Detecting Malicious Browser Extensions by Combining Machine Learning and Feature Engineering

13

Jacob Rydecki, Jizhou Tong, and Jun Zheng

Abstract

As the popularity of the internet continues to grow, along with the use of web browsers and browser extensions, the threat of malicious browser extensions has increased and therefore demands an effective way to detect and in turn prevent the installation of these malicious extensions. These extensions compromise private user information (including usernames and passwords) and are also able to compromise the user's computer in the form of Trojans and other malicious software. This paper presents a method which combines machine learning and feature engineering to detect malicious browser extensions. By analyzing the static code of browser extensions and looking for features in the static code, the method predicts whether a browser extension is malicious or benign with a machine learning algorithm. Four machine learning algorithms (SVM, RF, KNN, and XGBoost) were tested with a dataset collected by ourselves in this study. Their detection performance in terms of different performance metrics are discussed.

Keywords

Machine learning · Feature engineering · Browser extensions · Man-in-the-browser · Browser security · Support Vector Machine · Random Forest · K-Nearest Neighbors · Extreme Gradient Boosting · Balanced and imbalanced datasets

J. Rydecki
Department of Computer Science, New Mexico State University, Las Cruces, NM, USA
e-mail: jrydecki@nmsu.edu

J. Tong (✉)
Department of Computer and Information Science, Gannon University, Erie, PA, USA
e-mail: tong001@gannon.edu

J. Zheng
Department of Computer Science and Engineering, New Mexico Institute of Mining and Technology, Socorro, NM, USA
e-mail: jun.zheng@nmt.edu

13.1 Introduction

In this modern age, people are using computers more than ever – from the hourly usage of smart phones to the constant usage of desktop computers at work. Not only is the usage of computers at an all time high, but also the usage of the internet. People use web browsers to explore the internet for independent research, schoolwork, office work, entertainment, and anything else one can think of. The mass usage of web browsers such as Google Chrome, Mozilla Firefox, or Apple's Safari has led to users looking for better experiences and functionalities when using their web browser of choice, which are done through the usage of browser extensions. Depending on the web browser of choice, users can download many different types of browser extensions to enhance the functionality and their productivity when using the web browser.

Although the browser extensions can be incredibly useful, their extensive use has led to a dramatic increase of developers with intent to create malicious browser extensions [1, 2]. These malicious extensions can perform a wide range of malicious functionality such as stealing a user's usernames and passwords, redirecting the user to malicious links or outside advertisements, stealing user information (browser history, cookies, sessions, etc.), downloading malicious software such as Trojans to the user's computer. According to Cybernews [3], the web browsers and browser extensions have historically been an overlooked attack vector, with millions of people being infected with malicious extensions without even being aware of them.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

S. Latifi (ed.), *ITNG 2023 20th International Conference on Information Technology-New Generations*, Advances in Intelligent Systems and Computing 1445,
https://doi.org/10.1007/978-3-031-28332-1_13

Because of the aforementioned dangerous capabilities of malicious browser extensions and their continuous increased usage, this paper presents a method of detecting malicious browser extensions by using feature engineering and machine learning classification algorithms. The two essential issues of the proposed method are to identify the useful features for malicious browser extension detection and determine the detection algorithm with the identified features. The proposed method approaches the first issue by analyzing the static source code of both malicious and benign browser extensions. A number of features are extracted from the source code of three different file types in extensions which are listed and analyzed in Sect. 13.2. For a better detection performance, the proposed method does not just look at the binary "present" or "absent" condition of a feature, but rather counts its number of occurrences in a file. For this study, we only downloaded and analyzed browser extensions for Google Chrome – the current most popular web browser. All references to extensions or browser extensions in the following are for Google Chrome browser extensions.

The second essential issue is approached by using machine learning-based classification algorithms including Support Vector Machine (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), and Extreme Gradient Boosting (XGBoost). To test these algorithms, the 5-fold cross validation is used to train and test these algorithms. The experimental results are presented in Sect. 13.3.

