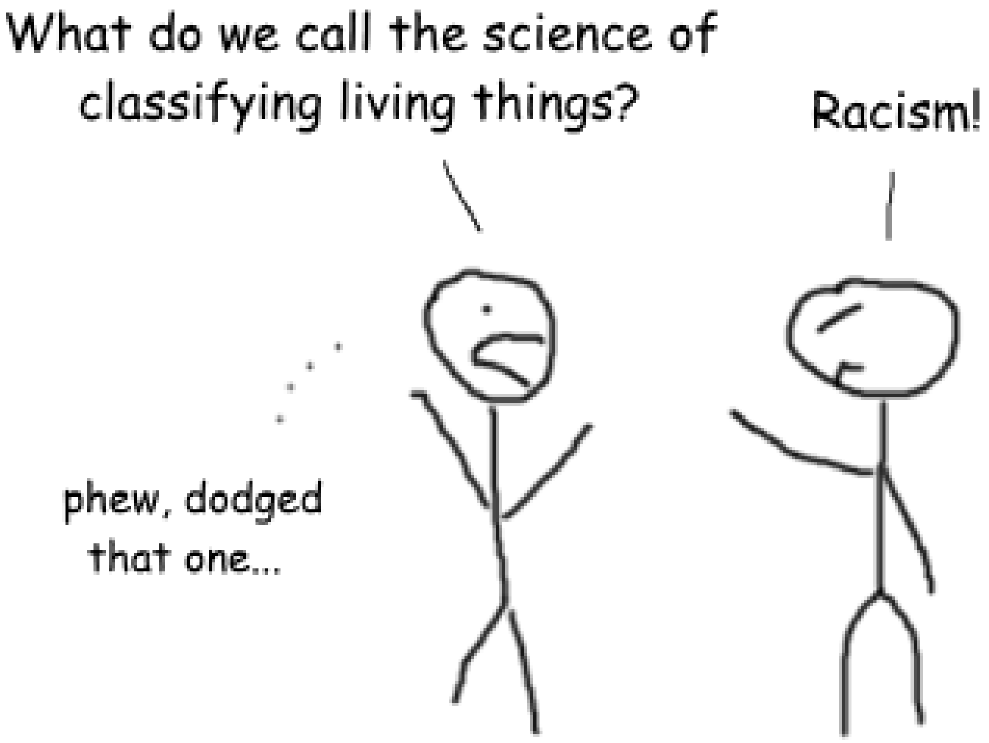

A Taxonomy of Browser Extensions

Researching metadata patterns of Chrome extensions related to security using Random Forest and k-modes

Master's thesis in Algorithms, Languages and Logic (MPALG)

Axel Arkheden, Fredrik Enetorp

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

Master's thesis 2018

A Taxonomy of Browser Extensions

Researching metadata patterns of Chrome extensions related to security using Random Forest and k-modes

Axel Arkheden

Fredrik Enetorp

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

A Taxonomy of Browser Extensions
Researching metadata patterns of Chrome extensions related to security using Random Forest and k-modes
Axel Arkheden
Fredrik Enetorp

Supervisor: Pablo Picazo-Sanchez, Gerardo Schneider, Department of Computer Science and Engineering
Examiner: Carl-Johan Seger, Department of Computer Science and Engineering

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

Typeset in LaTeX
Gothenburg, Sweden 2018

A Taxonomy of Browser Extensions
Researching metadata patterns of Chrome extensions related to security using Random Forest and k-modes
Axel Arkheden
Fredrik Enetorp
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Since the development of Google Chrome extensions is open to third party developers, there is an inherent risk of developers with malicious intents building extensions to attack end users, for example through stealing their personal information or exploiting their system resources. The sandbox system in place in Google Chrome designed to prevent such actions through warnings during installation has previously been deemed to be ineffective, consequently a new system of preventing malicious behavior or communicating risk to users is needed.
In this thesis, we investigate the feasibility of using machine learning and an extension's metadata, such as its permissions, file types, category, developer, rating, etc, to assess the security risk of an extension without examining code or executing the extension. The conclusions from our results are the following: (1) categories are basically indistinguishable in terms of metadata, which prevents outlier analysis using categories; (2) though strong feature relationships exists in the metadata, few of them are deemed relevant to security; and (3) k-modes clustering proved to be an effective way of detecting patterns in permission usage, detecting outliers and also detecting malicious extensions.

Keywords: Computer science, thesis, taxonomy, extension, chrome, machine learning, random forest, k-modes, security, metadata.

Acknowledgements

We want to thank our supervisors Pablo Picazo-Sanchez and Gerardo Schneider for their help, good advice and guidance along the way, and our examiner Carl-Johan Seger for giving good constructive criticism, helping us write the best project report possible.

Axel Arkheden & Fredrik Enetorp, Gothenburg, May 2018

Contents

List of Figures

List of Tables

1
Introduction

Web-browser extensions are on the rise, and are most popular for the more widely used browsers, such as Google Chrome and Mozilla Firefox [1]. Browser extensions are small programs that can be installed in the browser and extends its functionality. These applications, typically written in javascript, HTML and CSS, perform a wide variety of tasks, for instance changing the appearance of the browser or changing the content in a web page.

To achieve these capabilities, browser extensions can get access to several privileged API:s, acquiring a great degree of control over the browser. For instance, extensions can modify the web content through the Document Object Model (DOM) API, change HTTP headers or interact with sensitive data. These capabilities can be abused by developers with malicious intents to steal sensitive information or attack your computer [2].

Some malicious extensions are hard to detect, since they are engineered to only trigger under certain conditions, easily avoiding many tests. Therefore, it is useful to have another way of risk assessment when it comes to browser extensions, for example looking at the metadata of an extension and putting a risk-value on it. That is why in this project, a massive meta-data collection on browser extensions will be performed, where the data will be analyzed and correlated to create a taxonomy of browser extensions from a security and privacy point of view.

1.1 Problem Formulation

One of the main problems of browser extensions today, from a security point of view, is the poor communication of risk of using extensions to the user. The extent to which the security risk of installing an extension in Google Chrome is communicated by stating its permissions in a pop-up window during installation, which has been proven to be a poor method of preventing users from installing potentially malicious extensions [3]. To a user uneducated in computer security, communicating the permissions of an extension does not translate to what risks that entails, e.g. what harm the extension could do using those permissions. After a while the permission-information displayed during installation gets repetitive and monotonous, which makes the user prone to automatically clicking "accept" without thinking about the consequences of this action.

A massive metadata collection and analysis of browser extensions in Google Chrome has, to the best of our knowledge, not been done. Such an analysis could give an understanding of patterns in browser extension metadata, and help detect outliers in terms of those patterns. Outliers do of course not necessarily mean malicious extensions or extensions vulnerable to attacks, but could instead be badly written extensions, causing other problems such as having excessive file sizes or have bad performance, making them undesirable for a user anyway. A large data collection and analysis might also shine a light on problems and shortcomings in the security system currently in place for Google Chrome Extensions, and perhaps provide us with the tools to be able to suggest improvements to it.

1.2 Goal

The main goal of the project is to answer the question: Is there a good way to assess risk of browser extensions — in terms of how likely they are to steal personal information, take advantage of or inflict harm on the users hardware for example — which does not include analyzing code, and communicating this risk in a clear and intuitive way to a user? To answer that question properly, these are the most important milestones for this work:

- Collect a large amount of metadata about browser extensions and save it in a database for further studies. The data will be collected from both the Google Chrome Store and the browser extension's files.
- Perform a deep analysis of browser extensions based on more values than just the privileges declared by the extension.
- Study the metadata and apply it to various machine learning algorithms, with the goal of correlating the metadata to security risk and malicious intents.
- Gather a dataset of known malicious extensions to compare with our results.

1.3 Limitations

This project will be evaluating anomaly detection using *K-modes* only, a machine learning algorithm which can be read about in section 2.4. Comparisons could be done between different clustering algorithms to determine the most effective one, but this is not the goal of the project. Analysis of source code won't be performed in this work, only metadata of extensions will be used to assess risk.

1.4 Related work

Previous work similar to ours has been done, most of which is applied to Android applications where machine learning algorithms and other methods were used to put risk scores on apps or improve the warning messages during installation. One of these works are done by Peng et al. [4], where probabilistic classifiers were used to give Android applications a risk score based on their declared permissions. The scoring method used by them was to model the apps as vectors of independent Bernoulli variables, where each variable represents a permission in the system. An app having the value 1 on variable m means the app requests that permission, and a 0 that it does not. The risk score for an app was then the probability of an app using its particular set of permissions. To ensure the property that requesting more permissions leads to increasing the risk score — and decreasing the probability of the app's combination of permissions being declared — permissions used by more than 50% of apps were removed. In their case, this only included the INTERNET permission, which was the most popular one. The conclusions from their work were that Naive Bayes with informative priors was the best method, since different permissions can be differently weighted, making it possible for more dangerous permissions to raise an app's risk score more if requested than a less dangerous one. Dangerous in this case refers to how severe the consequences would be if the permission was abused. It also makes it possible to provide the developer with feedback of why the app has gotten its particular score, and what that developer could do to lower the score. What differs the most from this project to ours is that this method of risk scoring looks at permissions in isolation from each other, while our work instead looks at patterns in permission usage and which combinations of permissions are commonly used.

