Secure Software Engineering for Mobile Apps

Practical Exercise – Security Architecture, Threat Modeling, and Defense Mechanisms

Introduction to Mobile Application Security

Course: Mobile Application Development (Chapter 11)
Level: Technical University Students
Duration: 9–11 hours self-study + 2–2.5 hours online discussion
Format: Individual Assignment with Security Implementation (7–9 pages)

Exercise Overview

In this exercise, you will master comprehensive security engineering practices for mobile applications: understanding fundamental security principles and threat modeling methodologies, implementing secure coding practices and input validation, deploying encryption and authentication mechanisms, securing sensitive data storage with secure enclaves, mitigating common vulnerabilities including SQL injection and XSS attacks, protecting against reverse engineering and code tampering, and achieving compliance with industry standards like OWASP Mobile Security Testing Guideline (MASTG) and OWASP Top 10 Mobile Risks.

Key Learning Outcomes

- Understand security fundamentals: confidentiality, integrity, authenticity, non-repudiation
- Master threat modeling methodologies: STRIDE, PASTA, attack trees
- Conduct comprehensive security risk assessment for mobile applications
- Implement secure coding practices and input validation mechanisms
- Design and deploy encryption strategies (symmetric, asymmetric, hashing)
- Implement authentication protocols: OAuth 2.0, OIDC, mutual TLS, certificate pinning
- Secure sensitive data storage: keystore/Keychain, encrypted databases, secure preferences

- Mitigate common vulnerabilities: SQL injection, XSS, CSRF, insecure deserialization
- Protect against reverse engineering, code tampering, and debugger attacks
- Implement secure communication: TLS/SSL, certificate validation, API security
- Conduct security testing: static analysis, dynamic analysis, penetration testing
- Achieve compliance with OWASP MASTG, GDPR, and HIPAA standards
- Design and implement security monitoring and incident response

---

Part A: Security Fundamentals and Threat Modeling

Task 1: Security Principles and Threat Modeling (Self-Study: 2.5–3 hours)

Objective: Master security fundamentals and comprehensive threat modeling methodologies for mobile applications.

Instructions:

1. Core Security Principles

Confidentiality (Privacy):

- Ensure only authorized parties access sensitive information
- Examples: Encryption of user data, secure authentication, access controls
- Mobile context: User credentials, payment data, location information
- Implementation: End-to-end encryption, secure storage, access logging

Integrity (Accuracy):

- Ensure data has not been altered or corrupted during transmission or storage
- Detection: Message authentication codes (MAC), digital signatures, checksums
- Mobile context: API responses, stored data, code verification
- Implementation: HMAC validation, signature verification, tamper detection

Authenticity (Identity Verification):

- Verify the identity of users, servers, and data sources
- Methods: Digital certificates, OAuth tokens, mutual authentication
- Mobile context: Server authentication, user authentication, device authentication
- Implementation: TLS certificates, token-based auth, device identifiers

Non-Repudiation (Accountability):

- Ensure parties cannot deny their actions
- Methods: Digital signatures, audit logs, cryptographic proofs
- Mobile context: Transaction history, user actions, security events
- Implementation: Signed transactions, immutable logs, timestamped records

Authorization (Access Control):

- Ensure users can only access resources they have permission for
- Methods: Role-based access control (RBAC), attribute-based access control (ABAC)
- Mobile context: Feature access, data visibility, permission management
- Implementation: Permission frameworks, token scopes, role validation

2. Threat Modeling Methodologies

STRIDE Model (Microsoft):

Threat Type	Property	Examples	Mitigation
Spoofing Identity	Authenticity	Fake server, impersonation, forged credentials	Authentication, TLS, certificates
Tampering with Data	Integrity	Modified requests, altered storage, code injection	Signing, encryption, checksums
Repudiation	Non-repudiation	Deny actions, false claims, deleted logs	Audit logs, signatures, timestamps
Information Disclosure	Confidentiality	Data leaks, man-in-the-middle, storage access	Encryption, access controls, HTTPS
Denial of Service	Availability	Crash attacks, resource exhaustion, rate limits	Input validation, rate limiting, monitoring
Elevation of Privilege	Authorization	Bypass auth, privilege escalation, admin access	Access controls, input validation, least privilege

PASTA Methodology (Process for Attack Simulation and Threat Analysis):

1. **Stage 1: Define Objectives** - Identify security goals and organizational risk tolerance
2. **Stage 2: Define Technical Scope** - Map application architecture and data flows
3. **Stage 3: Decompose Application** - Break down into components, identify trust boundaries
4. **Stage 4: Analyze Threats** - Enumerate potential attacks against components
5. **Stage 5: Analyze Vulnerabilities** - Identify exploitable weaknesses

6. **Stage 6: Analyze Countermeasures** - Map existing and proposed security controls
7. **Stage 7: Risk Analysis** - Calculate likelihood × impact = risk score

Attack Trees (Visual Threat Analysis):

```
    Goal: Steal User Credentials
    /                         \
   /                           \
```

Intercept Login Capture from Device
(Network Attack) (Local Attack)
/ \ /
MITM Packet Malware Memory Storage
Sniff Capture Injection Dump Access

3. Android Threat Model Example

// Mobile app threat analysis structure
data class ThreatModel(
val threatId: String,
val threatName: String,
val category: STRIDECategory,
val affectedAssets: List<String>,
val likelihood: RiskLevel, // Low, Medium, High, Critical
val impact: RiskLevel,
val mitigations: List<String>,
val status: MitigationStatus // Open, Mitigated, Accepted
)

enum class STRIDECategory {
SPOOFING, TAMPERING, REPUDIATION, INFO_DISCLOSURE,
DENIAL_OF_SERVICE, ELEVATION_OF_PRIVILEGE
}

enum class RiskLevel { LOW, MEDIUM, HIGH, CRITICAL }

// Example threats for mobile banking app
val bankingThreats = listOf(
ThreatModel(
threatId = "T001",
threatName = "Attacker intercepts login credentials via MITM attack",
category = STRIDECategory.SPOOFING,
affectedAssets = listOf("User Credentials", "Authentication Server"),
likelihood = RiskLevel.HIGH,
impact = RiskLevel.CRITICAL,
mitigations = listOf(
"Implement certificate pinning",
"Use TLS 1.2+ with strong ciphers",
"Mutual TLS authentication"
),
status = MitigationStatus.MITIGATED
),

```
ThreatModel(
threatId = "T002",
threatName = "Attacker dumps memory to extract encryption keys",
category = STRIDECategory.INFO_DISCLOSURE,
affectedAssets = listOf("Encryption Keys", "User Data"),
likelihood = RiskLevel.MEDIUM,
impact = RiskLevel.CRITICAL,
mitigations = listOf(
"Use Android Keystore for key storage",
"Clear sensitive data from memory immediately after use",
"Prevent debugger attachment"
),
status = MitigationStatus.MITIGATED
),
ThreatModel(
threatId = "T003",
threatName = "Attacker reverses APK to extract hardcoded secrets",
category = STRIDECategory.INFO_DISCLOSURE,
affectedAssets = listOf("API Keys", "Hardcoded Credentials"),
likelihood = RiskLevel.HIGH,
impact = RiskLevel.HIGH,
mitigations = listOf(
"Never hardcode secrets",
"Use obfuscation (ProGuard/R8)",
"Implement code integrity checks"
),
status = MitigationStatus.MITIGATED
)
)

// Risk calculation
fun calculateRiskScore(threat: ThreatModel): Float {
val likelihoodScore = threat.likelihood.ordinal + 1 // 1-4
val impactScore = threat.impact.ordinal + 1 // 1-4
return likelihoodScore * impactScore.toFloat() // 1-16
}

// Risk visualization
class RiskMatrix {
fun plotRisks(threats: List<ThreatModel>) {
val criticalThreats = threats.filter {
calculateRiskScore(it) >= 12 // High priority
}
```

```
    // Prioritize remediation efforts
    criticalThreats.sortedByDescending { calculateRiskScore(it) }
        .forEach { threat ->
            println("CRITICAL: ${threat.threatName}")
            println("Mitigations: ${threat.mitigations}")
        }
}
```

```
}
```

4. iOS Security Threat Model

```
// iOS threat modeling structure
struct ThreatModel {
let threatId: String
let threatName: String
let category: STRIDECategory
let affectedAssets: [String]
let likelihood: RiskLevel
let impact: RiskLevel
let mitigations: [String]
let status: MitigationStatus
}

enum STRIDECategory {
case spoofing
case tampering
case repudiation
case infoDisclosure
case denialOfService
case elevationOfPrivilege
}

enum RiskLevel { case low, medium, high, critical }

// iOS-specific threats
let iosThreats: [ThreatModel] = [
ThreatModel(
threatId: "IOS-001",
threatName: "App's Keychain accessed by other apps",
category: .infoDisclosure,
affectedAssets: ["Credentials", "Encryption Keys"],
likelihood: .low, // iOS sandboxing strong
impact: .critical,
mitigations: [
"Use Keychain with kSecAttrAccessible",
"Implement app-specific encryption",
"Regular security audits"
],
status: .mitigated
),
ThreatModel(
threatId: "IOS-002",
threatName: "Man-in-the-middle attack via SSL bypass",
category: .spoofing,
affectedAssets: ["Network Communications"],
likelihood: .medium,
impact: .critical,
mitigations: [
"Implement certificate pinning",
"Disable NSURLPROTOCOL override",
"Use URLSession with TLS 1.2+"
],
status: .mitigated
```

)
]

func calculateRisk(likelihood: RiskLevel, impact: RiskLevel) -> Int {
let l = ["low": 1, "medium": 2, "high": 3, "critical": 4]
return l[likelihood]! * l[impact]!
}

Deliverable:

- [ ] Security principles explanation with real-world mobile examples
- [ ] STRIDE model threat enumeration for mobile app (6+ threats)
- [ ] PASTA methodology walkthrough with 7 stages
- [ ] Attack tree diagram for specific attack scenario
- [ ] Risk matrix with likelihood vs. impact analysis
- [ ] Threat model implementation (Android or iOS) with risk scoring

---

Task 2: Security Risk Assessment and Compliance (Self-Study: 2–2.5 hours)

Objective: Master security risk assessment methodologies and industry compliance standards.