13.2 Methodology

The primary goal of this study is to classify any single extension as malicious or benign which are done through three distinct steps: collection of browser extensions, feature engineering and dataset creation, and machine learning model training and testing.

13.2.1 Collection of Browser Extensions

For the purpose of training malicious browser extension detection models, it's necessary to obtain a significant amount of browser extensions. There are three main categories of browser extensions collected in this study: benign browser extensions from the chrome web store, malicious extensions that have been removed from the chrome web store, and self-made malicious browser extensions. The distribution of collected browser extensions is shown in Fig. 13.1.

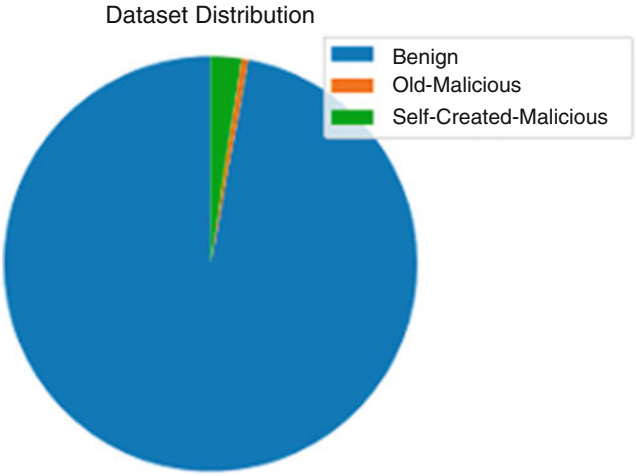

Fig. 13.1 The distribution of collected browser extensions

13.2.2 Feature Engineering and Dataset Creation

The second step is to perform a feature engineering by analyzing the code of collected extensions and create a dataset for our study. During this step, we search for certain keywords/phrases in the source code and count the number of occurrences of these keywords and phrases. After determining the number of occurrences, we populate a separate file that contains this information to be used by a machine learning algorithm.

For each browser extension's source code, we look for certain features contained within the code depending on the type of file being analyzed. Through the feature engineering, we identify 17 features from the HTML file, 20 features from the JavaScript file, and 3 features from the CSS styling file. Either the abundance, or dearth, of the identified features help to classify a browser extension as malicious or benign.

- **HTML Code Features**: The 17 features extracted from the HTML code of a browser extension are listed in Fig. 13.2. The features can be divided into two categories: the number of injection tags and the number of HREF JavaScript references.
  1. *HTML Injection Tags*: A larger number of certain HTML tags can be an indication of malicious code including iframe tags, form tags, object tags, img tags, and script tags. The first three were identified by Wang et al. [4]. and the next two were found from a website that recommends ways to avoid Cross Site

HTML	JavaScript	CSS
http	http	background-image
https	https	@import
external-JavaScript	*iframe* tags	Behavior
iframe tags	XMLHttpRequests	-
script tags	External-JavaScript	-
object tags	*POST*	-
body tags	*GET*	-
img tags	Document.write	-
fromCharCode	innerHTML	-
form tags	fromCharCode	-
document.write	createElement('script')	-
\x	Total Keywords	-
\	Tabs.executeScript()	-
%	appendChild	-
!	%u	-
href=javascript	\x	-
CoinHive	\	-
-	!	-
-	CoinHive	-
-	Keyword-Text Ratio	-

Fig. 13.2 List of extracted features

Scripting (XSS) attacks [5]. Each of these tags can be used by attackers to launch XSS attacks. In these attacks, outside scripts (usually JavaScript files) can be injected into the webpage the user is on and silently execute malicious code in the background – this can include downloading user information or downloading malicious software to the user's computer. These tags can have a reference to an outside malicious script, thereby making the detection more difficult. To this end, we count the number of occurrences of these tags since a higher frequency of external JavaScript files can be an indicator of a malicious extension.

2. *HTML HREF JavaScript Calls:* In HTML code, one can create a link that doesn't redirect a user to another URL but instead executes some JavaScript code or function. This situation is indicated by the line href = javascript. Malicious code can be called and executed from this code functionality, meaning that the number of occurrences of this line can be an indicator of potential malicious code.

