A DYNAMIC ANALYSIS FRAMEWORK FOR CLASSIFYING MALICIOUS WEBPAGES

Thesis

Submitted to

The College of Arts and Science of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree of

Master of Science in Computer Science

By

Allen Varghese

Dayton, Ohio

December, 2023

A DYNAMIC ANALYSIS FRAMEWORK FOR CLASSIFYING MALICIOUS WEBPAGES

Name: Varghese, Allen

APPROVED BY:

ABSTRACT

A DYNAMIC ANALYSIS FRAMEWORK FOR CLASSIFYING MALICIOUS WEBPAGES

Name: Varghese, Allen
University of Dayton

Advisor: Dr. Phu Phung

In today's interconnected digital landscape, the surge in malicious websites has caused a significant number of cyber-attacks and data breaches. These malicious entities largely employ JavaScript to execute attacks on web browsers. It is becoming increasingly apparent that attackers can evade traditional mechanisms, such as lexical analysis, content examination, and blacklists, through code obfuscation, which disguises the true intent of the code, and polymorphic or metamorphic code that alters itself with each execution. These techniques make it difficult for traditional static analysis tools to detect dynamically generated or altered code characteristics of sophisticated, evolving threats. Considering these challenges, notable research has progressed, proposing dynamic approaches that monitor JavaScript behavior in real-time. These dynamic methods can identify malicious patterns and activities, offering a significant advancement over static analysis by detecting and mitigating threats as they occur.

This thesis introduces an innovative runtime analysis method for JavaScript that encompasses all JavaScript executions, including traditionally on-the-fly generated code and advanced evasion techniques. Our approach centrally applies the security reference monitor technique, which mediates JavaScript's security-sensitive operations during execution.

This includes closely monitoring function calls and property access, ensuring a thorough capture of runtime behaviors, and effectively mitigating the risk of attack, regardless of the code's structure or the obfuscation techniques employed. We have implemented this method as an extension in the Chromium browser to intercept and log about 40 security-sensitive JavaScript operations, demonstrating its applicability in a real-world web browsing environment.

To evaluate the effectiveness of our framework, we have developed a toolset to automate the execution of the Chromium browser with our extension on a large-scale raw dataset of approximately 13,900 malicious and 13,500 benign websites. We counted the number of executions of each operation for each website as features and collected approximately 27,000 labeled records to train machine learning models. Preliminary results underscore the effectiveness of our approach, pinpointing malicious JavaScript content with a promising accuracy rate.

This work is dedicated to the silent warriors, who, far from home, seek knowledge and growth in lands not their own. You confront the shadows of solitude and the heartache of homesickness with unyielding determination and resolve. May this acknowledgment remind you that your struggles are seen, your tenacity is admired, and even in the quietest moments, you are never alone.

ACKNOWLEDGMENTS

I am thankful to Dr. Phu Phung and Dr. Ahmed El Ouadrhiri for their guidance and support during my master's program. Their insights and advice were valuable in the research and writing of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Overview

The web, as a digital ecosystem, is a vast expanse of constant evolution. With the rise of technologies like JavaScript, the landscape has seen both innovation and exploitation [1]. Web technologies drive modern platforms, but they also introduce vulnerabilities, making cybersecurity a top concern. As these technologies advance, cyber threats keep pace, necessitating more robust defense mechanisms. The detection of malicious content is crucial to web security. Recent strategies employ a hybrid analysis that combines static and dynamic approaches, utilizing machine learning classification algorithms to determine between benign and malicious content. This hybrid method is complemented by dynamic checks for malicious activity during the execution of web pages, a practice which has proven effective against the sophisticated threat of shellcode within dynamic web scripts [2]. Sophisticated threats, especially those embedded within dynamic web scripts, can bypass these conventional measures [3]. The adaptability of malicious actors further compounds the challenge, leading to a constant cybersecurity arms race. Dynamic analysis of web content, especially JavaScript, offers a potential solution. By examining scripts in real-time, as they execute, it's possible to capture malicious patterns that static analysis might miss [4]. Various frameworks and tools, such as EVILSEED, have explored this avenue, focusing on the dynamic characteristics of malicious web pages [5]. Instead of solely relying on static assessments, the suggested approach emphasizes the observation of websites in real-time. This method strives to recognize harmful trends that could go unnoticed in conventional techniques. By focusing on interactions, especially with Browser APIs, the research aims to reveal a deeper level of web conduct, which is considered crucial in detecting malicious motives [6]. Machine learning plays a pivotal role in enhancing this dynamic analysis approach. By training models on data sets derived from web interactions, this research aims to develop a detection mechanism that's both precise and adaptable [4]. Given the intricate nature of modern web threats, a combination of dynamic analysis and machine learning promises to offer a more holistic detection system. The need for such a system is evident. The research problems addressed here include the limitations of conventional methods, which though valuable, often fall short when faced with advanced, obfuscated threats. This research posits that a more comprehensive, real-time analysis, augmented by machine learning, can bridge this gap, offering a detection system that's both robust and adaptive.

1.2 Thesis Motivation and Hypothesis

As the web grows more complex, so does its array of vulnerabilities, necessitating a more refined approach to cyber-security. JavaScript, with its flexibility and ubiquity, represents a significant point of concern. Its power enables a plethora of functionalities, both benign and malicious [7]. The dual nature of JavaScript, as a tool for development and a weapon for exploitation, underscores the need for specialized detection mechanisms tailored to its unique challenges. This research hypothesizes that real-time analysis of web interactions can provide deeper insights into potential threats [6]. Traditional, static methods might miss subtle behavioral patterns indicative of malicious activities. In contrast, a dynamic approach, which captures web behavior as it unfolds, promises to be more revealing. Incorporating machine learning into this dynamic framework is anticipated to enhance its effectiveness [4]. By analyzing and learning from real-time web interactions, machine learning models can potentially offer a more nuanced detection mechanism, one that evolves with the threat landscape. In essence, this thesis operates on the premise that a combination of browser automation, real-time analysis, and machine learning can outpace and outperform conventional malicious content detection techniques. The aspiration is to create a system that's not only effective but also adaptive, keeping pace with the ever-evolving web threat landscape.

1.3 Contributions

This research carves out multiple avenues of contribution in the realm of malicious content detection, the contributions are as follows:

1.3.1 Introduction of a Novel Methodology

This research introduces a distinct methodology that synergizes browser automation with dynamic web analysis, promising to provide a granular level of insight into web interactions, unseen by conventional methods [6].

1.3.2 Creation and Curation of a Significant Dataset

A significant contribution is the dataset curated specifically for this research, drawn from renowned repositories, representing both benign and malicious web interactions and serving as the foundation for the machine learning models developed in this study [4].

1.3.3 Integration of Machine Learning

The integration of machine learning into the framework, utilizing the interaction-based dataset to train, refine, and test multiple models, aiming to offer a precise and scalable detection mechanism [4].

1.3.4 Comparative Analysis Against Existing Frameworks

This research is contextualized and contrasted against existing frameworks, highlighting its advantages, potential areas for improvement, and unique facets, ensuring the research remains grounded and aligned with the broader cybersecurity discourse [1, 5, 8].