Sarma et al. [5] worked on a method of communicating the risk of installing Android apps to a user which — similarly to the current system in place on Android and Google Chrome — tries to warn users during installation of risky apps. In their work they recognized the shortcomings of prompting the user with similar warnings on almost every install, and instead aimed to introduce risk signals triggering less often and about less common and more relevant usages of permissions. The main idea behind their method was to use the categories as a divider of the needs of apps and only trigger a warning signal when an app uses one or several permissions not commonly used within its category. They defined a signal called CRCP, meaning Category-based Rare Critical Permission, working like the previously mentioned signal except only for a subset of permissions defined by them as critical. They defined three success criteria for their risk signals, being that the signals needed to have simple semantic meaning and be easy to understand, being triggered only by a small percentage of apps and being triggered by many malicious apps. The third criteria was tested using a data set of known malware. They concluded the project to be a success according to their defined criteria. Just like our work, this project looks at detecting outliers in applications according to permission usage deviating from the norm. Unlike us, they used categories as the measurement for the norm, a possibility investigated in this project with Chrome extensions, explored in sections 3.4 and 4.5.

In work done by Bloedorn et al. [6], *K-means* was used to detect patterns in network connections and it was researched whether it could also be used to single out connections with malicious intents. Classification was also used in an attempt to be able to detect malicious connections. The results were really promising, showing a detection rate of 91% for the classification and a false alarm rate of 13%. Anomalies detected by the clustering — anomalies being deviations from the norm — were compared to the set of the malicious connections, and it was concluded that 87.5 - 91.5% of detected anomalies were also malicious connections, and 1.8 - 6.75% of connections detected as the norm were malicious. These results shows promise for using this method of detecting malicious entities, perhaps even of other types than network connections. The percentage intervals in the resulting detection rates were due to testing with different data sets. *K-means* clustering was therefore concluded to be effective in detection of malicious connections, and from the results a strong connection between anomalies and malicious connections could be observed.

As demonstrated by Liu et al. [7] with an extension they created, once an extension is installed on a browser it's very easy for it to perform various malicious activities such as email spamming, DDoS and phishing attacks. They show that by having access to the 'tabs' and '<all_urls>' permissions — two very common permissions — you can create a simple email spamming bot and perform DDoS attacks. Even if you are wary of the '<all_urls>' permission, by having access to an arbitrary domain (e.g. using the '*://*.yahoo.com' permission) the extension can inject content scripts to it in order to send http requests to arbitrary domains, without the need to make an explicit cross-site HTTP request (Chrome extensions can be read about in section 2.1).

2
Background

This chapter is intended to provide the background knowledge needed to follow the idea and approach of the project properly. It starts off by explaining the architecture of Chrome, the Chrome permissions API and the composition of Chrome extensions, followed by the machine learning algorithms used in this project, namely *Random Forest* regression and classification, *K-means* and *K-modes*. *K-means* is not used in in the project, but still needs to be explained since *K-modes* is an extension of it.

2.1 Chrome

In this section we briefly describe some background about Chrome extensions including their architecture and security model.

2.1.1 Architecture of Chrome

The Chrome browser is divided into multiple processes, where each tab is a separate process plus one main background process that runs in privileged mode in order to access system resources [8]. Each auxiliary process is run in a sandbox environment and can only access system resources through the main process. This architecture is meant to protect the user from websites with malicious javascripts, but does also work well against malicious javascripts run on websites by extensions installed by the user.

While websites are usually self contained and can make due in the sandbox environment, extensions often require additional privileges and system resources to properly provide their services. For example, an extension may provide the functionality of allowing the user to download online streamed videos to disk, requiring privileges to escape the sandbox and write to disk. For this, Chrome allows auxiliary processes to request access to system resources from the main process through an API handled by permissions.

2.1.2 The Chrome Permissions API

Google Chrome has an API in place for browser extensions, which blocks functionality from an extension unless the user has approved that functionality for it [9]. Such permissions include geolocation, where an extension can access geographical information of the device used, system.storage, which can grant information about available storage devices, and fileSystem, which allows the extension to write to and read from the user's local file system. To grant an extension these capabilities, the developer needs to list them in the manifest file of the extension. If a permission is requested by an extension, the user gets prompted when installing it in the browser, and needs to accept the permissions to be able to install it. All permissions won't cause prompts however, the API only warns the user about a subset of all permissions chosen by Google [10]. Some of the permissions fall under the same warning, meaning that an extension can require multiple similar permissions and only giving the user one warning.

The Chrome browser does things to prevent users from installing malicious extensions by conveying risks of an extension before installing it (See section 2.1.2), but doesn't do much to protect the user from an exploitative extension once it's been installed. This system relies on Chrome providing relevant and useful information to the user and the user's ability to accurately judge the trustworthiness of an extension using this information. In practice however, users tend not to give these warnings much thought. This is compounded by the fact that Chrome encourages the development of extensions by third parties. The Google Chrome Store now has over 60,000 extensions which — as suggested by Sruthi Bandhakavi el at. [11] regarding firefox extensions — makes it very hard to guarantee that none of them are malicious.

2.1.3 Composition of Chrome Extensions

An extension is a small software program meant to enhance and customize the browsing experience for the user. It modifies the Chrome functionality and behavior and tailors it to the users needs. An extension is most often designed for a single, well defined task or to solve a specific problem, and it can include multiple components and a range of functionality, as long as everything contributes to perform that task or solve that problem. They are built on simple and well known web technologies such as javascript, CSS and HTML. These files are zipped into a single .crx package which is made available on the Chrome Web Store for users to download and install. This guarantees that Chrome extensions do not use content from the web in order to not expose the user to unnecessary security risks.

In Chrome, extensions have three types of components: content scripts, extensions cores and native libraries [12]. A content script is a javascript that's injected into websites when a page loads and is used to insert functionality to a web page the user is using. It has direct access to the DOM of a web page, but have no privileges other than communicating with the extension core. An extension core is a persistent background web page written in HTML and/or javascript. It constitutes the main portion of an extension and is run in the background of the web browser. It contains most of the permissions privileges and can only communicate with web content by XMLHttpRequests. Native libraries are gateways to the machine that can use all of the users privileges on the local machine. Usage of these libraries are preferably avoided since they introduce security risks [13]. An example of this is a native library called Adobe Flash Player that's known to have a lot of security flaws [14].

Chrome extensions follow the two security principles of *least privilege* and *strong isolation*. For *least privilege*, Chrome defines a set of permissions (see section 2.1.2).

An extension needs to declare what permissions it needs in the manifest file in order to use their corresponding functionality. An example of a manifest file can be seen in figure 2.1 [15]. Users then need to accept these permissions before they install the extension. An extension also needs to declare the domains it intends to make use of. By default, Chrome isolates different origins from each other, i.e it follows the *same-origin policy*. Under this policy, a web browser will only permit a web page to access data from another web page if both web pages have the same origin, e.i. originates from the same Uniform Resource Identifier (most commonly, the Uniform Resource Locator (URL) address). For an extension to access another origin it has to declare that domain among the permissions in the manifest file. Doing so allows the extension to send XMLHttpRequests and inject code into that domain. Here it's possible to list a range of domains using the wild card character '*' (e.g. *://*.foo.com).

```
{
  "manifest_version": 2,
  "name": "My Awesome Extension",
  "version": "0.1",

  "background": {
    "scripts": ["background.js"]
  },

  "content_scripts": [
    {
      "matches": ["*://*.foo.com"],
      "js": ["content.js", "content2.js"]
    }
  ],

  "permissions": [
    "tabs", "notifications", "unlimitedStorage", "*://*.foo.com"
  ],
}
```

Figure 2.1: An example of the content of a manifest file for a simple extension.

2.2 Random Forest

Random Forest is a supervised machine learning algorithm developed by Leo Breiman in 2001 [16], used for statistical analysis of relationships between different variables (regression) or classification of data. The name forest refers to the use of multiple decision trees in the algorithm, which together form a forest. When analyzing data, each node in a decision tree asks a question about a property of the data, and each branch to this node is an answer. The most common decision trees are binary, which means that each node has up to two children. This makes each question asked at each node a yes or no question with some criteria, where each child represents one of the answers.

Decision trees are quite efficient and scale really well with larger data sets, they are however quite volatile and deeper trees have a high probability of over-fitting on the data. These problems are heavily reduced by the *Random Forest* approach, where several trees are used and their answers are combined. For the classifier, this means that each tree classifies the given data, and the final class is chosen by looking at the majority vote of the trees. For the regressor, each tree gives it's estimator of the value being predicted, and the final estimator of the combined forest is taken by averaging the estimators of all the individual decision trees.