- **JavaScript Code Features**: Fig. 13.2 also lists the 20 features extract from the JavaScript code of a browser extension which belong to three categories: the number of XML Requests, DOM changes, and the keyword to content ratio.
  1. *JavaScript XML Requests*: In JavaScript, there is a function that allows the communication between the web browser and a server, this function is XMLHttpRequest which allows the webpage either to send information to a server – it could be for storage, manipulation, querying, etc. or to retrieve information from the webserver. This type of functionality is notated by the POST argument or GET argument, respectively. Attackers can use this function to perform a variety of malicious activities, such as sending a user's username and password to store on a remote server, redirecting the user to a malicious webpage, or performing SQL injection attacks. Because of these capabilities, we count the occurrences of both the function call, XMLHttpRequest, and also the occurrences of POST and GET to help the differentiation of malicious and benign browser extensions.
  2. *JavaScript DOM Changes*: JavaScript is a powerful language for web development, in large part due to its capability of accessing all aspects of the HTML Document Object Model (DOM). This power of being able to change practically any aspect of a webpage allows JavaScript to be used by attackers to add malicious scripts to webpages. In this study, we look for certain keywords and functions that add to webpages, and a higher frequency of these functions within the JavaScript code can indicate a malicious browser extension. These functions include the JavaScript appendChild, createElement('script'), document.write, and innerHTML functions. The first two functions were suggested by Pantelaios et al. [6]. Each of these functions have the capability to inject a malicious script into a webpage the user is on, which like the XSS attack, can download the user's information or download malicious software to the user's computer. Because of the potential of these functions for malicious extension detection, we count the occurrence of each function for analysis.
  3. *JavaScript Keyword Ratio*: As mentioned by Wang et al. [4], malicious code oftentimes has a lower amount of three key words: this, var., and if within the JavaScript code compared to the total amount of words in the

whole JavaScript file. This is because the attackers tend to use other functions and variable manipulation to hide the malicious functionality of the code [4]. Because of this, we count the number of occurrences of those three keywords and calculate the ratio of the keywords to total words for malicious extension detection. We also count the total number of these three keywords for each extension as a potential indicator for malicious code.

- ***C***ommon Features in HTML and JavaScript Code: Some signs and indications of malicious code are found commonly in both HTML and JavaScript code, which are described together in this section. The feature types are the number of different types of links, and obfuscation indicators.
  1. *Different Link Types:* Three different types of links are useful for determining whether an extension is malicious or benign, which are http links, https links, and external JavaScript file calls. Http links are links to any website that does not encrypt all traffic coming in and out of the website. Https links indicate that all traffic moving in and out of the website are encrypted which create more secure connections for end users. Finally, we look for the number of external JavaScript files being called throughout the HTML files, i.e. the calling and executing of a script that is hosted on an outside webserver – not part of the browser extension package.
  2. *Obfuscation Indicators:* Within the source code of HTML and JavaScript files, attackers oftentimes try to hide their malicious intent through code obfuscation. Obfuscation in this context refers to the changing and manipulation of the extension source code to hide the functionality of the code. Malicious files can convert their strings used for the code into vast lines of hexadecimal, binary, base64, or ASCII values that need to be decoded before they can be read or understood. This can also make the detection of malicious functionalities more difficult.
- **CSS Code Features**: The common XSS attack vectors suggest that three features from CSS files could be indicators of malicious code, which are the number of background-image occurrences, @import occurrences, and behavior occurrences. Each of these features within the CSS code can have a source of any URL, which means that an attacker could create the source of the CSS background-image, behavior, or @import as a malicious external JavaScript file, or malicious webpage to execute script on the user's computer.

Dataset Creation A python script was used to extract all 40 features from each collected browser extension and save the information into a CSV file for further analysis. In the CSV file, each row is a unique browser extension with each column for one of the 40 extracted features. After possessing this CSV file, we normalized the data with the min-max normalization.

