Mobile Application Security Testing

Kristiina Rahkema
Terminology

- Risk level = Event potentiality + Impact level
- Event potentiality = Threat likelihood + Vulnerability level
- Threat likelihood = Threat agent + Attack method
- Not every vulnerability can be exploited
Terminology

• Risk level = Event potentiality + Impact level
• Event potentiality = Threat likelihood + Vulnerability level
• Threat likelihood = Threat agent + Attack method

• Not every vulnerability can be exploited

• Example 1:
  • Vulnerability: buffer overflow in function X (in third party library)
  • Function is never called, so no threat, therefore no risk
**Terminology**

- **Risk level** = Event potentiality + Impact level
- **Event potentiality** = Threat likelihood + Vulnerability level
- **Threat likelihood** = Threat agent + Attack method
- **Not every vulnerability can be exploited**

**Example 2:**
- **Vulnerability:** Logging locations locally
- **Not severe on its own**
- **In combination with other vulnerabilities:** high risk

Simplified extract of ISSRM domain model
OWASP (Open Web Application Security Project)

- nonprofit foundation that works to improve the security of software

- OWASP Top 10:
  - Mobile: report on mobile application security (2014, 2016)
  - focusing on the 10 most critical risks

- Links:
  - https://owasp.org/www-project-top-ten/
  - https://owasp.org/www-project-mobile-top-10/
OWASP Web Top 10

A01: Broken access control
A02: Cryptographic failures
  (Protection of data in transit and at rest)
A03: Injection
  (e.g. SQL injection and cross site scripting)
A04: Insecure design
  (Missing or ineffective security controls, cannot be fixed with proper implementation or configuration)
A05: Security misconfigurations
A06: Vulnerable and outdated components
A07: Identification and authentication failures
  (Including broken authentication)
A08: Software and data integrity failures
  (Including software updates, insecure CI/CD pipelines etc.)
A09: Security logging and monitoring failures
A10: Server-side request forgery

OWASP Mobile Top 10

M1: Improper platform usage
  (misuse of a platform feature or failure to use platform security controls)
M2: Insecure data storage
  (covers insecure data storage and unintended data leakage)
M3: Insecure communication
M4: Insecure authentication
  (problems authenticating the end user or bad session management)
M5: Insufficient cryptography
M6: Insecure authorization
M7: Client code quality
  (catch-all for code-level implementation problems in the mobile client)
M8: Code tampering
M9: Reverse engineering
M10: Extraneous functionality
  (hidden backdoor functionality or other internal development security controls not intended for production environment)
Mobile app security testing

- With source code access
  - Checking if best practices are followed e.g.
    - How is data stored?
    - How are web requests made?
    - Does the server trust data from app?
    - Are secure coding principles followed?

- Without source code access
  - Rooting
    - Android: Rooting, iOS: Jailbreak
    - Access to all files, can install tools
  - Monitoring and intercepting web requests
  - Reverse engineering and code tampering (method hooking, swizzling)
Mobile app security testing

• OWASP MSTG
  • Mobile Security Testing Guide
  • https://mobile-security.gitbook.io/mobile-security-testing-guide/

• OWASP MASVS
  • Mobile AppSec Verification Standard
  • https://mobile-security.gitbook.io/masvs/

• Intentionally vulnerable applications to learn about security testing:
  • DIVA (iOS and Android)
  • Android InsecureBank v2 (Android)
  • hpAndro Android AppSec (Android)
  • MSTG Hacking Playground (iOS and Android)
  • InjuredAndroid (Android)
  • AndroGoat (Android)
  • OWASP Crackmes (iOS and Android)
  • Sieve app (Android)
  • PIIVA (Android)
  • DVHMA (hybrid)
  • Damn Vulnerable Bank (Android)
OWASP Mobile AppSec Verification requirements (MASVS)

V1: Architecture, Design and Threat Modeling Requirements (M10)

V2: Data Storage and Privacy Requirements (M1, M2)

V3: Cryptography Requirements (M5)

V4: Authentication and Session Management Requirements (M4, M6)

V5: Network Communication Requirements (M3)

V6: Platform Interaction Requirements (M1, M7)

V7: Code Quality and Build Setting Requirements (M7)

V8: Resilience Requirements (M8, M9)

