Arquitectura Mobile

Recomendaciones Generales

Torre Ejecutiva Sur
Liniers 1324, piso 4
Montevideo – Uruguay
Tel/Fax: (+598) 2901.2929*
Email: arquitectura@agesic.gub.uy

www.agesic.gub.uy

Registro de Revisiones

N° de Versión	Fecha	Realizado por	Detalle de la Revisión
1.0	24/04/2018		Creación del Documento
1.1	12/09/2018		Cambio de Plantilla

Tabla de Contenidos

1. Introducción....4
1.1. Propósito....4
2. Arquitecturas de Referencia....4
2.1. Aplicación Nativa....4
2.2. Aplicación Web....4
2.3. Aplicación Híbrida....5
2.4. Consideraciones generales....6
3. Consideraciones de Diseño....8
3.1. Tipos de conectividad....8
3.2. Tipo de Servicio....9
3.3. Sincronización....10
3.4. Push Notifications....10
4. Seguridad....11
5. Seguridad – MASVS....22

1. Introducción

1.1. Propósito

El propósito del presente documento es brindar una visión comprensible de las diferentes arquitecturas para aplicaciones móviles.

Este documento profundiza principalmente en tres tipos de arquitecturas, diferentes formas de comunicación, tipos de conectividad de las aplicaciones móviles (online y offline), mecanismos para la implementación de push notifications.

2. Arquitecturas de Referencia

La arquitectura está representada por diferentes vistas de forma que permitan.

2.1. Aplicación Nativa

Aplicación que se desarrolla en el lenguaje nativo de cada dispositivo (Android, IOS, entre otros). Son archivos ejecutables que se instalan en los dispositivos y por lo general se descargan desde la tienda correspondiente (Marketplace de Android, App Store de Apple, entre otros).

Al ser una aplicación nativa, se conecta directamente con el sistema operativo móvil, sin intermediario. Accede a todas las APIs disponibles del Sistema Operativo, esto permite que la aplicación pueda tener características y funciones que son típicas del mismo.
Cada Sistema Operativo móvil cuenta con sus propias herramientas.
El código escrito para una plataforma móvil no se puede usar en otra.

2.2. Aplicación Web

Aplicación web de tipo cliente servidor.
Se desarrolla con tecnologías como HTML5, Cascading Style Sheets 3 (CSS3) y JavaScript, para lograr interfaces de usuario avanzadas.

Este tipo de aplicaciones puede contar con interfaces avanzadas, servicios de geo posicionamiento y disponibilidad offline.
Se parece a una aplicación nativa y se puede ejecutar a partir de un acceso directo que es idéntico al que se utiliza para lanzar aplicaciones nativas.
Una de las principales ventajas de una aplicáión Web es su soporte para múltiple plataformas y el bajo costo de desarrollo.

En la tabla a continuación se realiza una comparación entre una aplicación we para móviles y un sitio web para móviles:

	Aplicación web solo móviles	**Sitio web solo móviles**
Herramientas	HTML, CSS y Javascript	HTML, CSS y Javascript
Ejecución	Acceso Directo	Navegando por URL
Desempeño	UI reside local y capacidad de respuesta y acceso offline	Todo el código se ejecuta en el servidor

Las aplicaciones web, a diferencia de las aplicaciones nativas, que son ejecutables independientes y se conectan directamente con el SO, las aplicaciones web se ejecutan dentro del navegador.
El navegador es la aplicación nativa que tiene acceso directo a las APIs, pero muy pocas de esas APIs está expuestas a las aplicaciones Web que se ejecutan dentro del mismo.
Las aplicaciones nativas tienen acceso completo al dispositivo, para las aplicaciones Web muchas funcionalidades no están disponibles o solamente en forma parcial.

2.3. Aplicación Híbrida

Combina desarrollo nativo con tecnología Web.
Se escribe gran parte de su aplicáión en tecnología Web y mantienen el acceso directo a APIs nativas cuando lo necesita.
La parte nativa de la aplicación utiliza APIs para crear un motor de búsqueda HTML incorporado, que funciona como un intermediario entre el navegador y las APIs del dispositivo.
La parte Web de la aplicación pueden ser páginas que residan en un servidor o incorporados en el código de la aplicáión alojadas en el dispositivo.

2.4. Consideraciones generales

Cuando elegir el enfoque nativo

- Si la distribución debe ser a través de las diferentes tiendas.
- Se necesita una API específica del Sistema Operativo.
- Hay una alta frecuencia de uso de la aplicación
- Se va a desarrollar para una sola plataforma (Evitar tener que desarrollar en varios lenguajes a la vez o utilizar una herramienta que desarrolle en varias plataformas de forma simultanea)
- Hay requerimientos de UI sofisticados

Cuando elegir enfoque Web (Sitio)

- La frecuencia de uso es más baja
- La aplicación es más bien de consulta de datos, no tanto de ingreso.
- Es necesario la distribución por fuera de las tiendas.
- Hay actualizaciones de versiones frecuentes.
- No es necesario el uso de muchas Apis específicas.
- No hay requerimientos de performance complejos

Cuando elegir enfoque Hibrido

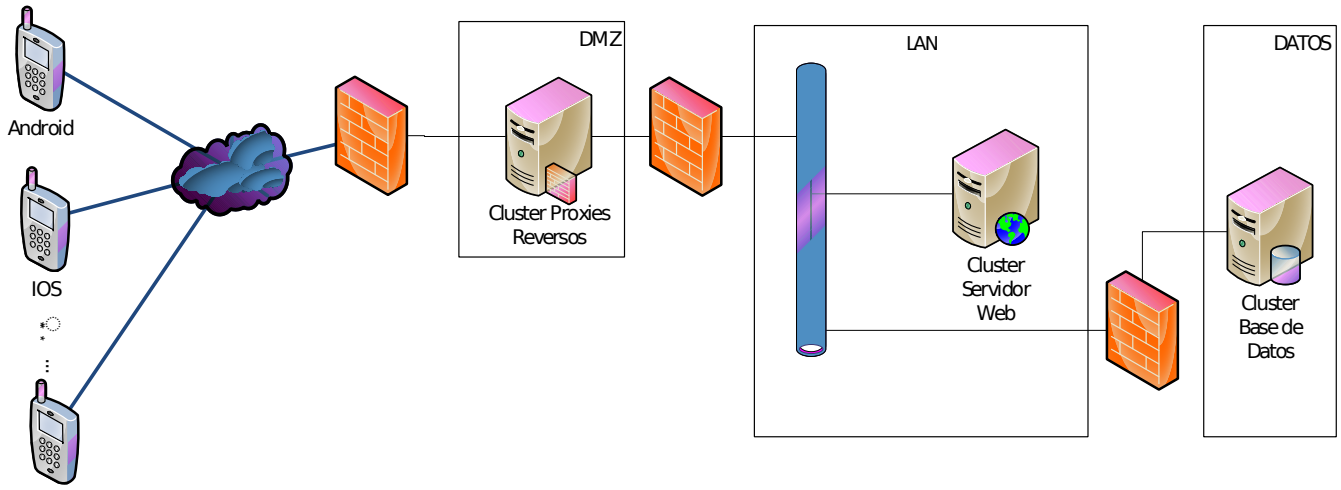
- En necesaria la distribución a través de las tiendas.
- Se puede realizar un desarrollo “cross-platforms” y no desarrollar varias aplicaciones.
- Hay necesidad media o alta de uso de APIs del Sistema Operativo.
- Exigencias de interfaz de usuario medias.