13.2.3 Machine Learning Model Training and Testing

After the dataset was created, we used four machine learning algorithms, SVM, RF, KNN, and XGBoost, to build detection models based on the features described in the previous section. For each algorithm, we used a 5-fold cross validation technique for model training and testing. The metrics for performance evaluation are accuracy, precision, recall, and F1 score. All algorithms were tested with the same data partition in each test.

In the real world, there are much more benign browser extensions than malicious ones. Thus, our collected dataset is highly imbalanced as shown in Fig. 13.1 which could lead to the overfitting of machine learning algorithms. To solve this problem, we used the Synthetic Minority Oversampling Technique (SMOTE) [7] implemented by the Python package Imbalanced-Learn [8] to synthetically generate malicious samples.

We performed experiments to determine the best parameters for the machine learning algorithms. We tested the RF algorithm with the number of trees varying from 5 to 300. The out-of-bag (OOB) error was used as the metric for performance comparison, which is the difference between the OOB score and the accuracy. As shown in Fig. 13.3, RF with 175 trees achieves the lowest OOB error. We also tested KNN with different number of neighbors. Figure 13.4 shows the performance of KNN with 1 neighbor to 40 neighbors in terms of accuracy and F1 score. As can be seen in Fig. 13.4, both accuracy and F1 score rapidly increase from 1 neighbor to the maximum at 5 neighbors, then start a downward trend from 6 neighbors all the way to 40 neighbors. Thus, 5 neighbors was used for all tests of KNN.

13.3 Performance Evaluation Results

In this section, we present the performance evaluation results and determine the best algorithms for malicious browser extension detection. We also analyze the importance of each feature for the detection algorithms.

13.3.1 Performance Comparison

The performance evaluation results for the balanced and imbalanced datasets are shown in Figs. 13.5 and 13.6, respec-