1.3.5 Comprehensive Guide to Methodologies, Datasets, Experimental Setups, and Evaluations

This thesis serves as a comprehensive guide detailing methodologies, datasets, experimental setups, and evaluations, positioning itself as both a resource for those in the domain and a foundation for future research.

CHAPTER II

BACKGROUND AND RELATED WORK

2.1 Background

2.1.1 Evolution of Malwares

The rise of malicious JavaScript necessitated the development of sophisticated detection mechanisms. Methodologies focusing on semantic analysis are instrumental in discerning malicious intent and enhancing protective measures against relentless threats, showcasing the dynamic nature of cybersecurity evolution [9]. The relentless pursuit of innovative solutions has led to significant advancements in the field, providing a robust response to the ever-evolving malware threats [10].

Innovative approaches in cybersecurity have been pivotal, integrating machine learning techniques to create a holistic and adaptive approach to emerging threats [11]. The emphasis on the static detection of obfuscated malicious JavaScript has been crucial in developing refined and precise mechanisms [12]. The constant evolution of threats necessitates continuous refinement of detection strategies, with advancements being integral to maintaining the security and integrity of cyberspace [13]. The inception of drive-by key extraction attacks highlighted the intricate nature of evolving cyber threats [14]. A comprehensive approach to analyzing and classifying malicious web pages became integral, addressing the increasingly sophisticated and adaptable cyber threats.The development of advanced protective measures and the constant enhancement of defensive strategies are emblematic of the ongoing evolution in cybersecurity measures, ensuring the proactive mitigation of threats [15].

Efficient detection and prevention of advanced cyber threats represent the significant progress made in the development of strategic responses [15]. The introduction of strength analysis marked a significant development in detecting suspicious malicious websites, combating JavaScript obfuscation, and enhancing the reliability of web security measures [16]. These advancements underscore the continuous efforts in refining cybersecurity measures, aiming to stay ahead of the evolving threats and ensuring the protection of the digital realm [17].

2.1.2 Challenges in Detecting Malicious JavaScript Content

Detecting malicious JavaScript poses multifaceted challenges, primarily due to JavaScript's dynamic and versatile nature. Sophisticated techniques, such as obfuscation, allow malicious code to evade traditional detection mechanisms, rendering static analysis often insufficient [18]. [12] enhanced detection method leverages the combination of static analysis and lightweight run-time to prevent the execution of the obfuscated malicious JavaScript code in browsers. The necessity for dynamic analysis is becoming increasingly evident to counter advanced evasion techniques, playing a crucial role alongside static analysis [9]. It observes JavaScript's behavior in real-time, discerning malicious activities that static analysis might overlook. The effectiveness of dynamic analysis is based upon the accuracy and comprehensiveness of the applied analysis methods [19].

Malicious web advertising further complicates the detection of malicious JavaScript. It frequently employs JavaScript to deliver exploit kits and malvertisements, leveraging legitimate advertising networks and websites [7]. Differentiating between benign and malicious advertising content, especially when delivered via reputable platforms, is challenging and necessitates advanced detection strategies [20].

Moreover, the efficiency and effectiveness of detection mechanisms are of paramount importance. The rapid proliferation of malicious websites demands scalable solutions capable of handling vast web scales [19]. Solutions must strike a balance between reducing false positives and ensuring that malicious content does not slip through the detection net, underlining the perpetual challenges in malicious JavaScript content detection [9].

2.1.3 Dynamic Analysis

Dynamic analysis has emerged as a pivotal component in facilitating profound insights into system behaviors and intricate interactions in real-time. It delves deep into the complexities of malware, revealing evasion techniques and malicious functionalities that remain concealed within seemingly benign codes [21]. Through meticulous examination of runtime behaviors, dynamic analysis uncovers sophisticated tactics deployed by malware to bypass detection, thus enabling the development of more robust and resilient anti-malware frameworks [22]. It is indispensable for understanding the diversified nature of modern threats and fortifying defense mechanisms against them.

In this modern technological era, dynamic analysis has found its applications extending to diverse platforms, including Android. Innovations such as IntelliDroid are testament to the effectiveness of dynamic analysis in mobile platforms, utilizing targeted input generation to dissect and analyze the intricacies of Android malware [23]. This targeted approach enables a comprehensive understanding of vulnerabilities and potential threats inherent within mobile applications, thereby enhancing the security posture of mobile ecosystems against malicious infiltrations [24]. Consequently, it is crucial to tackle the security concerns that emerge from the increasing utilization of mobile devices and safeguard the confidential data they contain

Further advancements in dynamic analysis methodologies have broadened their scope to the rapidly evolving field of the Internet of Things (IoT). Techniques that employ Convolution Neural Network Models are optimizing the detection of malicious entities in interconnected devices [25]. These innovations play a crucial role in strengthening the security frameworks of IoT ecosystems against a spectrum of emerging threats and vulnerabilities [26]. Given the pervasive nature of IoT devices and their integration into various aspects of our lives, enhancing the security measures for these devices is of paramount importance.Dynamic analysis marks a transformative phase in the evolution of cybersecurity paradigms. It reinforces defense mechanisms against the multifaceted and sophisticated nature of contemporary threats while laying down the foundation for continual advancements in security methodologies. The profound and comprehensive insights derived from dynamic analysis are pivotal for devising adaptive and advanced security strategies, ensuring the resilience and integrity of systems in the dynamic cyber landscape [27].

2.1.4 Machine Learning in Malicious Content Detection

Machine learning, a subset of artificial intelligence,has seen extensive application in the field of cybersecurity. The work in [28] leverages machine learning to classify URLs as malicious or benign, emphasizing the pivotal role of feature selection and extraction in enhancing model performance. It discusses various techniques and methodologies, highlighting the importance of an interpretative model for better understanding and improved accuracy.Another notable contribution is [29], where machine learning is deployed for detecting malicious web content gives an overview of the algorithms, their applications, and effectiveness in classifying web content. The complexities in dealing with adversarial environments are discussed in [30], focusing on the evolution of methodologies and the constant battle between intrusion detection systems and evasive malware.

The detection of social spammers has been another area where machine learning has been extensively applied. The integration of social honeypots and machine learning in [31] demonstrates the effectiveness of machine learning models in identifying and mitigating risks posed by social spammers. Automatic analysis of malware behaviour is also explored in [32], where machine learning models are trained to understand and classify malware behaviours, emphasizing the significance of feature engineering and model selection. [33], introduces a unique approach to detecting malicious JavaScript using a fixed-length vector representation. This methodology facilitates the efficient classification of scripts, contributing to broader efforts in malicious content detection.The convergence of machine learning with other cybersecurity measures has led to the development of robust and sophisticated mechanisms, capable of identifying intricate malicious patterns and behaviours in web content.

