National College of Ireland

Implementing Adaptive Rate Limiting and Honeytokens for Enhanced REST API Security in JavaScript Web Applications.

MSc Research Project
M.Sc. Cybersecurity

Chinelo Lauren Nwobbi
Student ID: x23333057

School of Computing
National College of Ireland

Supervisor: Khadija Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Chinelo Lauren Nwobbi
Student ID:	x23333057
Programme:	M.Sc. Cybersecurity **Year:** 2025
Module:	Practicum
Supervisor:	Khadija Hafeez
Submission Due Date:	September 15th, 2025
Project Title:	Implementing Adaptive Rate Limiting and Honeytokens for Enhanced REST API Security in JavaScript Web Applications.
Word Count:	9526 **Page Count** 23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Implementing Adaptive Rate Limiting and Honeytokens for Enhanced REST API Security in JavaScript Web Applications.

Chinelo Lauren Nwobbi
x23333057

Abstract

Application Programming Interfaces used for transmitting information remain vulnerable to various attack vectors. Current mitigation strategies involve employing API best practices such as static rate limiting, encryption and input validation, using machine learning algorithms to detect API misuse and other security frameworks. While these strategies support security, issues like computational complexity, costs and inflexibility hinder progress and security effectiveness. This paper analyses adaptive rate limiting and honeytoken mechanisms as a tool to enhance REST API security by proposing a model that integrates these mechanisms, analysing request behaviour when handling traffic. The proposed model obtained an accuracy rate of 86.36% when detecting bots and effectively applied rate limiting for identified bot and human threat actor traffic. This paper will support developers when building REST APIs with a ready tool for enhancing rate-limiting capabilities and strengthening API endpoints.

1 Introduction

Application Programming Interfaces (API) are an integral part of development of various technological components and are used extensively across various domains such as web and mobile applications, e-commerce, healthcare, machine learning (ML), finance and banking, and the internet of things (IoT). APIs are primarily used to enable communication of functions, data, and features across different software and hardware systems. In web applications, an API works to query a database by either retrieving or inserting data. By doing this, it fulfils its duty of data transference. In the IoT domain, APIs can be used in AgriTech industries to transmit data like soil characteristics or temperature levels derived from their sensors and are an integral part of daily operations. They can be used as a third-party service to add features that display customer reviews in an e-commerce site. The creation and use of APIs goes back to 1968 in the context of interactive computer graphics (Lamothe, Guéhéneuc and Shang, 2022) and since then has evolved greatly, where many industries have adopted and integrated them into the normal functioning of their applications. There are multiple kinds of APIs in play, and they can be categorized based on different criteria like accessibility which has APIs like Open APIs, Internal APIs, and Partner APIs. Another important categorization is based on protocol or architecture where APIs like Representational State Transfer (REST) APIs and Simple Object Access Protocol (SOAP) APIs are found, REST APIs are the most common and are the focus of this study whilst SOAP APIs use XML based protocols and are stricter with built-in security features. REST APIs define a way for a client component like a web browser to interact with a server; these interactions include sending requests and receiving responses in the form of

JavaScript Object Notation (JSON) or Extensible Markup language XML. REST can be defined by its key concept of statelessness, use of client server architecture, and uniform interface (Goodwin, 2024).

APIs are now deeply integrated with multiple components of everyday technology; these APIs widen the attack surface area by adding a new endpoint for possible exploitation. A compromise of this technology can lead to a loss of availability to different technology, exposure of sensitive data to threat actors and potential tampering of data being transmitted via APIs. Common security threats include information leakage, unauthorised access and injection attacks (Zhao, 2024). In 2024, 49 million Dell customers' data was stolen in a data breach resulting from a vulnerable API. This was done through a portal used to find order information. The threat actor identified as Menelik gained access to the portal and was able to write a program to scrape 49 million user data records using 5000 requests per minute due to missing rate limiting (Abrams, 2024). Another case of API abuse is of the popular social media company X formerly known as Twitter. Threat actors exploited a vulnerability that allowed unauthenticated users to enter phone numbers and email addresses into a Twitter API and retrieve the inputted data's Twitter account. This led to threat actors stealing user data, with 5.4 million records of user data being put up for sale in a hacker forum for 30,000 dollars (Abrams, 2022). These case studies show how important securing APIs are, as it leads to major financial and reputational losses for organizations and great psychological impact on affected customers. These incidents although having been around for a while are still on the rise, a 2024 study from Akamai Technologies found 84% of its responders having suffered an API security incident in the past year with an average cost of 474,000 euro to recover from this incident across America, The United Kingdom and Germany (Annie Brunholzl, 2024).

To combat the confidentiality, integrity and availability being compromised, the existing solutions include using detective methods such as logging, monitoring, and anomaly detection to identify threats early on (Nartovich, 2024). Another solution is employing preventative methods that follow security best practices during development such as implementing rate limiting to reduce the possibility of denial-of-service attacks, encrypting data to reduce the compromise of confidentiality and integrity, and the use of secure authentication and authorization practices to uphold strict confidentiality. This thesis proposes the integration of preventative and deceptive security measures into REST API development to enhance the principle of security by default in web applications. It focuses on implementing adaptive rate limiting (a method of rate limiting that involves changing limits based on defined metrics such as CPU usage, memory consumption or request behaviour (Deepak, 2024)) to dynamically control traffic and prevent denial-of-service (DoS) attacks and brute forcing, and the use of honeytokens (API endpoints solely designed to attract threat actors and alert usage (Vaideeswaran, no date)) to detect, monitor and limit unauthorized access attempts. These methods aim to strengthen the confidentiality, integrity, and availability of REST APIs by incorporating proactive and deceptive security techniques into the software development lifecycle.

1.1 Research Question and Objectives

The research question is '*How effective is implementing adaptive rate limiting and honeytoken mechanisms in a secure design framework to mitigate security vulnerabilities in REST APIs for JavaScript web applications*?'

The research objectives are:

1. Investigate the current REST API security best practices and their effectiveness.
2. Design a secure API and integrate proposed adaptive rate limiting and honeytoken features.
3. Implement a secure API prototype with adaptive rate limiting and honeytokens.
4. Evaluate prototype against baseline API models.

1.2 Contributions

The contributions this paper offers are the proposal of adaptive rate limiting and the use of honeytoken mechanisms in increasing the security posture of web applications that rely on REST APIs. Another contribution is in its proposal of practical defensive mechanisms to proactively mitigate rising API security threats. The study also adopts a shift left approach to secure APIs from the initial stages of development. A study published by Lamothe et al. showed that out of 369 surveyed papers on APIs, less than 19 papers were related to API security, highlighting a gap in API security research (Lamothe, Guéhéneuc and Shang, 2022), this research contributes to the gap by integrating novel preventative and deceptive techniques into API security development. Finally, the study offers a developer friendly approach that integrates proposed techniques into the development process which developers can adapt into their own processes.

1.3 Limitations

There are various limitations to this study as the study will be evaluated using a prototype and not production level APIs used in large-scale enterprise environments with real world usage patterns and threats, another limitation is that the study's technology stack is limited to JavaScript, Node.js and Express.js, using a different stack might vary its effectiveness and may lead to integration challenges, and finally the use of the proposed security measures i.e. adaptive rate limiting and honeytokens may lead to false positives and false negatives which may inadvertently block users from accessing the web application or give access to bots who slipped through filters. Finally, test scenarios were based on simulated bot traffic which limits generalisability of results and obscures how model might behave in real world scenarios.

