



Actividad 2.5

Informe de Buenas Prácticas.

Nombre: Diego Bastián Rojo Peralta

Asignatura: Programación Android

Docente: Oscar Monardez

Fecha: 23-10-2025

Herramienta Utilizada: MobSF Live



1. Introducción.

En el desarrollo de este informe a través del desarrollo de la aplicación Android representa una instancia clave para aplicar conocimientos técnicos en un entorno real, integrando funcionalidades multimedia, mapas, navegación entre actividades y gestión de permisos. Más allá de la funcionalidad, este proyecto se ha enfocado en incorporar buenas prácticas de seguridad, arquitectura y compatibilidad, alineadas con estándares internacionales como OWASP MASVS.

Cada decisión técnica fue justificada en función de su impacto en la seguridad, la mantenibilidad y la experiencia del usuario.

Este informe documenta las buenas prácticas implementadas, por ser la continuidad del informe anterior solucionado lo aparecido en el ya mencionado informe, aquí se mostrarán los fragmentos de código aplicados, las vulnerabilidades mitigadas y el razonamiento detrás de cada mejora. El objetivo es evidenciar un desarrollo consciente, seguro y alineado con los criterios académicos exigidos, demostrando la capacidad de integrar teoría y práctica en un proyecto funcional.



2. Seguridad en Configuración.

La configuración segura de una aplicación Android es el primer paso para reducir la superficie de ataque y proteger tanto los datos como el comportamiento interno de la app. En este proyecto se han aplicado medidas concretas que refuerzan la seguridad desde el manifiesto y el entorno de compilación.

2.1 – Eliminación de “android:debuggable” del AndroidManifest.xml

- Fragmentos del código Corregido:

1. AndroidManifest.xml

```
<!-- Declaración de hardware para justificar permisos -->
<uses-feature android:name="android.hardware.camera" android:required="false" />

<application
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.EvaluacionAndroid_Diego"
    android:label="@string/google_maps_key"
    android:allowBackup="false"
    android:dataExtractionRules="@xml/data_extraction_rules">

    <!-- Clave de API para Google Maps -->
    <meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="@string/google_maps_key" />
</application>
```

2. Build.gradle.kts

```
buildTypes {
    release {
        isMinifyEnabled = true
        isDebuggable = false
        proguardFiles(
            getDefaultProguardFile("proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
    }
    debug {
        isMinifyEnabled = false
        isDebuggable = true
    }
}
```

- Explicación: El mencionado atributo “android:debuggable” permitía que la app sea depurada desde herramientas externas. Si dicho atributo se



mantenía activo en la producción, un ataque podría llegar a conectar un depurador y acceder a clases internas en la aplicación, modificando el flujo de ejecución y/o extraer datos. Se tomó la decisión de eliminar del manifest y se puso para su control exclusivamente en el archivo de build.gradle.kts, asegurando que solo se mantenga activo en modo debug.

- Vulnerabilidad Mitigada: Así se evita la exposición de las clases internas y puntos de entrada en entornos productivos.
- Referencia: MASVS-ARCH — "La aplicación no debe estar depurable en producción."

2.2 – Desactivación del respaldo automático. (allowBackup)

- Fragmento del Código corregido.

1. AndroidManifest.xml

```
android:allowBackup="false"  
android:dataExtractionRules="@xml/data_extraction_rules">
```

- Explicación: Por defecto, Android permite que los datos de la aplicación sean respaldados en la nube del dispositivo. Esto incluye preferencias, archivos internos y configuraciones. Si un atacante accede a ese respaldo, podría extraer información sensible. Al desactivar, se evita este comportamiento. Además, se incluye para controlar la exportación de datos en Android 12+.
- Vulnerabilidad Mitigada: Previene la fuga de datos sensibles por respaldo no controlado.
- Referencia: MASVS-STORAGE — "Los datos sensibles no deben ser respaldados automáticamente."



3. Seguridad en Configuración.

La gestión adecuada de permisos en Android es fundamental para proteger recursos sensibles del dispositivo y garantizar que la aplicación solo acceda a lo estrictamente necesario.

3.1 – Solicitud de Permisos en Tiempo de ejecución.

