encoresky

SECURITY BEST PRACTICES

Mobile Apps, Backend Systems, & Websites

encoresky

Table of contents

1. Introduction

In today’s digital age, security breaches are more frequent and sophisticated than ever. Developers must prioritize robust security measures to protect sensitive data and build user trust. This e-book outlines essential practices to secure mobile applications, backend infrastructures and websites, covering common vulnerabilities, preventive strategies and advanced tips to ensure a robust system.

2. MOBILE APP SECURITY

2.1 - Secure Authentication

2.2 - Data Protection

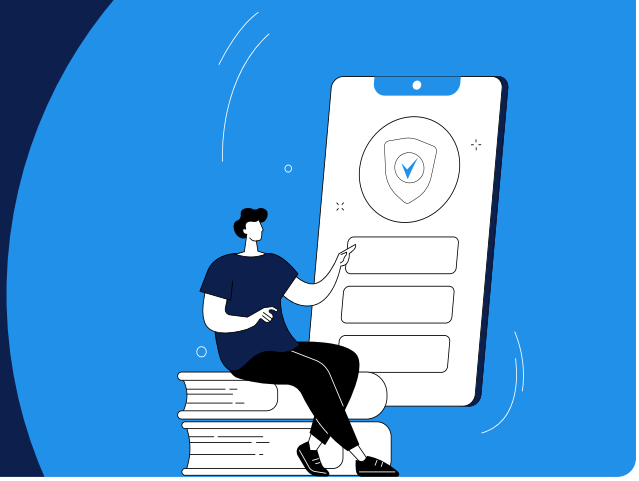
2.3 - API Security

2.4 - Secure Code Practices

2.5 - Secure Communications

2.6 - Device Security Considerations

Mobile App Security



2.1 Secure Authentication

- **Multi-Factor Authentication (MFA):**

  Go beyond traditional passwords by requiring a second authentication factor like biometrics (fingerprint or face recognition) or OTPs.

- **Token-Based Authentication:**

  Use JWTs (JSON Web Tokens) or similar tokens, stored securely in device-provided mechanisms like Keychain (iOS) or Keystore (Android).

- **Session Expiry:**

  Ensure sessions have timeouts to prevent unauthorized access if a device is left unattended.

2.2 Data Protection

- **Encryption:**

  Use AES-256 for encrypting sensitive data stored on devices. For data in transit, enforce TLS 1.2 or above.

- **Local Storage:**

  Avoid storing sensitive data like passwords or tokens in plaintext or insecure storage like Shared Preferences, User defaults, async storage etc. Use encrypted storage options.

- **Data Minimization:**

  Collect and store only what is absolutely necessary. Keeping the important keys and elements on cloud like AWS Secrets Manager.

2.3 API Security

- **API Gateway Security:**

  Apply rate limiting, IP whitelisting, and geo-blocking where applicable.

- **Authorization**

  Use role-based access controls (RBAC) to restrict access to sensitive data and functionalities. Ensure every API endpoint requires authentication.

- **Validation:**

  Validate all client-sent data server-side to prevent injection and tampering.

2.4 Secure Code Practices

- **Code Obfuscation:**

  Use tools like ProGuard or R8 to make reverse engineering difficult.

- **Secrets Management:**

  Never hardcode API keys or sensitive information in your code. Use environment variables or secure vault services like AWS Secrets Manager.

- **Third-Party Libraries:**

  Regularly update and audit libraries to patch vulnerabilities.

2.5 Secure Communications

- **SSL Pinning:**

  Pin certificates to your app to mitigate man-in-the-middle attacks.

- **DNS Security:**

  Use DNSSEC to prevent DNS spoofing.

2.6 Device Security Considerations

- **Jailbreak/Root Detection:**

  Implement mechanisms to detect compromised devices and limit app functionality if detected.

- **Secure Permissions:**

  Request only necessary device permissions and explain their use to the user clearly.