This study begins by introducing APIs to readers, giving a brief overview and highlighting the importance of implementing security measures, it analyses current literature on the topic in the **Related Work** section and goes on to the **Research Methodology, Design Specification and Implementation** sections that itemizes how the project was implemented, the outputs of the study and the process flow of the model, the **Evaluation** section explains how the project was evaluated based on different evaluation criteria. It ends with a conclusion section that suggests future work in the domain.

2 Related Work

This section gives an overview of existing literature and methods for securing APIs, using honey techniques, rate limiting, and bot detection.

2.1 Securing APIs

Securing APIs is a critical need for every organization. Ferworn and Sharieh introduced chaos engineering as a method for building confidence in APIs and observing how threats are withstood. The study discusses means of protecting REST APIs and API specific vulnerabilities such as Distributed Denial of Service (DDoS) attacks (Sharieh and Ferworn, 2021). It highlights ways in which bot detection can be used to mitigate DDoS attacks and has a real-world focus. The study is widely theoretical and has implementation gaps as no practical framework for chaos engineering or automated attack detection was developed, and tools like Apigee were mentioned without any evaluation metrics attached. This paper gives an introductory idea of using bot detection to mitigate vulnerabilities on web applications.

With machine learning and artificial intelligence on the rise, there has been an uptick in research related to implementing machine learning in API security practices. A paper by Joel Paul discusses integrating machine learning into API security frameworks and its potential to advance threat detection and response activities (Paul, 2024). This paper highlights the effectiveness of machine learning for anomaly detection in API use and mentions how to mitigate false positives and false negatives as well as how to integrate it into existing solutions. Although these are great advancements to securing APIs there are still several limitations this comes with. The effectiveness of these models is dependent of the quality of the data; bad data can introduce bias in results as well as increase rate of false positive and negatives making it inadequate of integrating in real world scenarios. Using ML models adds another point of attack for hackers who can exploit vulnerabilities in the model itself like corrupting data. This study takes a reactive approach to threats by reacting to anomalies as they occur whereas this thesis aims to apply a proactive approach by not only detecting odd behaviour but also actively working to mitigate attacks.

Adopting security best practices when developing APIs is an important baseline to start from to prevent the possibility of attacks like cross site scripting, broken authentication, and denial of service. These best practices include rate limiting, encryption, secure authentication and authorisation and input validation. A journal article by Mayank Hindka gives an in-depth, comprehensive guide to developers when implementing these practices whilst promoting a shift left approach similar to this thesis (Hindka, 2024). Whilst this article is a strong tool for securing APIs, it has no practical framework or evaluation metrics to compare its effectiveness.

Kornienko et al. discusses developing a secure REST API using Flask, a popular python framework. The study gives detailed insights into APIs, that is their importance, benefits and limitations and comparatively analyses various architectural styles helping readers select which is best suited to a project (Kornienko *et al.*, 2021). The study develops a prototype using best practices but does not provide empirical data on the prototype's efficiency or security. Similar to this thesis, this paper builds on security best practices but has not taken a further step to defend against API specific vulnerabilities on a granular level.

Hussain et al. takes another approach in securing APIs using an intelligent service mesh. This framework combines mTLS, OAuth and JWT tokens for multi-layered authentication and authorisation adopting a defence-in-depth strategy. It implements zero-trust principles to minimise the attack surface area initiated from its fine-grained access control (Hussain *et al.*, 2019). This paper is strongly limited to authentication and authorisation practices and not

focused on other API specific vulnerabilities with limited protection against API specific attacks, it also has a high dependence on Istio's control plane where a failure in that infrastructure can lead to an overall failure to the application.

Research from Nguyen and Baker developed a proof of concept (POC) to analyse effectiveness of applying spring security framework and OAuth2 on microservice architecture API. Experiments were carried out on an inventory management system with multiple test cases using both unit and manual testing. These cases included Cross Site Request Forgery, Cross Site Scripting and Brute forcing. These test cases all passed by 100% and were successful, providing empirical evidence of the framework's effectiveness (Nguyen and Baker, 2019). While this research had a high success rate, it was limited to very specific technologies that may not easily be applied to other technology stack. It also lacks a comparative analysis with other frameworks to prove efficiency of framework. While this thesis also uses a specific technology stack, it focuses on frameworks that can easily be translated across other variants of programming languages and frameworks

Hu et al. proposes a secure application framework, SEAPP that aims to manage application permissions through access control and encrypt API calls to guard against malicious actors in Software-Defined Networking (SDN)-enabled cloud environments. It proposes a two-component framework that incorporates static analysis, dynamic authorisation, and encryption for securing REST APIs. It includes extensive experiments that demonstrates the effectiveness of the framework (Hu *et al.*, 2021). This paper is limited to SDN-enabled cloud environments limiting its use in other applications, it also relies on static permission checks limiting its ability to dynamically respond to unexpected changes in an application. Furthermore, although this is a technical solution it does not provide a structured approach other developers can follow and integrate into their practices.

Hussain et al. discusses enterprise API security. They examine how machine learning solutions for threat detection and prediction can be applied to it, and the limitations of applying these ML solutions to API security landscape in terms of GDPR compliance (Hussain *et al.*, 2020). This paper suggests that adding ML solutions to this type of security introduces issues with GDPR compliance, this is important to note as the rules applied for adaptive rate limiting could have been based of suggestions of ML models which evidently may have introduced non-compliance, because of this, the paper is using bot detection/user behaviour to determine rules.

2.2 Cyber deception techniques: *Honeypots, Honeysites and Honeytokens*

A key cyber deception technique is the use of honeytokens. These are fake data entries, URLs, credentials etc that are used to lure threat actors. When an action is taken using these tokens system, administrators are notified, or a reactive event is automatically executed. Research from Prabhaker et al. developed a framework for generating and deploying honeytokens in relational databases using hierarchical modelling algorithm (HMA) synthesizers. This framework is an effective way to indicate early breach detection as using one indicates unauthorised access to a database and triggers an alert to system administrators. The use of the HMA as well as multiple tables full of honeytokens increases the difficulty when discerning between real and fake data, making deception of threat actors easier (Prabhaker, Bopche and Arock, 2024). Authors carried out experiments that performed well and will be

easy to adapt to real world scenarios, in spite of that this approach is limited to data breach specific threats and does not address other growing threats like brute forcing or DdoS. This framework is mainly detective unlike this thesis where a trigger of the honeytoken leads to reduced rate limiting or blocked access to the application.

(Li *et al.*, 2021) analyses and characterises malicious bot behaviour through honeysites. This study collected 26.4 million requests on honeysites from 287 thousand unique IP addresses increasing validity of work as findings are based on real world activities, it uses numerous fingerprinting techniques to identify bots such as browser fingerprinting, behaviour fingerprinting, and TLS fingerprinting. This study gives important insights into malicious bot behaviour as well as establishing a baseline for identifying majority of these bots. While this study is highly valuable in bot detection and is a clear baseline on how to detect bots and classify them, it focuses on observation rather than mitigation, as these classifications are done after bots have already attempted to deploy attacks on the honeysite. It lacks actionable recommendations and steps to real-time bot blocking, making the study reactive and not proactive. This thesis incorporates bot detection into adaptive rate limiting by dynamically reducing the rate at which identified bots or threat actors can access a web application.

The advancement of the threat landscape needs proactive approaches to growing threat actors; Rani et al. provides a step-by-step process of deploying honeypots to lure potential threat actors to capture their IP address and geographical location leveraging the use of canary tokens. This tool also aids in real time monitoring of deployed tokens with an alert being sent out when triggered (Rani *et al.*, 2024). While this is a strong step in detecting threat actors, advanced attackers may evade detection because of high experience with honeytraps. This approach also relies heavily on attackers triggering a token, passive attackers may remain undetected. Unlike this thesis, this approach lacks adaptive mechanisms like dynamic rate limiting or automated IP blocking for detected threat actors.