2.3 K-means

K-means is a common unsupervised machine learning algorithm, the idea behind which was first introduced by Hugo Steinhaus in 1957 [17] and later reintroduced by James MacQueen in 1967 [18] who also coined the term "*K-means*". It is used for finding patterns in data and grouping similar objects together in clusters. It works on data vectors with arbitrary size, where each entry in the vector is a real number. If the data is represented by vectors of n values, the data points can be represented as coordinates in an n-dimensional coordinate system. The algorithm is run by initially choosing a value k, which is the number of clusters to split the data into. k number of cluster center points are then initialized to random positions within the data space, and each data vector (point in the coordinate system) is assigned to the nearest cluster center. Distance is measured in euclidean distance in n dimensions, where n is the size of the data vectors. After this the center points are moved slightly towards the center of all their assigned data points, after which each data point is reassigned to the nearest cluster once again. This is repeated until the center points no longer move after data points are reassigned. The reason behind the name *K-means* is that the algorithm creates k clusters in the data, where each of the k center points is a mean value of all points in its cluster. This makes it so that the algorithm only works for numerical data and not categorical.

2.4 K-modes

To be able to utilize the *K-means* algorithm for categorical data or mixed numerical and categorical data, there had to be some changes to it. In 1998, Zhexue Huang introduced *K-modes* [19], the extension for the *K-means* algorithm that was needed. Instead of using the euclidean distance as a dissimilarity measure between two objects, the sum of categorical attributes with differing values is used instead. This makes it so that the more similar the objects, the smaller the distance is between them, just like for euclidean distance between numerical objects.

3 Methods

This chapter is dedicated to describing the methods of our work and motivating the usage of them. First, the way of downloading and storing the metadata is explained after which the way the machine learning algorithms are used in the project are described, starting off with *Random Forest* and finishing with *K-modes*.

3.1 Datasets

The extensions chosen for analysis were provided by our supervisor as a Comma-Separated Values (CSV) file, generated in January 2018. This CSV-file called `extension_ids.txt` contains the ID of each extension from the entire Chrome Web Store (at the time), a link to their chrome web-store pages and download links to the extension files, called CRX-files. Using this, a web crawler will be coded that downloads the metadata of all extensions and uploads it to an SQL-database. Extensions considered malicious by trusted sources will be looked for, adding their metadata to the database and marking them as malicious. This subset of the database will be referred to as the malicious dataset.

The web crawler is to download the CRX-file for each extension, unzip it and extract its metadata such as:

- **Permissions:** A list of permissions used by the extension (see section 2.1.2).
- **Files:** The number of files contained in the CRX-file / used by the extension.
- **Size:** The size of the entire extension, in bytes.
- **File types:** A list of all file types used in the extensions and how many of each.

The web crawler will also visit the store page for each extension and from it find and extract its metadata such as:

- **Rating:** The rating, from 1 to 5, given to the extension by users.
- **Users:** The number of users who've downloaded the extension.
- **Raters:** The number of users that have rated the extension.
- **Language:** The language used by the extension.
- **Category:** The name of the category the extension belongs to.
- **Developer:** The name of the developer of the extension.

3.2 Random Forest parameters

In this project, *Random Forest* regression is used to find strong variable correlations, and *Random Forest* classification is used to determine if there is a connection between extension categories and extension metadata. First, testing was done with the goal to find the best parameter values for the *Random Forest* so that it could produce the most accurate results possible while still being runnable on the available hardware.

There were three major parameters of the forest which were tested, which were the number of trees in the forest, the maximum depth of a tree, and the precision of features. Precision is a parameter introduced by us to both save time on the *Random Forest* tests and minimize the amount of irrelevant features. Precision controls the amount of different values of the categorical variables language and permissions, and also the amount of different file type variables used. The different values are chosen by taking the p most popular values used by extensions, for example with permissions it means the p permissions which are used the most according to the statistics gathered from the database. A higher precision means more numerical and categorical features, which also means increased running time. A too low precision on the other hand could cause important relationships among file types, languages and permissions which are not among the most used ones to be left out of the test results.

Between test runs, a new value was chosen for one of the three parameters with the others remaining static. This was done with the purpose of seeing the effect on the result, and try to find a good combination of parameter values giving as small of a mean squared error as possible. Testing a larger amount of parameters and the combination of them can be rather complex and is not the main point of the project, which meant only the most impactful parameters were chosen to be experimented with.

With an increasing number of trees in the forest, a higher accuracy in variable prediction can be achieved, but this also increases the running time of the algorithm significantly. Due to a lack of powerful hardware and this task not being one of the main prioritized ones, finding a parameter value large enough to keep the mean squared error down in a reasonable range while minimizing running time of the algorithm were the primary goals.

Similarly to the number of trees, the maximum tree depth is a parameter which can be used to balance the accuracy of the algorithm and its running time. Letting the maximum depth be too high a value can also introduce a large amount of overfitting, which could of course worsen the prediction of features, although increasing the number of trees could mitigate this overfitting. Testing of this parameter was once again done by running the algorithm with different maximum tree depths and comparing the running times and the mean squared errors of the results.

3.3 Testing feature relationships

A metadata outlier can be defined in several different ways. One of these ways is in variable correlations, which means deviations from the norm in how certain variables are related to each other. *Random Forest* regression was used to test and measure variable correlations in extensions, where each variable was predicted and the importance measure of each other variable was denoted. For each category of extensions, each metadata feature is tested in the *Random Forest* regressor by fitting the decision trees to the metadata followed by running the forest trying to predict the tested feature. In each test, the data set is split into a training set and an evaluation set where 85% of the data is chosen for the training set, and 15% for the evaluation set. The entries in each set are chosen at random each time. Each feature is tested a number of times and the importance of each other feature is denoted for each test, along with the mean-squared error for each test. For features in the metadata that are non-numerical, for instance the different permissions used, categorical variables are created. A CSV-file is created, where each row is one test, and each column is either the tested feature, the test number, the mean squared error of the test, or the importance of one other feature. A summary text file is also created for each category-run, which averages the importance of each feature over all tests, and lists them in order of most important to least important.

3.4 Testing the connection between categories and metadata

The point of this test is to investigate if there is any difference in extension metadata between categories, to see if the categories can be used for outlier analysis. For this task *Random Forest* classification is used, where each entry is fetched from the database and all extensions are split into a training and an evaluation set with categories used as labels. After training, the category of each entry in the evaluation set is predicted by the forest and classification error rate of all predictions is saved. Categorical features such as permissions and language are converted to categorical variables just as when using regression. For the splitting of the data set into a training and an evaluation set, 85% of the entries are put into the training set and 15% into the evaluation set. The entries in the sets are chosen at random each time. Two files are saved from the tests. The first one is a CSV-file containing all the tests, and the second one is a text-file containing a summary of the highest average importances among features over all the tests. Each row in the CSV-file is one test run, and each column is either the test number, the classification error or the importance of one feature.

3.5 Investigating patterns in permission usage using K-modes

K-modes clustering is used in this project in an attempt to detect patterns in permission usage among extensions, and to evaluate if these patterns relate to security risk. The algorithm was chosen due to it being well known, simple and efficient on huge amounts of data. As stated by Huang [19]: "The K-means algorithm is well known for its efficiency in clustering large data sets", which also applies to *K-modes* since it is *K-means* with slight changes made to it.

To solve this task, each extension in our database is converted into a vector of categorical variables, where each variable represents a single permission. Each variable has 2 possible values, 1 if the permission is used, and 0 if it is not. There are 113 different permissions in total. After each clustering, information about each cluster is stored in a text file, more precisely the amount of extensions in each cluster, the amount of extensions from the known malicious data set in each cluster and the amount of extensions using each permission in each cluster. The *K-modes* algorithm is run on all data vectors several times, each with a different k-value to determine which value of k is most appropriate. An appropriate k-value is a natural number as small as possible where the value above it does not create a new set of clusters distinctly different from the current set and where most of the malicious extensions are are concentrated in one or a few clusters if possible.

4
Results and Analysis

The results of the work performed in this project are displayed in this section. The results of each test will be explained in separate sections with an analysis of each result at the end of its respective section.

4.1 Datasets

The download was performed the 1st of March 2018 where 60228 out of the 63214 extensions from the `extension_ids.txt` file were successfully downloaded. Most of the missing extensions were not downloaded due to having been taken down. Along with these were some extensions with formatting errors due to mis-parsing the html-source of the web store page, but since these were very few in numbers we decided to let them slide.

Some of the metadata was gathered by parsing sequences of text from each extensions' web page in the Google Chrome Store. One problem was encountered with this method when extracting the parameter `language`, more precisely when the extension supported more than one language. This was the case since the languages were inserted with a javascript into the webpage instead of being included in the HTML page. To solve this problem, a new method of extracting the extension data would have been needed, hence it was decided to instead create a new language called `multiple languages`, assigned to each extension supporting more than one language.

Finding malicious extensions to test the clusterings with is difficult since Google removes them from the Chrome Web Store database upon discovery. Thus our method for downloading them were to first download all extensions from the Chrome Web Store and then at a later date mark extensions in our database as malicious based on report from Google highlighting removed extensions. This netted us 40 extensions in our database considered malicious by Google — an amount not even equal to 0.1% of our total number of extensions — meaning that our results won't be as statistically significant as we'd like. However, since another paper, namely Sarma et al. [5] that have done similar research with Androis apps, worked with a comparable ratio of malicious apps we don't see this as too large of an issue.

