



6. – Desarrollo móvil y comunicaciones seguras

Documento original de



Ciberseguridad en Dispositivos móviles
DISCA – ETS de Ingeniería informática (UPV)

Objetivos

- Aprender como proteger una aplicación
- Ser capaces de solucionar las vulnerabilidades que se identificaron mediante el análisis estático y dinámico en las aplicaciones analizadas en la unidad anterior.

Índice

1. Principios básicos de seguridad en el desarrollo
2. El ciclo de desarrollo de software seguro
3. El S-SDLC en entornos de desarrollo ágiles
4. Buenas prácticas en el desarrollo seguro
 1. Validación de entrada.
 2. Codificación de los elementos de salida.
 3. Autenticación y gestión de contraseñas.
 4. Gestión de sesiones.
 5. Control de acceso.
 6. Gestión de errores.
7. Protección de datos.
8. Seguridad en las comunicaciones.
9. Configuración del sistema.
10. Seguridad en la base de datos.
11. Gestión de memoria.
12. Otras consideraciones generales.
5. Buenas prácticas en el desarrollo móvil
 1. Protección de la aplicación
 2. Manejo de datos sensibles
 3. Logs y filtrado de información
 4. Componentes HTML en aplicaciones móviles
 5. Comunicaciones entre aplicaciones
 6. Autenticación en aplicaciones

1. Principios de seguridad en el desarrollo

Introducción

- Las vulnerabilidades y problemas de seguridad pueden afectar a un producto *software* durante todo su desarrollo:
 - No identificando requisitos de seguridad durante la fase de análisis.
 - Creando diseños con fallos de seguridad.
 - Generando vulnerabilidades durante la etapa de implementación.
 - Desplegando el *software* de forma inadecuada.
 - No respondiendo de forma oportuna a los incidentes de seguridad que ocurran.
- Estos problemas, afectan directamente al *software* que se ha desarrollado y la información almacenada, pero también pueden afectar a:
 - Otras aplicaciones que se ejecutan en el entorno compartido.
 - El sistema del usuario (incluidos los dispositivos móviles).
 - Otros sistemas que interaccionan con el *software* a desarrollar.

2. El ciclo de desarrollo de software seguro

Introducción

- El Ciclo de Desarrollo de *Software* Seguro, **S-SDLC** (*Secure Software Development Life Cycle*)
 - Es un proceso de desarrollo de *software* que incorpora la seguridad como un elemento transversal durante todo el proceso de desarrollo, denominándose “*Security By Design*”.
 - Tiene en cuenta desde el inicio del desarrollo de *software*, todos los aspectos de seguridad que puedan estar involucrados en el mismo:
 - Permite detectar vulnerabilidades durante etapas tempranas en el desarrollo.
 - Ahorro de costes en vulnerabilidades detectadas en sistemas en producción.

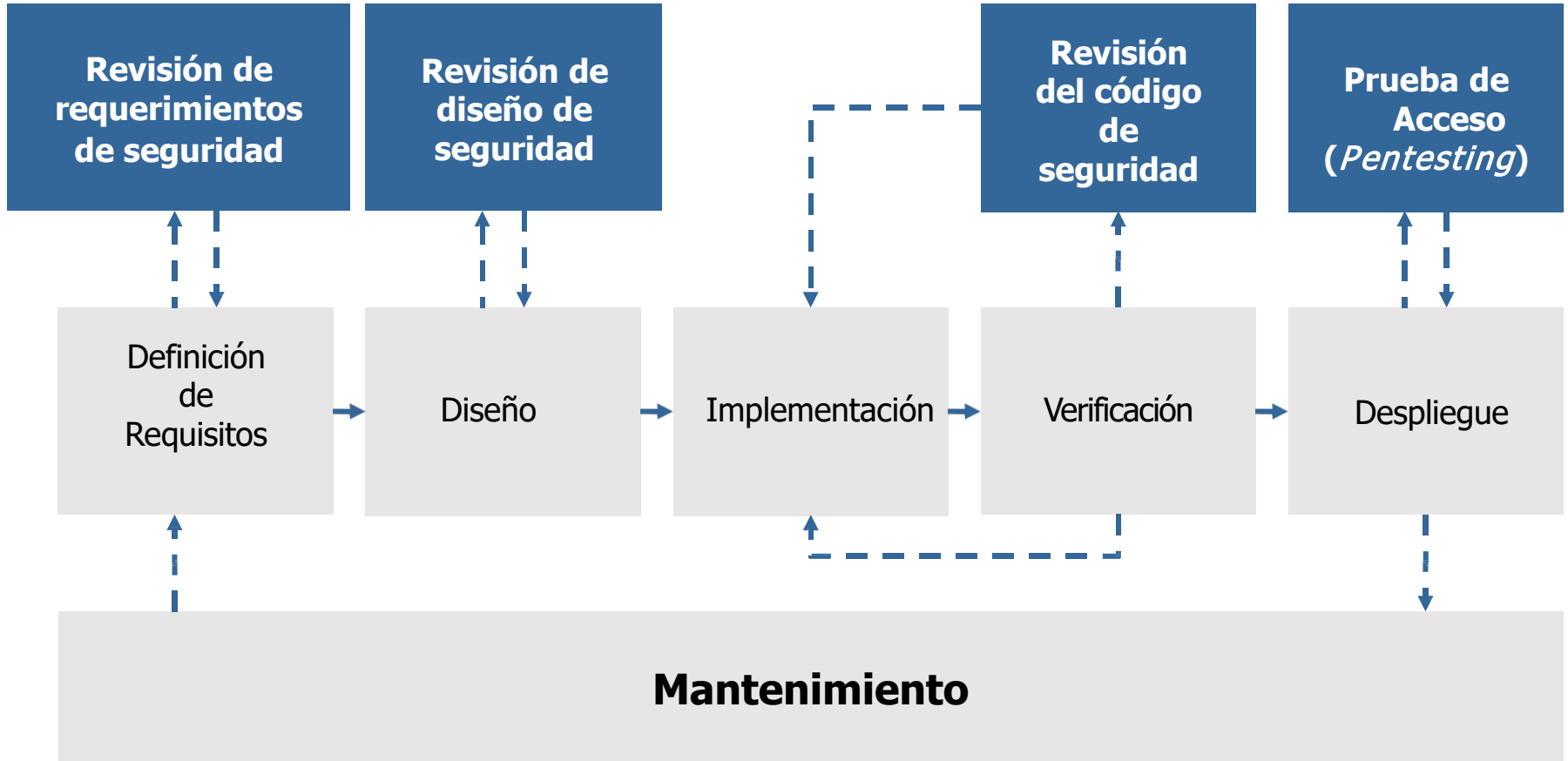
2. El ciclo de desarrollo de software seguro

Aproximaciones al S-SDLC

- Existen diferentes aproximaciones al S-SDLC:
 - OWASP CLASP (Comprehensive, Lightweight Application Security Process):
 - https://wiki.owasp.org/index.php/OWASP_Secure_Software_Development_Lifecycle_Project
 - Microsoft Secure Development Lifecycle:
 - Desarrollada por Microsoft pero aplicable a cualquier desarrollo.
 - <https://www.microsoft.com/en-us/sdl/>
 - Digital's Security Touchpoints:
 - Desarrollados por Gary McGraw.
 - <http://www.swsec.com/resources/touchpoints/>
 - NIST 800-64:
 - Conjunto de consideraciones de seguridad a tener en cuenta durante el SDLC propuestas por el NIST.
 - <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf>
- En general, todos los modelos incorporan una serie de actividades de seguridad al ciclo de vida del desarrollo.

2. El ciclo de desarrollo de software seguro

Fases



2. El ciclo de desarrollo de software seguro

Definición de Requisitos

- Además de los requisitos funcionales de la aplicación se deben tener en cuenta los requisitos de seguridad, privacidad y regulatorios sobre la protección de datos:
 - Se deben definir un conjunto mínimo de requisitos de seguridad.
 - Se deben implementar las medidas necesarias para poder trazar todo su desarrollo durante el SDLC.
- Se debe definir un conjunto de métricas de seguridad que se deben mantener durante las diferentes fases del desarrollo:
 - Establecer niveles de severidad para vulnerabilidades.
 - Indicar por cada fase del desarrollo los niveles mínimos aceptables.
 - Ej.: no se puede pasar a la fase de lanzamiento con alguna vulnerabilidad crítica.

2. El ciclo de desarrollo de software seguro

Definición de Requisitos:

Áreas clave en seguridad de aplicaciones móviles

- Almacenamiento local de datos (credenciales de usuario e información privada) :
 - almacenamiento local o la comunicación entre procesos (IPC) de forma incorrecta podría exponer datos confidenciales a otras aplicaciones que se ejecutan en el mismo dispositivo.
 - filtrar datos involuntariamente a la nube de almacenamiento, copias de seguridad o el caché del teclado.
 - los dispositivos móviles se pueden perder o robar más fácilmente en comparación con otros tipos de dispositivos, por lo que es más probable que un individuo pueda obtener acceso físico al dispositivo, lo que hace que sea más fácil recuperar los datos

2. El ciclo de desarrollo de software seguro

Definición de Requisitos:

Áreas clave en seguridad de aplicaciones móviles

- Comunicación con puntos finales de confianza:
 - Conexión a una variedad de redes, incluidas las redes WiFi públicas
 - Prevenir ataques de red
 - crucial mantener la confidencialidad e integridad de la información intercambiada entre la aplicación móvil y los puntos finales de servicio remoto.
 - requisito básico: las aplicaciones móviles deben configurar un canal de cifrado seguro para la comunicación de red utilizando el protocolo TLS con la configuración adecuada.

2. El ciclo de desarrollo de software seguro

Definición de Requisitos:

Áreas clave en seguridad de aplicaciones móviles

- Autenticación y autorización:
 - Incorporar marcos de autorización (como OAuth2) que delegan autenticación a un servicio separado o externalizar el proceso de autenticación a un proveedor de autenticación.
 - Utilizando OAuth2 permite que la lógica de autenticación del lado del cliente se externalice a otras aplicaciones en el mismo dispositivo (por ejemplo, el sistema navegador).
 - Los probadores de seguridad deben conocer las ventajas y desventajas de diferentes autorizaciones posibles marcos y arquitecturas.

2. El ciclo de desarrollo de software seguro

Definición de Requisitos:

Áreas clave en seguridad de aplicaciones móviles

- Interacción con la plataforma móvil:
 - Los sistemas operativos móviles implementan sistemas de permisos de aplicaciones que regulan el acceso a API específicas.
 - Ofrecen más servicios de comunicación entre procesos (IPC) (Android) o menos ricos (iOS) que permiten a las aplicaciones intercambiar señales y datos.
 - Estas características específicas de la plataforma vienen con su propio conjunto de trampas.
 - Por ejemplo, si las API de IPC se usan incorrectamente, los datos confidenciales o la funcionalidad pueden estar expuestos involuntariamente a otras aplicaciones que se ejecutan en el dispositivo¹.
- En <https://github.com/owasp/owasp-masvs> podemos ver una propuesta de listas de requisitos por áreas.

1. <https://chromium.googlesource.com/chromium/src.git/+master/docs/security/android-ipc.md>

2. El ciclo de desarrollo de software seguro

Diseño

- Durante esta fase se deben definir las soluciones de seguridad que cubrirán los requisitos de seguridad descritos en la fase anterior.
- Además, durante la fase de diseño se deberán especificar los detalles funcionales que no hayan sido especificados durante la fase de requisitos.
 - Ejemplo: algoritmos criptográficos a utilizar.

2. El ciclo de desarrollo de software seguro

Diseño – Principios de diseño seguro I

- **Defensa en profundidad:**
 - Consiste en crear diferentes capas de seguridad, de tal forma que si una falla, el sistema no se vea comprometido.
 - Requiere diseñar distintas estrategias de defensa para una misma amenaza.
- **Fallo seguro:**
 - Consiste en que todos los fallos lleven a un estado del sistema que se considere seguro (sin pérdida de confidencialidad, integridad y disponibilidad).
- **Mínimo Privilegio:**
 - Cada usuario o proceso debe poseer sólo, los mínimos privilegios posibles para llevar a cabo las tareas que le son permitidas.
 - Los privilegios se deben otorgar por el mínimo tiempo posible.
- **Separación de privilegios:**
 - Se asignarán privilegios a partes de la aplicación únicamente si son estrictamente necesarios¹

1. <https://developer.android.com/guide/topics/permissions/overview?hl=es-419>

2. El ciclo de desarrollo de software seguro

Diseño – Principios de diseño seguro II

- **Simplicidad:**
 - A mismo nivel de seguridad es preferible, por norma general, utilizar las soluciones menos complejas.
 - A mayor simplicidad, menor superficie de ataque en general.
- **Supervisión:**
 - Se debe comprobar durante la ejecución de cualquier tarea (acceso, escritura, modificación) que el usuario o proceso que la ejecuta está autorizado para ello.
 - Para evitar problemas de sincronización se recomienda no utilizar cachés de autorización.
- **Diseño abierto:**
 - Los detalles del diseño del sistema deben ser abiertos, evitando los casos de seguridad por oscuridad.
 - Este principio ayuda a crear sistemas seguros desde el diseño.
 - Este principio asegura que la publicación o revisión del diseño no impliquen de forma directa un incidente grave de seguridad.