2.2 Related Work

2.2.1 Existing methodologies and their limitations

The evolution of cybersecurity measures is crucial in comprehending the advancements in malicious content detection methodologies. Over time, the scope of cybersecurity has expanded, adopting complex strategies to counteract evolving threats. The methodologies have seen a gradual transformation, with each iteration attempting to address the limitations of its predecessors. However, even the most sophisticated methodologies harbor inherent limitations, necessitating continuous refinement and innovation.Machine learning techniques have become pivotal in malware analysis and detection, as they offer robust and scalable solutions [34]. These techniques, while diverse and adaptable, are not without constraints. The accuracy of detection can be significantly impacted by the quality of the training data, choice of features, and the selection of appropriate algorithms. The limitations in current methodologies underscore the need for more nuanced and sophisticated approaches.

A distinctive approach focusing on malicious JavaScript detection employs Fixed Length Vector Representation [33]. This methodology, while novel, is constrained by its reliance on the representational adequacy of fixed-length vectors. The subtle intricacies of malicious scripts may elude this representation, leading to potential oversights in detection.In the domain of email spam filtering, machine learning has been instrumental [35]. The methodologies employed have significantly evolved, leveraging advanced algorithms to discern patterns indicative of spam. However, these methodologies are continually challenged by the evolving tactics employed by malicious entities, necessitating constant refinement and enhancement of detection strategies.The application of machine learning for detecting JavaScript-based attacks has seen innovative developments [36]. By leveraging AST features and paragraph vectors, this methodology aims for precision in detection. However, the dynamic nature of JavaScript-based attacks and the constant evolution of attack vectors impose inherent limitations on the existing methodologies.Structural feature extraction methodology has been developed to detect malicious office documents [37]. While this methodology provides a structured approach to feature extraction, it is confined by its ability to adapt to the varying structural elements present in malicious documents. The ever-evolving complexity of malicious documents demands a methodology that is both flexible and comprehensive.

Table 2.1: Comparison of studies

Study	Experimental setup	Subjects: Client-side	Method: Dynamic analysis	Topic of study	Dynamic behavior	Objects & properties	Functions & calls	Code inclusion & generation	Other
Yue and Wang(2009)		✓	✓		✓			✓	
Jang et al. (2010)		✓	✓			✓		✓	
Ratanaworabhan et al. (2010)		✓	✓				✓	✓	✓
Richards et al. (2010)		✓	✓				✓	✓	✓
Richards et al. (2011b)		✓	✓						✓
Ocariza Jr. et al. (2011)		✓	✓						
Nikiforakis et al. (2012)		✓	✓						✓
Son and Shmatikov (2013)		✓	✓						
Behfarshad and Mesbah (2013)		✓	✓						
Nederlof et al. (2014)		✓	✓						
Pradel and Sen (2015)		✓	✓						
Wei et al. (2016)		✓	✓						✓
Selakovic and Pradel (2016)		✓	✓		✓	✓		✓	
Our Framework		✓	✓		✓	✓	✓	✓	✓

Table [2.1] provides a comprehensive comparison of various studies on JavaScript programs spanning from 2009 to 2016, alongside the methodologies and findings of the present research. The first part of the table focuses on the experimental setup of each study, detailing whether the subjects of the research were client-side or the method used was dynamic analysis. Most studies, including the present research, leverage dynamic analysis. The second segment of the table categorizes the main topics of study, emphasizing dynamic behavior, objects and properties, functions and calls, code inclusion and generation, and other. A quick glance at the table reveals the breadth and depth of each research endeavor, with checkmarks indicating the areas of focus. Notably, the present research is comprehensive, encompassing several facets of JavaScript analysis, reaffirming its significance and contribution to the field.

The exploration of existing methodologies and their limitations in the field of malicious content detection provides a comprehensive understanding of the state of the art. It is crucial to scrutinize these methodologies to discern the gaps and opportunities for enhancement and innovation.

2.2.2 Frameworks and tools focused on dynamic characteristics of malicious web pages

Prophiler provides a fast filter for large-scale detection of malicious web pages [13]. It focuses on the dynamic characteristics of web pages, enhancing the detection process. Despite its advancements, the continuous evolution of malicious web pages and their dynamic nature require ongoing improvements and optimizations. Machine learning has been instrumental in detecting malicious web content, providing a framework that allows for continuous learning and adaptation [29]. The dynamic characteristics of web pages present ongoing challenges, necessitating constant refinement of the detection methodologies and the development of more sophisticated tools and frameworks.

MalDAE offers an innovative approach, focusing on the correlation and fusion of static and dynamic characteristics to detect and explain malware [38]. The integration of static and dynamic analysis enhances the robustness of malware detection, allowing for more comprehensive insights into malicious activities.CBM provides a free, automatic malware analysis framework using API call sequences [39]. This framework is pivotal in understanding the dynamic behavior of malicious software, allowing for the development of more efficient detection methodologies. The continuous advancements in malware require the constant refinement of such frameworks.

MalInsight offers a systematic profiling-based malware detection framework [40]. It enables a deeper understanding of the dynamic characteristics of malicious software, contributing to the development of more sophisticated detection methodologies. The ongoing evolution of malicious software necessitates continuous research and development in this domain.The development of malware-aware processors presents a framework for efficient online malware detection [41]. This approach focuses on the dynamic behavior of malicious software, providing real-time insights into malicious activities. The advancements in this domain are pivotal in countering the evolving threats posed by malware.An analysis of the dynamic behavior of JavaScript programs provides insights into the intricacies of malicious web pages [42]. This analysis is crucial in understanding the evolving nature of web-based threats, allowing for the development of more advanced detection methodologies. The dynamic characteristics of JavaScript programs require ongoing research and development efforts.

This thesis focuses on the challenges of detecting malicious JavaScript content, using both dynamic analysis and machine learning to identify threats in real time. The approach is to go deeper into JavaScript, uncovering malicious patterns and intentions that are often missed by standard methods. It is built on existing research but aims to fill in the gaps where previous studies may not have delved as deeply. We've developed and implemented an browser extension to collect relevant data. This code is automated by another set of instructions. The data collected through these steps is used to train Machine Learning models. This method can be refined without much hassle to stay ahead of new and evolving threats. Table [2.2] gives an overview of the shortcomings of the existing work and how this thesis addresses it.

Table 2.2: Comparison of Existing Work and Our Approach in Addressing Malicious Code

Aspect	**Shortcomings in Existing Work**	**How Our Work Addresses It**
Obfuscation and Evasion	Malicious scripts frequently utilize obfuscation and evasion techniques to thwart detection, altering behaviors during code execution [28].	Our browser extension, actively monitors and logs potentially malicious activities at the client-side in real-time, thereby capturing malicious scripts and activities as they occur, including those deploying obfuscation and evasion techniques.
Dynamic Nature of Code	Malicious code, especially when employing obfuscation and evasion techniques, can dynamically alter its pattern to avoid detection (Polymorphic and Metamorphic Code, Encoding Data, Conditional Checks) [36].	Our method involves collecting and logging script activities, ensuring that malicious script behaviors are captured even when patterns change or evasion techniques are utilized. Additionally, Our machine learning models, utilize this dataset to identify potential threats.
Dynamic Analysis	Existing dynamic analyses can face difficulties, particularly against advanced malware that employs sophisticated evasion and anti-analysis techniques during execution (Environment Awareness, Time-based Evasions, Dynamic Code Loading, Anti-Debugging) [27, 22].	Our approach emphasizes real-time, client-side logging of script activities, ensuring that behaviors are captured as they occur, providing a form of dynamic analysis that observes real-time interaction and actions, thus capturing activities even if malicious scripts employ anti-analysis strategies during execution.

