CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Predicting Security of Browser Extensions Using Machine Learning

Master's thesis in Computer science and engineering

ALEXANDER SELMANOVIC & CAMILLA SÖDERLUND

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Predicting Security of Browser Extensions Using Machine Learning

ALEXANDER SELMANOVIC & CAMILLA SÖDERLUND

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Predicting Security of Browser Extensions Using Machine Learning
ALEXANDER SELMANOVIC & CAMILLA SÖDERLUND

Supervisor: Andrei Sabelfeld, Department of Computer Science and Engineering
Advisor: Martin Altenstedt, Omegapoint Göteborg AB
Examiner: Johan Karlsson, Department of Computer Science and Engineering

Master's Thesis 2022
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 2022

Predicting Security of Browser Extensions Using Machine Learning
ALEXANDER SELMANOVIC & CAMILLA SÖDERLUND
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Combining machine learning and cyber security can have great results in finding malicious web extensions and how those extensions exploit their users. In this thesis, a method for discovering extensions that are attacking their users with the help of *Query Stealing* attacks is proposed. With the help of machine learning, the suggested method is to build a model that with high accuracy predicts whether an extension is attacking its users or not based on static information found in both the manifest of the extensions and in the presence of predefined Keywords. The model is trained with previously gathered data that contains labels on extensions that categorise extensions as either malicious or safe. This data is suitable for a model taught using supervised machine learning. The final model achieves an F1 score of 0.9651, which indicates that the risk of an extension being misclassified is low. The model was then used to predict labels of new unlabelled extensions.

Keywords: Machine Learning, Cyber Security, Browser Extension, Random Forest Classifier, RFC.

Acknowledgements

We want to thank our supervisors, Andrei Sabelfeld and Martin Altenstedt, for their help with the work behind this thesis. Our opponents, Theodor Angergård and Tobias Karlsson, also deserve to be acknowledged for giving their outside perspective on this thesis and providing interesting discussions. Additional thanks to Benjamin Eriksson and Pablo Picazo-Sanchez for the time investment and thoughtful insights within the fields.

Alexander Selmanovic & Camilla Söderlund, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1
Introduction

The internet is a handy tool that is taking an increasingly large part of our everyday life [1]. As a help to navigate and manage the vast sea of information that is the internet, most web browsers have plugins or extensions that aid the users along the way. Extensions are available in a wide range of categories, from blocking advertisements and browser cookies, to keeping track of passwords. Around 450 new extensions are added daily to the Chrome Web Store [2]. These extensions can be created and uploaded by anyone with interest, meaning that the upload option is not locked to the companies behind the browsers [2]. However, when anyone has the option to create extensions, the risk of some of the extensions being malicious increases since there will always be someone trying to exploit the system and attack the users [3].

Malicious extensions can create severe problems for the affected users that install and use them. An issue that exists in some browser extensions is a cyber attack called *phishing*, where the extension extracts vulnerable information from the user [4]. The extracted information can be something as simple as usernames and email addresses, but could range all the way to application passwords and banking information.

Another common way in which a malicious extension could create problems for its users is by stealing queries that users of the extension send to their search engines. The extracted data from the queries can later be sold to various advertising companies and services without the user's knowledge. This kind of attack is called *Query Stealing* [5] and is not uncommon in the extension type *New Tab* [5] that was investigated in this thesis.

1.1 Problem

Since the damage that malicious extensions can be extensive, users should be careful when downloading and installing new applications on their browsers and computers. The two main methods used to find malicious extensions are either by trusting the Chrome Web Store to remove any malicious extensions it discovers, or by installing a malware detection system that scans for threats. However, these systems are not enough to catch all the malicious extensions since some still get through the security checks, meaning that there exists a need for updating the ways that malicious extensions are found [6].

The first method of evaluating the extensions is done behind the scenes by the browsers' own checks before the extensions become available for download. How this evaluation is done is not available to the public. Since, as previously mentioned, there exist malicious extensions that are available for download, e.g. from the Chrome Web Store, it would indicate that this method is not complete and is not alone enough to keep the end-users safe [6].

The other way of evaluating the security of browser extensions is by installing malware detection extensions that uncover security threats [7]. However, this kind of protection can only help find threats when they are already present in the system and are unable to prevent security intrusions before they have a chance of afflicting the user. This means that the users are vulnerable to attacks in the time it takes the malware detection program to find the security threats.

Using both the checks for malicious extensions implemented by the Chrome Web Store and a malware detection program add strong protection for the users. But, there still exists a need for improvement in these fields in order to protect the users from malicious software.
Users also have the option to themselves review the extensions. However, this option requires significant knowledge of the subject alongside a considerable amount of time for the code review. Even experienced security engineers are not able to cover more than 1000 lines in one hour, meaning that this option might not even be realistic for the trained professional eye [8]. Besides being time-consuming, manual code reviews risk not catching all the potentially malicious code. The developers could obfuscate the code and make it harder to find when not paying close attention [9].

1.2 Goals and Challenges

This thesis aims to research and develop a method that uses machine learning algorithms to classify and evaluate how safe browser extensions are with the help of different values. This method could theoretically be used as a tool that functions as a trusted third party with the ability to evaluate browser extensions before they are installed. It could also be used as an addition to the current browser's vulnerability checks. To reach this goal, previously gathered data from different extensions was be used. With the data available, a machine learning algorithm can predict whether an extension should be considered safe for the user or not.

Deciding which values that were to be taken into account in the training of the machine learning model was a critical factor since the values greatly affected which results were obtained. Suppose the values that were focused on were less important in the current context. In that case, the model could be misled and significantly impact the reliability of the security prediction of the evaluated extensions. The values that the model was trained with was gathered from either various fields in

the manifest file of the extensions or from the APIs, methods and function properties, which collectively were called *Keywords*.

Since several machine learning algorithms are available to work with, a challenge was to explore and find which algorithm was best suited for the stated goal. The algorithms were tested with the same data to find if an individual algorithm was best to use or if perhaps a combination of multiple algorithms could produce better results. In order to find the best model/models, the optimal parameters were explored.
With goals and challenges introduced, the final goal of the thesis was able to be formulated as:

Develop a method that using machine learning is able to classify whether a New Tab extension is attacking users using Query Stealing attacks or not based on the fields in the manifest file and the presence of Keywords in the code.

1.3 Limitations

There are many kinds of extensions available for users on different web browsers. Since this thesis had a limited research time, the focus was on extensions available for download on the Chrome Web Store, which still was a large amount of extensions. To limit the thesis further, the focus was only on New Tab extensions labelled as either malicious or safe in a data set created by Benjamin Eriksson, Pablo Picazo-Sanches and Andrei Sabelfeld in their unpublished paper [5].

The optimal goal would be to have a method that is able to predict the safety of any extension in any category and on any browser. However, since many of the papers previously written on this topic were created over a span of multiple years, it was be unlikely to achieve this goal within the time frame of this thesis.

1.4 Contributions

This thesis developed a method for using a combination of machine learning and static code analysis to search for patterns that can be used to discover extensions attacking users with Query Stealing attacks. This method is a proof of concept stating that since it is possible to distinguish extensions that contained Query Stealing attacks from those that did not. It should also be possible to create models that can do the same for any type of attack.

The Chrome Web Store could integrate the model created in this thesis into their already existing malware detection programs. This way, the users of the extensions downloaded from the Chrome Web Store could feel safer with fewer malicious extensions available for download.

Another alternative use of the created model is to create an open-source program or extension that can take any extension as input and give a prediction to the user whether the extension is safe to use or not. Since the code behind the model might not be intuitive for anyone, a graphical interface would be needed to simplify the prediction process.

A third way the model could be utilised could be inside a trusted third party that verifies the safety of extensions. With the help of a third-party safety verifier, the dependence on the Chrome Web Store would decrease, while on the other hand, a new party would have to be trusted.

1.5 Risk Analysis

When working with machine learning, evaluating any eventual risks that could arise from the created algorithm is important. In the case of this thesis, something that was considered with great care was that the data set used for the training and validation was not biased. If the training was done on only one type of data, the model would not perform as well as intended when new data was introduced, e.g. in the validation of the model. In general, when working with machine learning, it is vital to have data from various sources with different structures to avoid bias and ensure that the algorithm has enough material to produce reliable results.

Another risk that should be considered when creating an algorithm that finds malicious browser extensions is that too much trust and reliability in the model is created. If the model produces inaccurate results, the users of the algorithm might believe that they are safe when in fact, a malicious program is making problems in the background.

Putting too much trust into machine learning models happened to Google in 2021 when they used machine learning to remove malicious applications from the Android Play Store [10]. That year, over one million malicious applications were removed along with almost 200 000 developer accounts. However, the machine learning algorithm that automatically removed malicious applications also caught some unfortunate safe applications. While it is essential to catch as many malicious extensions as possible, falsely removing non-harmful extensions should also be avoided whenever possible.

Furthermore, once a machine-learned model exists and has been used in practice, there is a risk that malicious developers will find out what the machine learning algorithm reacts to and how the extensions are labelled. With this information, the developers could adapt their solution so that the machine learning tests label the extensions as safe even though they should have been marked as malicious.

2
Terms and Basic Concepts

This chapter focuses on describing the most important terms and concepts related to the data set and the extensions that were used in this thesis when creating the machine learned model.

2.1 Data Set

To classify extensions using machine learning as either malicious or safe, a baseline is required to use as validation. In this thesis, the baseline was a data set created by Benjamin Eriksson, Pablo Picazo-Sanches and Andrei Sabelfeld [5]. This data set included the source code of 3321 extensions as well as a label for whether each extension was considered to be malicious or not. Of these 3321 extensions, 1289 extensions were marked as malicious, while 2032 extensions were marked as safe, which makes this an imbalanced data set [11]. An extension was marked as malicious if the extension contained a Query Stealing attack, which steals information written into search engines by users and sends it to third parties. This is often made without the permission or knowledge of the users [5]. Whether an extension was conducting Query Stealing attacks or not, was checked with the help of a dynamic scanner which tracked where queries were being sent after written into a search engine. This type of attack could create problems for the users since sensitive data could fall into the wrong hands, for example, stealing personal information that could be used as blackmail.

The labelled extensions in the data set were all of the type New Tab, meaning that their primary purpose was to change the default functionality of the browser when a user was opening a New Tab. This could be done in many different ways, from changing the background image to follow a specific theme, to giving the user weather updates or the option to manage to-do lists [12].

The data set consisted of an additional 194 617 extensions that did not have a label and were not necessarily of the New Tab category. These extensions were not used in this thesis since there was not possible to validate whether any results achieved using these extensions were accurate or not. To correctly label these extensions, a future work might either have labelled data on the data set, or an unsupervised machine learning model could be used.