Instructions:

1. Security Risk Assessment Framework

// Comprehensive security risk assessment
class SecurityRiskAssessment {

```
// Step 1: Asset Identification
data class Asset(
    val assetId: String,
    val name: String,
    val type: AssetType,        // Data, Code, Infrastructure
    val criticality: Criticality // Low, Medium, High, Critical
)

enum class AssetType { USER_DATA, CODE, INFRASTRUCTURE, CREDENTIALS }
enum class Criticality { LOW, MEDIUM, HIGH, CRITICAL }

// Step 2: Vulnerability Identification
data class Vulnerability(
    val vulnId: String,
    val name: String,
    val cwId: String,           // CWE number
    val severity: Severity,
```

```kotlin
    val description: String,
    val affectedAssets: List<String>
)

enum class Severity { LOW, MEDIUM, HIGH, CRITICAL }

// Step 3: Risk Calculation
data class Risk(
    val riskId: String,
    val vulnerability: Vulnerability,
    val asset: Asset,
    val likelihood: Float,      // 0.0-1.0
    val businessImpact: Float,  // 0.0-1.0
    val riskScore: Float = likelihood * businessImpact
)

// Step 4: Mitigation Planning
data class MitigationControl(
    val controlId: String,
    val controlName: String,
    val controlType: ControlType,
    val residualRisk: Float,    // After mitigation
    val cost: Int,              // Implementation cost
    val timeline: String
)

enum class ControlType {
    PREVENTIVE,    // Stop attack before it happens
    DETECTIVE,     // Detect attack during/after
    CORRECTIVE     // Fix damage after attack
}

// Example: Assess SQL injection risk
fun assessSQLInjectionRisk(): Risk {
    val sqliVuln = Vulnerability(
        vulnId = "V001",
        name = "SQL Injection",
        cwId = "CWE-89",
        severity = Severity.CRITICAL,
        description = "Unsanitized user input in database queries",
        affectedAssets = listOf("User Database", "Transaction Records")
    )

    val userDatabase = Asset(
        assetId = "A001",
        name = "User Database",
        type = AssetType.USER_DATA,
        criticality = Criticality.CRITICAL
    )
    // Risk assessment
    val likelihood = 0.7f  // Common attack, likely to be attempted
    val businessImpact = 1.0f  // Complete data compromise

    return Risk(
        riskId = "R001",
        vulnerability = sqliVuln,
        asset = userDatabase,
        likelihood = likelihood,
        businessImpact = businessImpact
    )
}

// Example: Mitigation controls for SQL injection
fun mitigateSQLInjection(): List<MitigationControl> {
    return listOf(
        MitigationControl(
            controlId = "C001",
            controlName = "Parameterized Queries",
            controlType = ControlType.PREVENTIVE,
            residualRisk = 0.1f,  // Reduces risk from 0.7 to 0.1
            cost = 2000,          // Developer hours
            timeline = "Sprint 1"
        ),
        MitigationControl(
            controlId = "C002",
            controlName = "Input Validation",
            controlType = ControlType.PREVENTIVE,
            residualRisk = 0.15f,
            cost = 1500,
            timeline = "Sprint 1"
        ),
        MitigationControl(
            controlId = "C003",
            controlName = "Runtime Monitoring & Alerts",
            controlType = ControlType.DETECTIVE,
            residualRisk = 0.05f,  // Combined prevention + detection
            cost = 3000,
            timeline = "Sprint 2"
        )
    )
}
```

}

// Risk assessment report generation
class SecurityAssessmentReport {
fun generateRiskReport(risks: List<SecurityRiskAssessment.Risk>) {
val highRisks = risks.filter { it.riskScore >= 0.5 }

val mediumRisks = risks.filter { it.riskScore in 0.2..0.49 }
val lowRisks = risks.filter { it.riskScore < 0.2 }

```
    println("=== SECURITY RISK ASSESSMENT REPORT ===")
    println("High Risk: ${highRisks.size} issues (immediate action 
required)")
    println("Medium Risk: ${mediumRisks.size} issues (address within 30 
days)")
    println("Low Risk: ${lowRisks.size} issues (address within 90 
days)")

    highRisks.forEach { risk ->
        println("\nCRITICAL: ${risk.vulnerability.name}")
        println("Risk Score: ${String.format("%.2f", risk.riskScore)}")
    }
}
```

}

2. OWASP Mobile Top 10 & MASTG Compliance

Risk	Description	Mobile Impact	Mitigation
M1: Improper Platform Usage	Misuse of OS features, APIs	Crash exploits, permission bypass	Follow OS guidelines, proper API usage
M2: Insecure Data Storage	Unencrypted sensitive data	Credential theft, PII exposure	Keystore/Keychain, encryption
M3: Insecure Communication	MITM attacks, unencrypted channels	Password interception, data theft	Certificate pinning, TLS 1.2+
M4: Insecure Authentication	Weak auth, token management	Account takeover, identity spoofing	OAuth 2.0, MFA, secure tokens
M5: Insufficient Cryptography	Weak algorithms, poor implementation	Data decryption, key compromise	AES-256, RSA-2048, PBKDF2
M6: Insecure Authorization	Access control flaws	Feature/data access without permission	RBAC, input validation
M7: Client-Side Injection	SQL injection, XSS, command injection	Database compromise, code execution	Parameterized queries, sanitization
M8: Insufficient Input Validation	Unvalidated user input	Buffer overflow, injection attacks	Input validation, type checking

M9: Insecure Transmission	Exposed credentials in transit	Credential theft, MITM attacks	HTTPS, encrypted protocols
M10: Extraneous Functionality	Hidden/debug features, test code	Unauthorized access, privilege escalation	Remove test code, security audits

3. GDPR and HIPAA Compliance for Mobile Apps

// GDPR Compliance Framework
class GDPRComplianceManager {

```
// Article 5: Lawfulness, fairness, transparency
enum class LegalBasis {
    CONSENT,            // User explicitly agreed
    CONTRACT,           // Necessary for contract
    LEGAL_OBLIGATION,   // Required by law
    VITAL_INTERESTS,    // Protect vital interests
    PUBLIC_TASK,        // Perform public task
    LEGITIMATE_INTERESTS  // Legitimate business purpose
}

// Article 6: Consent management
data class ConsentRecord(
    val userId: String,
    val dataCategory: String,      // Location, Contacts, Photos, etc.
    val legalBasis: LegalBasis,
    val timestamp: Long,
    val version: String            // Consent form version
)

// Article 17: Right to be forgotten
fun deleteUserData(userId: String) {
    // Immediately delete:
    deleteFromPrimaryStorage(userId)

    // Schedule cascade delete from:
    scheduleDeleteFromBackups(userId)
    scheduleDeleteFromLogs(userId, minAgeDays = 30)

    // Notify: Data controller, processors, recipients
    notifyDataControllers(userId)
}

// Article 33: Breach notification
data class BreachNotification(
    val breachId: String,
    val discoveredDate: Long,
    val notifiedDate: Long,
    val affectedUsers: Int,
    val dataCategories: List<String>,
    val description: String,
    val mitigations: List<String>
)

fun notifyBreach(breach: BreachNotification) {
    // Notify within 72 hours of discovery
    require((System.currentTimeMillis() - breach.discoveredDate) < 72 *
60 * 60 * 1000)

    // Notify data authority
    notifyDataAuthority(breach)

    // Notify affected users (if high risk)
    notifyAffectedUsers(breach)
}
```

}

// HIPAA Compliance Framework (Healthcare)
class HIPAAComplianceManager {

```
// Privacy Rule: PHI (Protected Health Information) handling
enum class PHICategory {
    IDENTITY,          // Name, SSN, Patient ID
    MEDICAL_HISTORY,   // Diagnoses, treatments
    MEDICATIONS,       // Prescriptions, dosages
    BIOMETRICS,        // Heart rate, blood pressure
    GENETIC_INFO       // DNA, genetic markers
}

// Security Rule: Technical safeguards
data class SecurityControl(
    val controlId: String,
    val category: String,  // Access Control, Encryption, etc.
    val requirement: String,
    val implementation: String
)

val requiredControls = listOf(
    SecurityControl(
        controlId = "SC-1",
        category = "Access Control",
        requirement = "Unique user identification",
        implementation = "Username + strong password or MFA"
    ),
    SecurityControl(
        controlId = "SC-2",
        category = "Encryption & Decryption",
        requirement = "Encrypt PHI in transit and at rest",
        implementation = "AES-256, TLS 1.2+, HKDF"
    ),
    SecurityControl(
        controlId = "SC-3",
        category = "Audit & Accountability",
        requirement = "Log all PHI access",
        implementation = "Immutable audit logs with timestamps"
    ),
    SecurityControl(
        controlId = "SC-4",
        category = "Integrity",
        requirement = "Ensure PHI not altered or destroyed",
        implementation = "Digital signatures, checksums, versioning"
    )
)

// Breach notification (more stringent than GDPR)
fun notifyBreach(affectedCount: Int, description: String) {
    // Notify immediately (no 72-hour grace period)
    // Notify affected individuals
    notifyIndividuals(affectedCount, description)

    // Notify media if >500 residents
    if (affectedCount > 500) {
        notifyMedia(description)
    }

    // Notify HHS (Department of Health & Human Services)
    notifyHHS(affectedCount, description)
}
```

}

Deliverable:

- [ ] Security risk assessment framework with 3 sample risks
- [ ] Risk calculation and scoring methodology
- [ ] Threat-to-mitigation mapping for 5+ vulnerabilities
- [ ] OWASP Mobile Top 10 mapping to application
- [ ] GDPR compliance checklist (10+ items)
- [ ] HIPAA compliance requirements (healthcare apps)
- [ ] Risk report template with prioritization

Part B: Secure Coding and Input Validation

Task 3: Secure Coding Practices and Input Validation (Self-Study: 2.5–3 hours)

Objective: Master secure coding practices and comprehensive input validation mechanisms.