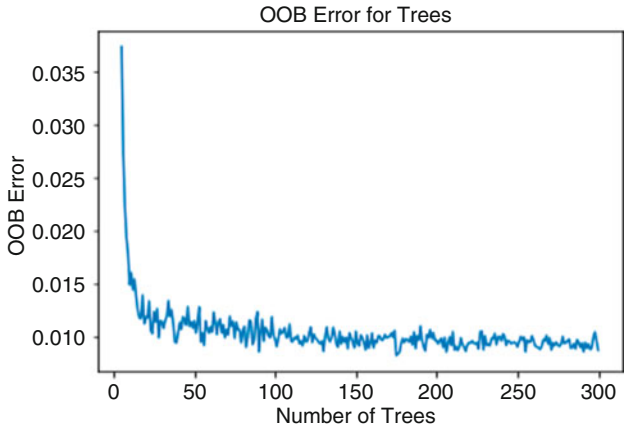

Fig. 13.3 OOB error rate in terms of the number of trees for RF

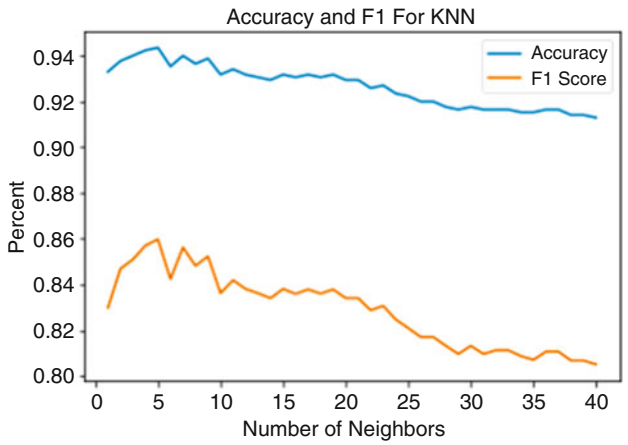

Fig. 13.4 Accuracy and F1 score in terms of the number of neighbors for KNN

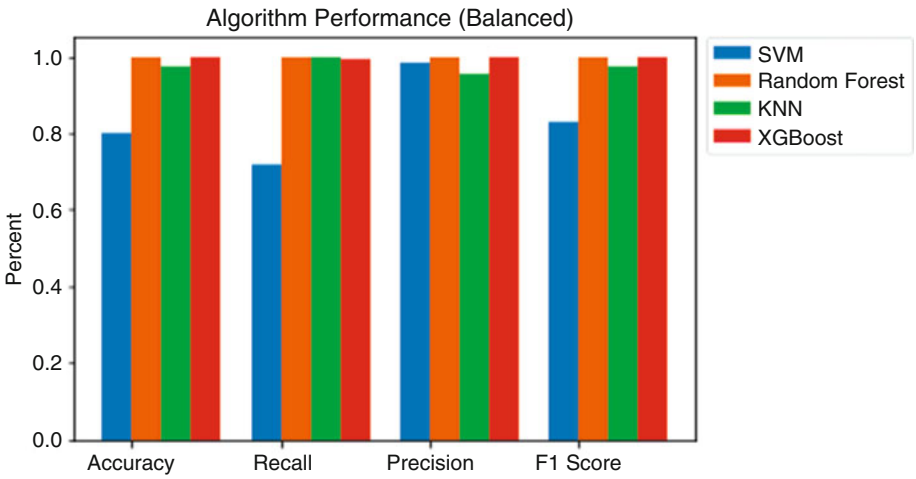

Fig. 13.5 Performance evaluation results for the balanced dataset

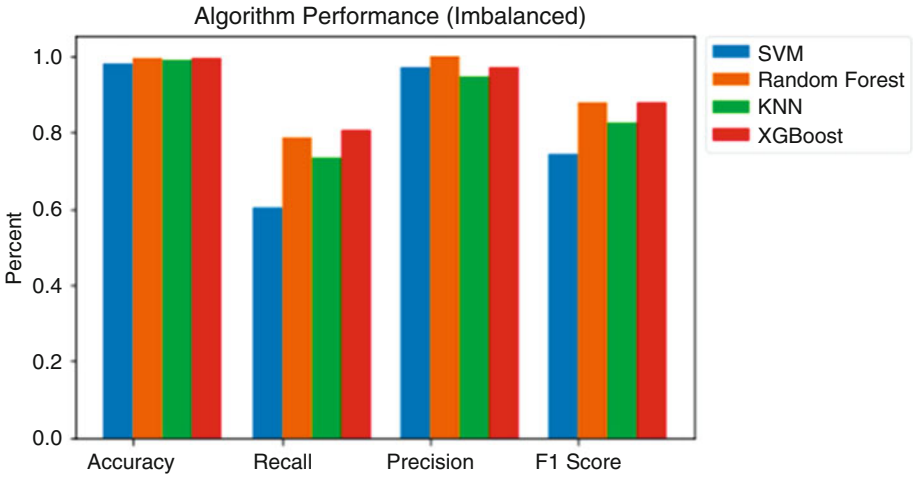

Fig. 13.6 Performance evaluation results for the imbalanced dataset

tively. The results are analyzed in the following based on each performance metric.

Accuracy Accuracy is calculated as the number of corrected classified extensions divided by the total number of extensions. Accuracy is not an appropriate metric to compare the performance of algorithms when the dataset is imbalanced as the algorithms will have the overfitting problem. Our results prove this problem that the four algorithms achieve high accuracy but low recall when the dataset is imbalanced. RF has the best accuracy of 99.79% for the balanced dataset.

Precision Precision is calculated as the number of true positives divided by the sum of true positives and false positives which indicates the ratio of correctly classified malicious browser extensions out of the extension being classified as malicious. RF is the best performed algorithm in terms of precision for both the balanced and imbalanced datasets. The precision scores of RF are 99.96% and 100% for the balanced and imbalanced datasets, respectively.

Recall Recall is calculated as the number of true positives divided by the sum of true positives and false negatives which indicates the ratio of correctly classified malicious browser extensions out of all malicious browser extensions. RF achieves the best recall score of 99.62% for the balanced dataset while XGBoost outperforms other algorithms for the imbalanced dataset with a recall score of 80.61%.