Each extension folder in the data set consisted of a CRX compressed file, which in

turn was composed of all the files that the extension was dependent on [13]. The file types included in the CRX compressed file were JavaScript files, *Hyper Text Markup Language files* (HTML), and various files focused on appearances, such as CSS and image files. The final type of file that was present in the CRX files were the manifest files which will be introduced in Section 2.2. An example of how an unzipped CRX file can look like can be seen in Figure 2.1.

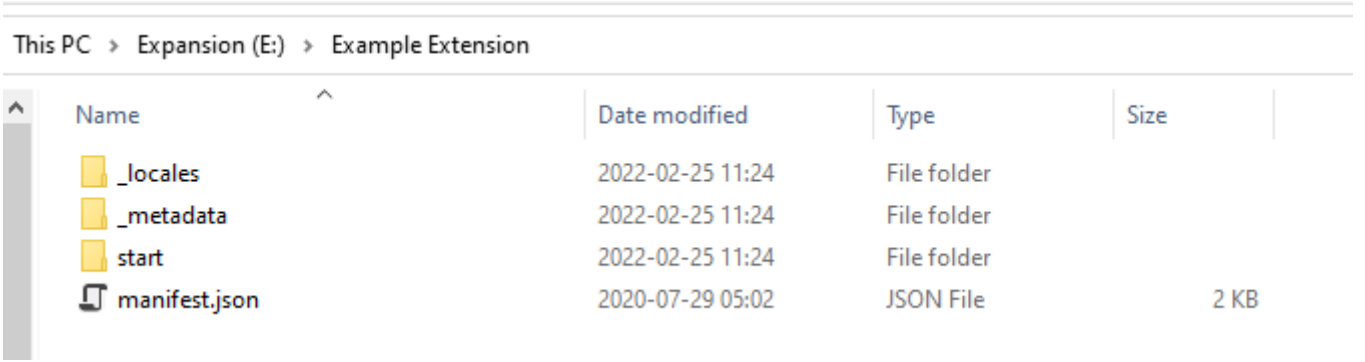


Figure 2.1: Example of contents inside a CRX file

The JavaScript files contained the code that was going to be executed when any extension was running, telling the extension what functionality was available. HTML-files were files that state what should be displayed when the extensions were active.

2.2 Manifest

The manifest file is a mandatory file in the JavaScript Object Notation (JSON) format that provides necessary metadata about the extension in different fields [14]. The fields are sorted sorted into mandatory, recommended and optional fields, as can be seen in Table A.1, Appendix A, where all the fields are listed. An example of a manifest file can be seen in Listing 1.

The mandatory fields that are available in the manifest are manifest version, name of extension and internal extension version. The data inside the extension's name and the internal extension version fields mirror the extension title. The manifest version specifies what file format the extension requires [15]. Different manifest versions give the developers access to different kinds of APIs and have other standards for security and privacy for the extension users. Manifest version 2 has been the standard version since support for the previous version, manifest version 1, stopped in Google Chrome version 18 in 2012 [16][17]. Manifest version 3 has been available for developers since the beginning of 2021 and will, in time, make the older version obsolete. Manifest 3 comes with new API additions and general security improvements. One security aspect that was updated with the introduction of manifest version 3 was that remotely hosted code was no longer allowed [18]. New extensions with manifest version 2 are not allowed to be uploaded onto the Chrome Web Store since January 2022, and by June of 2023, extensions using manifest version 2 will

```
{
"update_url": "https://clients2.google.com/service/update2/crx",

  "manifest_version": 2,
  "version": "1.1.2",
  "name": "Example Extension",
  "description": "Does something interesting when opening a new tab",
  "default_locale": "en",
  "chrome_url_overrides": {
      "newtab": "index.html"
  },
  "icons": {
      "128": "icon.png"
  },
  "browser_action": {
      "default_icon": "icon.png"
  },
  "permissions": ["storage", "unlimitedStorage"],
  "background": {
    "scripts": ["js/bg.js"]
  }
}
```

Listing 1: Example of a manifest file.

not be allowed to run on Chrome [19].

The recommended but not mandatory fields are: action, default locale, description, and icons. The description and icon fields are self-explanatory in that the description describes the extension, and the icons give the extension an image that the users will see during installation [20][21]. In the action field, developers can modify the extension's appearance in the toolbar, e.g., by declaring the image shown or the tooltip that shows up when hovering over the extension [22].

The field default locale lets the developers add the default language for the extension [23]. This field is required if the directory *_locales*, as seen in Figure 2.1, is present in the extension, and if that directory is not present, then the field should be left empty. The default locale value is usually set for "en", as in English. In the _locales directory, the developer also could add other languages to make it possible for the user to decide which language they want to use.

There are 47 optional fields in the manifest file that can be filled with various kinds of information. In this thesis, the optional fields that were used in the machine learning were: author, chrome settings override and permissions. The author field is a simple string stating which author developed the extensions. The chrome settings override field gives the developer access to changing the browser settings for the

extension user, including changing settings such as the default homepage or default search engine. The permissions field allows the developer to access specified APIs if the user gives their consent [24]. Some settings that could be changed with the chrome settings override field are the user's location, browser history or cookies. The user is prompted to accept the permissions when the extensions need them rather than when the extensions are being installed so that the user is more aware of when certain information is required.

2.3 Keywords

The functionality of the extensions is located in the JavaScript and HTML files. Through the code, the extensions have access to various APIs, methods, and function properties. Both the generic JavaScript code and the code from Google might depend on which permissions are active in the manifest. These different API, methods and function properties will henceforth be called Keywords.

Since using any of these Keywords is not mandatory, opportunities are created for the developers to create unique extensions with different combinations. Using some of the Keywords in combination with others, or on their own, can expose the user to different kinds of attacks. Since the data set that this thesis is based on focuses on the Query Stealing attack, the focus on the Keywords will be limited to the ones that could potentially have an impact in this area. As a baseline, Keywords that will be explored and used in this thesis are mainly extracted from the paper written by N. Pantelaios et al. [2] with alterations based on whether the Keyword is considered to be of greater importance in the scope of the thesis. The complete list of the Keywords used in this thesis can be seen in Table A.2, Appendix A.

2.3.1 eval

The functionality of the eval function is, in its core, relatively straightforward since it evaluates an expression inside of a string and returns a value [25]. However, since there are no checks on the input, it is considered a major security risk to use this function property for both the user of the extension containing eval, as well as for the developer of the extension. In the documentation of the eval function, it is stated that it is recommended to avoid utilisation as much as possible, and examples of how code can be rewritten are available in the code documentation [25]. A simple example of how eval can be used can be seen in Listing 2 and an example of how eval could be used with malicious intent can be seen in Section 2.4.

```
console.log(eval('2 + 2'));
    // Print in the console will be: "4"
```

Listing 2: A example of how eval can be used.

2.3.2 String.fromCharCode

String.fromCharCode is a static method that returns a character sequence of characters from a given specified charCode or sequence of charCode [26]. The charCode is one or more numbers in the range of 0 to 2^{16}. The input values can, aside from being denary values, also be either hexadecimal or octal. A simple example of how this method could be used can be seen in Listing 3.

```
console.log(String.fromCharCode(101))
// Print in the console will be: "e"
console.log(String.fromCharCode(72, 101, 108, 108, 111))
// Print in the console will be: "Hello"
```

Listing 3: An example of how the method String.fromCharCode can be used.

2.3.3 innerHTML

The element property innerHTML gives the developer an opportunity to get or set the HTML or XML values inside of an element through JavaScript code [27][28]. Examples of innerHTML can be used in practice can be seen in Listing 4, starting with getting the string value from the HTML or XML file into the variable "content". Listing 5 shows how the changing of the contents, by using set, can be done.

```
let content = theElement.innerHTML;
//This can also be done through getting contents from a
//specific element with an id
let content2 = document.getElementById("theID").innerHTML
```

Listing 4: An example of how innerHTML can be used.

```
theElement.innerHTML = "New content to add"
//This can also be done through setting contents to a specific
//element with an id
document.getElementById("theID").innerHTML = "New content to add"
```

Listing 5: An example of how innerHTML can change the content.

By observing the similarities with the other Keywords that have been introduced, it is possible to deduce that having the ability to inject code freely might be considered a security vulnerability. This is also something that the code documentation mentions to the developers. In the code documentation, they also inform the developer how to use this element property in a safe way, or even in which situations it should be avoided altogether [29].

One instance where innerHTML could be used in an attack from users of an extension is through an attack called *Cross-Site Scripting*, in which attackers inject code in otherwise secure programs or websites [27]. This is done by injecting executable code as a string, e.g., where developers are looking for users' input. These inputs can be executed either locally or on the server, depending on what the user with malicious intent is looking to do. An example of how this could be done can be seen in Listing 6.

```
const name = "<img src='x' onerror='alert(1)'>";
theElement.innerHTML = name;
//Shows an alert where a string with a name is expected
```

Listing 6: An example of how the attack Cross-Site Scripting can be performed.

Something that has been added in the latest HTML version, HTML5, is that it is since 2008 no longer possible to inject *<script>* - tags as executable code [27]. This was made to avoid a vulnerability using the <script> - tag. However, thinking that this is enough to completely stop all vulnerabilities that could arise from using innerHTML will create a false sense of security. This can be seen in the code snippet in Listing 6, where code was still able to be injected and executed without the *<script>* - tag.

Developers with malicious intent might use innerHTML to get around security checks that the web browsers have to catch suspicious behaviour. For example, Mozilla has implemented a check for in their browser extension code review, where an extension containing innerHTML might be rejected entry to their web store. Mozilla also advises their extension developers to use other methods to achieve the same result as they intend with innerHTML [27]. Whether this is also the case with the Chrome Web Store is still unknown at the point of writing.

