Machine Learning-Based Detection of Cross-Site Scripting Attacks

UNIVERSITÀ DEGLI STUDI
DI GENOVA

Roberto Gnisci

Department of Computer Science, Bioengineering, Robotics and System Engineering (DIBRIS)

University of Genova

Supervisor

Luca Oneto

In partial fulfillment of the requirements for the degree of

Master of Science in Computer Engineering - Artificial Intelligence and Human-Centered Computing

October 25, 2023

Contents

List of Figures

Abstract

The problem addressed by this thesis is the detection and mitigation of Cross-Site Scripting (XSS) attacks in web applications. The proposed approach leverages unsupervised learning techniques, employing a One-Class Support Vector Machine (SVM) model to decipher patterns within unlabeled data. This approach is founded on a comprehensive set of steps, including efficient pre-processing techniques, critical feature extraction, and model selection.

A pivotal innovation lies in the introduction of dual One-Class SVM models, specializing in the detection of XSS-related HTML and

JavaScript content. The ability to select between these models adds a layer of adaptability, enabling tailored defense mechanisms based on threat scenarios.

To address ambiguity in model predictions, we propose two evaluation strategies, providing flexibility in security responses. An advanced pre-processing pipeline, encompassing enhanced generalization and data refinement techniques, significantly elevates data quality, enhancing the classifier's efficacy.

The outcomes of this research enrich the possible solutions useful for combating XSS attacks, enhancing traditional rule-based software solutions with the power of machine learning techniques.

Acknowledgment

Acknowledgements

Chapter 1

Introduction

1.1 General context

In today's interconnected and digitized world, the importance of cybersecurity cannot be underestimated. As we delve into the heart of this master's thesis, it is imperative that we first establish a comprehensive understanding of the general context in which cybersecurity operates.

Cybersecurity, is the practice of safeguarding the digital systems, networks, and data from a myriad of threats, ranging from cyberattacks by individuals and organized groups to the ever-changing landscape of technological vulnerabilities. The interconnectedness of our modern society, driven by the proliferation of the Internet and digital technologies, has given us unprecedented opportunities for communication, innovation and efficiency. However, it has also exposed us to a new frontier of threats and risks. One need only glance at the headlines to recognize the prevalence and severity of cyber-attacks. From state-sponsored espionage to ransomware attacks against critical infrastructure, the stakes are higher than ever. These attacks can lead to financial loss, the compromise of sensitive personal and business data, and even jeopardize national security.

The consequences of failing to respond to these threats are profound, affecting not just individuals and organizations, but entire societies. Additionally, as we continue to embrace emerging technologies such as the Internet of Things (IoT), artificial intelligence, and blockchain, the attack surface for cyber threats expands exponentially. The proliferation of devices and data in this increasingly interconnected world amplifies the importance of cybersecurity in safeguarding the digital ecosystems.

However, the cybersecurity landscape is not solely defined by external threats. Insider threats, human error, and ethical data privacy considerations further complicate this arena. The balance between security and individual freedoms, particularly in the age of surveillance and data collection, is a critical aspect of the cybersecurity discourse.

In this scenario, one of the most common and harmful threats in the cybersecurity landscape are Cross-Site Scripting (XSS) attacks. According to the annual security report by "OWASP", the organization for web application security, *"27% of all reported vulnerabilities in 2022 were related to XSS attacks"*.

Cross-Site Scripting is a type of security vulnerability that plagues web applications. It occurs when a website or web application unwittingly includes untrusted data in its content, which is then executed by a victim's web browser. This untrusted code, often written in JavaScript, can lead to a range of malicious activities, from data theft and session hijacking to defacing web pages.

XSS attacks come in several forms:

1. *Stored XSS:* In this scenario, the malicious script is permanently stored on the target server, usually within a database. When a user accesses the compromised page, the script is served and executed.

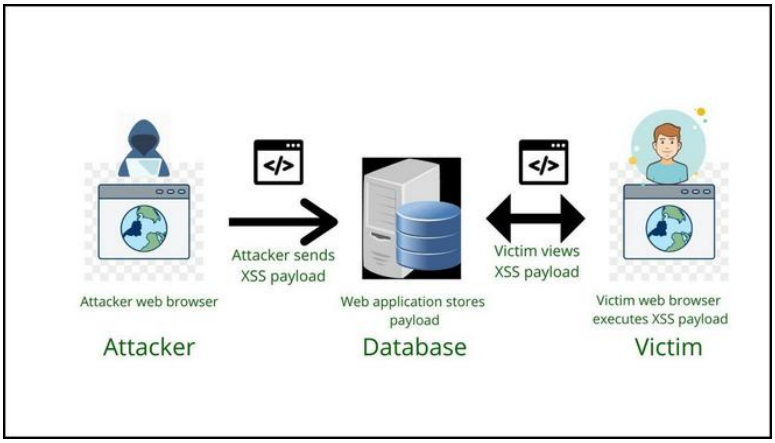

Figure 1.1: XSS Stored schema.

2. *Reflected XSS:* Contrary to stored XSS, reflected XSS doesn't involve storing the malicious script on the target server. Instead, it relies on tricking a user into clicking a malicious link that contains the script. The server then reflects this script back to the user, executing it in their browser.

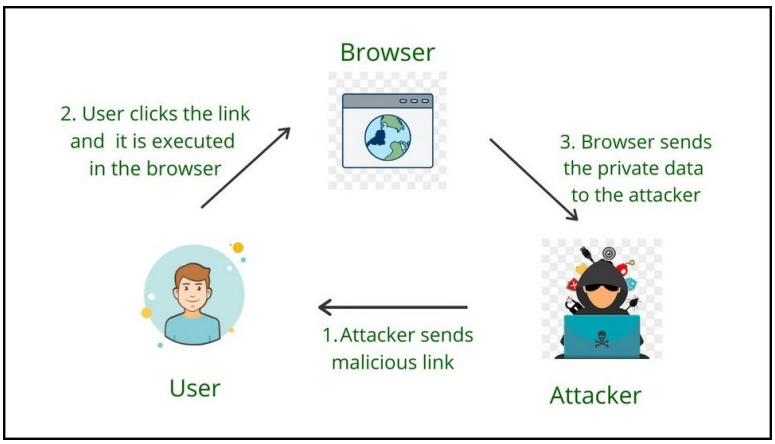

Figure 1.2: XSS Reflected schema.

3. *DOM-based XSS:* This variant occurs when the client-side script manipulates the Document Object Model (DOM), leading to security vulnerabilities.

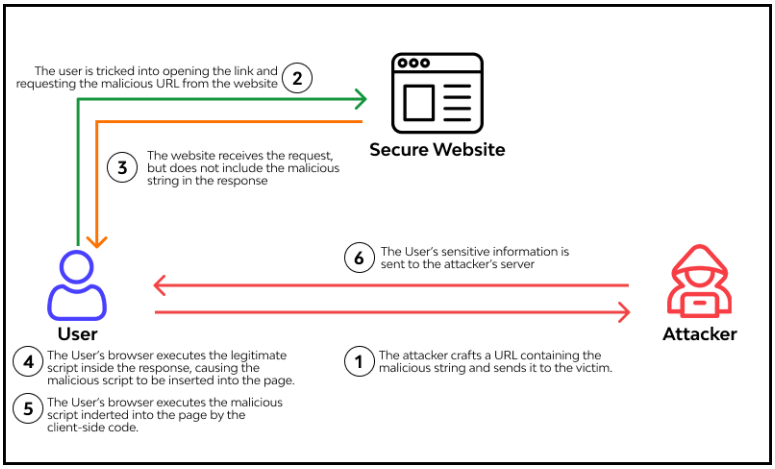

Figure 1.3: XSS DOM-based schema.

The consequences of XSS attacks are all too real. They have been responsible for numerous high-profile data breaches, compromising user data and causing significant damage to organizations' reputations.

1.1.1 Open problems

While significant progress has been made in understanding and mitigating XSS attacks, several challenges and open problems persist in this field of cybersecurity. Recognizing these unresolved issues is crucial for researchers, developers, and security professionals to further enhance the defenses against this threat, here are some of the open problems and challenges associated with detecting XSS attacks:

1. *Detection and Prevention of DOM-based XSS:* Many XSS attacks are executed in the client's web browser, making them challenging to detect and prevent at the server level. Browser vendors have implemented various security features like Content Security Policy (CSP), but the effectiveness of these mechanisms in real-world scenarios and their ease of implementation are areas that require further exploration.

2. *Content Security Policy (CSP) bypasses:* CSP is a security feature that helps prevent XSS attacks by specifying which sources of content are allowed to be loaded and executed. However, attackers are continually finding new ways to bypass CSP restrictions, and finding robust CSP policies that balance security and functionality is an ongoing challenge.

3. *Zero-day vulnerabilities:* They are previously unknown and unpatched security flaws, can be exploited by attackers before developers have a chance to release fixes. Detecting and mitigating zero-day XSS vulnerabilities in real-time remains a pressing concern for the security community.

4. *Context-aware XSS detection:* Many XSS detection tools and mechanisms rely on pattern matching or blacklist approaches. These methods can lead to false positives and negatives. Developing context-aware XSS detection mechanisms that can understand the context of the input and output in a web application is a challenging problem.

5. *Third-Party Dependencies and Supply Chain Attacks:* Modern web applications often rely on third-party libraries and dependencies. Vulnerabilities in these components can introduce XSS risks. Managing and securing the entire supply chain of web applications presents challenges, especially in large-scale development environments.

1.2 Specific context

In this case study, HTTP requests originate from a Web Application and API Protection (WAAP) system, this software is called Mithril and is manufactured and powered by AizoOn Technology Consulting.
The company specializes in offering advanced technology and digital solutions to

improve business operations and operational efficiency, providing strategic consulting services, bespoke software development, data analytics, artificial intelligence, automation and other digital solutions for a wide range of industries.
Mithril is a comprehensive Web Application and API Protection (WAAP) service that goes beyond the traditional capabilities of a Web Application Firewall (WAF). It offers cloud-delivered as-a-service deployment of WAF, bot mitigation, DDoS protection, and API security, while also extending its functionality with various modules to enhance web application or website protection and performance. Mithril's WAF is based on the OWASP Core Rule Set (CRS), a set of attack detection rules designed to safeguard web applications from a wide range of common attacks, including SQL Injection, Cross-Site Scripting, Local File Inclusion, and many others. Additionally, Mithril operates as a reverse proxy, inspecting all web traffic to detect and prevent attacks, and provides a user-friendly web console for configuring rules and modules in near real-time. It offers ongoing support, access to new rules and modules, and a dedicated Security Operations Center (SOC) to maximize web application protection.
The goal of this paper is the development of a machine learning model aimed at classifying HTTP requests, effectively differentiating benign requests from potentially malicious Cross-Site Scripting (XSS) attacks. I will demonstrate the ability to contribute to web security by effectively implementing anomaly detection solutions for real-world scenarios.

1.2.1 Open problems

Detecting cross-site scripting (XSS) attacks from HTTP requests using machine learning classifiers is a challenging problem, and there are several open issues and research areas within this domain. Here are some of the open problems and challenges associated with detecting XSS attacks using machine learning:

1. *Imbalanced Datasets:* Datasets for XSS detection are often highly imbalanced, with a vast majority of legitimate requests and only a small fraction of malicious ones. Handling this class imbalance while training machine learning models effectively is a challenge.

2. *Feature Engineering:* Extracting relevant features from HTTP requests that can capture the characteristics of XSS attacks is crucial. Determining which features are most informative and designing effective feature extraction methods is an ongoing problem.

3. *Data Preprocessing:* Cleaning and preprocessing the data to remove noise and irrelevant information is essential. Developing automated techniques for data preprocessing that improve model performance is a research area.

4. *Model Generalization:* Ensuring that the trained model generalizes well to different web applications and environments is essential. Models should not be overly specific to the training data and should work well in real-world scenarios.

5. *False Positives and Negatives:* Reducing false positives (legitimate requests misclassified as attacks) and false negatives (missed detections) is an ongoing challenge. Balancing these two aspects while optimizing model performance is critical.