A continuación de muestra un cuadro comparativo, con respecto al costo que tiene implementar algunas características de las aplicaciones, dependiendo el tipo.

Característica	Nativa	Híbrida	HTML/Web
Cross-Platform	No	Medio	Si
Costo desarrollo	Alto	Medio	Bajo

Interfaz de Usuario	Alto	Medio	Bajo
Distribución por Tienda	Si	Si	No
Acceso offline	Si	Si	Parcial
Acceso a APIs	Completo	Alto	Bajo
Performance	Alto	Medio	Bajo

Más allá de la decisión que se tome en cuanto al tipo de aplicación que se desarrolla, se puede tener una línea base con respecto al despliegue de estas aplicaciones. A continuación se presenta un despliegue como línea base.



El diagrama anterior representa de forma conceptual un despliegue propuesto.

Es un despliegue posible de la aplicación, que debe ser analizado según cada caso.

En el diagrama se diferencian 4 distintas segmentos de red: DMZ, LAN y DATOS.

En la DMZ se ubica un cluster de proxies reversos que permiten acceder a los servicios brindados por la LAN a Internet (o hacia el exterior).

En la zona LAN se ubican los servidores que implementan la lógica del sistema. En este caso se representa un servidor web.

Finalmente, en la zona DATOS se ubican los motores de base de datos.

3. Consideraciones de Diseño

Existen muchas consideraciones al momento de diseñar una aplicación de este tipo. Entre ellas está el tipo de conectividad a manejar, que se analiza a continuación.

3.1. Tipos de conectividad

3.1.1. Conectividad Online

Ventajas

- Backend actualizado al instante
- Toda la información reside en el servidor

Desventajas:

- Si no hay red, no funciona la aplicación
- La performance depende de la conexión

3.1.2. Conectividad Offline

Ventajas:

- Velocidad de acceso a los datos
- Operación continua, incluso sin red
- Performance

Desventajas:

- La inicialización puede tardar, ya que descarga los datos para trabajar offline

- Solo parte de los datos se descargan, pero podría llegar a necesitar datos que no fueron descargados
- La sincronización de los datos puede llegar a tener conflictos
- La información reside en los dispositivos

3.1.3. Conectividad mixta, online y offline al mismo tiempo

Ventajas:

- Búsquedas de información descargada es instantánea
- Continúa operando sin conexión

Desventajas

- Mayor costo en el desarrollo, ya que se deben contemplar ambos escenarios
- Inicialización puede tardar, ya que descarga datos para trabajar offline
- Información reside en los dispositivos
- La sincronización de los datos puede tener conflictos

3.2. Tipo de Servicio

Los tipos de servicios son REST o SOAP.

REST:

- Los mensajes son más livianos
- Tiene un uso menos de ancho de banda en comparación con SOAP.

SOAP:

- Son servicios más seguros que los REST.
- Tienen un mayor uso de ancho de banda por el tamaño de sus mensajes.

3.3. Sincronización

Consideraciones que se deben incluir a la hora de sincronizar datos:

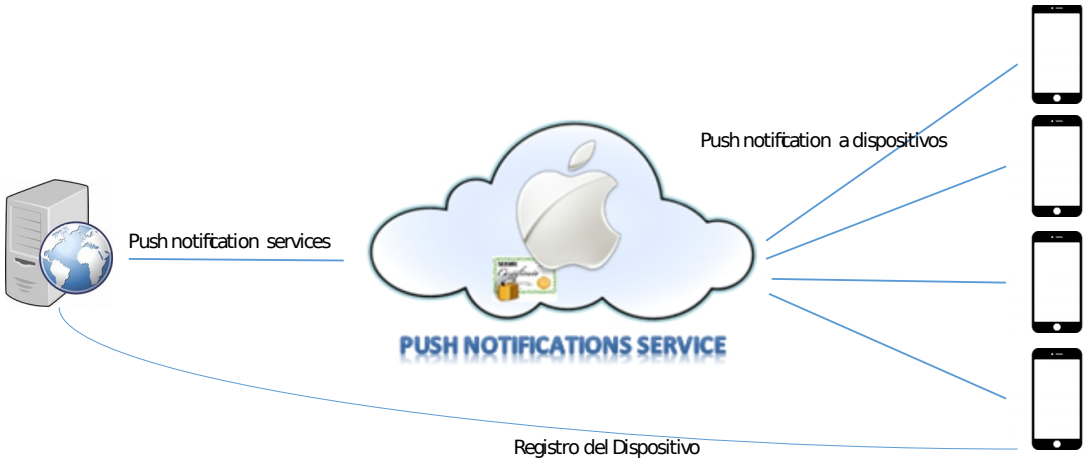
Cómo en toda sincronización se deben contemplar casos de:

- caídas de red en el proceso de sincronización
- ¿Se cancela la sincronización? ¿permitiendo retoma?
- Descargar la mínima información necesaria (seguridad).

3.4. Push Notifications

Para la implementación de notificaciones push, existen los servicios directos de Apple y Android. Y además, existen proveedores externos como por ejemplo One Signal.

Apple (iOS): Apple Push Notifications Service (APNS)
Android: Google Cloud Messaging (GCM)
Externo: ejemplo One Signal



En el diagrama anterior se puede ver cómo funciona el mecanismo de push notifications, en este caso el ejemplo es con el servicio de Apple.
Existe un registro del dispositivo en el servidor de notificaciones.
Cuando se lanza una notificación, lo hace el servidor de notificaciones hacia el servicio “Push Notifications Service”, y éste último se encarga de enviarlo a los diferentes dispositivos.

