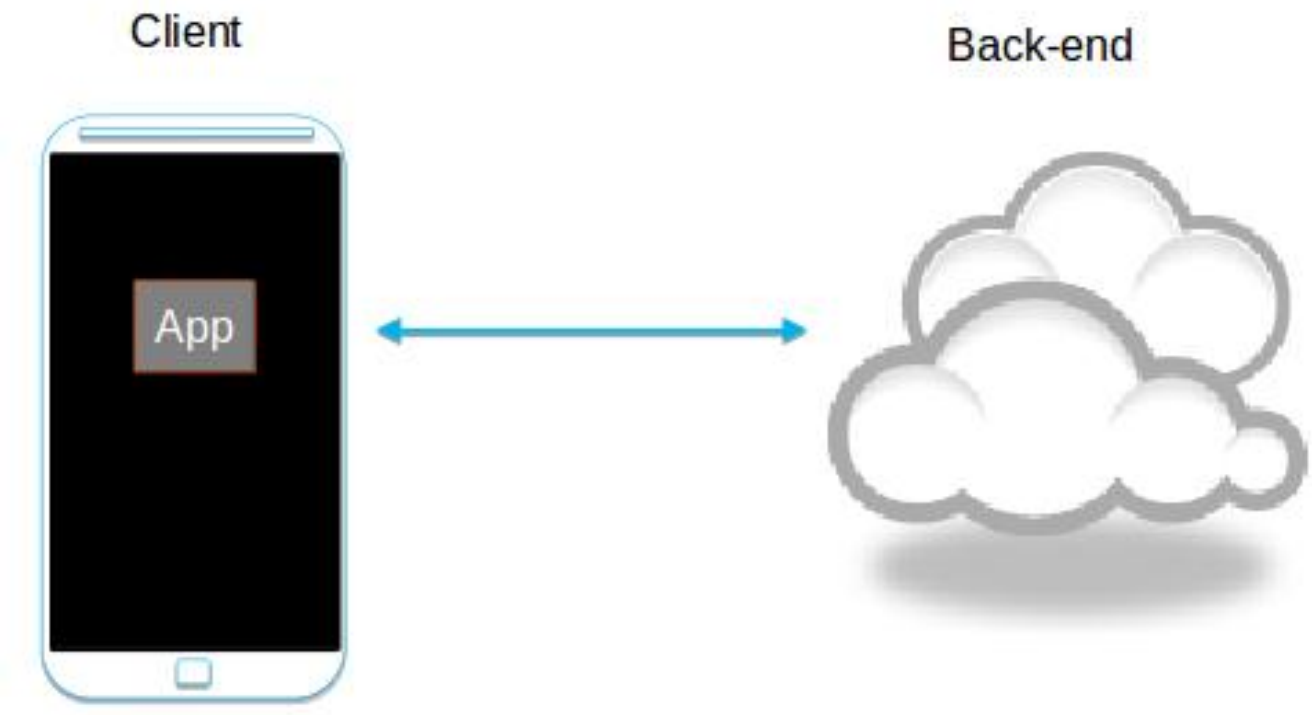
Vulnerabilidades en aplicaciones y desarrollo seguro en Android