Instructions:

1. Secure Coding Fundamentals

// PRINCIPLE 1: Principle of Least Privilege
class LeastPrivilegeExample {

```
// WRONG: App requests too many permissions
// <uses-permission android:name="android.permission.READ_CONTACTS"/>
// <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
// <uses-permission android:name="android.permission.RECORD_AUDIO"/>

// CORRECT: Request only needed permissions dynamically
fun requestOnlyRequiredPermissions() {
    val requiredPermissions = when (userFeature) {
        UserFeature.PROFILE_SYNC -> listOf(
            android.Manifest.permission.READ_CONTACTS
        )
        UserFeature.LOCATION_TRACKING -> listOf(
            android.Manifest.permission.ACCESS_FINE_LOCATION
        )
        else -> emptyList()
    }

    ActivityCompat.requestPermissions(activity,
        requiredPermissions.toTypedArray(),
        PERMISSION_REQUEST_CODE)
}
```

}

// PRINCIPLE 2: Input Validation (Whitelist Approach)
class SecureInputValidation {

```
// Pattern definitions for various inputs
private val emailPattern = Regex("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-
]+\\.[A-Z|a-z]{2,}$")
private val phonePattern = Regex("^[+]?[0-9]{10,15}$")
private val usernamePattern = Regex("^[A-Za-z0-9_-]{3,20}$")  //
```

```
Alphanumeric, dash, underscore only

fun validateEmail(email: String): Boolean {
    // Length check
    if (email.length > 254) return false

    // Pattern match
    if (!emailPattern.matches(email)) return false

    // Additional checks
    val parts = email.split("@")
    if (parts[0].length > 64) return false  // Local part too long

    return true
}

fun validatePhoneNumber(phone: String): Boolean {
    return phonePattern.matches(phone)
}

fun validateUsername(username: String): Boolean {
    if (username.length < 3 || username.length > 20) return false
    if (!usernamePattern.matches(username)) return false

    // Check against blacklist of reserved names
    val reserved = listOf("admin", "root", "system", "administrator")
    if (username.lowercase() in reserved) return false

    return true
}

// Server-side validation (ALWAYS do this too!)
fun validateOnServer(input: String): Boolean {
    // Never trust client validation alone
    // Always validate on server with same strict rules
    return true
}
```

}

// PRINCIPLE 3: Output Encoding (Prevent XSS)
class OutputEncodingExample {

```
fun displayUserContent(userInput: String): String {
    // HTML encode special characters
    return userInput
        .replace("&", "&")
        .replace("<", "<")
        .replace(">", ">")
        .replace("\"", """)
        .replace("'", "&#39;")
}

// Example: Display user-provided comment safely
fun renderComment(comment: String): String {
    val safeComment = displayUserContent(comment)
    return "<div class=\"comment\">$safeComment</div>"
}

// Example: Set URL safely
fun buildSafeURL(baseUrl: String, userParam: String): String {
    val encodedParam = java.net.URLEncoder.encode(userParam, "UTF-8")
    return "$baseUrl?param=$encodedParam"
}
```

}

// PRINCIPLE 4: Error Handling (Don't leak info)
class SecureErrorHandling {

```
fun loginUser(username: String, password: String): Result {
    return try {
        val user = database.findUser(username)
        when {
            user == null -> {
                // WRONG: "User not found" - reveals user existence
                // CORRECT: Generic message
                Result.Error("Invalid username or password")
            }
            !verifyPassword(password, user.passwordHash) -> {
                // CORRECT: Same message as unknown user
                Result.Error("Invalid username or password")
            }
            !user.isVerified -> {
                Result.Error("Account not verified")
            }
            else -> {
                // Rate limit check
                if (rateLimiter.isLimited(username)) {
                    Result.Error("Too many login attempts. Try again in
15 minutes.")
                } else {
                    Result.Success(createAuthToken(user))
                }
            }
        }
    } catch (e: Exception) {
        // WRONG: "Database connection error" - reveals backend info
        // CORRECT: Generic error without implementation details
        Log.e("Auth", "Login failed", e)  // Log internally
        Result.Error("An error occurred. Please try again.")
    }
}

// Never return stack traces or sensitive errors to client
fun safeErrorMessage(e: Exception): String {
    return when (e) {
        is SQLException -> "Database error"  // Don't reveal DB details
        is TimeoutException -> "Request timeout"
        else -> "An unexpected error occurred"
    }
}
```

}

// PRINCIPLE 5: Dependency Management
class SecureDependencyManagement {

```
// Check for known vulnerabilities
// Use tools: OWASP Dependency-Check, Snyk, WhiteSource

// build.gradle
// dependencies {
//     implementation 'com.squareup.okhttp3:okhttp:4.11.0'  // ✓ Latest
//     implementation 'com.google.code.gson:gson:2.10.1'    // ✓ Latest
//     // implementation 'com.android.volley:volley:1.1.0'   // ✗ 
Outdated, has CVEs
// }

fun scanDependencies() {
    // Run in CI/CD:
    // ./gradlew dependencyCheck
    // Fails build if vulnerabilities found
}
```

}

2. SQL Injection Prevention

// VULNERABLE: SQL Injection
class VulnerableSQLExample {
fun findUserByEmail(email: String): User? {
// NEVER DO THIS!
val query = "SELECT * FROM users WHERE email = '$email'"
return database.rawQuery(query).firstOrNull()?.toUser()
}

```
// Attack: email = "' OR '1'='1"
// Resulting query: SELECT * FROM users WHERE email = '' OR '1'='1'
// Result: Returns ALL users!
```

}

// SECURE: Parameterized Queries
class SecureSQLExample {
fun findUserByEmail(email: String): User? {
// CORRECT: Use parameterized query
val query = "SELECT * FROM users WHERE email = ?"
return database.query(
query = query,
args = arrayOf(email) // Email safely passed as parameter
).firstOrNull()?.toUser()
}

```
// Also works with Room ORM (Android's recommended approach)
// @Query("SELECT * FROM users WHERE email = :email")
// fun findUserByEmail(email: String): User?
```

}

// Using Room (Android recommended)
@Dao
interface UserDao {
@Query("SELECT * FROM users WHERE email = :email")
suspend fun findByEmail(email: String): User?

```
@Query("SELECT * FROM users WHERE age > :minAge")
suspend fun findAdults(minAge: Int): List<User>

// Batch operations (safer than string concatenation)
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertUser(user: User)

@Delete
suspend fun deleteUser(user: User)
```

}

3. Cross-Site Scripting (XSS) Prevention

// Mobile context: WebView XSS vulnerabilities
class SecureWebViewExample {

```
fun setupWebView(webView: WebView, userContent: String) {
    // WRONG: Loading untrusted content
    // webView.loadData(userContent, "text/html", "UTF-8")

    // CORRECT: Multiple defenses
    val settings = webView.settings
    // 1. Disable JavaScript (if not needed)
    settings.javaScriptEnabled = false

    // 2. Disable access to device features
    settings.allowFileAccess = false
    settings.allowContentAccess = false

    // 3. Use Content Security Policy (CSP)
    val cspHeader = "default-src 'self'; script-src 'self'; style-src
'self' 'unsafe-inline';"

    // 4. Escape user content
    val safeContent = escapeHtml(userContent)

    // 5. Set minimal permissions
    webView.webChromeClient = object : WebChromeClient() {
        override fun onPermissionRequest(request: PermissionRequest?) {
            // Deny all permission requests
            request?.deny()
        }
    }

    webView.loadData(safeContent, "text/html; charset=UTF-8", null)
}

private fun escapeHtml(unsafe: String): String {
    return unsafe
        .replace("&", "&")
        .replace("<", "<")
        .replace(">", ">")
        .replace("\"", """)
        .replace("'", "&#x27;")
}
```

}

// iOS WebView Security
class SecureWKWebViewExample {

```
func setupWebView(webView: WKWebView, content: String) {
    // 1. Configure security policies
    let config = WKWebViewConfiguration()

    // Disable file access
    config.preferences.javaScriptEnabled = false

    // 2. Content Security Policy
    let contentSecurityPolicy = """
    default-src 'self';
    script-src 'self';
    style-src 'self' 'unsafe-inline';
    """

    // 3. Disable plugins
    config.preferences.plugInFullscreenRequiresUserGesture = true

    // 4. Escape HTML
    let safeContent = escapeHTML(content)

    webView.navigationDelegate = self
    webView.load(HTMLString: safeContent, baseURL: nil)
}

private func escapeHTML(_ text: String) -> String {
    return text
        .replacingOccurrences(of: "&", with: "&")
        .replacingOccurrences(of: "<", with: "<")
        .replacingOccurrences(of: ">", with: ">")
        .replacingOccurrences(of: "\"", with: """)
        .replacingOccurrences(of: "'", with: "&#x27;")
}
```

