Enhancing Serverless Architecture Security: A Framework for Mitigating Event Injection Attacks

MSc Research Project
MSc In Cybersecurity

Prajwal Yadav
Student ID: 22210270

School of Computing
National College of Ireland

Supervisor: Prof. Mark Monaghan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Prajwal Balasaheb Yadav		
Student ID:	22210270		
Programme:	MSc in Cybersecurity	**Year:**	2024
Module:	Practicum		
Lecturer:	Prof. Mark Monaghan		
Submission Due Date:	12/08/2024		
Project Title:	Enhancing Serverless Architecture Security: A Framework for Mitigating Event Injection Attacks		
Word Count:	5490	**Page Count: 21**	

Signature:	Prajwal Yadav
Date:	12/08/2024

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

Enhancing Serverless Architecture Security: A Framework for Mitigating Event Injection Attacks

Prajwal Yadav
22210270

Abstract

Serverless computing has emerged as a transformative model in cloud service delivery, offering significant advantages such as cost efficiency, scalability, and streamlined development processes. However, this model also introduces unique security challenges, particularly in the form of injection attacks, which exploit vulnerabilities within serverless functions to execute unauthorized commands or access sensitive data. Motivated by the urgent need to address these security concerns, this project aims to develop a comprehensive security framework tailored to mitigate injection attacks in serverless computing environments. The research is guided by the central question: How can a comprehensive security framework be developed to effectively mitigate injection attacks in serverless computing environments? To answer this, the project sets out to investigate the current state of serverless security, design a multi-layered security framework, implement it in a simulated environment, and evaluate its effectiveness. The proposed framework incorporates preventive measures, real-time detection mechanisms, and automated response strategies. It is tested on platforms such as AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions, using various attack scenarios to assess its robustness. Key metrics such as detection accuracy, response time, and impact on system performance are evaluated to ensure the framework's effectiveness and efficiency.

1 Introduction

Serverless computing is a significant evolution in the field of cloud domain where developers didn't worry about the server management, they just handover to the service provider so that they can do their other work. This services not only reduces the operational cost however it also helps for enhancing the scalability as well as the acceleration of deployment cycles. There are few services such as AWS Lambda, Google Cloud function and Microsoft Azure functions. They enable for focusing on writing deployment codes rather than server management to companies. Also, it has some serious security concerns such as third party depended on `and security configurations etc and especially the event injection attacks. this injection attack exploits vulnerabilities within serverless functions which leads to unauthorized access and execution of malicious codes.

The motivation for this project came from urgent need for addressing security vulnerabilities in serverless architecture. As nowadays every company is adopting serverless architectures as it has numerous benefits. As traditional security measures often fall less in serverless environment because it has distinctive nature for example event driven execution and reliance on third parties. My project aims to fill the security gap in existing literature surveys by developing a comprehensive security framework tailored for mitigating injection attacks in serverless computing. By ensuring the advanced security this research seeks for both academic and industrial application from newly emerged threats so it can ensure both integrity and reliability

- **Rise of   Serverless Architecture's Usage:**
  Serverless computing is now commonly used in everyday life whether is it personal usage or corporate usage. It offers a significant advantages such as cost reduction, scalability and mainly the elimination of hardware of servers and its responsibilities as well. This model helps developers for focusing on writing a code while cloud service providers will manage the entire servers and infrastructure.. On the other hand, there are also some security challenges or problems faced during the use of serverless architecture.it includes rising of surface attacks, security misconfigurations and some complexities while using third-party dependencies. For ensuring robust security it requires a comprehensive approach that can highlights these vulnerabilities while using best practices, strict access control, API gateways, Continuous monitoring. From now onwards the use of serverless architecture will rise so there is a need of understanding these security risks along with their mitigation strategies. Also, it will help to preserve integrity and reliability(Li et al., 2022).

- **Network Security Challenges:**
  As we know, serverless computing is an important component of cloud infrastructure with benefits like cost-effectiveness, no need for server management, etc. However, some serious problems can face during its usage. The main problem is an injection attack. Injection attack leads to exploit vulnerabilities in serverless functions that shares a limited runtime environment, it allows for unauthorized access to sensitive data. This problem occurs due to the inherent nature of serverless architecture which lacks networking controls (Ahmadi, 2024).

Consequently, maintaining strong isolation between functions becomes crucial to prevent these attacks. Organizations must implement robust runtime protection mechanisms and stringent input validation measures to detect and mitigate injection attempts early. As serverless functions increasingly depend on third-party libraries and APIs, securing these dependencies is also vital to prevent them from becoming conduits for injection attacks. Addressing these specific security concerns through comprehensive measures, such as deploying Web Application Firewalls (WAFs) and employing advanced anomaly detection algorithms, is essential for preserving the integrity and reliability of serverless computing environments while mitigating the risk of injection attacks.

2 Related Work

Serverless computing has made significant changes in transforming the landscape of cloud computing by offering some amazing benefits such as scalability, cost-efficient, and easy-to-deploy solutions. However, it also introduces unique security challenges that need to be solved. This author has systematically reviewed almost 275 research articles, highlighting the state-of-the-art contributions, platforms, benefits, and ongoing challenges of serverless computing (Hassan, Barakat, and Sarhan, 2021).
The survey reveals that serverless computing enhances development efficiency and reduces operational costs. Yet, it also faces a few critical security issues such as event data injection, insecure deployment configurations, and third-party dependencies, which can compromise security. The paper proposes a comprehensive security framework that includes robust input validation, secure configuration management, minimizing reliance on external components, and continuous monitoring. Moreover, the review underscores the importance of addressing

Distributed Denial of Service (DDoS) attacks through AWS serverless architecture, highlighting the need for dynamic traffic management and real-time threat detection. Future research was to focus on developing an automated tools and frameworks that can support secure coding practices, enhancing the integration of advanced security measures, and exploring the application of serverless computing in different domains.

I. **Understanding of Security Challenges in Serverless Architectures**
Serverless architecture provides substantial benefits such as cost efficiency, scalability, and reliability which allows developers to focus on application development without managing any servers. Although, it introduces some unique security challenges to their event-driven nature. The author identifies a few critical security issues, such as event data injection, insecure deployment configurations, and third-party dependencies. (Sharaf, 2020) Event injection attacks help to exploit the function-triggering mechanism of serverless which, potentially leads to unauthorized access and data breaches. Insecure deployment configurations, tailored to specific customer needs, can introduce vulnerabilities if they are not properly configured. And the last, I.e. third-party dependencies also lead to significant risks, as a single compromised dependency can also affect multiple functions and services. So, (Sharaf, 2020) proposes a comprehensive framework which enhances serverless security, including strict access controls, regular configuration reviews, DevOps collaboration, perimeter security, and continuous monitoring.

II. **Mitigation of DDoS Attacks in Serverless architectures**
AWS serverless architecture offers a cost-effective solution for preventing DDoS attacks and ensures the availability and security of serverless applications. The authors proposed an integration of AWS Lambda functions for dynamically filtering and managing incoming traffic (Brahme et al., 2023). They used DynamoDB as well as AWS Lambda functions for tracking and blocking suspicious and malicious IP addresses. AWS CloudWatch triggers the alarms if, traffic exceeds a predefined limit and sends a notification via AWS Simple Notification Service (SNS). This holistic approach aims for protection of the server by dynamically adjusting to block malicious traffic. (Brahme et al., 2023) Also, authors highlighted some points like the robustness, scalability, reliability, and cost-effectiveness of AWS's solutions, however, they noted some potential delays in triggering alarms and reliance on AWS-specific services. Future work includes extending the framework to other cloud environments and integrating additional security parameters.