CHAPTER III

PROPOSED FRAMEWORK

3.1 Overview of Framework

In this framework, we concentrate on detecting and analyzing malicious scripts in real-time. Real-time monitoring ensures immediate detection and potentially, reduces the severity of threats. These scripts, which are designed with evasion in mind, can morph their appearance or behavior, making them difficult targets for periodic or outdated scanning mechanisms. This results in a security blind spot, which leaves the system exposed to potential threats. Static analysis has historically been a stalwart among the countless methods. However, its effectiveness wanes when confronted with dynamic scripts. Such scripts are unpredictable and can modify their characteristics and functional behaviors on the fly. With real-time monitoring, it becomes easy to detect threats as they emerge, which narrows down the time frame a malicious script takes to execute itself. There are many ways to analyze malicious scripts running on a website. A few of them are browser extensions, custom browsers, JavaScript hooks, Content Security Policy, etc. In this research, we are using a browser extension to analyze web pages for malicious content. With a browser extension, the data collection mechanism can be easily managed and is transparent, and it becomes easy to have access to a broad range of browser APIs. This comes in handy when we must customize the working of the extension. The real-time analysis of malicious scripts must be done on multiple websites instead of a few. With the help of browser automation, this process can be streamlined. This ensures a comprehensive dataset that represents a wide spectrum of web content. This ensures consistency by eliminating the variability introduced by manual browsing. Capturing real-time script behaviors helps in accumulating rich data that can be used for analysis purposes. The vast amount of data that is generated using the browser extension by running it on multiple websites can be used to feed the classification-centric machine learning models of the framework. This is one of the fortifying blocks of the real-time detection mechanism of the framework. This model serves as an analytical backbone, categorizing scripts into 'malicious' or 'benign'.

3.1.1 Browser Extension:

Upon the start, the add-on initializes certain event listeners. This crucial step ensures that the add-on is set to monitor the activities of scripts in real time. This initialization is important because it guarantees that any script activity is captured right from the time a page starts loading.

Event listeners in JavaScript are a way to detect specific actions or events that occur. For example, this code demonstrates the real-time monitoring of a webpage's loading event. By setting up this hook, any time a webpage finishes loading, the system is immediately aware and can capture and log this event in real-time.

```
old_WindowOnload = window.onload;
window.onload = function () {
console.log("Window.onload is being monitored");
const obj = this;
const args = arguments;
old_WindowOnload.apply(obj, args);
addtoCount('WindowOnload') + 1;
}
```

This code intercepts the window.onload event. Initially, it saves the original event function in old. Then, it replaces the original event with a custom function that, when executed, logs a message indicating the monitoring of the window.onload event. Within this custom function it uses the original window. onload event, ensuring any original actions still occur. Additionally, it calls the addtoCount function to keep a count of how many times the window.onload event has been triggered. There are many browser APIs that are being misused by attackers to track users and upload malicious payloads.

'HTMLCanvasElement.prototype.toDataURL' is used by attackers for fingerprinting attacks to identify and track users.

```
old_canvas_toDataURL = HTMLCanvasElement.prototype.toDataURL;
HTMLCanvasElement.prototype.toDataURL = function () {
console.log("HTMLCanvasElement.prototype.toDataURL is being monitored");
const obj = this;
const args = arguments;
old_canvas_toDataURL.apply(obj, args);
addtoCount('canvas.toDataURL') + 1;
}
```

'window.atob' and 'window.btoa' are the most straightforward means of embedding payload content into an HTML file.

```
old_atob = window.atob;
window.atob = function () {
```

```
console.log("window.atob is being monitored");

const obj = this;

const args = arguments;

old_atob.apply(obj, args);

addtoCount("atob") + 1;

};
```

eval () function can be exploited by attackers to run malicious scripts if user input is not properly sanitized.

```
old_eval = window.eval;

window.eval = function () {

console.log(" window.eval is being monitored");

const obj = this;

const args = arguments;

old_eval.apply(obj, args);

addtoCount('window.eval') + 1;

}
```

These are some of the APIs amongst many that are being monitored by the browser extension.

Table 3.1: Browser APIs and Their Potential Misuse by Malicious Actors

API Name	**Potential Misuse**
charCodeAt	Used to encode data or create obfuscated code that isn't easily readable, aiding in evasion techniques.
window.Uint8Array	To handle raw binary data which can be used in memory exploits, such as buffer overflows, or to process downloaded malicious payloads.
Math.random	Domain generation algorithms (DGAs) use Math.random to generate a large number of domain names that malware can use to communicate with without being easily blocked.
document.cookie	Accessed and manipulated to perform session hijacking.
prototype.toDataURL	HTMLCanvasElement.prototype.toDataURL can be used to capture the content of a canvas element, which can be used for browser fingerprinting or stealing information rendered on the canvas.
writeln/write	Exploited to inject malicious content into a webpage, such as through XSS attacks.
window.atob/btoa	Used to encode and decode base64 strings, which is commonly used in obfuscating payloads and command and control communications.
createElement	Used to dynamically create elements like `script` or `iframe` to load malicious code.
Window.location	Used to redirect users to malicious sites or manipulate page content for phishing or site spoofing.
fromCharCode	Likely used in obfuscation techniques to hide malicious code.
window.eval	Executes text as code, allowing for arbitrary JavaScript code execution, often used in XSS attacks.
Image.src	Modifying the `src` attribute can be used for CSRF attacks or to exfiltrate data.

CHAPTER IV

EVALUATION

The proposed framework presents a technique to classify malicious and benign JavaScript content. The primary focus revolves around monitoring, tracking, and analyzing website API calls. The approach aims to provide a comprehensive understanding of the nature of these API calls. Which in turn helps distinguish between benign and malicious activities. The predictive mechanism helps with the overall working and effectiveness of the framework. This, in turn, helps users with the insights needed to understand and avoid potential threats online. This framework aims to ensure that users can safely navigate the web, prepared with the knowledge of which sites are potentially making harmful API calls.

4.1 Implementation and Mechanism

The Proposed framework's design and development is based on the classification of JavaScript content, Differentiating between benign and malicious behaviors. This differentiation is achieved through a process of monitoring, tracking, and analyzing browser API calls.

Data Collection: The dataset that is obtained by capturing the Browser API calls using a browser automation tool is the core of this framework. Each entry in the dataset carries a label, marking the API call's name and the number of times it was called. This labeled data forms the foundation for different patterns associated with different types of activity of the API.