4. Seguridad

Definiciones:

- Identity Provider (**IDP**): Es el proveedor de identidades, es el actor del sistema de clave única al cual se le delega la autenticación.
- Service Provider (**SP**): Es la aplicación de negocio que actuará como cliente delegando la autenticación al IDP. O sea, es el sistema que se quiere integrar al mismo. Ej: portal de organismo o empresa.

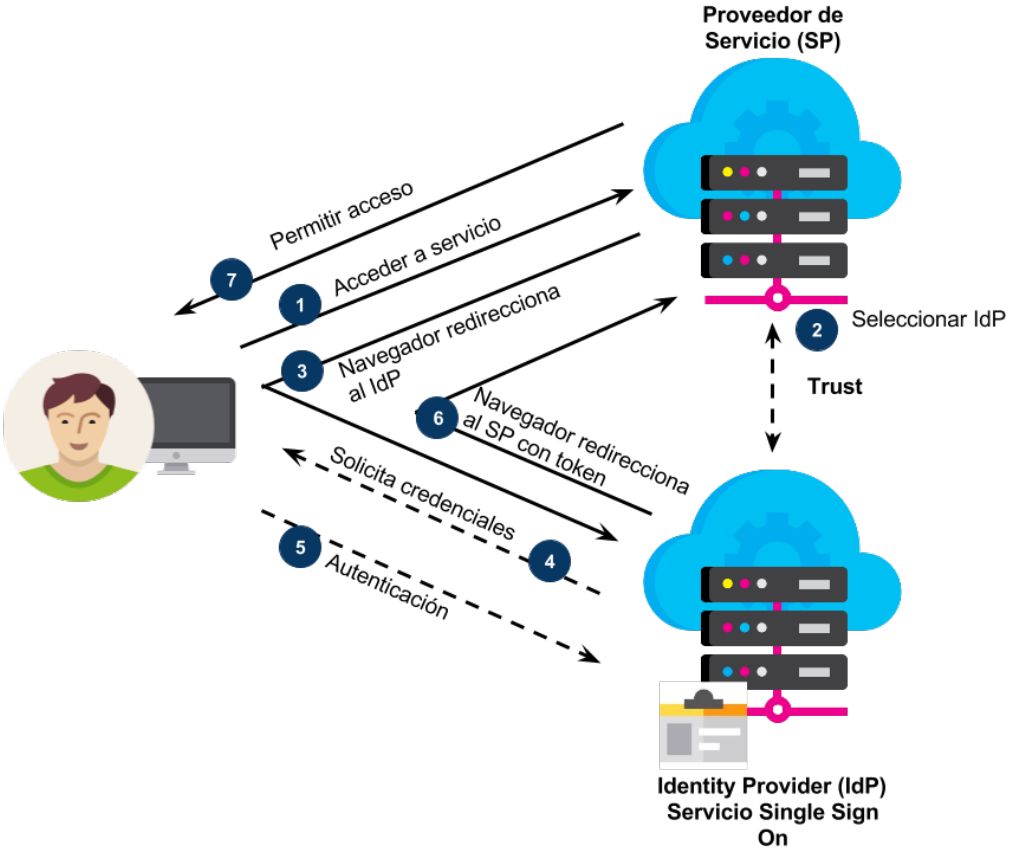
IDP de AGESIC para aplicaciones móviles

Para implementar la autenticación se podrían utilizar las siguientes opciones:

A. SAML
B. OpenID Connect

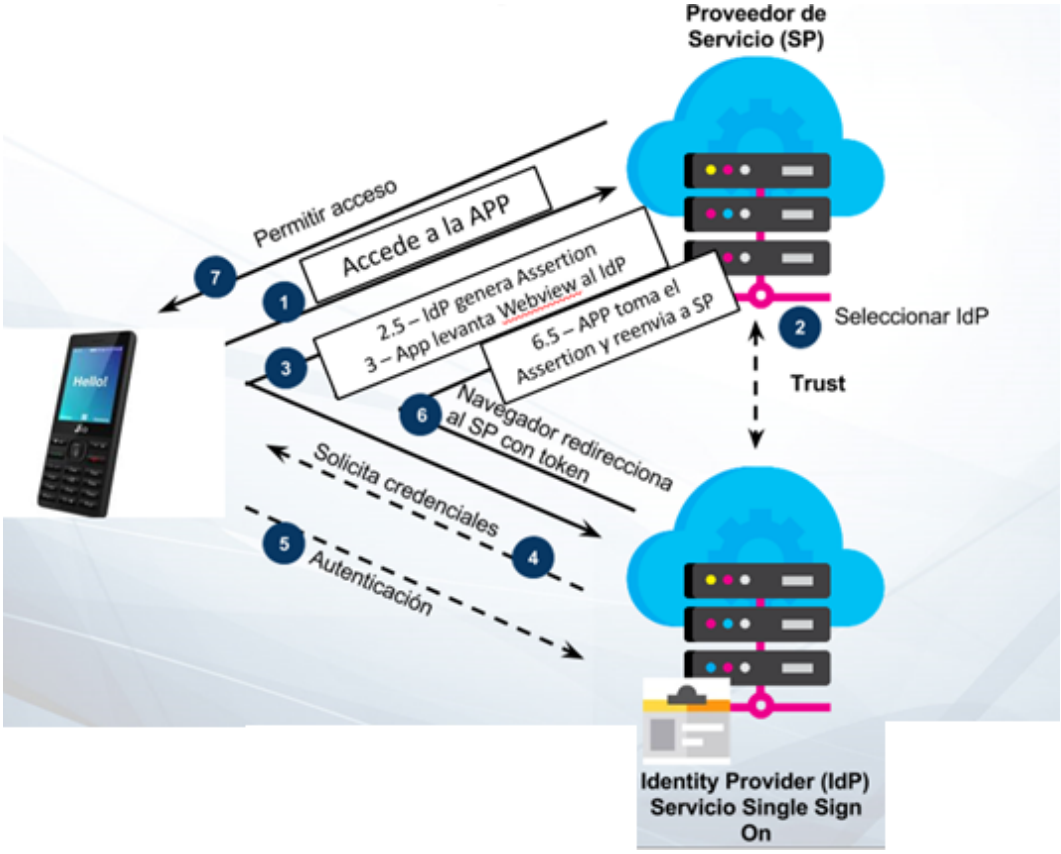
A) SAML (Security Assertion Markup Language)

- ¿Qué es? es un estándar OASIS basado en XML para el intercambio de información de la identidad de usuario y atributos de seguridad.
- ¿Cómo se utiliza para aplicaciones Web? (ya conocido)



- ¿Cuál es el problema de utilizar SAML para autenticarnos con aplicaciones móviles? No se puede generar el assertion en el dispositivo Mobile, porque se debería almacenar la clave privada dentro del mismo. Por lo tanto, la arquitectura tal como está hoy propuesta no se adecua a dispositivos Mobile.