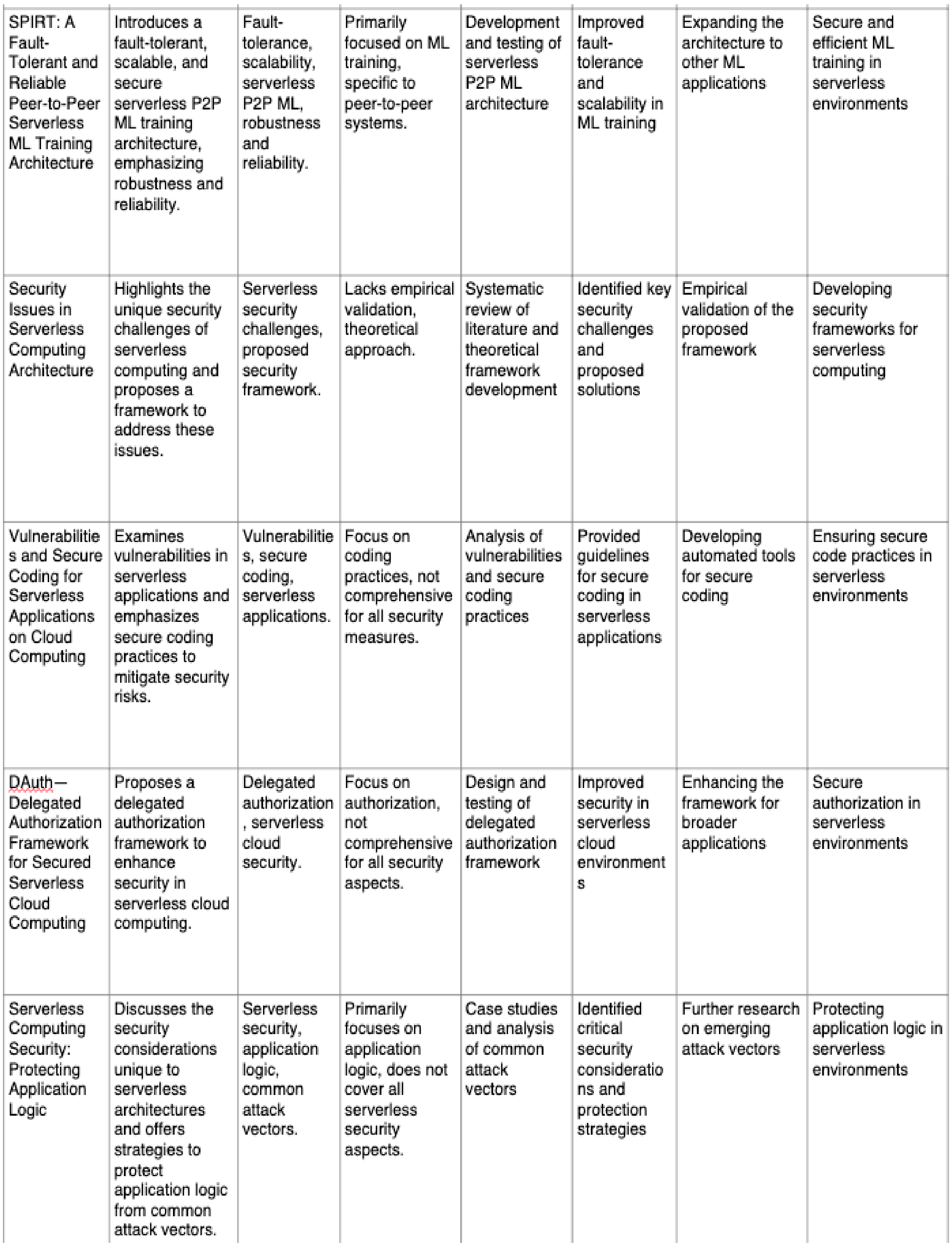
III. **Enhancing Security through Secure Coding Practices**
Serverless applications are vulnerable to several security risks, including insecure configurations, inadequate access controls, and data injection attacks. Mainly data injection attacks, so (Kim, Koo and Kim, 2022) explain why secure coding practices are important to mitigate these types of vulnerabilities. They also highlighted the need for strong input validation to any application for preventing data injection attacks via input and recommended some adopting the principle of least privilege to limit function permissions. The Author also shows some risks associated with third-party dependencies, advocating for thorough security assessments and minimizing external components. (Kim, Koo and Kim, 2022) The proposed secure coding practices include the following points input validation, secure configuration management, robust authentication mechanisms, and regular security testing. Continuous monitoring and incident response are essential for maintaining serverless security. Future research was. To develop automated tools and frameworks to support these practices.

IV. **Centralized Authorization Mechanisms for Serverless Security** (P. Padma and Srinivasan, 2023) DAuth is a delegated authorization framework, which enhances security in serverless environments by centralizing authorization decisions. Use of OAuth 2.0, DAuth separates authorization logic from the serverless functions and ensures consistent access control. Key components include the authorization server, resource server, and client. (P. Padma and Srinivasan, 2023)The authorization server manages access tokens and handles authorization requests, while the resource server enforces access control based on the tokens issued. This centralized approach simplifies access policy management and enhances serverless application security. DAuth's flexibility allows administrators to update access policies without modifying serverless functions, though it presents a single point of failure. Future work was integrating additional security measures and applying DAuth to other cloud service models.

V. **Practical Strategies for Securing Serverless Applications** Serverless applications have several security risks, such as misconfigurations and vulnerabilities related to the function execution workflows. (O'Meara and Lennon, 2020) proposed a practical strategy for mitigating these types of risks, such as implementing strict access controls, encrypted communications, and regular updates of dependencies. They highlighted the importance of logging and monitoring to detect unusual behaviour patterns and potential security threats. Case studies illustrate the effectiveness of these strategies in real-world applications, such as an e-commerce platform that successfully protected against common attack vectors like SQL injection and cross-site scripting (XSS) while maintaining high performance and availability. Future research should explore the development of automated security tools specifically for serverless environments (O'Meara and Lennon, 2020).

VI. **Practical Challenges and Solutions in Serverless Deployments** Deploying the serverless applications has some unique challenges and practical aspects, such as configuration management, security, and performance monitoring. Tembhekar (Prachi Tembhekar, Shanmugam and Devan, 2023), Shanmugam, and Devan focused on AWS Lambda for exploring these issues. Proper configuration management ensures that serverless functions are optimized for performance and secure from vulnerabilities. Developers should secure their serverless applications, as cloud providers handle the underlying infrastructure's security. (Prachi Tembhekar, Shanmugam and Devan, 2023)Performance monitoring is also crucial, as traditional tools may not suit the stateless and dynamic nature of serverless functions. AWS CloudWatch is recommended for detailed metrics on function execution times, error rates, and resource usage. Case studies highlight challenges and solutions, such as cold start latency and function timeout settings. The authors acknowledges that serverless computing may not suit all applications, particularly those with consistent workloads or long-running processes. Future research was to explore optimization techniques and tools to enhance serverless application deployment and management.

Paper	Summary	Key point	Limitation	Methodology	Result	Future work	Practical Implementation
DDOS Prevention System using AWS Serverless Architecture	Discusses using AWS serverless architecture to create a cost-effective solution for preventing DoS and DDoS attacks.	Cost-effective prevention, AWS serverless architecture, DoS/DDoS attacks.	Focuses on DoS/DDoS attacks, not comprehensive for all serverless security issues.	Implementation of AWS serverless architecture to mitigate DDoS attacks	Demonstrated cost-effective and scalable DDoS prevention	Suggested extending the framework to other cloud platforms	Implementation in real-world scenarios for small to medium-sized businesses
A Microservice and Serverless Architecture for Secure IoT System	Designs a secure IoT system using serverless architecture and microservices, addressing security challenges in cross-border logistics.	Secure IoT system, serverless architecture, microservices, cross-border logistics.	Focuses on IoT systems, not generalizable to all serverless applications.	Simulation and testing of secure IoT system	Improved security and efficiency in IoT systems	Exploring other IoT applications and enhancing security measures	Securing cross-border logistics in IoT systems
Implementing Serverless Architecture: Discuss the Practical Aspects and Challenges	Explores practical aspects and challenges of deploying computations to serverless backends, focusing on AWS Lambda.	Practical deployment, AWS Lambda, performance analysis.	Focus on AWS Lambda, limited scope for other serverless platforms.	Case study of AWS Lambda deployment	Identified key challenges and solutions in deployment	Further research on optimizing serverless deployments	Practical guidelines for organizations deploying serverless applications
BIoMT: A State-of-the-Art Consortium Serverless Network Architecture for Healthcare System Using Blockchain Smart Contracts	Proposes a serverless network architecture for healthcare using blockchain to ensure security, integrity, and transparency of health transactions.	Healthcare system, blockchain, serverless network, security and transparency.	Focused on healthcare, specific use case.	Design and simulation of blockchain-based serverless network	Enhanced security and transparency in healthcare transactions	Applying the model to other sectors such as finance	Securing healthcare transactions and data integrity
AERF-Adaptive ensemble random fuzzy algorithm for anomaly detection in cloud computing	Proposes an adaptive ensemble random fuzzy algorithm to enhance anomaly detection in cloud computing, focusing on improving detection accuracy and reducing false positives.	Anomaly detection, cloud computing, fuzzy algorithm.	Focused on anomaly detection, limited to specific security aspect.	Development and testing of fuzzy algorithm for anomaly detection	Improved detection accuracy and reduced false positives	Extending the algorithm to other security applications	Advanced anomaly detection in cloud environments



