SAVEETHA SCHOOL OF ENGINEERING

Saveetha Institute of Medical and Technical Sciences

Chennai – 602 105

DETECTING DATA LEAKS USING SQL

A CAPSTONE PROJECT REPORT

Submitted to

SIMATS ENGINEERING

By

SHAIK HANIFA (192211786)

Supervisor

Dr. V.GOKULAKRISHNAN

CSA1584 – CLOUD COMPUTING AND BIG DATA ANALYTICS FOR MEDICAL APPLICATIONS

June –2024

Table of Contents

Topic Descriptions:

Abstract

In the modern digital era, data leaks have become increasingly prevalent, posing significant risks to innocent users. These breaches can occur through various methods, such as password theft, keystroke logging by acquaintances, or ransom ware attacks. To combat these threats, this project aims to develop a cloud-based system secured with AESX encryption.

The system will leverage advanced content inspection and contextual analysis to detect data breaches effectively. By classifying users based on their online messages and behaviours, the system can identify and mitigate potential security vulnerabilities, whether intentional or unintentional. Users who engage in actions that could compromise security will be flagged, and the system will implement Data Loss Prevention (DLP) measures to restrict or take strict actions against these users.

This approach seeks to safeguard personal information on e-commerce sites, protecting both financial and mental well-being. Ultimately, the project aspires to enhance privacy and security for users, ensuring safer interactions and transactions online.

Introduction

In the contemporary digital landscape, the frequency and severity of data breaches have escalated, posing serious threats to the privacy and security of individuals. Data leaks, often resulting from stolen passwords, keystroke logging, or ransom ware attacks, can have devastating consequences for innocent users. With personal information increasingly stored and exchanged online, particularly on e-commerce platforms, the need for robust security measures is more critical than ever.

This project aims to address these security challenges by developing a sophisticated system using cloud technology, fortified with AESX encryption. The system will employ advanced techniques such as content inspection and contextual analysis to detect data breaches. By analyzing user messages and behaviours on the internet, the system will classify users and identify potential security threats. When a user's actions, whether intentional or inadvertent, pose security vulnerabilities, the system will flag these activities and implement Data Loss Prevention (DLP) measures to mitigate the risk.

The core objective of this project is to protect the privacy and security of personal information on e-commerce sites, where users frequently log in and conduct transactions. By preemptively identifying and restricting malicious activities, the system aims to prevent financial losses and safeguard users' mental well-being. This innovative approach promises to enhance online security, ensuring that users can interact and transact on digital platforms with greater confidence and safety.

Problem statement

Data breaches are increasingly compromising the privacy and security of individuals' personal information, especially on e-commerce platforms. Current security measures are often insufficient to prevent password theft, keystroke logging, and ransom ware attacks. There is a pressing need for a robust system that can effectively detect and mitigate these security threats to protect users' personal and financial information. This project seeks to develop a cloud-based solution with AESX encryption, utilizing content inspection and contextual analysis to identify and restrict malicious activities, thereby enhancing data security and user privacy.

Requirements Gathering

Requirements Gathering for Data Security and Protection System

Functional Requirements

1. **User Authentication and Authorization**:

   - Implement robust user authentication mechanisms (e.g., multi-factor authentication).

   - Define different user roles and permissions.

2. **Data Encryption**:

   - Utilize AESX encryption for data storage and transmission.

   - Ensure all sensitive user data is encrypted both at rest and in transit.

3. **Content Inspection and Contextual Analysis**:

   - Develop algorithms to inspect and analyze user messages and behaviours in real-time.

   - Implement machine learning models to classify users based on their online activities.

   - Detect suspicious activities and potential security threats based on predefined rules and patterns.

4. **Data Loss Prevention (DLP) Measures:**

   - Identify and flag messages or actions that pose security risks.

   - Implement automatic restrictions or alerts for flagged users.

   - Provide options for manual review and intervention by administrators.