On the application layer, honeytokens were injected into application traffic using the Netfilter queue API, python and scapy. A POC was created, highlighting HTTP packets being intercepted and manipulated to present a fake login page used to check for honey credentials (Reti, Angeli and Schotten, 2023). While this introduces proactive threat detection it also introduced an upward jump in latency from test scenario 1 to 3 of 737.5% due to python and scapy processing which may lead to packet loss or session timeout in TCP connections.

2.3 Rate limiting

Rate limiting is a core practice in this thesis, it controls the rate at which clients can make requests within a specified time, Serbout et al. provides a comprehensive guide for developers to follow by identifying trade-offs across different rate limiting patterns. Aiding developers' decision-making process when implementing rate limiting (Serbout *et al.*, 2023a). This paper extensively covers adoption of various rate limit methods such as client level and resource level rate limiting that mitigates against DdoS and brute force attacks. It also proposes a method of dynamic rate limiting by adjusting limits based on system load. While this paper offers substantial research on rate limiting, there is a lack of discussion on threat detection and most patterns defined rely heavily on static rules, making mechanisms weak against advanced threats.

Research from Sivaraman on the other hand takes a more proactive approach by proposing a reinforcement learning model for adaptive rate limiting. This model adjusts limits in real time based on user behaviour with experiments being carried out on an e-commerce API. It exhibits a 32% reduction in false positives (legitimate users blocked) and 25% lower false negatives (missed abusive traffic) compared to static rate limiting and addresses credential stuffing, scraping, and DDoS attacks by adapting to evolving abuse trends (Sivaraman, 2023). Regardless of the models noteworthy results, there still exists limitations such as high computational power, over dependence on quality training data as poor data can lead to bias and poor decisions, unfit for zero-day attacks as data in that criteria is not available till an attack occurs or vulnerability is found and the black box nature of machine learning models introduces a new constraint to audit checks and interpretations by security teams.

(Sharieh and Ferworn, 2021) give actionable insights onto chaos engineering. This approach remains theoretical without measurable outcomes. (Paul, 2024) takes an ML approach into securing APIs with limitations relating to over reliance on quality of data which leads to bias. These approaches support detection component of this study but highlight need for new and improved approaches. Several works emphasise the need for best practices in securing APIs such as (Kornienko *et al.*, 2021) and (Hindka, 2024). While these give a structured and practical steps to best practices, it borders on prevention and does not take a further step to mitigate dynamic attacks. (Nguyen and Baker, 2019) provides strong empirical results using OAuth2 and Spring but remain limited to their technology stack.

On deception (Li *et al.*, 2021) and (Prabhaker, Bopche and Arock, 2024) offer favourable honeytoken and honeysite mechanisms but largely focus on detection without action. (Reti, Angeli and Schotten, 2023) introduces proactive deception via manipulated traffic but at the cost of performance. These papers inform the use of honeytokens in this thesis where detection triggers adaptive rules.

Rate limiting is explored with papers from (Serbout *et al.*, 2023b) which outline rate limiting strategies useful for basic protection but inadequate for evolving threats. (Sivaraman, 2023) introduces ML-based adaptive rate limiting that reduces false positives and negatives but is resource intensive and unfit for audits due to black box nature. This study proposes behaviour informed rate limiting to enable adaptability and ensures transparency. Ultimately existing studies contribute greatly to API security. Most are passive, static and narrowly scoped. This thesis addresses these gaps by integrating bot detection, honeytokens and adaptive rate limiting into a security framework for modern web applications.

Study	Focus	Methodology	Limitations
Sharieh and Ferworn (2021)	Chaos engineering and bot detection	Theoretical discussion of chaos engineering and bot detection	No implementation therefore lacks evaluation or measurable outcomes
Paul (2024)	ML for API anomaly detection	Highlights effectiveness of machine learning in API security for anomaly detection.	Mostly reactive, data-dependent which may lead to bias and privacy risks
Hindka (2024)	API security best practices	Developer-focused guide on security best practices such as encryption, input validation and rate limiting	Lacks practical validation and evaluation metrics. Focuses mostly on prevention

Kornienko et al. (2021)	Secure API with Flask	Prototype establishing security best practices in REST APIs: It details a real-world prototype using Flask, Swagger, and PostgreSQL, offering actionable insights for developers.	Lack of Empirical Data: While the theoretical and practical aspects are well-covered, there is no performance metrics or empirical validation of the proposed solution's security or efficiency.
Hussain et al. (2019)	Service mesh	Discusses a framework designed to enhance API security and management within service mesh architectures, Combining mTLS, OAuth, and JWT tokens for multi-layered authentication and authorization.	Limited protection against specific API attacks and over dependence on Istio
Nguyen and Baker (2019)	Spring Security in microservices	Examines the possibility of applying Spring Security Framework and OAuth2 to secure microservice APIs examines the possibility of applying Spring Security Framework and OAuth2 to secure microservice APIs which are built on top of Spring Framework. By developing a Proof of Concept (POC) of an Inventory Management System using MSA on top of Spring Framework, Spring Security Framework and OAuth2	The findings are tied to Spring Framework and OAuth2, limiting their applicability to other technology stacks or security protocols. Lack of Comparative Analysis: The paper does not compare Spring Security and OAuth2 with alternative security frameworks or standards, which could have provided a broader perspective."
Hu et al. (2021)	Securing APIs in SDN cloud	The paper addresses security challenges in Software-Defined Networking (SDN)-enabled cloud environments, which is a niche but critical area in modern network security. It proposes a two-component framework that integrates static analysis, dynamic authorization, and encryption for securing REST APIs.	Limited to SDN; static controls, SEAPP is a technical solution but does not provide practical guidelines for developers to implement security during API design.
Hussain et al. (2020)	ML and GDPR compliance	Deep dive into GDPR compliance and its impact on ML-driven API security, addressing legal/ethical constraints.	Explores machine learning for anomaly detection and threat prediction in APIs.
Prabhaker et al. (2024)	Honeytokens in databases	Offers framework for generating and deploying honeytokens in relational databases that actively monitor sensitive records and quickly detect data breaches and their misuse.	Narrow focus in deploying honeytokens in Database, passive detection and monitoring
Li et al. (2021)	Bot analysis via honeysites	This study gives important insights into malicious bot behaviour as well as establishing a baseline for identifying majority of these bots. It identifies methods for bot detection such as TLS and browser fingerprinting	Focuses on observation rather than mitigation, does not propose defense like adaptive rate limiting or honey tokens. Lacks actionable recommendations for real-time bot blocking.
Rani et al. (2024)	Honeytokens with canary tokens	Uses honeypots as decoys to lure attackers, capturing their IP addresses and geographical locations via Canary Tokens. This aligns with modern deception-based cybersecurity strategies. Provides a step-by-step deployment guide	Detection-only, passive attackers remain undetected. The study is also widely theoretical as no prototype was developed.