There's also a problem of diversity since these extensions were found within a very limited time frame of 3 months. In particular, the batch of extensions reported by Google was mostly focused on extensions "which have all been observed injecting click-fraud malware after installation" [20], meaning that all these malicious extensions may have ended up in the same group due to them doing similar things. There were a few extensions gathered outside of this batch reported by Andrey Meshkov, all of which being malicious copies of ad-block software [21].

4.2 Data analysis

An analysis has been performed on the metadata stored in the database, which informs us of the most popular permissions, the largest extension categories, the popularity of different file types and more.

As seen in figure 4.1, the most popular permission declared by Chrome extensions is `tabs`, closely followed by `all_urls`. The `tabs` permissions gives the extension access to the chrome.tabs API, which lets your extension access information about the tabs, communicate with the content scripts in web pages, or inject scripts into web pages, among other things.

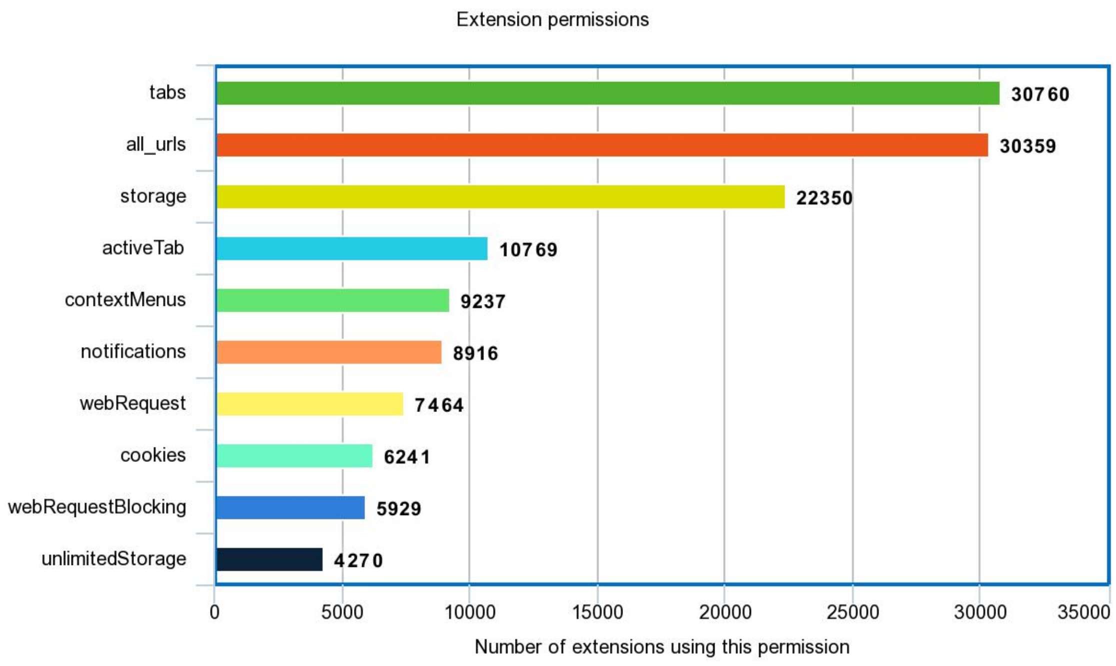

Figure 4.1: Bar chart of top 10 most used permissions

The `all_urls` permission lets an extension interact with the content of any web page, inject scripts, and send XMLHttp-requests to any web page, which makes this permission an extensive one. The fact that its the second most popular one seems worrying, since probably far from all extensions that declare this permission needs to be able to communicate and interact with all web pages. Interacting with the content of any web page seems very useful and probably necessary to a lot of extensions, but sending information to any web page on the other hand seems less useful. Especially combining these two functionalities under one permission seems unnecessary, since it also gives the extensions the ability to capture information about every different web page and send it to other web pages.

If the extension has declared both the tabs and the `all_urls permissions`, it also gets access to the `captureVisibleTab` functionality in the tabs API, which can take a screen shot of the visible content in the current active tab. In figure 4.1 we can also see that these two permissions are very close to each other in terms of usage, which probably means most extensions using one of these permissions are also using the other. Perhaps getting access to `captureVisibleTab` isn't the sole or biggest reason for so many extensions declaring both these permissions, but it seems like their usage numbers are too similar not to suspect it. Also, the reason behind needing to screen capture the visited websites is not clear to us.

`ActiveTab` is another popular permission and is a safer version of `all_urls`, where the extension declaring this permission is given the same rights as with `all_urls` except only when it's needed, in which case it prompts the user. This permission is clearly a much more user friendly version of `all_urls` which gives the user more insight in what the extension does and when it does it. It also prevents an extension from accessing important information and communicating with unwanted web pages without the users knowledge.

There are 11 different categories with varying popularity, which can be seen in figure 4.2. Some of the classes — especially some of the larger ones — are rather vague, like `productivity`, `accessibility` or `fun`. This is probably one of the reasons the categories are difficult to use in finding patterns in permissions and other metadata. The most supported languages can be seen in figure 4.3, where `English` is the most dominating one followed by `multiple languages`. The most widely used file types are displayed in figure 4.4, where the results are not very surprising either.

