Documento de Análisis de Ataques informáticos en función del proyecto para Android “NutriFlash”

Por Joey Alcocer Hanson

Desarrollador Unico con supervisión docente para Nutri Flash

1. Inyección de SQL en la Base de Datos (Room)

El proyecto utiliza androidx.room:room-runtime, lo que significa que estás manejando una base de datos local. Room es una capa de abstracción sobre SQLite que ayuda a prevenir la inyección de SQL, pero no es infalible si no se usa correctamente.

Un posible ataque es la inyección de SQL ocurre cuando un atacante logra insertar y ejecutar comandos SQL maliciosos a través de las entradas de la aplicación. Si construyes una consulta de Room concatenando strings directamente desde la entrada del usuario (por ejemplo, db.query("SELECT * FROM users WHERE name = '" + userName + "'", null)), eres vulnerable. Un atacante podría manipular esa entrada para leer, modificar o borrar datos de la base de datos

Afortunadamente, Room te protege en gran medida. Siempre que uses los parámetros de consulta de Room con la anotación :variable en tus consultas DAO (Data Access Object), Room se encarga de escapar las entradas de forma segura.

Por eso es importante que nunca se construya consultas crudas (@RawQuery) concatenando strings de entrada del usuario. Utiliza siempre los argumentos proporcionados por Room.Kotlin//

@Query("SELECT * FROM profile WHERE name = :profileName")

fun findProfileByName(profileName: String): Profile

Ejemplo de mal uso:

@RawQuery

fun findProfileRaw(query: SupportSQLiteQuery): Profile

Y luego lo llamas así:

val query = SimpleSQLiteQuery("SELECT * FROM profile WHERE name = '" + userInput + "'")

dao.findProfileRaw(query)

2. Exposición de Datos Sensibles en el Dispositivo•

El Problema: Las aplicaciones a menudo guardan datos de configuración o de sesión. Si no se cifran, estos datos son vulnerables.

Si un atacante obtiene acceso físico al dispositivo y logra "rootearlo" (obtener privilegios de administrador), puede navegar libremente por el sistema de archivos. Si tu aplicación guarda datos en SharedPreferences o en archivos de texto plano, el atacante puede leerlos y modificarlos sin problemas.

Para cualquier dato sensible (preferencias de usuario, tokens, etc.), no uses SharedPreferences directamente. En su lugar, se recomienda por parte del equipo de desarrollador Nutri Flash utilizar alguna implementación de Encrypted SharedPreferences de la biblioteca de seguridad de Jetpack. Esta clase cifrar automáticamente las claves y los valores antes de guardarlos.

Si se guarda algo más que configuraciones no críticas (como el modo oscuro), se recomienda implementar Encrypted SharedPreferences.

3. Vulnerabilidades en Dependencias de Terceros

El proyecto depende de múltiples bibliotecas (Compose, Lifecycle, Core KTX, etc.). Con el tiempo, se pueden descubrir vulnerabilidades de seguridad en versiones específicas de estas bibliotecas.

Un atacante podría crear una aplicación maliciosa que explote una vulnerabilidad conocida en una de las bibliotecas que tu app utiliza. Por ejemplo, una falla en el manejo de Intents o en el renderizado de la UI podría ser explotada.

Es importante Mantener las dependencias actualizadas:

Es crucial que se usen las últimas versiones estables recomendadas por Google. Android Studio te notificará cuando haya actualizaciones disponibles. Se podría recomendar Activar Dependabot en un repositorio de GitHub. Es una herramienta gratuita que analiza tus dependencias y crea automáticamente "Pull Requests" para actualizar aquellas que tienen vulnerabilidades de seguridad conocidas.

4. Ingeniería Inversa y Manipulación del Código

Por defecto, el código de una aplicación Android (empaquetado en un APK) puede ser descompilado con relativa facilidad, permitiendo que alguien vea la lógica de negocio.

Un atacante podría descompilar el APK para entender cómo funciona, encontrar lógica sensible (como algoritmos o claves secretas hardcodeadas), modificarla, recompilar la app y distribuirla como una versión maliciosa.

Se recomienda Habilitar la ofuscación de código con R8/ProGuard: Al generar el APK para producción (release), asegúrar de que la ofuscación y la minimización estén activadas. Esto cambia el nombre de tus clases, métodos y variables a nombres cortos e incomprensibles, haciendo la ingeniería inversa mucho más difícil. Por lo que nunca se deben guardar claves secretas o secrets en tu código.

Nunca escribas contraseñas, claves de API o cualquier otro secreto directamente en el código fuente.

Plan de estrategias para la proteccion para Ataques informaticos:

1.Revisar las consultas de Room: Confirma que estás utilizando parámetros (:variable) y no concatenación de strings para las consultas.

2.Cifrar los datos guardados: Usar EncryptedSharedPreferences para cualquier dato que no deba ser públicamente visible en un dispositivo rooteado.

3.Actualizar las dependencias: Usa las últimas versiones estables y considera activar Dependabot

.4.Configurar las ofuscación: Asegúrar de que R8/ProGuard esté activado para tus compilaciones de release para dificultar la ingeniería inversa.Al ser una aplicación local, el principal riesgo es la seguridad de los datos en el dispositivo.