		using Kali Linux, making it accessible for security practitioners.	
Reti et al. (2023)	Application-layer honeytokens	Modifies live traffic to inject decoys without changing backend servers. Uses Scapy for packet manipulation, enabling complex deception. Lures attackers with high-value decoys, triggering alerts upon interaction. Supports traditional defense by adding active deception to passive security mechanisms.	High latency and performance issues, Potential for false positives if legitimate users stumble on decoys.
Serbout et al. (2023a)	Static rate limiting	Identifies patterns covering the API Rate Limit pattern adoption starting from its documentation to its implementation. Highlights the trade-offs associated with different patterns and offer guidance to developers in making informed decisions when choosing the most suitable Rate Limit method, scope, and granularity for their service	Static Rules: Most patterns rely on predefined thresholds, making them ineffective against low-and-slow attacks or credential stuffing. Weak Against Advanced Attacks
Sivaraman (2023)	Adaptive rate limiting with Reinforcement Learning	Proposes a Reinforcement Learning (RL)-based adaptive rate-limiting model that dynamically adjusts request thresholds in real-time, unlike static rate-limiting methods. Effectively handling traffic patterns by learning from historical and real-time data. Demonstrates a 32% reduction in false positives (legitimate users blocked) and 25% lower false negatives (missed abusive traffic) compared to static rate limiting.	The RL model requires significant real-time computation, Performance hinges on high-quality, labelled data (e.g., historical abuse patterns). Poor data can lead to suboptimal policies or adversarial exploitation. Struggles with zero-day attacks or novel abuse patterns absent from training data, Decisions are hard to interpret for security teams, complicating audits or compliance checks.

Table 1: Summary of Related Work

3 Research Methodology

This research aims to evaluate the effectiveness of adaptive rate limiting and honeytoken mechanisms in enhancing API security. A quantitative experimental research design was employed to test the implementation of the proposed framework in a simulated environment. This approach was taken to allow measurable comparisons of model performance against baseline models defined by security best practices itemised by (Hindka, 2024). These included implementing best practices like encryption, static rate limiting, input validation, JWT authentication and authorisation. This section includes research procedures, techniques and equipment used, subsequently details on the steps involved in research is outlined from data collection to data analysis. This research was conducted using a VSCode Integrated Development Environment (IDE) with JavaScript used as its main language for both front and backend systems, and tests across multiple endpoints were conducted using Postman's Functional tests and Performance test with varying parameters set across each test case. Further tests were conducted using OWASP's ZAP vulnerability scanner. The applications spider tool and active scan was employed to analyse the API, and the Fuzzing tool was used to deploy Fuzz tests which were conducted on the base API endpoint with a custom user agent payload

set as the fuzzed data. This enabled performance and detection evaluation in real time. The metrics collected include request rate, error rate, response time and detection count.

3.1 Data Collection

This study uses two methods to determine if adaptive rate limiting rules should be employed on an IP address which are bot detection and honeytoken triggers. The bot detection technique was based on bot behaviour like request rate and user agent string. This technique was chosen based on the paper by (Li *et al.*, 2021). An open-source dataset provided by (DataDome, 2025) of real world known malicious or bot related user agents was integrated into the botDetectionMiddleware.js to automatically detect potential bots based on their user-agent string and log information like IP address, detection count, user-agent, description and a timestamp to the denylist table on the database. This dataset allows this model to have a more real-world approach and aids in improving detection of automated traffic.

Additional data was collected for this study from system logs and test outputs obtained during experimental testing of the Adaptive Rate Limiting (ARL) model. The experimental testing phase included deploying various test types on API endpoints using tools like Postman and ZAP. Logs captured honeytoken triggers, rate limit status, timestamps and IP addresses and request headers. Test outputs captured data like error rate, response time and request rate.

3.2 Model Development/Research Procedure

This section explains steps taken to integrate adaptive rate limiting practices and honeytoken mechanisms in REST APIs. The model is designed to detect suspicious bot and human behaviour through user agent string in request headers and bot behaviour, and honeytokens respectively. The model then deters malicious actors by employing adaptive rate limiting rules.

3.2.1 Development of Web Application

A web application for a hospital was built. The application comprised of a login page for administrators and regular users, in which logged in admins could create new users and view all created users whereas other logged in users could view their profile. The structure of the application implies strong role-based access control methods as well as authentication via the login page. Other security measures were implemented on input fields like input validation to reduce possibility of SQL injection and other related attacks.

3.2.2 Development of Baseline Models

Baseline models were developed to meet evaluation criteria. Models are to be evaluated against performance accuracy, error rate and response times. Because of this, two models were developed. The first model were API endpoints with no security mechanisms, the second endpoint was developed with best practices like input validation, static rate limiting, authentication and authorisation following research papers from (Hindka, 2024) who highlights best practices and (Gowda and Gowda, 2024) who establish best practices for API security and scalability such as encryption via HTTPS, authentication and authorisation, and performance optimisation.

3.2.3 Development of Adaptive Rate Limiting Model

The ARL model integrates traditional API security best practices while introducing adaptive rate limiting and honeytoken mechanisms to enhance security. The model combines adaptive rate limiting logic to evaluate request patterns and adjust thresholds and honeytoken traps that serve as decoys used to not only lure threat actors, but to also serve as another layer for bot detection. This model includes a bot detection and adaptive rate limiting middleware that work together to proactively detect and limit request of bots and threat actors. The development of this model adapts steps defined in research from (Rani *et al.*, 2024), where a honeytoken is designed and deployed, continuously monitors interactions and capture details of potential threat actors.

3.3 Data Analysis

This section involved analysing data obtained during the testing phase of the research. During the testing phase, a white box approach was taken as tester had access to full codebase which was essential to target specific weaknesses and components (Kajavalta, 2022) multiple test cases were established with manual and automated tests which enabled gathering different types of data. The primary goal of analysis was to determine how effective the proposed model was across different evaluation criteria.

Initial performance tests of the ARL model on the base API endpoint obtained an error rate of between 0% and 55.86%. With the first test obtaining an error rate of 54.26%, out of 2,162 requests from a bot with the same IP address the rate limiter only worked to limit 1,173 indicating that the ARL model was not working effectively. These outputs resulted in further finetuning of the ARL model, by developing a custom middleware integrated with Redis as opposed to using the express-rate-limit middleware offered by Express.js. With the use of the custom middleware error rate went up to 99.81% as out of 5,296 bot requests from the IP address the ARL model worked to limit 5,286 of those requests indicating the effectiveness of the rate limiter.

4 Design Specification

This section outlines the system architecture and technical design of the proposed model that builds upon the secure design by default security principle by integrating adaptive rate limiting and honeytoken mechanisms into API security strategies. The design of this model aims to detect suspicious and malicious actors and hinder them from the application focusing on OWASP API Security API4:2023 - Unrestricted Resource Consumption.

4.1 System Architecture

The design of the system follows a three-tier architecture, (IBM, 2021) a type of software application that divides an application into three tiers that is presentation, application and data tier.

1. Frontend (Presentation tier): This tier is the user interface of the application where a user interacts and communicates with the rest of the application and was previously

defined in Section 3.2.1, this tier was built using JavaScript, React.js provided by Vite and CSS.

2. Backend (Application tier): This tier is where requests from the frontend is processed using business logic. This is where API calls are defined and interacts with the data tier. This tier was where honeytoken APIs were set and bot detection and adaptive rate limiting middleware were defined. These elements were developed with Node.js, Express.js and JavaScript.
3. Database (Data tier): This tier is where all data is stored. In this case the database contained two separate tables a user table that stored user data such as email address, password, address phone number. The other table was the denylist table which contained details on potential threat actors such as IP address, user-agent tag, detection count, description and timestamp. During development two PostgreSQL relational databases were established. One that ran on the local machine using PgAdmin and the other used after deployment of presentation and application tier. This database is hosted on Neondb and is available online.

