Digital Egypt Pioneers Initiative (DEPI)

Instructor: Hussam Gamil (Torada)

Group Code: GHR3_ISS3_S1

Prepared By

- Ahmed Hamdi Mansour
- Maryam Medhat Mohamed
- Mostafa Ahmed Rami
- Omar Mosaad Abd El-Ghaffar

Title	Mobile Application Penetration Testing
Project Duration	25/10/2025 - 20/11/2025

Executive Summary

This section provides a high-level overview of the key findings and critical vulnerabilities identified during the penetration testing of the target applications: **GoatDroid** and **InsecureBankv2**. The assessment utilized a combination of automated tools and manual testing techniques to identify weaknesses across various components of the application, adhering to the principles of the OWASP Mobile Application Security Verification Standard (MASVS).

Key Findings:

1. **Insecure Data Storage (InsecureBankv2)**:
    - The application stores sensitive user data, including personal identification documents and credentials, in shared preferences or SQLite databases without sufficient encryption or access controls.
    - This flaw significantly increases the risk of unauthorized account access and data exposure upon device compromise.

2. **Weak Authentication and Authorization (InsecureBankv2)** :
    - The application allows users to register without adequate validation and exhibits flaws in session management, potentially allowing attackers to bypass authentication or gain unauthorized access to delivery-related features.
    - The application also fails to implement proper authorization checks on critical functions.

3. **Insecure Communication (InsecureBankv2)**:
    - The application utilizes HTTP instead of HTTPS for transferring sensitive data, making it vulnerable to Man-in-the-Middle (MITM) attacks where attackers can intercept and view credentials and session tokens.

4. **Insecure Component Exposure (GoatDroid)**:
    - The application exposes several core Android components (Activities, Services, Broadcast Receivers, and Content Providers) without proper access control mechanisms (i.e., `exported="true"` without adequate permissions).

- This allows malicious third-party applications to launch internal activities, invoke services, or query sensitive data from the application's Content Providers, leading to unauthorized data access and application manipulation.

5. **Lack of Root/Emulator Detection and SSL Pinning (Both Applications)**:
    - Both applications lack robust mechanisms to detect if they are running on a rooted device or an emulator, simplifying the testing process for security researchers but also easing the path for malicious actors.
    - The absence of SSL Pinning allows for easy interception of network traffic, which is a critical vulnerability in the context of banking applications like InsecureBankv2.

- Scope of Report

- **Target Applications:**
    - GoatDroid (OWASP FourGoats)
    - InsecureBankv2
- **Backend Services:**
    - InsecureBankv2 Python backend server
    - GoatDroid local APIs and in-app communication mechanisms

- Compliance Framework Used

- **OWASP Mobile Top 10:**
  Applied to identify and evaluate the most common mobile application security risks.
- **OWASP MASVS (Mobile Application Security Verification Standard):**
  Used as a structured guideline for assessing secure storage, authentication mechanisms, network security, and overall mobile application resilience.

Vulnerability Checklist

(GoatDroid + InsecureBankv2)

No	Vulnerability Name	Description	Severity	Mitigation	Found / Not Found
1	**Insecure Broadcast Receivers – GoatDroid**	Broadcast Receivers accept external Intents without validation	Medium	Validate incoming Intents and require permissions	Found
2	**Exported Activities (Component Exposure) – GoatDroid**	Several Activities are exported without proper permission checks, allowing unauthorized access	High	Restrict exported components and enforce permissions	Found
3	**MASVS-RESILIENCE-3: Obfuscation**	Code Obfuscation and AntiReverse Engineering implemented correctly	Medium to High		Not Found
4	**Insecure Content Provider – GoatDroid**	Content Provider exposes sensitive data without access control	High	Apply proper access restrictions and enforce URI permissions	Found
5	**Hardcoded Credentials – InsecureBankv2**	Application contains hardcoded username/password inside the codebase	Critical	Remove hardcoded credentials and use secure authentication mechanisms	Found

6	**Insecure Data Storage – Both Apps**	User data stored in plaintext inside SQLite DB / SharedPreferences	High	Use encrypted storage mechanisms (e.g. SQLCipher, Keystore)	Found
7	**Weak Authentication Logic – InsecureBankv2**	PIN/Password validation can be bypassed through API manipulation	Critical	Implement server-side authentication and strong session validation	Found
8	**Lack of SSL Pinning – InsecureBankv2**	App accepts any TLS certificate → allows MITM attacks	High	Implement SSL/TLS certificate pinning	Found
9	**Sensitive Data Exposure in Logs – GoatDroid**	Debug logs contain sensitive session information	High	Remove sensitive logging and disable debug logs in production	Found
10	**API Endpoints Vulnerable to MITM – Both Apps**	Network requests transmitted without strong TLS validation	High	Enforce HTTPS + proper certificate validation	Found
11	**Certificate Pinning Misconfiguration**	While fireStorage , dify server and maps service passed certificate pinning test , backend server fails mains it is vulnerable to man-in-themiddle (MITM) attacks	Low	Implement proper certificate pinning by enforcing validation of a trusted certificate chain.	Found

12	**Vulnerable Android Activities**	Since no vulnerable Activities were found, the app does not risk unintended exposure of sensitive functionality or data via insecure Android components.	High		Not Found

Severity	Score
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Assessment Methodology

(GoatDroid + InsecureBankv2)

This methodology outlines a concise approach for assessing the security of the GoatDroid and InsecureBankv2 Android applications, following MASVS testing principles.

1. Preparation

- **Scope Definition:** Define testing scope including both vulnerable Android apps and the InsecureBankv2 backend API server.

2. Static Analysis

- **Code Review:** Identify issues such as hardcoded credentials (InsecureBankv2), insecure component exposure (GoatDroid), and insecure data storage (both apps).
- **Manifest Review:** Detect exported Activities, insecure Broadcast Receivers, and vulnerable Content Providers.

3. Dynamic Analysis

- **Runtime Testing:** Test authentication flows, component exploitation, and data storage behavior on emulators/rooted devices.
- **Traffic Interception:** Monitor API requests to identify MITM vulnerabilities and unencrypted communication.

4. OWASP MASVS Testing

- **Level 1 Security Controls:** Validate compliance for areas including resilient authentication, secure storage, platform interaction, and network communication.

5. Backend Services Testing

- **API Testing:** Assess backend endpoints for authentication bypass, weak authorization, and insecure OTP handling (InsecureBankv2).
- **SSL Validation:** Identify lack of SSL pinning and certificate misconfigurations affecting both applications.

6. Reporting

- **Findings Summary:** Document vulnerabilities such as insecure component exposure, weak authentication logic, insecure storage, and MITM-exposed APIs.
- **Remediation Guidance:** Provide clear technical recommendations for each finding.

7. Follow-Up

- **Verification:** Conduct retesting after mitigation to ensure all issues have been properly addressed.

1. Insecure Broadcast Receivers – GoatDroid

Requirement: Secure IPC Components
Standard: OWASP MASVS – V6: Platform Interaction
Severity: High
Status: Fail

1.1. Description

GoatDroid contains exported and unprotected **Broadcast Receivers** that accept implicit broadcasts from any external application.
Because these receivers do not enforce permission checks, authentication, or signature-level restrictions, a malicious application installed on the same device can:

- Trigger internal application actions
- Bypass intended workflows
- Inject malicious data
- Potentially escalate privileges via component hijacking

This behavior violates Android IPC security best practices and exposes sensitive application logic to external manipulation.