SPIRT: A Fault-Tolerant and Reliable Peer-to-Peer Serverless ML Training Architecture	Introduces a fault-tolerant, scalable, and secure serverless P2P ML training architecture, emphasizing robustness and reliability.	Fault-tolerance, scalability, serverless P2P ML, robustness and reliability.	Primarily focused on ML training, specific to peer-to-peer systems.	Development and testing of serverless P2P ML architecture	Improved fault-tolerance and scalability in ML training	Expanding the architecture to other ML applications	Secure and efficient ML training in serverless environments
Security Issues in Serverless Computing Architecture	Highlights the unique security challenges of serverless computing and proposes a framework to address these issues.	Serverless security challenges, proposed security framework.	Lacks empirical validation, theoretical approach.	Systematic review of literature and theoretical framework development	Identified key security challenges and proposed solutions	Empirical validation of the proposed framework	Developing security frameworks for serverless computing
Vulnerabilities and Secure Coding for Serverless Applications on Cloud Computing	Examines vulnerabilities in serverless applications and emphasizes secure coding practices to mitigate security risks.	Vulnerabilities, secure coding, serverless applications.	Focus on coding practices, not comprehensive for all security measures.	Analysis of vulnerabilities and secure coding practices	Provided guidelines for secure coding in serverless applications	Developing automated tools for secure coding	Ensuring secure code practices in serverless environments
DAuth—Delegated Authorization Framework for Secured Serverless Cloud Computing	Proposes a delegated authorization framework to enhance security in serverless cloud computing.	Delegated authorization, serverless cloud security.	Focus on authorization, not comprehensive for all security aspects.	Design and testing of delegated authorization framework	Improved security in serverless cloud environments	Enhancing the framework for broader applications	Secure authorization in serverless environments
Serverless Computing Security: Protecting Application Logic	Discusses the security considerations unique to serverless architectures and offers strategies to protect application logic from common attack vectors.	Serverless security, application logic, common attack vectors.	Primarily focuses on application logic, does not cover all serverless security aspects.	Case studies and analysis of common attack vectors	Identified critical security considerations and protection strategies	Further research on emerging attack vectors	Protecting application logic in serverless environments

3. Implementation

I have created one simple Inventory management tool. In which there is one login page. And if you are login, it with admin you then are allowed to do any action, but if you are login by any of the normal user then your User Interface will be slightly different.
For this webapp I have used basic html templates and python along with Django framework. For securing this from bad actors for preventing mainly event injection attack, I have created a

framework in three steps.

In serverless architecture, securing any application from command injection attacks such as SQL injection, XSS, can exploit the vulnerabilities in serverless functions which leads to data breaches and service disruptions. To overcome this, I proposed an integrating machine learning techniques for improving the detection and prevention of these sophisticated attacks. ML is effective in identifying different patterns and anomalies which traditional rule-based security systems can miss, so providing a dynamic and adaptive multilayer to defence.

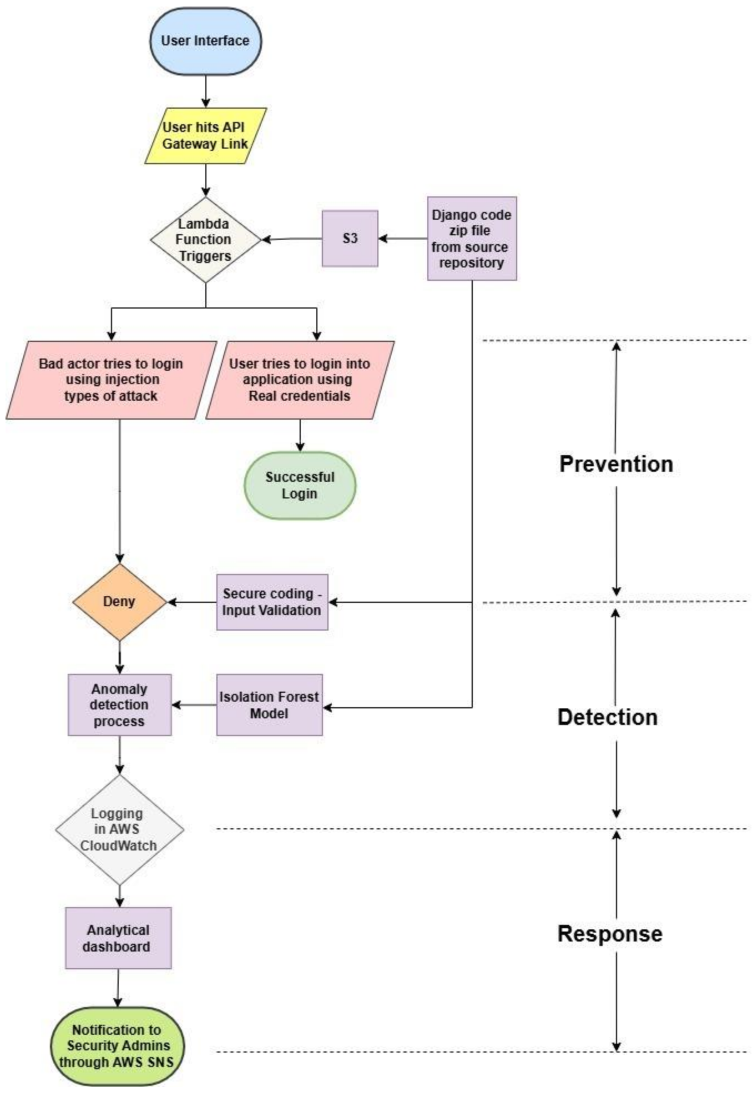

User Interface
User hits API Gateway Link
Lambda Function Triggers
S3
Django code zip file from source repository
Bad actor tries to login using injection types of attack
User tries to login into application using Real credentials
Successful Login
Prevention
Deny
Secure coding - Input Validation
Anomaly detection process
Isolation Forest Model
Detection
Logging in AWS CloudWatch
Analytical dashboard
Response
Notification to Security Admins through AWS SNS

Fig 1. Workflow of framework (Original Illustration)

3.1. Preventive Measures: Secure Code

Project begins with development and deployment of Inventory Management system on AWS serverless architecture. This application facilitates the management of products, orders, and staff details with a very user-friendly interface. Secure coding is a crucial step for preventing vulnerabilities that can be exploited thereby enhancing the overall security posture of our application. Implementation of secure coding also ensures the CIA triads if application and its data. To protect against various command injection attacks, such as SQL injection, Cross-Site Scripting (XSS), command injection, LDAP injection, and XML injection, I implemented a comprehensive input validation along with the logging mechanisms. This holistic multi-layered security approach helps detect and mitigate potential security threats very effectively. This code integrates robust input validation for preventing multiple types of injection attacks. It uses regular expressions for detecting and blocking malicious patterns commonly used in attacks. Additionally, logging mechanisms are also implemented to monitor and record significant events, which ensures the detecting a suspicious activity and troubleshooting errors.

Explanation of the Code Logic:

Input Validation: The validate_input function: checks for the input data against SQL injection, XSS, command injection, LDAP injection, and XML injection using regular expressions. If there is any malicious patterns are detected, a ValidationError will be raised. By integrating these secure coding practices, we can ensure that the application is fully protected against different types of injection attacks, enhancing its overall security posture and robustness. This approach integrates with the principles of proactive security and continuous monitoring to safeguard the serverless

3.2. Detection Mechanism:

Detection mechanisms are very important in cybersecurity as it helps for the real-time detection of attacks as well as anomalous activities. In serverless computing, where applications and resources dynamically scaled, monitoring and detecting unusual behaviour is mandatory and hard to maintaining the integrity and security of the system. If we have an effective detection mechanism, so we don’t need to worry if security incidents happened which will lead to significant damage.

