Dynamic Analysis Framework for Classifying Malicious Web Pages

1 Abstract

The growing reliance on web-based applications has necessitated robust and secure systems capable of safeguarding sensitive information. However, the complexity of these applications, particularly their use of JavaScript methods and properties, often leaves them vulnerable to exploitation. To address this, we propose a machine learning-based approach that detects malicious web pages based on their usage of JavaScript methods and properties. Our approach leverages a dataset extracted from various websites, detailing their usage of different JavaScript methods and properties. We employ a combination of Random Forest and XGBoost classifiers, tuned using GridSearchCV, to predict whether a website is malicious based on this data. This novel approach provides a real-time solution for detecting malicious web pages, offering a significant contribution to the field of cybersecurity. The expected outcome of this research is a functional machine learning model capable of accurately predicting malicious web pages. This not only enhances the security of web-based applications but also provides valuable insights into the usage patterns of JavaScript methods and properties across different websites.

2 Introduction

As we continue to delve deeper into the era of digitization, our reliance on the internet has become an inseparable facet of modern life. We access this vast cyberspace primarily through web browsers, a convenience that has its darker sides. This easy access has, unfortunately, opened the floodgates for cybercriminals who exploit the web to conduct illicit activities by crafting malevolent web pages. As highlighted by Kaspersky's 2021 report, around 25% of users encountered at least one web-based attack [1]. Concurrently, phishing, and crypto-related offenses were identified as the leading types of cyber threats, accounting for 85% of all identified threats that year, as reported by Cisco [2]. These rising instances of cyber-attacks underline the imperative need for innovative and effective methodologies to detect and neutralize malevolent web pages. Traditional detection techniques, including static and dynamic analysis, each present unique advantages but also pose considerable constraints. Static analysis, albeit simple, grapples with high false negative rates. On the other hand, dynamic analysis, despite providing comprehensive insights, is resource intensive. Moreover, existing machine learning and deep learning-based detection strategies frequently depend on single-modal features or elementary stacking of classifiers, thereby risking information loss and compromising the accuracy of the overall methodology.

Given these constraints, our study introduces a sophisticated multi-modal learning-based model to detect malicious web pages. This model employs machine learning algorithms, specifically Random Forest and XGBoost classifiers, and optimizes their performance using hyperparameter tuning through GridSearchCV. The model's robustness is further enhanced by leveraging an ensemble learning approach, which amalgamates predictions from both classifiers to deliver more accurate detection. Importantly, this method is not contingent on web content, such as text, and is therefore proficient in detecting malevolent web pages in multiple languages, thus broadening its universal relevance. The proposed novel model presents encouraging outcomes, indicating its efficacy in mitigating the continually expanding threat of web-based attacks.

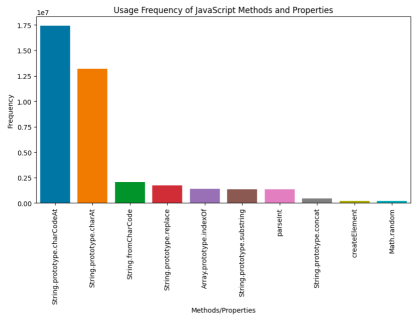
3 Related Work

The field of cybersecurity has seen a significant increase in research related to the detection of malicious web pages. Various machine learning models have been proposed and implemented to tackle this problem. In this section, we will discuss four such research papers that are closely related to our work. The first paper, titled "Detection and Analysis of Drive-by Download Attacks and Malicious JavaScript Code" [1], presents a comprehensive analysis of drive-by download attacks, a common method used by attackers to spread malware. The authors propose a method for detecting malicious JavaScript code, which is a key component of these attacks. This is particularly relevant to our work as our dataset includes features related to JavaScript methods and properties used by websites. However, our approach differs in that we use a machine learning model to predict the maliciousness of a website based on these features, rather than focusing solely on JavaScript code analysis. The second paper, "MM-ConvBERT-LMS" [2], introduces a novel machine learning model for text classification tasks. Although the model was not specifically designed for malicious URL detection, its architecture and techniques can be adapted for our purposes. our approach focuses on using ensemble machine learning model The third paper, "Detection of Malicious Web Pages Based on Hybrid Analysis" [3], proposes a hybrid analysis method for detecting malicious web pages. The authors combine static and dynamic analysis to extract features from web pages, which are then used to train a machine learning model. This approach is similar to ours in that we also extract features from web pages (i.e., usage of JavaScript methods and properties) to train our model. However, our work differs in the specific features used and the machine learning models employed. The fourth paper, "A Novel Approach for Malicious URL Detection Based on the Joint

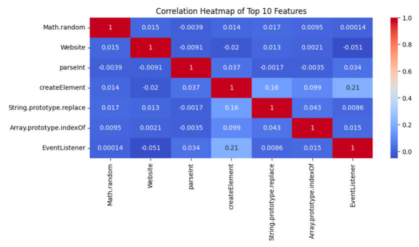
Model" [4], presents a joint model for malicious URL detection that combines lexical and host-based features. This approach is relevant to our work as it also involves predicting the maliciousness of a web page. However, our work differs in the features used (JavaScript methods and properties vs. lexical and host-based features) and the machine learning model employed. [3, 4, 5, 6].