6. *Real-time Detection:* Developing models that can provide real-time or near-real-time detection of XSS attacks without introducing significant latency into the web application is important.

7. *Incremental Learning:* Web applications and attack techniques evolve over time. Developing techniques for incremental learning that can adapt to new attack patterns and variations is important for long-term security.

8. *Privacy and Ethical Considerations:* As with any machine learning application, there are privacy and ethical concerns in collecting and using data for XSS detection. Research into privacy-preserving techniques and ethical data handling is important.

1.3 State of the Art

In the world of cybersecurity, a slew of innovative solutions has emerged to counteract the persistent threat of Cross-Site Scripting (XSS) attacks. To provide a clearer understanding, we can categorize these solutions into two distinct groups: Rule-Based Software and AI-Powered Software.

Rule-Based Software solutions:

In the realm of Rule-Based Software, Noxes, introduced by Kirda et al. [1], is a standout solution. It acts as a potent client-side defense mechanism, set to transform personal firewalls' effectiveness against XSS attacks. Noxes excels at assessing incoming HTTP web requests and can grant or deny access based on predefined firewall rules. These rules can be created manually, generated dynamically through Firewall Prompts, or automatically established using Snapshot Mode.

Another player in Rule-Based Software is SessionSafe, introduced by Johns [2]. It implements server-side methods like deferred loading, one-time URLs, and sub-domain switching to bolster web application security against XSS session hijacking. Notably, SessionSafe adds a layer of security without requiring source code modifications. For server-side XSS attack detection, consider XSSDS (XSS-Dec) by Johns et al. [3]. It provides an effective passive approach by analyzing

HTTP request-response variations to identify non-persistent XSS attacks. Persistent XSS attacks are handled through a training-based approach, alerting to potential attacks when unrecognized scripts are detected.
In the same category, XSS-Guard by Bisht and Venkatakrishnan [4] identifies scripts intended for HTML web requests and removes unauthorized scripts in the HTTP response web page. It creates a shadow web page to understand the web application's intent, allowing only legitimate scripts.
SWAP (Secure Web Application Proxy), proposed by Wurzinger et al. [5], counters XSS attacks using a reverse proxy approach, encoding static script calls into script IDs. SWAP decodes these IDs when no malicious scripts are found and alerts the client when malicious scripts are detected.
S2XS2 by Shahriar et al. [6] introduces an automated server-side XSS detection approach using "boundary injection" and "policy generation" to detect XSS attacks, particularly in JSP programs. employ a server-side JavaScript code injection technique based on randomly generated comment statements to detect injected JavaScript code.
Innovating XSS attack prevention, Noncespaces by Gundy and Chen [7] uses Instruction Set Randomization (ISR) techniques to randomize (X)HTML tags and attributes, enhancing security by detecting malicious content and preventing tampering with the Document Object Model (DOM) tree.

AI-Powered Software solutions:

The traditional method of XSS detection typically relies on rule-based matching and the extraction of predefined features, but it struggles to identify increasingly complex XSS attack sentences. However, with the rapid advancement of machine learning techniques, researchers have turned to machine learning algorithms to

address network security challenges, particularly in the domain of XSS attack detection. Several notable approaches have surfaced in this context, contributing to significant progress in the field.

Mereani et al. [8] explores the applicability of three machine learning algorithms: Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), and Random Forests, in the quest to identify and mitigate XSS attacks, whether they are known or unknown, by developing classifiers specifically designed for JavaScript code. The research highlights the effectiveness of a unique feature set that combines language syntax and behavioral characteristics, resulting in classifiers that achieve high levels of accuracy and precision when tested against extensive real-world datasets.

Another noteworthy contribution comes from Wang et al. [9], who proposed an XSS attack detection method rooted in Bayesian networks. Their approach involves extracting 17 XSS attack characteristics and incorporating information about malicious IP addresses and domain names. This strategy has demonstrated efficacy, particularly in identifying non-persistent XSS attacks.

The proliferation of Social Networking Services (SNSs) such as Twitter, Instagram, and Facebook has brought billions of users into the digital realm, exposing them to xss attack risk also on this environments, Rathore et al. [10] introduces a machine learning-based approach tailored specifically for the detection of XSS attacks within the context of Social Networking Services (SNSs), this approach leverages three key features: URLs, webpage content, and the distinct characteristics of SNSs. To create a robust dataset, the researchers collect and analyze 1,000 SNS webpages, extracting pertinent features from these pages. Subsequently, ten different machine learning classifiers are applied to this dataset to classify webpages into two categories: XSS or non-XSS.

Mokbal et al. [11] used a robust artificial neural network-based multilayer percep-

tron (MLP) scheme, detection scheme incorporates dynamic feature extraction mechanisms and leverages a substantial real-world dataset, achieving outstanding results, with an AUC-ROC score of 90.02%.

Fang et al. [12] with DeepXSS, presented a deep learning algorithm for the effective detection of Cross-Site Scripting (XSS) attacks. Leveraging word2vec for feature extraction and Long Short Term Memory (LSTM) recurrent neural networks for training and testing, DeepXSS achieves remarkable precision (99.5%) and recall (97.9%) rates in real-world datasets.

A fundamental challenge in the realm of machine learning-based XSS detection is the automatic definition and extraction of features. This issue has prompted significant research efforts to tailor deep learning models to the unique characteristics of the security field. Additionally, while deep learning models have demonstrated their potential, they often incur higher computational costs, especially when employing stacked models.

The integration of machine learning, into the XSS attack detection landscape represents a promising avenue for enhanced security. These approaches offer the potential to adapt to evolving attack vectors and patterns, providing a valuable tool in the ongoing battle against XSS attacks. However, they also bring challenges related to feature definition, computational efficiency, and model training that warrant continued research and innovation.

1.4 Proposed solution

The proposed approach involves an unsupervised classifier designed to understand patterns within unlabeled data. This process encompasses several key steps:

- Select an appropriate pre-processing pipeline to effectively cleanse the payloads by removing extraneous elements.
- Determine the most critical features for detecting XSS attacks.
- Extract features from the payloads.
- Opt for an unsupervised model that aligns well with the problem statement.
- Train the model using the extracted features.
- Evaluate the outcomes and optimize it.

The chosen model is One-Class SVM, renowned for its anomaly detection capabilities, the underlying concept is to instruct the model on how to discern XSS attacks. If the model fails to recognize a payload, it is considered benign.

1.5 Innovative content

This chapter explores the innovative aspects in building an XSS attack classifier. This research goes beyond conventional methods, incorporating several groundbreaking elements that enhance the robustness and adaptability of the system.

1. *Dual One-Class SVM Models:* This approach introduces the use of two distinct One-Class Support Vector Machine (SVM) models. The first model is tailored for detecting XSS-related HTML content, while the second focuses on identifying XSS-related JavaScript content. This dual-model setup allows us to specialize in capturing the unique characteristics of both HTML

and JavaScript payloads, increasing the classifier's accuracy and precision. This specialization is a novel feature of the work, emphasizing the commitment to addressing the diversity of potential XSS attacks.

2. *Model Selection:* One of the key innovations in this approach is the ability to select which of two One-Class SVM models to activate based on the situation. This adaptive approach enhances the versatility of the classifier. When faced with different input payloads, one can choose between the model that specializes in detecting HTML content and the one that specializes in identifying JavaScript content. This model selection allows us to tailor the defense mechanism to the specific threat scenario, maximizing the accuracy and effectiveness of the classifier.

3. *Handling Ambiguity in Model Predictions:* To address situations where the two models produce conflicting results for the same input payload, i propose two different evaluation strategies. The first solution prioritizes the model whose payload-to-hyperplane distance is greater. In the second solution, if one of the two models has a positive classification, than the payload is considered as a malicious one. This approach to handling ambiguity allows you to set the severity of the assessment, using the second model in which case you want a more stringent assessment.

4. *Advanced Pre-processing Pipeline:* One of the innovative components of this work is the development of an advanced pre-processing pipeline. This pipeline is designed to cleanse and prepare input payloads comprehensively. In addition to standard procedures like decoding and tokenization, i introduce an improved concept of generalization. By refining the removal of extraneous white spaces, eliminating redundant characters and unnecessary punctuation, converting text to lowercase, and stripping HTML com-

ments, i ensure that input data is thoroughly sanitized. This enhanced pre-processing pipeline significantly improves the quality of data fed into classification models. The term "generalization" itself may not be entirely novel, as it has been used in related work, but this approach represents a substantial improvement in the way it is applied to preprocess input payloads.

This research introduces several innovative elements, from an enhanced preprocessing pipeline to dynamic model selection, dual SVM models, and an unsupervised learning approach. These innovations collectively contribute to the effectiveness, adaptability, and robustness of the XSS attack classifier, making it a valuable asset in the realm of web security.

Chapter 2

Model development

2.1 Problem

2.1.1 Problem definition

The process begins with the HTTP request body, which is filtered through the Mithril WAAP service and subsequently stored in Elasticsearch, a distributed, real-time, and full-text search and analytics engine. To lay a foundation for a comprehensive understanding of the ensuing analysis, it is important to provide an overview of what constitutes an HTTP request.

The Hypertext Transfer Protocol (HTTP) serves as the cornerstone of communication across the World Wide Web. It establishes a standardized framework comprising rules and conventions for the exchange of data between web clients, typically in the form of web browsers, and web servers. HTTP operates as a request-response protocol, affording clients the ability to solicit resources from servers and receive responses containing the requested resources. HTTP requests represent the client's means of interaction with web servers, enabling the retrieval of web pages, images, documents, or any other resources hosted on remote servers.

An HTTP request comprises several pivotal components, each assigned a specific purpose:

1. *Request Method:* The request method specifies the type of action the client wishes to perform on the resource. Common HTTP methods include:
    - GET: Requests data from the server, typically used for retrieving web pages or resources.
    - POST: Submits data to the server, often used for form submissions or data uploads.
    - PUT: Updates a resource on the server with the provided data.
    - DELETE: Requests the removal of a resource from the server.
2. *Request URI(Uniform Resource Identifier):* The URI identifies the resource the client wants to interact with. It includes the server's domain name or IP address and the path to the resource on the server. For example, in the URI "https://www.example.com/page.html," "https" is the protocol, "www.example.com" is the server's address, and "/page.html" is the path to the resource.
3. *Request Headers:* HTTP headers provide metadata about the request and can include information such as the client's user-agent, accepted content types, and cookies. Headers are crucial for the server to understand how to process the request and provide the appropriate response.
4. *Request Body:* The request body, often used in methods like POST and PUT, contains data that the client wants to send to the server. This can include form data, JSON payloads, or other types of structured or unstructured data.

XSS attacks often exploit the way web applications handle user input in HTTP requests. Malicious payloads are injected into request parameters, such as form fields or query strings, and then processed by the application. These payloads can include JavaScript code that, when executed in the context of a user's browser, can perform actions like stealing user data, hijacking sessions, or performing other malicious actions. The objective is to develop a method capable of identifying the presence of an XSS (Cross-Site Scripting) payload within any type of request body. This entails the training of a model that can accurately discern the distinctive features indicative of such an attack.

2.1.2 Formalization

This chapter outlines the systematic steps taken to achieve the research objectives, which include the creation of an effective XSS attack classifier.

- *Define a Comprehensive Pre-processing Pipeline:* The first crucial step involves defining a robust pre-processing pipeline to cleanse the raw data obtained from potential XSS attack payloads. This pipeline is meticulously designed to remove extraneous and non-essential elements, ensuring that the data is in an optimal format for subsequent analysis.

- *Feature Identification:* The process of feature identification is pivotal. Here, we meticulously determine the features that will be extracted from the cleaned payloads. The selection of these features is a critical aspect of our approach as it directly impacts the classifier's ability to differentiate between benign and malicious content.

- *Feature Extraction:* Once the features have been identified, we proceed to extract them from the pre-processed payloads. This step involves the

transformation of raw data into structured and meaningful feature vectors that can be utilized by our classification algorithm.