How Detection Mechanisms Enhance Security?

After implementing the robust detection mechanisms, we can quickly detect as well as respond as possible to security threats, so it can significantly reduce potential damage and help for preventing the escalation of attacks. These mechanisms aim to continuously monitor the system activities, detect anomalies, and if necessary, It can trigger automated responses to mitigate future risks. This holistic approach ensures that potential threats are identified on time and addressed before they can cause growth and disruption.

Machine Learning Algorithm: Isolation Forest

To enhance our detection capabilities, I decided to integrate the Isolation Forest algorithm for anomaly detection. This machine learning algorithm is particularly well-suited for identifying unusual patterns as well as behaviors in serverless environments. Here’s why I chose it.

Implementation in Our Application: We have collected relevant data from our serverless application environment, such as resource usage API call patterns, and logs, and applied the Isolation Forest algorithm to detect anomalies.

Logging Mechanism

Logging is another crucial component of our detection mechanism. We had two options for logging: the built-in Django logging framework and AWS CloudWatch. We chose to integrate CloudWatch with Django to enhance our logging capabilities, especially for large-scale serverless environments. The log captures detailed information such as system events, user activities, or application errors. This data is very essential for detecting different types of anomalies, detecting possible potential security attacks, and providing a detailed audit trail that can be used for forensic analysis.

Implementation: Integrating CloudWatch with Django

We integrated AWS CloudWatch for capturing the logs as well monitor log data in real time. This integration allows us to use CloudWatch's advanced monitoring system and alerting features. It will help to enhance detection and response capabilities. We used the Watchtower library to send log data directly to CloudWatch from our Django application.

3.3 Response phase:

The response phase is critical in the security framework as it determines the immediate actions taken to mitigate the impact of a detected threat. Effective response mechanisms ensure that once a security incident is detected, it is swiftly contained, preventing further damage and restoring normal operations as quickly as possible. This phase involves both automated and manual actions to neutralize threats and minimize downtime.

How do Response Mechanisms Enhance Security?

Response mechanisms are designed to act quickly and efficiently, reducing the window of opportunity for attackers. By having predefined response strategies, organizations can ensure a coordinated and effective reaction to security incidents. This minimizes the potential damage and aids in quicker recovery, thereby maintaining the integrity and availability of the system.

Implementation: Automated Response Mechanisms

To enhance our response capabilities, I have implemented automated response mechanisms which will triggered if there is the detection of security incidents. These mechanisms use the logging and monitoring data collected via AWS CloudWatch to execute predefined actions. The techniques and/or architecture and/or framework that underlie the implementation and the associated requirements are identified and presented in this section. If a new algorithm or model is proposed, a word-based description of the algorithm/model functionality should be included.

4. Design Specification

The design and setup of the serverless architecture security framework aim to provide robust and scalable solutions to secure serverless applications against common and advanced security attacks and threats. The architecture uses AWS services, machine learning, and traditional security measures.

Research Procedure:

It involves the detection of the key security challenges that are faced by serverless architecture which primarily focuses on injection attacks such as such as SQL injection, XSS, command injection, etc., and anomaly detection. After this solution was decided to integrate with AWS services such as Lambda, CloudWatch, and SNS.

1. Problem Identification: Understanding the event injection attacks and research about the existing solutions.

2. Solution Design: Design a security framework that will incorporate the traditional approach of secure coding with advanced anomaly detection using ML.

3. Implementation strategy: Implementing this framework using AWS lambda for serverless computing and CloudWatch for logging, SNs for notification.

4. Testing and Evaluation: create test cases to validate input validation and monitor the framework's response to these attacks.

Components of my framework:

AWS Lambda: For executing serverless functions that can handle backend logic and processing. It provides a scalable, cost-effective approach without handling servers.

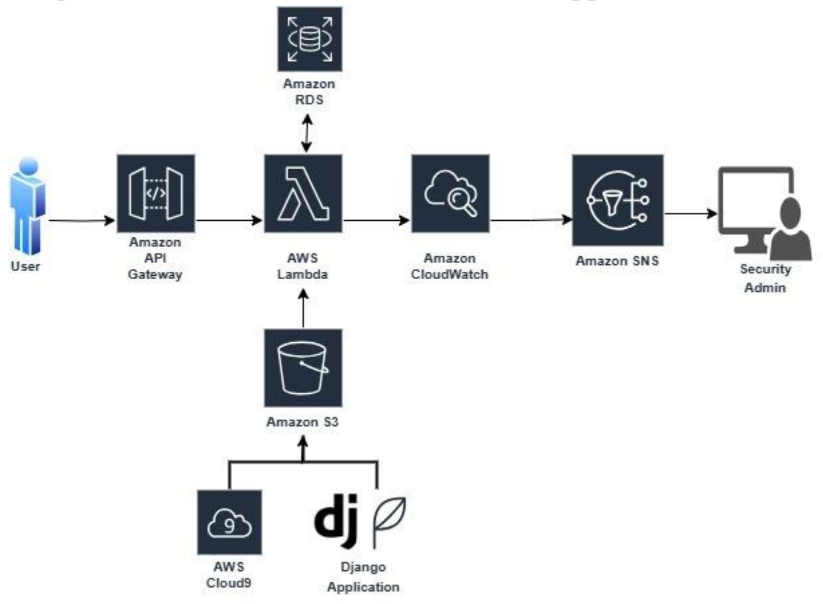


Fig. 2. Architecture Diagram (Original Illustration)

AWS API Gateway: It is used for managing and routing HTTP requests from front end to lambda functions. It helps simplify the creation, deployment, and management of APIs.
AWS RDS: It is used for storing and managing the data as it offers scalable and managed database service which integrates with Django and all.
AWS S3: It helps for storing static files, media, and other user-generated content. I used this to string my code there.
Django Framework: I have used this for creating web apps and for backend logic.
CloudWatch: I have used this CloudWatch for monitoring purposes. It monitors application performance, and tracklogs, and sets up alarms for any anomaly detection.
It offers real-time monitoring and logging capabilities which allows us to continuously watch the application's health and the detection of potential security threats.
AWS SNS: I have used the SNS for sending notifications and alerts when there is a certain threshold or any anomalies are detected.
It also ensures that alerts are promptly delivered which enables quicker response.
GitHub: I have used this for managing version controls for the codebases which facilitates collaboration and deployment.
It is a platform for source code management and enables continuous integration as well as collaboration among other members.
AWS Cloud9: I have used this to provide a cloud-based integrated development environment (IDE) for coding and debugging. Cloud9 allows everyone to write, run, and debug code in a

browse. It also has seamless integration with other AWS services which makes it easier to develop and deploy applications directly from the cloud.

ZIPAPP: I have used this for packaging of Django application and its dependencies into a deployable zip file. It offers a deployment process by bundling the entire process and application while ensuring all necessary components are included.

Why I chose that method only for anomaly detection:

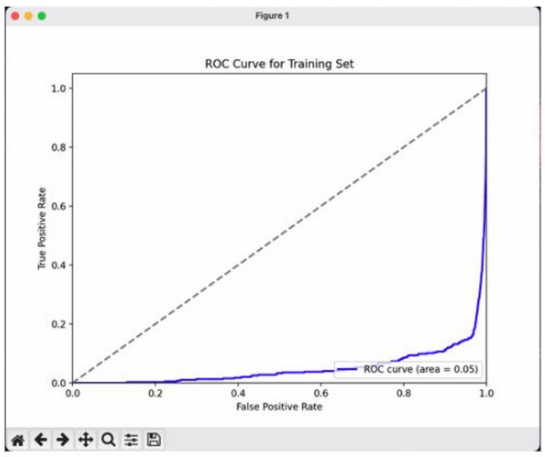
I chose specifically the Isolation Forest algorithm for anomaly detection in serverless security framework as it has unique characteristics and suitability for cybersecurity applications. Isolation Forest is efficient, scalable reliable, and mainly capable of handling large datasets typical in serverless environments (Geeksforgeeks, 2024). It starts work by isolating the anomalies through random partitioning, which makes it specifically adaption to identifying outliers that can be potential security threats. However, this method is non-parametric. It just requires minimal assumptions for data distribution. This makes it versatile for various types of anomalies like point, contextual, or collective anomalies (Geeksforgeeks, 2024). Also, its robustness and low maintenance requirements make it an ideal choice for real-time anomaly detection especially for dynamic cloud environments.