4.2 Bot detection Middleware

The bot detection middleware as previously mentioned is a critical tool in the application that works to identify bots using the user-agent string found in the request headers of a request and analyses request rate behaviour. Normal user-agent strings vary slightly from bot user-agent string sometimes having a bot name attached to it.

```
Mozilla/5.0 (Linux; Android 13; KINGKONG POWER Build/TP1A.220624.014; wv) AppleWebKit/537.36 (KHTML, l
ike Gecko) Version/4.0 Chrome/137.0.7151.115 Mobile Safari/537.36 Instagram 387.1.0.34.85 Android (33/
13; 480dpi; 1080x2172; CUBOT; KINGKONG POWER; KINGKONG_POWER; mt8788; pt_BR; 756711600; IABMV/1)
```

Figure 1: Example of bot user agent string gotten from (WhatIsMyBrowser.com, no date)

The middleware works to analyse the user agent string of all incoming requests against a predefined array of bot user-agent keywords. If a user-agent string contains a keyword in the predefined array the IP address, user-agent string and other related data is added to the denylist established in the data tier. This middleware contains two functions: a logBotToDatabase function that does the logging to the database using a query and a botDetectionMiddleware function that contains the predefined user-agent strings incoming requests are compared against. This is the main function that is exported and applied globally in the main backend file meaning this middleware is applied on all API endpoints.

4.3 Honeytoken APIs

These honey token modules were set in the application tier with endpoints such as *'/trap/bot'* and *'/trap/human'*. These endpoints were linked on the application layer as buttons, the bot endpoint was set as a hidden button to identify bots and web crawlers who scan websites and click all links and buttons both seen and unseen, the human endpoint was linked to a fake admin login form with an attractive name '*admin'*. Once any of these APIs are triggered, they are automatically logged into the denylist table along with its IP address, detection count (based on the number of times the button was clicked before rate limit is applied) and a description of bot or human depending on API that was triggered.

4.4 Adaptive Rate Limiting Middleware

The adaptive rate limiter heavily relies on the bot detection middleware and honeytoken APIs as rate limit rules are only applied on IP addresses already in the denylist. The middleware checks if an IP address is in the denylist. If the IP address is found in the denylist, it checks the description of the IP. If it is described as a human, a stricter limit of 1 request per hour is applied to that IP address but if the IP address is described as a bot, a less stringent allowance is given at 10 requests per hour.

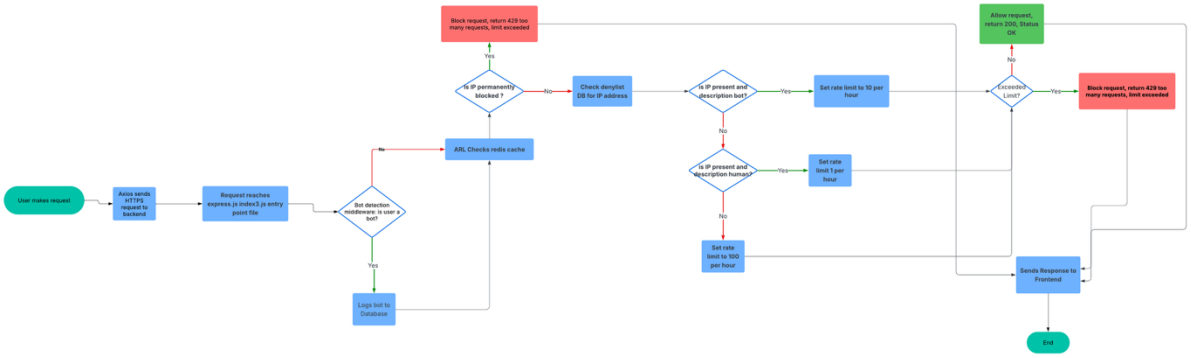

Figure 2: Flowchart depicting how ARL model handles requests.

5 Implementation

The implementation phase involved building the proposed model following the design specifications outlined in Section 4. The application was deployed across various hosting services such as Render, Vercel and Neon and evaluated using tools like Postman, OWASP's ZAP and Node.

5.1 Outputs

A simple hospital web application was built, in which system administrators can login, create new users and view current user data, and regular users can login and view their profile. This application could make requests to the backend who queried the database for data and responded to clients accordingly. The backend worked effectively to limit the rate of request based on identified suspicious behaviour. This resulted in fully functional APIs that effectively and dynamically limits requests based on client behaviour while logging that data to the Neon database. This application also produced API honeytokens that worked to lure potential threat actors and created another layer of detection for bots, these honeytokens log all attempted access to it.

This model offers a secure design framework for REST APIs that enables proactive threat detection through adaptive rate limiting mechanisms and honeytoken deployment and reduces risk of unauthorised access, resource exhaustion and API abuse.

5.2 Tools and Languages

- JavaScript: The application was primarily coded using JavaScript a common object-oriented programming language.
- React.js: This is an opensource library for user interface (frontend) development that allows the development of the user interface using components, the frontend application

was broken down into multiple components including Login.jsx, CreateUser.jsx, Home.jsx and App.jsx

- Node.js: This is a runtime environment used for server-side scripting; it allows developers create servers that can make calls to a database.
- Express.js: This is a Node.js framework designed for building web applications and APIs.

6 Evaluation

This section assesses the proposed model for enhancing API security, with a focus on its adaptive rate limiting mechanism, honeytoken deployment and bot detection middleware. The model is evaluated against three existing approaches: static rate limiting with baseline models established by the author, reinforcement learning-based adaptive rate limiting, and bot detection via honeysites and behavioural analysis. Testing rate limits involves checks to ensure an API can handle high volumes of requests and enforce rate limit rules effectively (Alharbi and Moulahi, 2023). Key evaluation criteria include performance accuracy, response time, flexibility, bot identification accuracy, deception effectiveness, implementation costs and computational accuracy.

6.1 Experiment / Case Study 1: Performance Accuracy

On performance accuracy, the performance of the model against various metrics such as error rate, false positive rate, false negative rate was evaluated. The results of these tests were compared against baseline models established with elements from best practice papers such as (Serbout *et al.*, 2023b), (Hindka, 2024) and results from (Sivaraman, 2023) paper which used reinforcement learning to execute adaptive rate limiting. The performance of the model was also evaluated using a confusion matrix revealing important insights into predictive capabilities. The model achieved an overall accuracy of 86.36%, correctly classifying 19 out of 22 requests from different IP addresses with a mix of user-agent strings.

6.1.1 Error rate

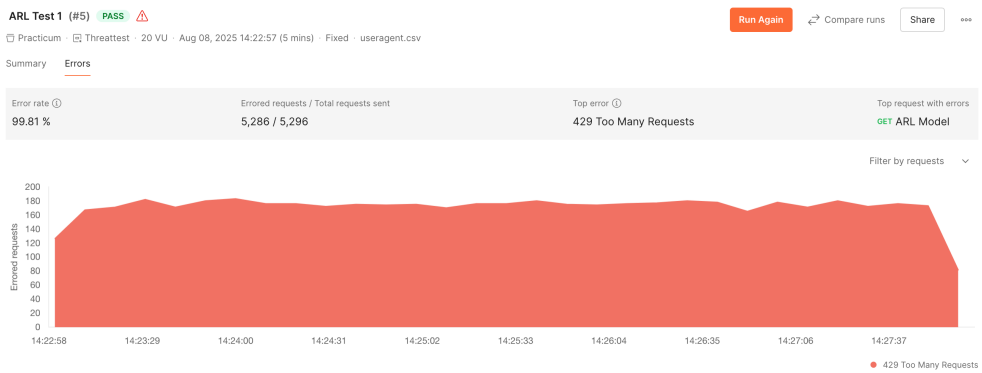

Figure 3: Results from ARL model showing error rate on Postman

The model achieved an error rate of 99.81% during its performance test on Postman test with a fixed load profile where 20 virtual users ran for 5 minutes executing requests sequentially. A total of 5,296 bot requests were sent from the same IP address and 5,286 of those requests were

errored increasing the error rate to 99.81%. This means the model effectively identified bot behaviour according to user agent string and applied rate limiting rules to it associated to bots.