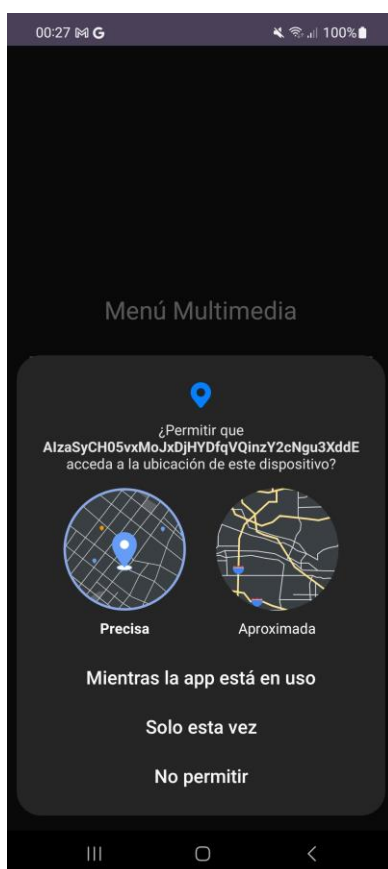
- Fragmento del Código:

1. MainActivity.java

```
btnIrMapas.setOnClickListener( View v -> {  
    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION)  
        != PackageManager.PERMISSION_GRANTED) {  
        ActivityCompat.requestPermissions( activity: this,  
            new String[]{Manifest.permission.ACCESS_FINE_LOCATION},  
                CODIGO_SOLICITUD_UBICACION);  
    } else {  
        abrirMapa();  
    }  
});
```

- Explicación: Android requiere que los permisos peligrosos (como ubicación, cámara o almacenamiento) sean solicitados explícitamente en tiempo de ejecución. Esta práctica evita que la aplicación acceda a recursos sensibles sin el consentimiento del usuario. En el proyecto, el permiso de ubicación se solicita justo antes de abrir la actividad de mapas, lo que mejora la transparencia y el control.
- Vulnerabilidad Mitigada: Evita el acceso no autorizado a la ubicación del usuario y reduce el riesgo de exposición de datos geográficos.
- Referencia: MASVS-P2 — "La aplicación debe solicitar permisos de forma explícita y controlada."

- Evidencia de funcionamiento:



3.2 – Manejo del resultado de la solicitud.

- Fragmento del código modificado.