OWASP Mobile Top 10

M1: Improper platform usage

M2: Insecure data storage

M3: Insecure communication

M4: Insecure authentication

M5: Insufficient cryptography

M6: Insecure authorization

M7: Client code quality

M8: Code tampering

M9: Reverse engineering

M10: Extraneous functionality
OWASP Mobile App Security Checklists

https://github.com/OWASP/owasp-mstg/releases/tag/v1.4.0
## Data Storage and Privacy Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>MSTG-ID</th>
<th>Detailed Verification Requirement</th>
<th>L1</th>
<th>L2</th>
<th>R</th>
<th>Android</th>
<th>iOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>MSTG-STORAGE-1</td>
<td>System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>MSTG-STORAGE-2</td>
<td>No sensitive data should be stored outside of the app container or system credential storage facilities.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>MSTG-STORAGE-3</td>
<td>No sensitive data is written to application logs.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>MSTG-STORAGE-4</td>
<td>No sensitive data is shared with third parties unless it is a necessary part of the architecture.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>MSTG-STORAGE-5</td>
<td>The keyboard cache is disabled on text inputs that process sensitive data.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>MSTG-STORAGE-6</td>
<td>No sensitive data is exposed via IPC mechanisms.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>MSTG-STORAGE-7</td>
<td>No sensitive data, such as passwords or pins, is exposed through the user interface.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.8</td>
<td>MSTG-STORAGE-8</td>
<td>No sensitive data is included in backups generated by the mobile operating system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>MSTG-ID</td>
<td>Detailed Verification Requirement</td>
<td>L1</td>
<td>L2</td>
<td>R</td>
<td>Android</td>
<td>iOS</td>
</tr>
<tr>
<td>-----</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----</td>
<td>----</td>
<td>---</td>
<td>---------</td>
<td>-----</td>
</tr>
<tr>
<td>8.1</td>
<td>MSTG-RESILIENCE-1</td>
<td>The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>MSTG-RESILIENCE-2</td>
<td>The app detects, and responds to, tampering with executable files and critical data within its own sandbox.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>MSTG-RESILIENCE-3</td>
<td>The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>MSTG-RESILIENCE-4</td>
<td>The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.5</td>
<td>MSTG-RESILIENCE-5</td>
<td>The app detects, and responds to, being run in an emulator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.6</td>
<td>MSTG-RESILIENCE-6</td>
<td>The app detects, and responds to, tampering the code and data in its own memory space.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.7</td>
<td>MSTG-RESILIENCE-7</td>
<td>The app implements multiple mechanisms in each defense category (8.1 to 8.6). Note that resiliency scales with the amount, diversity of the originality of the mechanisms used.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
M1: Improper platform usage

- **Explanation:** misuse of platform features or failure to use platform security controls
- **Might include:**
  - Android intents
  - Platform permissions
  - Misuse of TouchID, FaceID
  - Misuse of Keychain
  - Misuse of other security controls

- **How to prevent?** Secure coding and configuration practices, following platform best practices.
- **Related requirements:**
  - Platform interaction requirements
- **Related test cases**
  - iOS: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06h-Testing-Platform-Interaction.md
Improper platform usage, example: TikTok

- **Description:**
  - Malicious app could access any files
  - Malicious app could run arbitrary code
- **Problem:**
  - Input of an Android intent was not verified
- **Could have been prevented by:**
  - **MSTG-PLATFORM-2**: (All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.)
All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

1. Check for content providers
All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

### Testing instructions

1. **Check for content providers**

```xml
<provider
    android:name=".OMTG_CODING_003_SQL_Injection_Content_Provider_Implementation"
    android:authorities="sg.vp.owasp_mobile.provider.College">
</provider>
```

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05h-Testing-Platform-Interaction.md
Testing instructions

1. Check for content providers

**TickTock example**

```xml
<provider
    android:name="com.ss.android.ugc.aweme.livewallpaper.WallPaperDataProvider"
    android:exported="true"
>
    <service
        android:name="com.ss.android.ugc.aweme.livewallpaper.AmeLiveWallpaper"
        android:permission="android.permission.BOOT_COMPLETED"
    >
        <intent-filter>
            <action
                android:name="android.service.wallpaper.WallpaperService"/>
            <meta-data
                android:name="android.service.wallpaper"
                android:resource="@xml/k7y"/>
        </intent-filter>
    </service>
    <activity
        android:theme="@style/je_"
        android:name="com.ss.android.ugc.aweme.livewallpaper.ui.LocalLiveWallPaperPre..."
    >
    <activity
        android:theme="@style/jei"
        android:name="com.ss.android.ugc.aweme.livewallpaper.ui.LiveWallPaperPre..."
    >

MSTG-PLATFORM-2

All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

Testing instructions

2. For each public content provider check how input is handled.