- *Effectiveness Assessment:* The effectiveness of the extracted features is rigorously evaluated in this phase. We scrutinize potential correlations, patterns, and statistical properties within the feature set. Understanding the data characteristics is essential for building a robust classifier.

- *Algorithm Selection and Implementation:* After a comprehensive assessment, we carefully select and implement an unsupervised learning algorithm. This algorithm forms the core of our classification system and plays a vital role in detecting potential XSS attacks based on the extracted features.

- *Model Training:* The selected unsupervised learning algorithm is trained on a representative dataset. During this phase, the model learns to identify anomalies and distinguish between normal and potentially malicious content based on the extracted features.

- *Performance Evaluation:* To gauge the effectiveness of our classifier, we conduct a thorough performance evaluation. This evaluation involves various metrics and testing scenarios to assess the classifier's ability to accurately identify XSS attacks while minimizing false positives.

2.1.3 Data

The dataset comprises a collection of strings representing potential XSS (Cross-Site Scripting) attacks identified by Mithril (WAAP). Many of these recorded attacks are found within the HTTP request body, with the body exhibiting distinct characteristics:

1. *Content Type:* The content type, specified in the HTTP request headers using the "Content-Type" field, informs the server about the type of data present in the request body. Common content types include:
    - *application/x-www-form-urlencoded:* Used for submitting form data.
    - *application/json:* Indicates that the body contains JSON data.
    - *multipart/form-data:* Typically used for file uploads.
    - *text/plain:* Denotes plain text data.
2. *Data Format:* The format of the data within the request body depends on the content type. For example:
    - In a *application/x-www-form-urlencoded* request, data is formatted as key-value pairs separated by '&' symbols.
    - In a *application/json* request, data is structured as a JSON object or array.
3. *Size:* The size of the request body indicates the amount of data being sent. This is often measured in bytes and can impact server performance and request processing times.
4. *Encoding:* The request body may be encoded in various ways, such as UTF-8 or binary encoding. The encoding type should match the content type and data format.

5. *Data Payload:* This is the actual data being sent in the request body. It can vary significantly based on the purpose of the request. Common examples include:
    - *Form Data:* Key-value pairs representing form fields and their values.
    - *JSON Payload:* A structured data object or array containing relevant information.
    - *File Upload:* Binary data representing the file being uploaded.

However, the request body is not the sole source of payloads within our dataset. There are additional avenues for injecting XSS attacks:

1. *Request Parameters/Query Strings:* XSS attacks can be injected into the values of query parameters or form fields sent as part of an HTTP request. For example, if a web application allows users to input text into a search box, an attacker might inject a malicious script as part of the search query.

2. *Request Headers:* Although less common, it's possible for XSS attacks to be embedded in request headers, especially if the application processes header values and reflects them in responses without proper sanitization.

3. *Request Cookies:* Cookies sent with an HTTP request can potentially carry XSS payloads if the application doesn't properly sanitize and validate cookie values.

4. *Request URL:* In some cases, the URL itself can be manipulated to include an XSS payload, typically within query parameters. However, this is less common than other injection points.

5. *Response Content:* While not part of the request, XSS attacks become effective when they are reflected in the response content generated by the

server. This occurs when the server echoes user-supplied input back to the user without proper sanitization. The attacker tricks the server into serving the malicious script to other users.

2.1.4 Methodology Selection

Overall schema of the project:

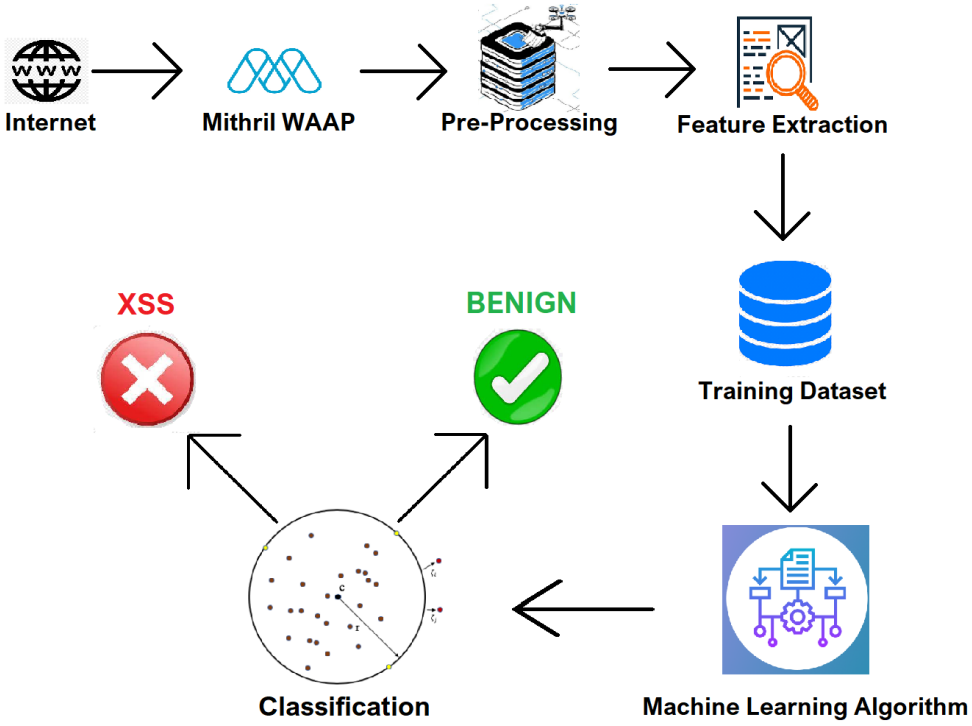

Figure 2.1: Project overall schema.

Figure 2.1 illustrates a crucial aspect of the research, where I leverage Mithril WAAP to capture internet requests. Mithril WAAP operates as a gatekeeper, making autonomous decisions regarding the admission of these requests into the system. It's important to note that Mithril's capabilities extend beyond merely identifying XSS attacks; it possesses the ability to detect various types of cyber

threats. However, for this specific study, I focus on isolating and analyzing XSS attacks from this broader spectrum of attacks, utilizing them to train the model. The XSS payloads collected by Mithril undergo a necessary pre-processing phase. During this phase, I aim to eliminate any extraneous information or noise surrounding the payload, ensuring to retain only the pertinent data for analysis. Once the payload emerges from this pre-processing pipeline, the next step is feature extraction. These features are essential as they provide the model with the necessary information to discern malicious payloads from benign ones. The training dataset exclusively consists of malicious payloads, enabling the model to discern the distinctive patterns associated with malignancy. To accomplish this task, I have chosen the One-Class SVM model.

The selection of the One-Class SVM model represents a strategic decision driven by the unique demands of this research in the realm of cybersecurity. This model choice is not arbitrary; rather, it is a result of careful consideration of the specific challenges and objectives associated with identifying and mitigating XSS attacks within internet traffic. It stands out as an apt choice primarily because of its specialization in anomaly detection. In this context, where the goal is to separate rare malicious payloads from a predominantly benign dataset, this model's ability to establish a decision boundary that encapsulates normal traffic is invaluable. By doing so, it efficiently pinpoints anomalies—malicious payloads that deviate from the established norm. This focused approach aligns seamlessly with the objective of identifying and countering XSS attacks within internet requests.

Upon the completion of the training phase, the model is well-prepared to enter the testing phase. During this phase, the model evaluates new payloads, determining whether they exhibit malicious characteristics or not. This evaluation process is a critical step in enhancing the system's security and safeguarding against potential cyber threats.

2.2 Machine Learning Model

2.2.1 Introduction to Machine Learning

In the last century we have witnessed new discoveries in technology, notably in 1943 Warren McCullock and Walter Pitts realized that the brain "despite being a soft, wet, gelatinous mass, the signaling in it is digital and, to be precise, binary," hence the first studies on the possibility of making machines learn.

Later in 1950 Alan Turing supported the same thesis with an article entitled "Computing Machinery and Intelligence" in which he discussed the possibility of making machines learn. In the 1950s and 1960s, Artificial Intelligence (AI) as a field of study emerged, giving rise to the first attempts to develop programs that could learn. Frank Rosenblatt's "Perceptron" and Arthur Samuel's "Checker-playing program" represented the first examples of machine learning applied to specific problems. Over the following decades, the field of machine learning underwent significant changes.

The 1970s and 1980s saw the development of more sophisticated learning algorithms, such as decision trees and the partial least squares method. But it was mainly in the 1990s and 2000s that machine learning began to flourish. During this period, statistical machine learning algorithms were developed, leading to remarkable advances in practical applications.

Since the 2010s, machine learning has experienced explosive growth. The combination of huge amounts of data and the availability of powerful computational resources has opened up new perspectives. Deep learning algorithms, based on deep neural networks, have revolutionized the machine learning landscape, leading to amazing advances in fields such as speech recognition, image recognition, and natural language processing.

Today, machine learning is ubiquitous. We find it in search engines, product

recommendations, virtual assistants, autonomous vehicles, and countless other aspects of our daily lives. The applications are broad and constantly expanding, helping to solve complex problems and adopting a central role in digital transformation. It has come a long way, from its inception as a visionary concept to becoming a fundamental pillar of modern technology, this dissertation will further explore the field of machine learning, focusing on the integration of this technology in cybersecurity.

2.2.2 One-Class Support Vector Machine

OCSVM is a machine learning technique that aims to identify "anomalous" or "outlier" data in a data set, and is based on the principle of Support Vector Machines (SVMs), one of the most powerful techniques for classification and regression. OCSVM is particularly useful when you have a data set in which one of the classes (the "positive class") is much larger or dominant than the other class (the "negative class" or the "outliers"). This approach finds application in a wide range of scenarios, such as fraud detection, complex systems monitoring, and cybersecurity.

Basic principles of OCSVM:
To understand in detail how OCSVM works, it is useful to start with the basic principles:

1. *Positive class definition:* In OCSVM, we begin by identifying the "positive class," which is the class of interest to be identified. Examples of this class are called "support vectors." This choice is critical because the model will focus on this class and try to identify outliers with respect to it.

2. *Support Vectors:* A crucial part of SVM in general is the discovery of

"support vectors." These are the training examples closest to the decision hyperplane. The hyperplane is the plane (in two-dimensional spaces) or the hyperplane (in multidimensional spaces) that separates the two classes. Support vectors define the location of this hyperplane.

3. *Margin:* The main goal of the OCSVM is to find a hyperplane that maximizes the "margin." The margin is the distance between the decision hyperplane and the nearest positive class data point. This distance is important because it indicates how safe the model is in its decision.

4. *Decision function:* Once the hyperplane is found, the model can be used to classify new data points. The distance between a data point and the decision hyperplane is calculated, and this value is compared with the margin. Data points that fall within the margin are considered anomalous, while those outside the margin are considered normal or belonging to the positive class.

5. *Parameters:* The OCSVM has parameters that need to be adjusted during training. The most important parameter is the margin width, which can be adjusted to fit the specific data and the tolerance level for outliers. Finding the optimal values for these parameters is crucial for accurate performance.

How OCSVM works:
Now that we have an understanding of the basic principles, let's look at how OCSVM works in practice:

1. *Training:* OCSVM training begins with defining the positive class and collecting examples of this class. The examples of this class will constitute the "support vectors" and will be used to find the decision hyperplane.

2. *Calculating the decision hyperplane:* Once the positive class examples have been collected, the OCSVM uses optimization algorithms to find the deci-

sion hyperplane that maximizes the margin between the hyperplane itself and the support vectors. This process can be computationally intensive but is essential to identify outliers accurately.

3. *Data classification:* After training the model and finding the decision hyperplane, the model can be used to classify new data points. The distance between a data point and the decision hyperplane is calculated and compared with the margin. If the distance is less than the margin, the data point is classified as an outlier; if it is greater than the margin, it is classified as normal or belonging to the positive class.