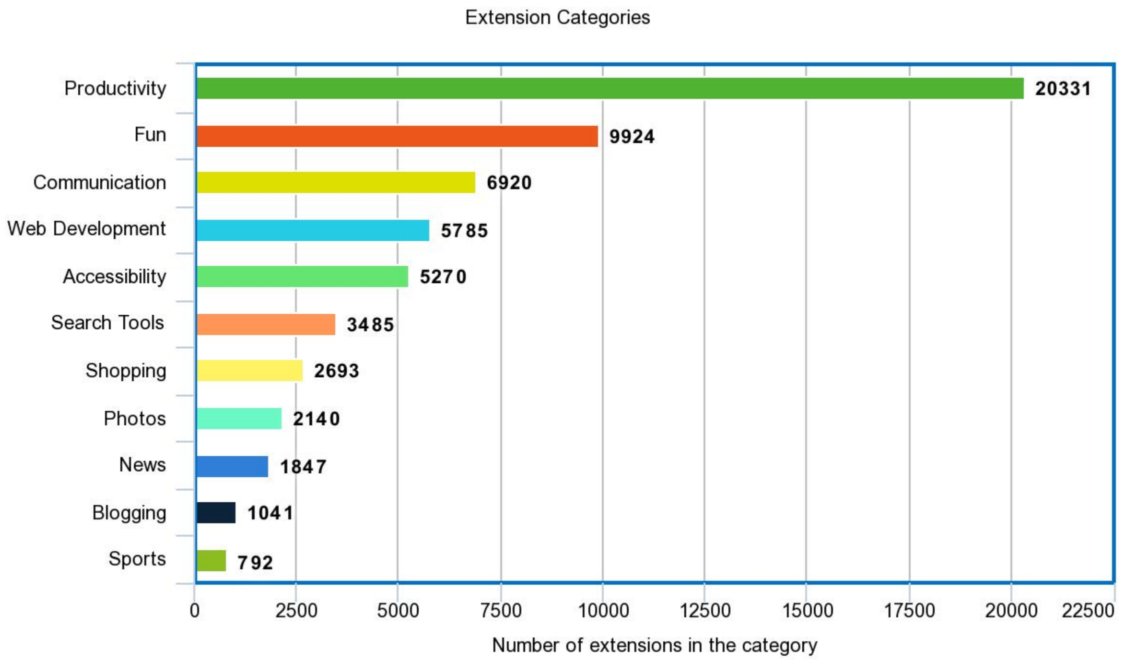

Figure 4.2: Bar chart of the different extension categories

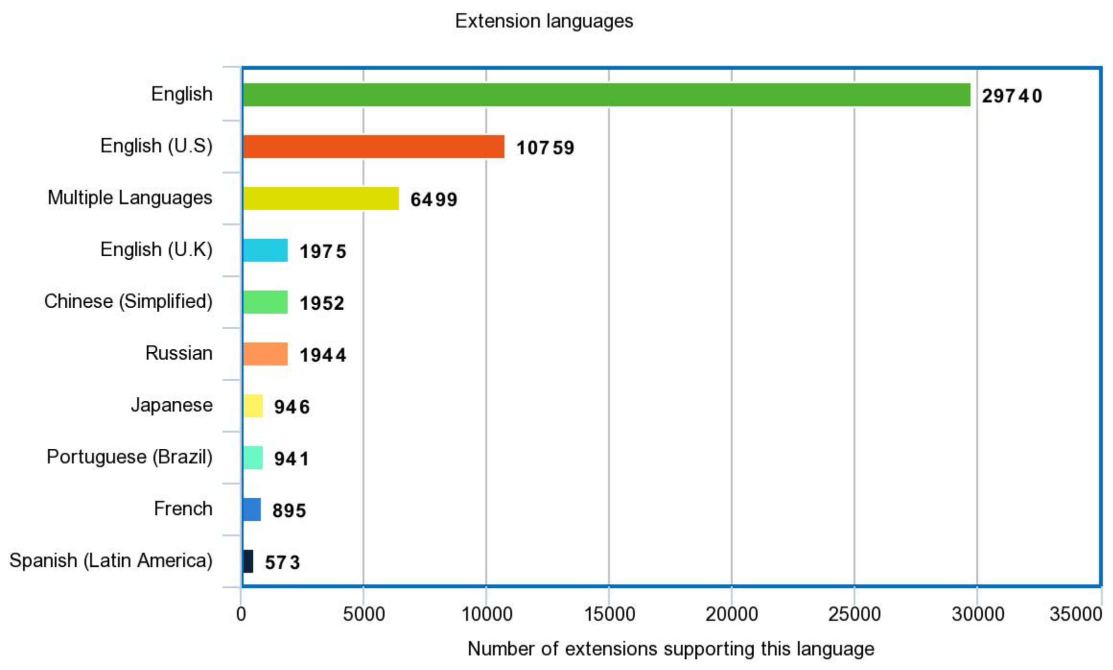

Figure 4.3: Bar chart of top 10 most supported languages

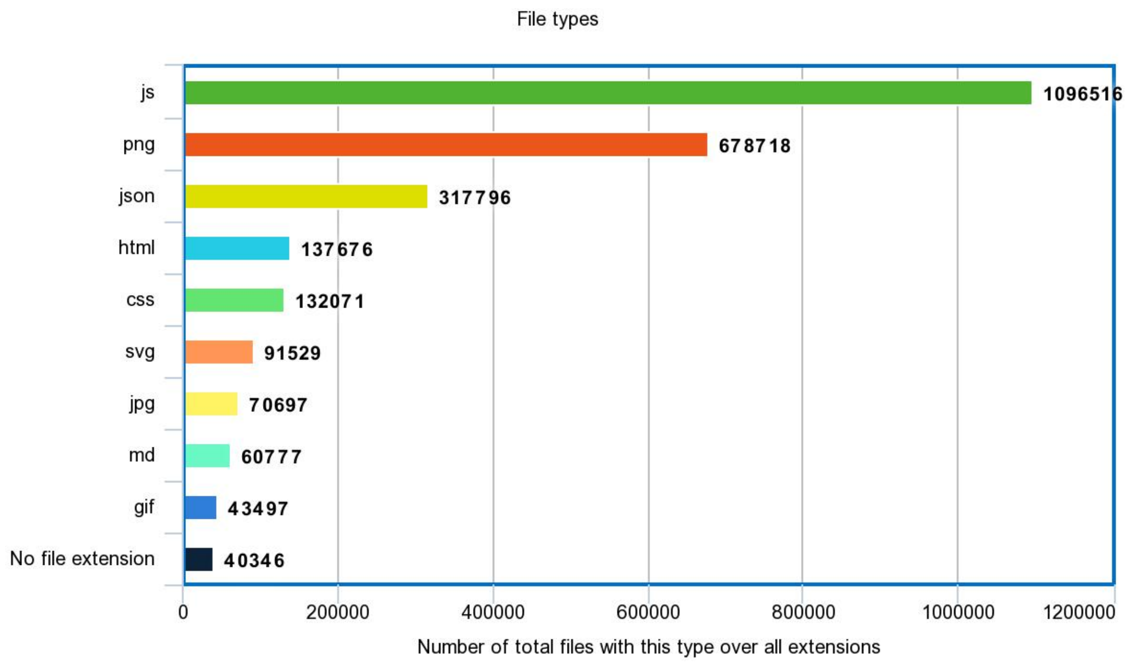

Figure 4.4: Bar chart of top 10 most used file types

4.3 Random Forest parameters

From visual inspection of the results, we concluded that the best parameters were a maximum tree depth of 16, a precision of 20 and 20 trees. With deeper trees than 16, there were no longer a notable drop in Mean Squared Error (MSE) from the earlier tests, and the same holds true for more than 20 trees. There was also a dip in MSE for several features between 10 and 20 precision, but it also leveled out for precisions above that.

4.4 Testing feature relationships

For most categories, we can see in table 4.1 that the total size of the extension is on average the most important feature in trying to predict the other features, which means it's the one that correlates the most to the other features. Another observation is that there seems to be no big differences in what features are important between the different category tests, which might suggest that feature relationships over different categories are rather similar and that testing the features by category is not necessary for outlier detection. As mentioned in section 4.3, tests were made with different values on the tree max depth, and what could be observed was when the trees grew deeper, `size` gained more importance, as some of the other features lost importance.

Category	**Feature / Importance (%)**	**Feature / Importance (%)**	**Feature / Importance (%)**	**Feature / Importance (%)**	**Feature / Importance (%)**
Productivity	Files / 9.5	Size / 9.5	Users / 6.4	.js / 5.5	Raters / 5.2
Fun	Size / 9.7	Files / 8.4	.js / 7.1	Users / 6.0	.png / 5.1
Communication	Size / 11.0	Files / 8.6	Users / 6.8	.js / 6.6	Raters / 6.0
Web Development	Size / 10.8	Files / 7.7	Users / 7.5	.js / 5.8	.png / 5.5
Accessibility	Size / 10.6	Files / 8.7	Users / 6.9	.js / 5.6	Raters / 5.6
Search Tools	Size / 10.0	Files / 7.8	.js / 6.9	Users / 6.8	.png / 4.2
Shopping	Size / 10.5	Files / 9.4	Users / 7.1	Raters / 5.7	.js / 5.5
Photos	Size / 7.5	Files / 6.6	.png / 5.9	.js / 5.4	.jpg / 4.8
News	Size / 10.2	Files / 8.9	Users / 6.9	.js / 5.9	Raters / 5.7
Blogging	Size / 9.7	Files / 8.6	Users / 8.1	.js / 6.4	Raters / 5.5
Sports	Size / 9.2	Files / 8.6	Raters / 6.0	Users / 5.4	.js / 5.3

Table 4.1: The features with the highest average importance in *Random Forest* regressor tests in each category.

There were some notable feature relationships in testing with the *Random Forest* regressor, which are displayed in table 4.2. There is a connection between users and raters which can be seen in figure 4.5, where extensions with fewer users usually have fewer raters, which seem rather intuitive. The strongest connection is for `raters` when predicting `rating`, which is displayed in figure 4.6. This connection is not at all as strong the other way around, this since `users` were more important than `rating`. Several of the top variable connections are rather trivial, which is why they are not displayed in figures, but they will be mentioned. Permissions `webRequest` and `webRequestBlocking` are only used by a vast minority of extensions of which most declare both, only a few declare just `webRequest`, and very few to none declare only `webRequestBlocking`. This is visualized in figure 4.7. `webRequest` is a permission used by extensions which desire to analyze traffic, intercept or modify requests. `webRequestBlocking` is used to also be able to block requests.

Predicted feature	Highest importance feature	Average importance (%)
rating	raters	98.7
permission__webRequestBlocking	permission__webRequest	94.2
permission__webRequest	permission__webRequestBlocking	93.5
.woff	.eot	78.7
.eot	.woff	78.4
permission__proxy	.pac	77.3
Multiple__languages	.json	74.8
.pac	permission__proxy	74.2
phps	php	74.0
permission__browser	permission__certificateProvider	73.0
permission__clipboardWrite	permission__clipboardRead	72.1
.dec	size	72.0
permission__identity	permission__gcm	71.6
.Articles	.Assist	71.0
users	raters	68.4
permission__gcm	permission__identity	68.1
permission__dataReductionProxy	permission__preferencesPrivate	68.0
permission__certificateProvider	permission__browser	66.0
English (U.S)	English	65.9
.bcmap	.properties	65.2

Table 4.2: The 20 strongest average feature relationships

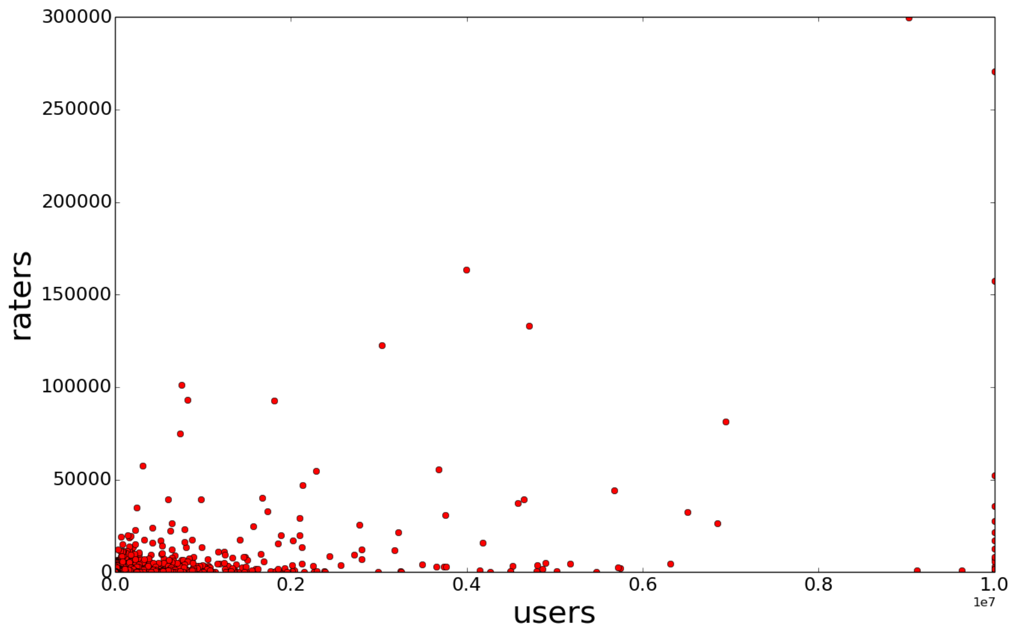

Figure 4.5: The relationship between users and raters.