2. El ciclo de desarrollo de software seguro

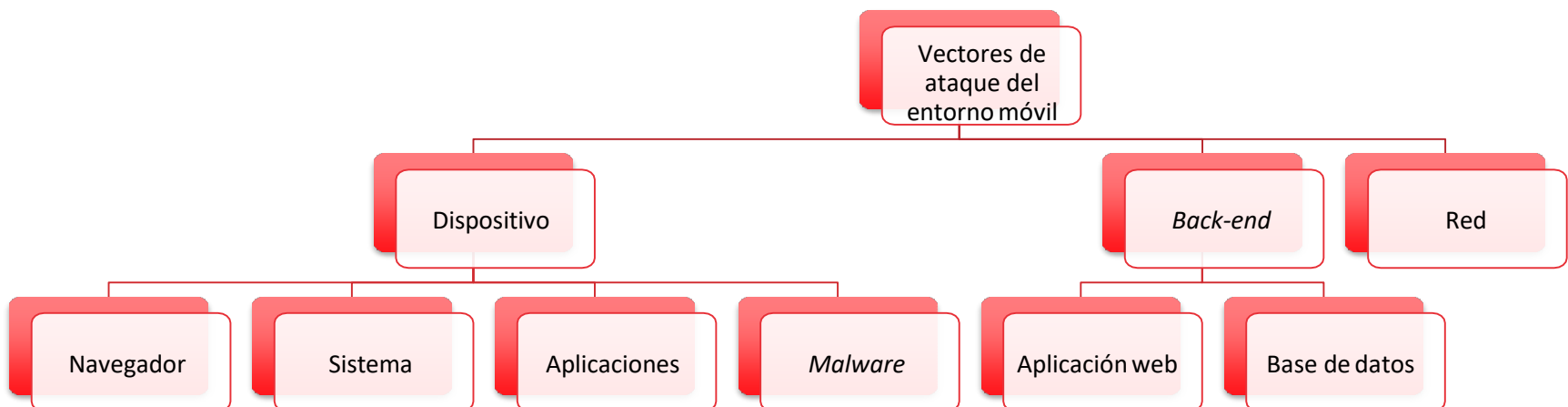
Diseño – Principios de diseño seguro III

- **Mínimo en común:**
 - Este principio desaconseja la utilización de un mismo mecanismo, aunque sea común a varios procesos o usuarios, si estos tienen diferentes niveles de privilegio.
- **Aceptabilidad:**
 - Los mecanismos de seguridad del sistema se deben diseñar teniendo en cuenta la aceptabilidad por parte de sus usuarios.
 - Si los usuarios tienen dificultades en usar las características de seguridad, buscarán mecanismos para saltárselas, haciéndolas inútiles.
- **Punto más débil:**
 - La seguridad de todo el sistema dependerá de su punto más débil.
- **Reutilización:**
 - Es preferible la utilización de componentes ya existentes y verificados que la creación de nuevos que puedan incrementar el riesgo de vulnerabilidades y superficie de ataque.

2. El ciclo de desarrollo de software seguro

Diseño – Superficie de ataque

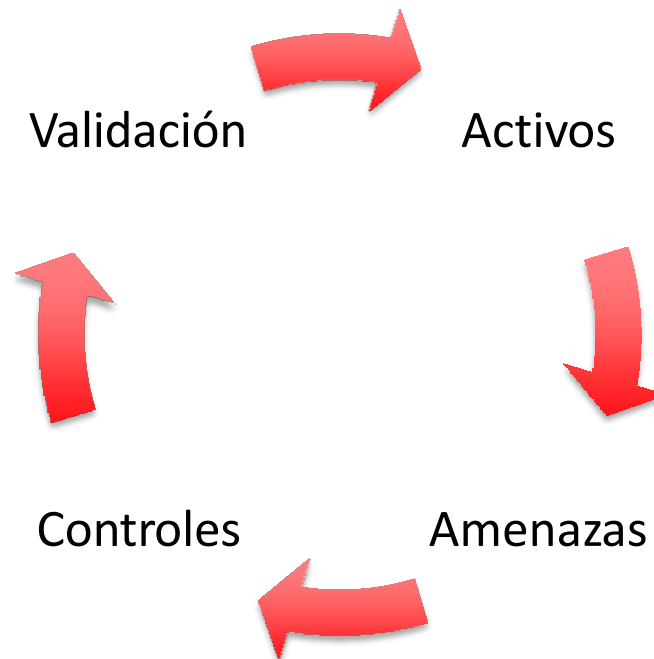
- Consiste en especificar, de manera estructurada, los puntos de entrada al sistema. Esta tarea debe ser realizada por el diseñador.
- Los puntos de entrada de la aplicación pueden categorizarse en:
 - Red.
 - Sistema de ficheros.
 - Usuario.
- Para cada punto de entrada se deben identificar los:
 - Recursos a los que se puede acceder a través del mismo.
 - Roles que tienen acceso al punto de acceso.
- Permite identificar fugas de recursos a roles que no deberían tener los privilegios necesarios



2. El ciclo de desarrollo de software seguro

Diseño – Modelado de amenazas

- Catalogar y evaluar los diferentes riesgos y amenazas a los que puede estar sometido un sistema.
 - Será la primera tarea realizada por cualquier atacante.
 - No efectuarlo reduce nuestra capacidad de protección.



2. El ciclo de desarrollo de software seguro

Diseño – Modelado de amenazas

- Por cada activo o capacidad identificar las amenazas potenciales.
 - Esta tarea requiere de cierta creatividad por parte del analista.
- Por cada amenaza evaluar el riesgo existente:
 - Utilizar árboles de amenazas que describan los distintos pasos que debe llevar el atacante para materializarla.
 - Medir factores como: impacto, reproducibilidad, explotabilidad y usuarios afectados.
- Por cada amenaza identificar los controles que sea factible implementar para su mitigación.
- Al final del proceso, deberían quedar cubiertos el mayor número de amenazas posibles.

2. El ciclo de desarrollo de software seguro

STRIDE

- STRIDE: Ayuda a identificar amenazas en los componentes de un sistema
- Es un modelo de amenazas que agrupa las mismas en 6 categorías:
 - **Spoofing** (Suplantación): suplantar la identidad de un sistema o usuario.
 - Ejemplo: intentar actuar como administrador del sistema.
 - **Tampering** (Manipulación): modificar los datos o el código.
 - Ejemplo: modificar el código fuente de la aplicación para desactivar protecciones.
 - **Repudiation** (Repudio): denegar que se ha realizado una acción específica.
 - Ejemplo: “Yo no envíe ese mensaje”.
 - **Information Disclosure** (Fuga de información): acceso a una pieza de información por parte de una entidad sin credenciales para ello.
 - Ejemplo: información personal filtrada al público.
 - **Denial of Service** (Denegación de servicio): bloquear o degradar un servicio.
 - Ejemplo: bloqueo de servidores por un conjunto alto de peticiones.
 - **Elevation of Privilege** (Elevación de privilegios): incrementar las capacidades sin la autorización apropiada.
 - Ejemplo: paso de usuario a administrador.

2. El ciclo de desarrollo de software seguro

STRIDE - Controles

- Frente a este tipo de amenazas se establecen los siguientes controles sobre el desarrollo, para mitigar el posible impacto de una brecha de seguridad.

Amenaza	Control/Servicio de seguridad
Suplantación	Autenticación
Manipulación	Controles de Integridad
Repudio	Métodos de no repudio
Fugas de información	Mecanismos de confidencialidad
Denegación de servicio	Disponibilidad
Elevación de privilegios	Autorización

2. El ciclo de desarrollo de software seguro

Implementación

- Las actividades del S-SDLC tienen como objetivo ayudar a los desarrolladores a implementar las funcionalidades requeridas de la forma más segura posible.
- Se deben considerar las siguientes actividades:
 - Configuración segura del entorno de desarrollo.
 - Revisión de código fuente de la aplicación.
 - Revisión de elementos de terceros.

2. El ciclo de desarrollo de software seguro

Implementación – Entorno de desarrollo seguro

- Se debe definir una configuración oficial para el entorno de desarrollo a utilizar durante la implementación del producto *software*.
- La configuración debe especificar:
 - Sistema o sistemas operativos válidos (con versiones).
 - Herramientas soportadas para el desarrollo (con versiones):
 - IDE.
 - Sistemas de control de versiones.
 - Restricciones para el acceso remoto.
- Se deben incluir los mecanismos necesarios para aplicar y restringir la configuración de las estaciones de trabajo a la aceptada:
 - Restricciones a cuentas de usuario.
 - Entornos pre-instalados.

2. El ciclo de desarrollo de software seguro

Implementación – Entorno de desarrollo seguro

- Un ejemplo muy claro de los problemas que puede ocasionar no seguir estas directrices es XcodeGhost.
- Un conjunto de ciber-delincuentes modificó y puso a disposición pública (en un servidor chino) una versión del IDE XCode para iOS y Mac OS.
- La modificación inyectaba código malicioso en la versión compilada de aplicaciones de iOS.
- Debido a la lentitud de la descarga desde servidores de Estados Unidos, muchos desarrolladores chinos optaron por descargar la versión disponible en los servidores de su país.
- Al utilizar la versión modificada para desarrollar sus aplicaciones, sin tener conocimiento de ello, muchos desarrolladores crearon aplicaciones maliciosas que fueron publicadas en la App Store.
 - <https://developer.apple.com/news/?id=09222015a>
 - https://www.incibe.es/technologyForecastingSearch/CERT/Bitacora_de_ciberseguridad/ataque_appstore

2. El ciclo de desarrollo de software seguro

Implementación – Revisión de código I

- La revisión del código fuente de la aplicación permite identificar vulnerabilidades que se introducen durante la fase de implementación.
- Las herramientas automáticas de análisis estático como las estudiadas durante la unidad anterior facilitan esta tarea, pero no son capaces de encontrar ciertas vulnerabilidades que necesitan una revisión manual.
- La revisión del código es una tarea adicional a la ejecución de los diferentes pruebas unitarias y de integración que se definen dentro del SDCL tradicional. En ningún momento es equivalente, ni debe sustituirlos.
- La revisión de código fuente se puede realizar:
 - De forma ligera durante el proceso de implementación.
 - De manera formal una vez se ha finalizado una parte del proceso de implementación.

2. El ciclo de desarrollo de software seguro

Implementación – Revisión de código II

- En cuanto a las revisiones ligeras del código fuente, se pueden realizar de las siguientes maneras:
 - **Técnicas de *pair-programming*:** dos personas se encargan de desarrollar código juntos en la misma máquina, supervisando el código escrito mutuamente.
 - **Revisión externa:** el autor explica el código a otro desarrollador que se encarga de la verificación del mismo.
 - **Revisión asistida:** se utilizan herramientas semi-automáticas que permiten, durante la programación, identificar problemas en el código.
 - **Revisión por “commit”:** cada vez que se efectúa un “commit” en el sistema de control de versiones se lanza una herramienta que:
 - Envía el elemento por correo a los revisores de forma automática.
 - Realiza un análisis del mismo a través una herramienta de análisis integrada con el control de versiones.
 - Ejemplo: <https://www.pullreview.com/> o <https://codeclimate.com/>

2. El ciclo de desarrollo de software seguro

Implementación – Elementos de terceros

- Durante la fase de implementación, es posible que sea necesario utilizar herramientas y librerías de terceros.
- Los controles que se realizan sobre nuestro propio código deben ser también implementados sobre este tipo de librerías.
- En concreto se deben realizar las siguientes tareas:
 - Si el código fuente está disponible, someterlo a un proceso de análisis de código fuente como el descrito anteriormente.
 - Revisar las vulnerabilidades o posibles problemas de seguridad asociados a la versión de la librería que estamos utilizando:
 - Almacenamiento seguro.
 - Comunicaciones cifradas.
 - Validación de datos de entrada.
 - Problemas de configuración o exposición de datos por defecto.
 - Comprobar la utilización de funciones o elementos que han sido declarados como obsoletos (*deprecated*) por los desarrolladores.

2. El ciclo de desarrollo de software seguro

Verificación

- En el SDLC tradicional esta fase incluye todas las actividades encaminadas a comprobar que el producto *software* funciona como está descrito en los requerimientos.
- Las tareas del S-SDLC en la etapa de verificación permiten realizar las comprobaciones de seguridad directamente sobre elementos de *software* que se han implementado en la etapa anterior:
 - **Análisis dinámico:** estudiado en el tema 5. Permite verificar las propiedades de seguridad del sistema y su comportamiento mediante su ejecución.
 - **Fuzzing:** es parte del análisis dinámico. Se comprueba si los controles implementados en los puntos de entrada en el sistema controlan correctamente las diferentes entradas posibles.
 - **Revisión de la superficie de ataque:** una vez terminado el código se puede verificar que la superficie de ataque real se ajusta a la identificada en las etapas anteriores del S-SDLC.

2. El ciclo de desarrollo de software seguro

Despliegue I

- Durante esta fase se prepara el producto *software* para su lanzamiento.
- En lo relativo al S-SDLC, esta fase incluye actividades para cubrir los aspectos de seguridad del producto más allá de su fecha de lanzamiento.
- Plan de respuesta ante incidentes:
 - Permite mitigar el alcance de incidentes de seguridad, reducir el riesgo y los costes de un incidente.
 - Debe identificar de forma clara los eventos que considerar para declarar la existencia de un incidente de seguridad.
 - Por cada incidente se identificarán de forma detallada las acciones a tomar.
 - Se deben incluir los roles de cada miembro del equipo de respuesta y su información de contacto.
 - Esta tarea es fundamental para poder responder con celeridad ante cualquier incidente de seguridad.
 - El plan de respuesta no es un documento estático. Evoluciona según se modifica el sistema o aparecen nuevas amenazas no tenidas en cuenta.

2. El ciclo de desarrollo de software seguro


Despliegue II

- **Revisión de seguridad final:**
 - Antes del lanzamiento, se debe verificar que todas las tareas de seguridad planeadas para llevar a cabo el S-SDLC se han completado.
 - Además, conviene realizar una revisión de cada una de las tareas realizadas para asegurar que no se ha cometido ningún fallo durante las mismas.
- **Certificación:**
 - Permite asegurar que el producto cumple con ciertas normativas/regulaciones de seguridad.
- **Archivado:**
 - Consiste en guardar una copia de todos los elementos envueltos en la versión del *software* que va a ser lanzada.
 - Será uno de los elementos a considerar en caso de incidente de seguridad.
- Tareas como análisis dinámico, *fuzzing* y otras revisiones de seguridad se siguen ejecutando durante esta etapa.

2. El ciclo de desarrollo de software seguro

Respuesta

- Esta fase sólo se activa en respuesta a sucesos que hayan sido declarados como generadores de un incidente en el plan de respuesta ante incidentes.
- Una vez activado se deben seguir las directrices marcadas por el plan, incluidos:
 - Personal al que notificar y orden de notificación.
 - Captura de datos para el análisis posterior del incidente.
 - Ejecución de tareas de mitigación de la amenaza.
 - Ejecución de tareas para el restablecimiento del servicio (en caso de que sea necesario).

A close-up photograph of a person's hands with red nail polish typing on a laptop keyboard. The laptop is silver and black. In the foreground, a pair of black-rimmed glasses and an orange pen with a silver tip are resting on a dark surface. The background is blurred, showing white flowers in a vase. A semi-transparent grey rectangular box is overlaid on the center of the image, containing the title text in white.

