An example of an extension object in the database can be seen in Listing 4.6.

```
{
    "_id" : ObjectId("58fd3d2a0aff00704727dd8e"),
    "version" : "2.7.1-signed",
    "name" : "Multi Dictionary Lookup",
    "author" : "Nohup Technologies",
    "browser" : "Mozilla Firefox",
    "addresses" : [
        ObjectId("58fd3d2a0aff00704727dd69"),
        ObjectId("58fd3d2a0aff00704727dd64"),
        ObjectId("58e253300aff0074b529f7e2"),
        ...
    ],
    "created_at" : ISODate("2017-04-24T01:47:54.962Z"),
    "modified_at" : ISODate("2017-04-24T01:47:54.962Z")
}
```

Listing 4.6: An example of an object in the extension collection that contains information about an extension.

4.2.3 Threat analysis

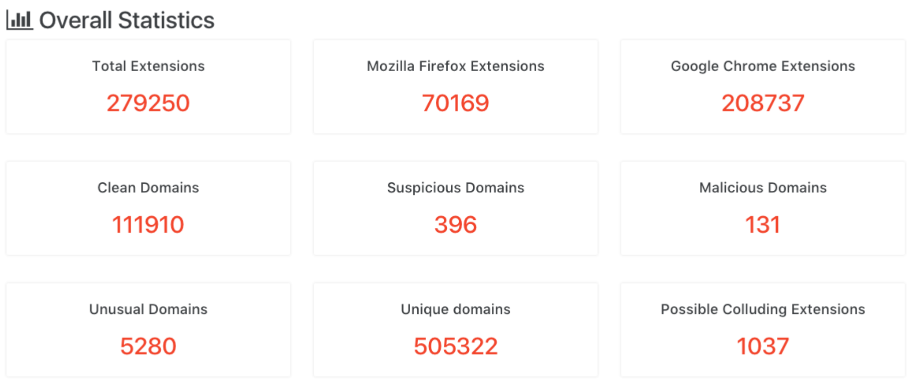
The last process is threat analysis from Recorded Future. This is done by reading every object in the domain collection and extracting the name of address. The address name is sent to Recorded Future via their API which returns a JSON object containing their threat analysis. The threat analysis from Recorded Future is appended to the address and stored under the key *recorded future*. When querying Recorded Future multiple times, we can simply retrieve objects which do not include this key. By doing this we prevent sending multiple requests for the same address to Recorded Future. An example of how an address object looks after threat analysis from Recorded Future can be seen in Listing 4.7.

```
{
    "_id" : ObjectId("5919bc7f0aff0066fe98736f"),
    "name" : "google.com",
    "recorded_future" : {
        "criticalityLabel" : "None",
        "criticality" : 0,
        "score" : 0,
        "evidenceDetails" : [ ]
    },
    "modified_at" : ISODate("2017-05-19T09:39:35.682Z")
}
```

Listing 4.7: An example of an object in the collection that contains information about a domain and threat analysis from Recorded Future.

4.3 Visualising collected data

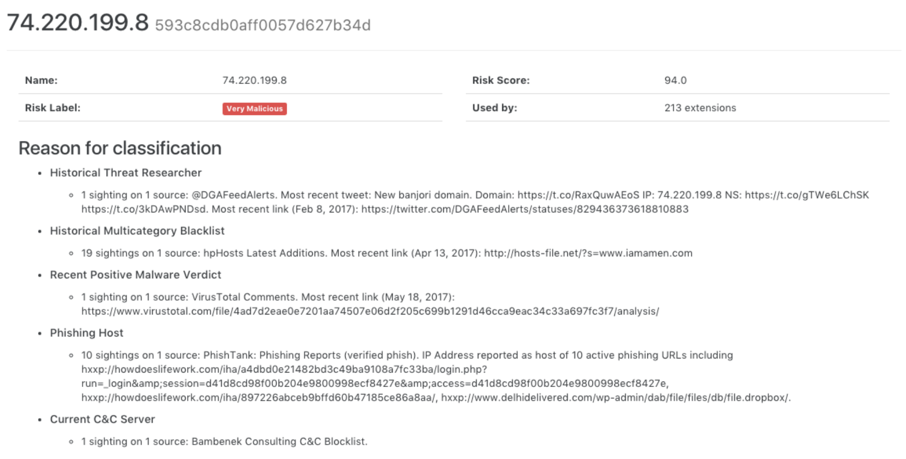
The final step in the implementation process is to visualise the collected data. This is done by creating a Flask [20] web application that interfaces with the database server. Visualising the data involves writing database queries which returns the desired results. Figure 4.6 shows how we visualise the overall statistics and figure 4.8 shows malicious domains grouped by country.


Overall Statistics

Total Extensions 279250	Mozilla Firefox Extensions 70169	Google Chrome Extensions 208737
Clean Domains 111910	Suspicious Domains 396	Malicious Domains 131
Unusual Domains 5280	Unique domains 505322	Possible Colluding Extensions 1037

Figure 4.6: A screenshot from the web application that shows some statistics.

Every address found has their own detail page where information about the collected address can be viewed. Figure 4.7 shows the evidence details for the received classification.