```java
@override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(STUDENTS_TABLE_NAME);

    switch (uriMatcher.match(uri)) {
        case STUDENTS:
            qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
            break;

        case STUDENT_ID:
            // SQL Injection when providing an ID
            qb.appendWhere(_ID + " = " + uri.getPathSegments().get(1));
            Log.e("appendWhere", uri.getPathSegments().get(1).toString());
            break;

        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }

    if (sortOrder == null || sortOrder == "") {
        /**
         * By default sort on student names
         */
    }
```
MSTG-PLATFORM-2

All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

Testing instructions

2. For each public content provider check how input is handled.

All app functions that process data coming in through the UI should implement input validation:

- For user interface input, Android Saripaar v2 can be used.
- For input from IPC or URL schemes, a validation function should be created. For example, the following determines whether the string is alphanumeric:

```java
public boolean isAlphaNumeric(String s){
    String pattern= "^[a-zA-Z0-9]*$";
    return s.matches(pattern);
}
```

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05h-Testing-Platform-Interaction.md
20

Testing instructions

2. For each public content provider check how input is handled.

TickTock example

Found in file com/ss/android/ugc/aweme/livewallpaper/ui/LiveWallPaperPreviewActivity.java

```java
25     public void onCreate(android.os.Bundle bundle) {
26         com.bytedance.apm.agent.instrumentation.ActivityInstrumentation.onTrace("com.ss.android.ugc.aweme.livewallpaper.ui.LiveWallPaperPreviewActivity.onCreate()");
27         super.onCreate(bundle);
29         if (this.mLiveWallPaper == null) {
30             finish();
31             com.bytedance.apm.agent.instrumentation.ActivityInstrumentation.onTrace("com.ss.android.ugc.aweme.livewallpaper.ui.LiveWallPaperPreviewActivity.onCreate()");
```
All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.
2. For each public content provider check how input is handled.