Mathematical formulation:
Two notable formulations of One-Class SVM are those proposed by Schölkopf and Platt and the one by Tax and Duin. Let's discuss each of these formulations:

1. *Schölkopf and Platt's Formulation:* The One-Class SVM formulation by Schölkopf and Platt is based on the idea of finding a hyperplane that separates the data from the origin (the center of the data distribution) while maximizing the margin.
   Given a dataset with n data points $x_1, x_2, \ldots, x_n$ in a feature space $\mathbb{R}^d$, the One-Class SVM seeks to find a hyperplane represented by the weight vector $\mathbf{w}$ and the bias term b that maximizes the margin around the origin while containing most of the data points. Additionally, it introduces a slack variable ξ_i for each data point to allow for some data points to fall within

the margin. The optimization problem can be formulated as follows:

$$\begin{aligned} \text{Minimize:} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - b \\ \text{Subject to:} \quad & \mathbf{w}\cdot\phi(\mathbf{x}_i) \geq b - \xi_i, \quad i = 1, 2, \ldots, n \\ & \xi_i \geq 0, \quad i = 1, 2, \ldots, n \end{aligned}$$

Where:

- $\phi(\mathbf{x}_i)$ is the feature map that maps the input data point $\mathbf{x}_i$ into a higher-dimensional space (typically the same space as the original data or a higher-dimensional space induced by a kernel function).
- $\|\mathbf{w}\|$ is the Euclidean norm of the weight vector $\mathbf{w}$.
- b is the bias term.
- ξ_i are slack variables.
- ν is a user-defined hyperparameter that controls the trade-off between maximizing the margin and tolerating inliers within the margin. It's typically a value between 0 and 1.

The goal of the optimization problem is to find the weight vector $\mathbf{w}$ and bias term b that minimize the objective function while satisfying the constraints. Once these parameters are found, the decision function for classifying new data points is given by:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}\cdot\phi(\mathbf{x}) - b)$$

Where $f(\mathbf{x})$ returns +1 for inliers and -1 for outliers.

2. *Tax and Duin's Formulation:* The formulation by Tax and Duin takes a different approach. It is based on finding the minimum enclosing ball that contains all the data points in the feature space. The objective is to minimize the radius of the enclosing hypersphere (ball) while maximizing the number of data points inside it.

   Minimize a cost function R^2 with the constraint that every point lies within or on the hypersphere. Therefore, our constraint is $|\mathbf{x}_i - \mathbf{a}|^2 \leq R^2$ for all i. As it stands, outliers will greatly affect the tuning. Let's address this by modifying the cost function to $R^2 + C\sum_i \xi_i$ and the constraint to $|\mathbf{x}_i - \mathbf{a}|^2 \leq R^2 + \xi_i$ for all i.

   ξ_i are the positive weights associated with each data point. The higher it is, the less that particular data point affects the tuning of R. Also, C provides a trade-off between volume and classification errors.

   Combining this with the method of Lagrange Multipliers, we are left with an optimization problem to solve:

$$\mathcal{L}(R, \mathbf{a}, \alpha_i, \gamma_i, \xi_i) = R^2 + C\sum_i \xi_i - \sum_i \alpha_i(R^2 + \xi_i - (|\mathbf{x}_i|^2 - 2\mathbf{a}\cdot\mathbf{x}_i + |\mathbf{a}|^2)) - \sum_i \gamma_i\xi_i$$

   where α_i and γ_i are non-negative Lagrange multipliers. $\mathcal{L}$ should be maximized with respect to α_i and γ_i, but minimized with respect to R, $\mathbf{a}$, and ξ_i.

Advantages of OCSVM:

1. *Anomaly Detection Focus:* One-Class SVM is renowned for its effectiveness in anomaly detection. In the context of cybersecurity, our primary concern is to identify malicious payloads among a sea of legitimate data. One-Class SVM excels at this task by building a boundary that encapsulates the malicious payloads while classifying outliers as normal (non-malicious) payloads.Non-Linearity Handling: Cyber threats are increasingly sophisticated and can exhibit complex, non-linear patterns. One-Class SVM is well-suited to handle non-linear data distributions through the use of kernel functions, which allows it to capture intricate relationships within the feature space. This flexibility is crucial for identifying evolving and intricate XSS attack patterns.

2. *Non-Linearity Handling:* Cyber threats are increasingly sophisticated and can exhibit complex, non-linear patterns. One-Class SVM is well-suited to handle non-linear data distributions through the use of kernel functions, which allows it to capture intricate relationships within the feature space. This flexibility is crucial for identifying evolving and intricate XSS attack patterns.

3. *Robustness to Imbalanced Data:* In cybersecurity, it's common to encounter imbalanced datasets where the number of malicious samples is significantly smaller than benign ones. One-Class SVM is robust to imbalanced data, making it ideal for this scenario. It does not require a perfectly balanced dataset and can still effectively distinguish malicious samples from the benign ones.

4. *Scalability:* Depending on the scale of internet traffic and the number of payloads, scalability can be a significant concern. One-Class SVM is com-

putationally efficient and can handle large datasets without a substantial increase in computational overhead, making it feasible for real-world, high-traffic applications.

5. *Interpretability and Tunability:* One-Class SVM provides interpretable results, allowing security experts to gain insights into the characteristics that define malicious payloads.

However, OCSVM also has some limitations:

1. *Sensitivity to parameters:* The effectiveness of OCSVM depends on the choice of parameters, especially the width of the margin. The choice of these parameters can be critical and requires careful optimization. Poor parameter choice can lead to poor model performance.

2. *Data representation:* The effectiveness of OCSVM can be affected by data representation. If the data are not represented properly, the model may fail to correctly identify outliers. It is important to carefully examine the data preparation stage to ensure that the model can take advantage of the available information.

3. *Limited to the positive class:* Because the OCSVM focuses only on the positive class, it may have difficulty identifying outliers that are very similar to the positive class. If the outliers are structurally similar to the positive class examples, the model may fail to detect them accurately.

Conclusions:
In summary, the One-Class Support Vector Machine is a powerful and versatile machine learning technique for detecting outliers in datasets dominated by a single class. However, it is important to understand the basic principles, experiment with the parameters, and prepare the data accurately for optimal performance.

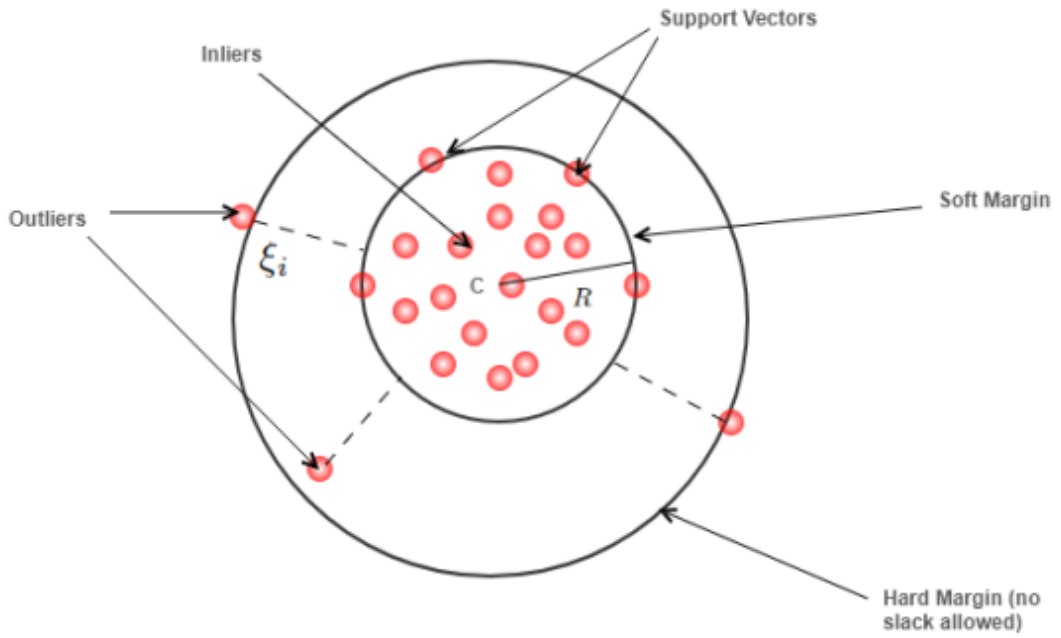

Figure 2.2: One-Class SVM.

2.2.3 Decision-Making with Dual One-Class SVM

In this research, two One-Class SVM models have been introduced, capable of working simultaneously if the user chooses to activate this mode. However, a critical question arises when these two models provide conflicting results for the same input payload. How should we make an informed decision about which result to trust? To address this challenge, this study presents two distinct evaluation modalities when utilizing both models together:

1. *Aggressive assessment of attacks*

    - In this evaluation mode, if at least one of the two models detects the payload as a potential XSS attack, the payload is classified as malicious without further inspection. While this approach may lead

to an increase in false positives, it significantly reduces the occurrence of false negatives.

- Each model provides a prediction, with +1 indicating an inlier and -1 indicating an outlier. Thus, when at least one of the two models predicts +1, the payload is classified as malicious.

2. *Weighted assessment of attacks*

   - In this evaluation modality, the decision is based on trusting the model for which the feature representation of the payload is farthest from the model's boundaries.
   - The degree to which the feature representation of the payload is distant from the model's boundaries reflects the robustness of the evaluation. Consequently, this approach prioritizes the more robust evaluation.

2.3 Technical Implementation

2.3.1 Research Subjects and Technological Components

As we have to treat the technical part of the elaborate, it's fundamental to underlying what are the subject of this work:

- HyperText Transfer Protocol (HTTP)
- Mithril WAAP
- Cross-Site Scripting (XSS)

HyperText Transfer Protocol:

The internet, in its present form, relies heavily on the Hypertext Transfer Protocol (HTTP) for the transfer of information between clients and servers. HTTP operates on a client-server model, where clients (e.g., browsers) send requests to servers hosting web resources. Servers respond with requested content or error messages. The statelessness of HTTP is maintained through this model.

HTTP requests consist of a request line, headers, and an optional message body. The request line contains the HTTP method, the requested resource (Uniform Resource Identifier or URI), and the protocol version.

Example request:

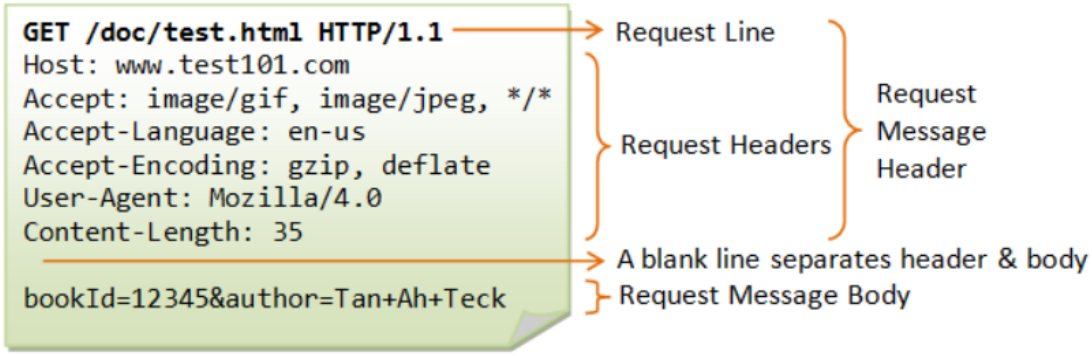

Figure 2.3: HTTP request example.

HTTP responses comprise a response line, headers, and an optional message body. The response line includes the protocol version, a status code, and a reason phrase.

HTTP headers provide detailed information about the request, including the client's capabilities, preferences, and the context of the request. These headers are crucial for servers to understand how to handle the request and respond appropriately.

While not all HTTP requests include a message body, some methods, such as

POST and PUT, do. The message body carries data to be sent to the server for processing. The format and content of the message body depend on the specific request and the server's expectations.