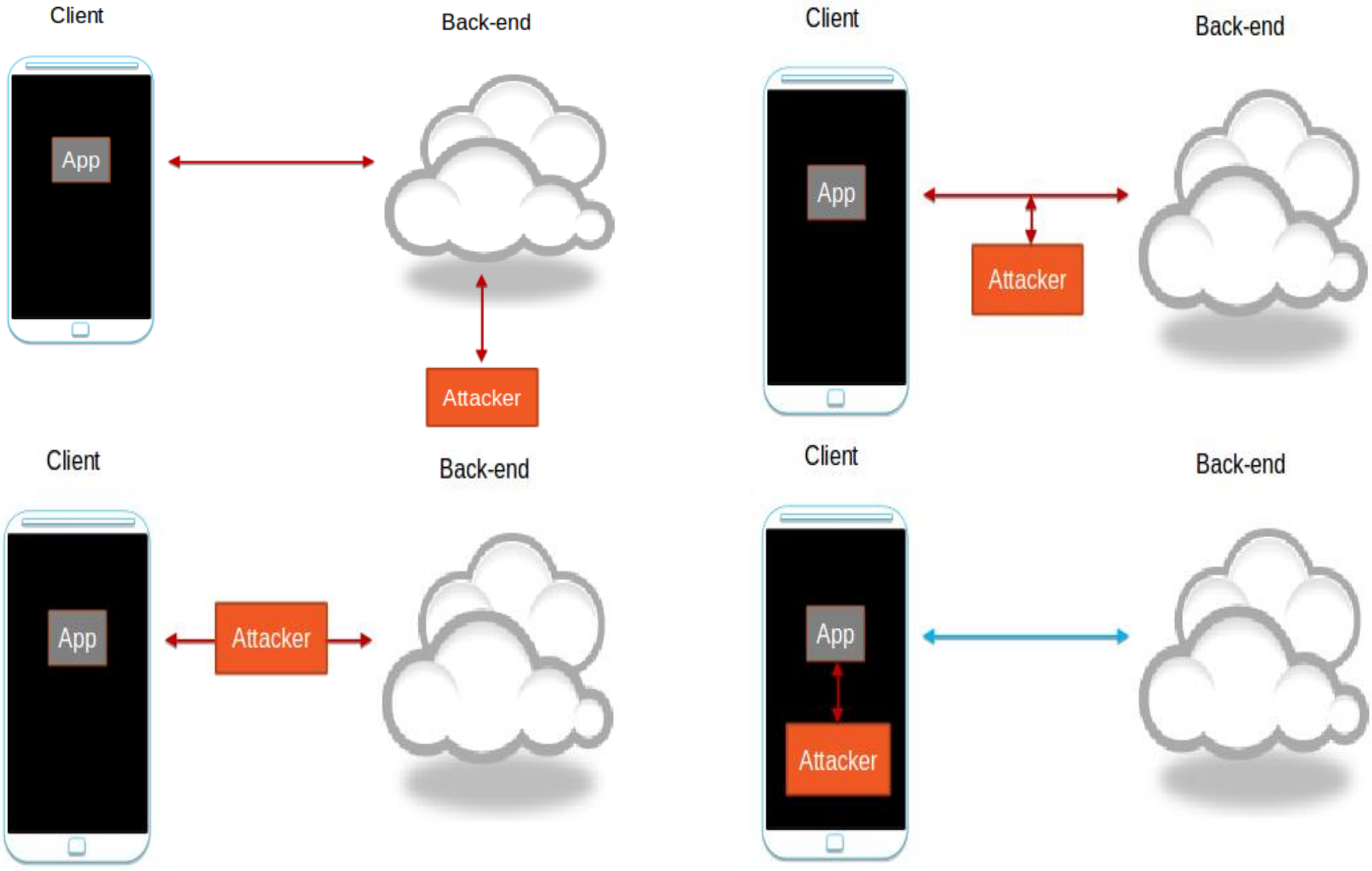
Juan Heguiabehere Joaquín Rinaudo

22° Escuela de Verano de Ciencias Informáticas
RIO 2015

Arquitectura típica

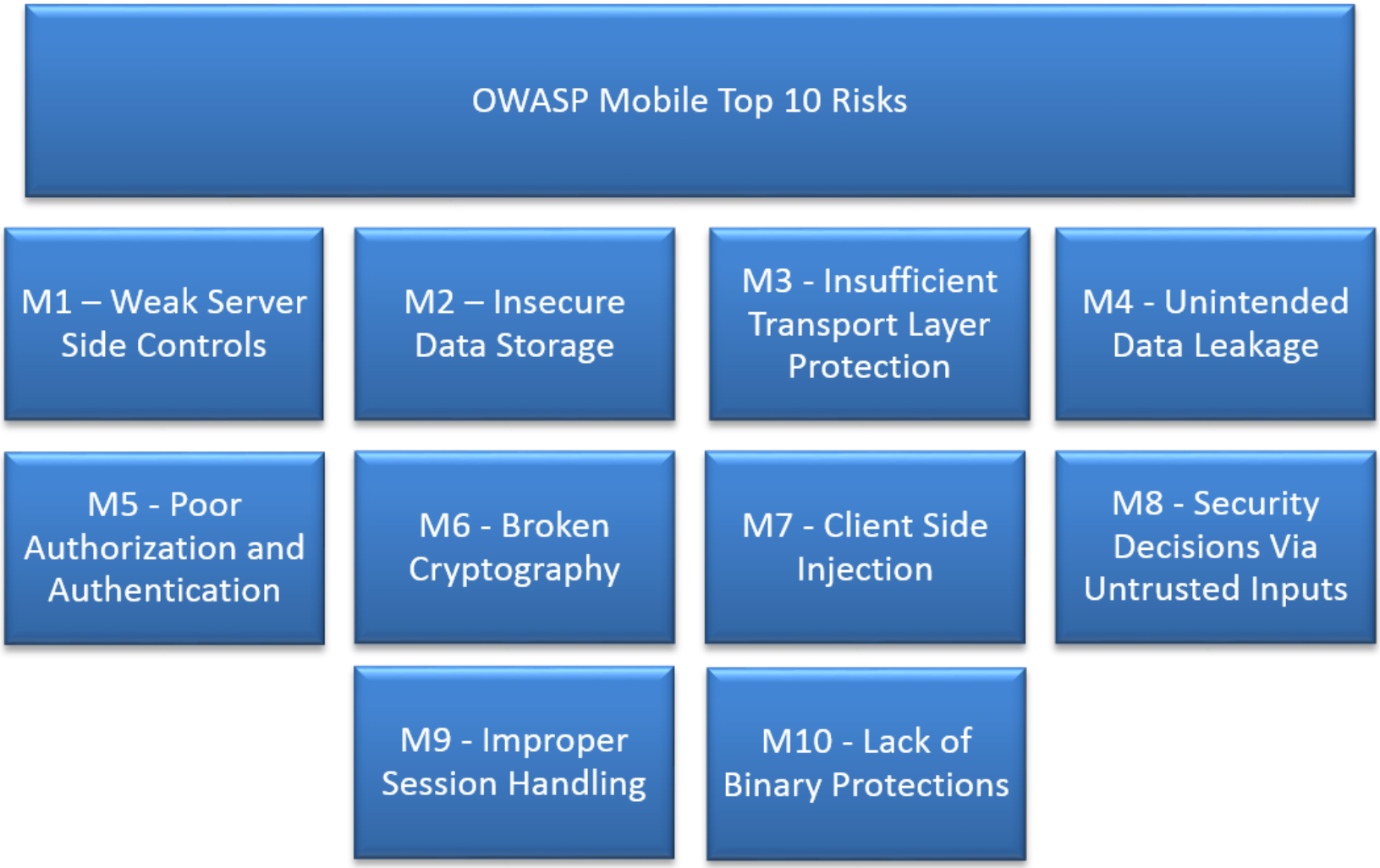


Vectores de ataque



Anatomía de un ataque a un dispositivo móvil

OWASP Top Ten Mobile



M1 - Controles pobres en el servidor

Todo aquello que no ocurre en el dispositivo Android:

- Sesiones determinísticas
- Problemas en autenticación o en los controles de autorización
- Vulnerabilidad a la inyección de SQL
- Exposición de datos sensibles
- Falta de validación del formato (y la lógica) de los pedidos de la API del dispositivo
- Utilizar componentes con vulnerabilidades conocidas (ej: Apache Server 2.2.0 en Windows)

M2 - Almacenamiento inseguro de datos privados

La memoria interna de la aplicación está en:
`/data/data/[Nombre de la aplicación]`

Otras aplicaciones no tienen acceso a esa carpeta por defecto.