5. **Incident Response and Reporting**:

   - Create mechanisms for automatic incident detection and alerting.

   - Develop a reporting system to log security incidents and user actions.

   - Provide real-time dashboards and regular reports on system security status and incidents.

6. **User Notifications:**

- Notify users of detected security threats or suspicious activities.
- Provide educational content on how to enhance personal security practices.

7. **Integration with E-commerce Platforms:**

- Ensure seamless integration with popular e-commerce platforms.
- Provide APIs for data exchange and security monitoring.

Non-Functional Requirements:

1. **Performance:**

- Ensure the system performs real-time analysis with minimal latency.
- Handle high volumes of data and user activities efficiently.

2. **Scalability:**

- Design the system to scale horizontally to accommodate growing user bases and data volumes.
- Utilize cloud resources effectively for dynamic scaling.

3. **Reliability:**

- Ensure high availability of the system with minimal downtime.
- Implement redundant systems and failover mechanisms.

4. **Security:**

- Adhere to industry-standard security practices and protocols.
- Conduct regular security audits and vulnerability assessments.
- Ensure compliance with relevant data protection regulations (e.g., GDPR, CCPA).

5. **Usability:**

- Develop a user-friendly interface for both end-users and administrators.
- Provide clear instructions and feedback for security-related actions.

6. **Maintainability:**

- Ensure the system is modular and easy to update.
- Provide comprehensive documentation for developers and administrators.

Necessary Features:

1. **User Management**:
   - User registration and profile management.
   - Role-based access control.
2. **Encryption Module**:
   - AESX encryption for data storage.
   - Secure data transmission protocols.
3. **Inspection and Analysis Engine:**
   - Real-time content inspection.
   - Contextual behaviour analysis.
4. **DLP System:**
   - Automated threat detection.
   - User activity restriction and alert mechanisms.
5. **Incident Management**:
   - Automated incident alerts.
   - Incident logging and reporting tools.
6. **User Notification System:**
   - Real-time alerts for users.
   - Educational notifications and tips.
7. **Integration APIs:**
   - APIs for integration with e-commerce platforms.
   - Data exchange and security monitoring APIs.
8. **Dashboard and Reporting:**
   - Real-time security dashboards.
   - Regular security reports for administrators.

By addressing these requirements and features, the project will create a comprehensive and effective system to protect user data and enhance security on e-commerce platforms.

Choose Cloud Provider best fits your needs

To determine the best cloud provider for the described data security and protection system, we need to evaluate several key factors, including security features, scalability, performance, integration capabilities, and cost-effectiveness. Among the leading cloud providers—Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP)—AWS appears to be the most suitable choice for this project. Below are the reasons supporting this decision:

1. Security Features:

Encryption: AWS provides comprehensive encryption solutions, including AWS Key Management Service (KMS) for managing encryption keys and AWS Cloud HSM for hardware-based key storage.

Compliance: AWS meets a wide range of security standards and compliance certifications (e.g., GDPR, CCPA, HIPAA, SOC 1/2/3, ISO 27001).

Identity and Access Management: AWS Identity and Access Management (IAM) allows for fine-grained access control and multi-factor authentication.

2. Scalability and Performance:

Elastic Compute: AWS offers Auto Scaling and Elastic Load Balancing to ensure the system can handle varying workloads efficiently.

Global Reach: AWS has a vast global infrastructure with numerous data centers, allowing for low-latency access and high availability.

Storage Solutions: AWS provides scalable storage solutions like Amazon S3 for object storage and Amazon RDS for managed relational databases, ensuring high performance and reliability.

3. Integration Capabilities:

APIs and SDKs: AWS offers extensive APIs and SDKs for seamless integration with e-commerce platforms and third-party applications.

Machine Learning Services: AWS provides services like Amazon Sage Maker for building, training, and deploying machine learning models, which can be utilized for contextual analysis and user behaviour classification.