Example response:

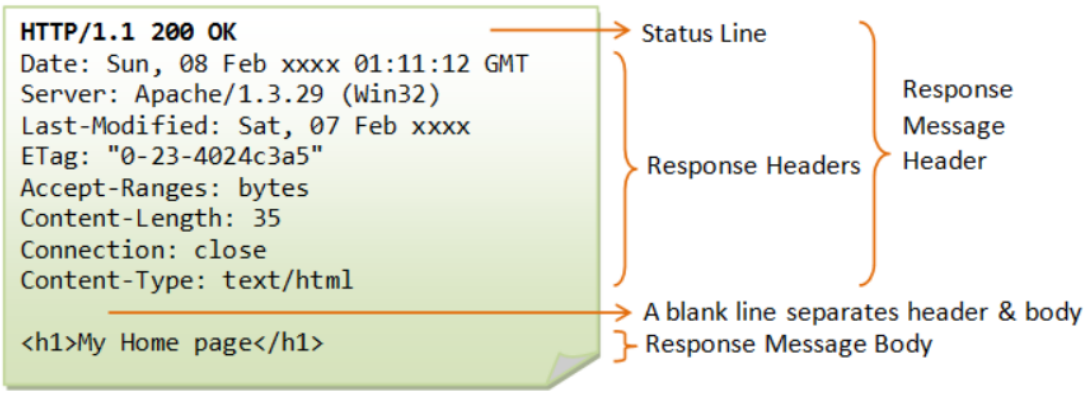

Figure 2.4: HTTP response example.

HTTP responses, like requests, consist of a response line, headers, and an optional message body. The response line serves as a crucial identifier for the server's reply and provides essential information for the client to interpret the server's response.

The *response line* comprises three primary components:

HTTP Version (this field indicates the version of the HTTP protocol being used by the server to generate the response), Status Code (the status code is a numeric three-digit code that succinctly communicates the result of the server's attempt to satisfy the client's request), Reason Phrase (the reason phrase is a human-readable description that provides further context for the status code. While it aids human understanding, it is not used by clients to determine the meaning of the status code).

HTTP response headers supply detailed information about the server's reply, including the content type, cache directives, and server information. These

headers are crucial for clients to understand how to process the server's response.
While not all HTTP responses include a *message body*, it is commonly present, especially in responses to GET requests or API calls. The message body carries the resource or data sent by the server in response to the client's request. The format and content of the message body depend on the specific response and the server's configuration.

Mithril WAAP:
A Web Application and API Protection (WAAP) solution is an advanced security system designed to safeguard web applications and APIs against a wide range of threats and vulnerabilities. It provides comprehensive protection for both web applications and the APIs that support them. Here's a technical explanation of how a WAAP solution functions:

1. *Ingress and Egress Traffic Inspection:*
    - A WAAP solution sits between clients (e.g., web browsers or mobile apps) and web applications or APIs, serving as a security gateway.
    - It inspects all incoming traffic (requests) and outgoing responses to and from web applications and APIs.
2. *Request and Response Analysis:*
    - When a client initiates an HTTP request or API call, the WAAP solution performs a detailed analysis of the request.
    - It examines various attributes, including HTTP methods, headers, parameters, payload content, and API endpoint URLs.
    - The WAAP uses a combination of signature-based detection,

behavioral analysis, and machine learning to identify potential threats, attacks, and anomalies.

3. *Security Policies and Rule Enforcement:*

   - WAAP solutions come with a set of predefined security policies and rules that are designed to recognize known attack patterns and malicious behavior.
   - Each incoming request and outgoing response is compared against these policies and rules to detect and prevent attacks and vulnerabilities.
   - Actions taken by the WAAP include blocking malicious requests, modifying responses to remove sensitive information, and logging incidents for further analysis.

4. *Protection Against Various Threats:*

   - WAAP solutions offer protection against a wide array of security threats and attacks, including:
     - SQL Injection: Detecting and mitigating attempts to inject malicious SQL queries into input fields.
     - Cross-Site Scripting (XSS): Preventing the injection of malicious scripts into web pages.
     - API Abuse: Detecting and blocking excessive or unauthorized API calls.
     - Brute Force Attacks: Throttling or blocking repeated login attempts.
     - Data Exfiltration: Identifying and preventing unauthorized data leaks from APIs.

- Rate Limiting: Managing the rate of incoming requests to prevent abuse and DDoS attacks.

5. *Learning and Adaptation:*

   - Many WAAP solutions have machine learning and adaptive capabilities to learn the normal behavior of web applications and APIs.
   - They adapt their security policies and rules based on the evolving threat landscape and the changing patterns of application usage.

6. *Logging and Reporting:*

   - WAAP solutions maintain detailed logs of all traffic, security incidents, and actions taken.
   - Security administrators can analyze these logs to gain insights into security posture, investigate incidents, and generate reports for compliance purposes.

7. *Customization:*

   - WAAP solutions offer customization options to tailor security policies and rules to the specific requirements of web applications and APIs.
   - Security professionals can fine-tune configurations to balance security and functionality.

8. *API Security:*

   - WAAP solutions specifically address API security concerns, ensuring that APIs are protected against threats such as injection attacks, excessive API calls, and data leakage.

Cross-Site Scripting:

Let's go back to the concepts previously illustrated and see in greater detail what we are dealing with.

Cross-Site Scripting (XSS) is a prominent and highly exploitable web application security flaw. It arises due to inadequate input validation and output encoding practices within web applications. Attackers leverage this vulnerability to inject malicious scripts into web pages, causing the scripts to execute in the browsers of unsuspecting users. The consequences of XSS attacks can be devastating, ranging from data leakage to complete compromise of user accounts. There are three primary types of XSS vulnerabilities:

1. *Stored XSS (Persistent XSS):* In Stored XSS, the malicious script is permanently stored on the target server and executed whenever a user retrieves the compromised content.
   Example:
   Consider a social media platform where users can create profiles with descriptions that are displayed to other users. An attacker creates a profile with the following bio:

```
<script>
  // Malicious script to steal user cookies
  var maliciousURL = 'https://attacker.com/steal.php?cookie='
  + document.cookie;
  var img = new Image();
  img.src = maliciousURL;
</script>
```

2. *Reflected XSS:* In Reflected XSS, the malicious script is embedded in a URL or form input, and it's reflected off the web server onto the victim's

browser. It's typically a one-time attack.

Example:

Suppose a vulnerable search feature on a website reflects the search query in the page content without proper validation:

User's input:

```
"><script>alert("XSS");</script><"
```

The resulting URL might look like this:

```
"https://example.com/search?query=%22%3E%3Cscript%3Ealert
(%22XSS%22)%3B%3C%2Fscript%3E%3C%22"
```

When a victim clicks on this URL, the script is executed in their browser, triggering an alert with "XSS."

3. *DOM-Based XSS:* They occurs when the client-side scripts in a web page manipulate the Document Object Model (DOM) in an insecure way, leading to the execution of malicious code.

   Example:

   Imagine a web application that retrieves a user's name from the URL and displays it on the page:

   URL:

   ```
   "https://example.com/page?name=
   <script>maliciousFunction()</script>"
   ```

   The JavaScript code on the page might include:

   ```
   // Extracts the name parameter
   var name = window.location.search.substring(6);
   document.getElementById("welcome-message").innerHTML =
   "Welcome, " + name + "!";
   ```

In this case, if the URL is manipulated as shown, the script

"<script>maliciousFunction()</script>"

is executed in the user's browser, leading to a security breach.

The technological components utilized in this study can be summarized as follows:

1. *Python:*
   Python served as the cornerstone for the entire implementation process, spanning from payload pre-processing to the final testing phase. Renowned for its versatility and robust capabilities, Python proved to be an ideal choice for addressing multifaceted challenges in the realms of machine learning and cybersecurity. Within the scope of this project, Python was harnessed to construct a One-Class Support Vector Machine (OneClass-SVM) classifier, designed specifically for discerning XSS (Cross-Site Scripting) attacks from benign requests.

2. *Machine Learning (ML) Libraries:*
   To lay the foundation for model development, training, and evaluation, the Scikit-learn library was employed. This open-source machine learning library for Python furnishes an extensive arsenal of tools tailored for data analysis and modeling. Scikit-learn stands out due to several pivotal attributes that firmly establish its role as a linchpin in the machine learning landscape. It offers an intuitive interface accommodating both novices and experts, facilitating streamlined model development. The library encompasses a wide array of machine learning algorithms spanning classification, regression, clustering, dimensionality reduction, and more, effectively serving as a versatile toolkit. Additionally, Scikit-learn

simplifies essential data preprocessing tasks such as normalization and feature scaling, thereby enhancing model performance. Furthermore, it excels in model selection and evaluation, providing indispensable tools for cross-validation, hyperparameter tuning, and metric assessment.
To ingest and manipulate data from the input ".csv" file representing the dataset, the Pandas library was harnessed. Pandas, an open-source Python library, assumes a pivotal role in data manipulation and analysis due to its core functionalities. Notably, Pandas introduces dataframes, akin to tabular structures, facilitating the intuitive representation and manipulation of structured data. The library excels in data import and export, fostering seamless interaction with diverse data sources. It proves invaluable in data cleaning and transformation endeavors, offering tools for addressing missing values, duplicates, and outliers with consummate ease. Furthermore, Pandas showcases remarkable prowess in data selection and indexing, enabling the effortless extraction of specific data subsets. Its potent grouping and aggregation functions bolster data summarization and analysis. For time series data, Pandas emerges as the preferred choice, armed with specialized features for time-related analyses. While not a dedicated data visualization library, Pandas integrates seamlessly with data visualization tools, thereby imbuing analyses with a layer of visual insight.
For visualizing evaluation results, the Matplotlib library was employed. Matplotlib, an open-source data visualization library for Python, bestows a comprehensive suite of features for crafting compelling visualizations. Its chief strength resides in generating publication-ready plots, encompassing various types such as line plots, scatter plots, bar charts, histograms, and more, all with a strong emphasis on customization. With

support for multiple output formats, Matplotlib adapts effortlessly to diverse presentation and publication requirements. Operating through an object-oriented interface, it promotes modular and reusable code development while granting precise control over plot elements. Matplotlib's versatility extends to both 2D and 3D plots, granting users the power to visualize data in three dimensions. The library seamlessly integrates with other Python tools like NumPy and Pandas, thus simplifying the transformation of data into graphical representations.

3. *Elasticsearch:*

   The XSS attack payloads are securely archived within the Elasticsearch database, which serves as the vital repository for the core data underpinning our research efforts. Elasticsearch, an open-source distributed search and analytics engine developed by Elastic, serves as a foundational component of this study. Originally conceived for full-text search, Elasticsearch has evolved into a potent platform for data exploration and analysis. It is celebrated for its exceptional speed, scalability, and user-friendliness, rendering it a popular choice across a broad spectrum of applications, ranging from website search engines to log and event data analysis.

   At its core, Elasticsearch constitutes an extraordinary search and analytics engine that offers a plethora of features and capabilities. Its standout feature lies in its lightning-fast full-text search capabilities across extensive datasets. The distributed architecture designed by Elastic ensures seamless scalability, promoting high availability and fault tolerance. Real-time data ingestion is yet another hallmark feature, making it the ideal choice for applications necessitating up-to-the-minute insights. Elasticsearch's flexible JSON-based document structure accommodates

diverse data types and simplifies schema-free data handling.
The RESTful API simplifies interactions through standard HTTP requests, while advanced querying options, including boolean operations, phrase matching, and aggregations, empower users to extract valuable insights efficiently. Elasticsearch's compatibility with data visualization tools like Kibana extends its utility, fostering enhanced data analysis. Its robust security features, integration with machine learning capabilities, and the vibrant user community further solidify its reputation as an indispensable tool for modern data management and analysis.

2.3.2 Data Pre-processing

Pre-processing pipeline:

Figure 2.5: Pre-processing pipeline schema.

In this master's thesis, the primary objective is to develop a predictive model capable of making predictions based on raw payloads. These payloads may be either obfuscated or unobfuscated, necessitating the inclusion of methods to identify and decode them. Subsequently, the decoded payload undergoes a preprocessing step where extraneous elements are removed to enhance the meaningful content for detection. This includes tasks such as eliminating excess white spaces, addressing character repetitions, excluding non-executable HTML comments (which are ignored by web browsers), and converting all text to lowercase. Finally, the processed payload is subjected to a Tokenization block, which serves to remove non-displayable characters and segment the payload into a list of tokens. These three phase are now explained in detail.