El S-SDLC en entornos de desarrollo ágiles

El S-SDLC en entornos de desarrollo ágiles

- Las metodologías ágiles son un proceso alternativo a las metodologías tradicionales que se basan en el desarrollo a través de iteraciones más pequeñas para incorporar funcionalidad (<http://agilemethodology.org>).
- Las tareas y actividades que establece habitualmente el S-SDLC, asumen el ciclo de vida tradicional (cascada) durante el desarrollo del *software*.
- Generalmente, los sistemas y productos desarrollados para entornos móviles son desarrollados mediante metodologías ágiles.
- La ejecución del S-SDLC y tal como lo hemos visto, requiere de adaptaciones para poder aplicarse sobre metodologías ágiles.
- En estos casos, las actividades del S-SDLC se ejecutan con tres frecuencias diferentes:
 - Por **sprint**: aquellas actividades que se deben ejecutar por cada *release* que se complete.
 - Por **bucket**: aquellas actividades que se deben ejecutar por cada conjunto de *sprints*.
 - Por **proyecto**: las actividades que se ejecutan una sola vez en todo el proyecto.

El S-SDLC en entornos de desarrollo ágiles

- **Actividades por *sprint*:**
 - Modelado de amenazas de la funcionalidad incluida en el *sprint*.
 - Todas las relacionadas con la fase de implementación del S-SDLC.
 - Revisión de seguridad final por *sprint*.
 - Certificación y archivado.
- **Actividades por *bucket*:**
 - Definir las métricas de seguridad con las que se evaluará el *bucket*.
 - Tareas de análisis dinámico, *fuzzing* y revisión de la superficie de ataque.
- **Actividades por proyecto:**
 - Definir los requisitos de seguridad.
 - Análisis de riesgos.
 - Definir la superficie de ataque.
 - Crear un plan de respuesta ante incidentes.

A hand is visible in the lower-left corner, pointing towards the center of the image. The background is a dark, pixelated digital screen with green and blue light effects. Faintly visible in the background are the words 'infected' and 'payload' in a digital font, along with some code snippets like '<include:payload'.

Buenas prácticas en el desarrollo seguro

Buenas prácticas en el desarrollo seguro

Listado

- El conjunto de prácticas que se estudiarán en esta sección incluye:
 - Validación de entrada.
 - Codificación de los elementos de salida.
 - Autenticación y gestión de contraseñas.
 - Gestión de sesiones.
 - Control de acceso.
 - Gestión de errores.
 - Protección de datos.
 - Seguridad en las comunicaciones.
 - Configuración del sistema.
 - Seguridad en la base de datos.
 - Gestión de memoria.
 - Otras consideraciones generales.

Buenas prácticas en el desarrollo seguro

Validación de entrada I

- Toda entrada al sistema debe considerarse como maliciosa (*All input is evil*):
 - Campos de texto.
 - URL.
 - *Cookies* y otros campos HTTP.
- La validación de los datos de entrada debe llevarse a cabo siempre en un sistema que sea considerado fiable, generalmente el *back-end*.
 - El dispositivo móvil no se puede considerar como elemento confiable.
- Se recomienda:
 - Validar los rangos y longitud de los datos.
 - Utilizar listas blancas para comprobar que todos los elementos de una entrada son válidos.
 - Validar que los tipos de datos que se reciben concuerdan con los esperados:
 - Las cabeceras HTTP deben contener solo caracteres ASCII.
 - Si se espera una imagen en un formato específico comprobar que se recibe ese formato.
 - Los campos de texto y parámetros de la URL deben contener el tipo de dato que se espera en la aplicación.

Buenas prácticas en el desarrollo seguro

Codificación de los elementos de salida

- Al igual que la validación de entrada, la codificación de los datos de salida se debe efectuar en un sistema confiable como el *back-end* de la aplicación.
- Se debe evitar reinventar la rueda. Existen múltiples librerías y métodos de codificación de salida ampliamente testeados y aceptados por la comunidad:
 - En iOS se puede utilizar el método `stringByAddingPercentEncodingWithAllowedCharacters` de la clase `NSString`.
 - En Android se puede utilizar [URLEncoder](#) o [DatabaseUtils](#).
- Los datos de salida se deben codificar dependiendo del uso que se va a hacer de ellos en la aplicación:
 - En caso de que la salida vaya a ser interpretada por un navegador web, evita que se puedan generar elementos interpretables en HTML, CSS, Javascript etc.
 - Si la salida va a ser interpretada por otro sistema hay que evitar que se puedan formar o modificar los comandos a los mismos (SQL, XML, LDAP, etc.).

Buenas prácticas en el desarrollo seguro

Autenticación y gestión de contraseñas I

- Todas las páginas, excepto aquellas que se definan estrictamente como públicas, deben requerir autenticación por parte de los usuarios.
- Para la implementación de los controles de autenticación se identifican las siguientes recomendaciones:
 - Los controles de autenticación se deben llevar a cabo siempre en un sistema fiable (*back-end*).
 - Todos los controles de autenticación deben estar centralizados en un único módulo, incluidas aquellas librerías que puedan realizar llamadas a servicios de autenticación externos.
 - La lógica de autenticación no debe estar acoplada a la lógica del recurso al que se accede.
 - Las peticiones de autenticación se deben realizar siempre mediante conexiones HTTP POST cifradas convenientemente (SSL).

Buenas prácticas en el desarrollo seguro

Autenticación y gestión de contraseñas II

- Para el proceso de autenticación se recomiendan las siguientes prácticas:
 - La validación de los datos de autenticación se debe llevar a cabo sólo si se han introducido todos los datos necesarios para llevarla a cabo (usuario y contraseña).
 - En caso de que se produzca un fallo de autenticación, no se deben ofrecer detalles, ni visuales, ni en el código fuente utilizado, sobre el fallo concreto en la autenticación (contraseña errónea, usuario erróneo, etc.).
 - El proceso de autenticación debe fallar de forma segura.
 - Independientemente del método de acceso, el campo para la introducción de la contraseña no debería mostrar los elementos tecleados.

Buenas prácticas en el desarrollo seguro

Autenticación y gestión de contraseñas III

- Bajo ningún concepto se deben guardar las contraseñas de la aplicación en claro.
- Todas las contraseñas deben almacenarse mediante una función criptográficamente segura, utilizando un salt para dificultar los ataques de fuerza bruta mediante *rainbow tables*.
- La aplicación debe obligar a los usuarios a utilizar contraseñas con un mínimo de complejidad:
 - Longitud mínima de 8 caracteres, pero más son recomendados.
 - Caracteres alfanuméricos, signos de puntuación y números.
- Si la aplicación genera contraseñas por defecto, obligar al usuario a cambiarla en el primer acceso.
- Si se realizan varios intentos fallidos de acceso desactivar el mismo durante un periodo de tiempo que sea lo suficientemente largo como para evitar ataques de fuerza bruta, pero no para provocar denegación de servicio al usuario.

Buenas prácticas en el desarrollo seguro

Autenticación y gestión de contraseñas IV

- En cuanto al restablecimiento de contraseñas:
 - Siempre que sea posible, se debe evitar la utilización de preguntas de seguridad. En el caso de que sean necesarias, se deben evitar preguntas cuya respuesta sea predecible o común:
 - Ej. incorrecto: ¿Cuál es el nombre de tu primera mascota?
 - Ej. correcto: ¿Calle en la que creció tu madre?
 - Se debe comprobar que el correo al que se envía una solicitud de restablecimiento está registrado en el sistema.
- En caso de que el usuario quiera efectuar alguna operación crítica en el sistema como por ejemplo el propio cambio de contraseña, se deberá volver a autenticar al usuario.
- Si es posible, implementar un doble factor de autenticación mediante:
 - Contraseña + aplicación móvil que genere *passwords* de un solo uso (Google authenticator).
 - Contraseña + elemento biométrico.

Buenas prácticas en el desarrollo seguro

Gestión de sesiones

- Conviene utilizar, si ofrece las suficientes garantías, el control de sesiones que incorpora el servidor framework en el que se desarrolla la aplicación:
 - [iOS](#)
 - [Android](#)
 - [Java EE](#)
 - [.Net](#)
 - [Diango](#)
 - [Ruby on Rails](#)
- Los identificadores deben ser creados por un sistema confiable (generalmente el *back-end*) con librerías que aseguren que son lo suficientemente aleatorios.
- Cada re-autenticación debería generar un nuevo identificador de sesión y eliminar el anterior activo.
- El usuario debe tener la posibilidad de cerrar una sesión de forma sencilla.
- Toda sesión debería expirar tras un periodo mínimo de inactividad.
- Se debe evitar exponer la información sobre la sesión o cookies a terceros: registro de *logs*, utilización de parámetros GET, etc.
- Dependiendo de las limitaciones de presupuesto, se debe ofrecer un sistema al usuario que permita el control y cerrado de sesiones activas.

Buenas prácticas en el desarrollo seguro

Control de acceso

- Las decisiones de control de acceso deben ser tomadas en base a información que provenga de sistemas confiables.
- Al igual que con la validación de entrada, salida y autenticación, conviene que el sistema de control de acceso esté centralizado y separado del resto de la lógica, en un único elemento del sistema.
- El control de acceso se deber realizar para todas las peticiones, incluidas aquellas que se hagan mediante tecnologías como AJAX.
- Los usuarios que no están autorizados, no deben poder acceder a elementos como datos de la aplicación y servicios.

Buenas prácticas en el desarrollo seguro

Gestión de errores

- Ante la aparición de un error se debe evitar revelar información sensible como detalles del sistema, identificadores de sesión o información sobre cuentas.
- La aplicación debería manejar todos los errores y no depender nunca de los errores por defecto del sistema.
- Ante la aparición de un error, la política por defecto de cara a la tarea que se está realizando debe ser la denegación.
- Los *logs* deben registrar los sucesos relevantes en el sistema:
 - Fallos en la validación de entrada.
 - Intentos de autenticación fallidos.
 - Intentos de conexión con sesiones expiradas.
 - Cambios en la configuración de elementos críticos.
 - Excepciones en el sistema y otros errores ocurridos durante la ejecución.

Buenas prácticas en el desarrollo seguro

Protección de datos

- Las contraseñas, *tokens* de autenticación y otra información sensible deben almacenarse cifrados.
- Se debe evitar bajo todos los medios almacenar credenciales dentro del propio código fuente de la aplicación o en archivos de configuración. En el caso de utilizar repositorios abiertos como GitHub se podrían estar publicando las credenciales de acceso a los servicios.
- Configura el servidor de aplicaciones para que los ficheros del código fuente de la aplicación *back-end* no se puedan descargar.
- Elimina los ficheros de documentación y configuración que se instalan por defecto.

Buenas prácticas en el desarrollo seguro

Seguridad en las comunicaciones

- Dado que la mayoría de conexiones para acceder a los servicios de la aplicación incluirán tokens de autenticación, sesiones o información sensible, las conexiones entre los diferentes clientes y el servidor deben realizarse cifradas.
- Las aplicaciones deben verificar la validez del certificado que se les presenta e incluso utilizar técnicas de “*certificate pinning*” para mitigar ataques al sistema de PKI.
- Los recursos accesibles a través de conexiones seguras no deben estar disponibles a través de conexiones que no lo son (*downgrade* a conexiones sin cifrar).

Buenas prácticas en el desarrollo seguro

Configuración del sistema

- Asegurarse de que las versiones que se están ejecutando de los diferentes elementos de terceros que se requieren para la ejecución de la aplicación son las aprobadas durante el diseño.
- Es necesario también revisar que durante el proceso de desarrollo no se haya registrado ninguna vulnerabilidad que afecte a las versiones aprobadas. En ese caso se deberán revisar y escoger de nuevo.
- El sistema en producción no debe contener los ficheros de código fuente y recursos que hayan sido utilizados para efectuar los diferentes *tests* y verificaciones durante el proceso de desarrollo.
- En caso de que parte de la aplicación sea ofrecida por un servidor web, utilizar el correspondiente fichero "*robots.txt*" para evitar el indexado. Hay que tener cuidado con este punto, ya que podemos ofrecerle información extra al atacante.
- Es recomendable que durante el desarrollo del *software* se utilice un sistema para el control de versiones.

Buenas prácticas en el desarrollo seguro

Seguridad en la base de datos

- El acceso a la base de datos debe realizarse, independientemente del tipo de base de datos, mediante consultas parametrizadas.
- Los parámetros utilizados, y resultados obtenidos en las consultas deben pasar por sus correspondientes procesos de codificación y validación (escapado y filtrado).
- Las diferentes aplicaciones y sistemas deben utilizar el menor nivel de privilegios posibles para acceder a las tablas de la base de datos.
- Los roles con diferentes niveles de acceso deben acceder mediante usuarios diferentes para asegurar la separación de privilegios.
- La conexión a la base de datos debe mantenerse durante el tiempo estrictamente necesario para completar las solicitudes que se requieran.
- Al igual que con el resto de subsistemas (servidor de aplicaciones, etc.) se deben eliminar todos los ficheros y configuración de la instalación por defecto que no sean necesarios.

Buenas prácticas en el desarrollo seguro

Gestión de memoria I

- Algunos lenguajes de programación se encargan de la gestión de memoria de forma automática a través de sus entornos de ejecución (Java, Javascript, Python, Swift, etc.) pero otros como C (que se puede utilizar para el desarrollo de aplicaciones móviles en Android o iOS) tienen un sistema de gestión de memoria manual.
- En aquellos casos es fundamental llevar a cabo una buena gestión de la misma. La mayoría de vulnerabilidades críticas se deben a problemas de gestión de memoria (desbordamiento de búfer).
- Los puntos más críticos en lo referente a la gestión de memoria son aquellos en los que:
 - Se realizan copias de búferes entre direcciones de memoria.
 - Se reserva espacio en memoria para variables de longitud no definida.
 - Se libera memoria previamente liberada.

Buenas prácticas en el desarrollo seguro

Otras consideraciones generales I

- Independientemente del aspecto a programar, si ya existe código testeado y verificado que realice esa operación, siempre es mejor la reutilización.
- Siempre que haya que realizar una tarea relacionada con el sistema operativo, se debe ejecutar a través de las API ofrecidas por el mismo. En ningún caso se deben enviar comandos directamente al sistema operativo a través de la consola.
- Siempre que se vaya a ejecutar código que no haya sido incluido en el despliegue inicial de la aplicación (ejecución dinámica) se debe verificar la integridad del mismo.
- Se deben utilizar los mecanismos de sincronización existentes en el sistemas operativo para evitar la aparición de condiciones de carrera.
- Todas las variables y fuentes de datos deben ser inicializadas antes de su primer uso.

Buenas prácticas en el desarrollo seguro

Otras consideraciones generales II

- Se debe tener en cuenta la representación numérica del lenguaje de programación para evitar errores en la realización de cálculos.
 - En concreto se debe tener en cuenta la precisión de las operaciones, tipos de datos con/sin signo, conversiones, castings y cómo el lenguaje de programación trata los números por encima y por debajo de los límites de representación.
- Si la aplicación va a implementar mecanismos de actualización automática, se debe revisar que el código recibido durante la misma procede de una fuente confiable.
 - Para ello se pueden utilizar mecanismos de firma de código como los utilizados en las tiendas de aplicaciones móviles. Una vez descargado el código y antes de la actualización se debe verificar la firma del mismo.