Data Loss Prevention (DLP): AWS Macie is a managed data security and data privacy service that uses machine learning to discover, monitor, and protect sensitive data.

4. Cost-Effectiveness:

Pricing Models: AWS offers a pay-as-you-go pricing model, reserved instances, and savings plans to optimize costs based on usage patterns.

Free Tier: AWS's free tier allows for experimentation and initial development without significant upfront costs.

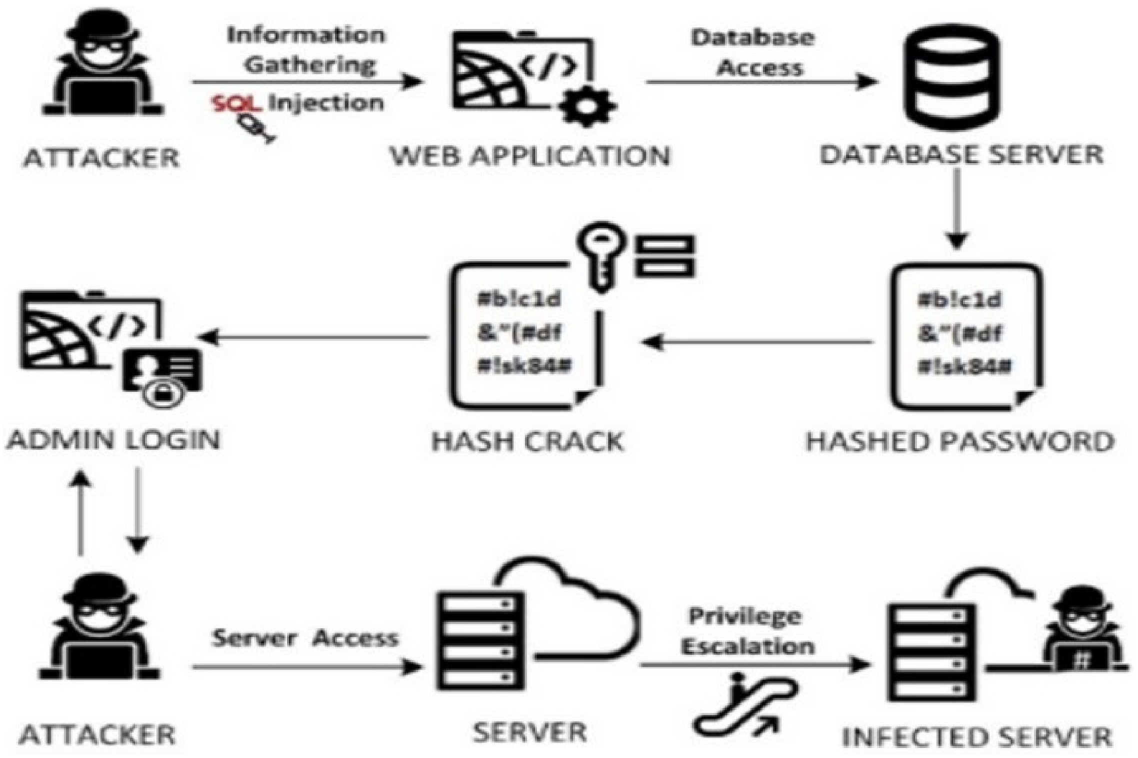
5. Support and Resources:

Comprehensive Documentation: AWS provides extensive documentation, tutorials, and training resources.

Customer Support: AWS offers various support plans, including 24/7 technical support and dedicated account managers.

Develop Frontend

1. **Layout:**



A well-structured and intuitive layout is crucial for providing an excellent user experience. The layout will be designed to ensure easy navigation and quick access to essential features.

Components of the Layout:

Header:

- Logo: Positioned at the top-left corner.
- Navigation Bar: Links to Home, Dashboard, Reports, Settings, and Help.