`Decoding:`

The initial payload undergoes a decoding process facilitated by a Decoder block that accommodates various decoding methods:

- *Unicode*
- *Hexadecimal*
- *Base64*
- *HTML encoding*
- *URL encoding*
- *double encoding*

This block functions by detecting the presence of any of the aforementioned encoding types within the payload. When an encoding type is detected, the block decodes the payload and subsequently reprocesses the newly decoded text through the decoding block. This recursive process continues until the input text remains unaltered, signifying the completion of the decoding operation.

`Generalization:`

Following decoding, the payload proceeds through a Generalization block responsible for enhancing its quality. This process involves the removal of excess white spaces, conversion of all text to lowercase, deletion of HTML comments, and the elimination of redundant punctuation and character repetitions. Specifically, character sequences repeated at least four times consecutively are replaced with three instances of the same character type, taking into account potential occurrences of three consecutive characters such as "www" which typically signify the beginning of a URL.

`Tokenization:`

After the payload has undergone the generalization process, it proceeds to a Tokenization block where further refinement takes place. Within this block, non-displayable characters are removed, and the payload is intelligently segmented into a coherent list of tokens. Tokenization is a crucial step in the data processing pipeline as it transforms the payload into a structured format that is amenable to subsequent analysis and prediction tasks. This list of tokens serves as the foundation upon which various analytical techniques and machine learning models can be applied to extract meaningful insights and make predictions with increased accuracy.
Now a real example of how the pre-processing line works.

`Raw Payload:`

query%3D%3CIMG%2B%22%22%22%3E%3CSCRIPT%3Ealert
%28%22cyb3r_pr3dat0r%22%29%3C/SCRIPT%3E%22%3E
%26fuzzy%3Dtrue%26ct%3Dnull%26autobounce%3Dtrue

In this example there is an URL-encoded string that contains an XSS attack, in the following step the decoder will decode it and it's possible to understand the malign content.

`Decoded Payload:`

query=<IMG+"""><SCRIPT>alert("cyb3r_pr3dat0r")
</SCRIPT>">&fuzzy=true&ct=null&autobounce=true

From the decoded payload we can see that this is a "Reflected XSS" attack.This type of attack occurs when a web application takes user input and includes it directly in the HTML response without proper validation or sanitization. Attackers exploit this vulnerability by injecting malicious scripts into the input,

which are then executed in the victim's browser when they view the web page. When this string is included in the web page as part of the HTML response, the browser interprets it as a series of HTML tags and JavaScript script.
Specifically, it includes an empty `<IMG>` tag (which is just a diversion) followed by a JavaScript script that displays a popup alert with the text `"cyb3r_pr3dat0r"` when the page loads.
The attack can be triggered if a user visits a URL that contains this malicious input, or if the input is reflected in a web page viewed by another user. The attackers' goal is typically to steal sensitive information or carry out malicious actions on behalf of the victim user.

`Generalized Payload:`

```
query=<img+"""><script>alert("cyb3r_pr3dat0r")
</script>">&fuzzy=true&ct=null&autobounce=true
```

Once decoded, this payload proceeds through a Generalization block, where it undergoes comprehensive refinement to ensure its quality and suitability for further processing. The Generalization block employs several essential operations:

- *Whitespace Removal:* Any extraneous blank spaces are diligently stripped away, ensuring a cleaner and more concise payload.
- *Lowercasing:* The entire text is uniformly converted to lowercase, ensuring consistent casing throughout the payload.
- *HTML Comment Removal:* Non-executable HTML comments, are intelligently deleted, enhancing the payload's clarity.
- *Punctuation and Character Cleanup:* Unnecessary punctuation and character repetitions are addressed.

The resulting generalized payload is now optimized for subsequent processing stages, making it more amenable to analysis and prediction tasks. Following generalization, the payload proceeds to tokenization.

`Tokenized Payload:`

```
['query=', '<', 'img+', '"', '"', '"', '>',
'<', 'script', '>', 'alert', '(', '"', 'cyb3r_pr3dat0r',
'"', ')', '<', '/script', '>', '"', '>', '&',
'fuzzy=true', '&', 'ct=null', '&', 'autobounce=true']
```

The provided tokenized payload is a list of individual elements, each representing a portion of the processed data. This list is the result of a Tokenization block applied to the generalized payload, which serves as the input. The Tokenization block conducts two key operations:

- *Removal of Undisplayable Characters:* Within this block, any non-displayable or extraneous characters are meticulously eliminated from the payload. This ensures that the list of tokens remains free of any elements that do not contribute to the payload's meaningful content.

- *Token Segmentation:* After the character cleanup process, the payload is intelligently split into discrete tokens. Each token represents a distinct unit of information or a meaningful component of the payload. These tokens are identified based on contextual boundaries within the text, such as spaces, punctuation, and other delimiters.

The payload is now ready for `Features Extraction`.

2.3.3 Features Extraction and Transformation

Features extraction is a fundamental process in data analysis and machine learning, playing a pivotal role in transforming raw data into meaningful information for the model. In the realm of data-driven research, understanding and mastering feature extraction techniques are paramount for extracting valuable insights and facilitating accurate modeling.
It refers to the process of selecting or transforming the most relevant attributes or characteristics from a dataset while discarding redundant or noisy information. It serves multiple essential purposes in various fields, including but not limited to:

- *Dimensionality Reduction:* In high-dimensional datasets, feature extraction reduces the number of variables, simplifying subsequent analysis and modeling. This aids in mitigating the curse of dimensionality, enhancing computational efficiency, and often improving the model's performance.

- *Improved Interpretability:* Extracted features are typically more interpretable than raw data, making it easier to discern patterns, relationships, and meaningful information within the data. This is especially crucial when the research aims to provide actionable insights.

- *Noise Reduction:* Feature extraction can help filter out irrelevant or noisy data, leading to more robust and accurate analyses. This is particularly important when dealing with real-world data, which often contains inconsistencies or irrelevant information.

- *Enhanced Generalization:* Extracted features can capture the underlying structure of the data, promoting better generalization in machine learning

models. By focusing on relevant information, models are less prone to overfitting on training data.

Common Feature Extraction Techniques:
Several feature extraction methods are available, each tailored to specific data types and research objectives. Some widely used techniques include:

- *Principal Component Analysis (PCA):* PCA is a dimensionality reduction technique that identifies linear combinations of features that explain the maximum variance in the data. It is particularly useful for reducing dimensionality while retaining as much information as possible.
- *Feature Scaling and Normalization:* Techniques like Min-Max scaling and Z-score normalization are used to transform data into a common scale, ensuring that features contribute equally to the analysis.
- *Feature Engineering:* This involves creating new features from existing ones or domain-specific knowledge. Feature engineering allows for the incorporation of domain expertise into the analysis and often leads to more informative features.
- *Deep Learning-based Techniques:* Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) can automatically learn relevant features from raw data, making them powerful tools for feature extraction in image, text, and sequential data analysis.

In the context of this master's thesis, as previously mentioned, we employ two distinct models: two One-Class SVM models, each tailored to specific tasks, one designed for analyzing JavaScript code, and the other optimized for HTML. Consequently, we are working with two distinct sets of features.

`JavaScript Features:`

- *"PayloadLength":* Length of the payload, under a logarithmic scale.
  - The length of the payload is crucial because XSS attacks often involve injecting malicious code into a web application. Longer payloads might indicate more complex and potentially dangerous attacks.
- *"document":* Represent all the call did to the document object representing the web page.
  - XSS attacks frequently manipulate the Document Object Model (DOM) to inject malicious scripts into a web page. Tracking calls to the "document" object helps identify attempts to modify or access the web page's content, which is a common XSS attack vector.
- *"jsObject":* Matches JavaScript object literals, e.g., " key: value ".
  - JavaScript object literals are often used to create and manipulate data structures. Recognizing them is crucial because attackers might use them to construct payloads that interact with the DOM or other sensitive objects in unexpected ways.
- *"jsAttr":* Matches JavaScript attribute references, e.g., ".attributeName".
  - Attacker-controlled attribute references can be exploited to inject malicious code or manipulate the behavior of HTML elements. Monitoring references to attributes helps detect attempts to alter the behavior or appearance of web page elements.
- *"jsMethods":* Matches JavaScript method calls, e.g., ".methodName(", plus a list of often used well-known methods.

  - XSS attacks may involve invoking JavaScript methods to perform actions on the page. Recognizing these calls, especially well-known and often abused methods, is essential for detecting potentially harmful activities within the code.

- *"cookie":* Matches calls to cookies.

  - XSS attacks often aim to steal or manipulate user session data, and cookies are a common target. Identifying code that interacts with cookies is crucial for spotting malicious attempts to read, write, or manipulate user session information.

- *"structural":* Number of punctuation used.

  - The number of punctuation characters used can provide insights into code complexity and potential obfuscation techniques employed by attackers. Unusually high or specific combinations of punctuation may indicate attempts to hide malicious intent.

- *"structComb":* Number of punctuation combinations used.

  - Punctuation combinations can be used to create complex and obfuscated code, making it difficult to detect malicious intent. Counting the number of unique punctuation combinations helps the model identify patterns consistent with XSS attack attempts.

- *"sensitiveWords":* list of words that are usually present on malicious payloads.

  - Maintaining a list of words commonly found in malicious payloads is important because it allows the model to flag payloads that contain

suspicious language. This feature helps detect known patterns and keywords that are often associated with XSS attacks.

- *"closeB":* Count of closed brackets.
  - Counting closed brackets is important because attackers often manipulate code syntax to evade detection by inserting unbalanced brackets or parentheses. Detecting an abnormal number of closed brackets can indicate an attempt to disrupt the code's structure, potentially revealing an attack.
- *"openB":* Count of opened brackets.
  - Counting opened brackets complements the closed bracket count. Anomalies in the number of opened brackets can indicate an attempt to manipulate code structure. Balancing opened and closed brackets is essential for valid JavaScript code, and detecting discrepancies can help identify potential attacks.

`HTML Features:`

- *"PayloadLength":* Length of the payload, under a logarithmic scale.
- *"HtmlTag":* Matches the HTML tags.
  - HTML tags are a fundamental component of web pages. Recognizing them is essential because XSS attacks often involve manipulating or injecting malicious code within HTML tags. Monitoring HTML tags helps identify attempts to inject scripts or manipulate the page's structure.
- *"HtmlAttr":* Matches the HTML attributes.

  - HTML attributes provide additional information about HTML elements. Attackers may attempt to exploit these attributes to inject malicious code or manipulate the behavior of web page elements. Identifying HTML attribute usage helps detect such attempts.

- *"ExtLink":* Search for external link called from "src" or "href" attributes.

  - External links specified in "src" or "href" attributes can be used to load malicious content from remote sources. Detecting external links helps identify attempts to load potentially harmful scripts or resources from untrusted domains.

- *"ExtJs":* Search for script inside the payload.

  - XSS attacks often involve embedding JavaScript within HTML. Detecting the presence of scripts within the payload is crucial because it can indicate an attempt to execute malicious code on the client side.

- *"sensitiveWords":* list of words that are usually present on malicious payloads, such as:
  `['javascript','base64','refresh','xml','robot',`
  `'behavior','payload','xss','exec','jscript']`.

  - A predefined list of sensitive words associated with malicious payloads is valuable for identifying known patterns in XSS attacks. Detecting these words can provide early indications of a potential attack.

- *"closeB":* Count of closed brackets.

  - The count of closed brackets is essential for ensuring proper HTML structure. Anomalies in bracket usage can indicate an attempt to break out of HTML context or execute malicious code. Monitoring closed brackets helps maintain HTML integrity.

- *"openB":* Count of opened brackets.

  - Similarly, the count of opened brackets is crucial for maintaining HTML syntax. Unusual or excessive opening brackets may indicate an attempt to inject or manipulate HTML code. Tracking opened brackets helps ensure code validity.

Once the features have been extracted, it's necessary to normalize them.