74.220.199.8 593c8cdb0aff0057d627b34d

Name:	74.220.199.8	**Risk Score:**	94.0
Risk Label:	Very Malicious	**Used by:**	213 extensions

Reason for classification

- Historical Threat Researcher
  - 1 sighting on 1 source: @DGAFeedAlerts. Most recent tweet: New banjori domain. Domain: https://t.co/RaxQuwAEoS IP: 74.220.199.8 NS: https://t.co/gTWe6LChSK https://t.co/3kDAwPNDsd. Most recent link (Feb 8, 2017): https://twitter.com/DGAFeedAlerts/statuses/829436373618810883
- Historical Multicategory Blacklist
  - 19 sightings on 1 source: hpHosts Latest Additions. Most recent link (Apr 13, 2017): http://hosts-file.net/?s=www.iamamen.com
- Recent Positive Malware Verdict
  - 1 sighting on 1 source: VirusTotal Comments. Most recent link (May 18, 2017): https://www.virustotal.com/file/4ad7d2eae0e7201aa74507e06d2f205c699b1291d46cca9eac34c33a697fc3f7/analysis/
- Phishing Host
  - 10 sightings on 1 source: PhishTank: Phishing Reports (verified phish). IP Address reported as host of 10 active phishing URLs including hxxp://howdoeslifework.com/iha/a4dbd0e21482bd3c49ba9108a7fc33ba/login.php?run=_login&amp;session=d41d8cd98f00b204e9800998ecf8427e&amp;access=d41d8cd98f00b204e9800998ecf8427e, hxxp://howdoeslifework.com/iha/897226abceb9bffd60b47185ce86a8aa/, hxxp://www.delhidelivered.com/wp-admin/dab/file/files/db/file.dropbox/.
- Current C&C Server
  - 1 sighting on 1 source: Bambenek Consulting C&C Blocklist.

Figure 4.7: A screenshot from the web application showing the domain information page. A IP which has been classified as Very Malicious.

Viewing malicious domains grouped by country is also possible to view, as can been seen in Figure 4.8. This gives an interesting indication to where most malicious domains registered.

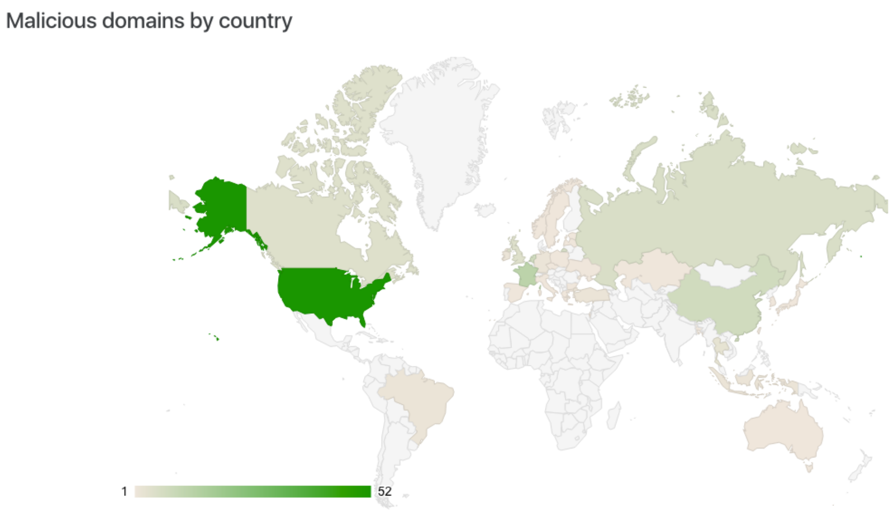


Figure 4.8: Malicious domains grouped by country.

5 Evaluation

Before performing the static analysis on the collected extensions, it is necessary to evaluate the analysis process in order to reduce the amount of false positives and verifying that addresses are being extracted.

The evaluation of the static analysis was performed in two phases. Phase one involved creating two custom made extensions that are colluding. Phase two involved choosing a total of 20 random extensions and performing both automatic and manual analysis on the extensions. The extracted addresses from the automatic and manual analysis are compared and the results are presented.

5.1 Custom made colluding extension

The first evaluation that was performed involved custom made extensions. Two extensions were created that together performs a collusion attack. Their critical code can be seen in Listing 5.1 and 5.2. As can be seen they both use the same external server (i.e., domain `dubell.io`) for communication.

```
var targetPage = "https://dubell.io/about?chromyextension=true"

var x = new XMLHttpRequest();
x.open("GET", targetPage);
x.send()
```

Listing 5.1: Example of one of the custom made extensions.

```
document.body.style.border = "5px solid red";

var targetPage = "https://dubell.io/about?foxyextension=true";

navigator.sendBeacon(targetPage, "hello");
```

Listing 5.2: Example of one of the custom made extensions.