encoresky

3. BACKEND SECURITY

3.1 - Authentication and Authorization

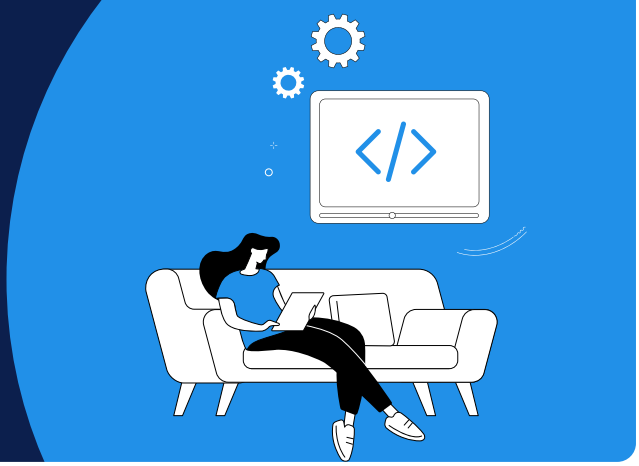
3.2 - Secure Database Practices

3.3 - Server Security

3.4 - Logging and Monitoring

3.5 - API Hardening

Backend Security



3.1 Authentication and Authorization

- **OAuth2.0/OpenID Connect:**

  Use these standards for secure user authentication and token exchange.

- **RBAC & ABAC:**

  Implement role-based access control (RBAC) or attribute-based access control (ABAC) to ensure least privilege.

3.2 Secure Database Practices

- **Injection Prevention:**

  Use parameterized queries and ORM frameworks to guard against SQL injection.

- **Encryption:**

  Encrypt sensitive fields such as passwords (use bcrypt or Argon2) and sensitive data like PII or financial details.

- **Backup Security:**

  Encrypt database backups and store them securely in restricted access locations.

3.3 Server Security

- **Firewall Configuration:**

  Use firewalls to monitor and control incoming and outgoing traffic.

- **Access Controls:**

  Restrict SSH and administrative access to trusted IPs. Use key-based authentication instead of passwords for SSH.

- **Container Security:**

  If using Docker or Kubernetes, scan container images for vulnerabilities and limit container permissions.

3.4 Logging and Monitoring

- **Centralized Logging:**

  Use centralized logging solutions like ELK Stack, Splunk or AWS CloudWatch to aggregate logs.

- **Anomaly Detection:**

  Implement tools to detect unusual behaviors like multiple failed login attempts or unauthorized access patterns.

3.5 API Hardening

- **Rate Limiting and Throttling:**

  Prevent abuse by setting strict request limits on sensitive endpoints.

- **Error Handling:**

  Avoid exposing sensitive information through error messages.