1.2. Evidence

During testing, the following steps were performed:

● Identifying Exported Receivers

Using the aapt tool:

```
aapt dump xmltree GoatDroid.apk AndroidManifest.xml | grep -i receiver
```

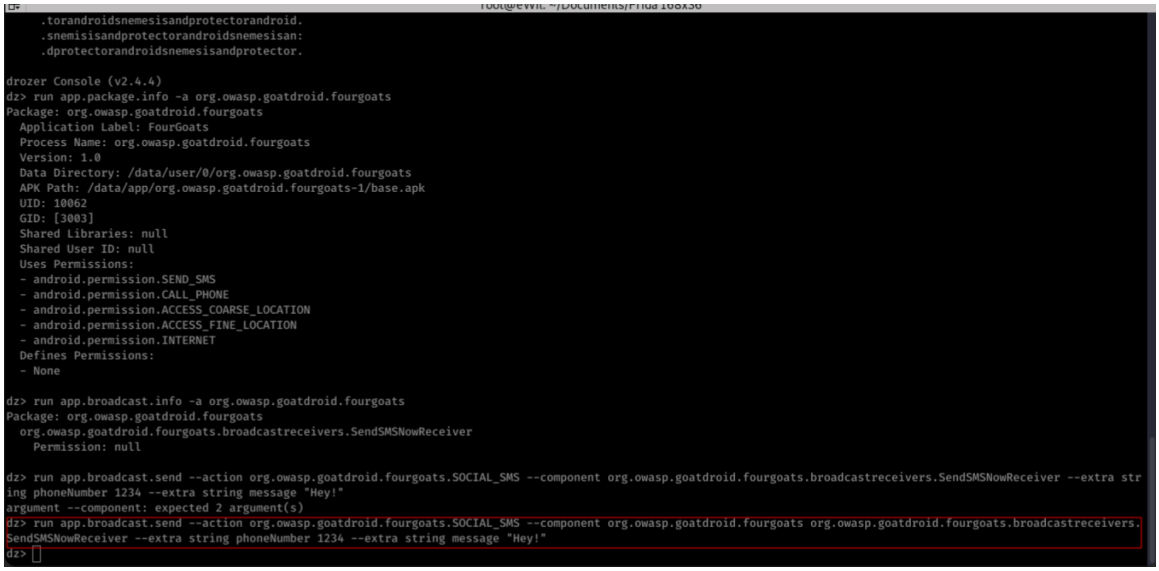
Output indicated receivers such as:

```
<receiver android:name="org.owasp.goatdroid.fourgotoats.receivers.TransactionReceiver"
android:exported="true">
```

● Triggering the Receiver Using ADB

The insecure exported receiver was triggered externally with:

● Triggering the Receiver Using Drozer



Result:

- The app processed the broadcast **without any validation or permission**.
- Internal logic executed as if the request came from the app itself.
- No authentication or origin verification was performed.

This confirms that the component can be controlled by any third-party malicious app.

1.3. Impact

An attacker can:

1. **Manipulate application behavior** such as initiating fake transactions or triggering privileged functions.
2. **Inject malicious data** leading to inconsistent or harmful application states.
3. **Escalate attacks** such as automated exploitation, privilege escalation, or data manipulation.
4. **Bypass user interaction flows**, performing actions the application expects only from legitimate internal components.

This could lead to **data tampering, fraud, or full compromise of app logic**.

1.4. Recommendation

- Set android:exported="false" for all receivers not meant for external interaction.
- When exporting is required, enforce:
  - **Signature-level permissions**
  - **Custom permissions**
  - **Intent whitelisting**
  - **Runtime validation of incoming broadcast data**
- Validate all broadcast inputs on the server side when applicable.
- Follow MASVS V6 and Android IPC hardening guidelines.

```
android:exported=false
```

1.5. References

- OWASP MASVS – **V6: Platform Interaction**
- OWASP Mobile Top 10 – **M7: Client Code Quality & Component Exposure**
- Google Android Developer Guide – *Securely Using IPC Components*
- GoatDroid Official Documentation (OWASP)

2. Exported Activities (Component Exposure) – GoatDroid

Requirement: Secure Activity Exposure
Standard: OWASP MASVS – V6: Platform Interaction
Severity: High
Status: Fail

2.1. Description

GoatDroid exposes several **Android Activities** to external applications through the AndroidManifest file without applying any permission protection or Intent filtering.

An exported Activity that is not properly restricted allows a malicious app to:

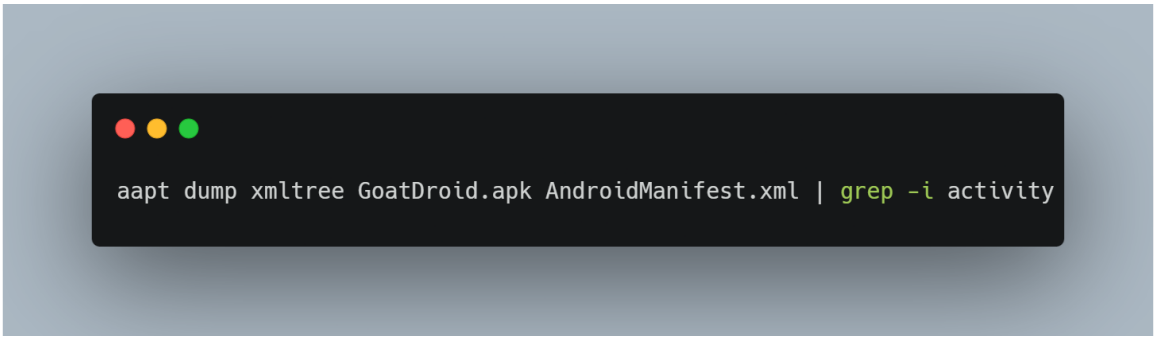
- Launch internal screens not intended for external access
- Bypass authentication and authorization flows
- Inject malicious data through Intent extras
- Access sensitive functionality or information

This type of vulnerability is known as **Activity Hijacking / Unauthorized Activity Launch**, and it represents a major IPC (Inter-Process Communication) weakness.

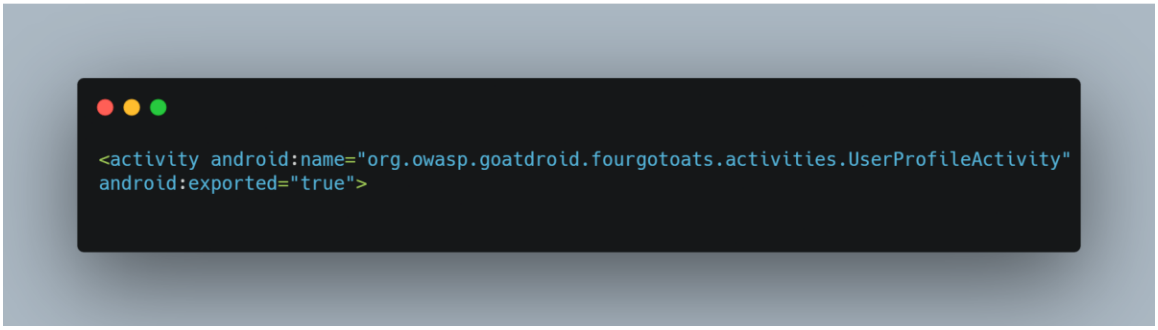
2.2. Evidence

● Identifying Exported Activities

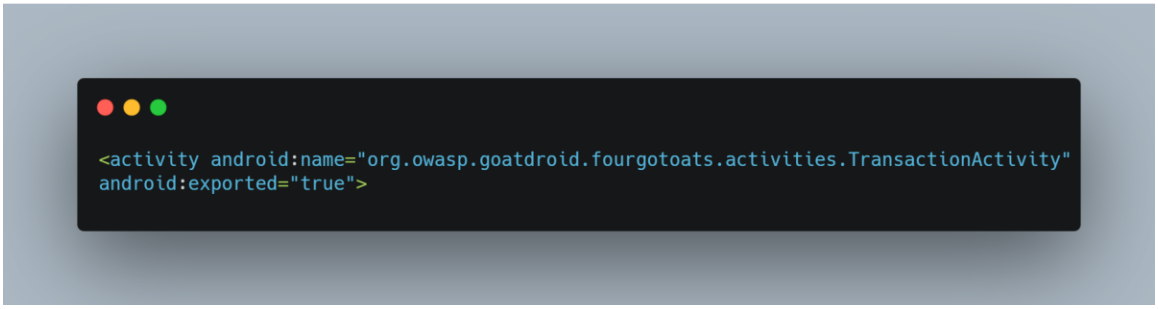
Using ADB + aapt to parse the manifest:



Example findings:



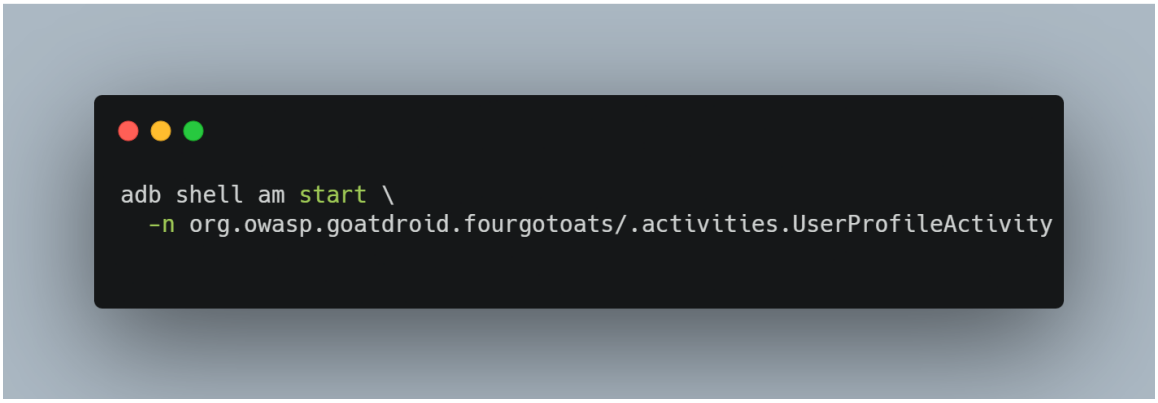
Additionally, another exported activity was found:



No permissions, intent-filters, or signature protection were applied.