The extensions were then checked in our analysis process to see whether the collusion could be detected. The result was that both extensions had received `dubell.io` under its addresses, as can be seen in Listing 5.3

```
> db.domains.find({"name": "dubell.io"}, {"name": 1})
{ "_id" : ObjectId("5938aa560aff00059dd60443"), "name" : "dubell.io
   " }
>
> db.analysis.find({"addresses": {$in: [ObjectId("5938
   aa560aff00059dd60443")]}}, {"name": 1})
{ "_id" : ObjectId("593c1fa70aff0057d3279388"), "name" : "
   ChromyDownloader" }
{ "_id" : ObjectId("593c1faf0aff0057d627948b"), "name" : "
   FoxyExtension" }
```

Listing 5.3: The result of the analysis process when processed on the custom made extensions. Both extensions are using `dubell.io` for communication.

5.2 Testing our Analysis

As over 270,000 extensions were downloaded, analysing them all manually is an unfeasible task. Therefore, we proceed to test how the previous automatic analysis works by performing some (manual) tests over a randomly selected subset of the extensions. Such analysis may give a slight indication on the behaviour of the automatic process, and how it performs.

Before the analysis started, 10 Google Chrome extensions and 10 Firefox Extensions were randomly selected from the extensions that were downloaded and available on the disk. Every file of the extensions was manually examined and compared with the automatic analysis. We focused on domains and IP addresses that were somehow linked to an external communication, other domains were not considered. A short summary of the result of the analysis can be seen Tables 5.1 and 5.2 and a detailed result can bee seen in Appendix A.

Extension No.	Test (manual analysis)	Automatic analysis	accuracy
Extension 1	6	7	85.7%
Extension 2	0	0	100%
Extension 3	5	2	40%
Extension 4	7	7	100%
Extension 5	10	9	90%
Extension 6	1	1	100%
Extension 7	0	0	100%
Extension 8	13	13	100%
Extension 9	15	14	93.3%
Extension 10	1	1	100%
Average accuracy: 90.90%			

Table 5.1: Evaluation of 10 random Google Chrome extensions and the accuracy of the script.

Extension No.	Test (manual analysis)	Automatic analysis	accuracy
Extension 1	2	2	100%
Extension 2	11	11	100%
Extension 3	0	2	0%
Extension 4	1	1	100%
Extension 5	8	2	25%
Extension 6	4	4	100%
Extension 7	2	2	100%
Extension 8	2	2	100%
Extension 9	2	2	100%
Extension 10	14	14	100%
Average accuracy: 82.5%			

Table 5.2: Evaluation of 10 random Mozilla Firefox extensions and the accuracy of the script.

The accuracy is calculated by counting the total amount of domains found during the manual and automatic analysis. The calculation can be expressed as $card(A \cap B) \div card(A \cup B)$, where A and B represent the set containing the domains found during the manual and automatic analysis, respectively.

The result from this evaluation shows that the automatic analysis process achieves 90.90% accuracy for Google Chrome extensions while it achieves 82.5% accuracy for Mozilla Firefox extensions. Even though these results are promising, we are not in the position of making claims about the general accuracy due to the low sample size for our tests.

6
Results

6.1 Extensions

In total 279,180 extension versions were downloaded and used in the analysis process, of these 70,154 were Mozilla Firefox extensions and the rest, 209,026, were Google Chrome extension. A detailed list of this can be seen in Table 6.1.

Mozilla Firefox extensions	
Number of extensions found from Mozilla Firefox Add-ons	18,841
Number of downloaded extensions	15,819
Number of failed downloads	173
Number of previous versions	70,154
Elapsed time	371 minutes
Disk Space	70 GB
Google Chrome extensions	
Number of extensions found from Google Chrome store	113,548
Number of failed downloads	6,915
Number of downloaded extensions	106,633
Elapsed time	476 minutes
Number of downloaded versions	209,026
Disk Space	636 GB

Table 6.1: Information about the downloaded extensions.

6.1.1 Mozilla Firefox extensions

During the download of Firefox extensions, there occurred 4 unique errors from a total of 173 errors. The errors include, `HTTP Error 404: Not Found`, `HTTP Error 500: Internal Server Error`, *BadZipFiles* and *HTTP Connection: Max retries exceeded*. When comparing the amount of downloaded extensions with the amount of extensions reported on Mozilla's extension store, a large discrepancy was found. The reason for this difference was because of the fact that there existed 3022 duplicate extensions on their store.

6.1.2 Google Chrome extensions

When downloading extensions from Google Chrome store, there occurred 6,915 errors. The most commons error messages were `HTTP Error 401: Unauthorised`, `Could not extract version from detail page` and `Not found`, they occurred 6,213, 577 and 87 times respectively. The first error message `HTTP Error 401`, was produced due to that the extension requires the user to be signed in with her Google account in order to download it. This requirement is normally present if the extension costs money and must be bought. The second error message is due to parsing problems, meaning the program could not extract information from the page. However, wrong pages were parsed since the links were redirected to another page pointing to *G Suite Marketplace*. The third and last error occurred when the Google chrome store returned HTTP status 404, which means the specific extension could not be found.