- Posible solución para utilizar SAML:



1 - La aplicación móvil provee un token de acceso del servidor SP.

2.A - El SP valida el token → autenticado.

2.B - El SP no valida el token → genera un Assertion SAML.

3 - La aplicación móvil muestra un navegador con el IdP con dicho assertion.

4 - El usuario ingresa su contraseña en el IdP. Retorna el Assertion

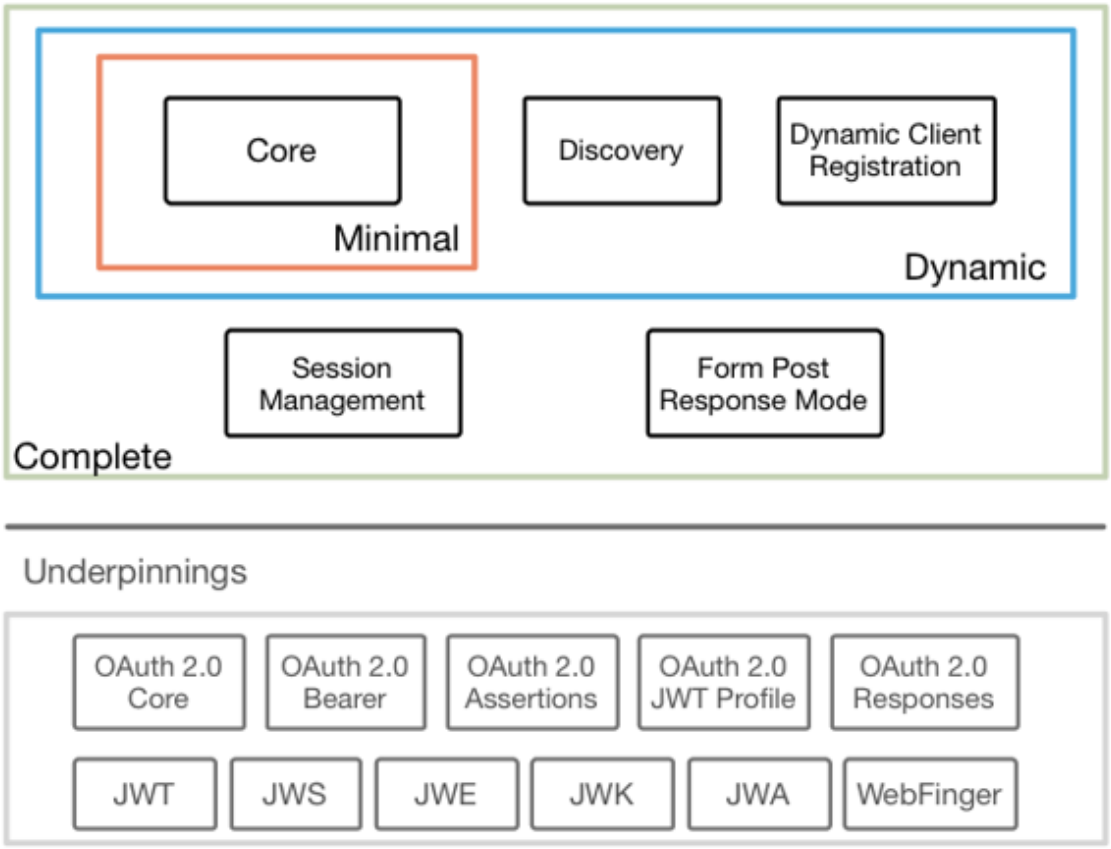
5 – La APP toma el assertion del webview y lo envía al SP.

6 – El SP crea un nuevo token y lo retorna a la APP. Vuelve al paso 1

Consideraciones: Esta solución funciona más como autenticador, el SP debería hacer la gestión de la sesión.

B) OpenID Connect

- ¿Qué es? OpenID Connect es una capa de identidad simple sobre el protocolo OAuth 2.0. En términos técnicos, OpenID Connect especifica una API RESTful HTTP, utilizando JSON como formato de datos.



- ¿Cuáles son sus características mas relevantes?
- Tokens de identidad: Las aplicaciones cliente reciben la identidad del usuario codificada en un JSON Web Token (JWT), llamado token ID. Este se encuentra firmado por el **proveedor de OpenID (OP).** Para obtener uno, el cliente debe enviar al usuario a su OP con una solicitud de autenticación.

Características del token ID:

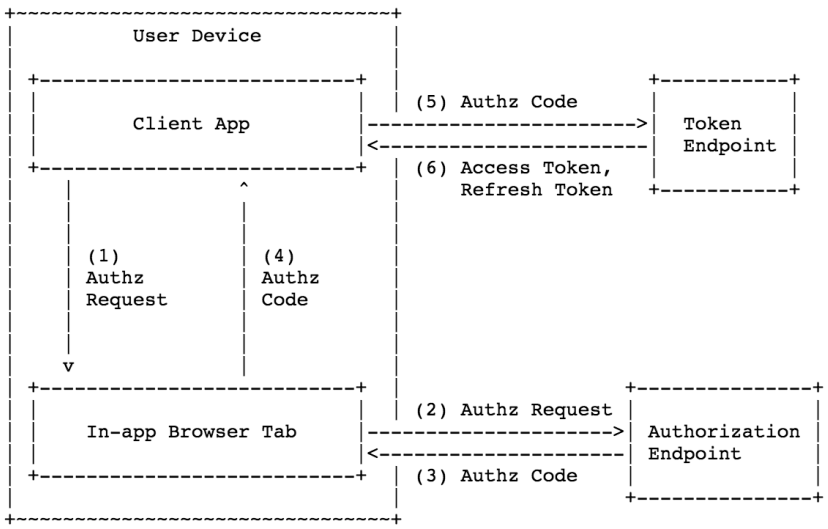
I. Afirma la identidad del usuario, llamado sujeto en OpenID (sub).
II. Especifica la autoridad emisora (iss).
III. Puede especificar cuándo (auth_time) y cómo, en términos de fuerza (acr), el usuario fue autenticado.

IV. Tiene un tiempo de expiración (exp).
V. Está firmado digitalmente, por lo que puede ser verificado por los destinatarios previstos.

1. Basado en el protocolo OAuth 2.0: Utiliza flujos de OAuth 2.0 para obtener tokens de identificación, diseñados para adaptarse tanto a aplicaciones web como a aplicaciones nativas / móviles.
2. Suficientemente simple como para integrarse con aplicaciones básicas, pero también tiene las características y las opciones de seguridad para satisfacer los requisitos empresariales más exigentes.