```java
java.lang.String str2 = "";
int match = this.f4b600a.match(uri);
if (match == 16) {
    str2 = com.ss.android.ugc.aweme.livewallpaper.util.LiveWallPaperManager.getInstance().getCurrentWallPaperPath();
} else if (match == 32) {
    str2 = com.ss.android.ugc.aweme.livewallpaper.util.C20655b.getC14979g(str2);
}
try {
    return android.os.ParcelFileDescriptor.open(new java.io.File(str2), 268435456);
} catch (java.lang.Exception unused) {
    return null;
}
```
M2: Insecure data storage

• **Explanation:** covers insecure data storage and unintended data leakage

• **Might include:**
  - Wrong keychain accessibility option
  - Insufficient file data protection
  - Access to privacy resources when using this data incorrectly

• **How to prevent?** Understand information assets that are processed and how APIs handle them.

• **Related requirements:**
  - Data Storage and Privacy Requirements

• **Related test cases**
  - Android: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md
  - iOS: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06d-Testing-Data-Storage.md
Insecure data storage, examples: Tinder

• Articles:

• Description:
  • Feature: showing people logged in near you
  • Tinder 1. fix: showing distance only, could still triangulate

• Problem:
  • The exact location near you was sent to the device

• Could have been prevented by:
  • MSTG-ARCH-2: (Security controls are never enforced only on the client side, but on the respective remote endpoints)
Insecure data storage, examples: Tinder

- **Articles:**

- **Description:**
  - Feature: showing people logged in near you
  - Tinder 1. fix: showing distance only, could still triangulate

- **Problem:**
  - The exact location near you was sent to the device

- **Could have been prevented by:**
  - **MSTG-ARCH-2:** (Security controls are never enforced only on the client side, but on the respective remote endpoints)
Insecure data storage, examples: Tinder, Bumble

- Articles:

- Description:
  - Feature: showing people logged in near you
  - Tinder 1. fix: showing distance only, could still triangulate

- Problem:
  - The exact location near you was sent to the device

- Could have been prevented by:
  - MSTG-ARCH-2: (Security controls are never enforced only on the client side, but on the respective remote endpoints)
M3: Insecure communication

• **Explanation**: if network traffic is monitored then data transmitted over insecure communication can be observed

• ** Might include:**
  • Lack of certificate pinning
  • Incorrect SSL versions
  • Cleartext communication of sensitive assets
  • HTTP

• **How to prevent?** Follow general best practices for secure communication.

• **Related requirements:**
  • Network Communication Requirements

• **Related test cases**
  • General: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x04f-Testing-Network-Communication.md
  • Android: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05g-Testing-Network-Communication.md
  • iOS: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06g-Testing-Network-Communication.md
Insecure communication, example: Think Mutual Bank Mobile Banking App

- Article:

- Description:
  - An attacker on the same network could listen to communication and capture login secrets

- Problem:
  - SSL certificates were not correctly verified

- Could have been prevented by:
  - MSTG-NETWORK-3: (The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted)
The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.

Test Case  Test Case
The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.

Testing instructions

Check that certificates are pinned

Static Analysis

Verify that the server certificate is pinned. Pinning can be implemented on server:

1. Including server's certificate in the application bundle and performing mechanisms whenever the certificate on the server is updated.
2. Limiting certificate issuer to e.g. one entity and bundling the intermediate attack surface and have a valid certificate.
3. Owning and managing your own PKI. The application would contain the application every time you change the certificate on the server, due to certificate to be self-signed.
The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.

**Testing instructions**

Check that certificates are pinned

**Static Analysis**

The following third-party libraries include pinning functionality:

- **TrustKit**: here you can pin by setting the public key hashes in your Info.plist or provide the hashes in a dictionary. See their readme for more details.
- **AlamoFire**: here you can define a `ServerTrustPolicy` per domain for which you can define the pinning method.
- **AFNetworking**: here you can set an `AFSecurityPolicy` to configure your pinning.

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06g-Testing-Network-Communication.md
The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.

Testing instructions

Check that certificates are pinned

The following third-party libraries include pinning functionality:

- **TrustKit**: here you can pin by setting the public key hashes in your Info.plist or provide the hashes in a dictionary. See their readme for more details.
- **AlamoFire**: here you can define a `ServerTrustPolicy` per domain for which you can define the pinning method.
- **AFNetworking**: here you can set an `AFSecurityPolicy` to configure your pinning.

Owning and managing your own PKI: The application would contain a certificate every time you change the certificate on the server, due to certificate to be self-signed.
M4: Insecure authentication

• **Explanation:** Problems authenticating the end user or bad session management

• **Might include:**
  - Failing to identify the user at all when that should be required
  - Failure to maintain the user's identity when it is required
  - Weaknesses in session management

• **How to prevent?** Avoid insecure mobile application authentication design patterns

• **Related requirements:**
  - Authentication and Session Management Requirements

• **Related test cases**
M4: Insecure authentication

1. Weaker authentication requirements for mobile app (when porting from web app)
2. Authenticating user locally
3. Encryption of local client data: key is not connected to login data
4. Remember me functionality stored user's password
5. Remember me functionality is opt-out
6. Not using device specific authentication tokens
7. Using device identifiers to geo-location to authenticate user (can be spoofed)
8. Using 4-digit PIN numbers as passwords

• How to prevent? Avoid insecure mobile application authentication design patterns

• Related requirements:
  • Authentication and Session Management Requirements

• Related test cases
Insecure authentication, example: Grab Android App

• Article:
  • https://hackerone.com/reports/202425 (2017)

• Description:
  • Can bypass 2FA by brute forcing 4 digit code, gain access to account

• Problem:
  • Login attempts were not blocked after too many tries, allowing brute force attacks

• Could have been prevented by:
  • MSTG-AUTH-6: (The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.)
M5: Insufficient cryptography

- **Explanation:** Cryptography was applied but insufficient in some way

- **Might include:**
  - Use of insecure and/or outdated algorithms
  - Use of custom encryption protocols
  - Poor key management

- **How to prevent?** Follow NIST guidelines on recommended algorithms

- **Related requirements:**
  - Cryptography Requirements

- **Related test cases**
  - General: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x04g-Testing-Cryptography.md
Insufficient cryptography, example: Ola app

- Article:

- Description:
  - Key embedded in code, could decrypt passwords, intercept requests and then fake requests for money

- Problem:
  - Weak cryptographic key, embedded in code

- Could have been prevented by:
  - MSTG-CRYPTO-5 (The app doesn't re-use the same cryptographic key for multiple purposes)
M6: Insecure authorisation

• **Explanation**: Able to execute overprivileged functionality, distinct from authentication

• **Might include**:
  • Not verifying that incoming identifiers match authorized account
  • IDOR (Insecure Direct Object Reference) vulnerabilities
  • Hidden endpoints
  • User Role or Permission Transmissions

• **How to prevent?** Backend code should independently verify that any incoming identifiers associated with a request match

• **Related requirements**:
  • Authentication and Session Management Requirements

• **Related test cases**
Insecure authorisation, example: Misafe smart watches

• Article:

• Description:
  • Attackers could retrieve real-time GPS coordinates of children, call children, spy on audio, send audio messages, retrieve photo, name, age, other PII

• Problem:
  • Insecure Direct Object Reference (IDOR) attack

• Could have been prevented by:
  • MSTG-AUTH-12 (Authorization models should be defined and enforced at the remote endpoint.)
M7: Client code quality

• **Explanation:** catch-all for code-level implementation problems in the mobile client

• ** Might include:**
  • Buffer overflows
  • Format string vulnerabilities
  • Various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.

• **How to prevent?** Code reviews, use coding patterns, prioritise buffer overflows and memory leaks

• **Related requirements:**
  • Code Quality and Build Setting Requirements

• **Related test cases**
Client code quality, example: WhatsApp

• Articles:
  • Similar: https://www.businessinsider.in/tech/apps/news/whatsapp-patches-a-severe-vulnerability-that-could-have-exposed-your-private-messages/articleshow/85892167.cms (2021)

• Description:
  • Making a call and sending a specially crafted series of packets resulted let the adversary run arbitrary code, used to install spyware

• Problem:
  • The specially crafted series of packets resulted in a buffer overflow

• Could have been prevented by:
  • MSTG-CODE-8 (In unmanaged code, memory is allocated, freed and used securely)
M8: Code tampering

• **Explanation:** exploiting code modifications
• **Might include:**
  • Binary patching
  • Local resource modification
  • Method hooking and swizzling
  • Dynamic memory modification

• **How to prevent?** Detecting root and jailbreaks
• **Related requirements:**
  • Resilience Requirements
• **Related test cases**
  • Android: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md
  • iOS: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md
Code tampering, example: Pokemon Go

- **Article:**

- **Description:**
  - Fed wrong geolocation data and time to find rare Pokemon and make eggs hatch faster

- **Problem:**
  - Application was reverse engineered, lacked detection for tampered data

- **Could have been prevented by:**
  - **MSTG-RESILIENCE-3** (The app detects, and responds to, tampering with executable files and critical data within its own sandbox)
M9: Reverse engineering

- **Explanation:** Reveal information about backend servers, cryptographic constants and ciphers, and intellectual property
- ** Might include:**
  - Reverse engineering source code, libraries, algorithms and other assets
  - Exploit other vulnerabilities in the application
  - Used for most examples above

- **How to prevent?** Obfuscating code
- **Related requirements:**
  - Resilience Requirements
- **Related test cases**
M10: Extraneous functionality

• **Explanation:** functionality in app that should not be there

• **Might include:**
  • Hidden backdoor functionality
  • Other internal development security controls not intended for production

• **How to prevent?** manual secure code review

• **Related requirements:**
  • Architecture, Design and Threat Modeling Requirements

• **Related test cases:** N/A
Extraneous functionality, example: Wifi File Transfer

- Article:

- Description:
  - App opens port, any computer can connect and gain full device access

- Problem:
  - Open port with no authentication

- Could have been prevented by:
  - MSTG-ARCH-3 (A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture)
1. Identify and protect sensitive data on the mobile device
2. Handle password credentials securely on the device
3. Ensure sensitive data is protected in transit
4. Implement user authentication, authorization and session management correctly
5. Keep the backend APIs (services) and the platform (server) secure
6. Secure data integration with third party services and applications
7. Pay specific attention to the collection and storage of consent for the collection and use of the user’s data
8. Implement controls to prevent unauthorized access to paid-for resources (wallet, SMS, phone calls etc.)
9. Ensure secure distribution/provisioning of mobile applications
10. Carefully check any runtime interpretation of code for errors
References:

• List of vulnerable apps: https://thedarksourcsource.com/vulnerable-android-apps/
• OWASP Top 10: https://owasp.org/www-project-top-ten/
• OWASP Mobile Top 10: https://owasp.org/www-project-mobile-top-10/
• OWASP MSTG: https://mobile-security.gitbook.io/mobile-security-testing-guide/
• OWASP MASVS: https://mobile-security.gitbook.io/masvs/
• Android intent vulnerabilities: https://snyk.io/blog/exploring-android-intent-based-security-vulnerabilities-google-play/
Thank you for listening!