4 Our Approach

Our proposed method for detecting malicious web pages involves using an ensemble machine learning model that combines the strengths of the Random Forest and XGBoost models. The ensemble model is trained on a dataset that includes various JavaScript methods and properties used by websites, as well as a binary target variable indicating whether each website is benign or malicious.



(a) Usage Frequency of JavaScript Methods and Properties top 10



(b) Correlation Heatmap of Top 10 Features

Figure 1: Usage Frequency and Correlation Heatmap

The Random Forest model is a powerful and versatile machine learning model that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees. For classification, two common criteria are used:

Gini Impurity: This is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. It is calculated as:

$$\text{Gini Impurity} = 1 - \sum (p_i)^2 \tag{1}$$

where p_i is the probability of an element being classified to a particular class.

Entropy: This is a measure of impurity, disorder, or uncertainty. The entropy of a set is zero when it contains instances of only one class. The entropy is calculated as:

$$\text{Entropy} = -\sum p_i * \log_2(p_i) \tag{2}$$

where p_i is the probability of an element being classified to a particular class.

The XGBoost model, on the other hand, is an implementation of gradient boosted decision trees designed for speed and performance. Gradient boosting is a method that goes through cycles to iteratively add models into an ensemble. It begins by initializing the ensemble with a single model, whose predictions can be naive. Then, we start the cycle: First, we use the current ensemble to generate predictions for each observation in the dataset. To make a prediction, we add the predictions from all models in the ensemble. These predictions are used to calculate a loss function (like mean squared error, for instance). Then, we use the loss function to fit a new model that will be added to the ensemble. Specifically, we determine model parameters so that adding this new model to the ensemble will reduce the loss.

The objective function to be optimized is given by:

$$\text{Obj}(\Theta) = \sum_{i=1}^{n} l(y_i, y^i) + \sum_{j=1}^{m} \Omega(f_j) \tag{3}$$

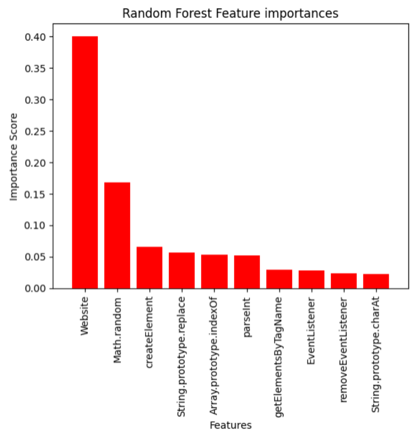
where l is a differentiable convex loss function that measures the difference between the prediction y^i and the target y_i, and Ω is a regularization term that penalizes the complexity of the model.

The prediction of the XGBoost model is given by a weighted sum of K base learners (decision trees):

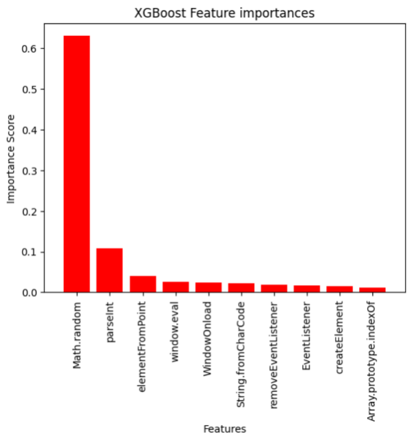
$$y^i = \sum_{k=1}^{K} f_k(x_i) \tag{4}$$

where f_k is a function in the functional space F, representing the structure of the decision tree.