F1 Score This metric gives an insight into the general detection performance of an algorithm by combing precision and recall using the harmonic mean. The F1 score is especially useful when the dataset is imbalanced. The best performed algorithm in terms of the F1 score for the balanced dataset is RF (99.79%) followed by XGBoost (99.65%), KNN (97.52%), and SVM (83.13%). The best performed algorithm for the imbalanced dataset in terms of the F1 score is also RF (87.98%) followed by XGBoost (87.94%), KNN (82.95%), and SVM (74.31%). The results show that RF and XGBoost have similar performance for both datasets. Both of them significantly outperform other two algorithms.

13.3.2 FPR and FNR of Algorithms

In this section, we directly compare the false positive rates (FPR) and false negative rates (FNR) of the four algorithms to better understand their strengths and weaknesses.

FNR Because of the nature of our research goal – a method for detecting malicious browser extensions, we want the FNR as low as possible to minimize the possibility of malware infection. Thus, directly comparing the FNRs of the four algorithms helps to understand the effectiveness of an algorithm. Figure 13.7 compares the performance of the algorithms in terms of FNR for the balanced dataset. KNN has the best FNR of 0.36% followed by RF (0.38%), XGBoost (0.49%), and SVM (28.11%). The results for the imbalanced dataset are shown in Fig. 13.8. It can be seen that all algorithms have significantly higher FNRs for the imbalanced dataset due the overfitting problem. XGBoost has the best FNR of 19.38% followed by RF (21.36%) for the imbalanced dataset. The results demonstrate that the overfitting problem caused by the imbalanced dataset is solved by creating synthetic malicious samples with SMOTE.

FPR Although FPR is not as critical as FNR, we still want a detection algorithm to achieve a low FPR when the FNR is low. Figures 13.9 and 13.10 compare the FPRs of the algorithms for the balanced and imbalanced datasets, respec-