2.4 Combination of Methods

Looking at the examples shown in Listing 3, it is possible to deduce that using the correct sequence of charCode, a developer could write anything using String.fromCharCode. This means that developers could obfuscate and hide malicious code within the charCode, which is what happened in an attack on WordPress in 2019 [30]. In that attack, a combination of String.fromCharCode and eval was used to inject malicious code from an external website. In the code snippets in Listing 7 and Listing 8, a part of the attack is shown.

```
<script type=text/javascript>eval(String.fromCharCode(118, 97,
    114, 32, 115, 99, 114, 105, 112, 116, 32, 61, 32, 100, 111,
    99, 117, 109, 101, 110, 116, 46, 99, 114, 101, 97, 116,
    101, 69, 108, 101, 109, 101, 110, 116, 40, 39, 115, 99,
    114, 105, 112, 116, 39, 41, 59, 10, 115, 99, 114, 105, 112,
    116, 46, 111, 110, 108, 111, 97, 100, 32, 61, 32, 102, 117,
    110, 99, 116, 105, 111, 110, 40, 41, 32, 123, 10, 125, 59,
    10, 115, 99, 114, 105, 112, 116, 46, 115, 114, 99, 32, 61,
    32, 34,104, 116, 116, 112, 115, 58, 47, 47, 101, 120, 97, 109,
    112, 108, 101, 46, 99, 111, 109, 47, 106, 97, 118, 97, 115,
    99, 114, 105, 112, 116, 46, 106, 115, 39, 59, 10, 100, 111,
    99, 117, 109, 101, 110, 116, 46, 103, 101, 116, 69, 108, 101,
    109, 101, 110, 116, 115, 66, 121, 84, 97, 103, 78, 97, 109,
    101, 40, 39, 104, 101, 97, 100, 39, 41, 91, 48, 93, 46, 97,
    112, 112, 101, 110, 100, 67, 104, 105, 108, 100, 40, 115,
    99, 114, 105, 112, 116, 41, 59));</script>
```

Listing 7: An attack using String.fromCharCode and eval.

```
<script type=text/javascript>
    eval("var script = document.createElement('script');
    script.onload = function() {};
    script.src = "https://example.com/javascript.js";
    document.getElementsByTagName('head')[0].appendChild(script);");
</script>
```

Listing 8: Listing 7 translated to string characters.

The code snippet in Listing 8 evaluates a string as JavaScript code, which then fetches code from a file located on the specified website. Using this kind of obfuscation, an ill-intended developer could hide and execute dangerous code that might be hard to detect in code evaluation. In this specific attack, the website *yourservice.live* was used, but it was removed from the snippet to avoid anyone from accidentally running the malicious code.

3
Machine Learning

This chapter focuses on the most important machine learning aspects used in this thesis. The start of the chapter will look at machine learning in general and then, in the later stages of the chapter, go over how machine learning was utilised in this thesis.

3.1 Machine Learning Methods

Machine learning is a continuously exciting research topic in computer science, as can be seen in the increasing number of companies that have started incorporating this into their business has steadily increased [31]. Machine learning is a subpart of computer science's artificial intelligence field of study. It is mainly used when a data set is explored to predict new values or find underlying patterns within the sample. These predictions are based on individually measurable values in the data set, more commonly known as features [32].

A few requirements must be met to classify an algorithm as a machine learning algorithm: a presence of a classification process, a loss function, and an optimisation process [33]. Combining these three requirements makes it possible to train a machine learning model. The first step in the training process is categorising the training samples into different classes. The next step is to use a loss function to see how much error the predicted classification had. The classification is redone using the optimisation process. The entire procedure is repeated until the error margin reaches its desired goal or the training has gone through the specified number of iterations.

While the process above describes the general procedure for using machine learning, there are still quite a few different ways in which the process can be done. Generally, machine learning is divided into three subgroups which depend on what kind of data is available and what the desired result of the algorithm is. The three subgroups are: supervised, unsupervised and semi-supervised learning.

Supervised learning is used when an extensive data set labelled according to some annotation is available. In other words, this kind of machine learning technique should be used when both the input and the output values are known for a large data set. There are several different supervised machine learning algorithms, of which the ones used in this thesis are *Random Forest Classifier* (RFC), *Logistic Re-*

gression (LR), *Support Vector Machines* (SVC) and *Multi-layer Perceptron* (MLP). One benefit of supervised learning is that the technique is easy for humans to understand since all the calculations can be visualised. However, a negative aspect of supervised learning is that it requires a vast amount of annotated data that might be computationally demanding to create.

In contrast to supervised learning, where the output value is known, unsupervised learning allows for learning without annotated data. With unsupervised learning, it is possible to, based on the features, categorise the data sample into clusters. Depending on which unsupervised learning algorithm is used, it is possible to use a predefined number of clusters or let the algorithm decide what is best for each situation. One unsupervised learning algorithm that allows the user to specify the number of clusters used is *KMeans*. Since there is no need for annotated data, unsupervised machine learning can compute results faster than supervised learning. Unsupervised learning is also suitable for gathering information about the content in an unlabelled data set where there is no available knowledge about the data set.

The third type of machine learning method is semi-supervised learning, a mix of supervised and unsupervised learning. Semi-supervised learning is used when there are annotated data points, but they are too few to be used in supervised learning. In that case, the labelled data is used to train a new data set that, in combination with the original data set, is used as a new training set for the following training iteration. This hybrid variation of supervised and unsupervised learning makes it possible to use the labels even if there are only a few, while at the same time, the labels are not ignored as they would have been in an unsupervised learning method.

3.1.1 Random Forest Classifier

A supervised machine learning algorithm used in this thesis is RFC, which is available as a software package developed by *scikit-learn* [34]. The algorithm is based on decision trees used together as an ensemble of trees to create a collaborative decision. A decision tree is an algorithm that divides the data set into two parts depending on whether they fulfil a condition. The dividing of the data set continues until the tree can categorise the class or when the algorithm has reached maximum depth. An example of a decision tree can be seen in Figure 3.1 where x represents a feature. The leaves in the trees represent different classes where in this example, class 0 could represent safe extensions, and class 1 could represent malicious ones.

RFC uses an ensemble of trees to classify data in samples. However, not all of the features present in the data set are used for decision trees created during training. Only a subset of features is used for the different trees, where each tree makes its own classification. The aggregate result of the classification made by most trees will be the final classification made by the algorithm.

RFC is suitable for both classification problems where discrete label values are predicted, and regression problems where continuous quantity is predicted. There is

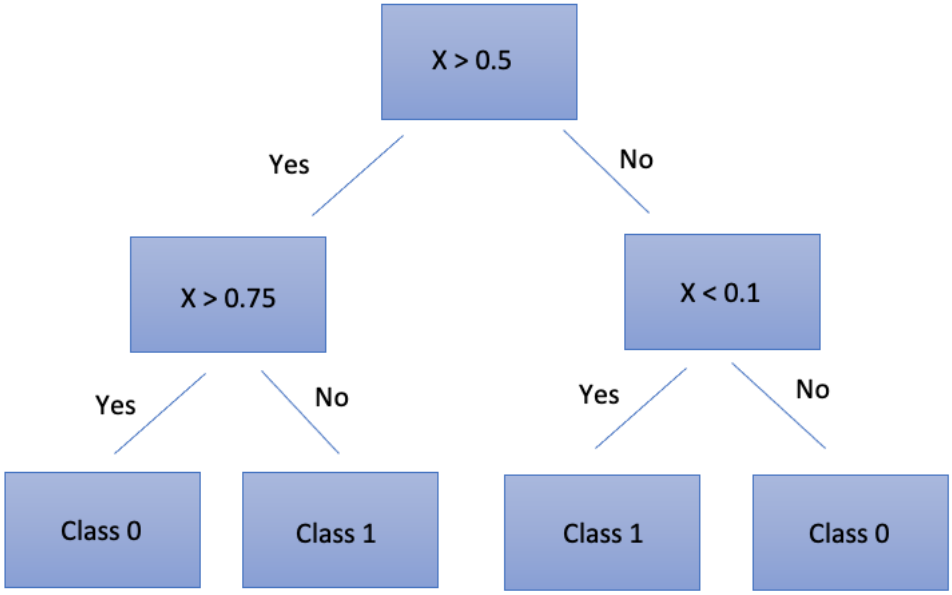


Figure 3.1: An example of a decision tree.

also no need for clearly distinguished values for RFC to work, meaning that the method works for both linear and nonlinear problems. Another benefit of RFC is that it works well with numerous features since each decision tree only uses a subset of all the features available. These different situations make it possible to use RFC for a variety of tasks.

On the other hand, the results from RFC are hard to interpret and work as a black box, meaning that it is not possible to retrieve information about how the results are obtained. Therefore, it is not possible to see what effects different features have on the results. Another drawback with RFC is that computationally heavy for larger data sets that require a large ensemble of trees [35].

3.1.2 KMeans

The unsupervised learning algorithm KMeans [36] is used in cases where no annotated data is available, and the desired result is finding patterns in the data set. This algorithm creates k number of clusters, where k is chosen by the user. The algorithm starts by randomly placing k centroids in the data set's universe. A centroid is a geometric centre representing the cluster's centre. The next step is to sign each data point to the closest centroids. This step is done with the help of Equation 3.1. In this equation, x is the data point to be signed to the centroid, m is the centroid, and K is the number of clusters. When all data points have been assigned to a centroid, the position of the centroids is recalculated according to Equation 3.2. Here, x is the data points assigned to the centroid, and m is the new place of the centroid. The number of data points assigned to a certain centroid is denoted with n. These two equations continue until the centroids are not relocated anymore, after which

the algorithm terminates.

$$C(\mathbf{x}_i) = \underset{1 \leq i \leq K}{\arg\min} ||\mathbf{x}_l - \mathbf{m}_i||^2 \tag{3.1}$$

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{C(x_l)=i} \mathbf{x}_l \tag{3.2}$$

The calculations described above are fast, which leads to the complete training having high performance [36]. The fast computations make it possible to find the best suitable configuration for each specific data set. However, even if it is simple to adjust the algorithm, there is always a risk that it will be stuck at a local minimum. That means that the algorithm will converge even if the achieved results are not the best. Another thing that can be a disadvantage with using KMeans is that the user needs to decide the number of clusters themselves. Since the data set does not have any labels, it is not beforehand known how many clusters exist, which is hard to predict since it takes time to check all the different numbers of clusters.

Even if it is hard to predict and takes time to go through all the different sizes of clusters, it is still possible and feasible. Later on, when the size of clusters has been decided, it is possible to check how good the size is with an internal index. An internal index is a score that focuses on how well the clusters are created. The silhouette score is an example of an internal index that can be used as an indicator. This score looks at how close the clusters' data points are and compares this with the distance between the clusters. The calculation of these values is shown in Equation 3.3. In this equation, a is the distance within the cluster, and b is the mean distance to the nearest cluster. The best possible silhouette value is 1, and the worst is -1.

$$\frac{b-a}{max(a,b)} \tag{3.3}$$

3.1.3 Confusion Matrix

There are different ways to present a machine learning result. One common form of displaying the results is using a confusion matrix. The matrix shows the result as false negative (FN), false positive (FP), true negative (TN) and true positive (TP), as can be seen in Figure 3.2. The figure illustrates that if an extension is labelled as safe and the algorithm predicts the value as safe, the extension will be classified as TN. Furthermore, if the extension is safe, but the algorithm has predicted it as malicious, then the confusion matrix will classify it as FP [37].

3.1.4 Precision, Recall and F1 score

Two scoring methods used during this thesis are the precision and recall scores [38] [39]. To calculate these values, Equation 3.4 and Equation 3.5 are used. For good results, the recall and precision scores should be as close to 1 as possible, which is

		Predicted value: Safe	Predicted value: Malicious
Real value	Safe	True Negative	False Positive
	Malicious	False Negative	True Positive

Figure 3.2: A confusion matrix showing how the extensions are classified.

achieved by minimising the amount of FN and FP extensions.