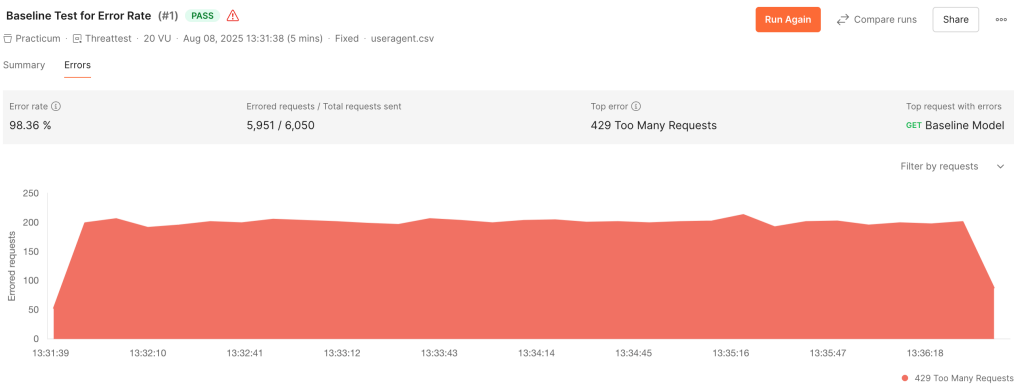

Figure 4: Results from baseline model showing error rate on Postman

On the other hand, the baseline model with static rate limiting rules implemented obtained an error rate of 98.36%, with the same load profile 6,050 bot requests were sent from the same IP address and 5,951 of those requests were errored. Although the model did not identify bot behaviour it still effectively applied rate limiting rules on users after it had exceeded 100 requests in an hour.

6.1.2 False Positive and False Negative rate

22 simulated IP addresses sent bot and legitimate user request to the API to evaluate how effective the API is in detecting bots and adequately allowing legit traffic. 19 out of 22 instances were correctly classified with all negative cases predicted correctly, however the model misclassified some cases increasing the false positive rate to 33.33%, allowing some bot traffic to pass even after hitting rate limit of 10 requests per hour. This means that although the model is highly reliable, it may fail to identify true positive instances i.e. true bot traffic, therefore future work would need to work to close this gap and reduce false negative rate by enhancing detection techniques. The Mean Absolute Percentage Error (MAPE) and Mean Percentage Error (MPE) were calculated with tools from PerMetrics. MAPE calculates the percentage difference between predicted and actual values with a MAPE of less than 20% regarded as a good predictive model (Coralogix, no date). In this case, the model obtained a score of 13.63%; this indicates the model performs reasonably well when making predictions, implying that the model is 13.63% off from actual values when predicting. MPE on the other hand was -13.63% this indicates an underestimation of denied requests, potentially allowing malicious bots. This bias enforces the need for further tuning to improve security by reducing false negatives.

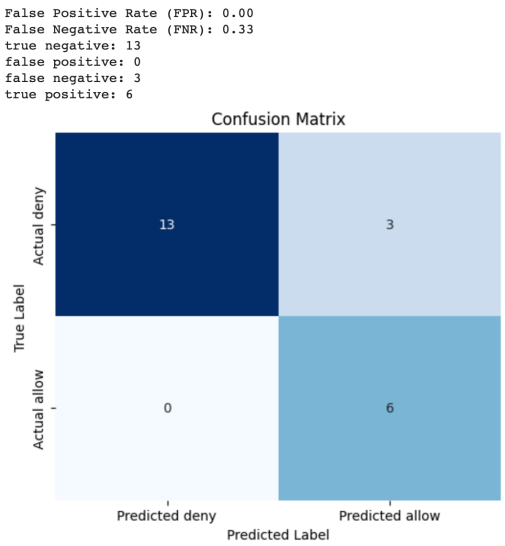

Figure 5: Confusion Matrix depicting actual classifed traffic as bots vs predicted.

6.2 Experiment / Case Study 2: Response Time

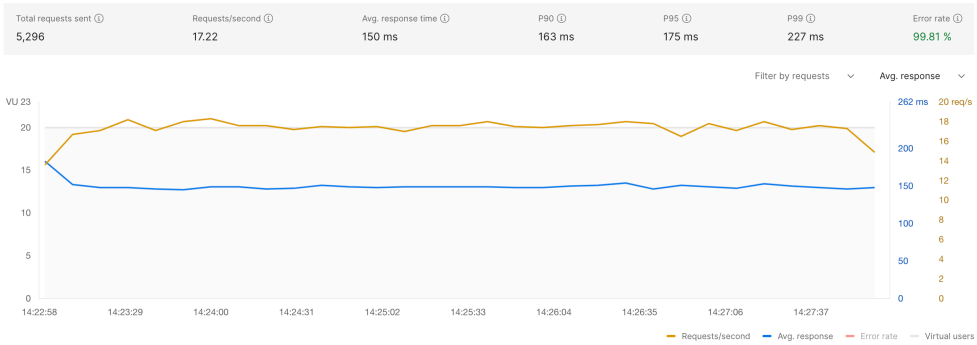

Figure 6: Graph depicting average response time and request/second on ARL model performance test.

Fixed performance testing on ARL model revealed a mean response time of 150 ms over 5 minutes with 90% of requests hitting the endpoint with a response time below 163 ms and 99% of requests hitting the endpoint with a response time below 227 ms.

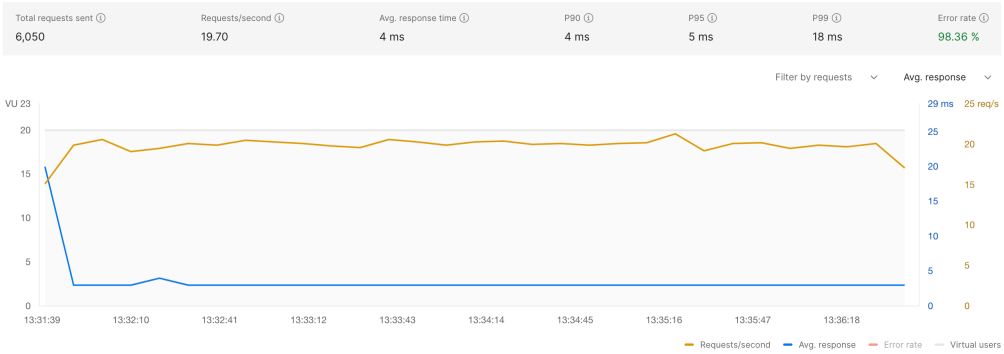

Figure 7: Graph depicting average response time and requests/second on baseline model performance test.

Similar performance tests were conducted on the baseline model which produced very different results. Requests had a mean response time of 4 ms with 90% of requests hitting the endpoint

with a response time below 4 ms and 99% of requests hitting the endpoint with a response time below 18 ms.

Fuzz tests were conducted to further analyse API behaviour of the ARL model using OWASP ZAP. Results obtained gave further insights into how requests were being responded to as well as requests round trip time (RTT). Figure 8 shows some requests going up to almost 700 ms.