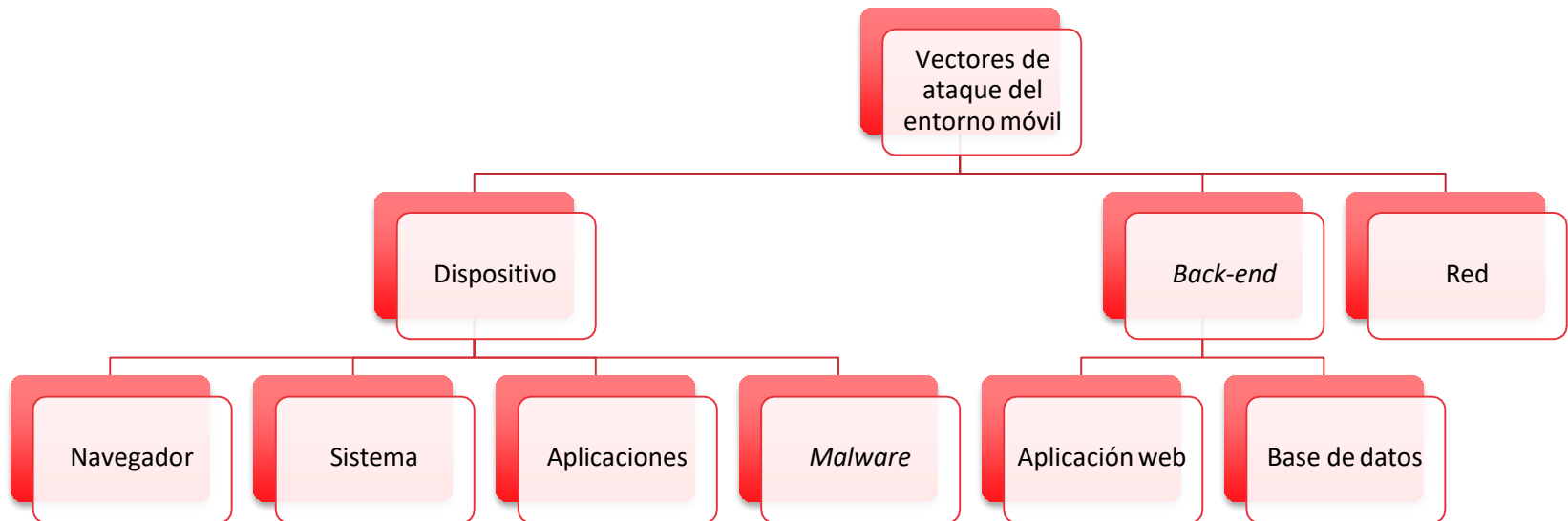
A hand is shown drawing on a green sticky note with a blue pen. In the background, several other colored markers (orange, blue, pink, green) are scattered on a wooden desk. A smartphone is visible in the foreground, displaying a grid of app icons. The scene is set on a desk with a white sheet of paper featuring hand-drawn sketches of a mobile app interface, including a list of items and a smartphone outline.

Buenas prácticas en el desarrollo móvil

Buenas prácticas en el desarrollo móvil

Introducción

- Las aplicaciones móviles comparten características con las aplicaciones web (muchos usuarios, desarrollo rápido, conectividad continua a la red) y con los sistemas de escritorio (almacenamiento compartido de datos, *malware*, existencia de aplicaciones vulnerables como el navegador) en un entorno de movilidad.
- La superficie de ataque en un dispositivo móvil es una combinación de los elementos que afectan a ambos tipos de aplicación.



Buenas prácticas en el desarrollo móvil

Aspectos específicos

- Durante el resto de la sección se van a estudiar los aspectos de seguridad, que hay que tener en cuenta durante el desarrollo de aplicaciones para mitigar las amenazas que generan los diferentes vectores de ataque del entorno móvil:
 - Protección del código de la aplicación.
 - Manejo de datos sensibles.
 - *Logs* y filtrado de información sensible.
 - Manejo de datos sensibles.
 - Componentes HTML en aplicaciones móviles.
 - Comunicación entre aplicaciones.
 - Autenticación en aplicaciones móviles.

Protección de la aplicación



Protección de la aplicación

Ofuscación del código

- Como se comprobó durante el tema 5 del curso, las técnicas de ingeniería inversa pueden ofrecer información muy valiosa sobre el funcionamiento de una aplicación.
- Incrementar la complejidad del código mediante la utilización de técnicas de ofuscación dificultará que el atacante pueda identificar vulnerabilidades y fallos que hayan pasados desapercibidos durante los distintos procesos de revisión.
 - ProGuard
 - DexProtector
- Para dificultar las tareas de ingeniería inversa se pueden utilizar las siguientes técnicas:
 - Restringir el uso de depuradores.
 - Detección de trazas.
 - Optimizaciones del depurador.
 - Destrucción de información de símbolos del binario.

Protección de la aplicación

Restricción del uso de depuradores

- Las aplicaciones pueden restringir, a través del sistema operativo, el uso de depuradores para inspeccionar su ejecución.
- Si bien estas técnicas pueden ser burladas mediante técnicas de *repackaging*, requiere al atacante la realización de un esfuerzo extra.
- En Android, se puede especificar mediante el atributo `android:debuggable="false"` en la etiqueta de la aplicación dentro del *manifest*.
- En iOS se puede introducir la siguiente llamada durante el inicio en la ejecución de la aplicación para que se cierre en caso de que se intente añadir un depurador a la misma:
 - `ptrace (PT_DENY_ATTACH, 0, 0, 0);`

Protección de la aplicación

Detección de trazas

- Dado que se pueden utilizar técnicas de *repackaging* para depurar la aplicación, conviene añadir controles para comprobar si la misma está siendo depurada.
- Si se detecta que la aplicación está conectada a un depurador, se puede:
 - Notificar al *back-end*.
 - Borrar los datos sensibles de la aplicación.
- En Android se puede identificar ejecutando la siguiente línea:

```
boolean depuracion= ( 0 != ( getApplicationInfo().flags &
ApplicationInfo.FLAG_DEBUGGABLE ) );
```
- En iOS se puede implementar el siguiente código ofrecido por Apple:
<https://developer.apple.com/library/ios/qa/qa1361/index.html>

Protección de la aplicación

Optimizaciones del depurador

- Las optimizaciones del depurador modifican el código para que sea más rápido en su ejecución en el procesador, pero también dificultan su lectura y comprensión.
- En Android, se pueden llevar a cabo dos alternativas:
 - Se puede programar parte de la aplicación en C para su utilización como librerías nativas.
 - ProGuard elimina el código no utilizado en la aplicación y modifica los nombres de los métodos, variables, clases y paquetes para dificultar su comprensión. La documentación de ProGuard se puede encontrar en <http://developer.android.com/tools/help/proguard.html>
- En iOS se pueden utilizar herramientas similares a ProGuard:
 - iOS Class Guard: <https://github.com/Polidea/ios-class-guard>
 - LLVM obfuscator: <https://github.com/obfuscator-llvm/obfuscator>

Protección de la aplicación

Destrucción de símbolos del binario

- La destrucción de símbolos del binario (o *binary stripping*) elimina la tabla de símbolos del binario.
- La tabla de símbolos es una estructura de datos creada por el compilador para identificar los nombres de las variables y métodos utilizados en el binario. Su eliminación dificulta la lectura y comprensión del código durante su depuración y análisis estático.
- En Android se puede:
 - Utilizar un compresor de ejecutables como UPX (<http://upx.sourceforge.net>).
 - Utilizar utilidades de consola como `strip`.
- En iOS se puede configurar en las opciones del proyecto bajo la pestaña *Build Settings* y sección *Deployment*.

► Deployment Postprocessing	Yes ⇅
Strip Debug Symbols During Copy	Yes ⇅
Strip Linked Product	Yes ⇅
Strip Style	All Symbols ⇅

Protección de la aplicación

Integridad de la aplicación

- El *repackaging* puede ser utilizado, además de para ataques de ingeniería inversa, para la inyección de código malicioso.
- La aplicación puede verificar la integridad de los componentes de la misma mediante resúmenes o firmas digitales de los distintos componentes de la misma.
- Si se detecta una modificación de la aplicación se puede notificar al *back-end* o borrar los datos sensibles de la aplicación.
- En Android se puede acceder a la información de la firma actual de la aplicación mediante el `PackageManager`:
 - ```
PackageManager packageInfo =
context.getPackageManager().getPackageInfo(context.getPackageName(),
PackageManager.GET_SIGNATURES);
```
  - `packageInfo.signatures`
- En iOS se puede verificar la validez del recibo de compra de la aplicación ofrecido por el sistema.
  - <https://developer.apple.com/library/mac/releasenotes/General/ValidateAppStoreReceipt/Introduction.html>
- Estas comprobaciones pueden ser eliminadas mediante las técnicas de *repackaging*.

# Protección de la aplicación

## Detección de *jailbreak* y *rooting*

- El *jailbreak* o *rooting* de un dispositivo elimina muchas de las medidas de seguridad del sistema para la protección de apps como el *sandboxing*.
- La detección se puede realizar ejecutando alguna de las tareas que sólo se pueden ejecutar en estas condiciones o localizando los ficheros que se instalan durante el proyecto.
- En Android se puede intentar localizar la aplicación supersu o las herramientas busybox:
  - La aplicación supersu tiene nombre de paquete `eu.chainfire.supersu`
  - Ejecutando un comando mediante `Process su = Runtime.getRuntime().exec("comando");`
- En iOS se puede intentar localizar cualquiera de las aplicaciones que se instalan en teléfonos con *jailbreak*.

```
+ (BOOL)isJailbroken{
 if ([[NSFileManager defaultManager] fileExistsAtPath:@"Applications/Cydia.app"]){
 return YES;
 }else if ([[NSFileManager defaultManager] fileExistsAtPath:@"Library/MobileSubstrate/MobileSubstrate.dylib"]){
 return YES;
 }else if ([[NSFileManager defaultManager] fileExistsAtPath:@"bin/bash"]){
 return YES;
 }else if ([[NSFileManager defaultManager] fileExistsAtPath:@"usr/sbin/sshd"]){
 return YES;
 }else if ([[NSFileManager defaultManager] fileExistsAtPath:@"etc/apt"]){
 return YES;
 }
 return NO;
}
```

- Este tipo de técnicas se pueden evadir modificando la localización o nombres de los ficheros o a través del *repackaging* de la aplicación.



A composite image featuring a tablet, a spiral notebook, a USB drive, and a pair of glasses. The tablet, which has a black bezel and a Windows logo at the bottom, is the central focus. Its screen displays a colorful, abstract background of curved lines in shades of red, orange, yellow, green, and blue. Overlaid on the screen is the time '15:51' in a large, white, sans-serif font, and below it, the date 'Donnerstag, 15. Jänner' in a smaller, white, sans-serif font. The tablet is resting on a dark blue, textured spiral-bound notebook. To the left of the notebook, a silver USB drive with the brand name '(Intenso)' printed on it lies horizontally. A pair of red-rimmed glasses with gold-colored hinges and temples is placed diagonally across the bottom of the notebook. The entire scene is set against a light-colored wooden surface.

# Manejo de datos sensibles



# Manejo de datos sensibles

## Datos sensibles de una aplicación

- Las aplicaciones móviles manejan datos sensibles de diferente naturaleza durante su ejecución.
- Los diferentes estados en los que se pueden encontrar los datos en un dispositivo son:
  - En reposo: aquellos datos localizados en el almacenamiento persistente del dispositivo como tarjetas de memoria o sistema interno de archivos.
  - En tránsito: los datos que son enviados y recibidos desde el *back-end* y otros dispositivos móviles.
  - En memoria: los datos sobre los que se están ejecutando operaciones y, por lo tanto, se encuentran almacenados en la memoria del dispositivo.
- Para asegurar el correcto tratamiento de los datos durante todo su ciclo de vida se deben seguir una serie de pautas para su tratamiento en cada uno de los estados en los que se puede encontrar.

# Manejo de datos sensibles

## Almacenamiento seguro de datos I

- Aquellos datos que se consideren sensibles requieren de una protección específica cuando se encuentran en reposo en el dispositivo.
- La utilización de sistemas de ficheros cifrados limita el acceso de un atacante externo al dispositivo si este se encuentra apagado, pero no evita que otras aplicaciones o procesos del dispositivo puedan realizar lecturas de los datos a través del sistema de ficheros (si los correspondientes permisos lo permiten).
- Para evitar el acceso por parte de terceros se debe evitar la utilización del almacenamiento externo del dispositivo (tarjetas SD, etc.) para el almacenamiento de información sensible.
- En Android se debe evitar el uso del atributo `android:installLocation` que permite que los usuario utilicen el almacenamiento externo para instalar la aplicación.

# Manejo de datos sensibles

## Almacenamiento seguro de datos - Android

- Para datos sensibles almacenados en la memoria interna, es recomendable además implementar una capa adicional de cifrado.
- Android no incluye librerías específicas para el cifrado de archivos en reposo (salvo certificados y claves). Para implementar este tipo de cifrado existen dos opciones:
  - Crear desde cero utilizando las librerías de cifrado estándar existentes en la API de Java.
  - Utilizar alguna librería para el almacenamiento seguro de datos como sqlcipher (<http://sqlcipher.net>).
- Para el almacenamiento de claves se pueden utilizar dos API diferentes:
  - KeyChain API para almacenar credenciales que vayan a ser utilizadas a lo largo de todo el sistema (Ej.: certificado raíz de una CA).
  - Keystore para almacenar claves que vayan a ser utilizadas por la aplicación. Desde Android 6.0 es capaz de almacenar claves de cifrado simétricas.

# Manejo de datos sensibles

## Almacenamiento seguro de datos - KeyStore

- Las claves generadas mediante el KeyStore incorporan dos medidas de seguridad:
  - Ninguna aplicación tiene acceso directo a las claves. La API del KeyStore se encarga de todas las operaciones.
  - En aquellos dispositivos que cuenten con un entorno de ejecución seguro o un “elemento seguro” (SE), la clave es almacenada dentro del propio SE. En este caso, las aplicaciones le piden al SE que realice las operaciones criptográficas. Las aplicaciones sólo podrán utilizar el SE para aquellas opciones criptográficas para las que el SE es compatible (algoritmos de cifrado, firma, verificación, etc.).
- Además, cada clave almacenada en el KeyStore puede configurarse de tal forma que sólo pueda utilizarse si el usuario ha desbloqueado el teléfono mediante el un patrón, PIN, contraseña o elemento biométrico (desde Android 6.0).