Another good scoring method to use, especially when the data set is imbalanced, is F1 score [40]. F1 score is built on precision and recall, and how it is calculated is shown in Equation 3.6. F1 score is also good to use when it is bad to have many FNs, as in the case of this thesis where predicting malicious extensions as safe is to be avoided [41].

$$Precision = \frac{TP}{(TP + FN)} \tag{3.4}$$

$$Recall = \frac{TP}{(TP + FP)} \tag{3.5}$$

$$\frac{2 * Precision * Recall}{(Precision + Recall)} \tag{3.6}$$

4
Methods

This chapter states the methods used to create the machine learning model that can predict the safety of extensions. The first part of the chapter closely examines the data set introduced in Section 2.1, from which a new data set is created with the features that will be used in the model training. The second half of this chapter describes how the training was conducted.

4.1 Preprocessing Data

An extension consists of mainly two parts that are of interest to this thesis, the manifest file and the combination of JavaScript and HTML files. The manifest fields important for the machine learning in this thesis were introduced in Section 2.2. Something crucial that was considered when the fields were chosen was that all the fields selected should be present in many of the extensions. Another important thing when selecting manifest fields was that the fields should not contain values unique to only one extension. Values that are unique or rarely used would only create problems for the machine learning by adding noise to the model.

Alongside selecting the features from the manifest, the Keywords from the JavaScript and HTML files were decided. As mentioned in Section 2.3, the Keywords that were used in this thesis are based on the metrics put forth by N. Pantelaios et al. [2] to avoid reinventing the wheel and spending many hours studying all the available API:s, methods, functions and element properties. They created the list by collecting all the available APIs from the Chromium Code Search and then narrowing down the results by cross-referencing them to what APIs were present in 55 verified malicious extensions. The list of Keywords used in this thesis was created by slightly altering the list created by N. Pantelaios et al. [2]. This was done by, for example, removing duplicate Keywords, such as, e.g., *document.createElement* and *createElement* which only exist in combination with each other making the existence of them both in the same list redundant. The complete list of Keywords used in this thesis can be found in Appendix A, Table A.2.

4.2 Modified Data Set

With the Keywords and manifest features defined, the next step was to create a new data set containing the extension ID:s, their categorised label as either malicious or safe, and all the features. The data available to work with was the data set

described in Section 2.1, containing the 3321 labelled extensions of the type New Tab and their extension source code.

The new data set that would be used to train the machine learning model was created by first populating it with applicable data. This was done by first extracting the files inside of the CRX compressed file. Each CRX file was located inside a folder with the extension ID's name. Without having access to an efficient way of extracting data from the CRX files, a few scripts were created in Bash and PowerShell to assist with the task. These scripts also renamed the extensions that did not follow the naming convention "extensionID.crx.zip". Renaming the extensions with incorrect naming conventions ensured that the IDs were not lost when extracting the files from the CRX. The steps for extracting the files were as follows:

1. The CRX files were renamed to include the extension IDs. Some of the extensions were named using their internal version number instead of their IDs and should therefore be renamed to retain that information, as seen in Figure 4.1. Other extensions were already named using their extension IDs, as seen in Figure 4.2, and were not renamed in this step. A renamed extension was renamed as extensionID_intervalversionNumber.crx.zip and moved out from their individual folders to a new folder only containing CRX files.

2. The remaining extensions still inside their individual folders were relocated to the same folder that the renamed CRX files were moved to in the previous step.

3. The extensions were unzipped from their CRX files to extract the files contained inside. This was done using the Windows file archiver WinRAR.

4. The newly created extension folders were renamed from either ID.crx.zip or ID_internalVersionNumber.crx.zip to simply their ID.

5. To reduce the computational power of the upcoming feature extraction from the manifest files, those files were moved out of their folders. To avoid confusion between different manifests, they were renamed from *manifest.json* to *ID.manifest.json*.

6. The last step was to rename the manifests, so they only kept their ID alongside the file type. The final names of the manifests were *ID.json*.

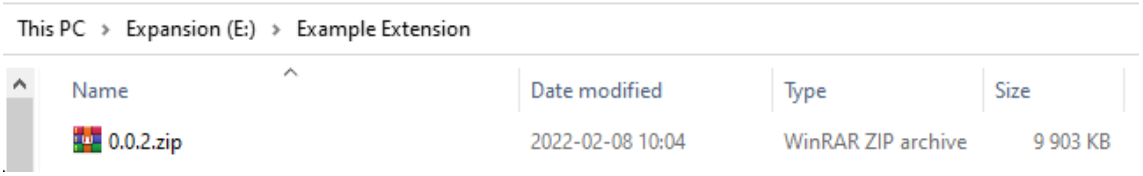


Figure 4.1: Extension with an incorrect naming on the CRX file inside a folder with the extension ID as its name.

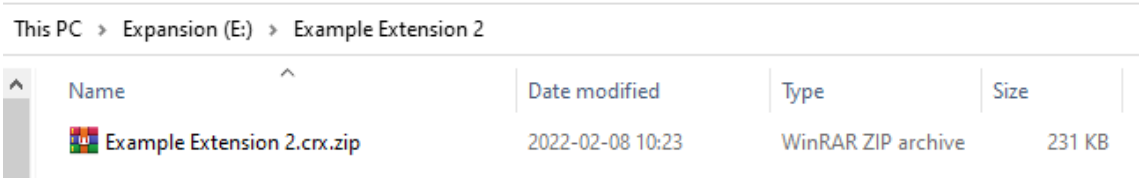


Figure 4.2: Extension with correct naming on the CRX file.

With all the manifests extracted from the extensions, it was now possible to gather the extension metadata as features. To assist with visualising the data, a Python tool called *pandas.DataFrame* was used [42]. This tool handles comma-separated values (CSV) files, commonly used by spreadsheet programs such as Microsoft Excel and Google Spreadsheets [43], and could display the information using a graphical interface.

The pandas.DataFrame was first filled with the features from all the manifests. Data fields in the manifest that had boolean values, meaning that they only stated whether a field was active or not, were converted to ones and zeroes. A value of one denoted that the specific field was active, while zeroes indicated that the field was inactive. However, since some of the data in the manifest fields contained various string values, they needed to be converted to numerical values for the machine learning to work. This was done using a method called *one-hot encoding*.

The one-hot encoding was performed by gathering all the possible values present in any extensions within one field. These values were then used to create a column in the data table. The columns were then populated with either ones or zeroes depending on whether an extension had that value in their manifest or not. One example of how this was implemented can be seen in Figure 4.3 where the extension *Example Extension* has the permissions *alarms* and *bookmarks* present in the permissions field in the manifest, while permissions like *activeTab* and *background* were not present. This procedure was done for the author and permissions fields since each had multiple possible values. In addition, a supplementary field was added that counted how many permissions each extension had active in its manifest. This was done to see if there was a correlation between whether any specific amount of permissions was more common in malicious extensions than safe ones.

ID object Extension 1 ..... 10% Extension 2 ..... 10% 8 others 80%	malicious int64 0 - 1	activeTab floa... 0.0 - 0.0	alarms float64 0.0 - 1.0	background flo... 0.0 - 0.0	bookmarks floa... 0.0 - 1.0
Extension 1	1	0	0	0	0
Extension 2	0	0	0	0	0
Extension 3	0	0	1	0	0
Extension 4	1	0	0	0	0
Extension 5	0	0	0	0	0
Extension 6	0	0	0	0	0
Example Extension	1	0	1	0	1
Extension 8	1	0	0	0	0
Extension 9	1	0	0	0	0
Extension 10	0	0	0	0	0

Figure 4.3: Part of the pandas.DataFrame table with some active and inactive permissions.

With the features extracted from the manifests, the next step was to extract all the Keywords that were present in each extension. Since the Keywords are not numerical, they needed to be one-hot encoded as well. The search for Keywords was done in all the JavaScript and HTML files in each extension before being added to the pandas.DataFrame. With all the Keywords found, a field was added for counting the number of Keywords present in each extension.

4.3 Training

Utilising the new modified data set created in Section 4.2, containing information about the manifest and the Keywords, it was now possible to start training the model using machine learning. A training sequence was created, starting with preprocessing the data for training, which was used to train the model.

4.3.1 Preprocessing the Data for Machine Learning Training

Before it is possible to use a data set for training a model, it first needs to be processed. The first step in preprocessing is to centralise and standardise the data. The main reason for this is to have all the data on the same scale so that no feature is seen as more important for the training algorithm [44]. This was also done to make the model converge faster and improve the computation time. To centre and standardise the data, StandardScaler [45] by scikit-learn was used. This function centres

1. Training on Permissions
2. Training on Author
3. Training on Manifest Version
4. Training on Default Locale
5. Training on Chrome Settings Override
6. Training on Keywords
7. Training on everything

Listing 9: The list of the order of how the training data was added to the training.

the data around 0 and has a standard deviation of 1 on each feature independently.

The next step in the preprocess was to divide the data into three sets: one training set, one validation set and one test set. The ratio chosen for the different data sets was 75% training data, 15% validation data and the remaining 10% was used for the test data. These different sets were made to ensure that the model does not get biased [46] which could happen if the whole data set was used for training. A biased model has a greater risk of overfitting and getting poor results [47] since it will treat any noise as legitimate data and train using that information. This issue could be avoided if the training set is only used in the training stage and the validation set is used to validate the results using the optimal model parameters. The last set, the test set, was only used in the final test when the model and the parameters have already been decided, giving the model a final test with a never before seen set of data.

4.3.2 Training with RFC

The training was divided into a sequence of steps, as seen in Listing 9, to get an overview of how each parameter affected the results. Training on the data in smaller steps instead of training on all the data simultaneously also ensured that all the information was extracted correctly and no unreliable results were obtained. For each step, a category was added. However, on step 6, the Keywords were not added to the previous data but were instead trained separately at first. Since the Keywords contained data from outside of the manifest, it might be valuable to be able to separate those results from the ones obtained from the manifest. To see how the combination of the manifest and the Keywords worked together, step 7 merged the two into one data set that was used for the final training.

The model used during the training was a RFC model with 100 trees, no max depth and a balanced class weight. To ensure that the different scores from the validation set were no lucky coincidences and to be able to compare them fairly, every training was made using 1000 iterations, and a mean value for the scores was calculated. The scoring methods used in the comparison were the F1 score in combination with the precision and recall scores.