5. Evaluation

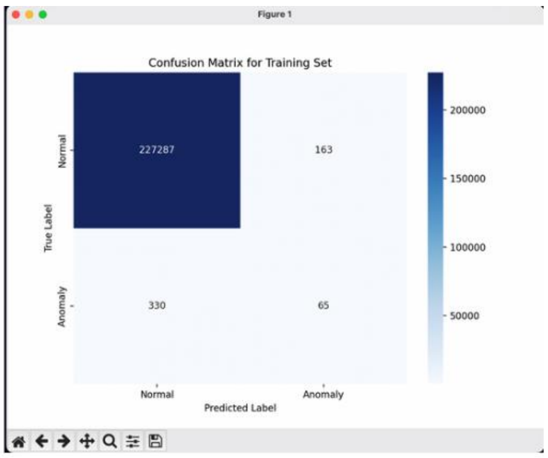
I have done an evaluation of three types first for checking the effectiveness of input validation and secure coding by 3 test cases, second was testing of anomaly detection And final was on the framework after launching it on the AWS lambda.

Evaluation of ML model:

The evaluation of our anomaly detection model was conducted locally first time before pushing the code in lambda. So, the primary goal was to ensure the model's effectiveness in detecting anomalies. The model was developed using Isolation Forest Algorithm a robust technique for anomalies detection. I trained the model on a dataset of credit card transactions. In which there were both transactions like a small portion of fraudulent transactions and normal transactions as well. The model was trained to find anomalies in behaviour.

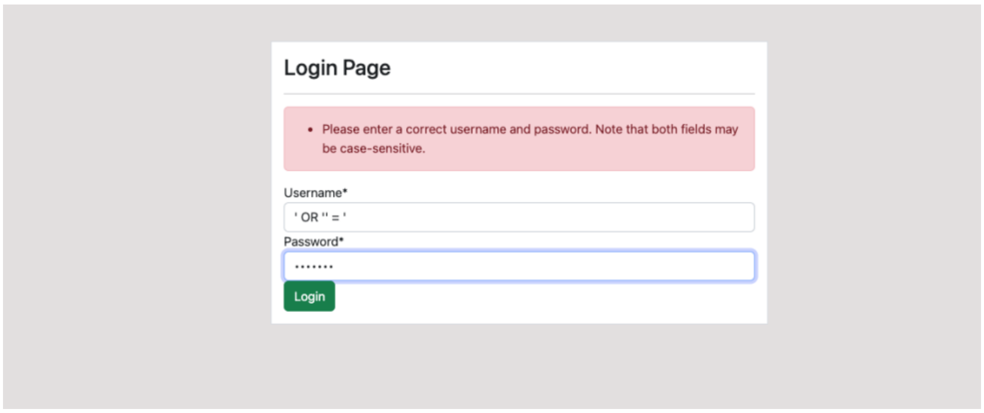


ROC CURVE



CONFUSION MATRIX

The second part was related to lambda functioning itself. I tried to attack the SQL injection on the Login page and then I found the warning about it. Below you can see the screenshots attached of it. And also we can see one alarm of it as we have.



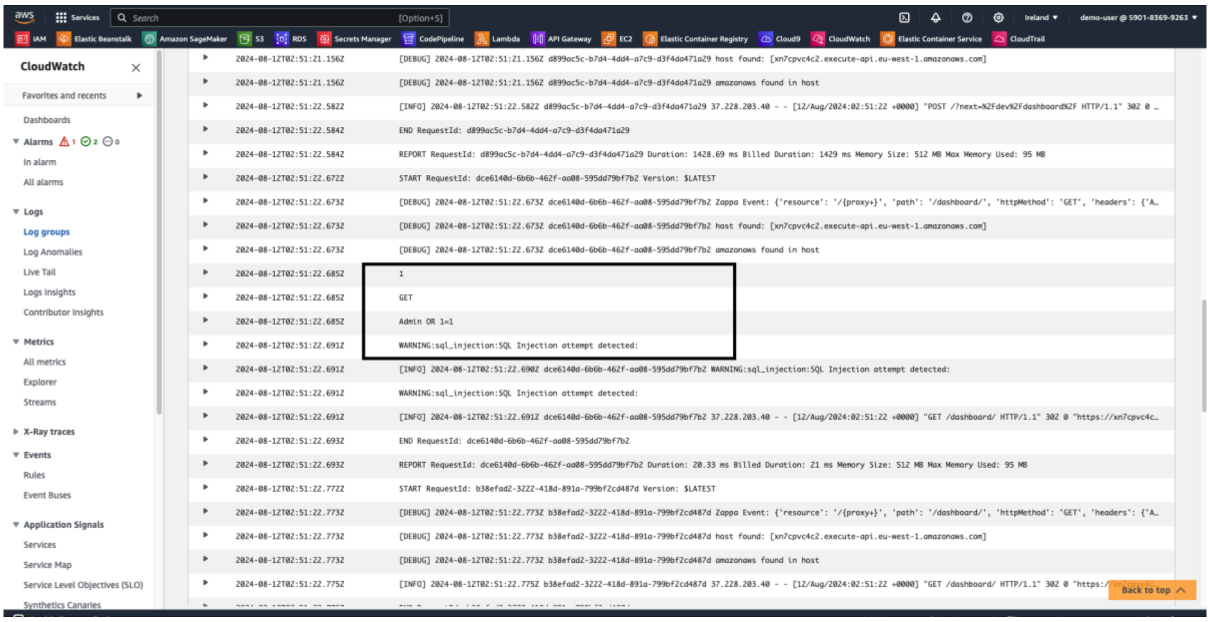
Login Page

- Please enter a correct username and password. Note that both fields may be case-sensitive.

Username* ' OR '' = '