User Profile: A dropdown menu at the top-right corner with options like Profile, Notifications, and Logout.

Sidebar (if applicable):

- Collapsible sidebar for easy access to main sections (Dashboard, Reports, Settings, etc.).
- Icons and text labels for clarity.

Main Content Area:

- Dashboard: Display real-time security status, alerts, and key metrics.
- Reports: Section to view detailed reports on security incidents.
- Settings: User and system settings for customization and control.
- Help: Access to FAQs, support, and documentation.

Footer:

- Quick links to Privacy Policy, Terms of Service, and Contact Us.
- Social media icons for company profiles.

2. User-Friendly Design:

Creating a user-friendly design involves focusing on ease of use, accessibility, and a seamless user experience.

Key Principles:

Simplicity:

- Clean and straightforward design with minimal clutter.
- Clear and concise labels and instructions.

Consistency:

- Consistent use of fonts, colors, and button styles throughout the application.
- Standardized navigation and interaction patterns.

Accessibility:

- Ensure the design adheres to WCAG (Web Content Accessibility Guidelines).
- Provide alternative text for images and keyboard navigability.

Responsiveness:

- Ensure the application is fully responsive, functioning well on desktops, tablets, and mobile devices.
- Use flexible grid layouts and media queries for adaptability.

Feedback:

- Provide real-time feedback for user actions, such as form submissions and button clicks.
- Use notifications and alerts for important updates and errors.

3. Color Selection:

Choosing the right color scheme is essential for creating an aesthetically pleasing and functional interface. The color palette should be carefully selected to ensure readability, accessibility, and a professional look.

Color Palette:

Primary Colors:

- Blue (#007BFF): Used for primary buttons, links, and active states.
- White (#FFFFFF): Background color for a clean and spacious look.

Secondary Colors:

- Gray (#6C757D):Used for secondary buttons, text, and less prominent elements.
- Light Gray (#F8F9FA): Background for cards, modals, and input fields.

Accent Colors:

- Green (#28A745): Success messages, status indicators, and confirmation buttons.
- Red (#DC3545): Error messages, alerts, and critical status indicators.
- Yellow (#FFC107): Warning messages and notifications.

Color Usage:

Background:

- Use white or light gray for the main background to ensure content stands out.
- Light gray backgrounds for cards and modals to differentiate sections.

Text:

- Dark gray or black for primary text to ensure high readability.
- Blue for links and interactive text elements.

Buttons and Links:

- Blue for primary action buttons.
- Gray for secondary buttons.
- Use accent colors (green, red, yellow) to indicate status and feedback.

By adhering to these principles and carefully considering layout, user-friendliness, and color selection, the frontend of the data security and protection system will provide an intuitive and engaging user experience. This approach ensures that users can efficiently interact with the application and effectively manage their security needs.

Develop Backend

1. Database Implementation:

A robust and scalable database is essential for storing and managing the data related to user activities, security incidents, and system configurations. The choice of database depends on the specific requirements, such as scalability, performance, and data integrity.

Database Selection:

Primary Database: PostgreSQL (for relational data and complex queries).

Secondary Database: Amazon S3 (for storing logs and large files).

Database Schema Design:

Users Table:

user_id:UUID (Primary Key)

username:VARCHAR(255)

email:VARCHAR(255)

password_hash: VARCHAR(255)

role: VARCHAR(50) (e.g., admin, user)

created_at: TIMESTAMP

updated_at: TIMESTAMP

User_Activities Table:

activity_id: UUID (Primary Key)

user_id: UUID (Foreign Key referencing Users table)

activity_type: VARCHAR(255) (e.g., login, message_posted)

activity_data: JSONB

timestamp: TIMESTAMP

Security_Incidents Table:

incident_id: UUID (Primary Key)

user_id: UUID (Foreign Key referencing Users table)

incident_type: VARCHAR(255) (e.g., SQL injection attempt, XSS attack)

incident_details: TEXT

status: VARCHAR(50) (e.g., detected, resolved)

detected_at: TIMESTAMP

resolved_at: TIMESTAMP (nullable)