4.3.3 Finding the Right Supervised Model

The training has so far been conducted using the RFC. However, this was done without knowing that this type of machine learning model is the best for the current data set. To select the best possible model for this particular data set, each relevant model tested was trained with the training set, and the model with the best scores in the validation set was selected. All the algorithms originate from scikit-learn, and their default values were used. The evaluated algorithms were: MLP, LR, RFC and SVC. Each model was trained on the training set over 1000 iterations to avoid unreliable results that could occur from a single training. The mean score out of the 1000 iterations was calculated and compared, with the best alternative being chosen and used for the remaining research. After preliminary testing, it was conducted that the RFC model had the best compatibility with the data set and was used for the remaining of the thesis.

4.3.4 Feature Selection

Not every feature that has been added to the data set is of the same interest for the model. Some of the features did not add new information or only added noise to the algorithm. To avoid training the model using these features, a feature selection was made. In this part of the training, a method called bootstrapping was used. Bootstrapping takes some samples from the set and creates a subset. This makes it possible to test different set combinations to see if the result differs.

To make the feature selection, half of the total features were randomly chosen and used to train a model. When the training was finished, the most used features were selected with the help of a SelectFormModel [48], a function that assigns weight to each feature. The features with an importance score greater than the mean of all the weights were marked as important. This procedure was repeated with a new set of randomly chosen subsamples over 1000 iterations. During this training, the number of times each feature was used and the number of times each feature was marked as important were recorded. Using these values, the percentage of times a feature was marked as important was calculated. All features marked as important at least 90% of the times they were used in the training were extracted, creating a new, minimised data set. Since this new data set only contained the most important features, the amount of noise that could originate from less important features were removed. A data set with fewer features would also reduce the computing power required for training the model.

4.3.5 Hyperparameter Tuning the Model

As described in Subsection 4.3.3, a method for finding the optimal supervised model for the examined data set was created. This model was used to hyperparameter tune the model, in other words, find the best parameters for the data set. The hyperparameter tuning was done using a random grid search. The random grid search is a method that randomly tests predefined values to see which combination is best for the specified data set. In this random grid search, six parameters were

- **Number of estimators:** This parameter searches for the optimal number of trees in the forest, ranging from 100 and 2000 estimators where the estimators increases in steps of 100.
- **Number of max features:** This parameter has multiple options to find the optimal maximum number of features to consider for finding the best split. The options tested in this thesis were the square root of the total number of features and the binary logarithm of the total number of features.
- **Max depth:** This parameter decides the maximum depth of the tree. The parameter can test values between 10 and 110 in steps of 10, where the value denotes the levels of the tree. The maximum depth parameter also tests whether it is better to have a limit or not.
- **Minimum number of samples split:** This parameter tests values between 2 and 10 to find the best minimum number of samples in a node to create a split.
- **Minimum number of samples leaf:** This parameter tests values between 1 and 5 to decide the minimum value needed for a valid number of samples to create a leaf in the tree.
- **Bootstrap:** This parameter has either a true or false value. If the value is true, a bootstrap of the samples was used to train each tree, while a false value indicates that all the training samples are used for each tree.

Listing 10: The parameters that were tested when tuning the model.

tested using the specifications found in Listing 10.

4.3.6 Predicting the Test Set

To validate the results of the algorithm, the test set was used. The test set was preprocessed in the same way as the training and validation test, described in Subsection 4.3.1. After the preprocessing, the created model was used to predict the test set.

4.3.7 Comparing Different Models

Preliminary tests on the model created in Subsection 4.3.6 showed that some extensions were misclassified. Further investigations were made to see if the misclassification was unique to this model or if the extensions were hard to classify in general. All the models in Subsection 4.3.3 were used to investigate this. While the RFC model was hyperparameter tuned with the most important features, while the other three models used their default versions during training. The results from the RFC model and the other three models were then compared to see how often the extensions were misclassified.

4.4 Predicting Security of New Extensions

The possibility of predicting extensions whose security risks are yet to be evaluated is the goal of this thesis and is now possible to do using the trained model. The data set used for the prediction consisted of 1661 unlabelled extensions of the same New Tab extension category as the previous data set. The newer data set was gathered later than the original data set used for the model's training. The data of the new, unlabelled data set was preprocessed similarly to the previous data set with the features from the manifest and JavaScript and HTML files extracted. Once the data was correctly preprocessed, the model could predict whether the new extensions should be marked as either malicious or safe.

4.5 Training with KMeans

Training using unsupervised machine learning allows for utilising all the available data since the requirement on the availability of labels is removed. The unlabelled data was preprocessed in the same way as the labelled data described in Subsection 4.3.1. The unsupervised training was conducted using the machine learning algorithm KMeans, which requires a set number of clusters before training, as introduced in Subsection 3.1.2. This opened up the possibility of continuing the research in two different directions.

The first option was to have two different clusters in the training. Using this setup, one cluster represented the safe extensions while the other one represented the extensions considered to be malicious.

The other alternative was to run experiments to find the optimal number of clusters. This was done by testing different numbers clusters, ranging from 2 to 75, to see which gave the highest scores.

5
Results

In this chapter, the main results will be presented. The results will be presented in the same order as they were written in the method chapter, Chapter 4. The results start with showing the training of the manifest and the Keywords and end with the KMeans training and some final results.

5.1 First Training

The results from the training with the different number of features can be seen in Table 5.1. This was a training session with 1000 iterations, and the mean value for each feature set-up was calculated. As mentioned in Subsection 4.3.2, the RFC, with the default setting, was used as the algorithm. In Table 5.1, we can see that only training with the manifest field gave an accuracy of about 73 % on the F1 score, no matter which of the set-ups were used. However, if the model was trained using the Keywords, the results increased significantly with an F1 score of above 93 %. Finally, the last row of Table 5.1 shows the results the model had when training using both the manifest fields and the Keywords, which achieved the highest results with an F1 score of 95%.

Table 5.1 also presents the precision and recall scores. As seen, the tests have a high recall value for all the set-ups. However, the precision score was drastically higher when the Keywords were added to the mix. This means, according to Equation 3.4, that there were fewer FN values when the Keywords were considered. In other words, in the final model, not as many malicious extensions were classified as safe by the algorithm. The high recall scores obtained for all feature sets state that only a few extensions were marked as FP, no matter the feature set-up. This means that there were always a low number of extensions that were classified as malicious, even though they were safe in reality.

5.2 Supervised Models

Test runs with the different models were performed to know which model was best for this data set. The results from this run can be seen in Table 5.2. All the algorithms received an F1 score of over 90 %, which are great results. However, the RFC was outperforming the other models by a small margin. The model with the lowest score in these tests was LR. This can depend on various variables, such as that the default option of the logistic model did not fit the data set very well and

Features	F1 score	Precision	Recall
Permissions	0.7321	0.5912	0.9612
Permissions & Author	0.7316	0.5913	0.9593
Permissions, Author & Manifest Version	0.7325	0.5914	0.9621
Permissions, Author, Manifest Version & Default Locale	0.7311	0.5913	0.9577
Permissions, Author, Manifest Version, Default Locale & Chrome Settings Override	0.7338	0.5923	0.9641
Keywords	0.9343	0.9519	0.9174
All features	0.9534	0.9748	0.9330

Table 5.1: The F1 score, precision and recall values for the different set-up of features.

Model	F1 score
Logistic Regression	0.9392
Support Vector Classifier	0.9569
Random Forest Classifier	0.9655
Multi-layer Perceptron	0.9593

Table 5.2: The different machine learning models tested, with their respective F1 score.

needed hyperparameter tuning. The following results were obtained using the RFC model since it has the most promising results.

5.3 Improved RFC

The RFC's most important features are shown in Figure 5.1. All the features in that figure were marked as important at least once. The features marked as important every time they were used were: nbr_perm, String.fromCharCode, nbr_metrics, runtime.onMessage, onMessage.addListener, XMLHttpRequest, innerHTML, document.cookie, document.createElement, default locale and eval. The features not presented in the diagram were never selected as important. In total, 15 features were selected as important 90 % of the time they were used. Using these features instead of all the original features, as the last row in Figure 5.1, increased the F1 score from 0.9534 to 0.9680.

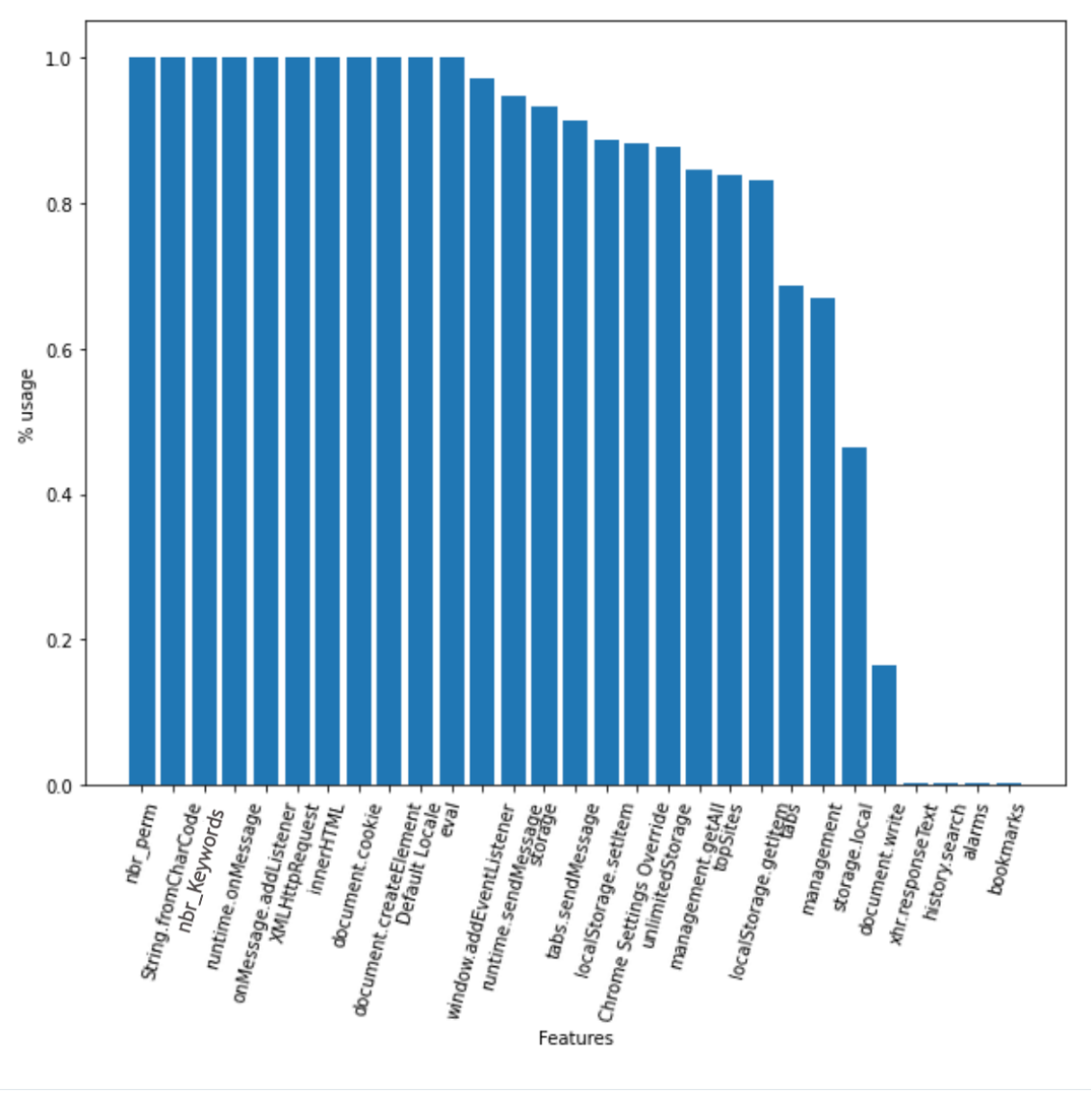


Figure 5.1: The most important features according to the RFC.