}

4. CSRF Protection

// Cross-Site Request Forgery Prevention
class CSRFProtectionExample {

```
// CSRF Token Generation
fun generateCSRFToken(): String {
    val randomBytes = ByteArray(32)
    SecureRandom().nextBytes(randomBytes)
    return Base64.getEncoder().encodeToString(randomBytes)
}

// Store token in app (secure storage)
fun storeCSRFToken(token: String) {
    val preferences = getSharedPreferences("csrf",
Context.MODE_PRIVATE)
    preferences.edit().putString("csrf_token", token).apply()
}

// Include token in request
fun makeProtectedRequest(endpoint: String): Response {
    val token = getStoredCSRFToken()

    val request = Request.Builder()
        .url(endpoint)
        .addHeader("X-CSRF-Token", token)  // Send in header
        .post(body)
        .build()

    return httpClient.newCall(request).execute()
}

// Server-side validation (CRITICAL)
fun validateCSRFToken(token: String, sessionToken: String): Boolean {
    return token == sessionToken && token.isNotEmpty()
}
```

}

Deliverable:

- [ ] Secure coding principles with mobile examples (5+ principles)
- [ ] Input validation framework with 5+ patterns
- [ ] SQL injection prevention examples (vulnerable vs. secure)
- [ ] XSS prevention in WebView/WKWebView (Android & iOS)
- [ ] Error handling guide without information leakage
- [ ] CSRF protection implementation
- [ ] Output encoding for all user-supplied data

Part C: Encryption, Authentication, and Data Security

Task 4: Encryption and Cryptographic Mechanisms (Self-Study: 2.5–3 hours)

Objective: Master encryption strategies and cryptographic implementations for mobile security.

Instructions:

1. Encryption Fundamentals

Encryption Type	Algorithm	Key Size	Use Case	Speed
Symmetric	AES	128-256 bits	Data at rest, bulk encryption	Fast

Symmetric	ChaCha20	256 bits	Modern alternative to AES	Fast, constant-time
Asymmetric	RSA	2048-4096 bits	Key exchange, signatures	Slow
Asymmetric	ECDSA	256-384 bits	Digital signatures, TLS	Medium
Hashing	SHA-256	256 bits	Password storage, integrity	One-way
Hashing	PBKDF2	Variable	Password derivation	Slow (intentional)
HMAC	HMAC-SHA256	256 bits	Message authentication	Fast

2. Android Encryption Implementation

// Symmetric Encryption: AES-256
class AESEncryption {
private val cipher = Cipher.getInstance("AES/GCM/NoPadding")
private val keyGenerator = KeyGenerator.getInstance("AES")

```
// Generate encryption key
fun generateKey(): SecretKey {
    keyGenerator.init(256)
    return keyGenerator.generateKey()
}

// Encrypt data
fun encrypt(plaintext: String, key: SecretKey): String {
    cipher.init(Cipher.ENCRYPT_MODE, key)
    val iv = cipher.iv  // Initialization vector (random)
    val ciphertext =
cipher.doFinal(plaintext.toByteArray(Charsets.UTF_8))

    // Return IV + ciphertext (IV must be transmitted)
    val combined = ByteArray(iv.size + ciphertext.size)
    System.arraycopy(iv, 0, combined, 0, iv.size)
    System.arraycopy(ciphertext, 0, combined, iv.size, ciphertext.size)

    return Base64.getEncoder().encodeToString(combined)
}

// Decrypt data
fun decrypt(encrypted: String, key: SecretKey): String {
    val combined = Base64.getDecoder().decode(encrypted)
    val iv = combined.sliceArray(0 until 12)  // GCM IV is 12 bytes
    val ciphertext = combined.sliceArray(12 until combined.size)
    val spec = GCMParameterSpec(128, iv)  // 128-bit auth tag
    cipher.init(Cipher.DECRYPT_MODE, key, spec)

    val plaintext = cipher.doFinal(ciphertext)
    return String(plaintext, Charsets.UTF_8)
}
```

}

// Asymmetric Encryption: RSA
class RSAEncryption {
private val keyFactory = KeyFactory.getInstance("RSA")
private val cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding")

```
// Generate RSA key pair
fun generateKeyPair(): KeyPair {
    val keyPairGen = KeyPairGenerator.getInstance("RSA")
    keyPairGen.initialize(2048)
    return keyPairGen.generateKeyPair()
}

// Encrypt with public key
fun encryptPublic(plaintext: String, publicKey: PublicKey): String {
    cipher.init(Cipher.ENCRYPT_MODE, publicKey)
    val ciphertext =
cipher.doFinal(plaintext.toByteArray(Charsets.UTF_8))
    return Base64.getEncoder().encodeToString(ciphertext)
}

// Decrypt with private key
fun decryptPrivate(ciphertext: String, privateKey: PrivateKey): String
{
    val encrypted = Base64.getDecoder().decode(ciphertext)
    cipher.init(Cipher.DECRYPT_MODE, privateKey)
    val plaintext = cipher.doFinal(encrypted)
    return String(plaintext, Charsets.UTF_8)
}
```

}

// Password Hashing: PBKDF2
class PasswordHashing {
private val algorithm = "PBKDF2WithHmacSHA256"
private val iterations = 100000
private val keyLength = 256

```
fun hashPassword(password: String): String {
    // Generate random salt
    val salt = ByteArray(16)
    SecureRandom().nextBytes(salt)
    // Derive key
    val factory = SecretKeyFactory.getInstance(algorithm)
    val spec = PBEKeySpec(password.toCharArray(), salt, iterations,
keyLength)
    val hash = factory.generateSecret(spec).encoded

    // Return salt + hash (both needed for verification)
    val combined = ByteArray(salt.size + hash.size)
    System.arraycopy(salt, 0, combined, 0, salt.size)
    System.arraycopy(hash, 0, combined, salt.size, hash.size)

    return Base64.getEncoder().encodeToString(combined)
}

fun verifyPassword(password: String, storedHash: String): Boolean {
    val decoded = Base64.getDecoder().decode(storedHash)
    val salt = decoded.sliceArray(0 until 16)
    val expectedHash = decoded.sliceArray(16 until decoded.size)

    val factory = SecretKeyFactory.getInstance(algorithm)
    val spec = PBEKeySpec(password.toCharArray(), salt, iterations,
keyLength)
    val actual = factory.generateSecret(spec).encoded

    // Constant-time comparison (prevent timing attacks)
    return MessageDigest.isEqual(actual, expectedHash)
}
```

}

// Message Authentication: HMAC
class HMACAuthentication {
fun generateHMAC(data: String, key: SecretKey): String {
val mac = Mac.getInstance("HmacSHA256")
mac.init(key)
val hash = mac.doFinal(data.toByteArray(Charsets.UTF_8))
return Base64.getEncoder().encodeToString(hash)
}

```
fun verifyHMAC(data: String, receivedHMAC: String, key: SecretKey):
Boolean {
    val expectedHMAC = generateHMAC(data, key)

    // Constant-time comparison
    return MessageDigest.isEqual(
        expectedHMAC.toByteArray(),
        receivedHMAC.toByteArray()
    )
}
```

}

// Android Keystore (Secure Key Storage)
class AndroidKeyStoreEncryption {

```
// Generate key in Keystore (hardware-backed if available)
fun generateKeyStoreKey(keyAlias: String) {
    val keyGenerator = KeyGenerator.getInstance(
        KeyProperties.KEY_ALGORITHM_AES,
        "AndroidKeyStore"
    )

    val keySpec = KeyGenParameterSpec.Builder(
        keyAlias,
        KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
    )
        .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .setKeySize(256)
        .setUserAuthenticationRequired(true)  // Require biometric/PIN
        .setUserAuthenticationValidityDurationSeconds(300)  // Valid
for 5 min
        .build()

    keyGenerator.init(keySpec)
    keyGenerator.generateKey()
}

// Retrieve key from Keystore
fun getKeyStoreKey(keyAlias: String): SecretKey {
    val keyStore = KeyStore.getInstance("AndroidKeyStore")
    keyStore.load(null)
    return keyStore.getKey(keyAlias, null) as SecretKey
}
```

}

3. iOS Encryption Implementation

// Symmetric Encryption: AES-256
class AESEncryption {
func generateKey() -> SymmetricKey {
return SymmetricKey(size: .bits256)
}

```
func encrypt(plaintext: String, key: SymmetricKey) -> String? {
    guard let data = plaintext.data(using: .utf8) else { return nil }

    do {
        let sealedBox = try AES.GCM.seal(data, using: key)
        return sealedBox.combined?.base64EncodedString()
    } catch {
        return nil
    }
}

func decrypt(ciphertext: String, key: SymmetricKey) -> String? {
    guard let data = Data(base64Encoded: ciphertext),
          let sealedBox = try? AES.GCM.SealedBox(combined: data) else {
        return nil
    }

    do {
        let decrypted = try AES.GCM.open(sealedBox, using: key)
        return String(data: decrypted, encoding: .utf8)
    } catch {
        return nil
    }
}
```