Browser Automation and Real-time Monitoring: The browser automation tool facilitates the continuous capture of API calls. This tool works alongside a dedicated browser extension, ensuring that API calls are monitored in real-time. It also plays a role in organizing this data creating CSV files for structured data storage.

Machine Learning Integration: After the data has been collected, it is transformed and used as an input to train the various machine learning models. These models are fine-tuned to recognize patterns and classify the nature of the API calls. The performance of these models is documented, offering insights into their accuracy and reliability in the classification task.

Contrast with Standard Browser Features: While standard browser settings offer some control over online interactions, they are often limited in their scope. Features are specific to browsers and don't offer a holistic solution to classify and understand the nature of JavaScript content. This framework addresses this limitation by providing a detailed, real-time analysis of web content.

4.1.1 Automation and Data Collection

The browser extension is a crucial component of the architecture that is being used in this experiment. While the browser extension helps with the real-time monitoring of websites, consistency in data collection is also important. Information from a single website is not enough to categorize if a code is malicious or non-malicious. For this purpose, in our experiment, we utilized two major datasets. Malicious URLs from URLhaus and benign websites from Tranco. URLhaus [43], a part of the Abuse.ch project, is widely recognized for its extensive database of malicious URLs, used by entities like the FBI, proving its reliability. Tranco [44], according to its research paper, offers a robust ranking of websites, combining multiple data sources to ensure stability and resistance to manipulation, making it an ideal source for benign websites. The size of these datasets ensures a comprehensive collection of websites, which is vital for our machine learning models. Visiting these websites manually is time-consuming and is also not the right approach. For this purpose, we use Puppeteer. Puppeteer is a Node library that provides a high-level API to control headless Chrome over the DevTools Protocol, it is useful for automating the Chrome browser. The first step towards automating the process is to set up the environment. We use the '**npm i puppeteer**' command to install the latest version of Puppeteer. Once the installation is complete, we install the necessary modules required for the experiment.

```
const fs = require('fs');
const path = require('path');
const puppeteer = require('puppeteer');
```

‘**fs**’ allows file system operations, ‘**path**’ deals with file and directory paths, and ‘**puppeteer**’ provides browser automation capabilities. The next step is to write a function that handles the browser automation and the data collection process. In the below code for each website in the list, a new browser instance is launched using Puppeteer. The browser is set to load the browser extension. In this way, we maintain a consistent monitoring of the browser APIs.

```
browser = await puppeteer.launch({
    headless: false,
    devtools: false,
    args: [
        ‘--disable-extensions-except=${pathToExtension}‘,
        ‘--load-extension=${pathToExtension}‘
    ]
});
```

The browser then opens a new tab and navigates to the current website. It waits for data to be available for extraction. After the page loads, the code checks for captured data. If data is found, it’s categorized by API and added to the counter. After processing all websites, the collected data is formatted into CSV structure and written to a file for further analysis. In this experimentation, there are two instances of Puppeteer that are being utilized. One for the list of malicious websites (urlhaus.abuse(dot)ch) and the other for non-malicious websites (trancolist(dot)eu).

4.1.2 Content Classification

The two CSV files that are generated by the automation script are concatenated using Python script after they have been flagged as malicious and benign for classification purposes. A total of 59 browser APIs have been selected for this dataset, and these 59 browser APIs form the features for the machine learning model. Benign APIs are selected based on the most used browser APIs by any website to maintain the dynamic nature of the website and keep it functioning. The malicious ones are selected based on their potential to be misused in a web-based attacks, such as executing unauthorized code or scripts (e.g. ‘’eval”), “window.open” for unwanted pop-up ads and “navigator.sendBeacon” can be used for unauthorized tracking. Since the framework must label whether the content is malicious or benign, we use the classification models.

The models being used in this framework are SVM (Support Vector Machine), Logistic Regression, GaussianNB (Gaussian Naive Bayes), KNN (K-Nearest Neighbors), Decision Tree, Random Forest, Ensemble Model, XGBoost, and Ensemble. Let’s understand each one of them with an example.

SVM (Support Vector Machine): SVM is a supervised learning algorithm used for classification or regression. It works by finding the hyperplane that best divides a dataset into classes. The ”support vectors” are the data points that lie closest to the hyperplane and influence its orientation and position. Kernel tricks can be used with SVM to handle non-linear data boundaries. E.g. Imagine you have red and blue marbles on a table, and you want to separate them using a straight ruler. SVM helps you place the ruler in the best way to separate the two colors.

Logistic Regression: Logistic Regression is used for binary classification. It models the probability that a given instance belongs to a particular category. It uses the logistic function to squeeze the output of a linear equation between 0 and 1, providing a probabilistic result. E.g. Think of a seesaw. If you have more weight on one side, it goes down. Logistic Regression is like predicting which side of the seesaw will go down based on the weight (or data) you have.

GaussianNB (Gaussian Naive Bayes): This is a probabilistic classifier based on Bayes' theorem with the assumption of independence among predictors. The Gaussian Naive Bayes version assumes that continuous features follow a normal distribution. E.g. It's like guessing if it'll rain based on past weather. If it often rained when there were dark clouds before, then the next time you see dark clouds, you'll guess it might rain.

KNN (K-Nearest Neighbors): KNN is a lazy, instance-based learning algorithm. For classification, it finds the 'k' training examples closest to a point and returns the most common output value among them. The distance metric (often Euclidean) and 'k' value are critical parameters. E.g. You want to know if you'll like a game. You'd ask a few close friends. If most of them like it, you might too! KNN works similarly; it asks its 'nearest neighbors' to make a decision.

Decision Tree: Decision Trees split the data into subsets based on the feature values. They are hierarchical structures where each node is a test on an attribute, each branch represents an outcome of the test, and leaf nodes represent classes or class distributions. They can be prone to overfitting, especially when the tree is deep. E.g. It's like a game of 10 questions. You ask questions to narrow down options and make a decision. "Is it bigger than a breadbox?" "Can it fly?" Each answer leads you closer to the right guess.

Random Forest: An ensemble method that creates a 'forest' of decision trees. Each tree is trained on a random subset of the data and makes its own predictions. Random Forest then aggregates these predictions to produce a final result. It's robust against overfitting and often provides higher accuracy than individual trees. E.g. Imagine you ask many of your friends to play the 10 questions game and then combine all their answers. That's what Random Forest does; it combines many decision trees to make a better guess.

XGBoost: An optimized gradient boosting library. Gradient boosting is an ensemble technique where new models are added to correct the errors made by existing models. XGBoost is particularly efficient and has been used to win many Kaggle competitions due to its speed and performance. E.g. Think of it as a smart student who learns from mistakes. Every time it gets a math problem wrong, it studies that problem type more. Over time, it gets really good at solving those problems.

Ensemble: In the context of machine learning, an ensemble is a combination of diverse models to improve the final predictive performance. By leveraging the strengths of each individual model and combining their outputs, ensemble methods can achieve better accuracy and reduce the likelihood of overfitting. E.g. It's like asking a group of friends for movie suggestions. Instead of just going with one friend's idea, you pick the movie that most friends suggest. The ensemble combines many models' answers to make a final decision.