After the feature selection was made, the best parameters for the algorithm needed to be found. The results of this hyperparameter tuning can be seen in Table 5.3 where the five best set-ups from the training are shown. The validation data set was used to validate the results and find the best model. The validation score can be found in the rightmost column in Table 5.3. As can be seen, set-up #4 with 1500 estimators was the model that performed the strongest.

The final results are shown in Figure 5.2, displayed as a confusion matrix. The test set was used for the first time to achieve these results. As seen in the figure, there were only 12 extensions marked as FN and 1 extension marked as FP. The results in the confusion matrix achieved an F1 score of 0.9651.

#	Number of Estimators	Minimum sample split	Minimum sample of Leaf	Maximum numbers of Features	Bootstrap	Validation F1 score
1	500	4	2	log2	False	0.9630
2	400	3	2	sqrt	False	0.9630
3	1000	4	3	sqrt	False	0.9577
4	1500	9	1	log2	True	0.9651
5	1800	9	2	sqrt	False	0.9630

Table 5.3: The top 5 best parameters set-ups for RFC.

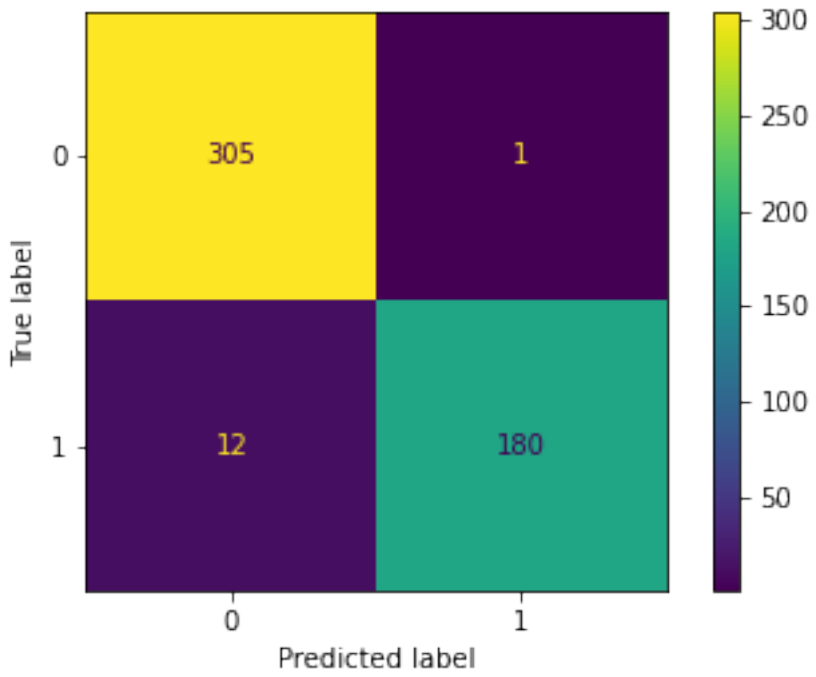


Figure 5.2: The confusion matrix presented the result of the test set.

5.4 Comparing Models

As seen in Figure 5.2, 12 extensions were FN, and 1 was FP. Out of the 12 extensions that were FN, all extensions were also masked as FN by at least one other algorithm. The full results from this can be seen in Table 5.4, where the right column shows if another algorithm besides RFC also marked it as FN. Furthermore, looking at the single FP extension, only the RFC model misclassified that extension. However, the other models had different extensions that they classified as FP.

5.5 Predict the Unlabelled Extensions

A prediction was created for the new extensions using the parameters and features that gave the best scores. The results can be seen in Table 5.5 where the algorithm detected around 300 new malicious extensions.

False Negative that have been found by RFC	Found by algorithm		
Extension 1		SVC	
Extension 2	MLP	SVC	LR
Extension 3	MLP	SVC	
Extension 4	MLP	SVC	
Extension 5	MLP	SVC	
Extension 6	MLP	SVC	
Extension 7	MLP	SVC	
Extension 8	MLP	SVC	LR
Extension 9		SVC	LR
Extension 10		SVC	
Extension 11	MLP	SVC	
Extension 12	MLP	SVC	LR

Table 5.4: The extensions that were categorised as FN by RFC and if some other algorithm also misclassifies them

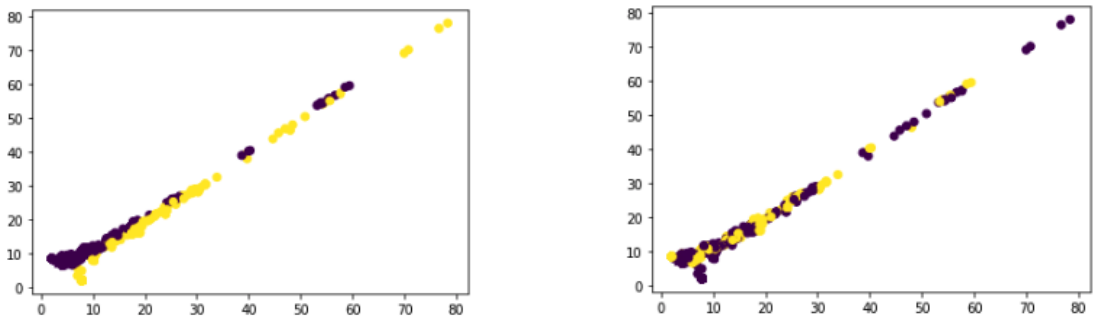
Safe	Malicious
1354	307

Table 5.5: The number of extension of the unlabelled that are safe respective malicious.

5.6 KMeans

Figure 5.3a shows the results of using KMeans with two clusters. Comparing these results to the actual labels shown in 5.3b, it can be seen that the accuracy of KMeans is not that high. The achieved F1 score of KMeans was 0.7153. In both graphs, the axis represents the cluster-distance space for each dimension, in other words, how far away the samples are from their cluster centres. Further comparison of the two figures shows that KMeans wants to group the extensions into bigger and separate clusters, which is not the case for the actual labels.

Since the results were not that high for the KMeans algorithm with two clusters, more clusters were tested. To know which cluster size was the best, a test was made to check every cluster size from 2 to 75. This result can be seen in Figure 5.4. The best silhouette score was achieved at 75 clusters. However, there is not a big difference between 75 clusters and 60 clusters. It is also possible to see that the scores increase with a higher number of clusters, which indicates that this might not be the best cluster size.



(a) The labels that KMeans algorithm have set for the data set.

(b) The real labels for the data set.

Figure 5.3: Two picture of how the KMeans cluster the classifies the data set and how the real labels are.

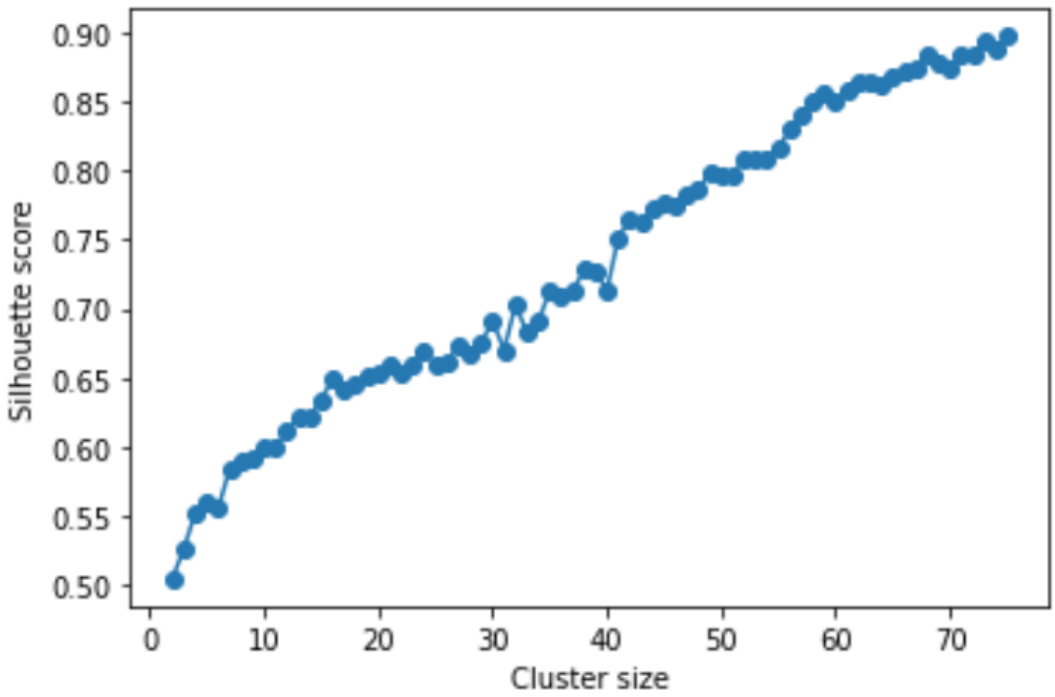


Figure 5.4: Silhouette score for the different clusters sizes.

6
Discussion

The main thing that this thesis aims to accomplish is to develop a method for using machine learning to evaluate the safety of various browser extensions. While creating a model that accurately predicts whether an extension is performing any type of attack on its users is out of reach of scope for this thesis, a proof of concept was still possible to create. This proof of concept shows that within the category New Tab, it was possible to predict, to a high degree as seen in Table 5.1, which extensions performed a Query Stealing attack on its users. With this information in hand, it could be reasoned that on a larger scale, with the correct data, it would be possible to predict any kind of attack on any type of extension. However, a few obstacles hinder such a claim from being accurate.

A model has to be trained using data about a specific attack to accurately predict whether an extension is using that type of attack on its users. To collect all the required data, methods of verifying whether an extension is conducting an attack or not would have to be available for all the different types of attacks. While this was impossible to create within this thesis's time frame without all the necessary data, it should be possible to accomplish with more time.