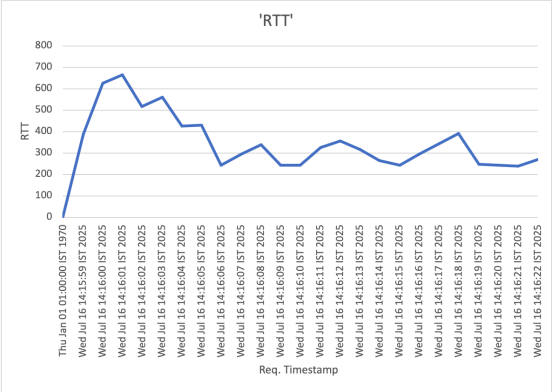

Figure 8: RTT across timestamps

Confidence intervals derived via Fieller's method were obtained using a GraphPad calculator to further evaluate these figures using mean response time, standard deviation and sample size. This test revealed the proposed model is 68 times slower than the baseline model. This time difference is most likely due to computational overhead associated with security measures.

6.3 Experiment / Case Study 3: Flexibility

This evaluation criteria analyses how models and existing studies react to changing user behaviour. In baseline models out of 5,962 bot requests from the same IP address none of these requests were flagged or reacted to until requests hit limit of 100 requests per hour. In research from (Rani *et al.*, 2024) and (Li *et al.*, 2021) models and frameworks proposed had a relatively passive approach to different user behaviour; only monitoring actions were taken on identified bots allowing them leeway to roam and further explore honey sites. In the ARL model identified bots access to sites were limited after requests exceed 10 in an hour, this allowed fine-grained control that reduces site disruption for legitimate users while hardening controls against suspicious activity, this is similar to research from (Sivaraman, 2023) which adjust thresholds based on ML model outputs.

6.4 Experiment / Case Study 4: Bot Identification Accuracy

While bot detection is not the focus of this paper, adaptive rate limiting rules are dependent on the ability of bot detection middleware to effectively identify bots and insert into denylist table, therefore evaluating the middleware's capabilities in detecting bots is paramount. The bot detection middleware currently hinges on three elements, the request headers user agent string, request rates and honeytokens deployed on the frontend. While this study uses user agent strings from real world, data spoofing of user agent is prevalent because of this bot detection middleware may not effectively identify bots with new spoofed bot user agents and add them to denylist. The use of honeytokens adds another layer of defense. This tool has been implemented to further identify bots that have been able to slip through detection

from user agent string and request rate checks. (Li *et al.*, 2021) itemised different techniques for effectively identifying bots such as browser fingerprinting, TLS fingerprinting and behavioural analysis. This study focuses on the behavioural analysis techniques.

6.5 Experiment / Case Study 5: Deception Effectiveness

As previously stated, most bots tend to click most things presented on a web page including hidden elements like our bot honeytoken. Honeytokens are deployed in ways that are meant to be deceptive to human threat actors although it might be harder to deceive human threat actors and more so experienced ones, they are unlikely to trigger suspicion from bots. This increases the likelihood of interaction and capture without alerting the attacker. Human threat actors are harder to deceive, due to this a 'hidden' endpoint was created that serves a fake admin form attractive to potential threat actors. Once this form is submitted or the endpoint is clicked, the honeytoken is triggered

6.6 Discussion

This section critically analyses results obtained across six evaluation criteria; these results highlight the strengths and limitations of the proposed adaptive rate limiting model. Comparisons were made against ML-based approaches and traditional static rate limiting models. The overall performance show promise in terms of efficacy, performance accuracy and deception effectiveness but there are areas for improvement in design.

Experiment 1 indicates a higher error rate of the ARL model after detection of bot traffic; this signifies that ARL model worked to apply rate limiting rules based on user behaviour highlighting the efficacy of this study. While this is in line with goals of the study, it is worth noting test scenarios were simulated and model was only evaluated against simulated traffic which limits generalisability of results and obscures how model might behave in real world scenarios.

Furthermore, the model's bot detection system heavily relies on user agent strings for bot detection. This is a simplistic approach and can be easily bypassed by threat actors as user-agent spoofing is widespread as noted by (Li *et al.*, 2021). While adaptive rate limit is the focus, the strength of bot detection is paramount. The study takes a further approach and aligns itself with the defense-in-depth principle by adding honeytokens and bot behaviour analysis techniques also highlighted by (Li *et al.*, 2021) to strengthen bot detection middleware. This is the right step in adequately detecting bots and closing the gap created from over-reliance on user-agent strings. Although this approach is effective, the model can make further improvements by adding further lines of defences on the network layer like TLS fingerprinting and browser fingerprinting.

The low false negative rates ensure legitimate traffic is not getting blocked due to adaptive limiting, while the false positive rate at 33.33% present a security challenge as some bot traffic may pass through set filters and cause harm to application depending on bot type. This further highlights the need for stronger bot detection techniques involved in the model operations.

Response time results revealed a high-performance overhead in the ARL model with a mean response time of 276 ms compared to 4 ms for the static model. This was expected due to additional processing involved and denylist checks. It is a significant difference which can cause bottlenecks in high traffic APIs. These results signals for improvements to design of the

model like caching denylisted IPs in memory using Redis or selective enforcement of detection rules after initial suspicion

The current strategy of honeytoken mechanism was fairly effective during tests but is unlikely to deceive sophisticated or well-designed bots as it assumes that all bots will interact with all elements on a web page, but more experienced threat actors and bots may avoid these triggers. To enhance, honeytokens need to be more randomised and placed more strategically to enrich the deception framework and improve detection rates.

7 Conclusion and Future Work

This paper analyses the effectiveness of integrating adaptive rate limiting and honeytokens to enhance REST API security in JavaScript web applications. The study addresses the research question '*How effective is implementing adaptive rate limiting and honeytoken mechanisms in a secure design framework to mitigate security vulnerabilities in REST APIs for JavaScript web applications*' key findings of the study indicate the proposed model was successful in detecting and mitigating bot traffic with a 99.8%1 error rate when applying adaptive rate limits, effectively deploying honeytokens for luring and monitoring threat actors, outperforming static rate limiting in flexibility by adjusting limits based on threat behaviour and reduced false negative (0%) but showed room for improvement in false positive (33.33%) indicating some bot traffic was undetected. However, limitations of the model include performance overhead with ARL model being 68x slower than baseline model due to additional checks and the use of a simulated test environment limiting its real-world applicability. Regardless of these limitations the study contributes a practical, structured framework that enhance API security through proactive and deceptive techniques.

Future work should focus on enhanced bot detection through other techniques like TLS fingerprinting, browser fingerprinting and ML-based approaches, prioritise performance optimisation and advance honeytoken deception strategies.

References

Abrams, L. (2022) *5.4 million Twitter users' stolen data leaked online — more shared privately*, *BleepingComputer*. Available at: https://www.bleepingcomputer.com/news/security/54-million-twitter-users-stolen-data-leaked-online-more-shared-privately/ (Accessed: 15 June 2025).

Abrams, L. (2024) *Dell API abused to steal 49 million customer records in data breach*, *BleepingComputer*. Available at: https://www.bleepingcomputer.com/news/security/dell-api-abused-to-steal-49-million-customer-records-in-data-breach/ (Accessed: 15 June 2025).

Alharbi, S.J. and Moulahi, T. (2023) 'API Security Testing: The Challenges of Security Testing for Restful APIs', *International Journal of Innovative Research in Science Engineering and Technology*, 8, pp. 1485–1499. Available at: https://doi.org/10.5281/zenodo.7988410.