4.1.3 Machine Learning Implementation

A Google Collab environment is used to train the different machine-learning models using the browser API data that was collected using the browser extension and automated browser. The first step for any model training is to import the required libraries to train the model. Once the required libraries are imported, we load the data that will be used to train the machine-learning model. The below table helps to understand the 'under the hood' of the machine learning models used for this framework. Let's go through the Table [4.1] and [4.2] in detail to understand each column and its significance.

Table 4.1: Differentiation of Machine Learning Models

Model Name	**Hyper parameter Tuning**	**Pipeline Definition**	**Fine Tuning**	**Steps Involved**
SVM	Yes (GridSearchCV) C: 10000, gamma: 1	SMOTE, Standard-Scaler, SVC	Yes	Data Loading, Data Splitting, Pipeline Creation, Hyperparameter Tuning, Model Training, Prediction
Logistic Regression	Yes (GridSearchCV) C: [0.001, 0.01, 0.1, 1, 10, 100, 1000]	SMOTE, Standard-Scaler	Yes	Data Loading, Data Splitting, Pipeline Creation, Hyperparameter Tuning, Model Training, Prediction
GaussianNB	Yes (GridSearchCV) var-smoothing: [1e-9, 1e-8, 1e-7]	SMOTE, Standard-Scaler	No	Data Loading, Data Splitting, Pipeline Creation, Model Training, Prediction
KNN	Yes (GridSearchCV) n-neighbors: [3, 5, 7]	SMOTE, Standard-Scaler	Yes	Data Loading, Data Splitting, Pipeline Creation, Hyperparameter Tuning, Model Training, Prediction

Model Name:

This column simply lists the names of the machine-learning models that are being used.

Hyperparameter Tuning:

Yes (GridSearchCV): This means that the model's hyperparameters were optimized using the GridSearchCV method. GridSearchCV performs an exhaustive search over a specified parameter grid. It trains the model on each combination of hyperparameters and selects the best combination based on cross-validation performance.

No: This means that no hyperparameter tuning was performed for the model.

Pipeline Definition:

SMOTE: Synthetic Minority Over-sampling Technique. A technique used to handle class imbalance. It works by creating synthetic samples in the feature space.

Standard Scaler: A preprocessing step that standardizes the features by removing the mean and scaling to unit variance.

SVC: Support Vector Classifier. It's the classification adaptation of the Support Vector Machine algorithm.

Voting Classifier: A classifier that fits base classifiers each on the whole dataset. It then averages the individual predictions to form a final prediction.

Table 4.2: Differentiation of Machine Learning Models

Model Name	**Hyper parameter Tuning**	**Pipeline Definition**	**Fine Tuning**	**Steps Involved**
Decision Tree	No	No	No	Data Loading, Data Preprocessing, Data Splitting, Model Training, Model Prediction, Model Evaluation
Random Forest	Yes (GridSearchCV) n-estimators: 300, min-samples-split: 2, max-depth: None	SMOTE	Yes	Data Loading, Data Splitting, Pipeline Creation, Hyperparameter Tuning, Model Training, Prediction
XGBoost	Yes (GridSearchCV) n-estimators: 200, max-depth: 15, learning-rate: 0.1, subsample: 1.0, colsample-bytree: 0.6	SMOTE, Standard-Scaler	Yes	Data Loading, Data Splitting, Pipeline Creation, Hyperparameter Tuning, Model Training, Prediction
Ensemble	Yes (GridSearchCV for individual models) same as for Random Forest and XGBoost	Voting Classifier (Random Forest and XGBoost)	No	Data Loading, Data Preprocessing, Data Splitting, Model Definition, Hyperparameter Tuning, Model Training, Model Evaluation, Ensemble Model Definition, Ensemble Model Training, Ensemble Model Evaluation

Fine Tuning:

Yes: Indicates that after the initial model training, further adjustments or optimizations were made to improve the model's performance.

No: Indicates that no additional adjustments were made after the initial model training.

Model Training:

Data Loading: The initial step where data is loaded into the environment from a source.

Data Preprocessing: Preparing the data for training. It can include encoding categorical variables, handling missing values, etc.

Data Splitting: The dataset is divided into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate its performance.

Pipeline Creation: A sequence of data processing steps, which can include oversampling, scaling, and the actual model training.

Hyperparameter Tuning: The process of optimizing the model's hyperparameters to improve its performance.

Model Training: The model is trained on the training dataset using the specified algorithm.

Model Evaluation: After training, the model's performance is evaluated on the testing dataset to determine its accuracy and other metrics.

Ensemble Model Definition: For the ensemble model, this step involves defining which models to include in the ensemble and how their predictions should be combined.

Ensemble Model Training: The ensemble model is trained on the training dataset.

Ensemble Model Evaluation: The performance of the ensemble model is evaluated on the testing dataset.

4.2 Results

Table 4.3: Results of Models for Classifying JavaScript Content

Model	**Benign**			**Malicious**			**Accuracy**
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Score**
SVM	0.97	0.64	0.77	0.73	0.98	0.84	0.81
Logistic Regression	0.985	0.57	0.72	0.70	0.99	0.82	0.78
Naïve Bayes	1.00	0.53	0.69	0.68	1.00	0.81	0.77
KNN	0.94	0.64	0.77	0.73	0.96	0.83	0.80
Decision Tree	0.97	0.68	0.80	0.76	0.98	0.85	0.83
Random Forest	0.88	0.88	0.88	0.88	0.88	0.88	0.88
XGBoost	0.92	0.78	0.84	0.81	0.93	0.87	0.86
Ensemble	0.86	0.90	0.88	0.89	0.85	0.87	0.87

Table [4.3] presents the results of the models used in the framework for classifying JavaScript content into Benign and Malicious. Each row represents the results for one model, and the columns correspond to the performance metrics precision, recall, F1-score, and accuracy for both benign and malicious content.

Precision: Indicates the proportion of positive identifications that were correct. A model with high precision will correctly identify the majority of malicious (or benign) content but might miss out on some.

Recall: Indicates the proportion of actual positives that were correctly identified. A model with high recall will identify most of the malicious (or benign) content but might also misclassify some benign content as malicious.

F1-Score: This is the harmonic mean of precision and recall and provides a balance between the two. If one of them is particularly low, the F1-score will reflect that.