- ¿Cómo solicitar un Token ID? Los tokens de identificación se solicitan a través del protocolo OAuth 2.0 (RFC 6749). Originalmente, OAuth se diseñó como un mecanismo de autorización simple para que las aplicaciones obtengan tokens de acceso para API web u otros recursos protegidos. Tiene flujos diseñados para todos los tipos de aplicaciones: aplicaciones web tradicionales basadas en servidor, aplicaciones solo de navegador (JavaScript) y aplicaciones nativas / móviles.

- ¿Cuál es el flujo para obtener un Token ID?

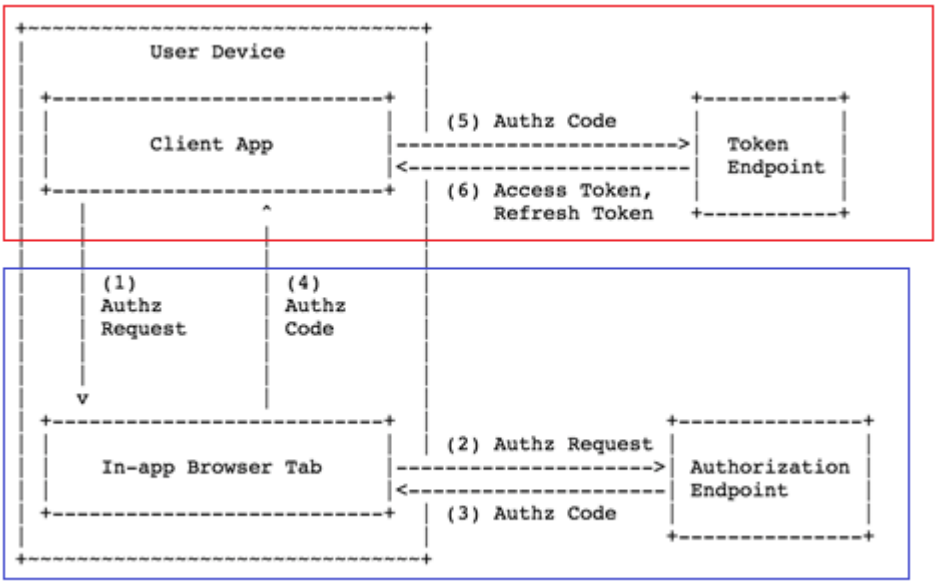


- Ejemplo de posible implementación de autenticación con OpenID Connect

El flujo de código tiene dos pasos:

	Step 1	Step 2
Purpose	1. Authenticate user 2. Receive user consent	1. Authenticate client (optional) 2. Exchange code for token(s)
Via	Front-channel request (browser redirection)	Back-channel request (app to web server)
To	Authorisation endpoint	Token endpoint
Result on success	Authorisation code (step 2 input)	ID token (+ OAuth 2.0 access token)

Donde el primero paso es lo marcado con color azul y el segundo paso el marcado con color rojo:



Paso 1:

El Relying Party (RP) inicia la autenticación del usuario al redirigir el navegador al punto final de autorización OAuth 2.0 del proveedor de OpenID (OP). La solicitud de autenticación de OpenID es esencialmente una solicitud de autorización de OAuth 2.0 para acceder a la identidad del usuario, indicada por un valor **openid** en el parámetro **scope**.

- Parámetros del pedido:
  - response_type: seteado en **code** para indicar el flujo de autorización
  - scope: Usado para especificar el alcance de la autorización solicitada en OAuth. El valor de **openid** señala una solicitud de autenticación OpenID y token de identificación.
  - client_id: El identificador del cliente del RP en el OP. Este identificador se obtiene normalmente cuando el RP se registra con el OP
  - state: Valor establecido por el RP para mantener el estado entre la solicitud y la devolución de llamada.
  - redirect_uri: URL de respuesta

- Pedido:

```
HTTP/1.1 302 Found
Location: https://openid.c2id.com/login?
          response_type=code
          &scope=openid
          &client_id=s6BhdRkqt3
          &state=af0ifjsldkj
          &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

- Respuesta:

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?
          code=SplxlOBeZQQYbYS6WxSbIA
          &state=af0ifjsldkj
```

Código de autorización

El OP llamará al cliente a redirect_uri con un código de autorización (en caso de éxito) o un código de error (si se denegó el acceso o se produjo algún otro error)

Paso 2:

En el paso dos se obtiene el **code** de la respuesta del paso 1, y se realiza un POST al **token endpoint.**

Además de la autenticación básica HTTP, OpenID Connect también permite la autenticación con aserciones JWT firmadas, que no exponen el secreto del cliente con la solicitud del token y por lo tanto, ofrecen una mayor seguridad.

- Parámetros del pedido:
  - grant_type: Seteado en **authorization_code**
  - Code: código obtenido en el paso 1
  - redirect_uri: URL de respuesta

- Pedido:

```
POST /token HTTP/1.1
Host: openid.c2id.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code
 &code=SplxlOBeZQQYbYS6WxSbIA
 &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

- Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
    yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
    NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
    fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
    AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
    Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
    NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
    QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
    K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
    XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
  "access_token": "SlAV32hkKG",
  "token_type": "Bearer",
  "expires_in": 3600,
}
```

Donde en la respuesta se obtienen el **id_token** con la información del usuario y el **access_token** que se utiliza para obtener las Claims.

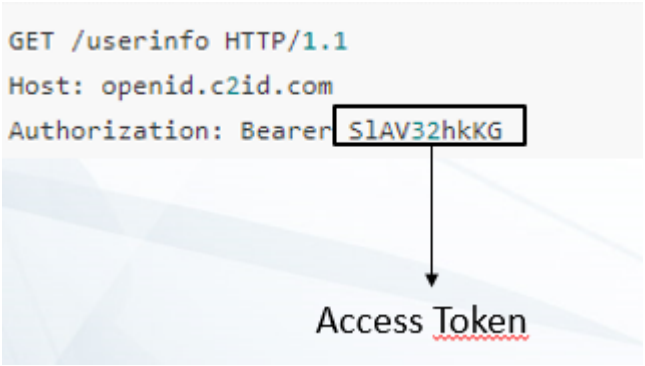
Por mas información de otras posibles implementaciones: https://www.youtube.com/watch?v=WVCzv50BsIE

- ¿Qué son las Claims (user info)?

OpenID Connect especifica un conjunto de declaraciones estándar o atributos de usuario. Están destinados a proporcionar a la aplicación del cliente los detalles del usuario como el correo electrónico, el nombre y la imagen del mismo.