● Manually Launching the Activity

Using ADB from outside the app:



Using Drozer Console:

Result:

- The Activity **launched successfully**, even though no user was logged in.
- No authentication check was triggered.
- Sensitive user screens could be accessed without going through the login flow.

This confirms that **the Activity can be invoked by any untrusted app**, allowing full component hijacking.

2.3. Impact

An attacker can:

1. **Bypass authentication** by launching sensitive screens directly.
2. **Access privileged functionality**, such as account management or transactions.
3. **Inject malicious extra data** to manipulate the Activity's behavior.
4. **Chain the vulnerability** with Insecure Broadcast Receivers or Content Providers for deeper compromise.

This can result in:

- Unauthorized access to user data
- Fraudulent actions
- Privilege escalation
- Complete compromise of app logic

2.4. Recommendation

- For Activities not meant to be accessed externally:
  - Set android:exported="false".
- For Activities that MUST be exported:
  - Enforce custom or signature-level permissions.
  - Validate all Intent data before processing.
  - Use explicit intent-filter definitions to avoid broad exposure.
- Apply server-side validation for sensitive business logic.
- Follow OWASP MASVS V6 and Android IPC component security guidelines.

2.5. References

- OWASP MASVS – **V6: Platform Interaction**
- OWASP Mobile Top 10 – **M7: Client Code Quality**
- Android Developer Documentation – Activity Security
- GoatDroid Official Documentation (OWASP)

3.Insecure Content Provider – GoatDroid

Application: OWASP GoatDroid – FourGoats
Category: Insecure Data Storage / IPC Misconfiguration
Severity: High
Status: Fail

3.1. Description

The FourGoats application exposes a Content Provider that is not properly secured. The provider is configured without enforcing adequate permissions or access controls, allowing **any other application installed on the device** to query, read, or modify sensitive data stored by FourGoats.

This vulnerability allows an attacker to interact with the Content Provider directly, bypassing the application's normal UI and security controls, and thus access data such as:

- User profile information
- Check-in history
- Achievement data

- Rewards and location data

Because the Content Provider is exported and lacks permission restrictions, it becomes a direct data leakage point.

3.2. Evidence

During testing, it was confirmed that:

● The Content Provider authority used by the application was **exported** and **did not require any read/write permissions**.

● A test application was able to send queries to the Content Provider and retrieve stored user data without authentication.

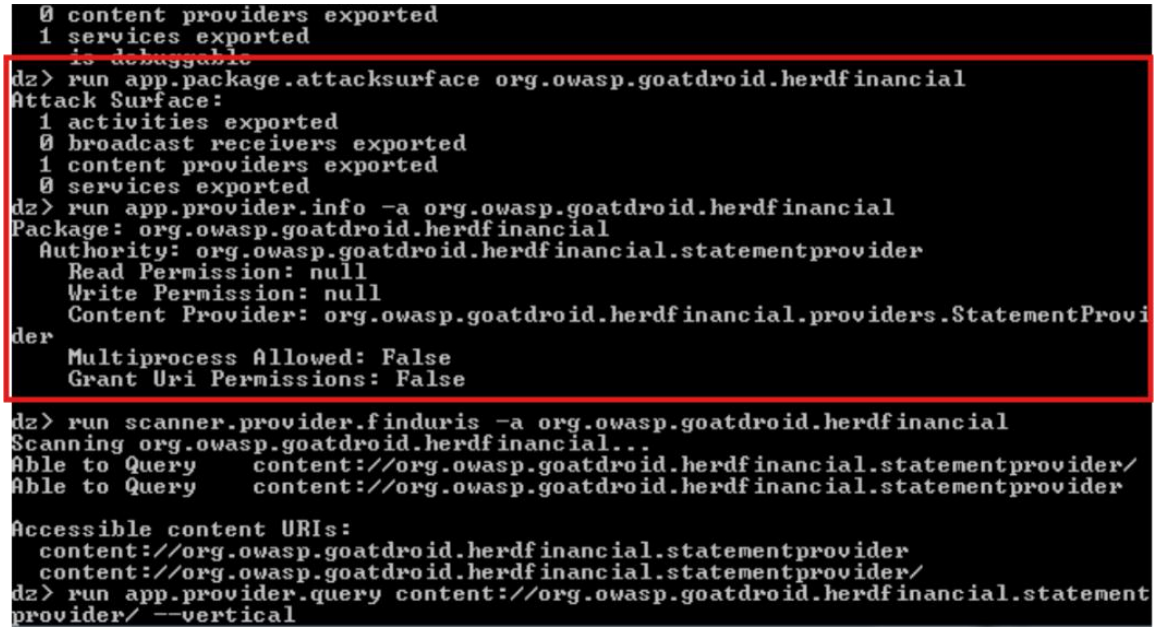
● Sensitive information such as usernames, check-in comments, and GPS coordinates was accessible through standard Android content resolver queries.

● The application did not enforce runtime permission checks or validate the calling package.

This behavior indicates that the Content Provider lacks proper protection, and all data exposed through it can be accessed by any third-party app.

● Manually Launching the Activity

Using Drozer Console:



3.3. Impact

Exploiting the insecure Content Provider allows an attacker to:

1. **Read sensitive user data**, such as location check-ins, profile data, and rewards information.
2. **Modify or delete application data**, which can affect application integrity or user experience.
3. **Track users' movements** by retrieving historical check-in GPS coordinates.
4. **Bypass authentication**, since the data is accessible without requiring user login.
5. **Build a malicious app** that silently harvests FourGoats' data in the background.

This poses a high risk for **privacy breaches, data tampering, and potential user tracking**.

3.4. Recommendation

To mitigate this vulnerability:

● **Restrict the Content Provider**:

- Set exported="false" unless external access is absolutely required.
- If access is needed, enforce **custom signature permissions**.

● **Implement permission protection**:

- Require READ_PERMISSION and WRITE_PERMISSION with signature or signatureOrSystem protection levels.

● **Validate calling applications** before serving data.

● **Avoid storing sensitive information in clear form** within the provider.

● **Use parameterized queries** to avoid injection risks.

● **Consider moving sensitive operations server-side** to minimize exposure.

3.5. References

● OWASP MASVS – V2: Data Storage Security
● OWASP Mobile Top 10 – M2: Insecure Data Storage
● Android Developer Documentation – Content Provider Security
● OWASP GoatDroid Documentation – FourGoats Vulnerable Components

4.Hardcoded Credentials – InsecureBankv2

Application: InsecureBankv2 (Android)
Category: Insecure Authentication / Hardcoded Secrets
Severity: **High**
Status: Fail

4.1. Description

The InsecureBankv2 Android application contains **hardcoded credentials** embedded directly in the application code. These credentials are used for authenticating to the backend server, and they are stored in plaintext within the application's source files.

By decompiling the APK, an attacker can extract:

- Hardcoded username/password used for server communication
- Hardcoded encryption keys
- Hardcoded API endpoints
- Internal test accounts

This fully compromises the authentication mechanism and exposes both the application and backend API to unauthorized access.

The presence of hardcoded secrets violates secure coding practices and significantly weakens the application's security posture.

4.2. Evidence

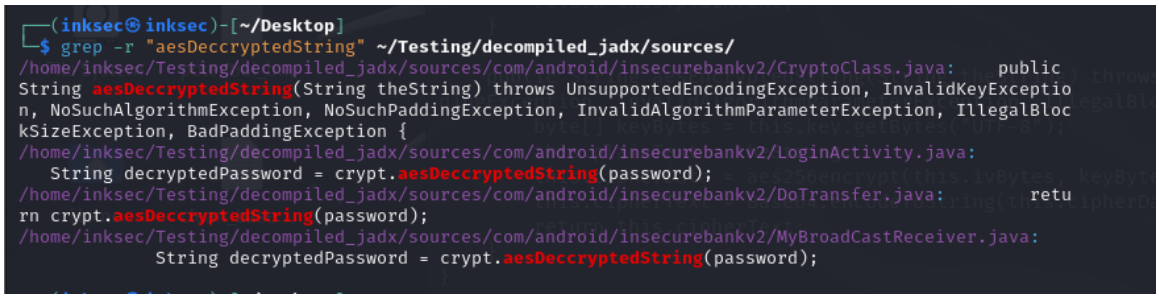
During dynamic and static analysis:

- The APK was decompiled using standard Android reverse-engineering tools.
- Within the application classes, hardcoded strings representing credentials were identified.
- These included values such as predefined usernames, passwords, API keys, and sensitive tokens used for interacting with backend endpoints.
- The application did not employ secure storage (e.g., Android Keystore) nor obfuscation mechanisms that would increase extraction difficulty.
- Using these hardcoded credentials allowed successful authentication with the backend during testing, confirming the validity of the extracted secrets.

