Enterprise Secure Development Lifecycle and Application Security Governance

1. Problem Statement

A retail technology company deploying cloud-native applications lacked formal security integration within its software development lifecycle. Security checks were performed postdeployment, increasing risk and rework. The organization required secure-by-design development practices integrated into CI/CD pipelines

I implemented this framework to integrate security into application design, development, deployment, and operations through standardized guidelines, architecture reviews, exception management, and developer training.

The objective was to reduce application security risks and improve overall security maturity.

2. Objectives

The primary objectives were to:

- Establish secure design and coding standards
- Prevent common application vulnerabilities
- Enforce least-privileged access
- Improve architecture security reviews
- Manage security exceptions effectively
- Increase developer security awareness
- Integrate security into the SDLC

3. Secure Design Principles

3.1 Least Privilege

Systems, services, and users were granted only the minimum permissions required.

3.2 Defense in Depth

Multiple layers of security controls were implemented across infrastructure and applications.

3.3 Secure by Default

Security controls were enabled by default in all environments.

4. Secure Coding Standards

4.1 Input Validation

All user inputs are validated and sanitized to prevent injection attacks.

4.2 Authentication and Authorization

OAuth 2.0 and OpenID Connect were implemented for authentication.

Role-based authorization was enforced.

4.3 Password Management

Passwords are hashed using bcrypt.

No plaintext passwords are stored.

4.4 Session Management

Sessions expire after inactivity.

Secure cookies and HTTPS are enforced.

5. Data Protection Controls

Sensitive data is encrypted:

- At rest using AWS KMS
- In transit using TLS

Key management policies were documented and enforced.

6. OWASP Top 10 Risk Management

Development teams were trained on OWASP Top 10 vulnerabilities, including:

- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Broken Authentication
- Insecure Deserialization

Mitigation controls were integrated into development practices.

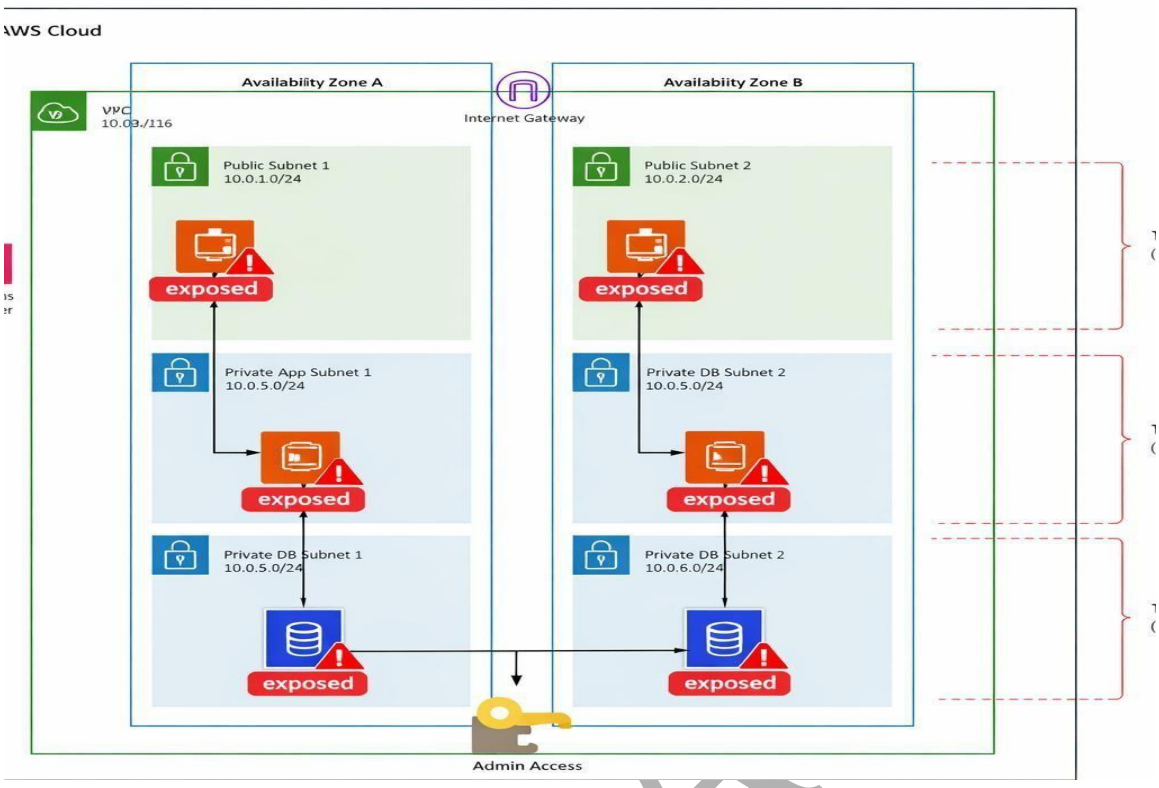
7. Architecture Security Review Process

7.1 Pre-Deployment Review

All proposed architectures were reviewed before deployment.