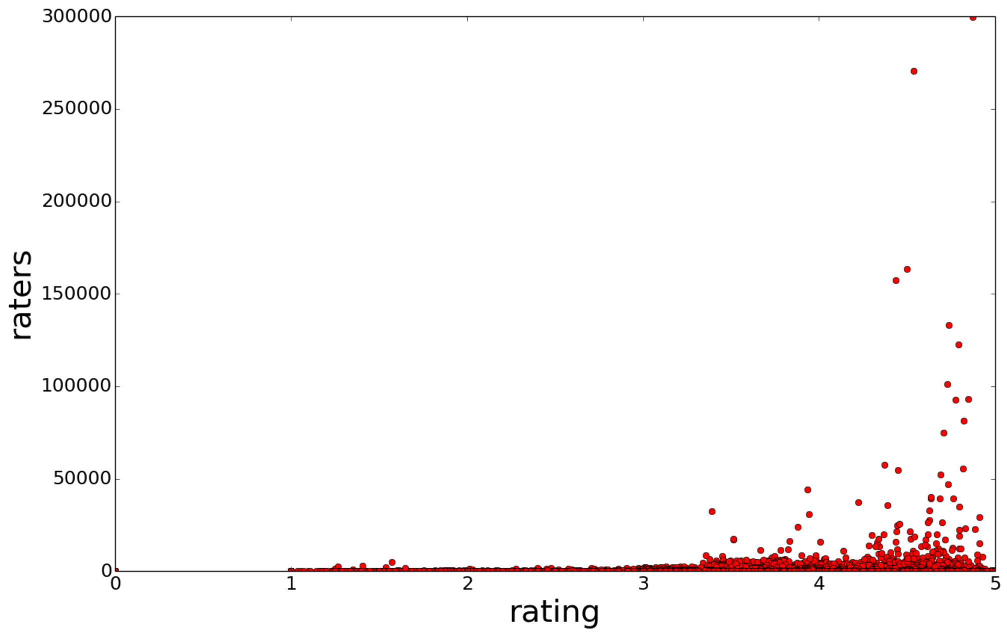

Figure 4.6: The relationship between rating and raters.

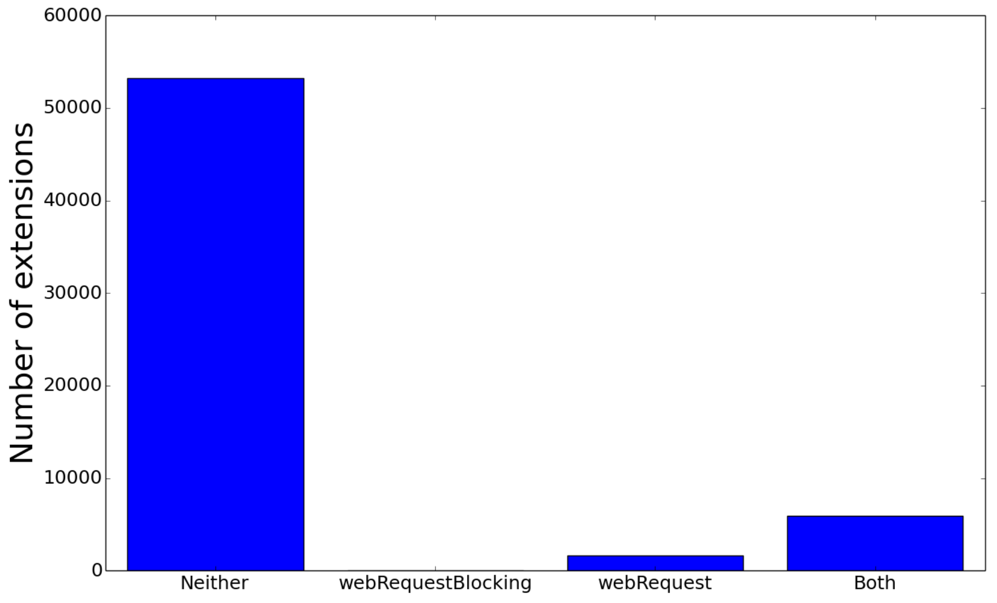

Figure 4.7: The relationship between permissions webRequest and webRequest-Blocking.

As can be seen in figure 4.8, there is partly a linear relationship between the file types `.woff` and `.eot`. Both file types are font files which are used by different browsers, `eot` is a font file format developed by Microsoft and primarily used by

Internet Explorer. `woff` is a fairly new font format which is recommended by the World Wide Web Consortium [22] and is used by Mozilla Firefox and Google Chrome among others. The seemingly linear relationship between them seems fairly obvious, for a web page to support the use of different browsers, both file formats needs to be available.

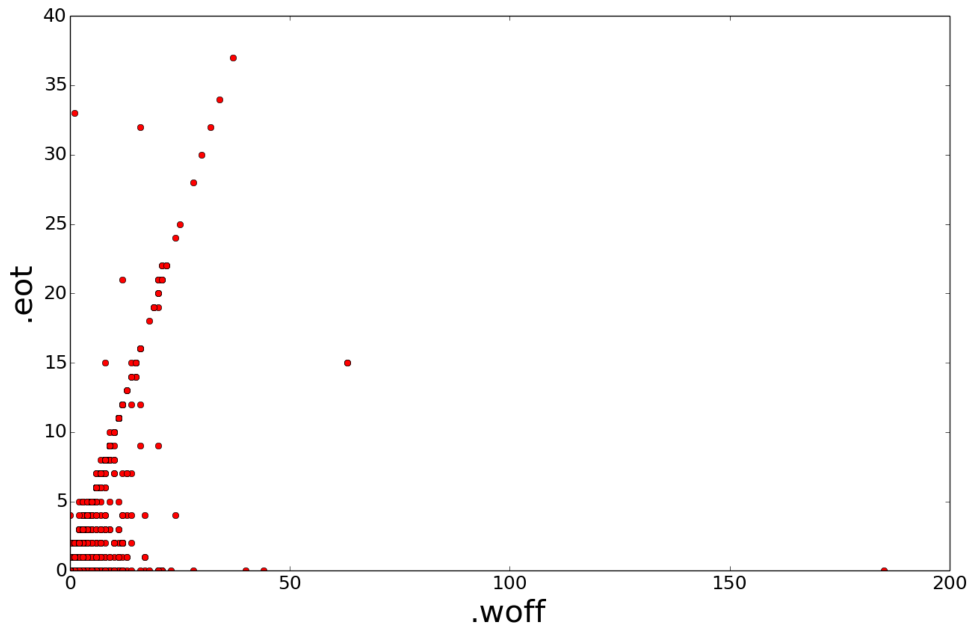

Figure 4.8: The relationship between file types .woff and .eot

There is a somewhat noteworthy relationship between the language `multiple languages` and the file type `.json`, which can be seen in figure 4.9. What can be noted is that extensions not declaring `multiple languages` have a larger spread of how many `.json` files they include, where those that do declare `multiple languages` are more concentrated around a lower amount of `.json` files. The reason behind this relationship is not known, and its relevance is uncertain, but since the connection is among the stronger ones and there is something to be noted from visual inspection, it was included as a figure.

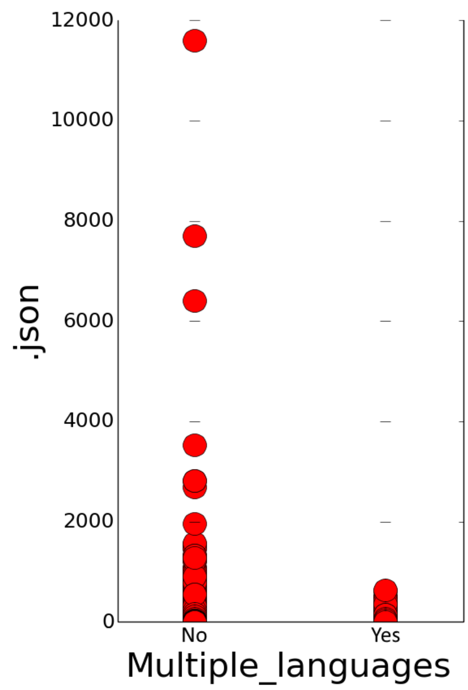

Figure 4.9: The relationship between the language multiple_languages and the amount of .json files in an extension.