}

// Password Hashing: PBKDF2
class PasswordHashing {
func hashPassword(_ password: String) -> String? {
let salt = (0..<16).map { _ in UInt8.random(in: 0...255) }

```
    var derivedKey = [UInt8](repeating: 0, count: 32)
    let result = CCKeyDerivationPBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password,
        password.utf8.count,
        salt,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA256),
        100000,  // iterations
        &derivedKey,
        derivedKey.count
    )

    guard result == kCCSuccess else { return nil }

    var combined = salt
    combined.append(contentsOf: derivedKey)
    return Data(combined).base64EncodedString()
}

func verifyPassword(_ password: String, hash: String) -> Bool {
    guard let decodedHash = Data(base64Encoded: hash) else { return
false }
```

```
    let salt = Array(decodedHash.prefix(16))
    let expectedKey = Array(decodedHash.suffix(from: 16))

    var derivedKey = [UInt8](repeating: 0, count: 32)
    let result = CCKeyDerivationPBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password,
        password.utf8.count,
        salt,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA256),
        100000,
        &derivedKey,
        derivedKey.count
    )

    return result == kCCSuccess && derivedKey == expectedKey
}
```

}

// Keychain (Secure Storage)
class KeychainStorage {

```
func storePassword(_ password: String, account: String) {
    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrAccount as String: account,
        kSecValueData as String: password.data(using: .utf8)!,
        kSecAttrAccessible as String:
kSecAttrAccessibleWhenUnlockedThisDeviceOnly
    ]

    SecItemAdd(query as CFDictionary, nil)
}

func retrievePassword(account: String) -> String? {
    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrAccount as String: account,
        kSecReturnData as String: true
    ]

    var result: AnyObject?
    let status = SecItemCopyMatching(query as CFDictionary, &result)

    if status == errSecSuccess,
       let data = result as? Data,
       let password = String(data: data, encoding: .utf8) {
        return password
    }
    return nil
}

func deletePassword(account: String) {
    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrAccount as String: account
    ]

    SecItemDelete(query as CFDictionary)
}
```

}

4. Key Derivation and Salt Management

// Key Derivation Function (KDF)
class KeyDerivation {

```
// HKDF (HMAC-based KDF) for derived keys
fun deriveKeys(
    masterKey: ByteArray,
    context: String,
    length: Int = 32
): ByteArray {
    val hmac = Mac.getInstance("HmacSHA256")
    hmac.init(SecretKeySpec(masterKey, 0, masterKey.size,
"HmacSHA256"))

    // HKDF-Expand
    val info = context.toByteArray(Charsets.UTF_8)
    val prk = hmac.doFinal(info)  // Extract

    // Expand
    hmac.init(SecretKeySpec(prk, 0, prk.size, "HmacSHA256"))
    var result = ByteArray(0)
    var block = ByteArray(0)
    var counter = 1.toByte()

    while (result.size < length) {
        hmac.reset()
        hmac.update(block)
        hmac.update(info)
        hmac.update(counter)
        block = hmac.doFinal()
        result = result.plus(block)
        counter++
    }
    return result.sliceArray(0 until length)
}
```

}

Deliverable:

- [ ] Encryption fundamentals comparison table
- [ ] Android AES-256 implementation (encrypt/decrypt)
- [ ] iOS AES-256 implementation (encrypt/decrypt)
- [ ] Password hashing with PBKDF2 (both platforms)
- [ ] RSA encryption for key exchange
- [ ] HMAC for message authentication
- [ ] Keystore/Keychain secure key storage
- [ ] Key derivation function examples
- [ ] Constant-time comparison for security

Task 5: Authentication and Session Management (Self-Study: 2–2.5 hours)

Objective: Master authentication protocols and secure session management.

Instructions:

1. OAuth 2.0 and OpenID Connect

// OAuth 2.0 Authorization Code Flow
class OAuth2Manager(private val context: Context) {

```
// Step 1: Generate authorization request
fun initiateLogin() {
    val clientId = "your-client-id"
    val redirectUri = "com.example.app://callback"
    val scope = "openid profile email"
    val state = generateRandomState()  // CSRF protection

    // Store state in secure storage
    secureStorage.saveState(state)

    val authUri = "https://auth-server.com/authorize?" +
            "client_id=$clientId&" +
            "redirect_uri=${Uri.encode(redirectUri)}&" +
            "scope=${Uri.encode(scope)}&" +
            "state=$state&" +
            "response_type=code"

    startCustomTab(authUri)
}

// Step 2: Handle redirect with authorization code
fun handleCallback(uri: Uri) {
    val code = uri.getQueryParameter("code") ?: return
    val state = uri.getQueryParameter("state") ?: return

    // Verify state (CSRF protection)
    val storedState = secureStorage.getState()
    if (state != storedState) {
        throw SecurityException("State mismatch - CSRF attack
detected")
    }

    // Step 3: Exchange code for token (server-to-server)
    exchangeCodeForToken(code)
}

// Step 3: Exchange authorization code for access token
private fun exchangeCodeForToken(code: String) {
    val clientId = "your-client-id"
    val clientSecret = BuildConfig.CLIENT_SECRET  // Not in app, fetch
from secure backend
    val redirectUri = "com.example.app://callback"

    val request = Request.Builder()
        .url("https://auth-server.com/token")
        .post(FormBody.Builder()
            .add("grant_type", "authorization_code")
            .add("code", code)
            .add("client_id", clientId)
            .add("client_secret", clientSecret)
            .add("redirect_uri", redirectUri)
            .build())
        .build()

    val response = httpClient.newCall(request).execute()
    val tokens = parseTokenResponse(response.body?.string() ?: "")

    // Store tokens securely
    secureStorage.saveAccessToken(tokens.accessToken)
    secureStorage.saveRefreshToken(tokens.refreshToken)
}
```

}

// Token Management
class TokenManager(private val context: Context) {

```
fun getValidAccessToken(): String? {
    var token = secureStorage.getAccessToken()

    // Check if expired
    if (isTokenExpired(token)) {
        // Refresh token
        val refreshToken = secureStorage.getRefreshToken()
        token = refreshAccessToken(refreshToken)
    }

    return token
}

private fun refreshAccessToken(refreshToken: String): String? {
    val request = Request.Builder()
        .url("https://auth-server.com/token")
        .post(FormBody.Builder()
            .add("grant_type", "refresh_token")
            .add("refresh_token", refreshToken)
            .add("client_id", "your-client-id")
            .build())
        .build()

    val response = httpClient.newCall(request).execute()
    val tokens = parseTokenResponse(response.body?.string() ?: "")

    // Update tokens
    secureStorage.saveAccessToken(tokens.accessToken)
    return tokens.accessToken
}

fun logout() {
    // Revoke tokens on server
    revokeTokens()

    // Clear local storage
    secureStorage.clearAccessToken()
    secureStorage.clearRefreshToken()
}
```

}

// OpenID Connect (Identity Layer on OAuth 2.0)
class OpenIDConnectManager {

```
fun validateIDToken(idToken: String): Map<String, Any>? {
    // 1. Decode JWT (without verification first)
    val parts = idToken.split(".")
    if (parts.size != 3) return null

    val header = decodeBase64(parts[0])
    val payload = decodeBase64(parts[1])
    val signature = parts[2]

    // 2. Verify signature with JWKS (JSON Web Key Set)
    val jwks = fetchJWKS("https://auth-server.com/.well-
known/jwks.json")
    if (!verifySignature(payload, signature, jwks)) {
        return null
    }

    // 3. Validate claims
    val claims = parseJWT(payload)

    // Check issuer
    if (claims["iss"] != "https://auth-server.com") return null

    // Check audience
    if (claims["aud"] != "your-client-id") return null

    // Check expiration
    val expTime = (claims["exp"] as? Number)?.toLong() ?: return null
    if (System.currentTimeMillis() / 1000 > expTime) return null

    // Check not before
    val nbfTime = (claims["nbf"] as? Number)?.toLong() ?: return null
    if (System.currentTimeMillis() / 1000 < nbfTime) return null

    return claims
}
```

}

2. Certificate Pinning

// Certificate Pinning (Prevent MITM via rogue CA)
class CertificatePinningManager {

```
fun createSecureHttpClient(): OkHttpClient {
    // Get your server's certificate
    val certificate = CertificateFactory.getInstance("X.509")
        .generateCertificate(context.assets.open("certificate.pem"))

    val trustStore = KeyStore.getInstance("BKS")
    trustStore.load(null)
    trustStore.setCertificateEntry("ca", certificate)

    // Create certificate pins
    val pins = setOf(
        "sha256/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="  // Leaf
    )

    return OkHttpClient.Builder()
        .certificatePinner(
            CertificatePinner.Builder()
                .add("api.example.com", *pins.toTypedArray())
                .build()
        )
        .build()
}

// iOS Certificate Pinning
func createSecureURLSession() -> URLSession {
    let certificate = NSBundle.main.urlForResource("certificate",
withExtension: "cer")!
    let data = NSData(contentsOfURL: certificate)!
    let certificate = SecCertificateCreateWithData(nil, data as
CFData)!

    let config = URLSessionConfiguration.default
    config.requestCachePolicy = .reloadIgnoringLocalCacheData
    config.urlCache = nil

    let delegate = SSLPinningDelegate(pinnedCertificates:
[certificate])

    return URLSession(
        configuration: config,
        delegate: delegate,
        delegateQueue: nil
    )
}
```

}

3. Mutual TLS

// Mutual TLS (Client Certificate Authentication)
class MutualTLSManager(context: Context) {

```
fun createMTLSHttpClient(): OkHttpClient {
    // Load client certificate
    val keystoreFile = context.assets.open("client-keystore.p12")
    val keystore = KeyStore.getInstance("PKCS12")
    keystore.load(keystoreFile, "keystore-password".toCharArray())

    // Create key manager
    val keyManagerFactory = KeyManagerFactory.getInstance("X509")
    keyManagerFactory.init(keystore, "key-password".toCharArray())

    // Load server certificate
    val truststoreFile = context.assets.open("server-truststore.bks")
    val truststore = KeyStore.getInstance("BKS")
    truststore.load(truststoreFile, "truststore-
password".toCharArray())

    // Create trust manager
    val trustManagerFactory = TrustManagerFactory.getInstance("X509")
    trustManagerFactory.init(truststore)

    // Create SSL context
    val sslContext = SSLContext.getInstance("TLS")
    sslContext.init(
        keyManagerFactory.keyManagers,
        trustManagerFactory.trustManagers,
        SecureRandom()
    )

    return OkHttpClient.Builder()
        .sslSocketFactory(
            sslContext.socketFactory,
            trustManagerFactory.trustManagers[0] as X509TrustManager
        )
        .build()
}
```

}

Deliverable:

- [ ] OAuth 2.0 authorization code flow implementation
- [ ] Token refresh and refresh token rotation
- [ ] OpenID Connect ID token validation
- [ ] Session management with secure token storage
- [ ] Certificate pinning for MITM prevention
- [ ] Mutual TLS client certificate setup
- [ ] Logout and token revocation
- [ ] CSRF state parameter protection

Part D: Data Protection and Reverse Engineering Prevention

Task 6: Secure Data Storage and Reverse Engineering Defense (Self-Study: 2–2.5 hours)

Objective: Master secure data storage and protection against reverse engineering attacks.