Features Normalization:
Features normalization is a fundamental data preprocessing step in the field of data science and machine learning. It involves transforming the numerical values of different features within a dataset into a common scale, often with a mean of zero and a standard deviation of one. This process aims to ensure that all features contribute equally to the analysis and modeling, preventing certain attributes from dominating the learning process due to their larger magnitudes. Leveraging the power of Scikit-learn, has been performed feature-wise normalization, ensuring that each feature (column) in the data has a consistent scale.

Chapter 3

Evaluation and Results

3.1 Evaluation

The model's performance was rigorously assessed using four distinct database types, each designed to challenge its capabilities in different ways:

1. *Dataset Containing Only Benign Payloads:*
   This dataset comprises a comprehensive collection of 7,339 benign payloads. These payloads are meticulously constructed, featuring random text generation enriched with commonly used punctuations found in HTML and JavaScript. This dataset serves as a crucial benchmark to evaluate the model's capacity to accurately identify and differentiate between harmless content and potentially harmful elements. It is paramount to scrutinize the model's False Positive rate in this context, as it determines its ability to avoid flagging benign content as malicious.

2. *Dataset Containing Only Malicious Payloads:*
   Comprising 4,690 entries, this dataset exclusively contains malicious payloads. These payloads were originally sourced from online XSS

datasets but were meticulously curated to eliminate duplicates and extraneous payloads that did not constitute XSS attacks. The focus here is to test the model's ability to effectively detect and classify malicious content amidst a sea of potential threats. It is crucial to understand the False Negative rate. In this context, a critical focus lies on assessing the model's False Negative rate, as it reflects its capacity to identify actual threats accurately.

3. *Mixed Dataset:*
   A total of 1,000 payloads make up this dataset, presenting a balanced mixture of benign and malicious content. Half of these payloads are benign and consist of benign XML payloads and randomly generated text. The other half are malicious payloads. The mixed dataset is designed to assess the model's versatility in handling real-world scenarios where both benign and malicious elements coexist, closely mirroring the complexity of web content it may encounter.

4. *Mithril Dataset:*
   This dataset encompasses 234 payloads that were blocked by Mithril WAAP due to their detection as XSS attacks. These payloads are derived from actual incidents, highlighting the model's capacity to identify and thwart real-world threats. The Mithril dataset is particularly valuable as it assesses the model's practical utility in a security application context.

By subjecting the model to these diverse datasets, we ensure a comprehensive evaluation of its performance, encompassing benign and malicious payloads, mixed-content scenarios, and real-world threat simulations.

3.2 Results

	JavaScript OCSVM	HTML OCSVM	JavaScript & HTML OCSVM (Weighted)	JavaScript & HTML OCSVM (Aggressive)	AVG
Benign	99,54%	89,74%	92,93%	89,28%	92,87%
Malicious	99,50%	95,71%	99,59%	99,76%	98,64%
Mixed	90,06%	96,58%	88,35%	89,05%	91,01%
Mithril	90,17%	96,58%	95,72%	100%	95,61%
AVG	94,81%	94,65%	94,14%	94,52%	94,53%

Table 3.1: Model Performance Across Datasets.

The outcomes of the model are presented in Table 3.1, which showcases the model's performance across the diverse datasets using four distinct operational modes. The table includes the following key metrics:

- *JavaScript OCSVM:* The model's performance when focusing solely only on JavaScript features.
- *HTML OCSVM:* The model's performance when concentrating solely on HTML features.
- *JavaScript & HTML OCSVM (Weighted):* The model's performance when considering both JavaScript and HTML features with "weighted assestment" (as mentioned in section 2.2.3).
- *JavaScript & HTML OCSVM (Aggressive):* The model's performance when considering both JavaScript and HTML features with "aggressive assestment" (as mentioned in section 2.2.3).
- *AVG:* The average performance across all operational modes and datasets.

Insights:

- *Benign Detection:* The model exhibits remarkable accuracy in identifying benign content, with JavaScript OCSVM and JavaScript & HTML OCSVM (Weighted) achieving accuracy rates of 99.54% and 92.93%, respectively.

- *Malicious Detection:* When it comes to detecting malicious content, the model performs exceptionally well across all operational modes, with the highest accuracy recorded at 99.76% in the JavaScript & HTML OCSVM (Aggressive) mode.

- *Mixed Payloads:* The model maintains a solid performance of approximately 91.01% in distinguishing between mixed benign and malicious payloads.

- *Mithril Dataset:* In the context of real-world threats, the model demonstrates its effectiveness by achieving an accuracy rate of 95.61% in detecting Mithril-blocked XSS attacks.

- *Overall Performance (AVG):* The model maintains a high average accuracy rate of 94.53% across all operational modes and datasets, reaffirming its robustness and adaptability in web application security.

The following images represents the confusion matrix for each test dataset previously mentioned:

OCSVM against benign test set:

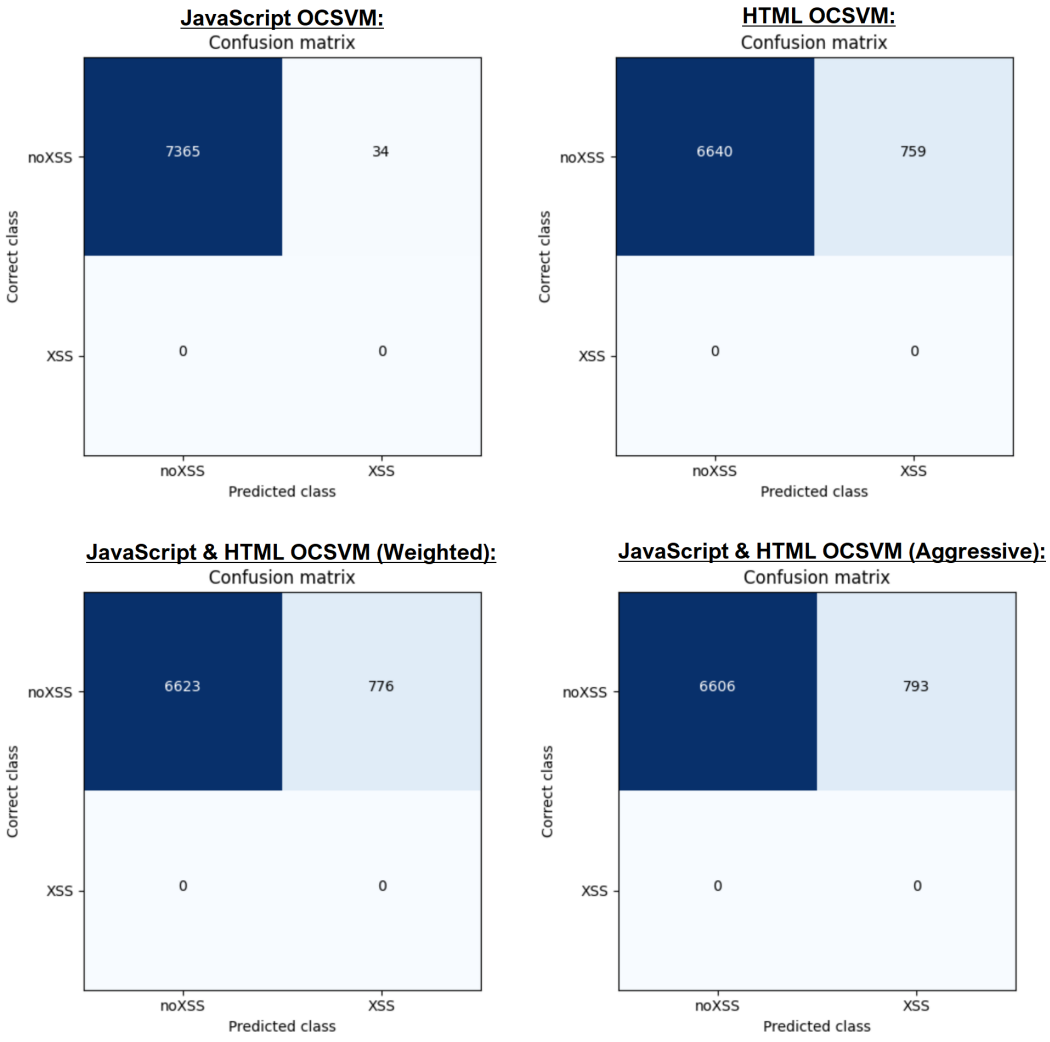

Figure 3.1: Confusion Matrix against benign test set.

OCSVM against malicious test set:

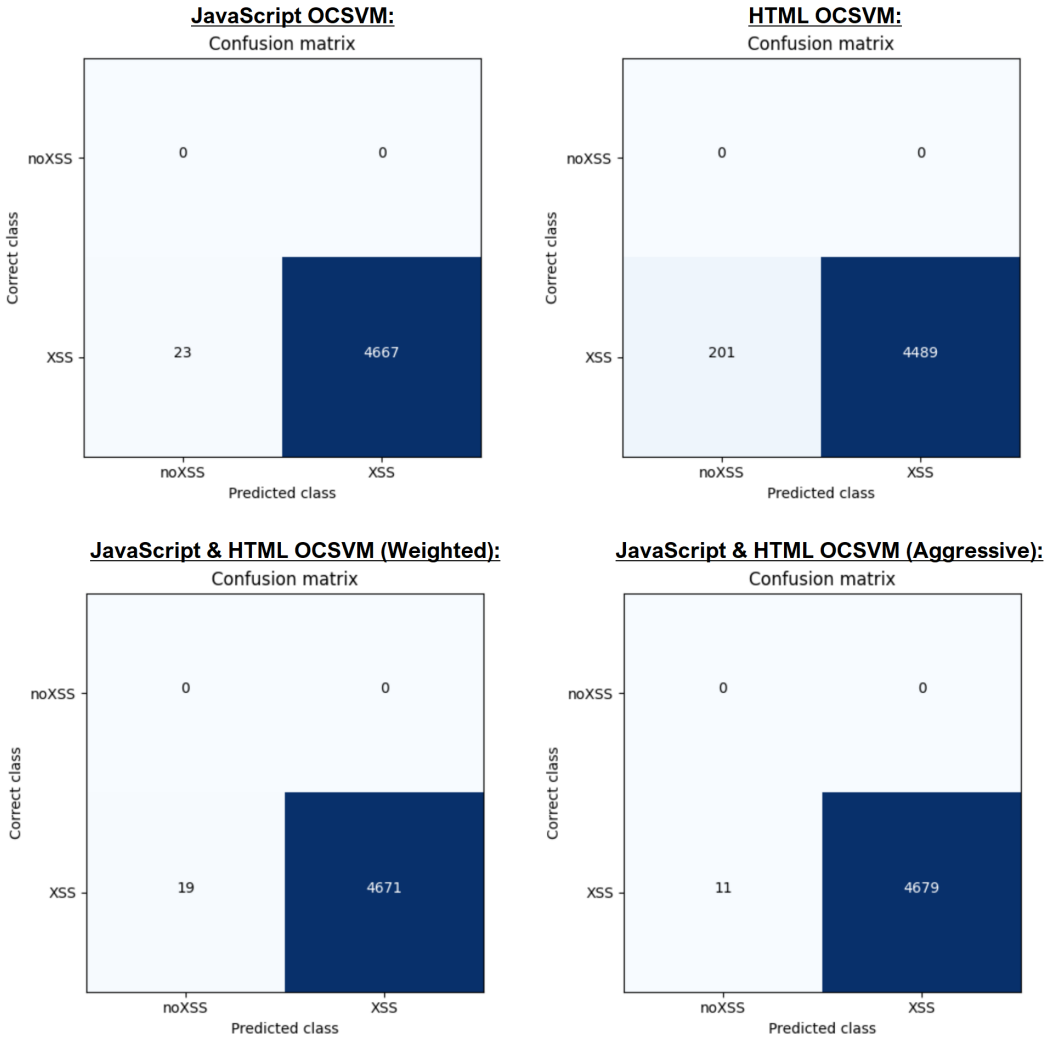

Figure 3.2: Confusion Matrix against malicious test set.