4.3 Accuracy and Recall Comparison

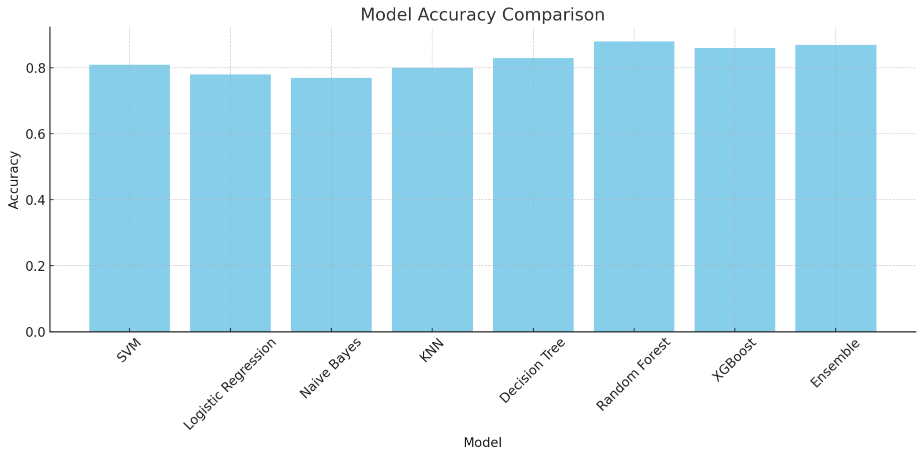

Figure 4.1: Model Accuracy Comparison

1. **Model Accuracy Comparison**:

   - The first graph illustrates the accuracy of each model.
   - The Random Forest and Ensemble models have the highest accuracy, closely followed by XGBoost and Decision Tree.
   - Logistic Regression, Naive Bayes, and SVM have comparatively lower accuracy scores.

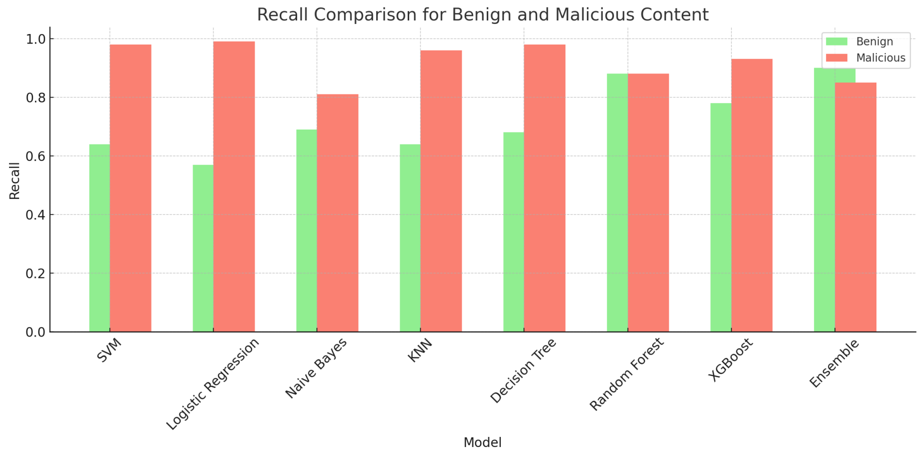

Figure 4.2: Recall comparison

2. **Recall Comparison for Benign and Malicious Content**:

   - The second graph compares the recall values for benign and malicious content for each model.
   - For benign content, the Ensemble model has the highest recall, followed by Random Forest and XGBoost.
   - For malicious content, Logistic Regression has the highest recall, followed closely by SVM, Decision Tree, and KNN.

4.4 Discussion

The **SVM** model achieved a precision of 0.97 for benign content and 0.73 for malicious content. The recall values were 0.64 for benign and 0.98 for malicious content. The model was highly precise in predicting benign content, but it was more comprehensive in identifying malicious instances.

The **logistic Regression** model achieved a precision of 0.985 for benign content. However, the recall for benign content was slightly lower at 0.57, Suggesting that while the model was very precise, it might have missed a significant portion of actual benign instances.

The **accuracy** column provides an overall measure of a model's performance. High accuracy indicates that the model is effective in distinguishing between benign and malicious JavaScript content. As observed from the table, the Random Forest model achieved the highest accuracy of 0.88, closely followed by the Ensemble model at 0.87.

CHAPTER V

CONCLUSION

5.0.1 Summary

To protect users from the ever-growing online threats, this thesis proposes a framework that emphasizes a mechanism designed to ensure a safe browsing experience. This mechanism can effectively identify harmful JavaScript content by capturing the obscure behavioral patterns of JavaScript when it interacts with Browser APIs. Methods such as lexical analysis, content examination, and blacklists were previously used to identify harmful content. However, these methods have shown limitations, especially when faced with sophisticated online threats that evolve rapidly.

To address these challenges, this research delves into a comprehensive analysis of JavaScript content behavior within web browsers. It identifies patterns that may indicate malicious intent and emphasizes the importance of understanding the behavior of interactive web elements, which are increasingly powered by JavaScript. The importance is on a dynamic and context-oriented examination of JavaScript content, offering an alternative to traditional detection approaches.

The framework utilizes browser automation techniques to closely monitor and analyze the execution of JavaScript within web environments. Such interactions can reveal intent, patterns, and sequences that might not be evident at a surface-level examination.

The dynamic nature of web content requires continuous monitoring and analysis. Static approaches, which analyze content only once, can miss out on threats that are only visible or evolve during a session's duration. This research's approach aims to fill this gap by actively monitoring the JavaScript behaviors throughout the browsing session.

5.0.2 Limitation & Future Work

Limitation:

Framework Scalability: As the volume of web pages increases, ensuring efficient operation of the framework could be challenging, given the broad nature of JavaScript behavior.

Dynamic Web Content: The complex and dynamic behavioral patterns of JavaScript as it interacts with Browser APIs might affect detection accuracy.

User Interface Complexity: The framework might require advanced user interfaces for feedback, which could impact end-user interaction and clarity.

Maintenance and Updates: Frequent updates might be needed to keep up with evolving online threats and web technologies.

Future Work:

Enhanced Analysis Algorithms: Further refinement of algorithms to better distinguish between benign and malicious JavaScript behaviors.

Framework Adaptability: Ensure the framework's adaptability to evolving web technologies and scripting standards.

Feedback Integration: Enhance the framework's effectiveness by incorporating a mechanism that allows it to adapt and improve based on direct feedback from users and data from real-world threat detection scenarios.

Broader Analysis: Enhance the framework's capabilities by extending its monitoring and detection to include a variety of other potential cybersecurity threats, in addition to those found in JavaScript.

Interplay with Other Detection Mechanisms: Research ways to combine the framework with various cybersecurity systems to develop a more comprehensive strategy for web threat protection.

BIBLIOGRAPHY

[1] E. Andreasen, L. Gong, A. Møller, M. Pradel, M. Selakovic, K. Sen, and C.-A. Staicu, “A survey of dynamic analysis and test generation for javascript,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 5, pp. 1–36, 2017.

[2] R. Wang, Y. Zhu, J. Tan, and B. Zhou, “Detection of malicious web pages based on hybrid analysis,” *Journal of Information Security and Applications*, vol. 35, pp. 68–74, 2017.

[3] S. Ndichu, S. Kim, and S. Ozawa, “Deobfuscation, unpacking, and decoding of obfuscated malicious javascript for machine learning models detection performance improvement,” *CAAI Transactions on Intelligence Technology*, vol. 5, no. 3, pp. 184–192, 2020.

[4] Y. Fang, C. Huang, L. Liu, and M. Xue, “Research on malicious javascript detection technology based on lstm,” *IEEE Access*, vol. 6, pp. 59 118–59 125, 2018.