Password* •••••••

Fig.3. Login Page



```
2024-08-12T02:51:21.156Z    [DEBUG] 2024-08-12T02:51:21.156Z d899ac5c-b7d4-4dd4-a7c9-d3f4da471a29 host found: [xn7cpvc4c2.execute-api.eu-west-1.amazonaws.com]
2024-08-12T02:51:21.156Z    [DEBUG] 2024-08-12T02:51:21.156Z d899ac5c-b7d4-4dd4-a7c9-d3f4da471a29 amazonaws found in host
2024-08-12T02:51:22.582Z    [INFO] 2024-08-12T02:51:22.582Z d899ac5c-b7d4-4dd4-a7c9-d3f4da471a29 37.228.203.40 - - [12/Aug/2024:02:51:22 +0000] "POST /?next=%2Fdev%2Fdashboard%2F HTTP/1.1" 302 0 …
2024-08-12T02:51:22.584Z    END RequestId: d899ac5c-b7d4-4dd4-a7c9-d3f4da471a29
2024-08-12T02:51:22.584Z    REPORT RequestId: d899ac5c-b7d4-4dd4-a7c9-d3f4da471a29 Duration: 1428.69 ms Billed Duration: 1429 ms Memory Size: 512 MB Max Memory Used: 95 MB
2024-08-12T02:51:22.672Z    START RequestId: dce6140d-6b6b-462f-aa08-595dd79bf7b2 Version: $LATEST
2024-08-12T02:51:22.673Z    [DEBUG] 2024-08-12T02:51:22.673Z dce6140d-6b6b-462f-aa08-595dd79bf7b2 Zappa Event: {'resource': '/{proxy+}', 'path': '/dashboard/', 'httpMethod': 'GET', 'headers': {'A…
2024-08-12T02:51:22.673Z    [DEBUG] 2024-08-12T02:51:22.673Z dce6140d-6b6b-462f-aa08-595dd79bf7b2 host found: [xn7cpvc4c2.execute-api.eu-west-1.amazonaws.com]
2024-08-12T02:51:22.673Z    [DEBUG] 2024-08-12T02:51:22.673Z dce6140d-6b6b-462f-aa08-595dd79bf7b2 amazonaws found in host
2024-08-12T02:51:22.685Z    1
2024-08-12T02:51:22.685Z    GET
2024-08-12T02:51:22.685Z    Admin OR 1=1
2024-08-12T02:51:22.691Z    WARNING:sql_injection:SQL Injection attempt detected:
2024-08-12T02:51:22.691Z    [INFO] 2024-08-12T02:51:22.690Z dce6140d-6b6b-462f-aa08-595dd79bf7b2 WARNING:sql_injection:SQL Injection attempt detected:
2024-08-12T02:51:22.691Z    WARNING:sql_injection:SQL Injection attempt detected:
2024-08-12T02:51:22.691Z    [INFO] 2024-08-12T02:51:22.691Z dce6140d-6b6b-462f-aa08-595dd79bf7b2 37.228.203.40 - - [12/Aug/2024:02:51:22 +0000] "GET /dashboard/ HTTP/1.1" 302 0 "https://xn7cpvc4c…
2024-08-12T02:51:22.693Z    END RequestId: dce6140d-6b6b-462f-aa08-595dd79bf7b2
2024-08-12T02:51:22.693Z    REPORT RequestId: dce6140d-6b6b-462f-aa08-595dd79bf7b2 Duration: 20.33 ms Billed Duration: 21 ms Memory Size: 512 MB Max Memory Used: 95 MB
2024-08-12T02:51:22.772Z    START RequestId: b38efad2-3222-418d-891a-799bf2cd487d Version: $LATEST
2024-08-12T02:51:22.773Z    [DEBUG] 2024-08-12T02:51:22.773Z b38efad2-3222-418d-891a-799bf2cd487d Zappa Event: {'resource': '/{proxy+}', 'path': '/dashboard/', 'httpMethod': 'GET', 'headers': {'A…
2024-08-12T02:51:22.773Z    [DEBUG] 2024-08-12T02:51:22.773Z b38efad2-3222-418d-891a-799bf2cd487d host found: [xn7cpvc4c2.execute-api.eu-west-1.amazonaws.com]
2024-08-12T02:51:22.773Z    [DEBUG] 2024-08-12T02:51:22.773Z b38efad2-3222-418d-891a-799bf2cd487d amazonaws found in host
2024-08-12T02:51:22.775Z    [INFO] 2024-08-12T02:51:22.775Z b38efad2-3222-418d-891a-799bf2cd487d 37.228.203.40 - - [12/Aug/2024:02:51:22 +0000] "GET /dashboard/ HTTP/1.1" 302 0 "https:/
```

Fig.4 . Logs

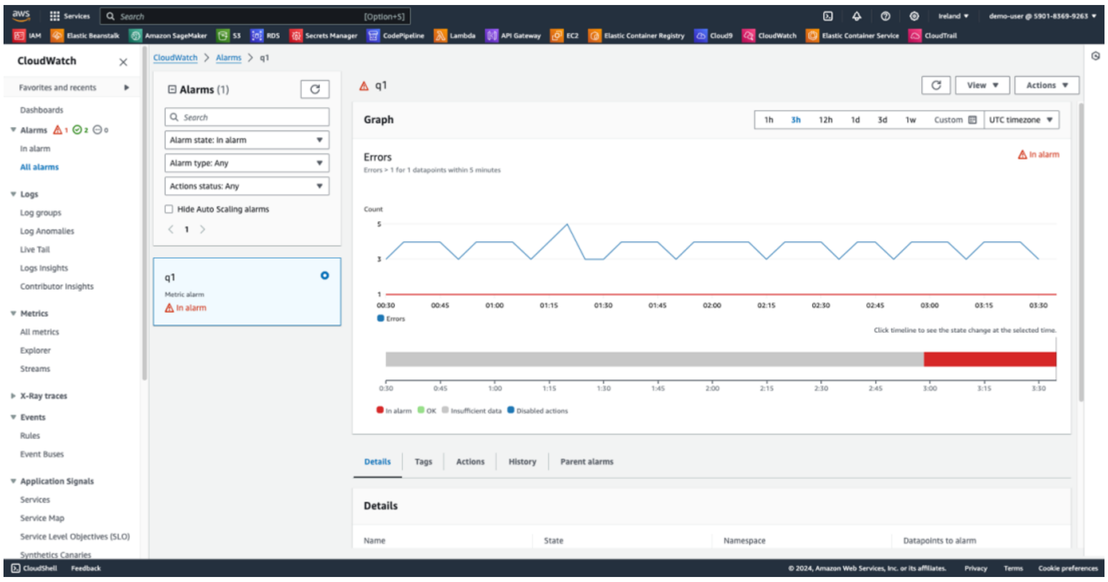
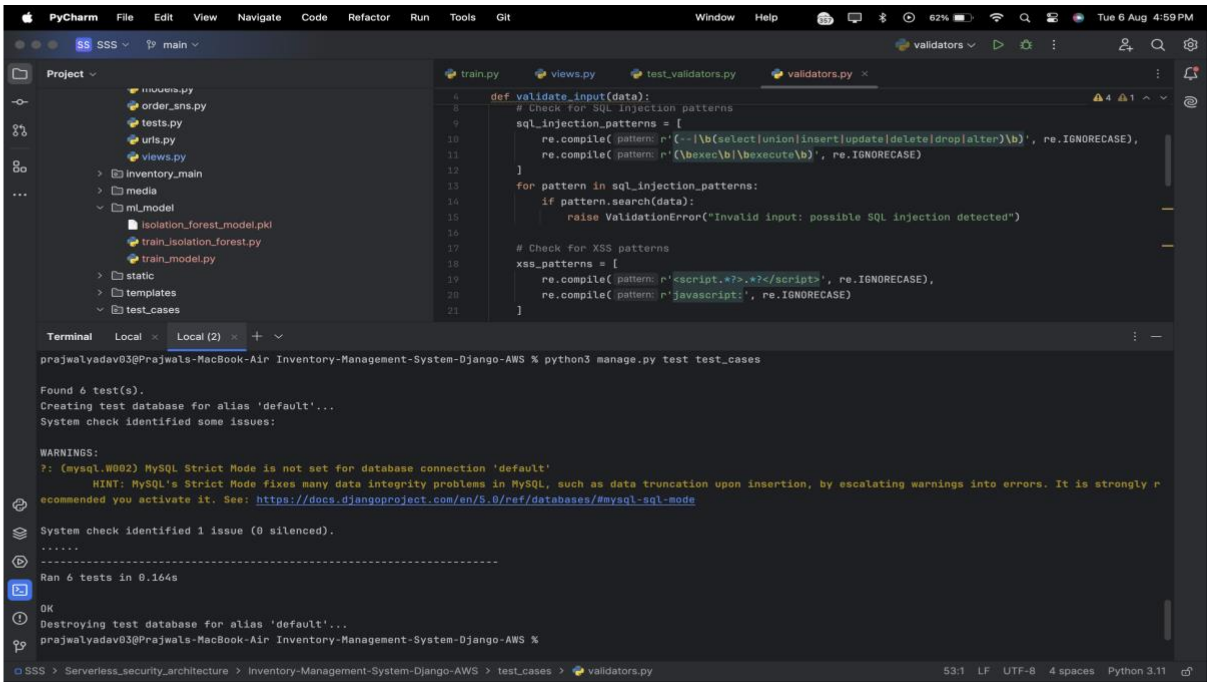


Fig.5. Alarm

According to the Confusion matrix, It indicates that the model was relatively accurate for detecting anomalies without flagging too many transactions which are normal but identifies as anomalous. ROC curve showed that the model demonstrated models modest but reliable performance in differencing between normal and anomalous.