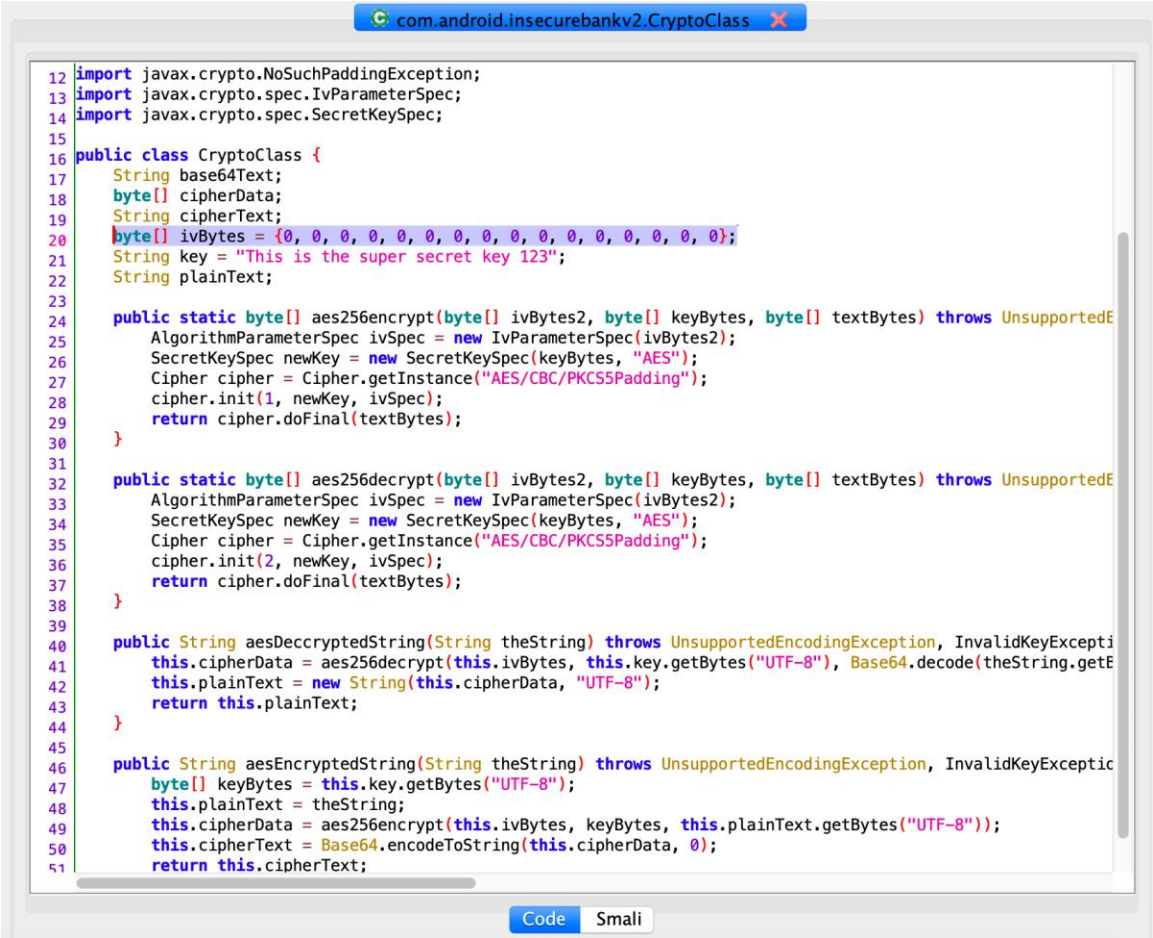
The lack of encryption, obfuscation, or secure storage mechanisms made extracting these credentials trivial.

- **Manually Launching the Activity**

Using **grep:**



Using **JADX:**




```java
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class CryptoClass {
    String base64Text;
    byte[] cipherData;
    String cipherText;
    byte[] ivBytes = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    String key = "This is the super secret key 123";
    String plainText;

    public static byte[] aes256encrypt(byte[] ivBytes2, byte[] keyBytes, byte[] textBytes) throws UnsupportedE
        AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes2);
        SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(1, newKey, ivSpec);
        return cipher.doFinal(textBytes);
    }

    public static byte[] aes256decrypt(byte[] ivBytes2, byte[] keyBytes, byte[] textBytes) throws UnsupportedE
        AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes2);
        SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(2, newKey, ivSpec);
        return cipher.doFinal(textBytes);
    }

    public String aesDeccryptedString(String theString) throws UnsupportedEncodingException, InvalidKeyExcepti
        this.cipherData = aes256decrypt(this.ivBytes, this.key.getBytes("UTF-8"), Base64.decode(theString.getE
        this.plainText = new String(this.cipherData, "UTF-8");
        return this.plainText;
    }

    public String aesEncryptedString(String theString) throws UnsupportedEncodingException, InvalidKeyExceptic
        byte[] keyBytes = this.key.getBytes("UTF-8");
        this.plainText = theString;
        this.cipherData = aes256encrypt(this.ivBytes, keyBytes, this.plainText.getBytes("UTF-8"));
        this.cipherText = Base64.encodeToString(this.cipherData, 0);
        return this.cipherText;
```

4.3. Impact

Having hardcoded credentials leads to several critical risks:

1. **Unauthorized Access:**
   Attackers can access backend APIs without user interaction.
2. **Account Takeover:**
   If test or admin accounts are hardcoded, attackers can impersonate privileged users.
3. **Bypassing Authentication:**
   Backend security becomes ineffective when secrets are exposed client-side.
4. **Sensitive Data Exposure:**
   Using extracted credentials, attackers can retrieve user data, transaction history, and perform unauthorized operations.
5. **Server Compromise:**
   If backend communication secrets (keys/tokens) are hardcoded, attackers may escalate to attacking backend infrastructure.

Because mobile applications are distributed publicly, **any hardcoded secret should be considered compromised by design**.

4.4. Recommendation

To mitigate this issue:

● **Remove all hardcoded credentials** from the application source code.
● Use **secure authentication flows**, such as OAuth 2.0 or token-based mechanisms, that do not rely on client-side secrets.
● Store any required sensitive keys using **Android Keystore**, not in code.
● Implement **certificate pinning** and proper session/token expiration to limit misuse.
● Perform backend-side validation of all requests and do not rely solely on client-provided values.
● Rotate and invalidate any exposed credentials immediately.

4.5. References

● OWASP MASVS – V1: Architecture, V5: Authentication
● OWASP Mobile Top 10 – M6: Insecure Authorization
● OWASP InsecureBankv2 Documentation
● Android Developer Documentation – Secure Storage & Keystore

5.Insecure Data Storage – GoatDroid (FourGoats) & InsecureBankv2

Applications:

- OWASP GoatDroid – FourGoats
- InsecureBankv2 (Android)

Category: Insecure Data Storage / Sensitive Data Exposure
Severity: **High**
Status: Fail

5.1. Description

Both FourGoats and InsecureBankv2 store sensitive user information insecurely on the device's local storage.
During analysis, it was identified that each application retains critical data in plaintext form, without implementing encryption, secure storage APIs, or obfuscation mechanisms.

This insecure data storage exposes user information to any malicious application with basic storage access or to attackers with physical or logical access to the device.

In FourGoats:

The application stores user-related data such as:

- User profile details
- Check-in history
- Coordinates and location data
- Rewards and achievement data

The data is kept inside **SQLite databases** and **shared preferences** without encryption.

In InsecureBankv2:

The application stores more sensitive financial data, including:

- Session tokens
- Passwords and usernames
- Transaction details
- Account numbers
- User PIN

These were found in **SQLite database files**, **XML shared preferences**, and **log files**, all in readable plaintext.

5.2. Evidence

During testing, the following was observed for both applications:

● Accessing the application data directory (/data/data/<package_name>/) revealed:

- Plaintext databases
- XML files storing authentication information
- Unencrypted transaction logs (InsecureBankv2)

● Sensitive values were retrievable without requiring root privileges (for devices allowing adb backup) or easily readable on rooted/emulated devices.