Annie Brunholzl (2024) 'API IMPACT SECURITY STUDY'. Akamai Technologies Inc. Available at: https://www.akamai.com/lp/report/api-security-study-2024 (Accessed: 4 June 2025).

Coop, R. (2021) *What is the Cost to Deploy and Maintain a Machine Learning Model? | phData*, *phData*. Available at: https://www.phdata.io/blog/what-is-the-cost-to-deploy-and-maintain-a-machine-learning-model/#elementor-toc__heading-anchor-2 (Accessed: 30 July 2025).

Coralogix (no date) 'A Comprehensive Guide to Mean Absolute Percentage Error (MAPE)', *Coralogix*. Available at: https://coralogix.com/ai-blog/a-comprehensive-guide-to-mean-absolute-percentage-error-mape/ (Accessed: 9 August 2025).

DataDome (2025) *Top bots crawler user agent 2025*, *DataDome*. Available at: https://datadome.co/bots/ (Accessed: 2 August 2025).

Deepak (2024) *Dynamic Rate Limiting | Syncloop*, *Syncloop*. Available at: https://www.syncloop.com/blogs/dynamic-api-rate-limiting.html (Accessed: 9 August 2025).

Goodwin, M. (2024) *What Is an API (Application Programming Interface)?*, *IBM*. Available at: https://www.ibm.com/think/topics/api (Accessed: 19 June 2025).

Gowda, P. and Gowda, G. (2024) 'Best Practices in REST API Design for Enhanced Scalability and Security', *Journal of Artificial Intelligence, Machine Learning and Data Science*, 2, pp. 827–830. Available at: https://doi.org/10.51219/JAIMLD/priyanka-gowda/202.

Hindka, M. (2024) 'Securing the Digital Backbone: An In-depth Insights into API Security Patterns and Practices', *ISA Transactions*, 14, pp. 35–41. Available at: https://doi.org/10.5923/j.computer.20241402.02.

Hu, T. *et al.* (2021) 'SEAPP: A secure application management framework based on REST API access control in SDN-enabled cloud environment', *Journal of Parallel and Distributed Computing*, 147, pp. 108–123. Available at: https://doi.org/10.1016/j.jpdc.2020.09.006.

Hussain, F. *et al.* (2019) 'Intelligent Service Mesh Framework for API Security and Management', in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 0735–0742. Available at: https://doi.org/10.1109/IEMCON.2019.8936216.

Hussain, F. *et al.* (2020) 'Enterprise API Security and GDPR Compliance: Design and Implementation Perspective', *IT Professional*, 22(5), pp. 81–89. Available at: https://doi.org/10.1109/MITP.2020.2973852.

IBM (2021) *What Is Three-Tier Architecture? | IBM*, *IBM*. Available at: https://www.ibm.com/think/topics/three-tier-architecture (Accessed: 18 July 2025).

Kajavalta, L. (2022) *REST API SECURITY: TESTING AND ANALYSIS*. Master thesis. Tampere University. Available at: https://trepo.tuni.fi/bitstream/handle/10024/139682/KajavaltaLasse.pdf?sequence=2&isAllowed=y (Accessed: 2 August 2025).

Kornienko, D.V. *et al.* (2021) 'Principles of securing RESTful API web services developed with python frameworks', *Journal of Physics: Conference Series*, 2094(3), p. 032016. Available at: https://doi.org/10.1088/1742-6596/2094/3/032016.

Lamothe, M., Guéhéneuc, Y.-G. and Shang, W. (2022) 'A Systematic Review of API Evolution Literature', *ACM Computing Surveys*, 54(8), pp. 1–36. Available at: https://doi.org/10.1145/3470133.

Li, X. *et al.* (2021) 'Good Bot, Bad Bot: Characterizing Automated Browsing Activity', in *2021 IEEE Symposium on Security and Privacy (SP)*. *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1589–1605. Available at: https://doi.org/10.1109/SP40001.2021.00079.

Nartovich, A. (2024) 'API Security Checklist: 12 Best Practices Everyone Should Implement', *Axway Blog*, 8 July. Available at: https://blog.axway.com/learning-center/digital-security/keys-oauth/api-security-best-practices (Accessed: 9 August 2025).

Nguyen, Q. and Baker, O. (2019) ‘Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API’, *Journal of Software*, pp. 257–264. Available at: https://doi.org/10.17706/jsw.14.6.257-264.

Paul, J. (2024) ‘Building a Robust API Security Framework with Machine Learning’.

Prabhaker, N., Bopche, G.S. and Arock, M. (2024) ‘Generation and deployment of honeytokens in relational databases for cyber deception’, *Computers & Security*, 146, p. 104032. Available at: https://doi.org/10.1016/j.cose.2024.104032.

Rani, M.S. *et al.* (2024) ‘Cyber Honeypot’, *International Journal of Scientific Research in Science and Technology*, 11(2), pp. 94–98. Available at: https://doi.org/10.32628/IJSRST52411168.

Reti, D., Angeli, T. and Schotten, H.D. (2023) ‘Honey Infiltrator: Injecting Honeytoken Using Netfilter’, in *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 465–469. Available at: https://doi.org/10.1109/EuroSPW59978.2023.00057.

Serbout, S. *et al.* (2023a) ‘API Rate Limit Adoption -- A pattern collection’, in *Proceedings of the 28th European Conference on Pattern Languages of Programs*. *EuroPLoP 2023: 28th European Conference on Pattern Languages of Programs*, Irsee Germany: ACM, pp. 1–20. Available at: https://doi.org/10.1145/3628034.3628039.

Serbout, S. *et al.* (2023b) ‘API Rate Limit Adoption -- A pattern collection’, in *Proceedings of the 28th European Conference on Pattern Languages of Programs*. *EuroPLoP 2023: 28th European Conference on Pattern Languages of Programs*, Irsee Germany: ACM, pp. 1–20. Available at: https://doi.org/10.1145/3628034.3628039.

Sharieh, S. and Ferworn, A. (2021) ‘Securing APIs and Chaos Engineering’, in *2021 IEEE Conference on Communications and Network Security (CNS)*. *2021 IEEE Conference on Communications and Network Security (CNS)*, Tempe, AZ, USA: IEEE, pp. 290–294. Available at: https://doi.org/10.1109/CNS53000.2021.9705049.

Shashkina, V. (2024) *Machine Learning (ML) Costs: Price Factors and Real-World Estimates — ITRex*, *ITrex*. Available at: https://itrexgroup.com/blog/machine-learning-costs-price-factors-and-estimates/ (Accessed: 30 July 2025).

Sivaraman, H. (2023) ‘Adaptive Rate Limiting Using Reinforcement Learning to Thwart API Abuse’, *Journal of Mathematical & Computer Applications*, pp. 1–4. Available at: https://doi.org/10.47363/jmca/2023(2)e139.

Vaideeswaran, N. (no date) *What are Honeytokens? | CrowdStrike*, *CrowdStrike.com*. Available at: https://www.crowdstrike.com/en-us/cybersecurity-101/identity-protection/honeytokens/ (Accessed: 9 August 2025).

WhatIsMyBrowser.com (no date) *Bot User Agents - WhatIsMyBrowser.com*, *WhatIsMyBrowser.com*. Available at: https://explore.whatismybrowser.com/useragents/explore/software_type_specific/bot/ (Accessed: 2 August 2025).

Zhao, C. (2024) ‘API Common Security Threats and Security Protection Strategies’, *Frontiers in Computing and Intelligent Systems*, 10(Vol. 10 No. 2 (2024)), p. 5. Available at: https://doi.org/10.54097/k5djs164.