La memoria externa (o SD) puede ser leída por todas las aplicaciones sin ningún permiso (hasta la API 19)

M2 - Almacenamiento inseguro de datos privados

Qué hay en la memoria interna de una app:

- `shared_prefs`: Archivos XML de diccionario clave-valor (muchas veces se guardan las contraseñas, claves de sesión y demás ahí)
- `databases`: Bases de datos SQLite, pueden ser de ContentProviders y datos de WebViews (En el archivo `webview.db` puede haber cookies)
- `libs`: Bibliotecas nativas utilizadas por la aplicación
- `cache`: Data cacheada por la aplicación
- `files`: Archivos en general (puede haber certificados hardcodeados para SSL)

M2 - Almacenamiento inseguro de datos privados

Qué problemas puede haber:

- Guardar información que no es necesaria
  (ej: contraseña si tengo clave de sesión)
- Modo `WORLD_READABLE/WRITEABLE` para un archivo, sharedPreference o base SQL.
- Información sin ningún tipo de encriptación
  (Peligro: `allowbackups=true` en el manifiesto)
- Encriptar con claves hardcodeadas en la aplicación
- Información privada en la SD sin encriptar

M3 - Transmisión insegura

Muchas aplicaciones se comunican con servidores externos utilizando distintos protocolos (HTTP/S, UDP/TCP, SMTP, VOIP, etc)

Es importante analizar desde punto de vista de un atacante que información podría llegar a obtener:

- Información privada utilizando protocolos sin ninguna encriptación
- Información encriptada pero utilizando claves hardcodeadas

M3 - Transmisión insegura

- No validar los certificados al usar SSL/TLS
  - TrustManager:
    - Verifica que la cadena de validación del certificado sea confiable.
    - SSLSocketFactory con TrustManager cuyo `checkServerTrusted()` no hace nada
  - Overridear WebViewClient con `onReceiveSSLError` y sólo llamar a `proceed()`
  - `SSLCertificateSocketFactory#getInsecure`
  - HostNameVerifier:
    - Verifica que el certificado incluya a la URL que se está visitando
    - SSLSocketFactory con `ALLOW_ALL_HOSTNAME_VERIFIER`

M4 - Caching y Logging: Fuga de datos

Una aplicación puede guardar información inconcientemente. Esta información puede llegar a ser leída por otras aplicaciones.

- Logging: Hasta 4.1 de Android, cualquier aplicación podía llegar a leer los logs del resto
- Copy/Paste: Clipboard es común a todas las apps
- Cache Web: no-cache header o `clearCache()` (Cookies, parámetros en URL, etc)
- Diccionario de palabras del usuario: Para autocorrección. Apps pueden acceder a él.

M5 - Autenticación/autorización pobre

Autenticación: Validar la identidad de un sujeto

Autorización: Validar que un sujeto puede realizar una acción o acceder a un recurso

- Permitir realizar pedidos a de API sin autenticar
- Permitir realizar pedidos como un usuario con las cookies o claves de sesión de otro
- Usar IMEI para la autenticación. Si vendo el celular, el comprador se loguea a mi cuenta
- Autenticación persistente: (Recordarme) NO guardar la contraseña del usuario en el dispositivo
- No bloquear la cuenta tras N intentos fallidos

M6 - Mal uso de criptografía

Dos tipos de mal uso: Diseño de un proceso de encriptación roto o utilización de algoritmos débiles

Consejo: encriptar contenido con una clave aleatoria y luego encriptarla con clave o PIN de usuario

Malos diseños:

- Mal manejo de las claves:
  - Incluir clave en el mismo directorio que contenido encriptado
  - Hardcodear clave en el binario
  - Crear claves determinísticas
- Utilización de protocolos o implementaciones de criptografía propios

M6 - Mal uso de criptografía

Utilizar algoritmos débiles:

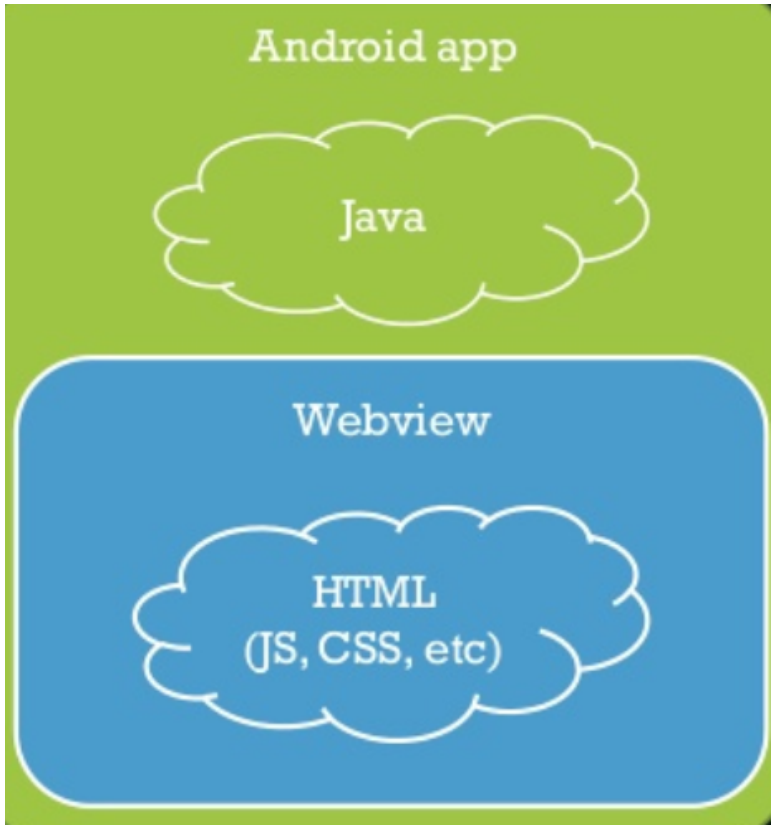
- MD4,MD5
- SHA1
- RC2
- Ni hablar de Caesar, Vigenere y demás...

WebViews

Una vista para mostrar contenido Web. Utiliza con la plataforma WebKit como motor de renderizado (hasta 4.4). Luego utiliza Chromium

Es un `layout` como un botón, texto, etc.

El entorno Java y el HTML están isolados



WebViews

Métodos :

- `setJavaScriptEnabled()`
- `addJavascriptInterface()`
- `setAllowFileAccess()`
- `setAllowFileAccessFromFileURLs()`
- `setAllowUniversalAccessFromFileURLs()`
- `setPluginsEnabled()`

WebViews

Métodos :

- `setJavaScriptEnabled()`: Habilitar ejecución de JavaScript. Deshabilitado por defecto
- `addJavascriptInterface()`
- `setAllowFileAccess()`
- `setAllowFileAccessFromFileURLs()`
- `setAllowUniversalAccessFromFileURLs()`
- `setPluginsEnabled()`

WebViews

Métodos :

- `setJavaScriptEnabled()`:
- `addJavascriptInterface()`: Habilitar ejecutar algunos métodos Java desde JavaScript
- `setAllowFileAccess()`
- `setAllowFileAccessFromFileURLs()`
- `setAllowUniversalAccessFromFileURLs()`
- `setPluginsEnabled()`

WebViews

Métodos :

- `setJavaScriptEnabled()`:
- `addJavascriptInterface()`:
- `setAllowFileAccess()`: Habilitar el esquema file:// para mostrar archivos como contenido HTML. Habilitado por defecto hasta 4.0.4
- `setAllowFileAccessFromFileURLs()`
- `setAllowUniversalAccessFromFileURLs()`
- `setPluginsEnabled()`

WebViews

Métodos :

- `setJavaScriptEnabled()`:
- `addJavascriptInterface()`:
- `setAllowFileAccess()`
- `setAllowFileAccessFromFileURLs()`: Acceso a archivos corriendo bajo esquema file:// a otros archivos file://. Prendido por defecto hasta 4.0.4
- `setAllowUniversalAccessFromFileURLs()`
- `setPluginsEnabled()`

WebViews

Métodos :

- `setJavaScriptEnabled()`:
- `addJavascriptInterface()`:
- `setAllowFileAccess()`
- `setAllowFileAccessFromFileURLs()`:
- `setAllowUniversalAccessFromFileURLs()`: Javascript de archivos corriendo en el contexto de esquema file:// pueden acceder al contenido de cualquier otro origen. Prendido por defecto hasta 4.0.4
- `setPluginsEnabled()`

WebViews

Métodos :

- `setJavaScriptEnabled()`:
- `addJavascriptInterface()`:
- `setAllowFileAccess()`
- `setAllowFileAccessFromFileURLs()`:
- `setAllowUniversalAccessFromFileURLs()`
- `setPluginsEnabled()`: Habilitar Plugins como Flash, deprecado a partir de Kitkat. Deshabilitado por defecto