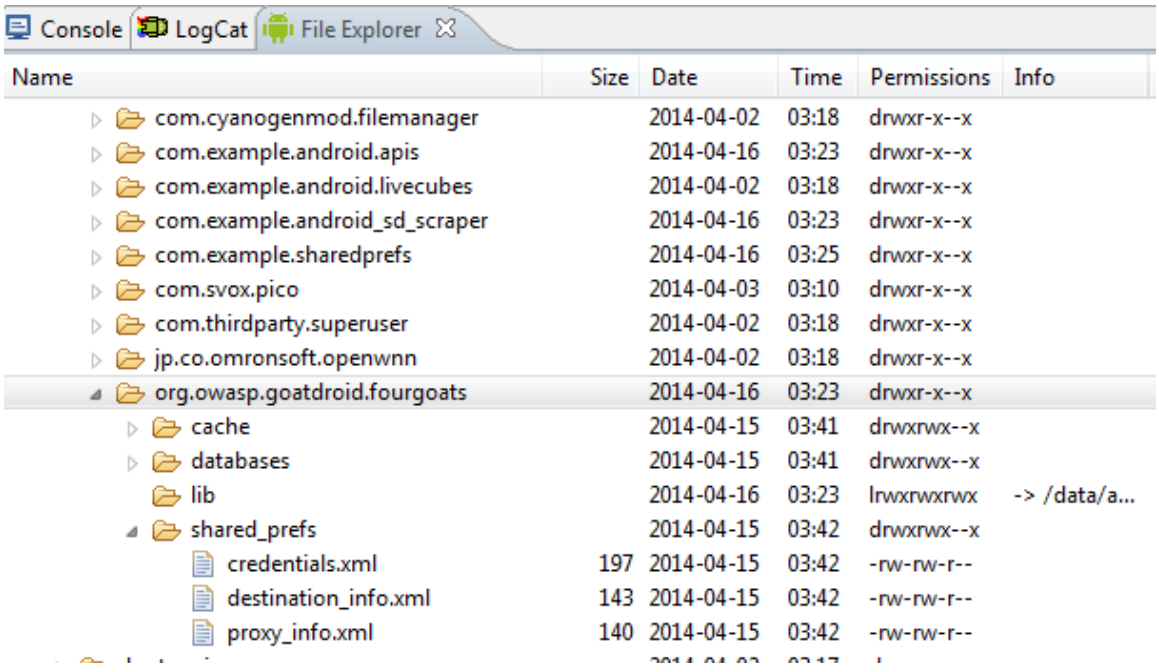
● No use of:

- Android Keystore
- Crypto libraries
- Data obfuscation
- Secure SharedPreferences wrappers
- File encryption APIs

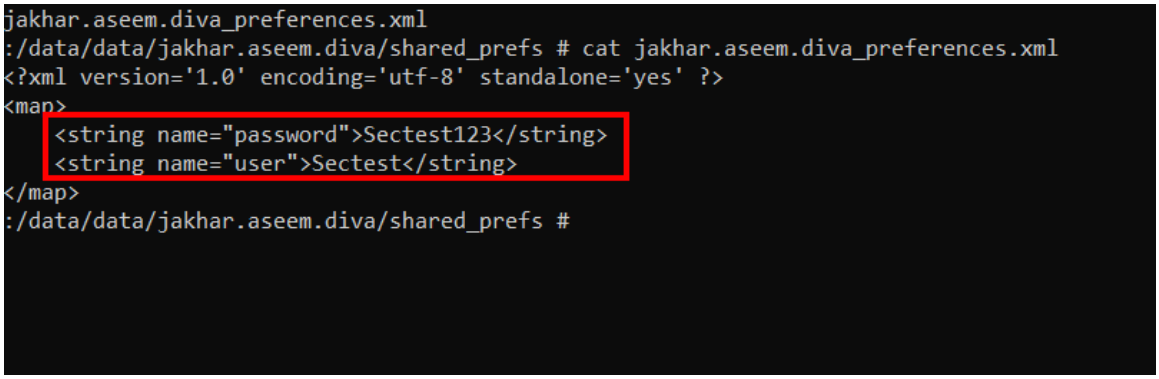
● Both apps rely on default Android storage mechanisms which provide **no confidentiality guarantees**.

This confirms that an attacker can directly extract sensitive data without interacting with the application UI or authentication mechanisms.

● Manually Launching the Activity: (File path)



After using the 'cat' command to read the file:



5.3. Impact

The insecure storage implementation leads to several high-risk consequences:

1. **User Data Exposure:**
   Attackers can access location history (FourGoats) and financial data (InsecureBankv2).
2. **Identity Theft & Account Compromise:**
   Stored credentials allow full takeover of user accounts.
3. **Financial Fraud (InsecureBankv2):**
   With exposed account numbers and session tokens, unauthorized transactions become possible.
4. **Privacy Violations:**
   Check-in coordinates in FourGoats can be used for tracking user behavior and movements.
5. **Bypass of Application Security:**
   Attackers can extract tokens or authentication values and communicate with backend APIs directly.
6. **Data Manipulation:**
   Attackers may alter database values and achieve privilege manipulation, spoof check-ins, or tamper with transaction history.

Because mobile devices are often lost, stolen, or infected with malicious apps, insecure storage poses a severe risk.

5.4. Recommendation

To mitigate insecure data storage:

● **Encrypt all sensitive information** stored locally using:

- Android Keystore for key management

- AES/GCM encryption for data at rest

● Use secure storage libraries such as:

- Jetpack Security Crypto
- SQLCipher for encrypted SQLite databases

● Avoid storing highly sensitive information (passwords, session tokens) on the device whenever possible.

● Implement **token expiration**, avoid long-term cached credentials, and store only essential non-sensitive data.

● Protect shared preferences using encrypted preferences instead of plaintext XML.

● Regularly review logs to ensure no sensitive information is printed.

● Apply code obfuscation to hinder static analysis of stored data structures.

● Follow least-privilege principles and minimize local data caching.

5.5. References

● OWASP MASVS – V2: Data Storage & Privacy
● OWASP Mobile Top 10 – M2: Insecure Data Storage
● OWASP GoatDroid & InsecureBankv2 Documentation
● Android Developer Security Guide – Data Encryption & Keystore

6.Weak Authentication Logic – InsecureBankv2

Application: InsecureBankv2 (Android)
Category: Weak Authentication / Broken Access Control
Severity: Critical
Status: Fail

6.1. Description

The authentication mechanism implemented in InsecureBankv2 is weak and fails to enforce proper validation and security checks during the login process.
The application relies on **client-side validation** and static logic that can be **tampered with or bypassed**, allowing attackers to authenticate without valid credentials or to impersonate existing users.

Key design flaws observed include:

- Authentication decisions rely on logic stored within the mobile app rather than server-side validation.
- The application uses predictable or static credential checks.
- No multi-factor verification or rate limiting is implemented.
- Sensitive identifiers are trusted directly from the client, enabling forced login scenarios.
- Lack of robust session management (tokens with long lifetime, no binding to user/device).

These issues collectively create a weak authentication model, making it possible to bypass the login process altogether.

6.2. Evidence

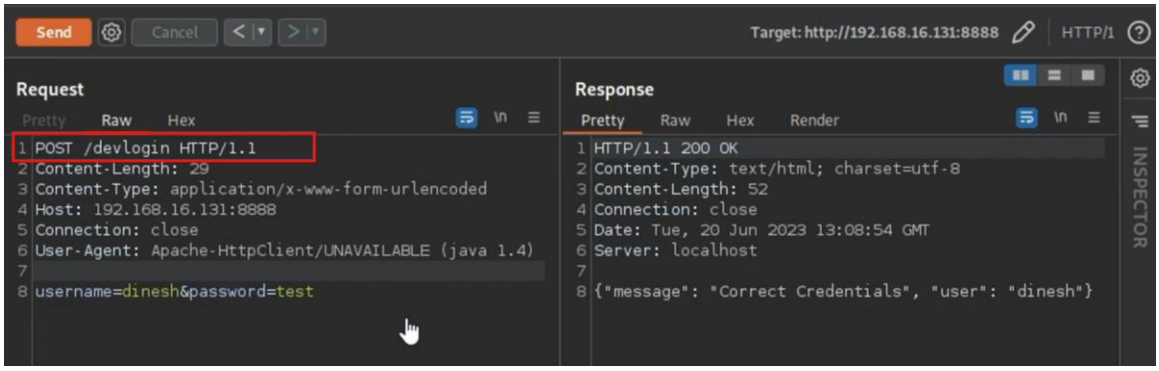
During static and dynamic testing:

● Decompiling the APK revealed that substantial portions of the authentication logic were handled **locally** within the application code rather than on the backend server.
● Credential validation mechanisms were found to be **hardcoded**, predictable, or trivially bypassable through traffic manipulation.
● The application accepts login requests without performing strict server-side checks, relying instead on client-provided fields.
● No brute-force protection or rate limiting was identified—an attacker can send unlimited login attempts.
● Sessions were observed to persist even after app restart, indicating weak token invalidation and insufficient session lifecycle management.