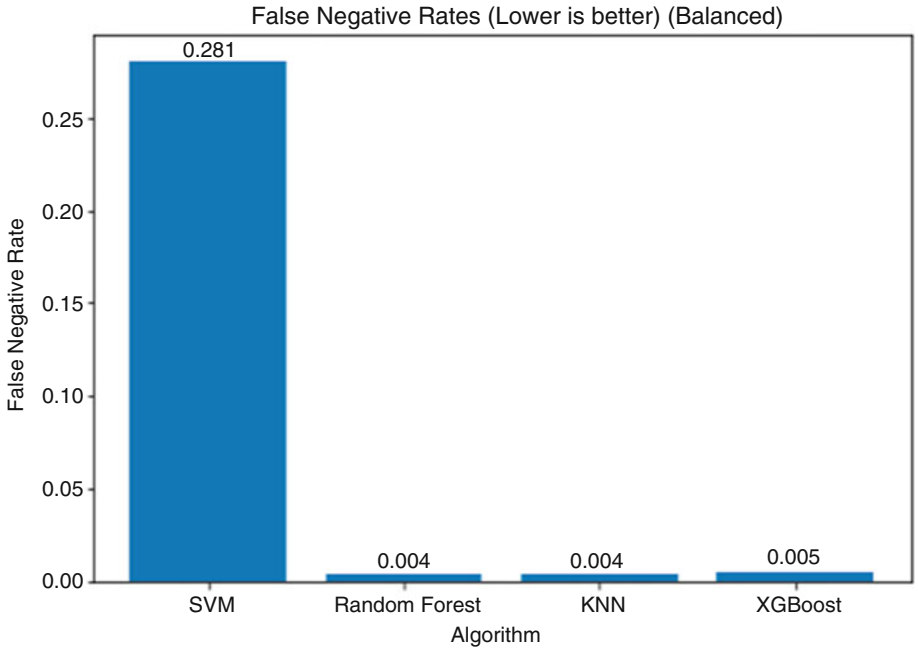

Fig. 13.7 FNRs of the algorithms for the balanced dataset

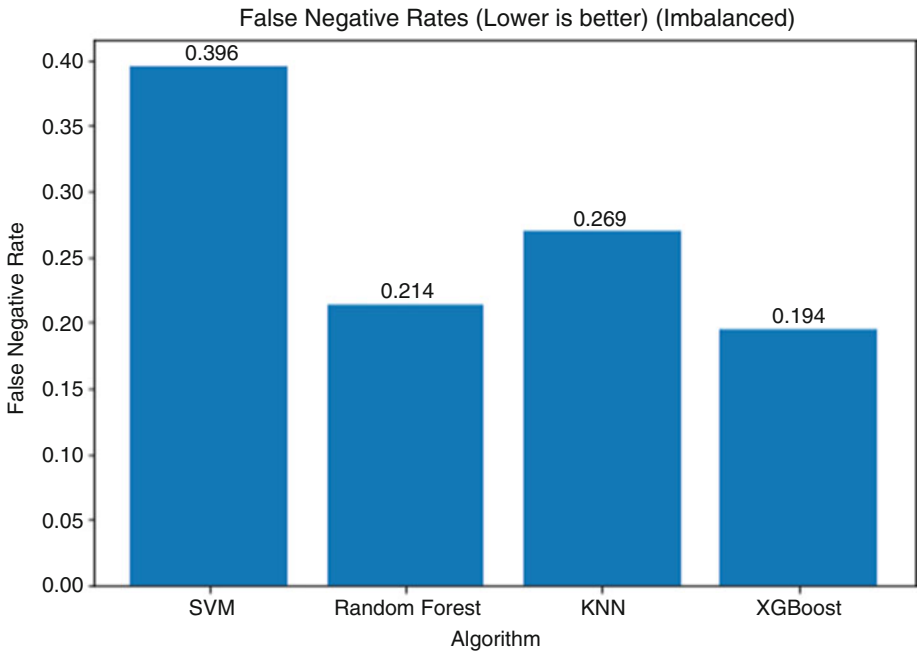

Fig. 13.8 FNRs of the algorithms for the imbalanced dataset

tively. For the balanced dataset, RF has the best FPR of 0.039% followed by XGBoost at 0.2%. KNN has the worst FPR of 4.48%. For the imbalanced dataset, RF achieves a perfect FPR followed by XGBoost at 0.0998%. KNN is still the worst performed one with a FPR of 0.17%.

13.4 Conclusion

Because of the popularity of browser extensions for web users, the prevalence of malicious browser extensions has increased which creates a need to effectively detect these mali-