6.2 Analysis

All extensions were analysed and 505,322 domains and IP addresses could be extracted from 190,479 extensions. No domains or IP addresses were found from the remaining the 88,771 extensions. Of the extracted domains, only 117,717 were given a classification from Recorded Future. Recorded Future's threat analysis also showed that 111,910 domains and IP addresses were not considered to be any threat. However, the remaining did receive a classification which is illustrated in Figure 6.1.

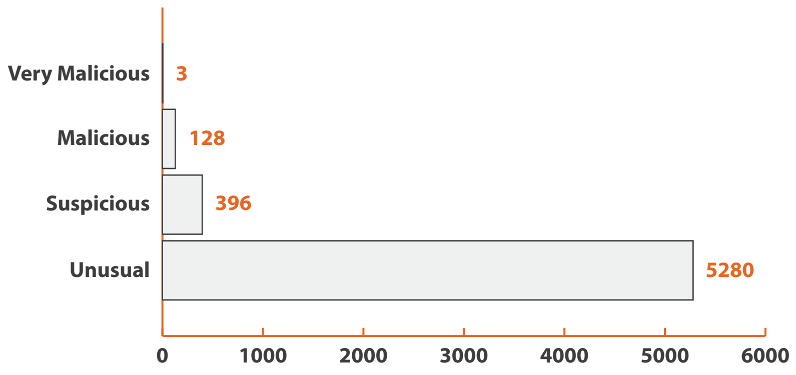


Figure 6.1: Number of domains and IP addresses that were given classifications: very malicious, malicious, suspicious, unusual from Recorded Future's threat analysis.

173,433 extensions used a domain or IP address that can be found in other extensions. A total of 17,046 extensions used unique domains, i.e., domains only found in those particular extensions. Of those extensions that used a domain or IP address also used by other extensions, 109,972 of them used a domain that was given a classification from Recorded Future. Most of these extensions used domains or IP address that was classified as *None* or *Unusual*, the numbers were 87,454 and 88,091

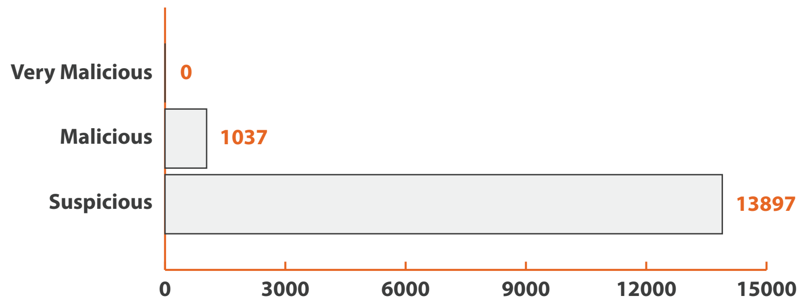


Figure 6.2: Number of extensions that used domains and IP addresses that were given classifications: very malicious, malicious, suspicious from Recorded Future's threat analysis. Only domains and IP addresses also used by other extensions are shown.

respectively. The remaining extensions used domains with a different classification, which can be seen in Figure 6.2.

Of the malicious domains used by multiple extensions, 948 could be found in Google Chrome extensions. The remaining 89 domains were used by Mozilla Firefox extensions. Similar proportion can be seen with the suspicious domains, in which 11,899 were used by Google Chrome extensions and 1,962 by Mozilla Firefox extensions. On average, domains with the classification `Malicious` and `Suspicious` did occur more frequently in Google Chrome, if compared with the total amount of extensions. This is illustrated in Figure 6.3.

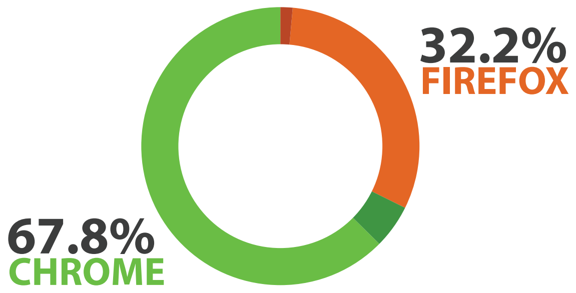


Figure 6.3: Illustrates the average usage of malicious, suspicious domains and IP addresses in Google Chrome and Mozilla Firefox. The amount of malicious domains in this circle diagram is 5.00% for Google Chrome and 1.40% for Mozilla Firefox, this is illustrated with a darker shade in the diagram.

From our results we can conclude that the amount of possible colluding extensions is 1,037, which is 0.37% of all extensions that we analysed.

7
Discussion

Before this research we did not know if there existed browser extensions performing collusion attacks via external servers. The goal of this research has been to uncover potential colluding browser extensions and to give an indication to which extensions and domains are malicious. In order to achieve this goal, a platform was developed for downloading and performing static analysis on browser extensions, as discussed in Chapter 4. Recorded future's web intelligence engine was used for classifying domains and IP addresses, their platform was discussed in Chapter 2.