5.1 Test Case 1

Here, I have performed a few test cases for evaluating the effectiveness of the implementation of secure coding, particularly for Command injections and SQL injection. I have done it in two ways. The first way that Django has an inbuilt feature for testing where it covers different types of input, including valid inputs and inputs that are designed for exploitation purposes only. And second was to test it manually. The first test case was done for testing 6 things which are SQL injection patterns, XSS patterns, Command injection patterns, LDAP injection patterns, XML injection attacks, and lastly custom field validation. There was one error related to the MYSQL strict mode which prevented the data integrity issues.



```python
def validate_input(data):
    # Check for SQL injection patterns
    sql_injection_patterns = [
        re.compile( pattern: r'(--|\b(select|union|insert|update|delete|drop|alter)\b)', re.IGNORECASE),
        re.compile( pattern: r'(\bexec\b|\bexecute\b)', re.IGNORECASE)
    ]
    for pattern in sql_injection_patterns:
        if pattern.search(data):
            raise ValidationError("Invalid input: possible SQL injection detected")

    # Check for XSS patterns
    xss_patterns = [
        re.compile( pattern: r'<script.*?>.*?</script>', re.IGNORECASE),
        re.compile( pattern: r'javascript:', re.IGNORECASE)
    ]
```

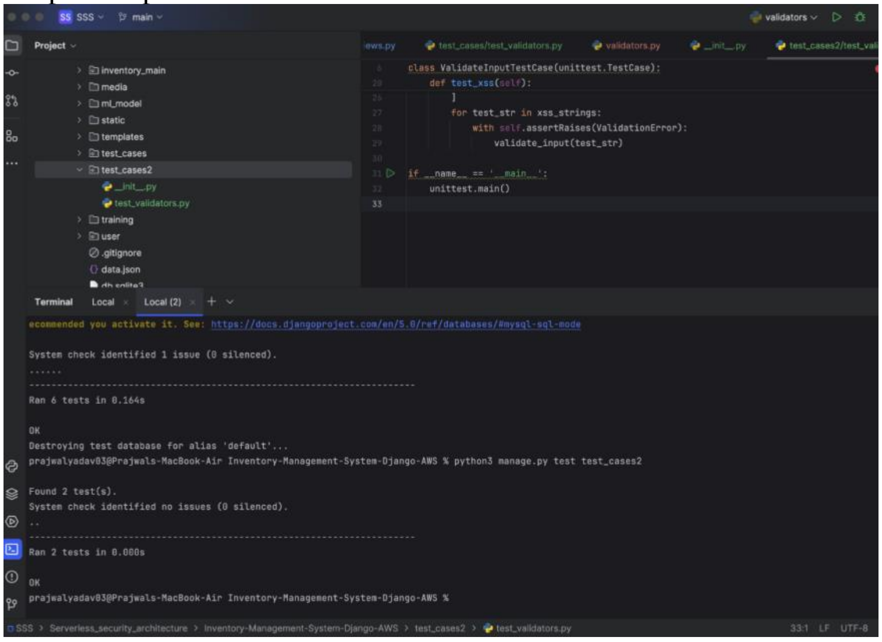
```
prajwalyadav03@Prajwals-MacBook-Air Inventory-Management-System-Django-AWS % python3 manage.py test test_cases

Found 6 test(s).
Creating test database for alias 'default'...
System check identified some issues:

WARNINGS:
?: (mysql.W002) MySQL Strict Mode is not set for database connection 'default'
        HINT: MySQL's Strict Mode fixes many data integrity problems in MySQL, such as data truncation upon insertion, by escalating warnings into errors. It is strongly r
ecommended you activate it. See: https://docs.djangoproject.com/en/5.0/ref/databases/#mysql-sql-mode

System check identified 1 issue (0 silenced).
......
----------------------------------------------------------------------
Ran 6 tests in 0.164s

OK
Destroying test database for alias 'default'...
prajwalyadav03@Prajwals-MacBook-Air Inventory-Management-System-Django-AWS %
```

Fig.6.Test case 1

Result:
6/6= All test cases passed effectively.
The test results indicate that the input validation function is working very effectively for detecting and preventing the potentially harm inputs which was crucial in securing the application against different types of injection attacks. This validation logic enhanced the overall security posture of our serverless application by preventing common vulnerabilities such as SQL injection, XSS, command injection, LDAP, and XML injection.

5.2 Test Case 2

The second test case was particularly for the detection of SQL injection and XSS injection. According to SC media XSS and SQL injection are very dangerous as they led to the compromise of almost 65 websites and over 2M jobseeker's data were compromised due to those attacks, (Staff, 2024). So, I decided to do a testing for such as so that in future this framework will help to stop these kinds of attacks.



```python
class ValidateInputTestCase(unittest.TestCase):
    def test_xss(self):
        ]
        for test_str in xss_strings:
            with self.assertRaises(ValidationError):
                validate_input(test_str)

if __name__ == '__main__':
    unittest.main()
```

```
ecommended you activate it. See: https://docs.djangoproject.com/en/5.0/ref/databases/#mysql-sql-mode

System check identified 1 issue (0 silenced).
......
----------------------------------------------------------------------
Ran 6 tests in 0.164s

OK
Destroying test database for alias 'default'...
prajwalyadav03@Prajwals-MacBook-Air Inventory-Management-System-Django-AWS % python3 manage.py test test_cases2

Found 2 test(s).
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
prajwalyadav03@Prajwals-MacBook-Air Inventory-Management-System-Django-AWS %
```

Fig.7.Test case 2

Result:
This test case was very crucial for ensuring whether our application effectively detects and handles these types of injection attacks which are common in web apps. 2/2 = All tests passed successfully.

The validate input function was successfully detected and raised validation error for all SQL injection payloads as well as all XSS payloads. So, this indicates that the function is effective in detecting SQL injection and XSS attacks as well. So, after passing this the function has demonstrated the capacity to prevent two of the most common and dangerous web application vulnerabilities.

5.3 Test Case 3

This test case was designed to check and validate the custom field input based on predefined patterns ensuring that they are following the security requirements.

Result:

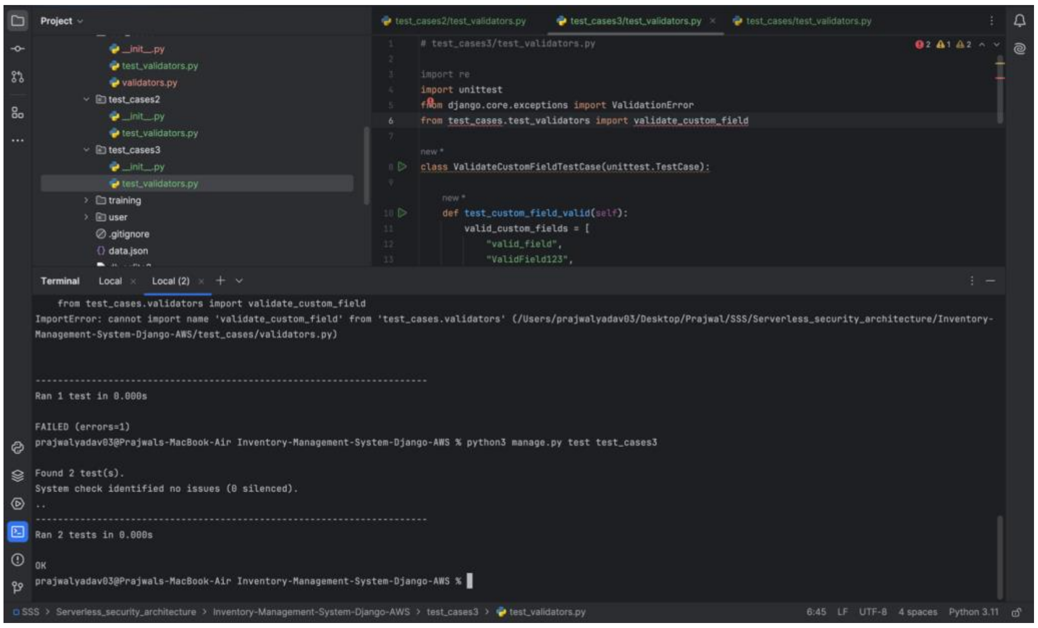
2/2= Successfully passed

The test cases evaluated multiple valid and invalid custom field inputs such as,

Valid input: valid field and ValidField123

Invalid input: invalid field! And invalid!d@Field

The results of our test cases indicate that our input validation mechanisms are robust as well as capable of detecting various types of injection attacks and invalid inputs. The serverless security framework successfully prevented all possible attacks such as XSS, and SQL injection, and ensured that it also followed the custom invalid inputs. These results can significantly contribute to overall security posture. After ensuring that inputs are properly sanitized and validated effectively.



```python
# test_cases3/test_validators.py

import re
import unittest
from django.core.exceptions import ValidationError
from test_cases.test_validators import validate_custom_field

class ValidateCustomFieldTestCase(unittest.TestCase):

    def test_custom_field_valid(self):
        valid_custom_fields = [
            "valid_field",
            "ValidField123",
```

```
    from test_cases.validators import validate_custom_field
ImportError: cannot import name 'validate_custom_field' from 'test_cases.validators' (/Users/prajwalyadav03/Desktop/Prajwal/SSS/Serverless_security_architecture/Inventory-Management-System-Django-AWS/test_cases/validators.py)


----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
prajwalyadav03@Prajwals-MacBook-Air Inventory-Management-System-Django-AWS % python3 manage.py test test_cases3

Found 2 test(s).
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
prajwalyadav03@Prajwals-MacBook-Air Inventory-Management-System-Django-AWS %
```