Settings Table:

setting_id: UUID (Primary Key)

setting_name: VARCHAR(255)

setting_value: TEXT

Logs Table (stored in Amazon S3):

log_id: UUID (Primary Key)

user_id: UUID (Foreign Key referencing Users table)

log_data: JSONB

created_at: TIMESTAMP

2. Execution:

Backend Framework and Environment:

Framework: Node.js with Express.js for the web server

Environment: AWS (Amazon Web Services) for deployment and infrastructure

API Design:

Authentication and Authorization Endpoints:

POST /api/auth/register: User registration

POST /api/auth/login: User login

POST /api/auth/logout: User logout

User Management Endpoints:

GET /api/users: Retrieve a list of users (admin only)

GET /api/users/:id: Retrieve a specific user

PUT /api/users/:id: Update a user’s information

DELETE /api/users/:id: Delete a user

User Activity Endpoints:

GET /api/activities: Retrieve user activities

POST /api/activities: Log user activity

Security Incident Endpoints:

GET /api/incidents: Retrieve security incidents

GET /api/incidents/:id: Retrieve a specific incident

POST /api/incidents: Report a security incident

PUT /api/incidents/:id: Update incident status

Settings Endpoints:

GET /api/settings: Retrieve system settings

PUT /api/settings/:id: Update a system setting

Logs Endpoint:

GET /api/logs: Retrieve logs (admin only)

Core Services and Modules:

Authentication Service:

- Implement JWT (JSON Web Token) for secure token-based authentication.
- Use bcrypt for password hashing.

User Service:

- Handle CRUD operations for user data.
- Ensure role-based access control.

Activity Logging Service:

- Log user activities in real-time.
- Store logs in PostgreSQL and Amazon S3.

Security Incident Service:

- Detect and record security incidents.
- Analyze user activities for suspicious patterns using machine learning models.
- Provide an interface for administrators to review and resolve incidents.

Notification Service:

- Send email or push notifications to users and administrators for important events (e.g., security incidents, account changes).

Deployment and Monitoring:

CI/CD Pipeline:

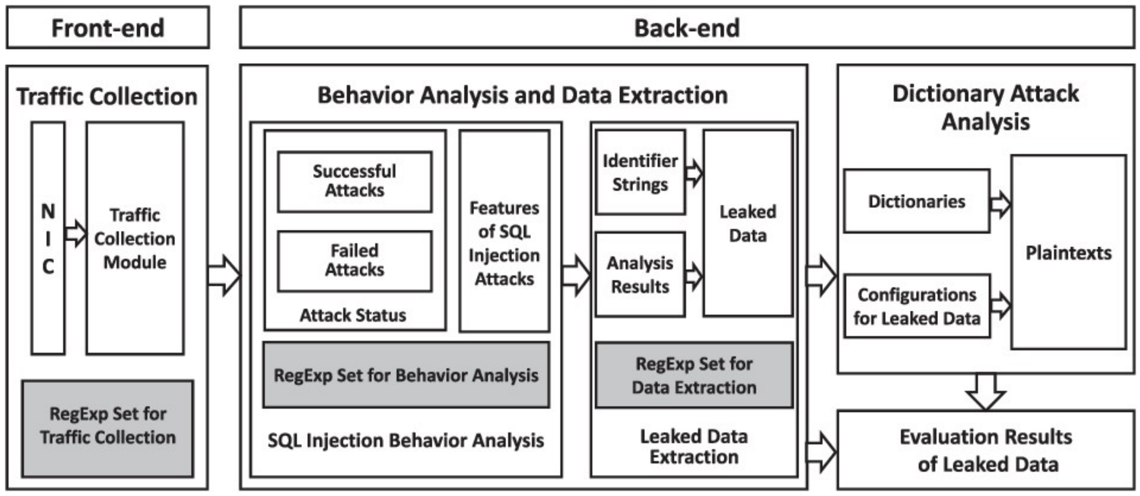
- Use AWS Code Pipeline for continuous integration and deployment.
- Implement automated testing with Jest and integration tests with Postman.