(5)



(a) Random Forest Feature importances top 10



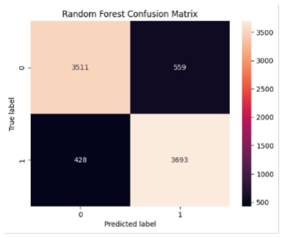
(b) XGBoost Feature importances top 10

Figure 2: Feature importances for Random Forest and XGBoost

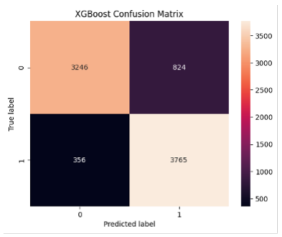
The ensemble model combines the predictions of the Random Forest and XGBoost models using a voting mechanism.

$$E(x) = \arg\max_{v} \sum_{j=1}^{J} I(v_j = v) \tag{6}$$

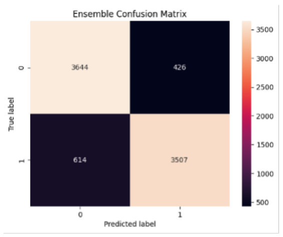
where $E(x)$ is the prediction of the ensemble model, v_j is the prediction of the j-th model, and I is the indicator function that equals 1 if $v_j = v$ and 0 otherwise.