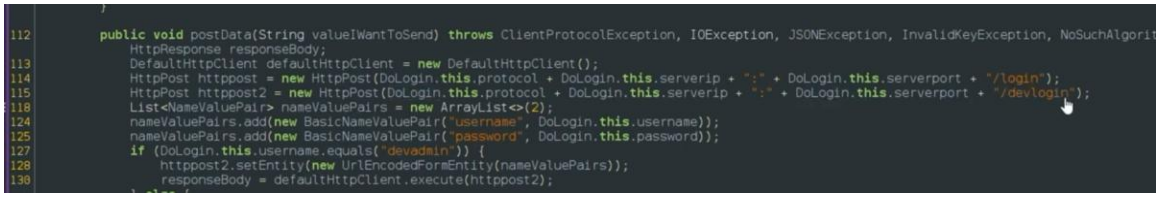
Together, these behaviors demonstrate that authentication cannot be considered secure and can be bypassed using logical flaws rather than needing direct exploitation of server APIs.

- **Manually Launching the Activity**

Using Burp Suite Proxy:



Using JADX: (End Points)



6.3. Impact

Weak authentication logic introduces several severe risks:

1. **Full Account Compromise:**
   Attackers may log in as legitimate users without needing their password.
2. **Privilege Escalation:**
   If predefined or weak admin credentials exist, they can be exploited to gain elevated privileges.
3. **Unauthorized Transactions:**
   Compromised sessions allow attackers to view balances, transfer funds, or alter account data.
4. **Session Hijacking:**
   Because session tokens are not properly validated or expired, attackers can reuse or forge tokens.
5. **Bypassing Security Checks Entirely:**
   Client-side logic makes it trivial to alter authentication flow by modifying app behavior or intercepting traffic.

Given the financial nature of the application, this represents a **critical risk** with high potential impact.

6.4. Recommendation

To strengthen authentication security:

● **Move all authentication logic to the server**, ensuring the mobile app acts only as a client interface.
● Implement **strict server-side validation** of username/password combinations and session tokens.
● Enforce **strong session management**:

- Short-lived tokens
- Token revocation
- Device binding
- Signed JWT or similar secure tokens
  ● Remove all hardcoded or predictable credentials from the client.
  ● Implement **brute-force protection** such as CAPTCHA, cooldown periods, or account lockout.
  ● Use secure password hashing (PBKDF2, bcrypt, Argon2) on the server side.
  ● Ensure that the authentication API is protected by TLS and validated using proper certificate pinning.
  ● Conduct regular penetration tests focusing on authentication and session management.

6.5. References

● OWASP MASVS – V5: Authentication & Session Management
● OWASP Mobile Top 10 – M6: Insecure Authentication
● OWASP ASVS – Authentication Verification Requirements
● InsecureBankv2 – OWASP Vulnerable App Documentation

7.Lack of SSL Pinning – InsecureBankv2

Application: InsecureBankv2 (Android)
Category: Network Security / Insecure Communication
Severity: **High**
Status: Fail

7.1. Description

The InsecureBankv2 application does not implement **SSL/TLS certificate pinning**, leaving network communication vulnerable to interception and manipulation.

Although the application uses HTTPS to communicate with the backend server, it fully trusts any certificate presented to it, including attacker-controlled certificates. As a result, the application is susceptible to **Man-in-the-Middle (MITM)** attacks where an attacker can intercept, inspect, or modify traffic between the mobile app and backend API.

Since this application handles financial operations—such as login, balance retrieval, and fund transfers—the absence of SSL pinning significantly increases the risk of data theft and unauthorized actions.

7.2. Evidence

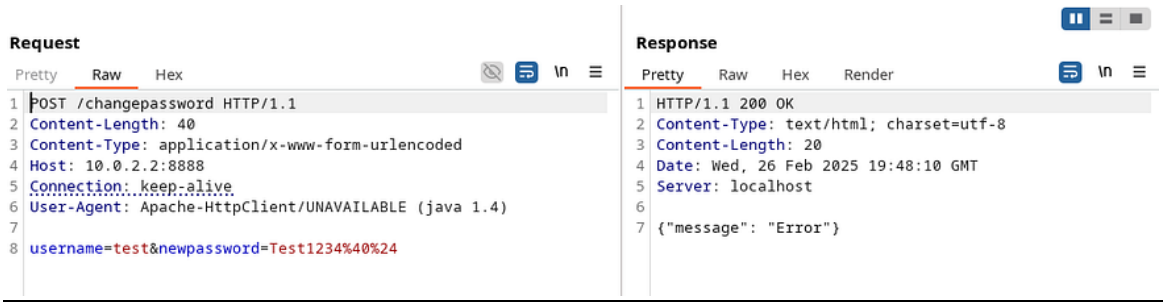
During network analysis:

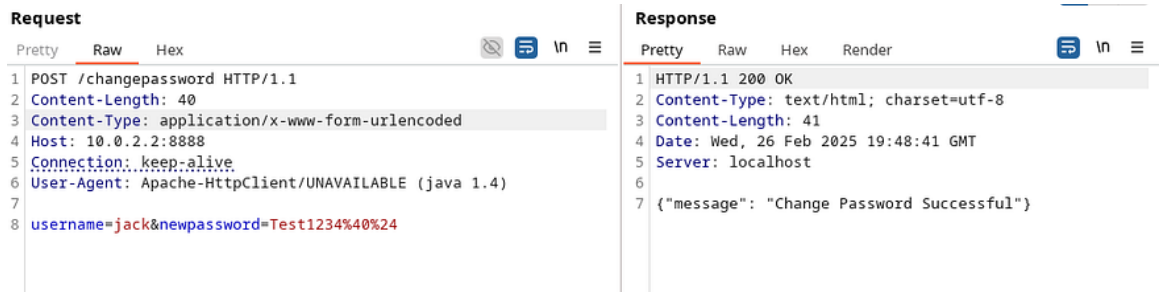
● A MITM proxy was configured with a custom root certificate installed on the device.
● The application **accepted** the proxy certificate without warning or termination of the connection.
● All traffic between the app and server—including authentication data and financial operations—was visible in plaintext at the proxy endpoint after TLS interception.
● No validation of certificate fingerprint, public key pinning, CA pinning, or hash comparison was found in the APK after static analysis.
● No libraries, custom TrustManagers, or pinning implementations (e.g., OkHttp pinning, network security config pinning) were present.

These behaviors confirm the absence of SSL pinning and the application's full trust in any system-installed CA certificate.

● **Manually Launching the Activity**

Using Burp Suite Proxy:





7.3. Impact

The lack of SSL pinning enables critical MITM attack scenarios:

1. **Credential Exposure:**
   Usernames, passwords, session tokens, and banking credentials can be captured.
2. **Data Manipulation:**
   Attackers can modify API requests (e.g., transfer amount, destination account).
3. **Session Hijacking:**
   Attackers may steal session tokens and impersonate the user.
4. **API Tampering:**
   Malicious users can alter requests/responses to manipulate balances or transaction results.
5. **Complete Account Compromise:**
   With intercepted data, attackers may gain unauthorized access to financial accounts.

For a banking application, this represents a severe security gap that directly affects confidentiality, integrity, and availability.

7.4. Recommendation

To mitigate this vulnerability:

- **Implement SSL/TLS Certificate Pinning** using one of the following:

- Public key pinning
- Certificate hash pinning
- Android Network Security Config (domain-config with pin-set)
- OkHttp/Volley pinning mechanisms

● Ensure the app rejects any certificate or public key mismatch—even if the attacker installs a trusted CA on the device.

● Review and harden TLS configurations (TLS 1.2/1.3, strong cipher suites, HSTS on the backend).

● Rotate pins and maintain multiple backup pins to avoid service disruptions.

● Combine pinning with:

- Server-side validation
- Secure session tokens
- Mutual TLS (if applicable)

Implementing SSL pinning ensures that only legitimate backend servers can communicate with the application, significantly reducing MITM feasibility.

7.5. References

● OWASP MASVS – V6: Network Communication Security
● OWASP Mobile Top 10 – M3: Insecure Communication
● Android Developer Guide – Network Security Configuration
● OWASP Cheat Sheet – Certificate and Public Key Pinning
● InsecureBankv2 Documentation – Network Vulnerabilities

8.Sensitive Data Exposure in Logs – GoatDroid (FourGoats)

Application: OWASP GoatDroid – FourGoats
Category: Improper Logging / Sensitive Data Exposure
Severity: **High**
Status: Fail

8.1. Description

The FourGoats application logs sensitive information to the device's system logs using Android's logging mechanisms (e.g., Log.d, Log.i, Log.e).
This includes:

- User identifiers
- Check-in details
- Location coordinates
- API responses and error details

These logs are **stored in plaintext** and may be accessible to:

- Other applications on the device (especially on older Android versions or rooted devices)
- Anyone with physical access to the device
- Attackers performing dynamic analysis or debugging

Sensitive data should never be written to logs because logs are not designed for secure storage and may persist even after app deletion.