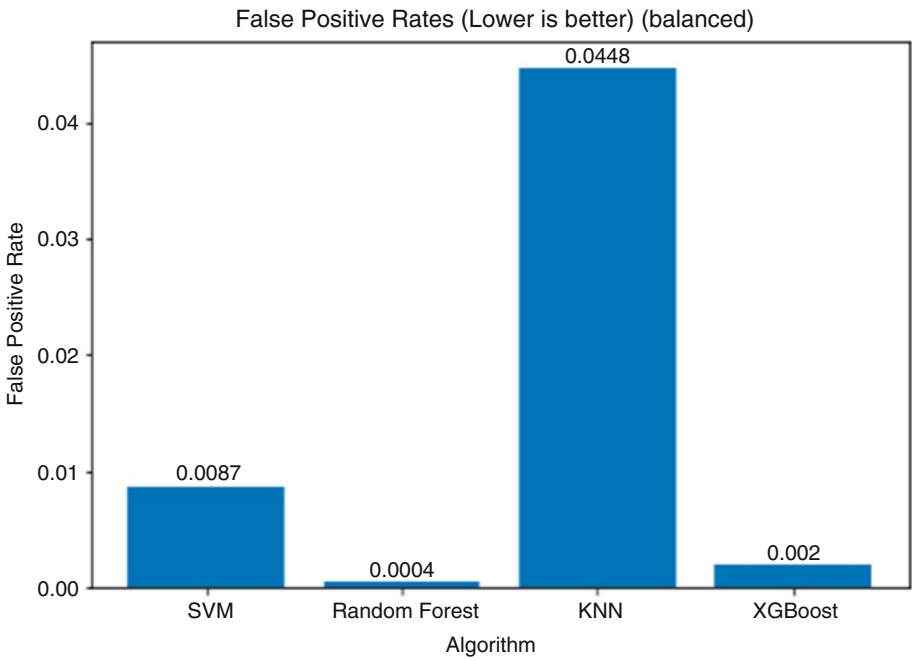

Fig. 13.9 FPRs of the algorithms for the balanced dataset

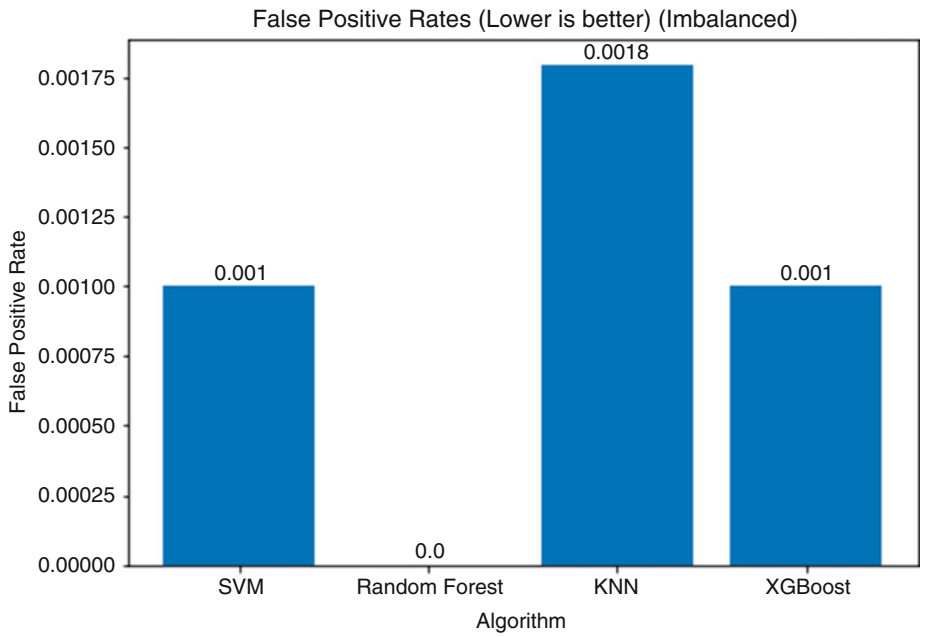

Fig. 13.10 FPRs of the algorithms for the imbalanced dataset

cious extensions for the protection of users and their information. This study explores the use of feature engineering and machine learning algorithms for detecting such malicious extensions. Our results show that the use of static code analysis for feature engineering and various machine learning algorithms can lead to an effective malicious browser extension detection method. The results also show that the overfitting problem caused by the collected imbalanced dataset can be solved by balancing the dataset with synthetically generated malicious samples.

Acknowledgements This work was supported by the National Science Foundation under grant CNS-2150145 and EPSCoR Cooperative Agreement OIA-1757207.

References

1. G. Varshney, S. Bagade, S. Sinha, Malicious Browser Extensions: A Growing Threat: A Case Study on Google Chrome: Ongoing Work in Progress, in 2018 International Conference on Information Networking (ICOIN), pp. 188–193 (2018)
2. S. Rauti, Man-in-the-browser Attack: A Case Study on Malicious Browser Extensions, in International Symposium on Security in Computing and Communications (SSCC), pp. 60–71 (2019)
3. https://cybernews.com/security/the-cybersecurity-threat-of-browserextensions
4. Y. Wang, W. Cai, P. Lyu, W. Shao, A combined static and dynamic analysis approach to detect malicious browser extensions. Secur. Commun. Netw. **2018**, 7087239 (2018)
5. G. Habibi, N. Surantha, XSS Attack Detection With Machine Learning and n-Gram Methods, in 2020 International Conference on Information Management and Technology (ICIMTech), pp. 516–520 (2020)
6. N. Pantelaios, N. Nikiforakis, A. Kapravelos, You've changed: Detecting Malicious Browser Extensions through Their Update Deltas, in The 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS'20), pp. 477–491 (2020)
7. N. Chawla, K. Bowyer, L. Hall, W. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
8. G. Lemaitre, F. Nogueira, C. Aridas, Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. J. Mach. Learn. Res. **18**, 1–5 (2017)