Initial reviews identified:

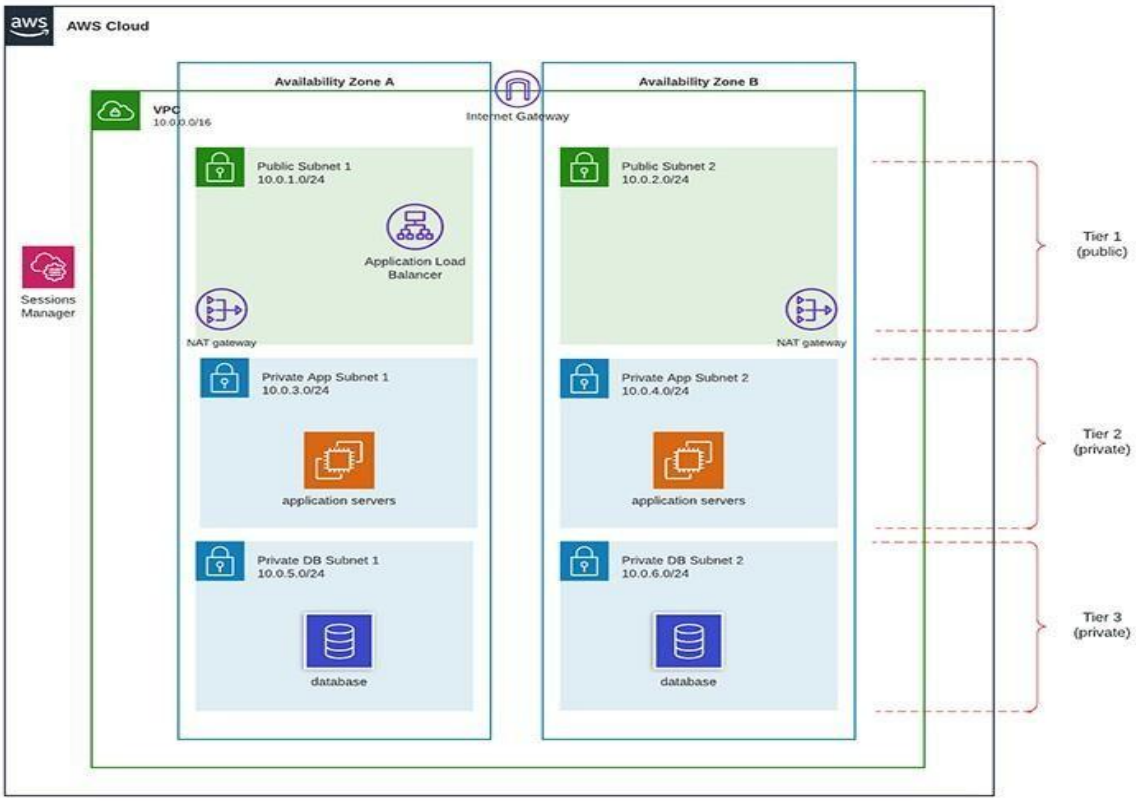
- Public EC2 exposure
- Excessive IAM permissions
- Missing WAF protection



7.2 Remediation Actions

Identified risks were remediated by:

- Moving workloads to private subnets
- Enforcing least privilege IAM roles
- Enabling AWS WAF
- Implementing network segmentation



7.3 Post-Review Validation

Improved architectures were validated to confirm risk reduction.

8. Security Exception Management

8.1 Exception Review Process

Formal procedures were established for handling security exceptions.

Requests included business justification and risk analysis.

8.2 MFA Exception Case

A temporary MFA exception was approved for a legacy system.

Controls implemented:

- IP restrictions
- Enhanced logging
- Time-limited access

Exceptions were reviewed quarterly.

9. Developer Security Training Program

9.1 Training Overview

Regular knowledge-sharing sessions were organized.

9.2 Training Objectives

- Improve secure coding practices
- Increase cloud security awareness
- Promote early security engagement

9.3 Topics Covered

- OWASP Top 10
- Secure API design
- AWS security best practices
- Incident response
- Secrets management

9.4 Training Materials

- Slide decks
- Developer cheat sheets
- Reference guides

9.5 Outcomes

Training improved compliance and reduced recurring security issues.

10. Governance and Documentation

All standards, review procedures, exception decisions, and training materials were documented.

This supported audits, consistency, and knowledge transfer.

11. Outcomes and Impact

This implementation delivered:

- Improved application security posture
- Reduced design-time vulnerabilities
- Strong SDLC security integration
- Better risk management
- Increased developer accountability
- Enhanced governance

12. Conclusion

I designed and implemented an enterprise-grade secure development and application security governance framework.

Through standardized guidelines, architecture reviews, exception management, and continuous training, this solution embeds security into the software lifecycle and strengthens organizational resilience.