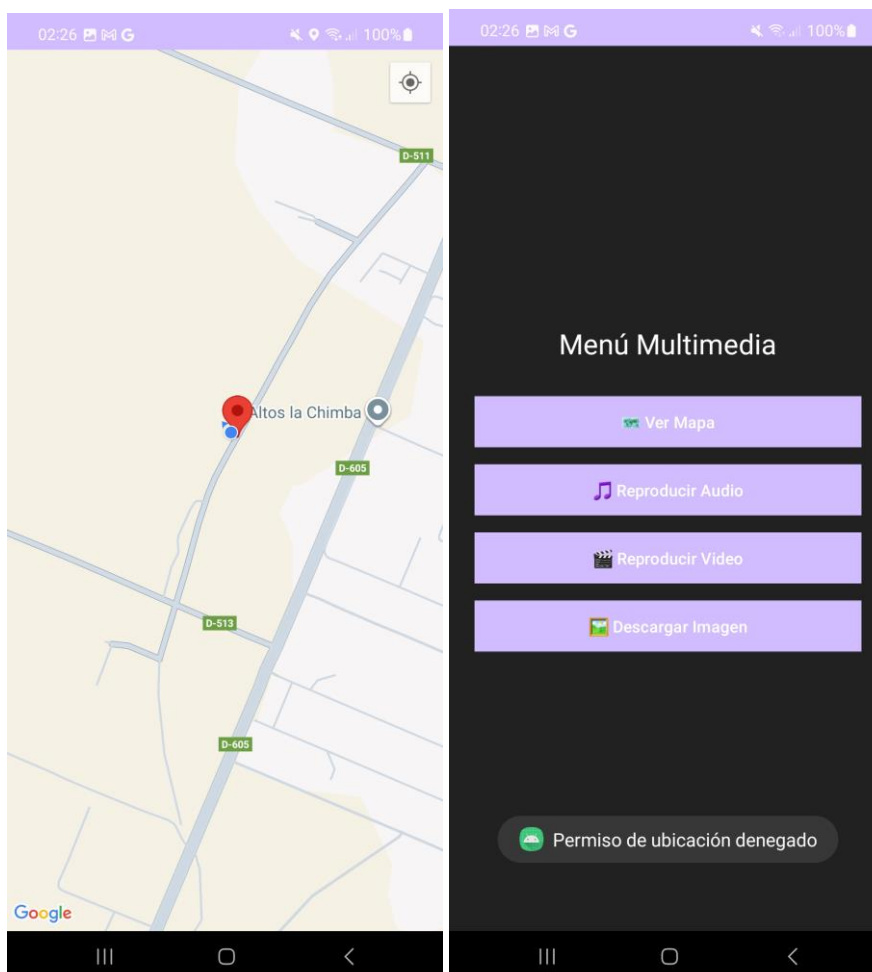
1. MainActivity.java

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == CODIGO_SOLICITUD_UBICACION) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            abrirMapa();
        } else {
            Toast.makeText(context, this, text: "Permiso de ubicación denegado", Toast.LENGTH_SHORT).show();
        }
    }
}
```

- Explicación: El resultado de la solicitud de permisos debe ser gestionado correctamente para evitar que la aplicación continúe con operaciones sensibles si el usuario ha denegado el acceso. En este caso, la actividad de

mapas solo se abre si el permiso fue concedido, y se muestra un mensaje claro si fue rechazado.

- Vulnerabilidad Mitigada: Evita el acceso no autorizado a la ubicación del usuario y reduce el riesgo de exposición de datos geográficos.
- Referencia: MASVS-P2 — "La aplicación debe solicitar permisos de forma explícita y controlada."
- Evidencia del Funcionamiento:



(Imagen 1 = Resultado de Aceptar el Permiso)

(Imagen 2 = Resultado de Negar el Permiso)



3.3 – Separación de permisos por versión de Android.

- Fragmento del código modificado:

1. AndroidManifest.xml

```
<!-- Compatibilidad con Android 13+ -->
<uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
<uses-permission android:name="android.permission.READ_MEDIA_VIDEO" />
<uses-permission android:name="android.permission.READ_MEDIA_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />

<!-- Compatibilidad con Android 10 o inferior -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
    android:maxSdkVersion="29" />
```

- Explicación: En Android 13 introdujo permisos más específicos para acceder a medios. Usar en versiones modernas genera advertencias y puede bloquear el acceso. Esta separación asegura que la app solicite solo lo necesario según la versión del sistema operativo, mejorando la compatibilidad y reduciendo la superficie de ataque.
- Vulnerabilidad Mitigada: Evita solicitudes excesivas de permisos y mejora la compatibilidad con Android 13+.
- Referencia: MASVS-P2 — "La aplicación debe solicitar solo los permisos estrictamente necesarios."

4. Cifrado de Claves Sensibles.

El manejo seguro de claves API y otros datos sensibles es fundamental en el desarrollo de aplicaciones móviles. Almacenar estos valores en texto plano, por ejemplo, en o directamente en el código fuente, representa un riesgo crítico, ya que pueden ser fácilmente extraídos mediante ingeniería inversa. En este proyecto se implementó cifrado de la clave de API de Google Maps utilizando, una solución recomendada por Android para proteger datos en reposo.

4.1 – Almacenamiento cifrado con EncryptedSharedPreferences.

- Fragmento de código modificado:

1. MainActivity.java

```
private void guardarClaveGoogleMaps() {
    try {
        String masterKeyAlias = MasterKeys.getOrCreate(MasterKeys.AES256_GCM_SPEC);
        SharedPreferences prefs = EncryptedSharedPreferences.create(
            fileName: "secure_prefs",
            masterKeyAlias,
            context: this,
            EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
            EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
        );

        if (!prefs.contains("google_maps_key")) {
            prefs.edit().putString("google_maps_key", "AIzaSyCH05vxMoJxDjHYDfqVQinzY2cNgu3XddE").apply();
        }
    }
}
```

- Explicación: EncryptedSharedPreferences utiliza cifrado AES-256 para proteger tanto las claves como los valores almacenados. La clave maestra se gestiona mediante el Android Keystore, lo que impide que sea accedida directamente por el código. Este enfoque asegura que la clave de API esté cifrada en reposo y solo pueda ser accedida por la propia aplicación.
- Vulnerabilidad mitigada: Reduce el riesgo de extracción de claves por ingeniería inversa o análisis de archivos internos.
- Referencia: MASVS-RESILIENCE — "Los secretos deben almacenarse de forma cifrada y protegida contra extracción."

4.2 – Recuperación segura de la clave en tiempo de ejecución.

- Fragmento del código modificado:

1. MapaActivity.java