The rest of the feature relationships listed in table 4.2, not shown in plots or bar charts in this section, are excluded for being very obvious or not really displaying anything useful. To give some examples, the permissions `browser` and `certificateProvider` are almost never used, and the same goes for `dataReductionProxy` and `preferencesPrivate`. The plots of `permission_proxy` and `.pac`, and `.Articles` and `.Assist` displayed no clear patterns from visual inspection and it was difficult to even extract any relevant information from the plots which is why they were left out.

Since many of the feature relationships are fairly obvious and not very closely related to security, it seems clear to us that for example finding an extension outlier in terms of the relationship between different font file types is not very effective for finding security risks. There are however a few relationships which could provide the user with relevant information. An example of this could be `rating` and `raters`. According to figure 4.6, an extension having many `raters` and a low `rating` would be considered an outlier and would probably also be unusual in a negative way, which could warn the user not to install it. An outlier could also be an extension with fairly high `rating` and very many `raters` which could be either positive or negative, either a well liked extension by many users or a developer with malicious intents faking reviews to get more downloads. These things could contribute to the user making a more informed decision.

4.5 Testing the connection between categories and metadata

The *Random Forest* classification was run 10 times with the categories as labels. For each of the tests, the classification error was between 56% and 58%. These results further strengthens the claim in section 4.4, which is that classes are almost indistinguishable from metadata alone. It is obvious that the idea behind the categories is not grouping extensions together based on close relations in metadata, but instead after their usage areas. The categories being very similar in terms of metadata makes finding outliers in a specific category difficult. Instead, finding outliers in terms of variable correlations seems more useful, since there are multiple strong correlations. The counterpoint to this however would again be that many of the strongest variable correlations does not have strong parallels to security.

4.6 Investigating patterns in permission usage using K-modes

Table 4.3 shows the resulting groups and distribution of extensions for the *K-modes* algorithm run with k-values from 5 to 10 where one of these groupings, namely the result with a k-value of 7, is shown in greater detail in table 4.4. The result shows that 85% or more of all malicious extensions are, with the *K-modes* algorithm, consistently put into one or two groups with a total group size of less than 10% of the total amount of extensions (k-values of 6 and 7 puts the malicious extensions in group 2 and other k-values divides them up into two groups, a larger one and a smaller one. This suggests that there exists a common pattern most malicious extensions in our dataset adhere to.

k-value:	**5**	**6**	**7**	**8**	**9**	**10**
Group 0:	21384 (0)	44771 (1)	32943 (0)	24351 (1)	37614 (0)	28160 (0)
Group 1:	14352 (1)	7370 (0)	7458 (0)	20690 (0)	7364 (0)	10038 (0)
Group 2:	12266 (23)	4402 (37)	6764 (40)	7468 (0)	4904 (0)	9460 (0)
Group 3:	7422 (0)	1812 (1)	6519 (0)	2850 (9)	3682 (28)	3217 (34)
Group 4:	4804 (16)	1227 (1)	4086 (0)	2399 (25)	1519 (0)	2213 (1)
Group 5:	-	646 (0)	1720 (0)	1510 (5)	1388 (0)	2170 (0)
Group 6:	-	-	738 (0)	646 (0)	1346 (10)	1666 (5)
Group 7:	-	-	-	314 (0)	1213 (2)	1472 (0)
Group 8:	-	-	-	-	1198 (0)	1158 (0)
Group 9:	-	-	-	-	-	674 (0)

Table 4.3: The result of the *K-modes* algorithm runs with k-values from 5 to 10. Each cell shows the number of extensions in each group for each k-value with the number of malicious extensions in the parenthesis.

Group Number:	0	1	2	3	4	5	6
Number of Extensions:	32943	7458	6764	6519	4086	1720	738
Malicious Extensions:	0	0	40	0	0	0	0
tabs	11423	7458	5667	1622	2887	1531	95
<all_urls>	1618	1475	2822	518	435	404	18
storage	4472	7458	4912	1601	2743	1087	16
activeTab	0	1891	1203	6519	677	459	0
contextMenus	2915	1487	2309	774	0	1720	7
notifications	927	0	2025	108	4086	1720	6
webRequest	124	0	6749	187	242	142	6
cookies	974	1136	2734	285	555	539	3
webRequestBlocking	1	0	5859	47	6	5	5
unlimitedStorage	582	598	2257	119	310	388	3
webNavigation	529	619	2371	172	228	193	4
management	519	396	1711	46	244	168	7
alarms	424	423	633	107	653	185	1
background	595	321	381	164	499	191	4
topSites	260	155	1223	26	36	67	0
identity	261	315	366	155	527	134	0
bookmarks	582	258	568	75	97	98	0
clipboardWrite	499	264	241	191	115	220	3
downloads	319	231	341	134	106	102	1
history	344	224	404	49	80	79	0
desktopCapture	0	69	41	57	48	21	738
nativeMessaging	337	69	324	46	33	35	6
geolocation	229	75	254	49	79	45	1
idle	96	58	248	16	137	142	3
gcm	8	16	97	8	457	100	0

Table 4.4: The result of the *K-modes* algorithm run with 113 different permissions and a k-value of 7, where only the top 25 permissions are shown in this table.

There are a few observations to be made and a few conclusions to be drawn from these results. Examples for this analysis will be drawn from the clustering of the data with a k-value of 7 — the results of which can be seen in table 4.4 — unless stated otherwise. First off, the clusters are of varying sizes, the largest one about the size of half of all the extensions in the database. The rest of the clusters are rather close to each other in size, all being significantly smaller than a third of the largest cluster. The argument can therefore be made the largest cluster represents the norm of permission usage among extensions, and the smaller groups various outliers from that norm. The same pattern can be seen from the clusterings with other k-values in table 4.3, where one or two clusters are significantly larger than the rest. According to the results of the studies from Bloedorn et al. [6] which studied internet connections, it was concluded that most of the time malicious connections were also outliers. Using this argument, extensions placed in smaller clusters pose a larger threat of the reason that they are deviating from the norm.

An interesting observation from table 4.4 is that 99% of all extensions using the `webRequestBlocking` permission — a very suspicious permission that allows the extension to block http-requests sent by the user — is put in the group with all the malicious extensions. This could suggest that the *K-modes* algorithm found a clear divide between safe and potentially malicious extensions with this permission being a great main identifier for them. However, an article written by Chia et al. [23] concludes that "we regard all Chrome permissions to be both dangerous and dangerous-and-information-relevant, except the permission 'Your tabs and browsing activity'" suggesting that it's the specific combination of multiple permissions that defines the group rather than a specific permission. This would also explain why there isn't any other obvious permission distributions in our results, but the correlation is certainly striking. The permissions `tabs`, `<all_urls>`, `storage` and `cookies` are usually among the most popular as well in that high risk group. The only other one is the heavy distribution of the `desktopCapture` permission — a permission that allows an extension to capture content of the screen, individual windows or tabs — into one group, suggesting for there to be another high-risk group. This group has no malicious extensions put in it, but that might be a result of the issue discussed in section 4.1. Also, outside `desktopCapture` the permission usage in this group is extremely low, making the potential risks of these extensions very narrow. The largest risk considered by us for this group is an extension spying on a user, capturing the content of their computer and their actions. Without the permissions of accessing arbitrary web servers though, the ability to take screen shots and sending them to an attackers server is non-existent without help of outside applications or other extensions, minimizing the risk of these extensions in a vacuum.

There are differences in permission usage between clusters, both in terms of which permissions are being used the most and the rate of how often they are used among extensions. Looking at the results from clusterings of all the different k-values, extensions in the largest cluster tends to use permissions less often than in most others, and since according to Chia et al. [23] most permissions can be considered at a similar level of risk, using more permissions implies a higher risk.

5 Discussion

In this chapter we discuss some issues with the methods used and assumptions made. We also talk generally about other methods of possibly improving security of browser extensions in Chrome or the communication with a user.

5.1 Correlation bias

One thing that might have skewed the result in our *Random Forest* tests, is the phenomenon described by Toloşi and Lengauer [24] which they call *correlation bias.* This is when doing regression analysis and looking for variable importance, smaller groups of variables that correlate to the main variable get favored over larger groups that have equal correlation. This effect could to an extent explain the high average importance of the single value variables over all category tests, and the low average importance for the larger categorical variables.

5.2 Outliers compared to malicious extensions

There is the question about how malicious extensions compare to outliers, and how big of an overlap these two sets have. It is not clear exactly how effective the method of finding outliers to also find dangerous extensions is, and to know for certain, the project would have needed access to a larger set of known malicious extensions. In that case, the results from the outlier analysis using *K-modes* could have been compared to the known malicious extensions more reliably than in the current analysis. What the outlier analysis does well however, is inform a user of irregularities in a certain extension, and what makes it differ from the norm. This helps the user make a more informed decision about installing odd extensions, which use more permissions than they might need to or are larger in size than what they probably need to be.

The work done by Bloedorn et al. [6] — previously mentioned in section 1.4— was performed on network connections and showed that a vast majority of malicious connections were also detected as anomalies by the clustering algorithm *K-means.* Although network connections are not the same as browser extension metadata, this still gives some weight to the argument that malicious entities, for example network connections or browser extensions, usually deviate from the norm in some or several ways.