OCSVM against mixed test set:

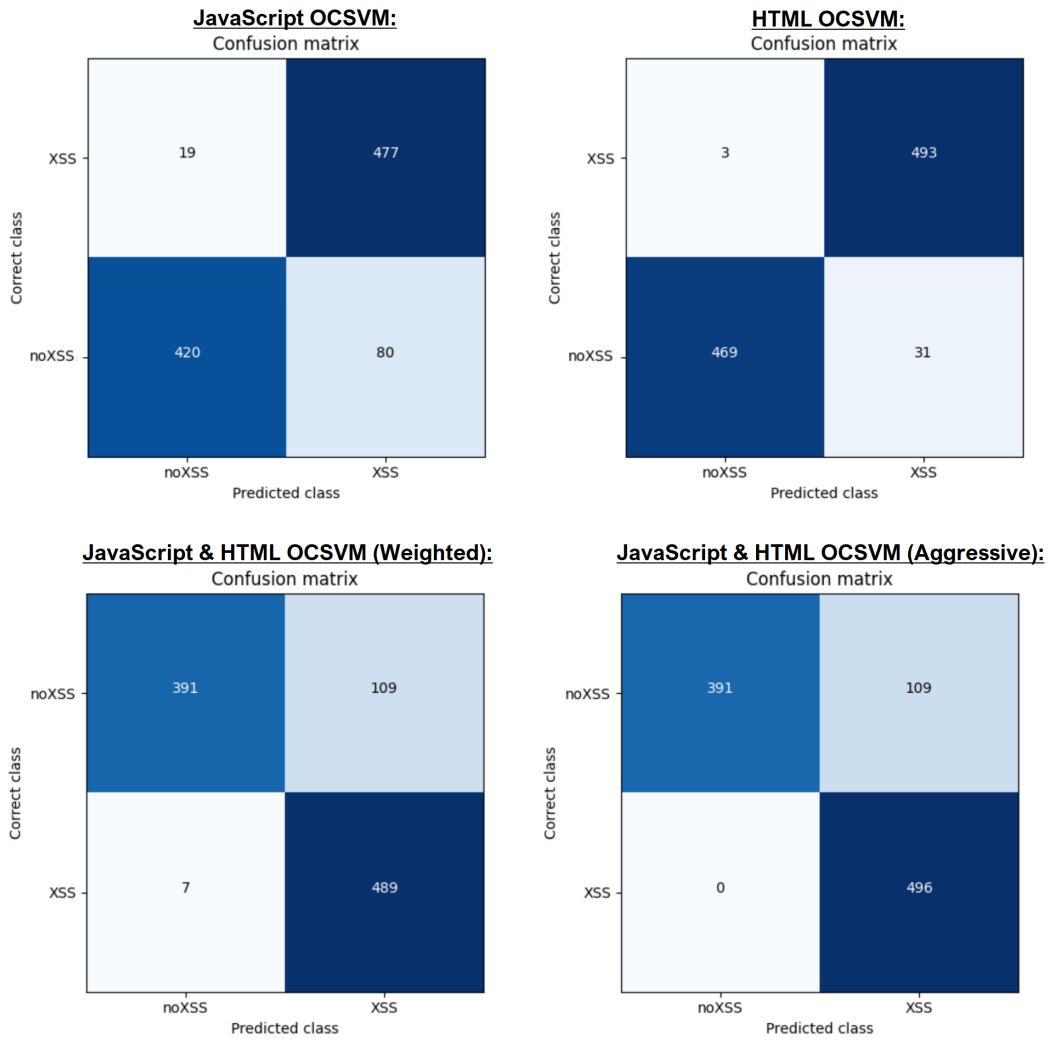

Figure 3.3: Confusion Matrix against mixed test set.

OCSVM against Mithril test set:

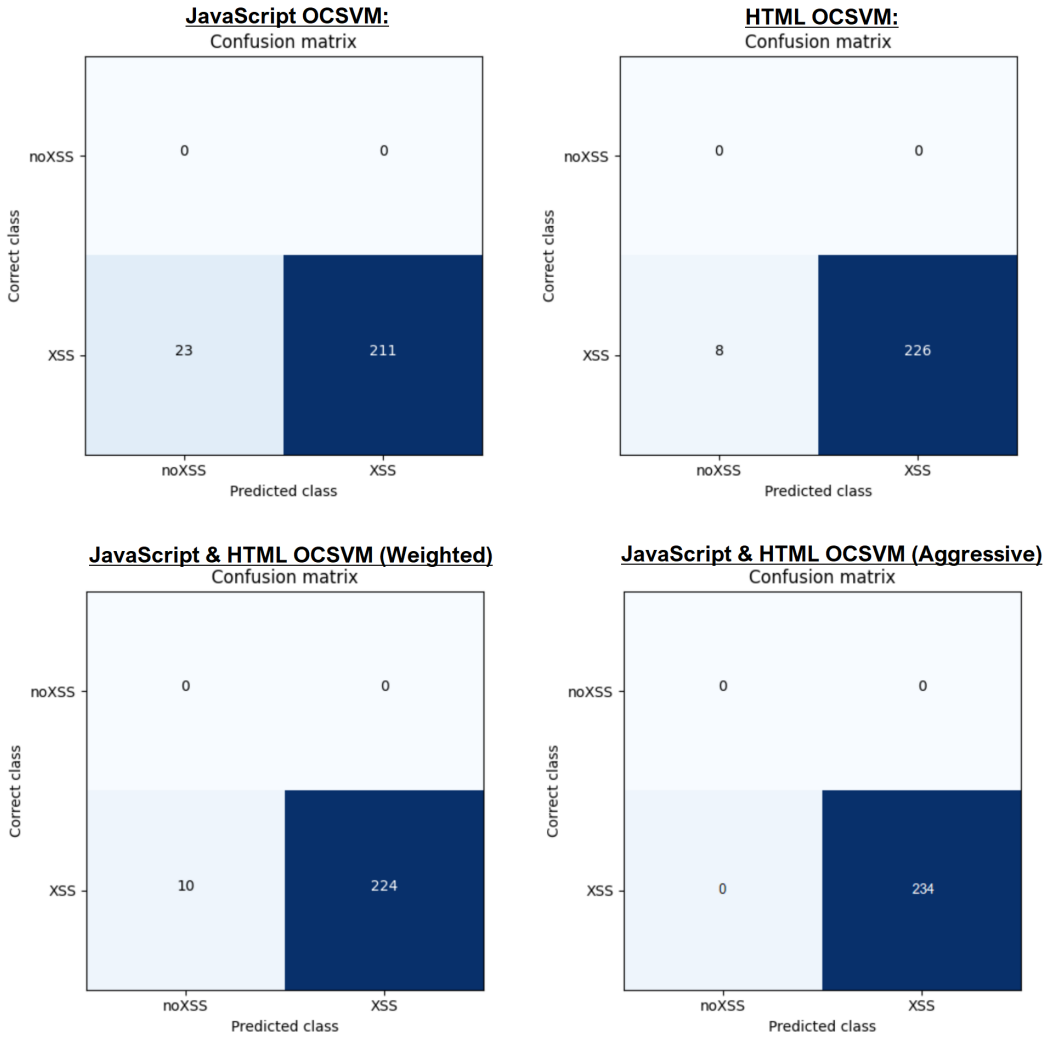

Figure 3.4: Confusion Matrix against Mithril test set.

Chapter 4

Conclusions and future developments

In this thesis, we embarked on a journey to design and develop an innovative XSS attack classifier, striving to enhance the robustness and adaptability of web security systems. Through a comprehensive exploration of unsupervised learning techniques and pioneering strategies, we have achieved significant milestones in the field of web application security.

Our proposed solution is centered around an unsupervised classifier, leveraging the power of One-Class Support Vector Machines (One-Class SVM) to decipher patterns within unlabeled data. This approach involved several critical steps, including an effective pre-processing pipeline to cleanse payloads, the extraction of essential features, and the training of the model. The chosen model, the One-Class SVM, is renowned for its anomaly detection capabilities, serving as a sentinel against XSS attacks by discerning malicious payloads from benign ones.

A key innovation in our work is the introduction of dual One-Class SVM models. These models specialize in detecting XSS-related HTML and JavaScript content, providing a more nuanced and accurate defense mechanism. Moreover, we have implemented a model selection approach, allowing adaptability based on specific threat scenarios.

This selection optimizes the accuracy and effectiveness of the classifier.
To address ambiguity in model predictions, we have proposed two evaluation strategies, prioritizing the model with a greater payload-to-hyperplane distance or considering a payload as malicious if either model produces a positive classification. This approach empowers users to tailor the severity of assessments, ensuring flexibility in security responses.
This research also introduces an advanced pre-processing pipeline, meticulously designed to comprehensively cleanse input payloads. This pipeline includes innovative techniques such as enhanced generalization, refining the removal of extraneous elements, and optimizing data quality. These enhancements collectively elevate the classifier's efficacy in identifying XSS attacks.
In this scenario several possible future developments are possible:

1. *Decoding:* Include other decoding techniques so that even less frequent encoding can be correctly detected.

2. *Improved Real-time Performance:* While the current implementation serves as an offline classifier, our ultimate goal is to optimize its performance to operate in real-time. We recognize the importance of reducing latency to the order of milliseconds, enabling its integration into real-time security systems. Future efforts will focus on fine-tuning the system for swift and efficient real-time XSS attack classification.

3. *Incorporation of Additional Data Sources:* To further enrich the model's feature set, we foresee the integration of additional data sources beyond HTTP requests. Exploring user behavior, session information, and device characteristics could provide invaluable insights. These supplementary cues may enhance the classifier's accuracy in identifying XSS attacks by capturing contextual and behavioral patterns.

4. *Continuous monitoring and updates:* As attackers' tactics are constantly evolving, it is essential to keep the model up-to-date and implement constant monitoring to adapt to new threats.Future endeavors will center on maintaining the model's relevance and effectiveness. This involves ongoing monitoring of emerging attack tactics and the swift incorporation of updates to adapt to new threats.

By addressing the challenges mentioned above, we aspire to develop a cutting-edge system capable of robust real-time protection against XSS attacks while remaining adaptable to the ever-changing landscape of cybersecurity threats.

References

[1] E. Kirda, C. Krügel, G. Vigna, and N. Jovanovic, "Noxes: A client-side solution for mitigating cross-site scripting attacks," 04 2006.

[2] M. Johns, "Sessionsafe: Implementing xss immune session handling," 09 2006.

[3] M. Johns, B. Engelmann, and J. Posegga, "Xssds: Server-side detection of cross-site scripting attacks," 12 2008.

[4] P. Bisht and V. Venkatakrishnan, "Xss-guard: Precise dynamic prevention of cross-site scripting attacks," 07 2008.

[5] P. Wurzinger, C. Platzer, C. Ludl, E. Kirda, and C. Kruegel, "Swap: Mitigating xss attacks using a reverse proxy," 05 2009.

[6] H. Shahriar and M. Zulkernine, "S2xs2: A server side approach to automatically detect xss attacks," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 2011.

[7] M. Gundy and H. Chen, "Noncespaces: Using randomization to defeat cross-site scripting attacks," *Computers Security*, 06 2012.

[8] F. Mereani and J. Howe, *Detecting Cross-Site Scripting Attacks Using Machine Learning.* 01 2018.

[9] Y. Zhou and P. Wang, "An ensemble learning approach for xss attack detection with domain knowledge and threat intelligence," *Computers Security*, 2019.

[10] S. Rathore, P. Sharma, and J. Park, "Xssclassifier: An efficient xss attack detection approach based on machine learning classifier on snss," *Journal of Information Processing Systems*, 01 2017.

[11] F. Mokbal, W. Dan, A. Imran, L. Jiuchuan, F. Akhtar, and W. Xiaoxi, "Mlpxss: An integrated xss-based attack detection scheme in web applications using multilayer perceptron technique," *IEEE Access*, 07 2019.

[12] Y. Fang, Y. Li, L. Liu, and C. Huang, "Deepxss: Cross site scripting detection based on deep learning," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, Association for Computing Machinery, 2018.