[5] L. Invernizzi, P. M. Comparetti, S. Benvenuti, C. Kruegel, M. Cova, and G. Vigna, “Evilseed: A guided approach to finding malicious web pages,” in *2012 IEEE symposium on Security and Privacy*. IEEE, 2012, pp. 428–442.

[6] V. K. Malviya, S. Rai, and A. Gupta, “Development of web browser prototype with embedded classification capability for mitigating cross-site scripting attacks,” *Applied Soft Computing*, vol. 102, p. 106873, 2021.

[7] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, “Knowing your enemy: understanding and detecting malicious web advertising,” in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 674–686.

[8] D. R. Patil, J. Patil *et al.*, “Survey on malicious web pages detection techniques,” *International Journal of u-and e-Service, Science and Technology*, vol. 8, no. 5, pp. 195–206, 2015.

[9] Y. Fang, C. Huang, Y. Su, and Y. Qiu, “Detecting malicious javascript code based on semantic analysis,” *Computers & Security*, vol. 93, p. 101764, 2020.

[10] P. N. Hiremath, “A novel approach for analyzing and classifying malicious web pages,” Ph.D. dissertation, University of Dayton, 2021.

[11] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu, “A survey on machine learning techniques for cyber security in the last decade,” *IEEE access*, vol. 8, pp. 222 310–222 354, 2020.

[12] W. Xu, F. Zhang, and S. Zhu, “Jstill: mostly static detection of obfuscated malicious javascript code,” in *Proceedings of the third ACM conference on Data and application security and privacy*, 2013, pp. 117–128.

[13] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: a fast filter for the large-scale detection of malicious web pages," in *Proceedings of the 20th international conference on World wide web*, 2011, pp. 197–206.

[14] D. Genkin, L. Pachmanov, E. Tromer, and Y. Yarom, "Drive-by key-extraction cache attacks from portable code," in *Applied Cryptography and Network Security: 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings 16*. Springer, 2018, pp. 83–102.

[15] K. Rieck, T. Krueger, and A. Dewald, "Cujo: efficient detection and prevention of drive-by-download attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference*, 2010, pp. 31–39.

[16] B.-I. Kim, C.-T. Im, and H.-C. Jung, "Suspicious malicious web site detection with strength analysis of a javascript obfuscation," *International Journal of Advanced Science and Technology*, vol. 26, pp. 19–32, 2011.

[17] P. Seshagiri, A. Vazhayil, and P. Sriram, "Ama: static code analysis of web page for the detection of malicious scripts," *Procedia Computer Science*, vol. 93, pp. 768–773, 2016.

[18] P. Laskov and N. Šrndić, "Static detection of malicious javascript-bearing pdf documents," in *Proceedings of the 27th annual computer security applications conference*, 2011, pp. 373–382.

[19] B. Eshete, A. Villafiorita, and K. Weldemariam, "Malicious website detection: Effectiveness and efficiency issues," in *2011 First SysSec Workshop*. IEEE, 2011, pp. 123–126.

[20] S. Gupta and B. B. Gupta, "Xss-safe: a server-side approach to detect and mitigate cross-site scripting (xss) attacks in javascript code," *Arabian Journal for Science and Engineering*, vol. 41, pp. 897–920, 2016.

[21] P. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.

[22] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–28, 2019.

[23] M. Y. Wong and D. Lie, "Intellidroid: a targeted input generator for the dynamic analysis of android malware." in *NDSS*, vol. 16, no. 2016, 2016, pp. 21–24.

[24] H. Darabian, S. Homayounoot, A. Dehghantanha, S. Hashemi, H. Karimipour, R. M. Parizi, and K.-K. R. Choo, "Detecting cryptomining malware: a deep learning approach for static and dynamic analysis," *Journal of Grid Computing*, vol. 18, pp. 293–303, 2020.

[25] J. Jeon, J. H. Park, and Y.-S. Jeong, "Dynamic analysis for iot malware detection with convolution neural network model," *IEEE Access*, vol. 8, pp. 96 899–96 911, 2020.

[26] T. Bhatia and R. Kaushal, "Malware detection in android based on dynamic analysis," in *2017 International conference on cyber security and protection of digital services (Cyber security)*. IEEE, 2017, pp. 1–6.

[27] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—a state of the art survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–48, 2019.

[28] D. Sahoo, C. Liu, and S. C. Hoi, "Malicious url detection using machine learning: A survey," *arXiv preprint arXiv:1701.07179*, 2017.

[29] Y.-T. Hou, Y. Chang, T. Chen, C.-S. Laih, and C.-M. Chen, "Malicious web content detection by machine learning," *expert systems with applications*, vol. 37, no. 1, pp. 55–60, 2010.

[30] P. Laskov and R. Lippmann, "Machine learning in adversarial environments," pp. 115–119, 2010.

[31] K. Lee, J. Caverlee, and S. Webb, "Uncovering social spammers: social honeypots+ machine learning," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, 2010, pp. 435–442.

[32] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of computer security*, vol. 19, no. 4, pp. 639–668, 2011.

[33] S. Ndichu, S. Ozawa, T. Misu, and K. Okada, "A machine learning approach to malicious javascript detection using fixed length vector representation," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[34] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[35] E. G. Dada, J. S. Bassi, H. Chiroma, A. O. Adetunmbi, O. E. Ajibuwa *et al.*, "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, 2019.

[36] S. Ndichu, S. Kim, S. Ozawa, T. Misu, and K. Makishima, "A machine learning approach to detection of javascript-based attacks using ast features and paragraph vectors," *Applied Soft Computing*, vol. 84, p. 105721, 2019.

[37] A. Cohen, N. Nissim, L. Rokach, and Y. Elovici, "Sfem: Structural feature extraction methodology for the detection of malicious office documents using machine learning methods," *Expert Systems with Applications*, vol. 63, pp. 324–343, 2016.

[38] W. Han, J. Xue, Y. Wang, L. Huang, Z. Kong, and L. Mao, "Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics," *computers & security*, vol. 83, pp. 208–233, 2019.

[39] Y. Qiao, Y. Yang, J. He, C. Tang, and Z. Liu, "Cbm: free, automatic malware analysis framework using api call sequences," in *Knowledge Engineering and Management: Proceedings of the Seventh International Conference on Intelligent Systems and Knowledge Engineering, Beijing, China, Dec 2012 (ISKE 2012)*. Springer, 2014, pp. 225–236.

[40] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "Malinsight: A systematic profiling based malware detection framework," *Journal of Network and Computer Applications*, vol. 125, pp. 236–250, 2019.

[41] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 651–661.

[42] G. Richards, S. Lebresne, B. Burg, and J. Vitek, "An analysis of the dynamic behavior of javascript programs," in *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2010, pp. 1–12.

[43] (2021) Bern university of applied sciences. [Online]. Available: https://www.bfh.ch/en/news/press-releases/2021/abusech-erhaelt-neue-heimat-an-der-bfh/

[44] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," *arXiv preprint arXiv:1806.01156*, 2018.