5.3 Improving security in browser extensions

There are other approaches to securing the browser from malware extensions other than statistical analysis, with varying degrees of complexity and effectiveness. Analyzing code is one of them, which means searching source code for dangerous code snippets. This is a method used in popular anti virus software [25] and it is much more accurate in detecting known exploits and potentially dangerous code than statistical analysis. One large flaw of this method however is that its effectiveness decreases over time, along with new exploits being discovered and potentially new flaws being introduced as the way of coding and software in general change and evolve. To be effective in detecting a certain flaw or exploit, the exploit needs to be known by anti virus software developers, which usually happens an arbitrary amount of time after it is discovered by an attacker. These reasons put code analysis always a step behind attackers, and in constant need of updating to not lose effectiveness.

Risk-ranking permissions and displaying a risk score for an extension to the user at install time is an approach that seems natural and effective, but has been proven to be quite the contrary. In work done by Felt et al. [26] which analyzed permissions used by Android applications and Chrome extensions, it was found that most applications and extensions trigger similar warnings at install time, which makes the warnings lose their effectiveness rather fast. This suggests assessing risk of an application in comparison to other applications is a more effective method, since it causes fewer prompts and gives a relative risk assessment.

Recognizing dangerous behaviour from extensions and blocking it or alerting the user could be a method of preventing extensions from performing malicious tasks. This could be an extension trying to perform a risky sequence of actions, for example access sensitive information such as cookies and then sending information to an untrusted server. This method in combination with a system of marking trusted web-sites which you trust with your data for example, could form a powerful tool in preventing extensions doing dangerous things with the amount of pop-ups to the user minimized. Rules could be defined of such dangerous action combinations which when performed by extensions, needs the users permission to do so and could be given for example in a pop-up window. This approach suffers from some of the same problems as analyzing code, which is that the exploits needs to be publicly known to be able to protect the browser from them, and also that the countermeasures needs to constantly be updated to not lose effectiveness.

Something which could help users gain insight into the need for applications to declare certain permissions is if it was explained by the developer in some manner why their application is in need of the declared permissions. An example of this could be natural language motivations in the manifest file for the applications next to each permission, which could be displayed to the user at install time. A missing or poor motivation would alert the user that the app is over-privileged or that the developer is trying to get away with something suspicious. The downside to this approach is that installing an application might require reading a lot of text, which is commonly something most users are not willing to do, a good example of this being long terms and conditions when using an internet service. It is most likely also a lot less useful for users with low technical knowledge, since they might not have enough insight into how the system works and if the developers motivations makes sense. Developers also still have the possibility of being dishonest about their intention of their permission-usage, and ill-intended developers with high creativity and imagination wouldn't suffer very large consequences from such a system.

6
Conclusion

This project has analyzed metadata of browser extensions in Google Chrome in a few distinct steps. First, all extensions in the Google Chrome store were downloaded, metadata was extracted from them and stored in a database. Some statistic analysis was performed on the metadata, researching the most popular categories, permissions, file types and more. *Random forest* regression was used to find strong feature relationships, and *Random forest* classification was used to determine the differences between the categories in terms of metadata. *K-modes* clustering was finally run on the permissions used by extensions, in an attempt to find patterns in permission usage and use those patterns to assess the risk of installing particular extensions.

As results in section 4.4 shows, average variable importances over all the different categories when doing regression testing of features were very similar, introducing the idea that in regards to metadata, there is no huge difference in variable relationships between the categories. This was further backed by the results found in classification testing between categories, which can be found in section 4.5. These tests displays a very large classification error between the categories, again suggesting that the difference in metadata between them are minimal. It was also noted in both the regression and classification tests that permissions were generally low in importance, meaning that what rather ineffective predictions could be made about features or the category of extensions, the permissions used by them were not very influential of the result. This might not be very surprising, due to extension categories not being a product of grouping with respect to security risk ranking or similar metadata patterns, but rather use cases of the extensions. However, the fact that categories have little to no connection to security level and that the prompting of permissions used at install time being a poor method of risk communication, we find a new system would need to be put in place if users being under-informed about security risks in this area is seen as a problem.

The results from the *K-modes* clustering on extensions based on permissions showed promise that security categories for extensions based on this clustering would be a good idea when it comes to communicating risk to a user. The *K-modes* algorithm managed to detect clear patterns in permission usage among extensions and even managed to consistently put the vast majority of the extensions from our known malicious extensions data set into one or two clusters for k-values of 5 and above. As mentioned in section 4.1 however, our malicious data set is very small in comparison to all extensions tested and the set lacks diversity, so this result in particular is unfortunately not statistically significant for malicious extensions in general. Ignoring the malicious data set, the created clusters shows consistent patterns over k-values above 5, where different clusters can be assessed to be on different levels of risk as shown in section 4.6. Communicating this risk of extensions relative to other extensions in the form of a fairly low number of security groups, each group with a rather distinct pattern in permission usage, we believe to be both a clear and effective way to alert a user of the implications of installing an extension, where extensive technical knowledge is not needed.

6.1 Future Work

Our work has some notable flaws which could be improved on in future work, largely caused by a lack of resources. One of the major ones is not enough quantity and diversity within the set of known malicious extensions. The results from the *K-modes* clustering shows the existence of clear patterns in permission usage between these extensions and that grouping the malicious extensions together with unsupervised learning is both possible and highly probable. Because of the previously mentioned shortcomings however, these results can not be expected to represent potential clustering results of all malicious extensions. It would therefore be very insightful to do a similar study with a more extensive data set of known malicious extensions to more extensively evaluate this method in assessing risk and how accurate it is when it comes to predicting bad extensions.

The introduction of the precision parameter into our scripts was a method of improving the performance of the scripts, making them runnable on our machines within a reasonable time span while trying to lose minimal precision in our tests. With better hardware the same tests could probably be run without cutting out any features. Seeing if that change increases the precision of results or instead increases over-fitting would be both useful and interesting.

Bibliography

[1] The most popular browsers. https://www.w3schools.com/browsers/, 2017.

[2] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. Trends and lessons from three years fighting malicious extensions. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 579–593, Berkeley, CA, USA, 2015. USENIX Association.

[3] Sara Motiee, Kirstie Hawkey, and Konstantin Beznosov. Do windows users follow the principle of least privilege?: Investigating user account control practices. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS '10, pages 1:1–1:13, New York, NY, USA, 2010. ACM.

[4] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 241–252, New York, NY, USA, 2012. ACM.

[5] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android permissions: A perspective combining risks and benefits. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 13–22, New York, NY, USA, 2012. ACM.

[6] Eric Bloedorn, Lisa Talbot, and Dave Debarr. Data mining applied to intrusion detection: Mitre experiences. pages 65–88. Springer, 01 2006.

[7] Lei Liu, Xinwen Zhang, Guanhua Yan, and Songqing Chen. Chrome extensions: Threat analysis and countermeasures. 04 2018.

[8] Justin Warner Spencer Walden. Functions in chrome. https://www.google.com/chrome/browser/features.htmlsecurity.

[9] Google. chrome.permissions. https://developer.chrome.com/apps/permissions.

[10] Google. Permission warnings. https://developer.chrome.com/apps/permission_warnings.

[11] Sruthi Bandhakavi, Nandit Tiku, Wyatt Pittman, Samuel T. King, P. Madhusudan, and Marianne Winslett. Vetting browser extensions for security vulnerabilities with vex. *Commun. ACM*, 54(9):91–99, September 2011.

[12] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting browsers from extension vulnerabilities. In *NDSS*, 2010.

[13] Leyla Bilge and Tudor Dumitraş. Before we knew it: An empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 833–844, New York, NY, USA, 2012. ACM.

[14] S. Ford, M. Cova, C. Kruegel, and G. Vigna. Analyzing and detecting malicious flash advertisements. In *2009 Annual Computer Security Applications Conference*, pages 363–372, Dec 2009.

[15] Google. What are extensions? https://developer.chrome.com/extensions.

[16] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[17] Hugo Steinhaus. Sur la division des corps matériels en parties. Bull. Acad. Pol. Sci., Cl. III 4, 1957.

[18] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[19] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery*, 2(3):283–304, 1998.

[20] Google. More extensions, more money, more problems. https://www.icebrg.io/blog/more-extensions-more-money-more-problems, 2018.

[21] Andrey Meshkov. Over 20,000,000 of chrome users are victims of fake ad blockers. https://adguard.com/en/blog/over-20-000-000-of-chrome-users-are-victims-of-fake-ad-blockers/, 2018.

[22] J Kew, T Leming, and E van Blokland. Woff file format 1.0. *World Wide Web Consortium. www. w3. org/TR/WOFF/. Accessed*, 7, 2011.

[23] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. Is this app safe?: A large scale study on application permissions and risk signals. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 311–320, New York, NY, USA, 2012. ACM.

[24] Thomas Lengauer Laura Toloşi. Classification with correlated features: unreliability of feature ranking and solutions. volume 27, pages 1986–1994, July 2011.

[25] Sarika Choudhary. How anti-virus software works?? 3:483–484, 04 2013.

[26] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development*, pages 7–7, 2011.