# Manejo de datos sensibles

## Almacenamiento seguro de datos – iOS

- En iOS, los datos pueden ser cifrados con una capa adicional de cifrado mediante la utilización de la Data Protection API.
- Cada vez que se crea un fichero mediante `NSFileManager` en iOS se pueden especificar cuatro clases diferentes de protección:
  - `NSFileProtectionComplete`: el fichero se cifrará con una clave derivada del código de bloqueo. La clave solo es accesible con el dispositivo desbloqueado y se descarta 10 segundos después del bloqueo.
  - `NSFileProtectionCompleteUnlessOpen`: la misma protección que en el caso anterior pero si el fichero está abierto mientras el dispositivo se bloquea, la clave no se descarta hasta que el fichero es cerrado.
  - `NSFileProtectionCompleteUntilFirstUserAuthentication`: la clave de cifrado se obtiene tras la primera autenticación y no se olvida hasta que el dispositivo es apagado.
  - `NSFileProtectionNone`: no ofrece ninguna protección adicional.



# Manejo de datos sensibles

## Almacenamiento seguro de datos - iOS

- Además, iOS ofrece el Keychain, un contenedor cifrado (también con una clave derivada del código de bloqueo) para el almacenamiento de credenciales por parte de las aplicaciones.
- Los elementos añadidos al Keychain incluyen una serie de atributos que especifican su tipo (usuario, contraseña, certificado, etc.), tipo de autenticación para la que pueden ser utilizados y condiciones para su acceso.
- Además de las condiciones descritas para los ficheros, se añaden nuevas:
  - `kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly`
  - `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`
  - `kSecAttrAccessibleAlwaysThisDeviceOnly`
  - `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`
- Que tienen por objetivo evitar la replicación de los elementos introducidos a otros dispositivos que compartan la misma cuenta de iCloud (y tengan el Keychain en la nube activado).

# Manejo de datos sensibles

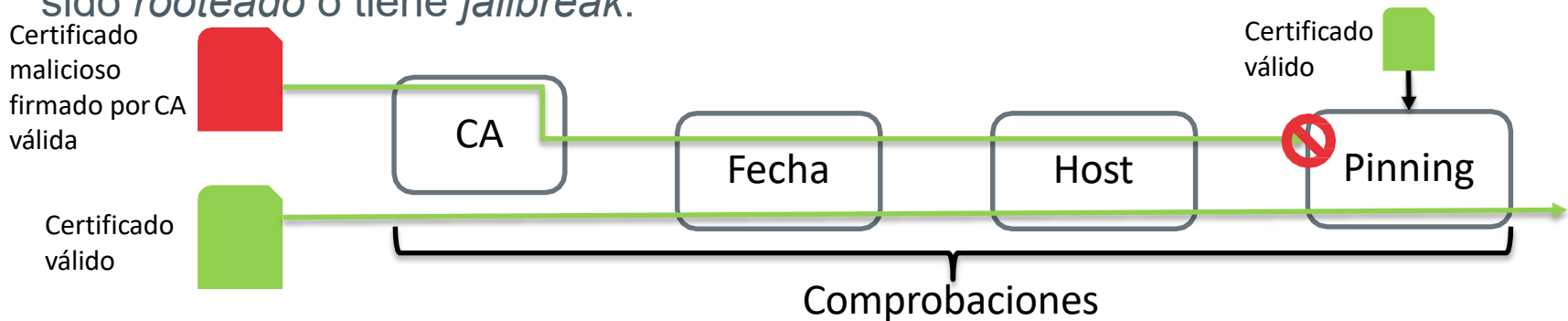
## Seguridad en el transporte de datos – SSL

- Todas las conexiones que incluyan algún tipo de información sensible deberían ser realizadas a través de conexiones SSL que hayan sido complemente validadas.
- Para ello, durante el establecimiento de la conexión hay que verificar una serie de propiedades del certificado:
  - El certificado ha debido ser firmado por una autoridad de certificación válida. Se puede considerar como autoridades válidas aquellas que ya estén en la lista de autoridades válidas del dispositivo o la propia aplicación puede establecer una autoridad válida para sus conexiones (mediante la inclusión del certificado de CA correspondiente).
  - El certificado no debe haber caducado ni debe estar incluido en una lista negra de certificados.
    - Android mantiene una lista negra de certificados no seguros que puede ser actualizada de forma remota.
    - En iOS esa lista se pone al día con las actualizaciones del sistema operativo.
  - El nombre del servidor del certificado debe coincidir con el solicitado.

# Manejo de datos sensibles

## Seguridad en el transporte de datos – *Pinning*

- El *certificate pinning* es una técnica que aprovecha que una aplicación móvil se conecta de forma general a un número limitado de servidores.
- Además de las comprobaciones anteriores la aplicación comprobará que el certificado ofrecido por el servidor corresponde a un certificado que la aplicación conoce previamente.
- Las autoridades de certificación comprometidas no afectan a la seguridad de la aplicación. Si se implementa *pinning*, no es necesaria la firma de una CA confiable.
- El pinning puede desactivarse a través de la *repackaging* o si el dispositivo ha sido *rootado* o tiene *jailbreak*.



# Manejo de datos sensibles

## Seguridad en el transporte de datos – *Pinning* en Android

- Android permite la implementación de *Certificate Pinning* mediante la definición de un `TrustManager` específico.
- Para ello se debe incorporar el certificado de la CA entre los recursos de la aplicación.

```
// Se crea un KeyStore que contenga el certificado de nuestra CA
Certificate ca = //se lee el certificado de un fichero
keyStoreType = KeyStore.getDefaultType();
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);
// Se crea un TrustManager que confie solo en nuestra CA
String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);
// Se crea un contexto (que se utilizará para crear la HttpsURLConnection) que incluya nuestro
TrustManager
Context context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null
```

# Manejo de datos sensibles

## Seguridad en el transporte de datos – *Pinning* en iOS

- En iOS, el pinning se realiza a través del delegado de `NSURLConnection` que incluye el método `willSendRequestForAuthenticationChallenge`.

```
...
SecTrustRef serverTrust = challenge.protectionSpace.serverTrust;
SecCertificateRef certificate = SecTrustGetCertificateAtIndex(serverTrust, 0);
NSData *remoteCertificateData = CFBridgingRelease(SecCertificateCopyData(certificate));
NSData *localCertData = [NSData dataWithContentsOfFile:[NSBundle mainBundle]
pathForResource:@"nombre_fichero" ofType:@"cer"];
if ([remoteCertificateData isEqualToData:localCertData]) {
 NSURLConnection *credential = [NSURLCredential credentialForTrust:serverTrust];
 [[challenge sender] useCredential:credential forAuthenticationChallenge:challenge];
}else {
 // MENSAJE DE ERROR
 [[challenge sender] cancelAuthenticationChallenge:challenge];
}
```

- También se puede implementar mediante la librería TrustKit.



# Manejo de datos sensibles

## Variables en memoria

- Cuando el contenido de una variable que contenga información sensible (claves, *tokens* de autenticación, *cookies*, etc.) deje de ser necesario su contenido debe eliminarse de la memoria.
- Es recomendable que este tipo de valores se guarden en arrays de bytes en vez de en objetos como *Strings*.
  - La reasignación de un objeto tipo *String* suele reservar un nuevo espacio de memoria para la nueva referencia pero no elimina la antigua hasta que pasa el recolector de basura.
  - La utilización de un *array* de bytes permite sobre escribir el contenido de la variable en cualquier momento.

# Manejo de datos sensibles

## Cachés

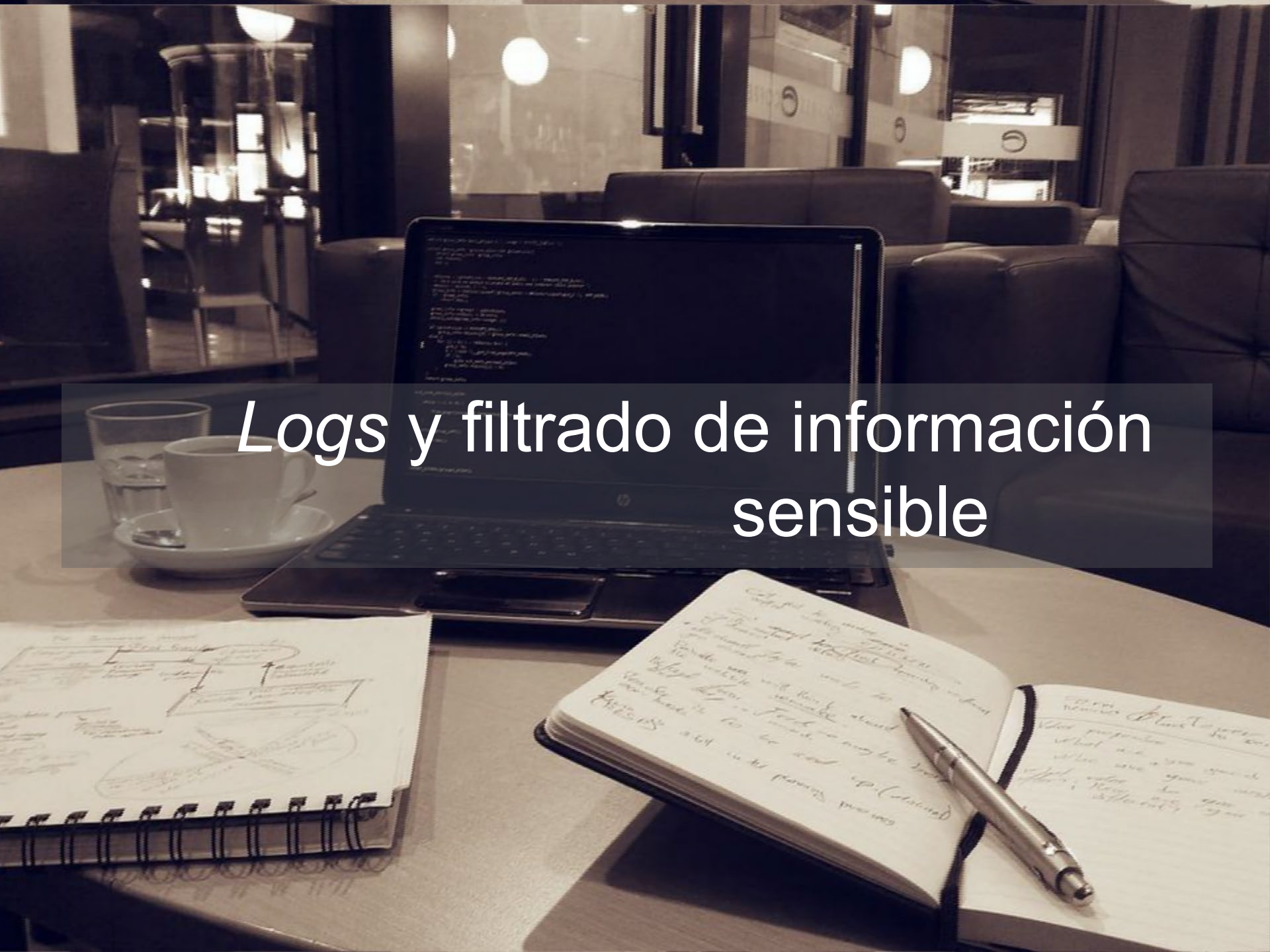
- Siempre que sea posible hay que evitar almacenar datos sensibles en cachés, incluidos:
  - *Cookies*.
  - Ficheros.
  - Bases de datos SQLite.
  - Caché de sitios web.
  - En iOS se puede limitar el caché de sitios web devolviendo `null` en el método `willCacheResponse` del delegado de `NSURLConnection`.

```
- (NSCachedURLResponse *)connection:(NSURLConnection *)connection
willCacheResponse:(NSCachedURLResponse *)cachedResponse {
 return nil;
}
```

- En Android se puede desactivar la caché para conexiones HTTP mediante el método `setUseCaches` de los objetos `URLConnection`.

```
URLConnection connection = miURL.openConnection();
connection.setUseCaches(false);
```

# Logs y filtrado de información sensible



# Logs y filtrado de información sensible

## Eliminación de *logs*

- Siempre que sea posible es recomendable eliminar la mayor cantidad posible de logs que genera la aplicación en el dispositivo.
- En Android se puede configurar ProGuard para eliminar los *logs* añadiendo a su fichero de configuración *proguard.cfg*.

```
-assumenosideeffects class android.util.Log {
 > public static *** d(...);
 > public static *** v(...);
 > public static *** i(...);
 > public static *** e(...);
}
```

- En iOS se puede utilizar un macro que en los versiones del desarrollo en producción elimine la sentencia de *log*.

```
#define NSLog(s,...)
```



# Logs y filtrado de información sensible

## Caché de teclado

- Se debe evitar que el sistema operativo guarde en su caché de teclado datos sensibles tecleados por el usuario.
- Para las contraseñas se deben utilizar los campos específicos de contraseñas. La información tecleada en estos campos no es guardada en la caché del teléfono.
- Para el resto de campos que puedan almacenar datos sensibles se debe desactivar la opción que guarda los datos tecleados en la caché.
- En Android la única manera consistente de realizar esta operación y que funcione en todos los dispositivos es definir el campo como un campo de contraseña con texto visible. Esto se consigue mediante el atributo `android:inputType="textVisiblePassword"` del campo de texto.
- En iOS se consigue configurando la propiedad `autocorrectionType` del `UITextField` con el valor `UITextAutocorrectionTypeNo`.