Instructions:

1. Secure Data Storage

// Android: EncryptedSharedPreferences
class SecurePreferencesStorage(context: Context) {

```
private val encryptedPreferences by lazy {
    EncryptedSharedPreferences.create(
        context,
        "secret_shared_prefs",
        MasterKey.Builder(context)
            .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
            .build(),
        EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
        EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
    )
}

fun saveSecret(key: String, value: String) {
    encryptedPreferences.edit().putString(key, value).apply()
}

fun getSecret(key: String): String? {
    return encryptedPreferences.getString(key, null)
}
```

}

// iOS: Keychain
class SecureDataStorage {

```
func saveCredential(_ credential: String, account: String) {
    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrAccount as String: account,
        kSecValueData as String: credential.data(using: .utf8)!,
        kSecAttrAccessible as String:
kSecAttrAccessibleWhenUnlockedThisDeviceOnly
    ]

    SecItemAdd(query as CFDictionary, nil)
}

func retrieveCredential(account: String) -> String? {
    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrAccount as String: account,
        kSecReturnData as String: true
    ]

    var result: AnyObject?
    SecItemCopyMatching(query as CFDictionary, &result)

    if let data = result as? Data {
        return String(data: data, encoding: .utf8)
    }
    return nil
}
```

}

// Android: Room Database Encryption
@Database(entities = [User::class], version = 1)
abstract class EncryptedDatabase : RoomDatabase() {
abstract fun userDao(): UserDao

```
companion object {
    fun createEncrypted(context: Context): EncryptedDatabase {
        val passphrase = MasterKey.Builder(context)
            .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
            .build()

        return Room.databaseBuilder(
            context,
            EncryptedDatabase::class.java,
            "encrypted_db"
        )
.openHelperFactory(FrameworkSQLCipherOpenHelperFactory(passphrase))
            .build()
    }
}
```

}

// Clear sensitive data from memory
class SecureMemoryClearing {

```
fun clearSensitiveData(data: ByteArray) {
    // Use Java Arrays.fill to securely clear
    java.util.Arrays.fill(data, 0.toByte())
}

fun clearSensitiveString(sensitive: String?) {
    // Strings are immutable, but we can clear password arrays
    if (sensitive is CharArray) {
        java.util.Arrays.fill(sensitive, '\u0000')
    }
}
```

}

2. Reverse Engineering Defense

// Code Obfuscation (R8/ProGuard)
// build.gradle
// buildTypes {
// release {
// minifyEnabled true
// shrinkResources true
// proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
// }
// }

// proguard-rules.pro
// -keep class com.example.critical.** { *; }
// -keepclassmembers class * {
// private <methods>;
// }
// -repackageclasses 'com.obfuscated'
// -allowaccessmodification
// -useuniqueclassmembernames

// Debugger Prevention
class DebuggerDetection {

```
fun isDebuggerAttached(): Boolean {
    // Check if debugger is connected
    return Debug.isDebuggerConnected() ||
           isDebuggable() ||
           isEmulator() ||
           hasTracer()
}

private fun isDebuggable(): Boolean {
    return (context.applicationInfo.flags and
ApplicationInfo.FLAG_DEBUGGABLE) != 0
}

private fun isEmulator(): Boolean {
    return Build.FINGERPRINT.contains("generic") ||
           Build.FINGERPRINT.contains("unknown") ||
           Build.HARDWARE.contains("goldfish") ||
           Build.PRODUCT.contains("emulator")
}
```

```
private fun hasTracer(): Boolean {
    val file = File("/proc/${android.os.Process.myPid()}/status")
    if (!file.exists()) return false

    return file.readText().contains("TracerPid") &&
           !file.readText().contains("TracerPid:\t0")
}

fun preventIfDebugger() {
    if (isDebuggerAttached()) {
        // Exit app or disable sensitive features
        android.os.Process.killProcess(android.os.Process.myPid())
    }
}
```

}

// Code Integrity Verification
class CodeIntegrityChecker(context: Context) {

```
fun verifyAPKSignature(): Boolean {
    try {
        val packageInfo = context.packageManager.getPackageInfo(
            context.packageName,
            PackageManager.GET_SIGNATURES
        )

        for (signature in packageInfo.signatures) {
            val md = MessageDigest.getInstance("SHA256")
            val publicKey = md.digest(signature.toByteArray())
            val hash = Base64.getEncoder().encodeToString(publicKey)

            // Compare with expected signature
            if (hash == EXPECTED_SIGNATURE_HASH) {
                return true
            }
        }
    } catch (e: Exception) {
        return false
    }
    return false
}

companion object {
    private const val EXPECTED_SIGNATURE_HASH = "your-app-signature-
hash"
}
```

}

// Root/Jailbreak Detection
class RootDetection {

```
fun isDeviceRooted(): Boolean {
    return checkRootFiles() || checkDangerousProperties()
}

private fun checkRootFiles(): Boolean {
    val rootFiles = arrayOf(
        "/system/bin/su",
        "/system/xbin/su",
        "/data/local/xbin/su",
        "/data/local/bin/su",
        "/system/sd/xbin/su"
    )

    for (file in rootFiles) {
        if (File(file).exists()) return true
    }
    return false
}

private fun checkDangerousProperties(): Boolean {
    val dangerousProps = mapOf(
        "ro.secure" to "0",
        "ro.debuggable" to "1"
    )

    for ((prop, value) in dangerousProps) {
        if (getProperty(prop) == value) return true
    }
    return false
}
```

}

// iOS: Jailbreak Detection
class JailbreakDetection {

```
func isDeviceJailbroken() -> Bool {
    // Check for Cydia
    if UIApplication.shared.canOpenURL(URL(string: "cydia://package")!)
{
        return true
    }

    // Check for common jailbreak files
    let jailbreakFiles = [
        "/Applications/Cydia.app",
        "/var/cache/apt",
        "/Library/MobileSubstrate/MobileSubstrate.dylib",
        "/bin/bash"
    ]

    for path in jailbreakFiles {
        if FileManager.default.fileExists(atPath: path) {
            return true
        }
    }

    // Check for suspicious library loading
    let dyldInsertLibrariesEnv =
ProcessInfo.processInfo.environment["DYLD_INSERT_LIBRARIES"]
    if dyldInsertLibrariesEnv != nil {
        return true
    }

    return false
}
```

}

3. Tamper Detection

// Runtime Integrity Checking
class TamperDetection(context: Context) {

```
// Verify APK hasn't been modified
fun verifyAPKIntegrity(): Boolean {
    try {
        val apkPath = context.applicationInfo.sourceDir
        val apkFile = File(apkPath)

        // Calculate hash of APK
        val fis = FileInputStream(apkFile)
        val md = MessageDigest.getInstance("SHA256")
        val buffer = ByteArray(4096)
        var bytesRead: Int

        while (fis.read(buffer).also { bytesRead = it } != -1) {
            md.update(buffer, 0, bytesRead)
        }
        fis.close()

        val hash = Base64.getEncoder().encodeToString(md.digest())

        // Compare with expected hash
        return hash == EXPECTED_APK_HASH
    } catch (e: Exception) {
        return false
    }
}

// Verify dynamic library integrity
fun verifyLibraryIntegrity(): Boolean {
    try {
        val nativeLibPath = File(
            context.applicationInfo.nativeLibraryDir,
            "libnative.so"
        )

        val md = MessageDigest.getInstance("SHA256")
        md.update(nativeLibPath.readBytes())

        val hash = Base64.getEncoder().encodeToString(md.digest())
        return hash == EXPECTED_LIBRARY_HASH
    } catch (e: Exception) {
        return false
    }
}

companion object {
    private const val EXPECTED_APK_HASH = "your-apk-hash"
    private const val EXPECTED_LIBRARY_HASH = "your-library-hash"
}
```

}

Deliverable:

- [ ] Encrypted SharedPreferences implementation (Android)
- [ ] Keychain secure storage (iOS)
- [ ] Room database encryption with SQLCipher
- [ ] Code obfuscation configuration (ProGuard/R8)
- [ ] Debugger detection and prevention
- [ ] APK signature verification
- [ ] Root/jailbreak detection (both platforms)
- [ ] Code integrity verification at runtime
- [ ] Tamper detection mechanisms

Part E: Security Testing and Compliance

Task 7: Security Testing and Vulnerability Assessment (Self-Study: 2–2.5 hours)

Objective: Master security testing methodologies and vulnerability assessment.