During the implementation phase we identified some limitations and obstacles. Recorded Future allows us to classify addresses and provides evidence for its classification. However, one can not claim with 100% certainty that a specific domain or IP is clean or malicious. If an address has been classified as clean, that means that Recorded Future has not recorded any malicious activity regarding this address. The specific address could still be malicious, Recorded future has just not collected any sightings for it. Similarly if an extension is found to be colluding via a domain which has been classified as suspicious or malicious, this particular domain could be part of a popular online service that has been used by other malware and thus been classified as malicious. Nevertheless, Recorded Future does provide evidence for its classification and can thus give an indication to why a domain has received a particular classification. Other limitations were found during the static analysis process, that proved to be difficult.

When performing the static analysis, regular expressions are used to find patterns that match domain names and IP addresses. This works fine in general but in the case of this research, the goal is to find addresses used in the context of external communication. With regex, one can define many patterns that look for addresses in different contexts, e.g., variable definitions and during function calls. While this method works, there is a chance some addresses will be disregarded because they were specified in a format that was not anticipated. There is also the possibility that valid addresses will be matched, but are not used for any communication. These could be addresses specified in a random string.

Since browser extensions are dynamic and event driven, it is difficult to understand the behaviour of an extension with a high degree of certainty, simply through static analysis. Browser extensions can be triggered in many different ways, a few of these ways include activating when a specific website is being visited or during a specific time during the day. These types of extensions are very hard if not impossible to analyse correctly through static methods.

Other methods that could be used to accurately detect addresses used in communication include dynamic analysis and Artificial Intelligence *(AI)*. Dynamic analysis involves running the browser extension in a controlled environment and analysing its behaviour. This method allows researchers to identify how browser extensions may interact with external servers. However as previously mentioned, browser extensions can be programmed to only execute under set of pre-determined parameters, e.g., time of day.

Another option would be to use AI. By training an AI on multiple extensions to learn when an address is correct, i.e., being used for external communication, the analysis process to could possibly with a higher degree of certainty detect the correct addresses. This method could potentially be combined with dynamic analysis. By analysing an extension's behaviour and monitoring its input/output, an AI can perhaps learn how communication with external servers look like. Understanding how and when a specific browser extension is executed is vital, for performing analysis with high accuracy.

Dealing with obfuscation is also an important aspect. While our analysis process can deal with some obfuscation techniques, it is not enough. There exists infinitive different ways to obfuscate a piece of code. Simply trying to detect certain obfuscation methods are not enough. To overcome obfuscated code when performing an analysis of a browser extension, pre-compiling the code would compute all the values, functions and therefore "remove" the obfuscated code. This method would greatly contribute to the field.

7.1 Future work

We believe there is much research to be done when performing analysis of browser extensions. As discussed in the previous section, areas that could greatly improve future research include:

- Dynamic Analysis
- De-obfuscation techniques
- Artificial Intelligence methods

Expanding the analysis process to support more browser extension stores such as the Windows Store, in order to analyse more browser extensions would offer greater insight to the types of threats that are present for each store. Modern browser extensions are increasingly becoming more cross-platform, meaning browser extensions developed for Google Chrome will work in Mozilla Firefox and vice verse.

In addition to our work, we would like to develop an extension that users can install in their browser to find out if any of their other installed extensions are colluding.

This would be done by hosting our platform online in order for the extension to query the platform for information. By providing this service users can see which of their extensions are communicating with malicious servers. This would also be useful in the event of an extension turning malicious in the future by introducing malicious servers, one the browser extension has received a great deal of users. The platform would be constantly analysing extensions and would detect this change.

8 Conclusion

There exists different types of collusion attacks regarding browser extensions. In this paper, a particular from of collusion attack was investigated which are browser extensions performing collusion attacks in collaboration with external servers. In order to detect these browser extensions, a platform was developed to download and perform static analysis on all extensions provided by the Mozilla Firefox and Google Chrome store. Recorded Future's threat intelligence was used for classifying domains and IP addresses extracted from extensions. The proposed method found 1,037 potentially colluding extensions and 131 domains and IP addresses classified as malicious by Recorded Future.

The method has however, shown to have some limitations. The static analysis is based on regular expressions which searches for domains and IPs in the context of communication. Domains and IPs can be specified in several different ways and can also be obfuscated in order to hide the address. As a result, static analysis in combination with regular expressions have proven to be difficult to with 100% accuracy find addresses used in external communication. Regular expressions are based on creating search patterns, meaning there might exist a domain specified in a way that the search pattern did not anticipate. Therefore, addresses used in external communication might be disregarded. Similarly, by creating a more relaxed regular expression, addresses not used in external communication might be extracted, producing a false-positive.

Regular expressions, although very good for finding patterns, is not the ideal tool for statically analysing browser extensions. When using regular expressions, it is necessary to create a balance between matching false-positives and addresses being used in external communication. However, despite these limitations, the results of this research has proven the existence of possible colluding extensions and provided a great starting point for future research.

Protecting against colluding browser extensions can be very difficult for the average internet user. This is due to the fact that the browser does not inform the user when extensions collect private information and send them to an external server. Browsers could force a more granular permission model which allow users to grant permission to specific actions made by extensions, such as sending data to a specific server. Other countermeasures include providing the user with a complete history of an extension's activity, which would include events such as collecting information from web pages and transmitting the data to an external server. This kind of activity could reveal malicious intent which would help detecting colluding extensions.