Scope value	Associated claims
email	email, email_verified
phone	phone_number, phone_number_verified
profile	name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, updated_at
address	address

➢ Ejemplo pedido:



➢ Ejemplo de respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "sub"                       : "alice",
  "email"                     : "alice@wonderland.net",
  "email_verified"            : true,
  "name"                      : "Alice Adams",
  "picture"                   : "https://c2id.com/users/alice.jpg"
}
```

- ¿Cuáles son los endpoints que nos provee OpenID Connect?

Core endpoints	Optional endpoints
▪ Authorisation ▪ Token ▪ UserInfo	▪ WebFinger ▪ Provider metadata ▪ Provider JWK set ▪ Client registration ▪ Session management

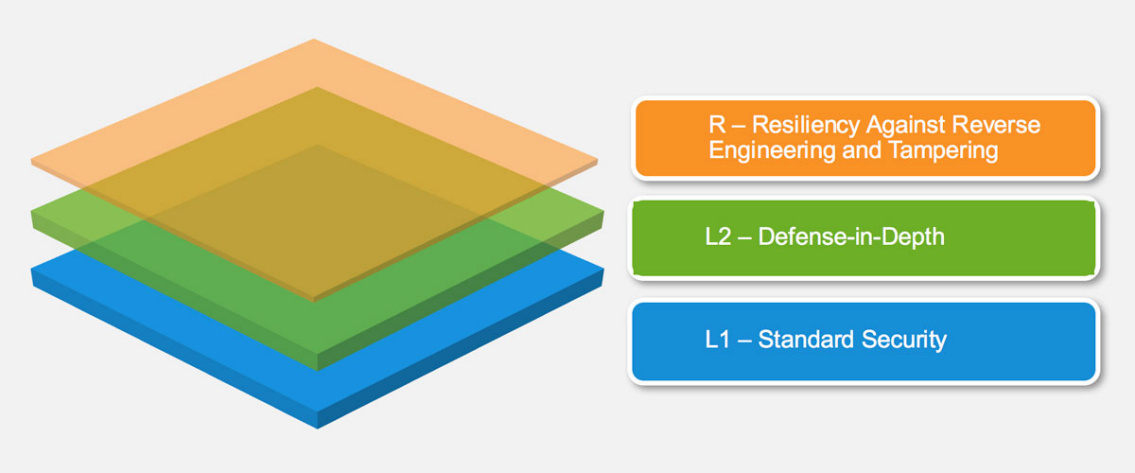
5. Seguridad – MASVS

Estándar de Verificación de Seguridad de Aplicaciones Móviles (MASVS):

El MASVS se puede utilizar para establecer un nivel de confianza en la seguridad de las aplicaciones móviles. Los requerimientos fueron desarrollados con los siguientes objetivos:

- Usar como una métrica: para proporcionar un estándar de seguridad contra el cual las aplicaciones móviles existentes pueden ser comparadas por desarrolladores y los propietarios de las aplicaciones.
- Utilizar como guía: proporcionar una guía durante todas las fases del desarrollo y prueba de las aplicaciones móviles.
- Usar durante la contratación: proporcionar una línea de base para la verificación de seguridad de aplicaciones móviles.

Niveles de verificación detallados



L1: Seguridad Estándar

Una aplicación móvil que logra el nivel MASVS-L1 se adhiera a las mejores prácticas de seguridad en aplicaciones móviles. Cumple con los requerimientos básicos en términos de calidad de código, manejo de los datos sensibles e interacción con el entorno móvil. Debe existir un proceso de pruebas para verificar los controles de seguridad. Este nivel es apropiado para todas las aplicaciones móviles.

L2: Defensa en Profundidad

MASVS-L2 introduce controles de seguridad avanzados que van más allá de los requisitos estándar. Para cumplir con el nivel L2, debe existir un modelo de amenaza y la seguridad debe ser una parte fundamental de la arquitectura y el diseño de la aplicación. Este nivel es apropiado para aplicaciones que manejan datos sensibles, como las aplicaciones de banca móvil.

R: Resistencia contra la ingeniería inversa y la manipulación

La aplicación cuenta con el nivel de seguridad específico para la aplicación y también es resistente a ataques específicos y claramente definidos en el lado del cliente, como alteración, modificación o ingeniería inversa para extraer código o datos sensibles. Esta aplicación OWASP Mobile Application Security Verification Standard v1.0 10 aprovecha las características de seguridad del hardware o bien técnicas de protección de software suficientemente fuertes y verificables. MASVS-R es adecuado para las aplicaciones que manejan datos altamente confidenciales y puede servir como medio para proteger la propiedad intelectual o la manipulación de una aplicación.

Uso recomendado

Las aplicaciones pueden ser verificadas contra el nivel MASVS L1 o L2 de acuerdo con la evaluación previa del riesgo y el nivel general de seguridad requerido. L1 es aplicable a todas las aplicaciones móviles, mientras que L2 se recomienda generalmente para las aplicaciones que manejan datos y/o funciones sensibles. MASVS-R (o partes de él) puede aplicarse para verificar la resistencia frente a amenazas específicas, como el reempaquetado o la extracción de datos sensibles, además de una verificación de seguridad adecuada.

En resumen, están disponibles los siguientes tipos de verificación:

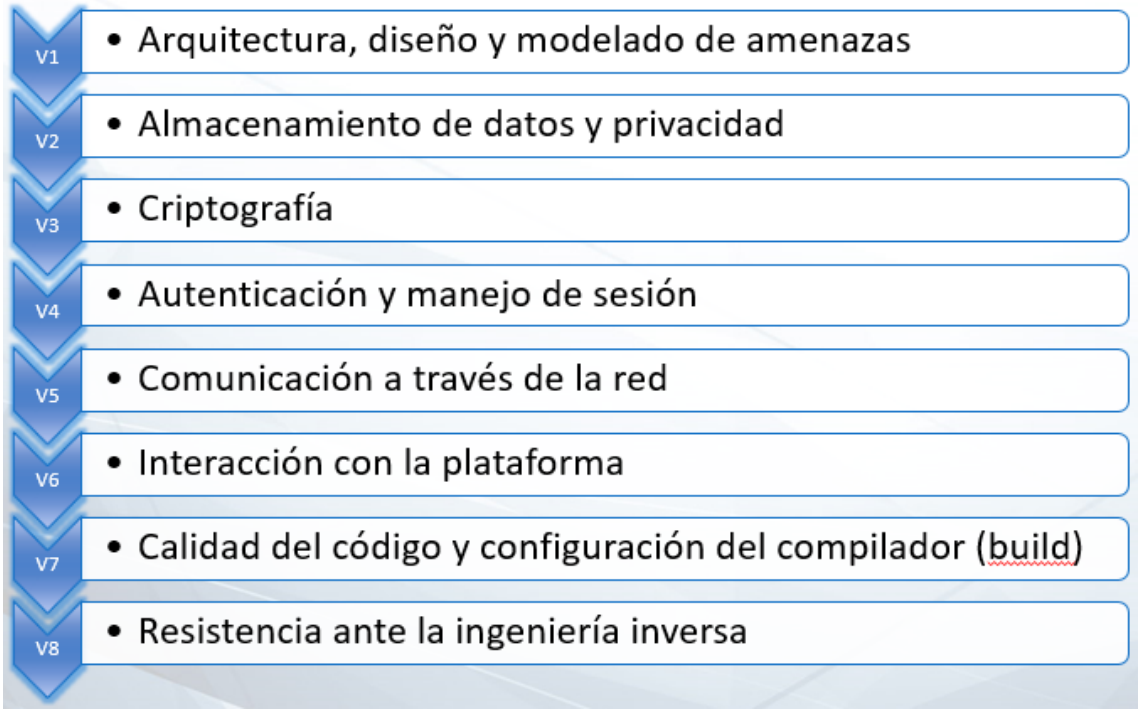
- MASVS-L1
- MASVS-L1+R
- MASVS-L2
- MASVS-L2+R

Las diferentes combinaciones reflejan diferentes grados de seguridad y resistencia. El objetivo es permitir la flexibilidad: Por ejemplo, un juego móvil puede no requerir controles de seguridad del MASVS-L2, como la autenticación de 2 factores por razones de usabilidad, pero seguramente deba prevenir la manipulación del código por razones del negocio.

¿Qué tipo de verificación elegir?

La implementación de los requisitos del nivel MASVS L2 aumenta la seguridad, mientras que al mismo tiempo aumenta el costo de desarrollo y potencialmente empeora la experiencia del usuario final (el compromiso clásico). En general, L2 debe utilizarse para aplicaciones siempre que tenga sentido desde el punto de vista del riesgo contra el costo que conlleva (es decir, cuando la potencial pérdida causada por un compromiso de confidencialidad o integridad sea superior al costo que suponen los controles de seguridad adicionales). Una evaluación del riesgo debe ser el primer paso antes de aplicar el MASVS.

Categorías



- V1 - Requerimientos de Verificación de Seguridad

#	Descripción	L1	L2
1.1	Todos los componentes se encuentran identificados y asegurar que son necesarios.	✓	✓
1.2	Los controles de seguridad nunca se aplican sólo en el lado del cliente, sino que también en los respectivos servidores remotos.	✓	✓