(a) Random Forest Confusion Matrix

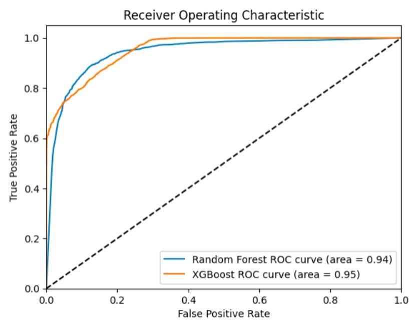


(b) XGBoost Confusion Matrix

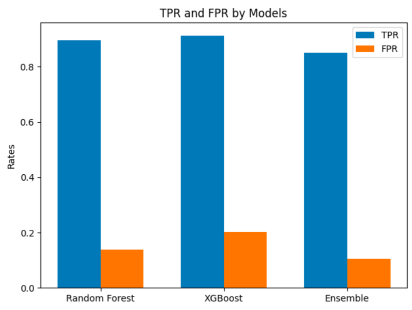


(c) Ensemble Confusion Matrix

Figure 3: Confusion Matrices for Random Forest, XGBoost, and Ensemble

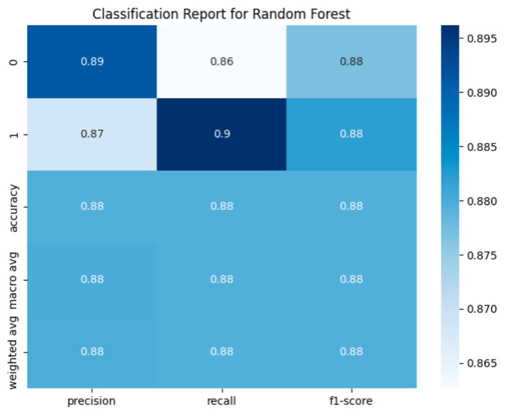


(a) Receiver Operating Characteristic for Random Forest and XGBoost

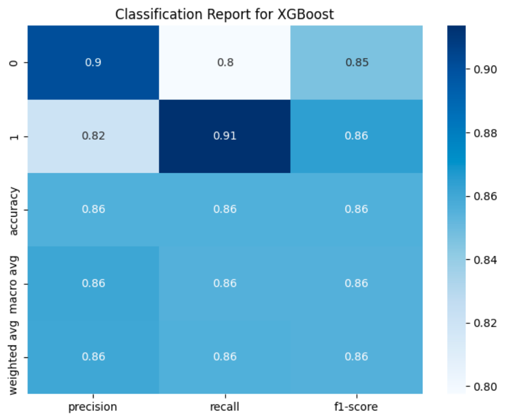


(b) TPR and FPR by Models

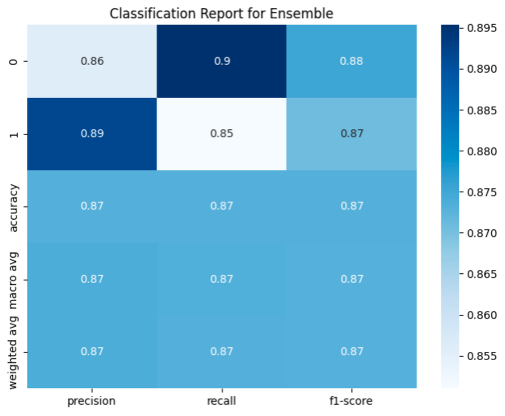
Figure 4: ROC and TPR/FPR for Models



(a) Classification Report for Random Forest



(b) Classification Report for XGBoost



(c) Classification Report for Ensemble

Figure 5: Classification Reports for Random Forest, XGBoost, and Ensemble

5 Experiment

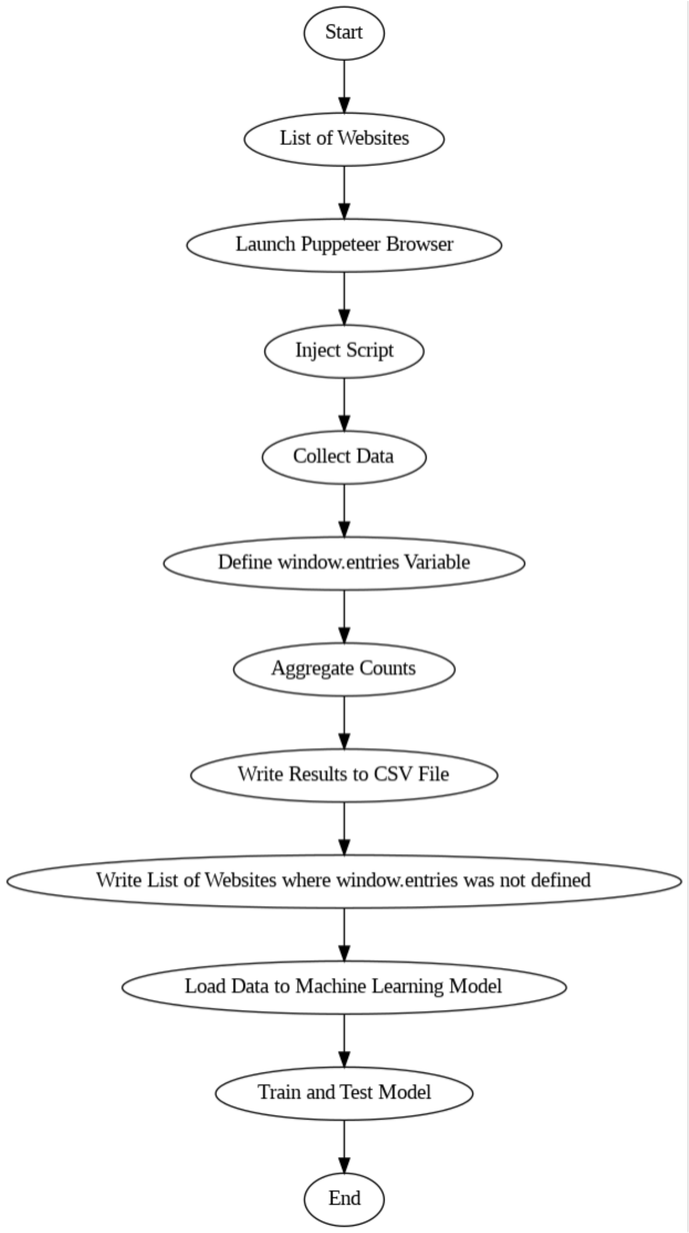
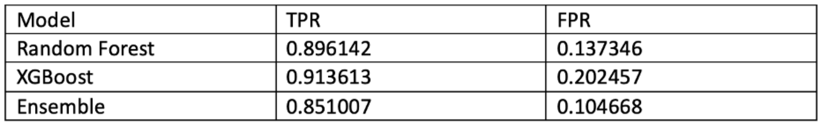

Start
List of Websites
Launch Puppeteer Browser
Inject Script
Collect Data
Define window.entries Variable
Aggregate Counts
Write Results to CSV File
Write List of Websites where window.entries was not defined
Load Data to Machine Learning Model
Train and Test Model
End

Figure 6: This flowchart provides a visual representation of the process your system follows to collect data, train the machine learning model, and test the model's performance.