Bibliography

[1] Google Chrome, *Manifest file format*, 2017-03-16, [Online]. Available: `https://developer.chrome.com/extensions/manifest`.

[2] ——, *Permission warnings*, 2017-03-16, [Online]. Available: `https://developer.chrome.com/extensions/permission_warnings`.

[3] L. Liu, X. Zhang, G. Yan, S. Chen, *et al.*, "Chrome extensions: Threat analysis and countermeasures.", in *NDSS*, 2012.

[4] A. Saini, M. S. Gaur, V. Laxmi, T. Singhal, and M. Conti, "Privacy leakage attacks in browsers by colluding extensions", in *Information Systems Security: 10th International Conference, ICISS 2014, Hyderabad, India, December 16-20, 2014. Proceedings*, Springer International Publishing, 2014, pp. 257–276.

[5] F. Schaub, A. Marella, P. Kalvani, B. Ur, C. Pan, E. Forney, and L. F. Cranor, "Watching them watching me: Browser extensions' impact on user privacy awareness and concern", in *NDSS*, 2016.

[6] L. Bauer, S. Cai, L. Jia, T. Passaro, and Y. Tian, "Analyzing the dangers posed by chrome extensions", in *Communications and Network Security (CNS), 2014 IEEE Conference on*, IEEE, 2014, pp. 184–192.

[7] M. D. Network and individual contributors, *What are webextensions?*, Mar. 2017. [Online]. Available: `https://developer.mozilla.org/en-US/Add-ons/WebExtensions/What_are_WebExtensions`.

[8] NetMarketShare, *Desktop top browser share trend.* [Online]. Available: `https://www.netmarketshare.com/`.

[9] StatCounter, *Top 5 browsers on dec 2016.* [Online]. Available: `http://gs.statcounter.com/#desktop-browser-ww-monthly-201612-201612-bar`.

[10] Recorded Future, *Patented technology: Web intelligence engine*, 2017-02-21, [Online]. Available: `https://www.recordedfuture.com/web-intelligence-engine/`.

[11] Raymond Hill, *Ublock origin - an efficient blocker for chromium and firefox. fast and lean.* 2017-05-21, [Online]. Available: `https://github.com/gorhill/uBlock`.

[12] P. K. Akshay Dev and K. P. Jevitha, "Stride based analysis of the chrome browser extensions api", in *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications : FICTA 2016, Volume 2*, S. C. Satapathy, V. Bhateja, S. K. Udgata, and P. K. Pattnaik, Eds. Singapore: Springer Singapore, 2017, pp. 169–178, ISBN: 978-981-10-3156-4. DOI: `10.1007/978-981-10-3156-4_17`. [Online]. Available: `http://dx.doi.org/10.1007/978-981-10-3156-4_17`.

[13] Eric Capuano, *When browser extensions go rogue*, 2017-02-21, [Online]. Available: `https://blog.ecapuano.com/when-browser-extensions-go-rogue/`.

[14] M. Weissbacher, *These browser extensions spy on 8 million users*, 2016.

[15] A. Saini, M. S. Gaur, V. Laxmi, and M. Conti, "Colluding browser extension attack on user privacy and its implication for web browsers", *Computers & Security*, vol. 63, pp. 14–28, 2016.

[16] Python, *Welcome to python*, 2017-04-24, [Online]. Available: `https://www.python.org/`.

[17] VMware, *Free VMware vSphere Hypervisor, free virtualization (esxi)*, 2017-04-24, [Online]. Available: `http://www.vmware.com/products/vsphere-hypervisor.html`.

[18] The Python Standard Library, *Multiprocessing — process-based parallelism*, 2017-04-26, [Online]. Available: `https://docs.python.org/3.6/library/multiprocessing.html`.

[19] mongodb, *Mongodb for giant ideas | mongodb*, 2017-04-24, [Online]. Available: `https://www.mongodb.com/`.

[20] Flask, *Welcome | flask (a python microframework)*, 2017-04-24, [Online]. Available: `http://flask.pocoo.org/`.

[21] Mozilla, *Mozilla addons - most popular extensions*, 2017-05-16, [Online]. Available: `https://addons.mozilla.org/en-US/firefox/extensions/?sort=users`.

[22] Leonard Richardson, *Beautiful soup*, 2017-05-16, [Online]. Available: `https://www.crummy.com/software/BeautifulSoup/`.

[23] mdamian, *Github - mdamien/chrome-extensions-archive: Archive all the chrome extensions.* [Online]. Available: `https://github.com/mdamien/chrome-extensions-archive`.

[24] Google Chrome, *Crx package format*, 2017-04-25, [Online]. Available: `https://developer.chrome.com/extensions/crx`.

[25] D. Marié, *Chrome extensions archive*, 2017-05-28. [Online]. Available: `https://crx.dam.io/`.

[26] D. Crockford, "The application/json media type for javascript object notation (json)", 2006, RFC 4627.

[27] Mozilla Foundation, *Install manifests*, 2017-05-16, [Online]. Available: `https://developer.mozilla.org/en-US/Add-ons/Install_Manifests`.