WebViews vulnerabilidades: `addJavascriptInterface()`

Cargar contenido malicioso en el Webview y explotar la interfaz para ejecutar código nativo

Ejemplo:

```
android_interface.getClass().forName("java.lang.Runtime").
      getMethod("getRuntime",null).invoke(null,null).exec("ls");
```

Apartir de 4.2 sólo se puede llamar a métodos que estén anotados con @JavascriptInterface

Si app tiene `targetSDK` menor a 17 (4.2), es vulnerable hasta dispositivos corriendo 4.4.4

WebViews vulnerabilidades

Esquema `file://` puede llevar a vulnerabilidades al forzar Webview a abrir un archivo malicioso (por ejemplo uno en la SD)

- Inyectar una cookie con Javascript y redirigir a

  `file://data/data/pkg/databases/webviewCookiesChromium.db`

  donde se guardan las cookies
- `setAllowFileAccessFromFileURLs()` se puede usar JS para leer archivos en la memoria interna de la aplicación
- `setAllowUniversalAccessFromFileURLs()` se puede obtener además el contenido de sitios `http://` o `https://`

WebViews: Protecciones

- Deshabilitar todo lo que no se use (Javascript, uso de `file://`, plugins, etc)
- No exportar actividades con WebViews o evitar pasar URL por Intent
- WebViewClient `shouldOverrideUrlLoading()`: Whitelist para evitar cargar sitios de terceros
- Cargar los recursos remotos por canales seguros como SSL
- WebviewClient `shouldInterceptRequest()`: Whitelist interceptar pedidos de IFRAME, atributo src de scripts, HTML y pedidos de XmlHttpRequest

M7 - Inyección de código en el cliente

Input malicioso en forma de datos que termina siendo procesado como si fuese código

- Inyección SQL a ContentProviders exportados: Parametrizar las queries
- WebViews: Cargar sitios maliciosos que puedan ejecutar Javascript, XSS, CRSF y lo anterior

M8 - Componentes exportados: Inputs maliciosos

`android:exported=true` o `<intent-filter\>` hacen que un componente esté exportado

Pueden estar protegidos por permisos del sistema o propios.

Permisos propios: especificar

`protection-level: normal, dangerous, signature, signatureOrSystem`

M8 - Componentes exportados: Inputs maliciosos

Componentes exportados desprotegidos:

- ContentProvider: SQLInjection, Path Transversal, leer o escribir datos privados
- Service: Ejecutar tareas del Binder exportado o obtener información sensible
- Receiver: Engañar a la aplicación para que ejecute una acción privilegiada
- Activity: Realizar acciones por el usuario, inyectar código (WebView)

Componentes exportados: Intents implícitos

Hay que tener cuidado con los intents implicítos.
Componentes maliciosos podrían:

- Escuchar Broadcasts pasivamente si se envían Intents implícitos sin permiso Pueden hacer denial-of-service o modificar los datos en OrderedBroadcast
- Activity Hijacking: Llevar a abrir una Activity maliciosa que haga phishing o o devolver respuestas falsas desde esa Activity Más díficil, por que Android le pregunta al usuario
- Service Hijacking: Terminar conectándose a un servicio distinto del legítimo

PendingIntents: Siempre especificar explícitamente el componente destinatario

M9 - Manejo inapropiado de sesiones

Aplicaciones usan tokens de sesiones para mantener estado en protocolos sin uno como HTTP o SOAP. Tokens como resultado de autenticación del usuario con el servidor:

- No establecer time-out
- Claves de sesión determinísticas
- No enviar pedido de log-out para invalidar clave de sesión
- No rotar cookies cuando uno se autentica (Fijación de sesión)
- Enviar la clave de sesión por canales inseguros

M10 - Protección del APK

- Ofuscar el código previene ingeniería reversa
- Certificate pinning contra análisis dinámicos
- Controles de Checksum contra modificaciones
- Verificar dinámicamente firma de desarrollador
- Detección de Debugger
- Detección de ambientes virtuales