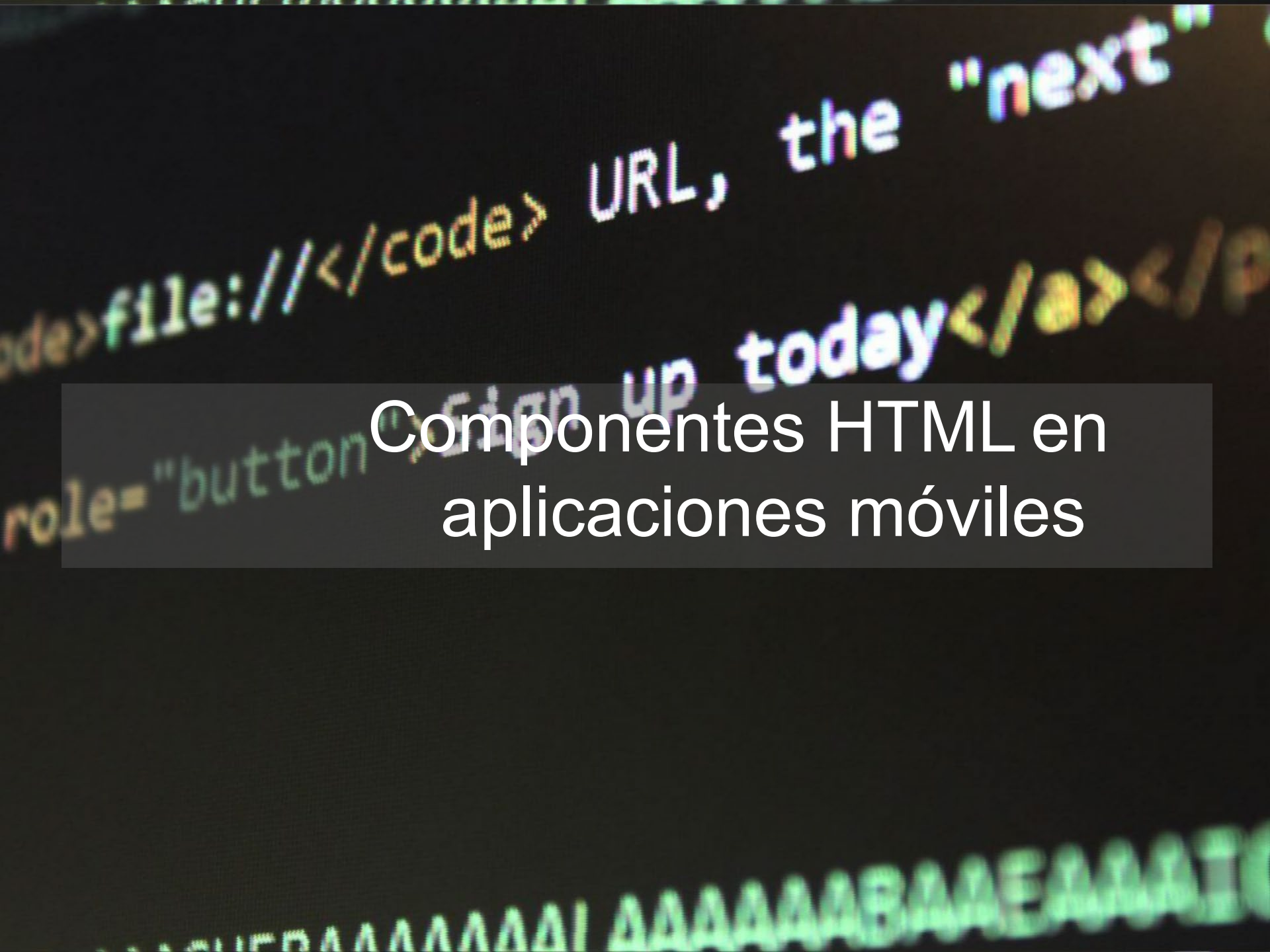
# Logs y filtrado de información sensible

## Portapapeles

- Otras aplicaciones pueden acceder a la información almacenada en el portapapeles del sistema sin necesidad de ningún permiso especial.
- En Android se debe crear una subclase de la clase `EditText` y reimplementar los métodos `isSuggestionsEnabled` y `canPaste` para que devuelvan `false`.
- En iOS existen dos opciones:
  - Mediante la creación de una subclase de `UITextField` que no permita las acciones de *copy* o *cut*.

```
-(BOOL)canPerformAction:(SEL)action withSender:(id)sender {
 if (action == @selector(copy:) || action == @selector(cut:)) {
 return NO;
 }
 [super canPerformAction:action withSender:sender];
}
```

- Cuando las funciones `copy` y `cut` se ejecuten, se llama al método `pasteboardWithUniqueName` para obtener el portapapeles específico de la aplicación.



# Componentes HTML en aplicaciones móviles

# Componentes HTML en aplicaciones móviles

## Introducción

- Todos los sistemas operativos móviles permiten a las aplicaciones mostrar contenido HTML a través de vistas web.
- La utilización de estas vistas conlleva ciertos riesgos de seguridad que hay que tener en cuenta durante su utilización.
- Para reducir la superficie de ataque generada por estas vistas, se recomienda, independientemente de la plataforma:
  - No cargar nunca contenido remoto mediante conexiones sin cifrar (Sin SSL).
  - Asegurarse de que se utiliza una configuración segura de cifrado SSL y se valida completamente el certificado SSL ofrecido por el servidor.
  - Prohibir el acceso al sistema de ficheros del dispositivo desde la vista web.
  - Desactivar Javascript y cualquier otro *plugin* siempre que sea posible.
  - Verificar que la vista web solo carga URL de los dominios que necesita.
  - No exponer métodos nativos a través de Javascript.
  - No permitir la carga de URL a través de los mecanismos de comunicación con otras aplicaciones.

# Componentes HTML en aplicaciones móviles

## Asegurando componentes en Android

- Desactivar Javascript.

```
WebView webview = new WebView(this);
webview.getSettings().setJavaScriptEnabled(false);
```

- Desactivar el acceso al sistema de ficheros.

```
WebView webview = new WebView(this);
webview.getSettings().setAllowFileAccess(false);
```

- Verificar las URL que se cargan en la vista web (extensión de WebView).

```
private class MiWebViewClient extends WebViewClient {
 @Override
 public boolean shouldOverrideUrlLoading(WebView view, String url) {
 //COMPROBACIONES PARA LA URL INICIAL
 }
 @Override
 public WebResourceResponse shouldInterceptRequest(final WebView view, String url){
 //COMPROBACIONES PARA TODAS LAS PETICIONES
 }
}
```



# Componentes HTML en aplicaciones móviles

## Asegurando componentes en Android

- Para evitar la carga de URL desde elementos externos a la aplicación se deben eliminar del `manifest` todos los atributos de *exported* de las actividades definidas en la aplicación que tengan una vista web como vista principal.
- La eliminación de los datos de la caché de la vista web se debería realizar una vez se ha dejado de utilizar (método `onPause()` de la actividad).

```
@Override
protected void onPause() {
 ...
 webView.clearCache();
 ...
 super.onPause();
}
```

- Finalmente, se debe evitar exponer el código nativo a la vista utilizando el método `addJavaScriptInterface()`.

# Componentes HTML en aplicaciones móviles

## iOS - UIWebView

- Desde iOS 9 , se disponen de tres tipos de vista web.
- UIWebView su uso no está recomendado desde iOS8 pero se puede utilizar en las aplicaciones.
- Utiliza un motor de renderizado no optimizado, por lo que las páginas cargan más lentamente.
- Solo permite la configuración a través de los métodos del delegado, de los cuales el único relevante `webViewShouldStartLoadWithRequest`, que permite identificar la URL a la que se va a conectar la vista web.

```
- (BOOL)webView:(UIWebView*)webView shouldStartLoadWithRequest:(NSURLRequest*)request
navigationType:(UIWebViewNavigationType)navigationType {
 NSURL *url = request.URL;
 //COMPROBACIONES
 return ...;
}
```

- Las vistas web de este tipo no permiten desactivar el motor de Javascript.

# Componentes HTML en aplicaciones móviles

## iOS - WKWebView

- WKWebView es la vista web por defecto utilizada a partir de iOS 8.
- Durante su inicialización se pueden definir una serie de parámetros a través de un objeto del tipo WKWebViewConfiguration.
- Por ejemplo, para desactivar Javascript en la vista se puede utilizar el siguiente código:

```
WKWebViewConfiguration *conf = [[WKWebViewConfiguration alloc] init];
conf.preferences.javaScriptEnabled = NO
WKWebView *webView = [[WKWebView alloc] initWithFrame:self.view.frame
configuration:conf];
```

- Permite efectuar *certificate pinning* a través del delegado tal y como se mostró anteriormente en la sección de protección de datos sensibles:
  - En iOS 8 existe un error en el delegado, por lo que el *pinning* se debe realizar a revisando los certificados incluidos en la propiedad `certificateChain`.

# Componentes HTML en aplicaciones móviles

## iOS - SafariViewController

- SafariViewController ofrece un controlador específico que permite la navegación web desde el interior de la aplicación.
- Este controlador se ejecuta en un proceso diferente a la aplicación que lo invoca.
- El usuario puede acceder a toda la funcionalidad de Safari, incluidos el autorelleno de contraseñas, botón para ejecutar acciones y barra de direcciones en modo lectura.
- La aplicación no puede acceder ni a los sitios que se navegan desde la vista ni a los datos que el usuario introduce en la misma.
- Si la aplicación quiere tener un control mayor sobre el contenido mostrado, deberá utilizar la vista `WKWebView`.





# Comunicación entre aplicaciones



# Comunicación entre aplicaciones

## Comunicación entre aplicaciones en Android

- La comunicación entre aplicaciones en Android supone un vector adicional de ataque que hay que controlar.
- En primer lugar, todos los elementos de la aplicación que no sirvan específicamente para comunicarse con otras aplicaciones deben ser marcados en el *manifest* con la propiedad `android:exported=false`.

```
<activity android:name=".MyActivity" android:label="Etiqueta"
android:exported=false >
 <intent -filter>
 <action android:name="accion.intent"></action>
 </intent>
</activity>
<service android:enabled="true" android:name=".MyService"
android:exported=false ></service>
<receiver android:enabled="true" android:exported=false
 android:label="My Broadcast Receiver"
 android:name=".MyBroadcastReceiver"
</receiver>
```

# Comunicación entre aplicaciones

## Intents

- Se debe evitar enviar información sensible a través de `BroadcastIntents`, ya que actividades maliciosas pueden definir el mismo filtro con mayor prioridad para capturar la información sensible.
- Toda la información recibida de un *Intent* debe ser validada como cualquier otra entrada.
- Se debe evitar definir `IntentFilters` si la actividad no es pública. De esta manera la única manera de acceder a los mismos será a través del nombre de componente.

```
//INTENT CON INFORMACIÓN SENSIBLE: SI ES CAPTURADO LOS DATOS SON COMPROMETIDOS
public static Intent crearIntent(Context context, Usuario user) {
 Intent i = new Intent(context, DetallesUsuario.class);
 i.putExtra(EXTRA_USUARIO, user);
 return i;
}
//INTENT QUE ENVÍA SOLO EL ID DEL USUARIO
public static Intent crearIntent(Context context, String userId) {
 Intent i = new Intent(context, DetallesUsuario.class);
 i.putExtra(EXTRA_USER_ID, userId);
 return i;
}
```

# Comunicación entre aplicaciones

## Content Providers

- Los *ContentProviders* ofrecen los datos de la aplicación a aplicaciones de terceros.
- Los *ContentProviders* pueden requerir a otras aplicaciones permisos para leer (`readPermission`), escribir (`writePermission`) o ambos (`permission`) en la base de datos de la aplicación a través del *provider*.

```
<provider android:name="MiProvider"
 android:authorities="com.miapp.provider.MisDatos"
 android:readPermission="permiso.de.lectura"
 android:writePermission="permise.de.escritura"
 android:permission="permiso.para.ambos">
</provider>
```

- Los permisos definidos en el interior del *provider* deben haber sido declarado antes en la aplicación a través de elementos de tipo `<permission>`.
- Un *provider* debe validar todas las peticiones que reciba para evitar inyección de código SQL o acceso a ficheros no autorizados.
- Mediante el atributo `android:grantUriPermissions="true"`. Una aplicación también puede ofrecer acceso esporádico a elementos del *ContentProvider* a aplicaciones que lo soliciten, aunque no hayan definido ningún permiso específico.

# Comunicación entre aplicaciones

## Servicios

- Los servicios también pueden exponer funcionalidades o información sensible a aplicaciones de terceros que deben ser protegidos convenientemente.
- Un servicio puede validar método a método los permisos que tiene una aplicación para acceder a una funcionalidad específica a través del método `checkPermission()` del `PackageManager`.
- También se pueden definir los permisos necesarios para comunicarse con el servicio a través del *manifest* de la aplicación.

```
<permission android:name="com.mipermiso" android:label="mi_permiso"
android:protectionLevel="dangerous"></permission>`
<service android:name="com.MiServicio" android:permission="com.mipermiso">
 <intent-filter>
 <action android:name="com.MI_ACCION"/>
 </intent-filter>
</service>
```