Monitoring and Logging:

- Use AWS Cloud Watch for monitoring application performance and logging.
- Set up alerts for critical events (e.g., high error rates, downtime).

Scalability and Load Balancing:

- Deploy the application using AWS Elastic Beanstalk for easy scaling and management.
- Use AWS Elastic Load Balancer (ELB) to distribute traffic across multiple instances.


Front-end
Back-end
Traffic Collection
N I C
Traffic Collection Module
RegExp Set for Traffic Collection
Behavior Analysis and Data Extraction
Successful Attacks
Failed Attacks
Attack Status
Features of SQL Injection Attacks
RegExp Set for Behavior Analysis
SQL Injection Behavior Analysis
Identifier Strings
Analysis Results
Leaked Data
RegExp Set for Data Extraction
Leaked Data Extraction
Dictionary Attack Analysis
Dictionaries
Configurations for Leaked Data
Plaintexts
Evaluation Results of Leaked Data

By following this backend development plan, we ensure a secure, scalable, and efficient system capable of protecting user data and mitigating security threats. This approach leverages the strengths of AWS infrastructure and modern development practices to create a robust backend for the data security and protection system.

Implementation and Integrate with Cloud Services

1. Implementation:

Backend Implementation Steps:

Set Up Development Environment:

- Install Node.js and required dependencies.
- Initialize a new Node.js project using npm or yarn.

Develop Backend Components:

Database Setup: Configure Postgre SQL database locally or on AWS RDS.

Backend Framework: Use Express.js to create RESTful APIs.

Authentication: Implement JWT-based authentication using bcrypt for password hashing.

API Endpoints: Develop endpoints for user management, activity logging, security incident reporting, settings management, and logging.

Services: Implement core services such as authentication service, user service, activity logging service, security incident service, and notification service.

Integrate with External Services:

Email Service: Integrate with AWS SES (Simple Email Service) for sending email notifications.

Storage Service: Use AWS S3 for storing logs and large files securely.

Frontend Implementation Steps:

Set Up Frontend Development Environment:

- Install necessary tools like Node.js, npm or yarn, and a frontend framework like React.js or Angular.

Design and Develop UI Components:

- Create UI components for the dashboard, reports, settings, and user profile using HTML, CSS (or SCSS), and JavaScript (or TypeScript).
- Implement responsive design for various devices.

Integrate with Backend APIs:

- Use Axios or Fetch API to communicate with backend APIs.
- Implement state management (e.g., Redux for React) for managing application state.

2. Cloud Integration and Deployment:

AWS Integration and Deployment:

Database Deployment: Set up Postgre SQL database instance on AWS RDS.

Backend Deployment:

- Use AWS Elastic Beanstalk for deploying Node.js application.
- Configure environment variables for secure handling of sensitive information (e.g., database credentials, API keys).
- Implement continuous integration and deployment (CI/CD) pipeline using AWS Code Pipeline and AWS Code Build.

Frontend Deployment:

- Deploy frontend application using AWS Amplify or AWS S3 for static hosting.
- Set up CDN (Content Delivery Network) for improved performance and global accessibility.

3. Testing:

Backend Testing:

Unit Testing: Write unit tests using Jest for testing individual functions and modules.

Integration Testing: Use Postman or Newman for testing API endpoints and integration flows.

Security Testing: Perform security audits and vulnerability assessments to ensure robust security measures.

Frontend Testing:

UI Testing: Conduct UI testing to ensure consistency and usability across different browsers and devices.

Integration Testing: Test frontend components with backend APIs using mock data or stubs.