[28] R. Cox, *Regular expression matching can be simple and fast*, Jan. 2007. [Online]. Available: `https://swtch.com/~rsc/regexp/regexp1.html`.

A

Evaluation

The following formula was used to calculate the accuracy for the test:

$$card(A \cap B) \div card(A \cup B)$$

A represent the set of domains found in the manual analysis while B represent the set found in the automatic analysis. If $A \cap B = A \cup B$ then the accuracy for the test is 100%. The accuracy is decreased if false positives is found in the manual analysis and if domains are present in automatic analysis but missing in the manual test.

1 Evaluation of Google Chrome Extensions

10 random Google Chrome Extensions have been selected and manually analysed and compared with our script.

1.1 Extension number 1

Random integer	24292
Extension name	Code Climate
Extension version	620
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`docs.codeclimate.com`	`docs.codeclimate.com`
`api.codeclimate.com`	`api.codeclimate.com`
`codeclimate.com`	`codeclimate.com`
`github.com`	`github.com`
`api.segment.io`	`api.segment.io`
`notify.bugsnag.com`	`notify.bugsnag.com`
	`gmail.com`
Total: 6	**Total:** 7
Accuracy: 85.7%	

1.2 Extension number 2

Random integer	123247
Extension name	Smart TOC
Extension version	0.3.5
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
None	*None*
Total: 0	**Total:** 0
Accuracy: 100%	

1.3 Extension number 3

Random integer	39522
Extension name	So Sánh Giá - Mã Giàm Giá (SoSanh24h.Com)
Extension version	5.85
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`www.sosanh24h.com`	`www.sosanh24h.com`
`sosanh24h.com`	`sosanh24h.com`
`go.sosanh24h.vn`	
`go.masoffer.net`	
`giamgiachotui.com`	
Total: 5	**Total:** 2
Accuracy: 40%	

1.4 Extension number 4

Random integer	144129
Extension name	StopAll Ads
Extension version	1.0.91

Found domains & IP addresses

Manual analysis	**Automatic analysis**
`www.stopallads.com`	`www.stopallads.com`
`malwaredomains.com`	`malwaredomains.com`
`stopallads.s3.amazonaws.com`	`stopallads.s3.amazonaws.com`
`easylist.stopallads.org`	`easylist.stopallads.org`
`www.fanboy.co.nz`	`www.fanboy.co.nz`
`adblockplus.org`	`adblockplus.org`
`www.tweakingtechnologies.com`	`www.tweakingtechnologies.com`
Total: 7	**Total:** 7

Accuracy: 100%

1.5 Extension number 5

Random integer	7138
Extension name	US AirForce Tab
Extension version	1.0.1
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`www.facebook.com`	`www.facebook.com`
`twitter.com`	`twitter.com`
`plus.google.com`	`plus.google.com`
`www.pinterest.com`	`www.pinterest.com`
`www.linkedin.com`	`www.linkedin.com`
`www.reddit.com`	`www.reddit.com`
`ssl.google-analytics.com`	`ssl.google-analytics.com`
`happyhey.com`	`happyhey.com`
`www.happyhey.com`	`www.happyhey.com`
`www.google-analytics.com`	
Total: 10	**Total:** 9
Accuracy: 90.0%	

1.6 Extension number 6

Random integer	104366
Extension name	Button Google Smart
Extension version	0.1
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`blog.google`	`blog.google`
Total: 0	**Total:** 0
Accuracy: 100%	

1.7 Extension number 7

Random integer	181365
Extension name	Cursor Sparkles
Extension version	1.1
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
None	*None*
Total: 0	**Total:** 0
Accuracy: 100%	

1.8 Extension number 8

Random integer	102445
Extension name	thinkContext
Extension version	1.0.1
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`schema.org`	`schema.org`
`www.thinkcontext.org`	`www.thinkcontext.org`
`twitter.com`	`twitter.com`
`www.facebook.com`	`www.facebook.com`
`developer.mozilla.org`	`developer.mozilla.org`
`adsonar.com`	`adsonar.com`
`msn.com`	`msn.com`
`doubleclick.net`	`doubleclick.net`
`overture.com`	`overture.com`
`ad.doubleclick.net`	`ad.doubleclick.net`
`plus.google.com`	`plus.google.com`
`www.bing.com`	`www.bing.com`
`lin1.thinkcontext.org`	`lin1.thinkcontext.org`
Total: 13	**Total:** 13
Accuracy: 100%	

1.9 Extension number 9

Random integer	110228
Extension name	Better Destiny.gg
Extension version	1.7.1
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`schema.org`	`schema.org`
`api.overrustle.com`	`api.overrustle.com`
`www.destiny.gg`	`www.destiny.gg`
`downthecrop.xyz`	`downthecrop.xyz`
`api.betterttv.net`	`api.betterttv.net`
`strawpoll.me`	`strawpoll.me`
`www.strawpoll.me`	`www.strawpoll.me`
`www.overrustle.com`	`www.overrustle.com`
`www.destiny.gg`	`www.destiny.gg`
`destiny.gg`	`destiny.gg`
`twemoji.maxcdn.com`	`twemoji.maxcdn.com`
`cdn.jsdelivr.net`	`cdn.jsdelivr.net`
`9inevolt.github.io`	`9inevolt.github.io`
`ts.downthecrop.xyz`	`ts.downthecrop.xyz`
`www.reddit.com`	
Total: 15	**Total:** 14
Accuracy: 93.3%	

1.10 Extension number 10

Random integer	98393
Extension name	AnimeClick.it
Extension version	2.2
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`www.animeclick.it`	`www.animeclick.it`
Total: 0	**Total:** 0
Accuracy: 100%	

2 Evaluation of Mozilla Firefox Extensions

10 random Mozilla Firefox Extensions has been selected and manually analysed and compared with our script.