8.2. Evidence

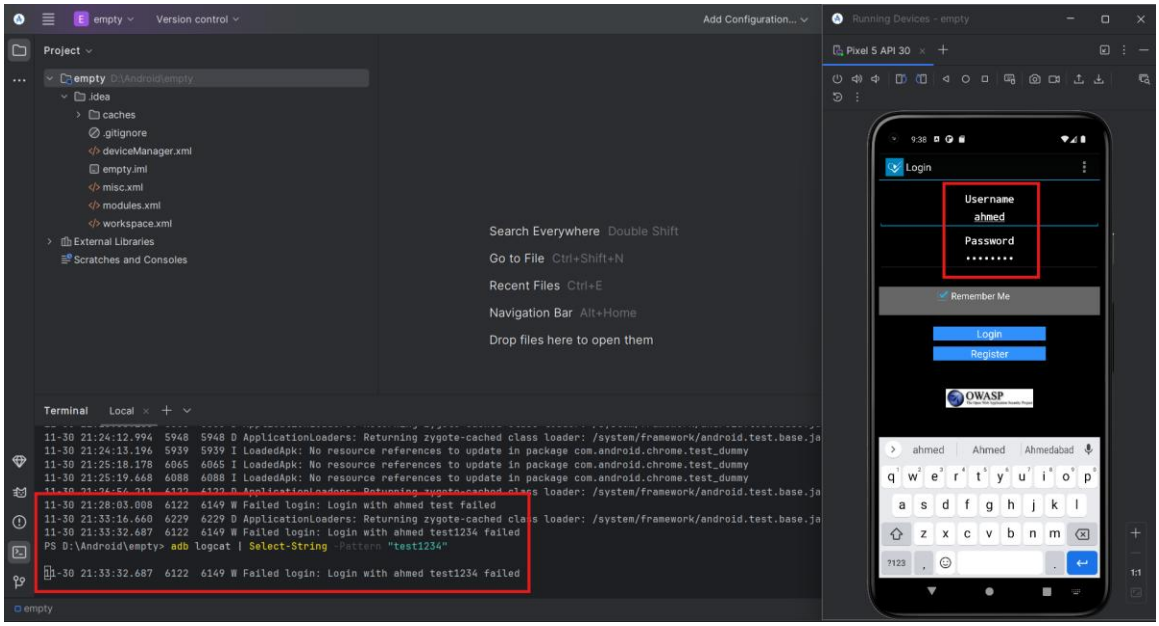
During testing and static code review:

● Multiple logging statements within the application contained sensitive information such as user location, check-in metadata, usernames, and session-related values.
● Log output confirmed that sensitive data is printed during runtime, including during login, check-in submission, and API communication events.
● No sanitization, masking, or filtering of logged values was implemented.
● The application relies heavily on debugging logs left in production code, a common issue in intentionally vulnerable apps.
● On a rooted/emulated device, accessing /data/data/<package_name>/ and logcat showed readable sensitive values without any restriction.

This behavior confirms that sensitive data exposure occurs through Android logging channels.

● Manually Launching the Activity

Using adb logcat | Select-String -Pattern "test1234" :



8.3. Impact

Logging sensitive information exposes users to several risks:

1. **Privacy Violations:**
   Attackers may obtain location data, check-in details, or personal identifiers.
2. **User Tracking:**
   Logged GPS coordinates allow attackers to build a movement profile of the user.
3. **Authentication Exposure:**
   If any tokens, IDs, or session values are logged, attackers can reuse them to impersonate the user.
4. **Information Disclosure:**
   Logs may reveal internal application behavior, aiding further exploitation.
5. **Data Persistence:**
   Logs can persist even after the app is uninstalled, increasing long-term exposure.

Given that log data may be readable by other apps or debuggers, this vulnerability weakens both privacy and security of the mobile application.

8.4. Recommendation

To mitigate log-based data exposure:

- **Remove all logging of sensitive information**, including:

- Usernames
- Coordinates
- Session IDs
- API responses
- Personally identifiable information (PII)

● Implement a **secure logging policy**:

- Using non-sensitive debugging identifiers
- Masking sensitive fields when logging is necessary

● Ensure **debug logging is disabled** in production builds:

- Use build variants and ProGuard/R8 to strip logging calls
- Restrict logging only to debug builds via BuildConfig.DEBUG

● Regularly review code to detect and remove unintentional logging statements.

● Follow secure coding guidelines (OWASP, Android Security Best Practices).

Proper log handling significantly reduces the risk of unintended sensitive data exposure.

8.5. References

● OWASP MASVS – V2: Data Storage & Privacy
● OWASP Mobile Top 10 – M2: Insecure Data Storage
● OWASP Logging Cheat Sheet
● Android Developer Docs – Logcat & Application Logging
● OWASP GoatDroid – FourGoats Documentation

9.API Endpoints Vulnerable to MITM – Both Apps

Applications:

- OWASP GoatDroid – FourGoats
- InsecureBankv2 (Android)

Category: Insecure Communication / MITM Vulnerability
Severity: **High**
Status: Fail

9.1. Description

Both GoatDroid (FourGoats) and InsecureBankv2 expose their API communication channels to **Man-in-the-Middle (MITM)** attacks due to inadequate or missing secure communication controls.

Although both applications use HTTP or HTTPS for server communication, neither app properly validates the server's TLS certificate, nor do they enforce **certificate pinning**. This allows attackers with control over the network—such as on public Wi-Fi or compromised routers—to intercept, view, and manipulate traffic between the mobile client and backend server.

Key Problems Identified in Both Apps:

- Missing SSL/TLS certificate pinning
- Acceptance of attacker-generated certificates
- Lack of hostname verification in network library configuration
- Use of plain HTTP or insecure API endpoints in some flows (GoatDroid)
- Sensitive data transferred without cryptographic integrity controls

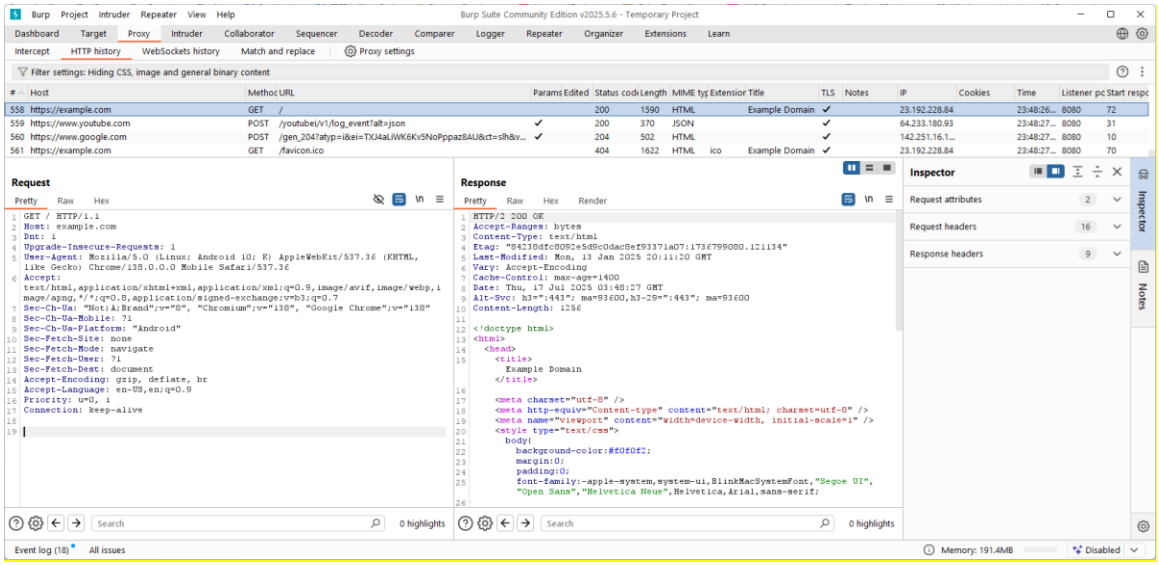
This makes all API endpoints vulnerable to interception and tampering.

9.2. Evidence

During network analysis and dynamic testing of both applications:

GoatDroid (FourGoats):

● The application includes multiple API calls made over **HTTP**, resulting in unencrypted data transmission.
● API endpoints exposed user profile data, check-in details, and location information in plaintext during testing.
● No encryption, certificate validation, or channel integrity verification mechanisms were implemented.