Performance Testing:

Load Testing: Use tools like Apache JMeter or AWS Load Testing tools to simulate heavy traffic and measure system performance.

Stress Testing: Evaluate system behaviour under peak loads to identify potential bottlenecks and optimize performance.

Deployment Testing:

Deployment Validation: Conduct smoke tests after each deployment to verify application functionality and stability.

Rollback Procedures: Define rollback procedures in case of deployment failures or critical issues.

By following these implementation and integration steps, leveraging AWS cloud services, and conducting comprehensive testing, the data security and protection system can be effectively developed, deployed, and maintained to ensure robust performance and security for users.

Performance Evaluation

Performance evaluation is crucial to ensure that the data security and protection system meets the expected level of responsiveness, scalability, and reliability. Here’s a structured approach to evaluating the performance of the implemented system:

1. Response Time:

Objective: Measure the time taken for the system to respond to user requests.

Tools: Use monitoring tools like AWS Cloud Watch metrics or application performance monitoring (APM) tools (e.g., New Relic, Data dog).

Metrics: Monitor average response time, peak response time, and response time distribution (e.g., 95th percentile) for critical endpoints (e.g., login, data retrieval).

2. Throughput:

Objective: Measure the number of requests processed per unit of time to assess system capacity.

Tools: Use load testing tools like Apache JMeter, AWS Load Testing tools, or custom scripts.

Metrics: Evaluate maximum throughput achieved under load, identify performance bottlenecks (e.g., database queries, API rate limits), and estimate peak capacity.

3. Scalability:

Objective: Assess the system's ability to handle increasing workload by scaling resources.

Tools: Conduct load tests with varying levels of concurrent users and transactions.

Metrics: Measure system performance as resources (e.g., CPU, memory) scale up or down dynamically using AWS Auto Scaling or manual adjustments.

4. Resource Utilization:

Objective: Monitor resource consumption (CPU, memory, disk I/O) to optimize efficiency and cost-effectiveness.

Tools: Use AWS Cloud Watch metrics, APM tools, or database performance monitoring tools (e.g., Amazon Cloud Watch, Prometheus).

Metrics: Track resource utilization during peak load periods, identify inefficiencies or underutilized resources, and optimize configuration (e.g., database indexes, caching strategies).

5. Error Rates and Availability:

Objective: Measure the frequency of errors and system availability to ensure reliability.

Tools: Monitor error logs, set up alerts for critical errors using AWS Cloud Watch alarms.

Metrics: Calculate error rates (e.g., HTTP 5xx status codes) during load testing and production, ensure high availability targets (e.g., 99.9% uptime), and implement failover mechanisms for fault tolerance.

6. Security Performance:

Objective: Evaluate the performance impact of security measures (e.g., encryption, authentication) on system responsiveness.

Metrics: Measure encryption/decryption overhead, authentication/authorization latency, and impact on overall system performance during peak loads.

7. Database Performance:

Objective: Assess database performance to ensure efficient data storage and retrieval.

Tools: Use database monitoring tools (e.g., AWS RDS Performance Insights, Postgre SQL pg_stat_statements extension).

Metrics: Monitor database query execution times, analyze slow queries, optimise database schema and indexing strategies to improve performance.

8. Continuous Monitoring and Optimization:

Objective: Implement continuous monitoring of performance metrics to identify trends, anomalies, and opportunities for optimization.

Tools: Use automated monitoring tools, establish performance baselines, and conduct periodic performance reviews.

Metrics: Compare performance metrics over time, implement optimizations (e.g., code refactoring, infrastructure tuning), and validate improvements through retesting.

Conclusion

Performance evaluation ensures that the data security and protection system operates efficiently under varying conditions, meets user expectations for responsiveness and reliability, and supports scalability as user demands grow. By systematically measuring and optimizing performance metrics, organizations can enhance the system’s effectiveness in safeguarding sensitive data and maintaining operational excellence.