2.1 Extension number 1

Random integer	31494
Extension name	Contextual Google Image Search
Extension version	1.1.1-signed.1-signed
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`www.google.com`	`www.google.com`
`www.mozilla.org`	`www.mozilla.org`
Total: 2	**Total:** 2
Accuracy: 100%	

2.2 Extension number 2

Random integer	68890
Extension name	Classic Theme Restorer
Extension version	1.6.0
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`www.camp-firefox.de`	`www.camp-firefox.de`
`github.com`	`github.com`
`forums.mozillazine.org`	`forums.mozillazine.org`
`developer.mozilla.org`	`developer.mozilla.org`
`addons.mozilla.org`	`addons.mozilla.org`
`piro.sakura.ne.jp`	`piro.sakura.ne.jp`
`dummy.addons.mozilla.org`	`dummy.addons.mozilla.org`
`caligonstudios.com`	`caligonstudios.com`
`ithinc.cn`	`ithinc.cn`
`www.mozilla.org`	`www.mozilla.org`
`www.w3.org`	`www.w3.org`
Total: 11	**Total:** 11
Accuracy: 100%	

2.3 Extension number 3

Random integer	43801
Extension name	mangareader
Extension version	2.2.0
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
	`http://www.mangareader.net/`
	`http://mangafox.me/`
Total: 0	**Total:** 2
Accuracy: 0%	

2.4 Extension number 4

Random integer	47116
Extension name	TagMyGift
Extension version	0.0.13
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`staticmagick.in`	`staticmagick.in`
Total: 1	**Total:** 1
Accuracy: 100%	

2.5 Extension number 5

Random integer	18740
Extension name	SejaPremium
Extension version	0.0.27
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`www.sejapremium.com.br`	`www.sejapremium.com.br`
`upsto.re`	`upsto.re`
`uploaded.net`	
`underseo.com`	
`uploaded.net`	
`solus.sejapremium.com.br`	
`lumos.sejapremium.com.br`	
`aqua.sejapremium.com.br`	
Total: 8	**Total:** 2
Accuracy: 25%	

2.6 Extension number 6

Random integer	29920
Extension name	letaljc
Extension version	initial.rev43.1-signed.1-signed
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`icdn.pro`	`icdn.pro`
`www.mozilla.org`	`www.mozilla.org`
`www.w3.org`	`www.w3.org`
`mozilla.org`	`mozilla.org`
Total: 4	**Total:** 4
Accuracy: 100%	

2.7 Extension number 7

Random integer	22783
Extension name	Choosy
Extension version	1.1.1-signed
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`icdn.pro`	`icdn.pro`
`www.mozilla.org`	`www.mozilla.org`
Total: 2	**Total:** 2
Accuracy: 100%	

2.8 Extension number 8

Random integer	14703
Extension name	Rikaichan Japanese-Russian Dictionary File
Extension version	2.01.160101
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`polarcloud.com`	`polarcloud.com`
`www.mozilla.org`	`www.mozilla.org`
Total: 2	**Total:** 2
Accuracy: 100%	

2.9 Extension number 9

Random integer	58946
Extension name	SmileySidebar
Extension version	1.3.4.1-signed.1-signed
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`www.mozilla.org`	`www.mozilla.org`
`www.kolobok.us`	`www.kolobok.us`
Total: 2	**Total:** 2
Accuracy: 100%	

2.10 Extension number 10

Random integer	69592
Extension name	Surfmark Toolbar
Extension version	3.6.1-signed.1-signed
Found domains & IP addresses	
Manual analysis	**Automatic analysis**
`surfmark.uservoice.com`	`surfmark.uservoice.com`
`twitter.com`	`twitter.com`
`www.facebook.com`	`www.facebook.com`
`ad.doubleclick.net`	`ad.doubleclick.net`
`blog.surfmark.net`	`blog.surfmark.net`
`www.yahoo.co`	`www.yahoo.co`
`search.yahoo.co`	`search.yahoo.co`
`www.google.co`	`www.google.co`
`use.typekit.com`	`use.typekit.com`
`typekit.com`	`typekit.com`
`lab.arc90.com`	`lab.arc90.com`
`surfmark.uservoice.com`	`surfmark.uservoice.com`
`www.w3.org`	`www.w3.org`
`www.mozilla.org`	`www.mozilla.org`
Total: 14	**Total:** 14
Accuracy: 100%	