The experiment conducted involves a two-part system designed to monitor and log the usage of various JavaScript APIs across a multitude of websites. The system leverages Puppeteer, a powerful Node.js library, to automate this process. Puppeteer provides a high-level API to control headless Chrome or Chromium browsers, enabling the automation of web tasks and the ability to interact programmatically with web content. The first part of the system is a script that is injected into each website visited by the Puppeteer browser. This script employs a technique known as "method hooking" or "function hooking", a concept used in programming to enhance or alter the behavior of a function at runtime. In this case, the script replaces various JavaScript functions with new functions that add two main behaviors: logging and counting. When a hooked function is called, the script logs a message to the console, providing a real-time trace of the function usage. Simultaneously, it increments a count associated with that function. This count serves as a quantitative measure of how frequently each API is used. These counts are presumably stored in a global object or variable, which acts as a storage for the data collected. The second part of the system is a script that acts as the orchestrator or the main driver of the system. It begins by reading a list of websites from a text file. For each website in this list, it launches a Puppeteer browser and navigates to the website. The script then waits for the injected script to define a window.entries variable. This variable is expected to contain an array of pairs, each pair consisting of an API name and a count. This data structure provides a snapshot of the API usage for each website. The script then aggregates these counts across all websites, creating a comprehensive view of API usage patterns. It writes the results to a CSV file, providing a structured and easily analyzable output. Additionally, it writes a list of websites where window.entries was not defined to a separate file. This could serve as a troubleshooting tool to identify websites where the injection of the first script or the data collection was unsuccessful. The dataset for this experiment was collected from the most popular websites listed on https://tranco-list.eu/ and the list of malicious websites from https://urlhaus.abuse.ch/. A web-addon was built for this experimentation. The task of this addon was to collect the count of browser APIs that the browser called when it visited a list of websites. This is an automated browser that installs an add-on as soon as it is initiated. The browser add-on then crawls through the list of websites provided to it. Two different instances are being used over here to collect data. One for malicious websites and other for benign websites. The data collected from these automated browsers are stored in excel files. The data transformation step after the collection of data involves, flagging of malicious and benign websites. This excel sheet is then concatenated and converted to excel files. After the data is transformed, it is loaded to the machine learning model, which in turn trains the model and then tests the model for accuracy and other evaluation metrics.



Model	TPR	FPR
Random Forest	0.896142	0.137346
XGBoost	0.913613	0.202457
Ensemble	0.851007	0.104668

Figure 7: All models TPR and FPR

6 Conclusion

In this paper, we proposed a method for detecting malicious web pages using an ensemble machine learning model that combines the strengths of the Random Forest and XGBoost models. Our model was trained on a dataset that includes various JavaScript methods and properties used by websites, with a binary target variable indicating whether each website is benign or malicious. Our experiment showed that our ensemble model performed well in correctly identifying both benign and malicious websites, achieving an accuracy of 0.873, a precision of 0.86 for benign websites and 0.89 for malicious websites, a recall of 0.90 for benign websites and 0.85 for malicious websites, and an F1-score of 0.88 for benign websites and 0.87 for malicious websites.

References

[1] Kaspersky Security Bulletin 2021. Statistics.

[2] 2021 Cisco Cyber security threat trends:.

[3] Research Article A Novel Approach for Malicious URL Detection Based on the Joint Model JianTing Yuan ,1 YiPeng Liu ,2 and Long Yu 3.

[4] Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code Marco Cova, Christopher Kruegel, and Giovanni Vigna University of California, Santa Barbara marco,chris,vigna@cs.ucsb.edu.

[5] Article MM-ConvBERT-LMS: Detecting MaliciousWeb Pages via Multi-Modal Learning and Pre-Trained Model Xin Tong 1, Bo Jin 1,2,*, JingyaWang 1, Ying Yang 2, Qiwei Suo 1 and Yong Wu 3

[6] Detection of malicious web pages based on hybrid analysis Rong Wang , Yan Zhu , Jiefan Tan , Binbin Zhou.

[7] Malicious website identification using design attribute learning Or Naim1 · Doron Cohen1 · Irad Ben-Gal1

[8] BINSPECT: Holistic Analysis and Detection of Malicious Web Pages Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam Fondazione Bruno Kessler (FBK-IRST), Trento, Italy eshete,adolfo,sisai@fbk.eu

[9] A NOVEL APPROACH FOR ANALYZING AND CLASSIFYING MALICIOUS WEB PAGES Panchakshari Nijagunashivayogi Hiremath

[10] A Malicious Mining Code Detection Method Based on Multi-Features Fusion Shudong Li, Member, IEEE, Laiyuan Jiang, Qianqing Zhang, Zhen Wang, Zhihong Tian, Senior Member, IEEE, and Mohsen Guizani, Fellow, IEEE,

[11] Understanding the Web browser threat: Examination of vulnerable online Web browser populations and the ”insecurity iceberg”

[12] Google. Safe Browsing API. http://code.google.com/apis/safebrowsing/.

[13] A. Ikinci, T. Holz, and F. Freiling. Monkey-Spider: Detecting Malicious Websites with Low-Interaction Honeyclients. In Proceedings of Sicherheit, Schutz und Zuverlässigkeit, April 2008.

[14] Likarish P , Jung E , Jo I . Obfuscated malicious javascript detection using classifi- cation techniques. In: Malicious and unwanted software (MALWARE), 2009 4th international conference on. IEEE; 2009

[15] A static approach to detect drive-by-download attacks on webpages. In: Control communication and computing (ICCC), 2013 international conference on. IEEE; 2013

[16] Prophiler: a fast filter for the large-scale detection of malicious web pages

[17]

[18]

[19]

[20]