# Autenticación en aplicaciones móviles





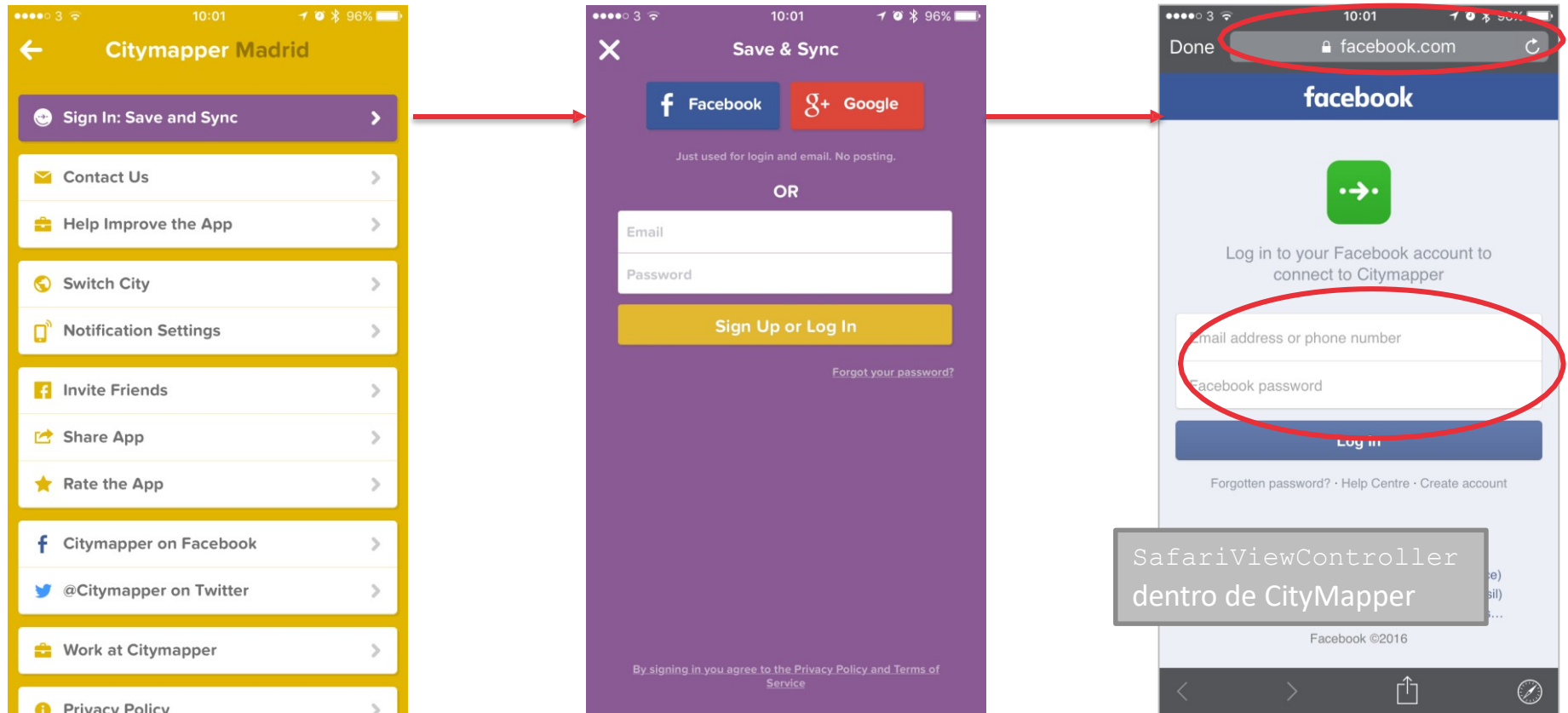
# Autenticación en aplicaciones móviles

## Introducción a OAuth

- OAuth es un estándar para la autorización de acceso a recursos/servicios a través de la web.
- Desde el punto de vista del usuario final, OAuth ofrece las siguientes ventajas:
  - Utilización de una sola cuenta para acceder a múltiples servicios.
  - Es necesario recordar menos contraseñas.
  - No es necesario compartir las credenciales (usuario y contraseña generalmente) con un servicio externo en el que no se confía.
  - Se pueden revocar autorizaciones otorgadas de forma sencilla.
- Desde el punto de vista del desarrollador, ofrece las siguientes ventajas:
  - Se simplifica el manejo y cantidad de información sensible a almacenar.
  - No se requiere de un sistema de gestión o renovación de contraseñas.
  - Su implementación se lleva a cabo mediante librerías ampliamente testadas.
  - Existen multitud de proveedores de identidad y otros servicios que ya utilizan OAuth.

# Autenticación en aplicaciones móviles

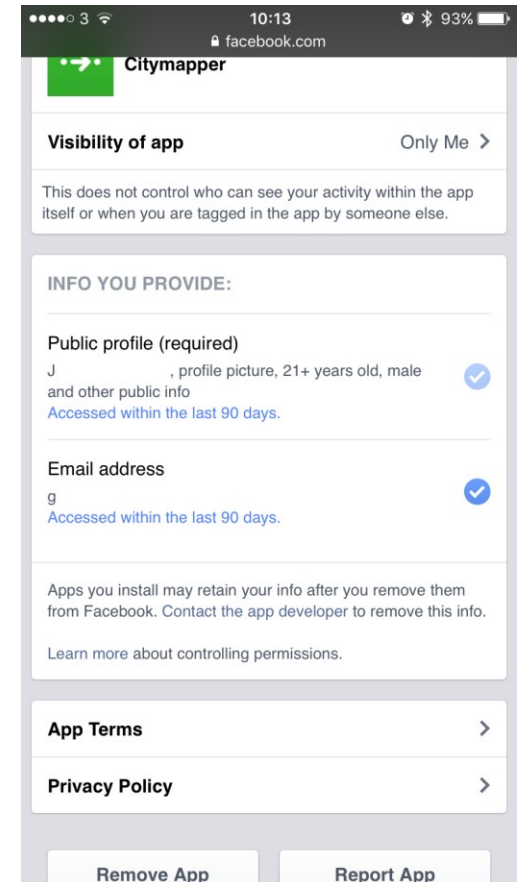
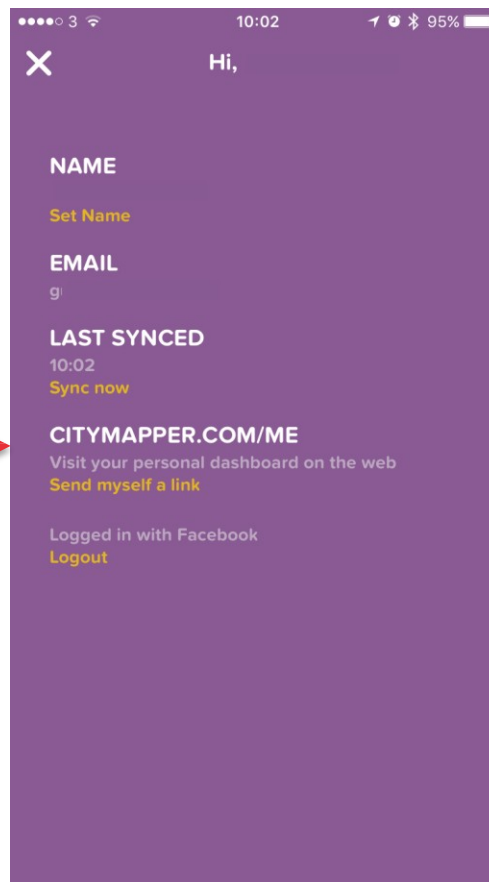
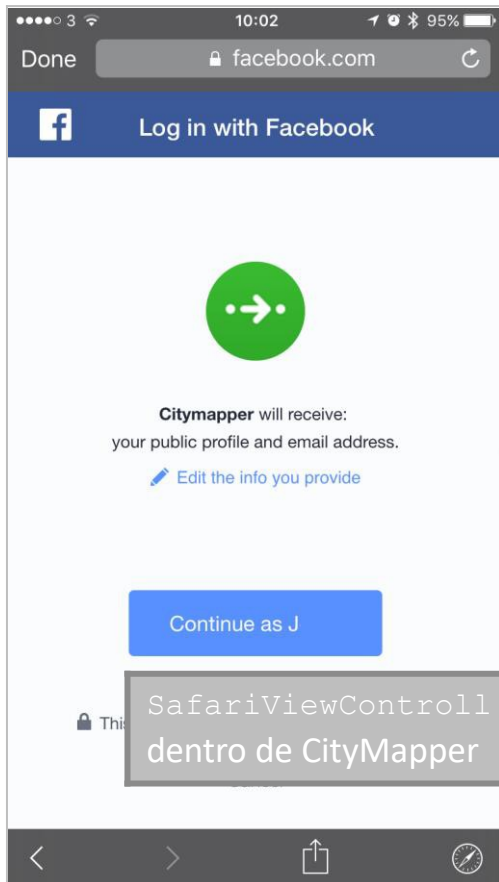
## Con OAuth



Las credenciales sólo se envían al proveedor de la autorización

# Autenticación en aplicaciones móviles

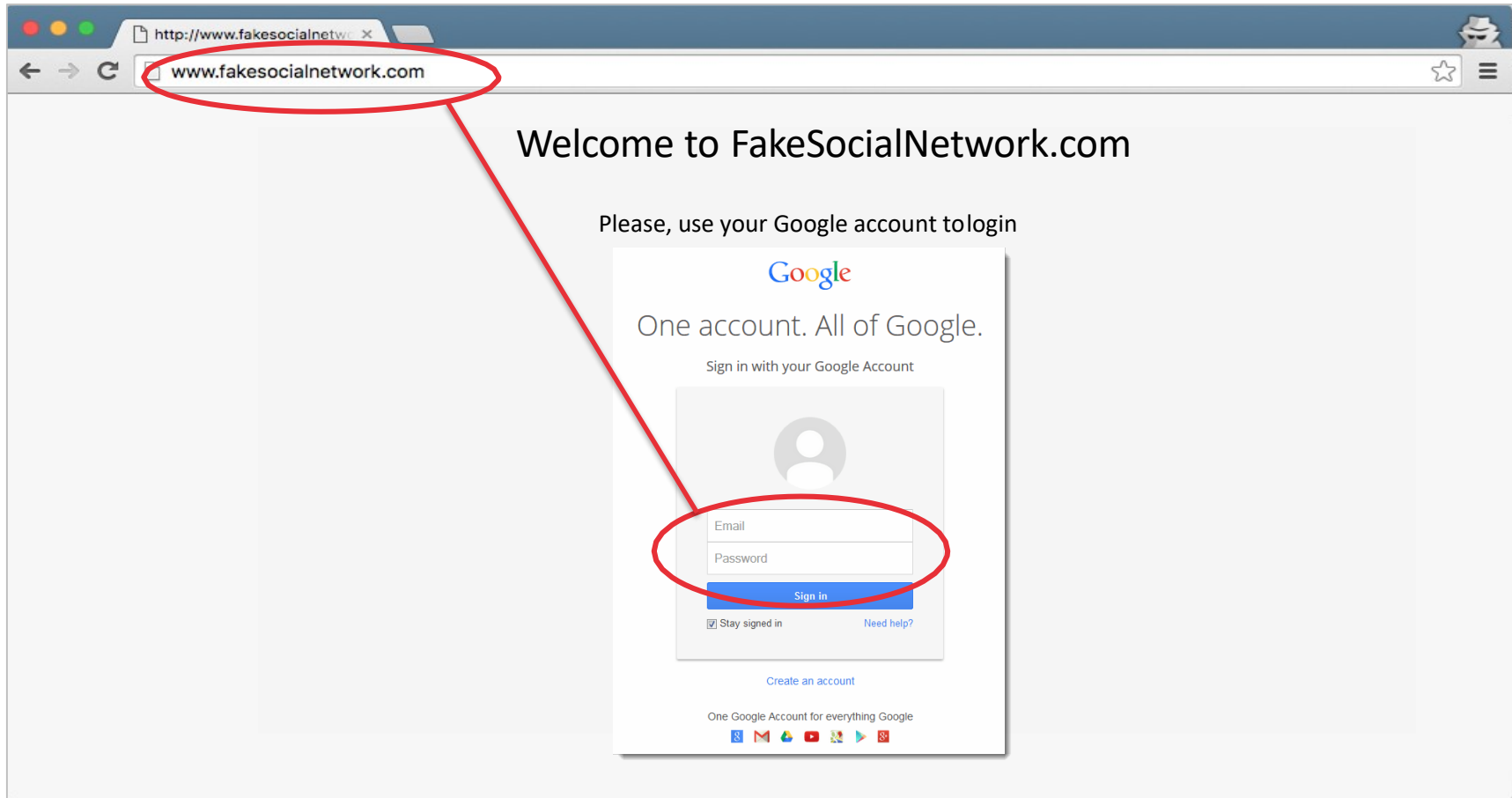
## Con OAuth



El usuario autoriza la información que quiere compartir y en cualquier momento puede modificarla a través del proveedor de la misma

# Autenticación en aplicaciones móviles

## La autenticación antes de OAuth



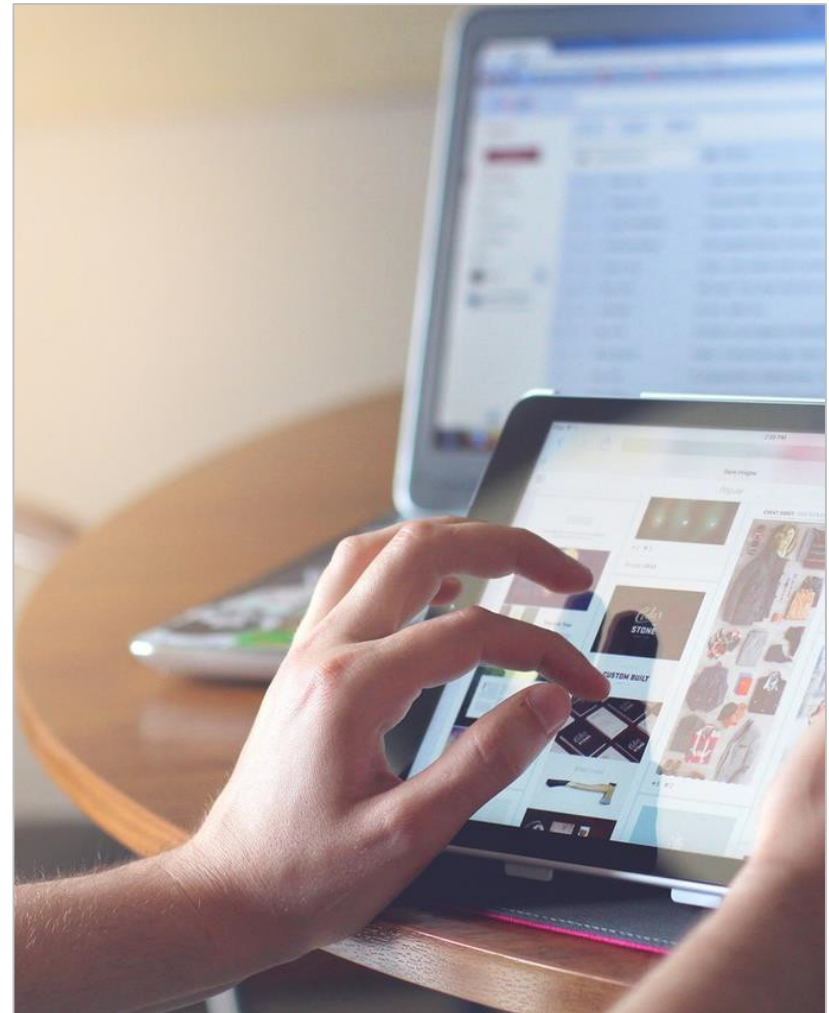
Para autorizar o autenticarnos en una web debíamos introducir las credenciales del proveedor de la autorización en una web no confiable