Instructions:

1. Static Application Security Testing (SAST)

```
// Examples of code patterns SAST tools detect

// VULNERABILITY: Hardcoded credentials
class VulnerableAPI {
val API_KEY = "sk_live_abc123xyz" // ✗ SAST will flag this
val DATABASE_PASSWORD = "admin123" // ✗ SAST will flag this
}

// VULNERABILITY: SQL injection
class SQLVulnerability {
fun queryUser(email: String): User? {
val query = "SELECT * FROM users WHERE email = '$email'" // ✗ SQL Injection
return database.rawQuery(query)
}
}

// VULNERABILITY: Weak cryptography
class WeakCrypto {
fun encryptPassword(password: String): String {
// ✗ MD5 is insecure
return MessageDigest.getInstance("MD5")
.digest(password.toByteArray())
.joinToString("") { "%02x".format(it) }
}
}

// SAST Tools for Android:
// - Android Studio (built-in lint)
// - FindBugs / SpotBugs
// - Checkstyle
// - SonarQube
// - OWASP Dependency-Check

// build.gradle configuration for SAST
// plugins {
// id 'org.sonarqube' version '3.5.0'
// }
//
// sonarqube {
// properties {
// property 'sonar.projectKey', 'mobile-security'
// property 'sonar.sources', 'src'
// property 'sonar.exclusions', '/test/'
// property 'sonar.android.lint.report', 'lint-report.xml'
// }
// }
```

2. Dynamic Application Security Testing (DAST)

// DAST focuses on runtime behavior and API interactions

class DynamicSecurityTesting {

```
// Test 1: Intercept HTTP traffic
// Tools: Burp Suite, OWASP ZAP, Charles Proxy
// Method: Set proxy and examine requests/responses
// Check for:
// - Unencrypted sensitive data
// - Weak SSL/TLS
// - Missing security headers

// Test 2: Test API endpoints
fun testAPIEndpoints() {
    // Test authentication bypass
    testAuthenticationBypass()

    // Test authorization flaws
    testAuthorizationFlaws()

    // Test input validation
    testInputValidation()

    // Test rate limiting
    testRateLimiting()
}

private fun testAuthenticationBypass() {
    // Try requests without authentication
    val request = Request.Builder()
        .url("https://api.example.com/users")
        .build()

    val response = httpClient.newCall(request).execute()

    // Should return 401 Unauthorized, not 200
    assert(response.code in 400..499) {
        "API endpoint should require authentication"
    }
}

private fun testAuthorizationFlaws() {
    // Try accessing other users' data
    val request = Request.Builder()
        .url("https://api.example.com/users/other-user-id")
        .addHeader("Authorization", "Bearer $currentUserToken")
        .build()

    val response = httpClient.newCall(request).execute()
    // Should return 403 Forbidden, not 200
    assert(response.code == 403) {
        "Should not access other users' data"
    }
}

private fun testInputValidation() {
    // Try SQL injection
    val request = Request.Builder()
        .url("https://api.example.com/search?q=' OR '1'='1")
        .build()

    val response = httpClient.newCall(request).execute()

    // Should handle safely, not expose database
    assert(!response.body?.string()?.contains("SQL")!!) {
        "Should not expose SQL errors"
    }
}

private fun testRateLimiting() {
    // Send 100 requests in rapid succession
    repeat(100) {
        val request = Request.Builder()
            .url("https://api.example.com/action")
            .post("".toRequestBody())
            .build()

        val response = httpClient.newCall(request).execute()

        // Eventually should get 429 Too Many Requests
        if (it > 50 && response.code == 429) {
            println("Rate limiting engaged after ${it+1} requests ✓")
            return
        }
    }

    assert(false) { "Rate limiting not implemented" }
}
```

}

3. Penetration Testing

// Penetration Testing Checklist

class PenetrationTestChecklist {

```kotlin
fun conductPenetrationTest(): TestReport {
    val findings = mutableListOf<Finding>()

    // 1. Network Security
    findings.addAll(testNetworkSecurity())

    // 2. Authentication & Session Management
    findings.addAll(testAuthenticationSecurity())

    // 3. Data Exposure
    findings.addAll(testDataExposure())

    // 4. Business Logic
    findings.addAll(testBusinessLogic())

    // 5. Client-Side Security
    findings.addAll(testClientSideSecurity())

    return TestReport(findings)
}

private fun testNetworkSecurity(): List<Finding> {
    val findings = mutableListOf<Finding>()

    // Check TLS version
    if (!checkMinimumTLSVersion("TLS1.2")) {
        findings.add(Finding(
            severity = Severity.HIGH,
            description = "TLS version < 1.2 is insecure"
        ))
    }

    // Check certificate validity
    if (!verifyCertificate()) {
        findings.add(Finding(
            severity = Severity.CRITICAL,
            description = "Invalid SSL certificate"
        ))
    }

    // Check for weak ciphers
    val weakCiphers = detectWeakCiphers()
    if (weakCiphers.isNotEmpty()) {
        findings.add(Finding(
            severity = Severity.HIGH,
            description = "Weak ciphers enabled: $weakCiphers"
        ))
    }

    return findings
```

```kotlin
}

private fun testAuthenticationSecurity(): List<Finding> {
    val findings = mutableListOf<Finding>()

    // Test default credentials
    if (tryDefaultCredentials()) {
        findings.add(Finding(
            severity = Severity.CRITICAL,
            description = "Default credentials not changed"
        ))
    }

    // Test password policy
    if (!checkPasswordPolicy()) {
        findings.add(Finding(
            severity = Severity.HIGH,
            description = "Weak password policy"
        ))
    }

    // Test session timeout
    if (sessionTimeout > 3600) {  // > 1 hour
        findings.add(Finding(
            severity = Severity.MEDIUM,
            description = "Session timeout too long"
        ))
    }

    return findings
}

private fun testDataExposure(): List<Finding> {
    val findings = mutableListOf<Finding>()

    // Check for hardcoded secrets
    if (searchForHardcodedSecrets()) {
        findings.add(Finding(
            severity = Severity.CRITICAL,
            description = "Hardcoded secrets found in code"
        ))
    }

    // Check data in logs
    if (sensitiveDataInLogs()) {
        findings.add(Finding(
            severity = Severity.HIGH,
            description = "Sensitive data exposed in logs"
        ))
    }
    return findings
}
```

}

data class Finding(
val severity: Severity,
val description: String,
val remediation: String = ""
)

enum class Severity { LOW, MEDIUM, HIGH, CRITICAL }

data class TestReport(val findings: List<Finding>) {
fun summary(): String {
val critical = findings.count { it.severity == Severity.CRITICAL }
val high = findings.count { it.severity == Severity.HIGH }
return "Critical: $critical, High: $high"
}
}

Deliverable:

- [ ] SAST tool integration (SonarQube, FindBugs, lint)
- [ ] DAST testing framework (API endpoint testing)
- [ ] Burp Suite interception guide
- [ ] Penetration testing checklist (20+ items)
- [ ] Vulnerability discovery process
- [ ] Security findings reporting template
- [ ] Remediation guidance for common issues
- [ ] Continuous security testing in CI/CD

Task 8: Security Monitoring and Incident Response (Self-Study: 1.5–2 hours)

Objective: Master security monitoring and incident response procedures.

Instructions:

1. Security Monitoring

// Real-time security event monitoring
class SecurityMonitoring(context: Context) {

```kotlin
private val analyticsClient = SecurityAnalyticsClient()

// Monitor suspicious activities
fun monitorSecurityEvents() {
    // Track authentication failures
    trackAuthenticationFailures()

    // Track unusual API access patterns
    trackAnomalousAPIAccess()

    // Track data access anomalies
    trackDataAccessAnomalies()

    // Track device tampering
    trackDeviceTampering()
}

private fun trackAuthenticationFailures() {
    val failedAttempts = analytics.getFailedLoginAttempts()

    if (failedAttempts > 5) {  // 5+ failures in 10 minutes
        analyticsClient.logSecurityAlert(
            eventType = "BRUTE_FORCE_DETECTED",
            severity = AlertSeverity.HIGH,
            details = mapOf(
                "failedAttempts" to failedAttempts,
                "timeWindow" to "10_minutes"
            )
        )

        // Implement rate limiting/account lockout
        triggerSecurityResponse()
    }
}

private fun trackAnomalousAPIAccess() {
    val apiAccessPattern = analytics.getAPIAccessPattern()
    val baseline = loadBaselinePattern()

    if (isAnomaly(apiAccessPattern, baseline)) {
        analyticsClient.logSecurityAlert(
            eventType = "ANOMALOUS_API_ACCESS",
            severity = AlertSeverity.MEDIUM,
            details = mapOf(
                "deviation" to calculateDeviation(apiAccessPattern,
baseline)
            )
        )
    }
}
```

```
private fun trackDeviceTampering() {
    // Monitor root/jailbreak indicators
    if (RootDetection().isDeviceRooted()) {
        analyticsClient.logSecurityAlert(
            eventType = "DEVICE_ROOTED",
            severity = AlertSeverity.CRITICAL,
            details = emptyMap()
        )

        // Disable sensitive features
        disableSensitiveFeatures()
    }

    // Monitor debugger attachment
    if (DebuggerDetection(context).isDebuggerAttached()) {
        analyticsClient.logSecurityAlert(
            eventType = "DEBUGGER_ATTACHED",
            severity = AlertSeverity.HIGH,
            details = emptyMap()
        )
    }
}

private fun triggerSecurityResponse() {
    // Lock account temporarily
    lockAccountTemporarily(durationMinutes = 15)

    // Send notification to user
    notifyUser("Unusual login activity detected")

    // Require additional verification
    requireMFAVerification()
}
```