Project
test_cases2/test_validators.py
test_cases3/test_validators.py
test_cases/test_validators.py
__init__.py
test_validators.py
validators.py
test_cases2
__init__.py
test_validators.py
test_cases3
__init__.py
test_validators.py
training
user
.gitignore
data.json
Terminal
Local
Local (2)
SSS > Serverless_security_architecture > Inventory-Management-System-Django-AWS > test_cases3 > test_validators.py
6:45 LF UTF-8 4 spaces Python 3.11

Fig.8.Test case 3

• Discussion

The findings from experiments and evaluation conducted in this thesis demonstrated that the proposed serverless architecture security framework can effectively mitigate all the injection attacks within serverless computing environments, especially through the integration of advanced ML methods like the Isolation Forest for real-time anomaly detection. The framework successfully identified and prevented various types of injection attacks, such as SQL injection, XSS, Command Injection, LDAP injection, and XML injection, showing the robustness of the input validation mechanisms. Future research should include, incorporating more advanced machine learning techniques ensemble learning, or LLMs. Also, develop a modular validation system for more scalability. Integrating it with real-time threat intelligence helps improve detection capabilities. Overall, the framework proved effective, continuous adaptation and enhancement are necessary for maintaining its effectiveness.

Empowering Small Businesses with Scalable Security:
My proposed framework is not only designed to address the critical security aspects and challenges in serverless computing environments but also to enhance small businesses by providing them with enterprise-level security that is cost-effective as well as scalable. By using the advanced machine learning models such as the Isolation Forest, with the AWS Lambda functions. The framework also ensures real-time detection and prevention of event injection attacks, it offers a robust security integration that is typically accessible to larger organizations. The proposed framework covers multiple injection vulnerabilities or attacks such as SQL, XSS, and Command injections. It also ensures a broad protection scope across serverless environments. This is particularly useful for small businesses that may lack the resources for deploying and maintaining the extensive security infrastructure. With the use of AWS services such as Lambda, RDS, and S3, the framework offers solid scalability and flexibility, allowing small businesses to scale their security measures seamlessly as they grow, without having any

big upfront investment. Additionally, the integration with AWS CloudWatch and SNS helps with automated alerts and responses, it also reduces downtime and enhances overall system resilience. This operational efficiency is very important for small businesses, as it reduces the need for extensive IT resources and allows them to extensively focus on their core activities without maintaining servers. After automating security processes, the framework will not only minimize operational costs but also provide small businesses with the confidence to scale their operations securely and efficiently, making it an ideal solution for those looking to leverage the benefits of serverless computing without compromising on security.

6. Conclusion and Future Work

In this thesis, my primary research question was around the implementation of a robust security framework specially focused on serverless architectures, with a particular focus on the detection and mitigation of event injection attacks. My objectives were to develop a system that not only identifies the potential security threats but also provides real-time responses to prevent exploitation. To achieve this objective, I implemented different steps such as secure coding practices, detection mechanisms with the use of machine learning, and real-time monitoring with AWS CloudWatch. My work involved integrating multiple layers of security measures, such as input validation, and anomaly detection, ensuring a comprehensive approach for protecting serverless functions. The secure coding phase effectively captures major injection attacks such as SQL injection, XSS, and LDAP injection, on the other hand, the detection phase uses advanced machine learning to detect the anomalies in real-time. By integrating AWS CloudWatch, we ensured continuous monitoring and logging of suspicious activities. Concerning a research question, I successfully showed that a multi-layered security framework can significantly improve the overall security posture of serverless architectures. My key findings were that combining traditional input validation techniques with advanced anomaly detection models significantly provides more robust security against event injection attacks. The integration with AWS CloudWatch further solidified the system's ability to monitor and respond to threats in real-time. However, while our framework proved effective, it is not without limitations. The machine learning model's accuracy was highly dependent on the quality as well as the quantity of the data available for training. In real-world applications, obtaining these diverse and representative datasets will be slightly challenging.

For future work, there are a few suggestions to enhance and extend our research. One of the main areas is to create a cross-platform security framework that can be easily integrated with various cloud providers, not just AWS. This can also include adapting our system to be more flexible and compatible with different environments. On the other hand, further research could also focus on improving the machine learning algorithms by using more advanced algorithms or use of ensemble learning techniques. Another meaningful approach for future research is to use of hot topic, integration of blockchain technology for creating a decentralized, tamper-proof logging mechanism. It will help to ensure the integrity of log data and also makes it more harden for attackers to cover their tracks. For business purposes, exploring the potential for commercializing this framework as a security-as-a-service (SaaS) could also provide organizations with an accessible and scalable solution to protect their serverless architectures.

In conclusion, my project has completed all the key objectives and provided a solid foundation for securing serverless environments, By building on this work, future research can continue to advance the security of serverless architectures.

Reference

Ahmadi, S. (2024). Challenges and Solutions in Network Security for Serverless Computing. *International Journal of Current Science Research and Review*, [online] 07(01). doi https://doi.org/10.47191/ijcsrr/v7-i1-23.

Brahme, A., Deshmukh, A., Vathare, A., Patil, C. and Tarapore, Z. (2023). *DDOS Prevention System using AWS Serverless- Architecture*. [online] Available at: https://www.ijfmr.com/papers/2023/6/9563.pdf [Accessed 12 Aug. 2024].

Geeksforgeeks (2024). *Anomaly detection using Isolation Forest*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/anomaly-detection-using-isolation-forest/.

Hassan, H.B., Barakat, S.A. and Sarhan, Q.I. (2021). Survey on serverless computing. *Journal of Cloud Computing*, [online] 10(1). doi:https://doi.org/10.1186/s13677-021-00253-7.

Kim, Y., Koo, J. and Kim, U.-M. (2022). Vulnerabilities and Secure Coding for Serverless Applications on Cloud Computing. *ACM*, 3, pp.145–163. doi:https://doi.org/10.1007/978-3-031-05412-9_10.

Li, Z., Guo, L., Cheng, J., Chen, Q., He, B. and Guo, M. (2022). The Serverless Computing Survey: A Technical Primer for Design Architecture. *ACM Computing Surveys*. doi:https://doi.org/10.1145/3508360.

O'Meara, W. and Lennon, R.G. (2020). *Serverless Computing Security: Protecting Application Logic*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ISSC49989.2020.9180214.

P. Padma and Srinivasan, S. (2023). DAuth—Delegated Authorization Framework for Secured Serverless Cloud Computing. *Wireless Personal Communications*, 129(3), pp.1563–1583. doi:https://doi.org/10.1007/s11277-023-10189-7.

Prachi Tembhekar, Shanmugam, L. and Devan, M. (2023). Implementing Serverless Architecture: Discuss the practical aspects and challenges. *Journal of knowledge learning and science technology*, 2(3), pp.560–580. doi:https://doi.org/10.60087/jklst.vol2.n3.p580.

Sharaf, S. (2020). Security Issues in Serverless Computing Architecture. *International Journal of Emerging Trends in Engineering Research*, 8(2), pp.539–544. doi:https://doi.org/10.30534/ijeter/2020/43822020.

Staff, S. (2024). *Over 2M jobseekers' data compromised in SQL injection, XSS attacks*. [online] SC Media. Available at: https://www.scmagazine.com/brief/over-2m-jobseekers-data-compromised-in-sql-injection-xss-attacks.