1.3	Se definió una arquitectura de alto nivel para la aplicación y los servicios y se incluyeron controles de seguridad en la misma.	✓	✓
1.4	Se identificó claramente la información considerada sensible en el contexto de la aplicación móvil.	✓	✓
1.5	Todos los componentes de la aplicación están definidos en términos de la lógica de negocio o las funciones de seguridad que proveen.		✓
1.6	Se realizó un modelado de amenazas para la aplicación móvil y los servicios en el que se definieron las mismas y sus contramedidas.		✓
1.7	La implementación de los controles de seguridad se encuentra centralizada.		✓
1.8	Existe una política explícita para el manejo de las claves criptográficas (si se usan) y se refuerza su ciclo de vida. Idealmente siguiendo un estándar del manejo de claves como el NIST SP 800-57.		✓
1.9	Existe un mecanismo para imponer las actualizaciones de la aplicación móvil.		✓
1.10	Se realizan tareas de seguridad en todo el ciclo de vida de la aplicación.		✓

- V2 - Requerimientos en el Almacenamiento de datos y la Privacidad

#	Descripción	L1	L2
2.1	Las funcionalidades de almacenamiento de credenciales del sistema son utilizadas para almacenar la información sensible, como credenciales del usuario y claves criptográficas.	✓	✓
2.2	No se escribe información sensible en los registros de la aplicación.	✓	✓
2.3	No se comparte información sensible con servicios externos salvo que sea una necesidad de la arquitectura.	✓	✓
2.4	Se desactiva el caché del teclado en los campos de texto donde se maneja información sensible.	✓	✓
2.5	Se desactiva el portapapeles en los campos de texto donde se maneja información sensible.	✓	✓
2.6	No se expone información sensible mediante mecanismos entre procesos (IPC).	✓	✓
2.7	No se expone información sensible como contraseñas y números de tarjetas de crédito a través de la interfaz o capturas de pantalla.	✓	✓
2.8	No se incluye información sensible en los respaldos generados por el sistema operativo.		✓
2.9	La aplicación remueve la información sensible de la vista cuando la aplicación pasa a un segundo plano.		✓
2.10	La aplicación no conserva la información sensible en memoria más de lo necesario y la memoria es limpiada luego de su uso.		✓
2.11	La aplicación obliga a que exista una política mínima de seguridad en el dispositivo, como que el usuario deba configurar un código de acceso.		✓
2.12	La aplicación educa al usuario acerca de los tipos de información personal que procesa y de las mejores prácticas en seguridad que el usuario debería seguir al utilizar la aplicación.		✓

- V3 - Requerimientos de Criptografía

#	Descripción	L1	L2
3.1	La aplicación no depende de únicamente de criptografía simétrica con "claves a fuego".	✓	✓
3.2	La aplicación utiliza implementaciones de criptografía probadas.	✓	✓
3.3	La aplicación utiliza primitivas de seguridad que son apropiadas para el caso particular y su configuración y sus parámetros siguen las mejores prácticas de la industria.	✓	✓
3.4	La aplicación no utiliza protocolos o algoritmos criptográficos que son considerados deprecados para aspectos de seguridad.	✓	✓
3.5	La aplicación no reutiliza una misma clave criptográfica para varios propósitos.	✓	✓
3.6	Los valores aleatorios son generados utilizando un generador de números suficientemente aleatorios.	✓	✓

- V4 - Requerimientos de Autenticación y Manejo de Sesiones

#	Descripción	L1	L2
4.1	Si la aplicación provee acceso a un servicio remoto, un mecanismo aceptable de autenticación como usuario y contraseña es realizado en el servidor remoto.	✓	✓
4.2	Si se utiliza la gestión de sesión por estado, el servidor remoto usa tokens de acceso randómicos para autenticar los pedidos del cliente sin requerir el envío de las credenciales del usuario en cada uno.	✓	✓
4.3	Si se utiliza la autenticación basada en tokens sin estado, el servidor proporciona un token que se ha firmado utilizando un algoritmo seguro.	✓	✓
4.4	Cuando el usuario se desloguea se termina la sesión también en el servidor.	✓	✓
4.5	Existe una política de contraseñas y es aplicada en el servidor.	✓	✓
4.6	El servidor implementa mecanismos, cuando credenciales de autenticación son ingresadas una cantidad excesiva de veces.	✓	✓
4.7	La autenticación biométrica, si hay, no está atada a un evento (usando una api que simplemente retorna "true" o "false"). Sino que está basado en el desbloqueo del keychain (iOS) o un keystore (Android).		✓
4.8	Las sesiones y los tokens de acceso expiran luego de un tiempo predefinido de inactividad.		✓
4.9	Existe un mecanismo de segundo factor de autenticación (2FA) en el servidor y es aplicado consistentemente.		✓
4.10	Para realizar transacciones o acciones que manejan información sensible se requiere una re-autenticación.		✓
4.11	La aplicación informa al usuario acerca de los accesos a su cuenta. El usuario es capaz de ver una lista de los dispositivos conectados y bloquear el acceso desde ciertos dispositivos.		✓

- V5 - Requerimientos de Comunicación a través de la red