}

enum class AlertSeverity { LOW, MEDIUM, HIGH, CRITICAL }

// Security event logging
class SecurityEventLogger {

```
data class SecurityEvent(
    val timestamp: Long,
    val eventType: String,
    val userId: String,
    val action: String,
    val status: String,
    val details: Map<String, String>
)
```

```
fun logSecurityEvent(event: SecurityEvent) {
    // Log to secure backend (immutable, tamper-proof)
    sendToSecurityBackend(event)

    // Also log locally for offline capability
    saveLocally(event)
}

private fun sendToSecurityBackend(event: SecurityEvent) {
    val json = json {
        "timestamp" to event.timestamp
        "eventType" to event.eventType
        "userId" to event.userId
        "action" to event.action
        "status" to event.status
        "details" to event.details
    }

    val request = Request.Builder()
        .url("https://security.example.com/events")

.post(json.toString().toRequestBody("application/json".toMediaType()))
        .build()

    httpClient.newCall(request).execute()
}
```

}

2. Incident Response Plan

// Incident Response Framework
class IncidentResponsePlan {

```
// Step 1: Detection & Analysis
fun detectIncident(alert: SecurityAlert) {
    val severity = calculateSeverity(alert)

    when (severity) {
        AlertSeverity.CRITICAL -> escalateToSecurity()
        AlertSeverity.HIGH -> assignToSecurityTeam()
        AlertSeverity.MEDIUM -> logAndMonitor()
        else -> {}
    }
}

// Step 2: Containment
fun containIncident(incident: Incident) {
    // Isolate affected systems
    quarantineUser(incident.affectedUserId)

    // Revoke compromised tokens
    revokeSessions(incident.affectedUserId)

    // Reset credentials
    forcePasswordReset(incident.affectedUserId)

    // Block malicious IPs
    blockIPRange(incident.attackerIP)
}

// Step 3: Eradication
fun eradicateIncident(incident: Incident) {
    // Remove malware/backdoors
    scanAndClean()

    // Patch vulnerabilities
    deploySecurityPatches()

    // Update defense mechanisms
    updateFirewallRules()
}

// Step 4: Recovery
fun recoverFromIncident(incident: Incident) {
    // Restore from clean backups
    restoreData(incident.affectedUserId)

    // Verify system integrity
    runIntegrityChecks()

    // Gradually restore access
    restoreAccessGradually(incident.affectedUserId)
}

// Step 5: Post-Incident Review
fun conductPostIncidentReview(incident: Incident) {
    val report = createIncidentReport(incident)

    // Root cause analysis
    analyzeRootCause(incident)

    // Identify lessons learned
    identifyImprovements(incident)

    // Update security controls
    updateSecurityControls(incident)

    // Communicate findings
    notifyStakeholders(report)
}
```

}

data class Incident(
val incidentId: String,
val detectionTime: Long,
val severity: AlertSeverity,
val description: String,
val affectedUserId: String,
val attackerIP: String,
val scope: List<String>
)

3. Compliance Reporting

// Compliance Audit Trail
class ComplianceReporting {

```
// Generate OWASP MASTG compliance report
fun generateMASTGComplianceReport(): ComplianceReport {
    val findings = mutableListOf<ComplianceFinding>()

    // Storage
    findings.addAll(auditDataStorage())

    // Encryption
    findings.addAll(auditEncryption())

    // Authentication
    findings.addAll(auditAuthentication())

    // Network
    findings.addAll(auditNetworkSecurity())

    // Code Quality
    findings.addAll(auditCodeQuality())

    return ComplianceReport(findings,
calculateComplianceScore(findings))
}

private fun auditDataStorage(): List<ComplianceFinding> {
    return listOf(
        ComplianceFinding(
            control = "MASTG-STORAGE-1",
            description = "Sensitive data not exposed in logs",
            status = checkNoSensitiveDataInLogs()
        ),
        ComplianceFinding(
            control = "MASTG-STORAGE-2",
            description = "Sensitive data encrypted at rest",
            status = checkDataEncryption()
        )
    )
}

private fun calculateComplianceScore(findings:
List<ComplianceFinding>): Float {
    val passed = findings.count { it.status == ComplianceStatus.PASS }
    return passed.toFloat() / findings.size
}
```

```
}

data class ComplianceReport(
val findings: List<ComplianceFinding>,
val complianceScore: Float
) {
fun summary(): String {
val passed = findings.count { it.status == ComplianceStatus.PASS }
return "passed/{findings.size} controls passed (${(complianceScore * 100).toInt()}%)"
}
}

data class ComplianceFinding(
val control: String,
val description: String,
val status: ComplianceStatus
)

enum class ComplianceStatus { PASS, FAIL, PARTIAL, NOT_APPLICABLE }
```

Deliverable:

- [ ] Real-time security event monitoring framework
- [ ] Alert escalation and response procedures
- [ ] Incident detection playbooks (5+ scenarios)
- [ ] Incident response plan with 5 phases
- [ ] Post-incident review process
- [ ] Security event logging (immutable, tamper-proof)
- [ ] OWASP MASTG compliance reporting
- [ ] Compliance score calculation

Evaluation Criteria

Criteria	Excellent (9–10)	Good (7–8)	Acceptable (5–6)	Needs Improvement
Threat Modeling	Comprehensive STRIDE/PASTA analysis	STRIDE or PASTA implemented	Basic threat enumeration	Minimal threat analysis
Secure Coding	5+ secure patterns implemented	3-4 patterns implemented	Basic patterns	Lacks implementation
Cryptography	Symmetric + asymmetric + hashing	Multiple algorithms	Single approach	Generic understanding
Authentication	OAuth 2.0 + OIDC + MFA	OAuth or OIDC + token mgmt	Basic auth	Simple auth
Data Protection	Keystore + encrypted DB + clear memory	2 storage methods	Single method	Basic storage
Reverse Eng. Defense	Obfuscation + debugger detection + integrity checks	2 defenses	Single defense	None
Security Testing	SAST + DAST + penetration testing	2 testing types	Single testing type	No testing
Compliance	OWASP + GDPR + HIPAA	2 frameworks	Single framework	Basic awareness
Code Quality	Clean, production-ready	Well-structured	Readable	Difficult to follow
Documentation	Clear explanations, diagrams	Good explanations	Basic explanations	Minimal documentation

---

Deliverable Checklist

Submit as DOCX (7–9 pages):

- [ ] Task 1: Threat modeling (STRIDE/PASTA/attack trees) + risk assessment
- [ ] Task 2: Risk assessment framework + OWASP Top 10 + compliance (GDPR/HIPAA)
- [ ] Task 3: Input validation + secure coding + error handling + CSRF/XSS

- [ ] Task 4: Encryption (AES/RSA/PBKDF2) + Keystore/Keychain + key derivation
- [ ] Task 5: OAuth 2.0 + OpenID Connect + certificate pinning + mutual TLS
- [ ] Task 6: Secure storage + reverse engineering defense + tamper detection
- [ ] Task 7: Security testing (SAST/DAST/penetration) + vulnerability assessment
- [ ] Task 8: Security monitoring + incident response + compliance reporting

---

Resources & References

[1] OWASP. (2024). Mobile Security Testing Guide (MASTG). https://mobile-security.gitbook.io

[2] OWASP. (2024). Top 10 Mobile Risks. https://owasp.org/www-project-mobile-top-10/

[3] OWASP. (2024). Secure Coding Practices. https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

[4] Google. (2024). Android Security & Privacy Practices. https://developer.android.com/privacy-and-security

[5] Apple. (2024). Security Documentation. https://developer.apple.com/security/

[6] NIST. (2024). Cybersecurity Framework. https://www.nist.gov/cyberframework/

[7] European Commission. (2023). General Data Protection Regulation (GDPR). https://ec.europa.eu/info/law/law-topic/data-protection_en

[8] HHS. (2024). HIPAA Security Rule. https://www.hhs.gov/hipaa/for-professionals/security/

[9] CWE/SANS. (2024). Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/

[10] Schneider, B., & Ferguson, N. (2003). *Practical Cryptography*. John Wiley & Sons.

[11] Microsoft. (2024). Threat Modeling Tool. https://www.microsoft.com/en-us/securityengineering/tmtoolkit

[12] Burp Suite. (2024). Web Security Testing. https://portswigger.net/burp

---

Timeline Guidance

Self-Study (9–11 hours):

- Hours 1–2.5: Task 1 (threat modeling, risk assessment)
- Hours 2.5–4.5: Task 2 (compliance, GDPR/HIPAA)
- Hours 4.5–6: Task 3–4 (secure coding, encryption)

- Hours 6–7.5: Task 5–6 (authentication, data protection)
- Hours 7.5–9: Task 7 (security testing)
- Hours 9–11: Task 8 (monitoring, incident response)

Online Sessions (2–2.5 hours):

- 0.5 hours: Threat modeling methodology
- 0.5 hours: Encryption and authentication protocols
- 0.5 hours: Secure coding practices demonstration
- 0.5 hours: Security testing tools and compliance

Extension Tasks (Optional)

1. Implement end-to-end encryption for user communications
2. Build Hardware Security Module (HSM) integration
3. Create mobile security certification program
4. Implement differential privacy for analytics
5. Build secure multiparty computation (MPC) system
6. Develop zero-knowledge proof authentication
7. Create bug bounty program infrastructure
8. Build security awareness training platform