4. WEBSITE SECURITY

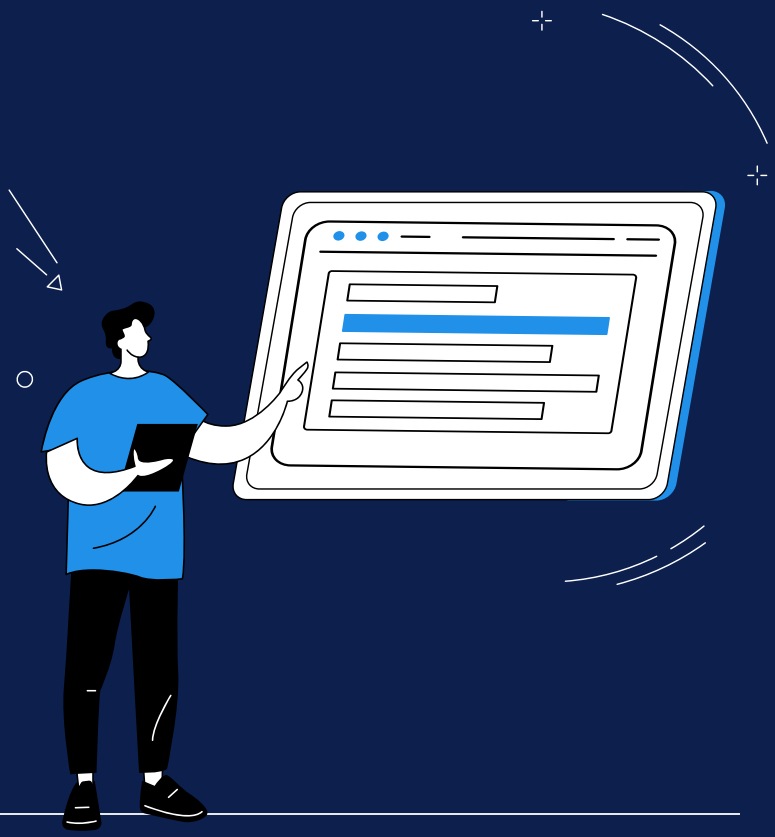
4.1 - Input Validation and Sanitization

4.2 - Secure File Handling

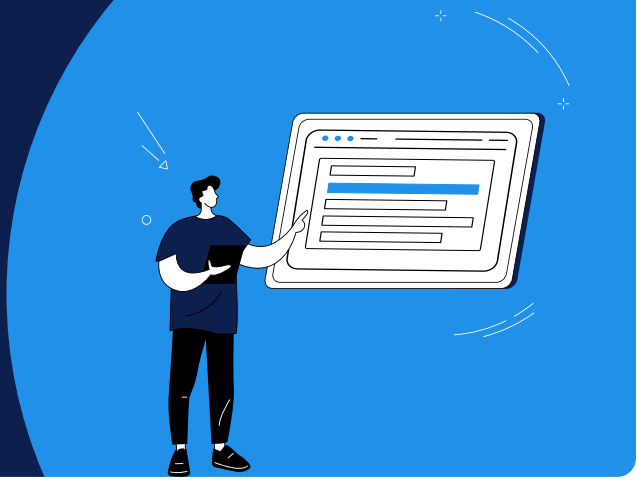
4.3 - Secure Cookies and Sessions

4.4 - Protection Against Attacks

4.5 - HTTPS and Certificates



Website Security



4.1 Input Validation and Sanitization

- **Whitelist Input Validation:**

  Define strict rules for input values and reject anything outside the acceptable range.

- **Output Encoding:**

  Sanitize outputs to prevent HTML or JavaScript injection.

4.2 Secure File Handling

- **File Validation:**

  Validate file type, size, and content before accepting uploads.

- **Non-Executable Storage:**

  Store uploaded files in a directory where execution is disabled to prevent malicious script execution.

4.3 Secure Cookies and Sessions

- **Cookie Flags:**

  Use HttpOnly, Secure, and SameSite flags on cookies to prevent unauthorized access and cross-site attacks.

- **Session Management:**

  Use unique session tokens with short expiration times and regularly regenerate them.

4.4 Protection Against Attacks

- **Anti-CSRF Tokens:**

  Implement anti-CSRF mechanisms to secure state-changing requests.

- **Content Security Policy (CSP):**

  Define a CSP to restrict script execution from untrusted sources.

- **CAPTCHA**

  Prevent automated bots from abusing your website with CAPTCHAs

4.5 HTTPS and Certificates

- **TLS Everywhere:**

  Enforce HTTPS with HSTS headers to secure all traffic.

- **Regular Renewals:**

  Ensure SSL/TLS certificates are valid and up to date.

5. COMMON VULNERABILITIES AND THEIR PREVENTION

5.1 - Authentication Vulnerabilities

5.2 - Exposed Source Code or Configurations

5.3 - XSS (Cross-Site Scripting)

5.4 - SQL Injection

5.5 - Denial-of-Service (DoS)

Common Vulnerabilities & Their Prevention

5.1 Authentication Vulnerabilities

- **OTP Bypass:**

  Validate OTPs on the server and use secure random generators.

- **Password Policy:**

  Enforce strong password policies with complexity requirements.

5.2 Exposed Source Code or Configurations

- **Prevent Directory Exposure:**

  Configure servers to block access to .git directories or .env files.

- **Static Code Analysis:**

  Use tools like SonarQube to detect secrets or misconfigurations in codebases.

5.3 XSS (Cross-Site Scripting)

- **Input Sanitization:**

  Use libraries to sanitize and encode input data.

- **CSP Enforcement:**

  Block inline scripts with a strict CSP

5.4 SQL Injection

- **Prepared Statements:**

  Always use parameterized queries.

- **Database Permissions:**

  Limit database access roles to only what the application needs.

5.5 Denial-of-Service (DoS)

- **Auto-Scaling Infrastructure:**

  Use cloud-based tools like AWS Auto Scaling to handle traffic spikes.

- **Load Balancers:**

  Distribute traffic and use WAF rules to block malicious requests.

6. ADVANCED SECURITY PRACTICES

6.1 - Zero Trust Architecture:

6.2 - Threat Modeling:

6.3 - Secure CI/CD Pipelines:

6.4 - Bug Bounty Programs:

Advanced Security Practices

6.1 Zero Trust Architecture:

Assume no component is secure and enforce continuous verification.

6.2 Threat Modeling:

Regularly assess potential attack vectors using frameworks like STRIDE.

6.3 Secure CI/CD Pipelines:

Use security scans and approvals in CI/CD pipelines to detect vulnerabilities before production.

6.4 Bug Bounty Programs:

Additionally engage ethical hackers to uncover vulnerabilities through controlled programs.

7. Conclusion

A secure application is the result of careful planning, proactive measures and continuous improvement. By adopting the strategies and best practices outlined in this guide, developers can build robust systems that safeguard user data and maintain trust

encoresky