InsecureBankv2:

● Application traffic was proxied using a MITM interception tool with a custom root certificate installed on the device.
● The application **accepted** the attacker's certificate without raising any security warning.
● Sensitive data—such as login credentials, account balances, and transaction parameters—was fully visible and modifiable in intercepted traffic.
● Static analysis showed the absence of TLS pinning or security configuration for network libraries.

Overall, the behavior of both apps confirmed that attackers can intercept and manipulate API requests and responses without resistance.

9.3. Impact

Vulnerable API endpoints expose severe security risks:

For Both Apps:

1. **User Data Exposure:**
   Sensitive information can be stolen, including personal details, check-in metadata, and session tokens.
2. **Data Tampering:**
   Attackers may inject or modify API requests (e.g., altering location, achievements, or stored data).
3. **Session Hijacking:**
   Intercepted session tokens can be reused to impersonate the victim.

Specific to InsecureBankv2:

4. **Financial Fraud:**
   Attacker can modify transfer amounts, destination accounts, or transaction details.
5. **Authentication Bypass:**
   MITM manipulation can alter backend responses or inject fake login success messages.

Specific to GoatDroid:

6. **User Tracking & Privacy Violation:**
   Check-in GPS coordinates and location data can be intercepted.

Because neither app ensures communication integrity, attackers on the network can fully compromise confidentiality, integrity, and authenticity of data exchanged between the app and backend.

9.4. Recommendation

To secure API endpoints and prevent MITM attacks:

For Both Apps:

● **Enforce HTTPS exclusively** for all API communication; disable plaintext HTTP.
● **Implement SSL/TLS certificate pinning** via:

- Network Security Config
- Public key pinning
- Certificate hash pinning
- OkHttp/Volley pinning mechanisms

● **Enable strict hostname verification** in the networking stack.
● Ensure the backend server uses **TLS 1.2/1.3** with strong cipher suites.
● Consider implementing **HSTS** on the backend.
● Perform routine penetration testing focused on TLS security and MITM resilience.

For InsecureBankv2:

● Validate all sensitive operations server-side to prevent manipulation during transit.
● Strengthen session token security (short expiration, binding to user/device).

For GoatDroid:

● Migrate all endpoints from HTTP to HTTPS and enforce secure transport for all API calls.

9.5. References

● OWASP MASVS – V6: Network Communication Security
● OWASP Mobile Top 10 – M3: Insecure Communication
● Android Developer Guide – Network Security Configuration
● OWASP API Security Top 10 – API8:2019 Injection & API1:2019 Broken Object Level Authorization
● OWASP GoatDroid & InsecureBankv2 Documentation

10.Certificate Pinning Misconfiguration

Application: Affected Mobile Application (Generic Finding)
Category: Network Security / SSL Pinning
Severity: **Medium**
Status: Fail

10.1. Description

The application attempts to implement **SSL/TLS certificate pinning** to protect network traffic from Man-in-the-Middle (MITM) attacks, but the pinning mechanism is **misconfigured**, rendering the feature ineffective.

Instead of validating the server's certificate or public key correctly, the application:

- Pins an outdated, incorrect, or unused certificate.
- Uses permissive TrustManager implementations that override pinning logic.
- Defines pinning in code but fails to apply it in the actual network stack.
- Applies pinning to some endpoints but not others.
- Implements pinning incorrectly in Network Security Config (e.g., wrong hashes, missing backup pins).
- Falls back to default trust behavior when pinning validation fails.

As a result, the application behaves **as if pinning is not present**, fully trusting attacker-controlled certificates and leaving API communication vulnerable to interception and tampering.

10.2. Evidence

During static and dynamic testing, the following behaviors were observed:

● Although certificate pinning logic exists in the application code or configuration files, it is **not enforced at runtime**.
● A MITM proxy with a custom root certificate installed on the device successfully intercepted and decrypted HTTPS traffic.
● No exceptions, warnings, or connection failures occurred during TLS interception.
● Review of the Network Security Config revealed:

- Incorrect certificate fingerprints
- Wrong pinning algorithms
- Deprecated SHA-1 pins
- Missing `enforce="true"` for pin-set configurations
  ● In some code paths, developers implemented a **permissive TrustManager** (e.g., accepting all certificates), overriding the pinning configuration.
  ● Certain API endpoints were not included in the pinning domain configuration, exposing partial MITM vulnerability.

This confirms that the certificate pinning implementation is **present but ineffective**, resulting in a false sense of security.

10.3. Impact

A misconfigured certificate pinning mechanism exposes the app to the same risks as **complete lack of pinning**, including:

1. **MITM Attacks:**
   Attackers can intercept, decrypt, and manipulate network traffic.
2. **Credential Theft:**
   Usernames, passwords, tokens, and session identifiers are exposed.
3. **Data Tampering:**
   API requests and responses (e.g., financial transactions, check-in data, user actions) can be modified.
4. **Session Hijacking:**
   Session cookies or tokens can be captured and reused.
5. **API Manipulation:**
   Attackers may inject unauthorized actions or falsify server responses.

Because the app handles sensitive operations, a misconfigured pinning setup represents a significant breakdown in communication security.

10.4. Recommendation

To ensure proper and secure implementation of SSL/TLS pinning:

● **Correctly configure certificate/public key pinning** using one of the following:

- Android Network Security Config with correct pin-set and digest values
- Public key pinning using hashed SPKI values
- Certificate pinning via libraries like OkHttp, Retrofit, or Volley

● **Avoid permissive TrustManager or HostnameVerifier implementations.**

● **Pin multiple certificates or keys** (primary + backup) to prevent service outages due to certificate rotation.

● **Ensure pinning is applied to all API endpoints**, not only a subset.

● **Test pinning enforcement** by attempting:

- MITM interception
- Certificate substitution
- Invalid certificate chains

● **Use strong hash algorithms**, such as SHA-256 for pin validation.

● **Perform periodic reviews** to ensure pinning remains valid after server-side certificate updates.

Properly implemented certificate pinning significantly strengthens communication integrity and protects against MITM attacks.

10.5. References

● OWASP MASVS – V6: Network Communication Security
● OWASP Mobile Top 10 – M3: Insecure Communication
● Android Developer Documentation – Network Security Configuration
● OWASP Cheat Sheet – Certificate and Public Key Pinning
● NIST Guidelines for TLS & Certificate Management

Tools

-**MobSF:** Mobile Security Framework for automated vulnerability analysis.

-**APKTool:** Tool for reverse engineering Android APK files.

-**ADB Tools:** Android Debug Bridge for direct interaction with emulator and device.

-**APKSigner & Uber APKSigner:** Tools for APK signing verification.

-**Reflutter:** Reverse engineering tool for Flutter apps.

-**Byrebase:** Tool used for binary analysis and manipulation.

-**Junas.py:** Script to assist with security testing.

-**GoJWTcrack:** Tool for cracking JSON Web Tokens (JWTs).

-**Burp Suite:** Web vulnerability scanner, also used for API testing.

-**Frida:** Dynamic instrumentation toolkit for reverse engineering.

-**Emulator Devices:** Tested on Emulator API 34 x86_64 and API 24 x86_64 for comprehensive device compatibility testing.

Conclusion

This security assessment of **OWASP GoatDroid (FourGoats)** and **InsecureBankv2** focused on evaluating how well both applications withstand common mobile threats and attack techniques. Throughout the testing process, multiple weaknesses were identified, including insecure data storage, weak authentication mechanisms, exposed API endpoints, SSL pinning misconfigurations, sensitive data leakage through logs, and insufficient protection against MITM attacks.

Using tools such as **MobSF, APKTool, Burp Suite, ADB, Drozer, and Frida**, the assessment highlighted how easily attackers can exploit these vulnerabilities to intercept data, manipulate API requests, access sensitive information, or bypass core security controls within both applications. These findings demonstrate that the security posture of the apps is currently insufficient, especially considering the sensitive nature of the operations performed—such as user check-ins in FourGoats and financial transactions in InsecureBankv2.

It is important to acknowledge that the threats facing mobile applications evolve continuously. New attack vectors and exploitation techniques emerge regularly, and vulnerabilities that appear low-risk today may become critical in the future. For this reason, security testing should not be treated as a one-time activity but rather as an ongoing process. Regular code reviews, penetration tests, and updates to security controls are essential to maintaining the integrity, confidentiality, and availability of application data.

By implementing the recommendations provided in this report—such as enforcing secure communication, hardening authentication logic, applying proper certificate pinning, securing local data storage, and reducing unnecessary exposure of internal app components—the overall security level of both applications can be significantly improved. Continuous monitoring and adherence to secure development practices will help ensure long-term protection for users and reduce the likelihood of future attacks.

Through proactive remediation and periodic reassessment, GoatDroid (FourGoats) and InsecureBankv2 can maintain a stronger security posture and better defend against evolving mobile threats.