The initial data set was created between March and August of 2021 [5], while the data set used for the prediction was gathered in early 2022. Among other things that happened in that time frame, extensions are, as of January of 2022, no longer accepted if they utilise manifest version 2. These extensions will not be allowed to run on browsers starting from June of 2023 [19]. Comparing the data sets in this thesis, the number of extensions using the two different manifest versions varies vastly, as seen in Table 6.1. This likely affected the results in that the Query Stealing attacks might have to be executed using methods other than what was previously used to circumvent any new security measures.

Different manifest versions change what each extension has the opportunity to do by changing the available libraries, adding and removing functionality and by offering other extension security measures. Changes to the manifest might remove previously known vulnerabilities that could be exploited in attacks against the user. However, new manifest versions could not unlikely open up new ways of attacking the users.

Updates to manifest versions, changes to available APIs and updated security policies are deemed to continue. With these changes, malicious developers will be forced

Data set	Extensions using manifest version 3	% of extensions
Labelled	58 out of 3321	1.7
Unlabelled	440 out of 1661	26.5

Table 6.1: Presence of manifest version 3 in the initial data set versus the data set used for the prediction.

to alter their ways of attacking the users. This is something that the model will not be able to catch at the moment since it has not been trained using that kind of data. For the model to avoid becoming obsolete, it would be required to be updated constantly with new data as soon as new types of attacks are detected and new ways of performing old attacks are discovered.

The choice of Keywords and which manifest fields were focused on in the machine learning could be further improved. The results seen in Chapter 5 are based on one set of features used throughout the thesis. While the features were chosen to fit the data set as well as possible, they could have been decided more specifically. Having features more focused on the currently researched attack and extension category would likely result in higher result scores. On the other hand, features that are too specific to the particular data set might not work well in a more general model for extensions and attacks, meaning that they would have to be altered any time the model is trained on new data.

In the preprocessing of the data, the URLs in the permissions were removed. This removal was made since multiple unique values in this field did not positively contribute to the results. A data set that contains many unique or rarely used values makes the work of the machine learning algorithm harder since more features are taken into account. These features could be seen as noise since too few data points contained the same URLs and could create unreliable results. After removing the URLs, the data set became clearer and achieved better results.

The manifest field Chrome Setting Override was another value that was transformed during the training. The data within this field was similar to the removed URLs in the permissions field since many values were unique or rarely used. In this category, it was only meaningful if the extension had the option present or not since that meant that the extension had the permission to override default settings.

As could be seen in Table 5.1 in the results, only training on the manifest did not provide satisfactory scores. In order to understand why the manifest achieved worse results than the Keywords, it was helpful to look at the active features. When examining the manifests, one thing that came up was that two extensions could be identical with different labels. Having similar manifests with opposite labels makes it hard for the algorithm to learn how to act when similar situations arise in the future. This problem was not as common when examining the Keywords.

Another thing that could be observed in Table 5.1 was that the F1 score did not

noticeably increase when adding different content from the manifest. When adding the author field, the algorithm's F1 score decreased instead of increasing. The reason that the F1 score behaved this way could have multiple explanations, but one thing that could affect the results was that only 110 of the extensions, or about 3%, had an author specified. This made it hard for the algorithm to get valuable information from the field. Furthermore, looking at the data set, it could be noted that some authors had multiple extensions marked with different labels. As previously discussed, the algorithm has trouble working with data where labels vary with the same value. For these reasons, it could be concluded that the author field did not give any data of importance for the results. Preliminary testing without the author field taken into consideration achieved better results than with the field used as a feature.

Table 5.3 shows that the set-up with 1500 trees had the best set of parameters, with the set-ups using 400, 500 and 1800 trees following closely. With the best options being relatively close to each other, the best option could be to choose the set-up with fewer trees since the computational times become shorter while at the same time using less memory. While the computational times and memory usage might not be an issue while working with the data set in this thesis, it could substantially improve the performance when working with larger data sets. This shows that deciding which set-up to utilise differs depending on what data set is used and what hardware is available.

In Figure 5.1, the feature selection results are shown. In this figure, it can be seen that most of the important features can be found in the list of Keywords. Comparing this with the results from the first training, as shown in Table 5.1, it is logical that most of the important features came from the Keywords since the highest scores were obtained when the Keywords were added to the data set. The results show that training with only the Keywords is about 20 percentage points better than the best feature set-up using only the manifest fields.

Another interesting aspect that can be derived from Figure 5.1 is that the field default locale is one of the features marked as important every time it was used in the feature selection. This means that the model considered that the default locale field had data that could be useful for the model. When studying the data set, nothing indicates whether an extension is malicious or not, solely depending on the value of the default locale. Therefore, one conclusion that could be reached was that default locale was important for the model, but only in combination with other features since it did not provide anything on its own.

One part that could assist in reaching the goal of this thesis was to see if it was possible to categorise the different extensions with the help of unsupervised machine learning. For doing this, two main ideas were tested. The first idea was to examine whether it was possible to divide the data set into two clusters and if that could give any exciting findings. However, looking at the results in Figure 5.3, it was concluded that this was not a feasible option for the data set since the training only resulted in an F1 score of 0.7153. The fact that two clusters will not work for this data set can

also be seen by studying Figure 5.3b where it can be seen that the labelled clusters are not separated well enough.

The other idea when using KMeans was to test a lot of different cluster sizes and validate which one was the best with the help of the silhouette score. Figure 3.3 illustrates how the silhouette scores increased the more clusters were present. This gave another indication that the data set was hard to divide into smaller groups to find a common denominator.

Looking at the last row of the results achieved by using supervised learning shown in Table 5.1, it can be seen that the results are high. Achieving an F1 score of around 95 % was not something that was expected. Since the model works well for this data set, it could be said that the model was generalised for this particular data set. This indicates that the model was not overfitted, and that it was possible to use it for unlabelled data.

Another way to look at the results for the test set is to look at Figure 5.2. Here, it can be seen that out of 498 test points, only 13 were incorrectly categorised. Most of the wrong categorised extensions were FN. This is something that can both be seen as a positive and a negative thing. For the user, having a lot of FN is harmful because it creates a false sense of security for the user that everything is safe even though it might not be. However, having a few FNs or FPs is positive since it shows that a machine learning model can predict incorrectly and that the results should not be trusted blindly.

From the results in Table 5.4, it was possible to see that in all FNs that the RFC classified as FN, at least one other model also marked it as FN. Most of the extensions were found by two or three other models. These results make it possible to say that the 12 misclassified extensions were hard to classify and that the algorithm was not the problem. A reason why all the different models had trouble classifying these extensions might be that they were too similar to actually safe extensions. To solve this issue, the model might need more features that differ from the safe extensions or have more data points to train on.

6.1 Future Work

The research in this thesis focused on one type of attack, one type of extension, and mostly one type of supervised learning. The investigation shows that it is possible to use machine learning for attacks against users and could be used in the future. However, to create a tool that fully can say that an extension is safe, more work is required. Furthermore, to create a fully operational tool, more recent data needs to be collected in order to give the algorithm more information to work with.

As previously stated, for future research to be successful, more data is needed, and for optimal results, data should advantageously have been collected for a more extended period. The developers of malicious extensions will continuously evolve how

the attacks are executed and will continue to strive to develop software that gets the work done without being detected. To develop a model that works for a longer period and can find different malicious extensions, the model needs to have an opportunity to train using different extensions that could be malicious.

One thing that could improve the model results is the use of unsupervised learning. Due to lack of time, this field was not well-investigated and needed more work. An example of something that could be done in future work is to use large numbers of clusters to detect if there is any number of clusters that would consist of more malicious extensions than others. Other unsupervised learning methods could also be tried to see if they have any better results, such as *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) [49] or *Agglomerative Clustering* [50]. Since no investigation of these algorithms was made, it is hard to tell if they could have improved results over those of KMeans.

The benefit of unsupervised learning is that there is no need for annotated data, which means that there is a possibility to directly categorise extensions into groups that would be more or less likely to be malicious. Using this method would save time since there would be no need to gather annotated data as was needed for the supervised learning model.

Lastly, one large part that could be worked on in the future would be incorporating an algorithm that can transfer learning from a preexisting model to a new and improved model. To use transfer learning, a model trained on a related subject with a good test score is used to hypertune the model with the data set that should have improved results.

7
Related Work

Using machine learning to find security flaws in different programs is not a new concept. Looking into the papers currently available within the field, there are some that offer some interesting insights. This chapter will discuss some of the papers that provided the most ideas.

In the article *I Spy with My Little Eye: Analysis and Detection of Spying Browser Extensions* by Anupama Aggarwa et al. [51], different aspects of how user data could be collected through extensions are mentioned. For example, this article gives an excellent start to defining when something could be classified as malicious or not. According to A. Aggarwals et al. [51], most spying applications look through the users browsing history, IP addresses and geolocation. This information suggested what should be examined with the Keywords in the extensions when looking for signs of malicious behaviour.

Another aspect that the authors presented in the paper was how the use of recurrent neural network (RNN) improved a previous investigation made by Nav Jagpal et al., which is presented in the article *Trends and Lessons from Three Years Fighting Malicious Extensions* [52]. In the paper by A. Aggerwals et al. [51], memory's importance when doing dynamical analysis using RNN models is highlighted. The dynamical analysis is based on the executing scheme of the APIs in extensions. The RNN model learns that different execution sequences can be malicious, and patterns can be found. However, while there has been some research on how dynamic analysis can be conducted, the paper does not explore how static analysis can be done and what kind of impact that could have.

N. Pantelaios et al. [2] focus, on the other hand, more on the lifetime of the extensions. This article can be divided into two parts. In the first part, the authors investigate how the reviews of the extensions could be used to predict if the extensions are malicious or not. This is done by collecting the extensions that received over 50 reviews in the form of comments on the Chrome Web Store. From these comments, keywords were extracted and, with the help of an algorithm, used to give scores. A lower score meant that an extension containing that keyword had a higher probability of being malicious. Part of the results from this investigation was a list of words that often appeared in comments on malicious extensions

The second part of the paper *You've Changed: Detecting Malicious Browser Extensions through Their Update Deltas* [2] focuses on extracting various APIs, or what

they define as APIs, from the code. From these APIs, sequences were generated based on the order of the API calls, as well as a syntactical analysis with the help of the Esprima library. The analysis and API sequences were made on different versions of the extensions with which a delta was calculated. This delta was then used as input for a machine learning algorithm. The algorithm they used was the DBSCAN [49] which created 7,419 different clusters. These clusters were manually examined, searching for at least one malicious extension and then compared to the other extensions in the same cluster to see if they had the same malicious intent.