#	Descripción	L1	L2
5.1	La información es enviada cifrada utilizando TLS. El canal seguro es usado consistentemente en la aplicación.	✓	✓
5.2	Las configuraciones del protocolo TLS siguen las mejores prácticas o tan cerca posible mientras que el sistema operativo del dispositivo lo permite.	✓	✓
5.3	La aplicación verifica el certificado X.509 del servidor al establecer el canal seguro y solo se aceptan certificados firmados por una CA válida.	✓	✓
5.4	La aplicación utiliza su propio almacén de certificados o realiza una fijación del certificado o la clave pública del servidor y no establece una conexión con servidores que ofrecen otros certificados o clave por más que estén firmados por una CA confiable.		✓
5.5	La aplicación no depende de un único canal de comunicaciones inseguro (email o SMS) para operaciones críticas como registros o recuperación de cuentas.		✓
5.6	La aplicación sólo depende de bibliotecas de conectividad y seguridad actualizadas.		✓

- V6 - Requerimientos de Interacción con la Plataforma

#	Descripción	L1	L2
6.1	La aplicación requiere la mínima cantidad de permisos.	✓	✓
6.2	Toda entrada del usuario y fuentes externas es validada y si es necesario sanitizada. Esto incluye información recibida por la UI, y mecanismo IPC como los intents, URLs y fuentes de la red.	✓	✓
6.3	La aplicación no exporta funcionalidades sensibles vía esquemas de URL, salvo que dichos mecanismos estén debidamente protegidos.	✓	✓
6.4	La aplicación no exporta funcionalidades sensibles a través de mecanismos IPC salvo que los mecanismos estén debidamente protegidos.	✓	✓
6.5	JavaScript se encuentra deshabilitado en los WebViews salvo que sea necesario.	✓	✓
6.6	Los WebViews se encuentran configurados para permitir el mínimo de los manejadores (idealmente, solo https). Manejadores peligrosos como file, tel y app-id se encuentran deshabilitados.	✓	✓
6.7	Si objetos nativos son expuestos en WebViews, verificar que solo se cargan JavaScripts contenidos del paquete de la aplicación.	✓	✓
6.8	Serialización de objetos, si se realiza, se implementa utilizando API seguras.	✓	✓

- V7 - Requerimientos de Calidad de Código y Configuración del Compilador

#	Descripción	L1	L2
7.1	La aplicación es firmada y provista con un certificado válido.	✓	✓
7.2	La aplicación fue liberada en modo release y con las configuraciones apropiadas para el mismo (ej. non-debuggable).	✓	✓
7.3	Los símbolos de debug fueron removidos de los binarios nativos.	✓	✓
7.4	La aplicación no contiene código de prueba y no realiza log de errores o mensajes de debug.	✓	✓
7.5	Todos los componentes de terceros se encuentran identificados y revisados por vulnerabilidades conocidas.	✓	✓
7.6	La aplicación captura y maneja debidamente las posibles excepciones.	✓	✓
7.7	Los controles de seguridad deniegan el acceso por defecto.	✓	✓
7.8	En código no administrado, la memoria es pedida, usada y liberada de manera correcta.	✓	✓
7.9	Funcionalidades de seguridad gratuitas se encuentran activadas.	✓	✓

- V8 - Requerimientos de Resistencia ante la Ingeniería Inversa

#	Descripción	R
8.1	La aplicación detecta y responde a la presencia de un dispositivo rooteado, ya sea alertando al usuario o finalizando la ejecución de la aplicación.	✓
8.2	La aplicación previene el debugging o detecta y responde al debugging de la aplicación. Se deben cubrir todos los protocolos.	✓
8.3	La aplicación detecta y responde a modificaciones de ejecutables y datos críticos de la propia aplicación.	✓
8.4	La aplicación detecta la presencia de las herramientas de ingeniería reversa o frameworks mas utilizados.	✓
8.5	La aplicación detecta y responde al ser ejecutada en un emulador.	✓
8.6	La aplicación detecta y responde ante modificaciones de código o datos en su propio espacio de memoria.	✓
8.7	La aplicación implementa múltiples mecanismos de detección para los puntos del 8.1 al 8.6. Nótese que a mayor cantidad y diversidad de mecanismos usados, mayor la resistencia.	✓
8.8	Los mecanismos de detección disparan distintos tipos de respuestas, incluyendo respuestas retardadas y silenciosas.	✓
8.9	La ofuscación es aplicada a las defensas del programa, lo que a su vez impide la des-ofuscación mediante el análisis dinámico.	✓
8.10	La aplicación implementa un "enlace al dispositivo" utilizando una huella del dispositivo derivado de varias propiedades únicas al mismo.	✓
8.11	Todos los archivos ejecutables y bibliotecas correspondientes a la aplicación se encuentran cifrados, o bien los segmentos importantes de código se encuentran cifrados o empaquetados. De este modo el análisis estático trivial no revela código importante o datos.	✓
8.12	Si el objetivo de la ofuscación es proteger el código propietario, se utiliza un esquema de ofuscación que es apropiado tanto para la tarea particular como robusto contra los métodos de des-ofuscación manual y automatizada, considerando la investigación publicada actualmente. La eficacia del esquema de confusión debe verificarse mediante pruebas manuales. Tenga en cuenta que las características de aislamiento basadas en hardware son preferibles a la ofuscación siempre que sea posible.	✓