```
private String obtenerClaveGoogleMaps() {
    try {
        String masterKeyAlias = MasterKeys.getOrCreate(MasterKeys.AES256_GCM_SPEC);
        SharedPreferences prefs = EncryptedSharedPreferences.create(
            fileName: "secure_prefs",
            masterKeyAlias,
            context: this,
            EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
            EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
        );
        return prefs.getString(key: "google_maps_key", defValue: null);
    } catch (GeneralSecurityException | IOException e) {
        return null;
    }
}
```

- Explicación: La clave se recupera en tiempo de ejecución desde el contenedor cifrado. Si no está presente, se maneja el error con un mensaje al usuario. Este



enfoque evita que la clave esté disponible en memoria o en archivos accesibles durante la instalación.

- Vulnerabilidad mitigada: Evita exposición accidental de la clave en logs, archivos temporales o memoria persistente.
- Referencia: MASVS-STORAGE — "Los datos sensibles deben recuperarse de forma segura y controlada."

5. Arquitectura y Componentes.

Una arquitectura bien definida y el uso correcto de componentes en Android permiten mejorar la mantenibilidad, reducir errores y reforzar la seguridad interna de la aplicación. En este proyecto se aplicaron prácticas que favorecen la claridad estructural, el aislamiento de responsabilidades y la protección de componentes sensibles.

5.1 – Uso de viewBinding para acceso seguro de vistas.

- Fragmento de código modificado.

```
buildFeatures {  
    viewBinding = true  
}
```

- Explicación: viewBinding reemplaza el uso tradicional de findViewById, ofreciendo una forma segura y tipada de acceder a las vistas definidas en los archivos XML. Esto reduce el riesgo de errores en tiempo de ejecución por referencias incorrectas y mejora la mantenibilidad del código.
- Vulnerabilidad mitigada: Evita fallos por referencias inválidas a vistas, que pueden provocar bloqueos o comportamientos inesperados.
- Referencia: MASVS-CODE — "El código debe ser robusto, legible y seguro frente a errores comunes."

5.2 – Separación de Entornos debug y reléase.

- Fragmento del código modificado:

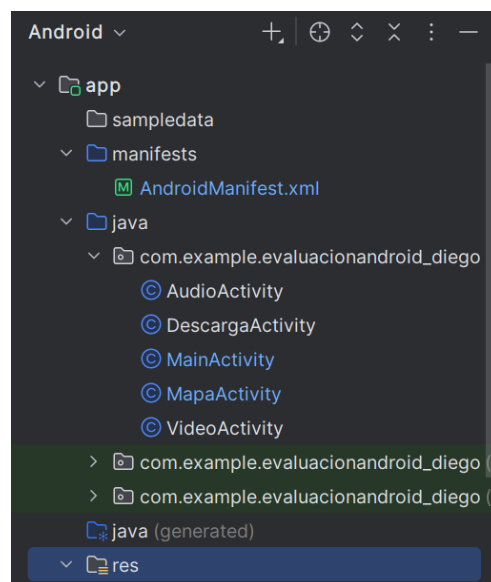
```
buildTypes {
    release {
        isMinifyEnabled = true
        isDebuggable = false
        proguardFiles(
            ...files = getDefaultProguardFile( name = "proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
    }

    debug {
        isMinifyEnabled = false
        isDebuggable = true
    }
}
```

- Explicación: La separación clara entre entornos de desarrollo (debug) y producción (release) permite aplicar configuraciones específicas para cada caso. En este proyecto, se desactiva la depuración y se activa la obfuscación en modo release, mientras que en debug se mantiene la visibilidad para facilitar pruebas.
- Vulnerabilidad mitigada: Evita que la app sea depurada en producción y dificulta la ingeniería inversa del código fuente.
- Referencia: MASVS-ARCH y MASVS-RESILIENCE — "La aplicación no debe estar depurable en producción" y "El código debe estar ofuscado para dificultar su análisis."

5.3 – Separación de responsabilidades por actividad.

- Fragmento estructural del código:





- Explicación: Cada funcionalidad está encapsulada en una actividad independiente, lo que permite aislar responsabilidades, facilitar pruebas unitarias y reducir el acoplamiento entre componentes. Esta estructura modular mejora la claridad del proyecto y permite aplicar medidas de seguridad específicas por actividad.
- Vulnerabilidad Mitigada: Reduce el riesgo de errores por mezcla de responsabilidades y facilita el control de acceso a cada componente.
- Referencia: MASVS-ARCH — "La arquitectura debe ser modular y facilitar el aislamiento de componentes."

6. Compatibilidad y Actualizaciones.

Mantener la aplicación alineada con las versiones más recientes del sistema operativo Android y sus librerías oficiales es esencial para garantizar seguridad, estabilidad y acceso a nuevas funcionalidades. En el proyecto se aplicaron medidas que aseguran compatibilidad con Android 13+ y previenen errores por uso de APIs obsoletas.

6.1 – Actualización de compileSdk y targetSdk

- Fragmento de código modificado:

```
android {  
    namespace = "com.example.evaluacionandroid_diego"  
    compileSdk = 36  
  
    defaultConfig {  
        applicationId = "com.example.evaluacionandroid_diego"  
        minSdk = 26  
        targetSdk = 36  
        versionCode = 1  
        versionName = "1.0"  
  
        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
    }  
}
```

- Explicación: Actualizar compileSdk y targetSdk permite que la aplicación compile contra las APIs más recientes y adopte los comportamientos modernos del sistema. En este caso, se utilizó la



versión 36 para asegurar compatibilidad con Android 14 y con librerías como androidx.activity:activity:1.11.0

- Vulnerabilidad mitigada: Evita incompatibilidades, errores de ejecución y exposición a vulnerabilidades corregidas en versiones recientes.
- Referencia: MASVS-ENV — "La aplicación debe estar compilada y orientada a las versiones más recientes del sistema operativo."

6.2 – Compatibilidad condicional con versión anteriores.

- Fragmento de código corregido:

```
<!-- Compatibilidad con Android 10 o inferior -->  
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"  
    android:maxSdkVersion="29" />
```

- Explicación: Para mantener compatibilidad con dispositivos que usan Android 10 o inferior, se incluye el permiso READ_EXTERNAL_STORAGE limitado mediante maxSdkVersion=29. Esto evita advertencias en versiones modernas y asegura que la app funcione correctamente en dispositivos antiguos sin solicitar permisos obsoletos en los nuevos.
- Vulnerabilidad mitigada: Evita errores de instalación y ejecución en versiones anteriores sin comprometer la seguridad en versiones modernas.
- Referencia: MASVS-P2 — "La aplicación debe ser compatible con versiones anteriores sin comprometer la seguridad."



7. Conclusión.

Después de muchas horas frente al computador, revisando documentación, corrigiendo errores, y tratando de entender cada detalle técnico, puedo decir que este proyecto me dejó agotado, pero también satisfecho. No fue solo programar una app funcional, sino aprender a hacerla segura, compatible y bien estructurada. Cada ajuste que parecía pequeño, como mover una línea de código o cambiar un atributo en el manifest, tenía implicancias reales en la seguridad y el comportamiento de la aplicación.

Implementar buenas prácticas no es solo seguir reglas, es entender por qué existen y cómo protegen al usuario y al desarrollador. Aprendí a cifrar claves sensibles, a controlar permisos de forma responsable, y a justificar cada decisión técnica con base en estándares como OWASP MASVS. Aunque el proceso fue exigente, me ayudó a pensar como un profesional, no solo como alguien que quiere que el código "funcione".

Este informe resume ese camino. No es perfecto, pero refleja el esfuerzo, el aprendizaje y el compromiso por entregar algo más que una aplicación: una solución consciente, segura y alineada con lo que se espera en el mundo real.



8. Bibliografía.

1. OWASP Foundation. (2023). *Mobile Application Security Verification Standard (MASVS)*. <https://owasp.org/www-project-mobile-security/>
2. Android Developers. (2023). *Security Best Practices*. Google. <https://developer.android.com/topic/security/best-practices>
3. Android Developers. (2023). *EncryptedSharedPreferences*. Google. <https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences>
4. Android Developers. (2023). *Permissions Overview*. Google. <https://developer.android.com/guide/topics/permissions/overview>
5. Android Developers. (2023). *Network Security Configuration*. Google. <https://developer.android.com/training/articles/security-config>
6. Android Developers. (2023). *App Manifest Overview*. Google. <https://developer.android.com/guide/topics/manifest/manifest-intro>