8
Conclusion

The method used for creating the model that could predict malicious extensions gave excellent results. Utilising the same procedure to update the model to handle new extension categories and attacks on users would result in an extra step of security. Updating the model with the latest extension data is required for the model to stay relevant. The current model would lose some of its accuracy over time since the restrictions set by the Chrome Web Store are constantly changing to plug any security risks, forcing malicious developers to change their attack patterns and exploits. This is something that the model would have to be updated with in order to be able to catch malicious extensions.

The goal of this thesis, as stated in Section 1.2, was to develop a method that could use machine learning to predict whether an explored extension should be considered safe or malicious. This goal could be considered reached since the prediction was possible in a smaller sample of extensions and within a limited scope, meaning that it should also be possible to apply on a larger scale. The prediction was possible with high F1, precision and recall scores, indicating that the risk of misclassifying extensions was low.

The results indicated that the manifest contents were not as important as the presence of various Keywords. This is something that could suggest to Google that further examination of how various APIs and methods are used to improve the safety of the extension users.

The Random Forest Classifier was a good machine learning model for this data set for multiple reasons. The first reason was that it was easy to visualise and understand how it worked. This made it simpler to verify that the model worked as intended and that no random values were created. Another reason, and probably the biggest, was that the RFC model achieved the highest results out of the examined supervised machine learning models.

Bibliography

[1] J. Johnson, "Global daily internet minutes per capita 2021." [Online]. Available: https://www.statista.com/statistics/1009455/daily-time-per-capita-internet-worldwide/

[2] N. Pantelaios, N. Nikiforakis, and A. Kapravelos, *You've Changed: Detecting Malicious Browser Extensions through Their Update Deltas.* New York, NY, USA: Association for Computing Machinery, 2020, p. 477–491. [Online]. Available: https://doi.org/10.1145/3372297.3423343

[3] S. E. Olaniyan, "How People Always Make the Best Exploit," Jun. 2021. [Online]. Available: https://www.linkedin.com/pulse/how-people-always-make-best-exploit-segun-ebenezer-olaniyan

[4] KnowBe4, "Phishing | What Is Phishing?" [Online]. Available: https://www.phishing.org/what-is-phishing

[5] B. Eriksson, P. Picazo-Sanchez, and A. Sabelfeld, "No signal left to chance: Driving browser extension analysis by download patterns," unpublished.

[6] E. Georgescu, "Have You Ever Installed a Malicious Chrome Extension?" Jan. 2021. [Online]. Available: https://heimdalsecurity.com/blog/malicious-chrome-extension/

[7] E. Mcgowan, "Third-party extensions for Facebook, Instagram, and others have infected millions," Dec 2021. [Online]. Available: https://blog.avast.com/malicious-browser-extensions-avast

[8] M. Dowd, J. McDonald, and J. Schuh, *The art of software security assessment: identifying and preventing software vulnerabilities.* Indianapolis, Ind: Addison-Wesley, 2007, . ISBN: 978-0321444424.

[9] D. Buttner, "The Importance of Manual Secure Code Review," *MITRE*, Jan. 2014. [Online]. Available: https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/the-importance-of-manual-secure-code-review

[10] S. Kafka and K. Shams, "How we fought bad apps and developers in 2021," Apr. 2022. [Online]. Available: https://security.googleblog.com/2022/04/how-we-fought-bad-apps-and-developers.html

[11] J. Brownlee, "A Gentle Introduction to Imbalanced Classification," Dec. 2019. [Online]. Available: https://machinelearningmastery.com/what-is-imbalanced-classification/

[12] “Chrome Web Store - New Tab Page.” [Online]. Available: https://chrome.google.com/webstore/category/collection/customize_your_new_tab_page

[13] “CRX File Extension - What is a .crx file and how do I open it?” [Online]. Available: https://fileinfo.com/extension/crx

[14] “Manifest file format,” Oct. 2021. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/manifest/

[15] “Manifest Version,” Jul. 2021. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/manifest/manifest_version/

[16] “Manifest Version,” Apr. 2018. [Online]. Available: https://web.archive.org/web/20210505013751/https://developer.chrome.com/docs/extensions/mv3/manifest/manifest_version/

[17] “Google Chrome version history,” Apr. 2022, page Version ID: 1083826396, visited 2022-04-21. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Google_Chrome_version_history&oldid=1083826396

[18] “Manifest V3 now available on M88 Beta.” [Online]. Available: https://blog.chromium.org/2020/12/manifest-v3-now-available-on-m88-beta.html

[19] “Manifest V2 support timeline.” [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/mv2-sunset/

[20] “Manifest - Description,” Apr. 2018. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/manifest/description/

[21] “Manifest - Icons,” Apr. 2022. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/manifest/icons/

[22] “chrome.action.” [Online]. Available: https://developer.chrome.com/docs/extensions/reference/action/

[23] “Manifest - Default Locale,” Apr. 2018. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/manifest/default_locale/

[24] “chrome.permissions,” chrome Developers, visited 2022-04-10. [Online]. Available: https://developer.chrome.com/docs/extensions/reference/permissions/

[25] MDN contributors, “eval(),” Apr. 2022, mdn web docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval

[26] ——, “String.fromCharCode(),” Feb. 2022, mdn web docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/fromCharCode

[27] ——, “Element.innerHTML,” Apr. 2022, mdn web docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML

[28] “HTML DOM Element innerHTML Property,” w3schools, visited 2022-04-12.

[Online]. Available: https://www.w3schools.com/jsref/prop_html_innerhtml.asp

[29] G. Diaz, "DOM innerHTML Vulnerability," Feb. 2020, medium. [Online]. Available: https://medium.com/@gregadiaz89/dom-innerhtml-vulnerability-8821a03ef2b8

[30] M. Veenstra, "Recent WordPress Vulnerabilities Targeted by Malvertising Campaign," Jul. 2019, wordfence. [Online]. Available: https://www.wordfence.com/blog/2019/07/recent-wordpress-vulnerabilities-targeted-by-malvertising-campaign/

[31] E. Burns, "What Is Machine Learning and Why Is It Important?" visited 2022-05-02. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

[32] R. Brown, "What are Features in Machine Learning and Why it is Important?" Dec. 2021. [Online]. Available: https://cogitotech.medium.com/what-are-features-in-machine-learning-and-why-it-is-important-e72f9905b54d

[33] IBM Cloud Education, "What is Machine Learning?" Jul. 2020. [Online]. Available: https://www.ibm.com/cloud/learn/machine-learning

[34] "Random Forest Classifier Algorithm," visited 2022-04-15. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[35] J. Singh, "Random Forest: Pros and Cons," Dec. 2020. [Online]. Available: https://medium.datadriveninvestor.com/random-forest-pros-and-cons-c1c42fb64f04

[36] "KMeans Algorithm," visited 2022-04-15. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.cluster.KMeans.html

[37] J. Brownlee, "What is a Confusion Matrix in Machine Learning," Nov. 2016. [Online]. Available: https://machinelearningmastery.com/confusion-matrix-machine-learning/

[38] "Precision score," visited 2022-05-30. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.metrics.precision_score.html

[39] "Recall score," visited 2022-05-30. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.metrics.recall_score.html

[40] "F1 score," visited 2022-05-30. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.metrics.f1_score.html

[41] P. Huilgol, "Accuracy vs. F1-Score," Aug. 2019. [Online]. Available: https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2

[42] "pandas DataFrame — pandas 1.4.2 documentation," visited 2022-05-13. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[43] C. Hoffman, "What Is a CSV File, and How Do I Open It?" Apr. 2018. [Online]. Available: https://www.howtogeek.com/348960/what-is-a-csv-file-and-how-do-i-open-it/

[44] Manikanth, "What is the use of data standardization and where do we use it in machine learning," Mar. 2021. [Online]. Available: https://medium.com/analytics-vidhya/what-is-the-use-of-data-standardization-and-where-do-we-use-it-in-machine-learning-97b71a294e24

[45] "Standard scaler," visited 2022-05-02. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[46] T. Shah, "About Train, Validation and Test Sets in Machine Learning," Dec. 2017. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[47] A. Kumar, "Machine Learning - Training, Validation & Test Data Set," Jun. 2021. [Online]. Available: https://vitalflux.com/machine-learning-training-validation-test-data-set/

[48] "Select from model," visited 2022-05-30. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html

[49] "DBSCAN Algorithm," visited 2022-04-22. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.cluster.DBSCAN.html

[50] "Agglomerative clustering," visited 2022-04-22. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

[51] A. Aggarwal and et al., "I spy with my little eye: Analysis and detection of spying browser extensions," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, 2018, pp. 47–61. [Online]. Available: https://doi.org/10.1109/EuroSP.2018.00012

[52] N. Jagpal and et al., "Trends and lessons from three years fighting malicious extensions," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 579–593. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/jagpal

A

Appendix 1

Field	Requirement
Manifest Version	Mandatory
Name	Mandatory
Version	Mandatory
Action	Recommended
Default Local	Recommended
Description	Recommended
Icons	Recommended
Author	Optional
Automation	Optional
Background	Optional
Chrome settings overrides	Optional
Chrome URL overrides	Optional
Commands	Optional
Content capabilities	Optional
Content scripts	Optional
Content security policy	Optional
Converted from user script	Optional
Cross origin embedder policy	Optional
Cross origin opener policy	Optional
Current locale	Optional
Declarative net request	Optional
Devtools page	Optional
Differential fingerprint	Optional
Event rules	Optional
Externally connectable	Optional
File browser handlers	Optional
File system provider capabilities	Optional
Homepage URL	Optional
Host permissions	Optional
Import	Optional
Incognito	Optional
Input components	Optional
Key	Optional
Minimum Chrome version	Optional
Nacl modules	Optional

Natively connectable	Optional
OAuth2	Optional
Offline enabled	Optional
Omnibox	Optional
Optional host permissions	Optional
Optional permissions	Optional
Options page	Optional
Options UI	Optional
Permissions	Optional
Platforms	Optional
Replacement web app	Optional
Requirements	Optional
Sandbox	Optional
Short name	Optional
Storage	Optional
System indicator	Optional
TTS engine	Optional
Update URL	Optional
Version name	Optional
Web accessible resources	Optional

Table A.1: All fields available in the manifest file of an extension [14]

KeyWords
bookmarks.getTree
cookies.getAll
document.createElement
document.write
document.cookie
downloads.download
eval
extension.sendRequest
history.search
history.getVisits
innerHTML
localStorage.setItem
localStorage.getItem
management.getAll
onConnect.addListener
onMessage.addListener
runtime.connect
runtime.onMessage
runtime.sendMessage
storage.local
storage.sync
String.fromCharCode
tabs.executeScript
tabs.sendMessage
window.addEventListener
window.location.replace
xhr.readyState
xhr.responseText
XMLHttpRequest

Table A.2: List of Code Metrics