# Autenticación en aplicaciones móviles

## Roles

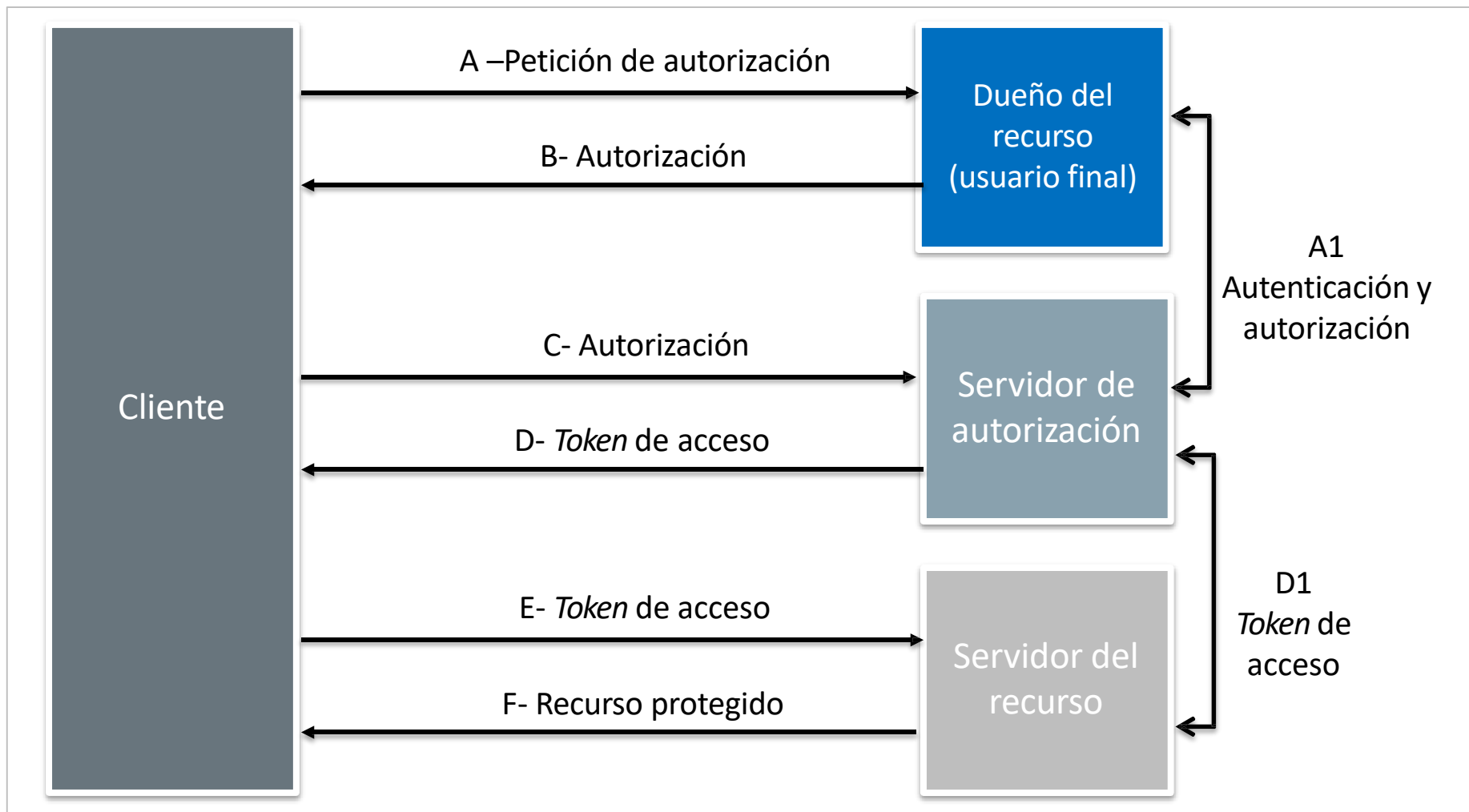
- Durante la ejecución del protocolo OAuth participan cuatro roles diferentes:
  - **Dueño el recurso:** es la entidad capaz de autorizar el acceso al recurso en concreto. En la mayoría de los escenarios en el entorno móvil será el usuario final.
  - **Servidor del recurso:** es el servidor que almacena el recurso del cliente. Es capaz de dar acceso al mismo si se le presentan las credenciales oportunas (*tokens* de acceso que se describirán más adelante).
  - **Cliente:** es la aplicación que solicita el acceso al recurso en nombre del dueño del recurso. En el caso de las aplicaciones móviles puede ser la propia aplicación o el *back-end* de la misma.
  - **Servidor de autorización:** es el servidor que genera los *tokens* de acceso para el cliente después de que el dueño del recurso se haya autenticado ante el servidor del recurso. En muchos escenarios los roles de servidor de recursos y servidor de autorización son ejecutados por la misma entidad.





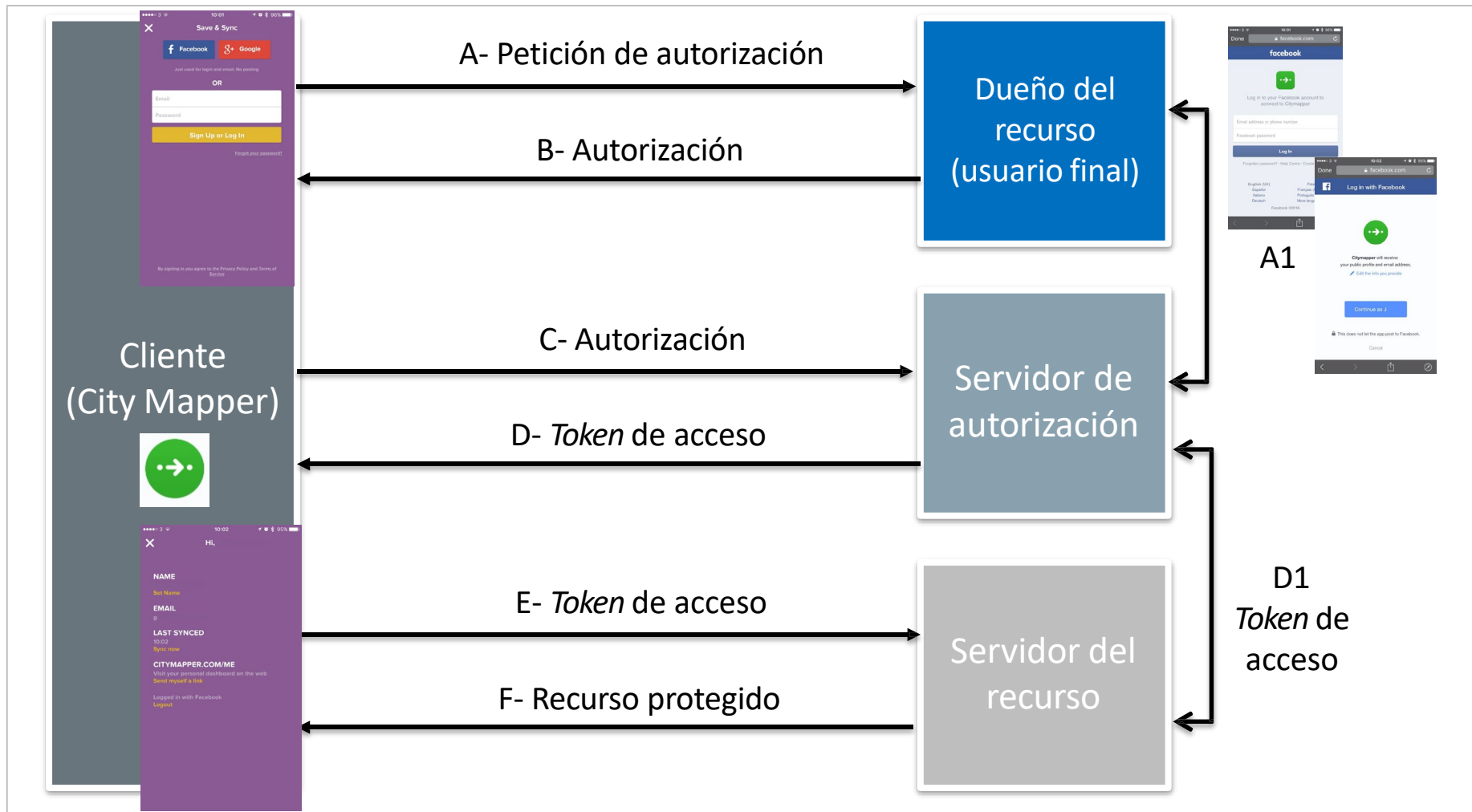
# Autenticación en aplicaciones móviles

## Esquema



# Autenticación en aplicaciones móviles

## Esquema en el ejemplo de City Mapper



# Autenticación en aplicaciones móviles

## Descripción del esquema

- A) El cliente solicita la autorización del dueño del recurso. La autorización puede ser resuelta de forma local (en el dispositivo móvil) si el recurso (cuenta) está configurado dentro del propio dispositivo (*frameworks* de gestión de cuentas). Si la cuenta no está configurada, se puede obtener con el servidor de autorización como intermediario (A1).
- B) El cliente recibe la autorización (una credencial que representa que el cliente tiene acceso para ese recurso específico). OAuth 2.0 define cuatro tipos diferentes de autorización:
  - Código de autorización: un código de autorización que se obtiene cuando se ejecuta el paso A1.
  - Código implícito: el cliente recibe el *token* de autenticación directamente después de A1. Siempre que sea posible debe evitarse debido a las implicaciones de seguridad.
  - Contraseña: se utiliza un par de credenciales usuario y contraseña. Elimina toda la seguridad del esquema por lo que debe evitarse siempre.
  - Credenciales de cliente: si el recurso pertenece al mismo cliente o ya ha sido permitido su acceso basta con utilizar las credenciales del cliente.

# Autenticación en aplicaciones móviles

## Descripción del esquema

- C) El cliente se autentica ante el servidor de autorización con sus propias credenciales (debe estar dado de alta) y la autorización que reciba en el paso anterior.
- D) El servidor valida las credenciales del cliente y la autorización. Si son correctas genera un *token* de acceso:
  - El servidor de autorización y servidor de recursos comparten el *token* de acceso (en caso de que no sean la misma entidad) a través de un canal fuera de banda.
  - El *token* de acceso puede incluir además de los recursos a los que permite el acceso, el tiempo durante el cual el acceso es permitido por el mismo.
- E) El cliente solicita acceso al recurso al servidor de recursos. Para ello presenta el *token* de acceso obtenido en el anterior paso.
- F) El servidor de recursos valida el *token* de acceso y si la validación es correcta envía el recurso solicitado.

# Autenticación en aplicaciones móviles

## Autorización en los sistemas operativos móviles

- En el entorno móvil, el servidor de autorización y recursos puede ser accedido a través de tres mecanismos diferentes:
  - **API de cuentas del sistema operativo:** la aplicación que ofrece los recursos debe haber registrado alguna cuenta en el propio sistema operativo. El cliente realiza la solicitud al sistema de cuentas del sistema, que redirige la solicitud a la API de autenticación/autorización específica de la cuenta seleccionada por el usuario.
  - **Aplicaciones:** es similar al caso anterior, pero el cliente solicita directamente la autorización al servidor de recursos/autorización a través de una aplicación específica ya instalada en el cliente. Para ello se utilizan las librerías de comunicación entre aplicaciones.
  - **Vistas web:** si la aplicación del servidor de recursos/autorización no está instalada en el dispositivo, se puede utilizar una vista web para la realización del proceso. Una vez el usuario se ha autorizado, el cliente puede extraer el *token* de acceso de la vista web correspondiente.



# Autenticación en aplicaciones móviles

## Autorización en Android – AccountManager I

- El `AccountManager` de Android se puede utilizar para realizar peticiones de autorización a la cuenta de Google instalada en el dispositivo o a cuentas de otros proveedores que hayan implementado los métodos necesarios dentro de su aplicación.
- Permisos requeridos para acceder a las cuentas del dispositivo.

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
```

- Durante la creación de la actividad se verifica si el usuario ya ha dado su autorización.

```
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 AccountManager accountManager = AccountManager.get(this);
 if (tokenGuardado() != null) {
 accionesAutenticado(tokenGuardado());
 } else {
 escogerCuenta();
 }
}
```

# Autenticación en aplicaciones móviles

## Autorización en Android – AccountManager II

- Si el cliente no ha sido autorizado se deberá solicitar la cuenta con la que queremos acceder al recurso (com.paquete.cuenta).

```
private void escogerCuenta() {
 int ACCOUNT_REQUEST_CODE = 1601;
 Intent intent = AccountManager.newChooseAccountIntent(null, null,
 new String[] { "com.paquete.cuenta" }, false, null, null, null, null);
 startActivityForResult(intent, ACCOUNT_REQUEST_CODE);
}
```

- La respuesta incluirá el nombre de cuenta, por lo que debemos obtener la cuenta y realizar la petición del *token* de autenticación para ese recurso.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
 super.onActivityResult(requestCode, resultCode, data);
 if (resultCode == RESULT_OK) {
 if (requestCode == ACCOUNT_REQUEST_CODE) {
 String accountName = data.getStringExtra(AccountManager.KEY_ACCOUNT_NAME);
 for (Account account : accountManager.getAccountsByType("com.paquete.cuenta")) {
 if (account.name.equals(accountName)) {
 userAccount = account;
 break;
 }
 }
 ///RECURSO DEPENDERÁ DEL RECURSO AL QUE QUERAMOS ACCEDER
 ///PARA HANGOUTS SE REQUIERE "https://www.googleapis.com/auth/googletalk";
 accountManager.getAuthToken(userAccount, "oauth2:" + RECURSO, null, this,
 new OnTokenAcquired(), null);
 }
 }
}
```

# Autenticación en aplicaciones móviles

## Autorización en Android – AccountManager III

- Una vez obtenido el *token* de autorización se llama a un nuevo objeto de tipo `OnTokenAcquired` que hace de *callback* para la recepción del *token* de autenticación. El *token* debería ser guardado como un dato sensible de la aplicación.

```
private class OnTokenAcquired implements AccountManagerCallback<Bundle> {
 @Override
 public void run(AccountManagerFuture<Bundle> result) {
 try {
 Bundle bundle = result.getResult();
 Intent launch = (Intent) bundle.get(AccountManager.KEY_INTENT);
 if (launch != null) {
 //AUTORIZACION FALLIDA, SE DEBE VOLVER A INTENTAR
 startActivityForResult(launch, AUTHORIZATION_CODE);
 } else {
 String token = bundle.getString(AccountManager.KEY_AUTH_TOKEN);
 guardarToken(token);
 accionesAutenticado(token);
 }
 } catch (Exception e) {
 throw new RuntimeException(e);
 }
 }
}
```

# Autenticación en aplicaciones móviles

## Autorización en Android – SDKs de aplicaciones

- Es posible que el servidor de recursos acepte peticiones de recursos a través de su propia aplicación, pero sin la integración con el sistema de cuentas de Android.
- En esos casos, el propio servicio suele ofrecer documentación exhaustiva de cómo utilizar la aplicación para la solicitud de *tokens* de autenticación.
- La documentación suele incluir el proceso de registro de la aplicación cliente y como realizar las llamadas entre ambas aplicaciones para solicitar y recibir el *token* de autorización.
- Ejemplos de este tipo de autorización son:
  - Facebook: <https://developers.facebook.com/docs/facebook-login/android>
  - Twitter: a través de las API de:
    - Twitter Kit para Android <https://github.com/twitter/twitter-kit-android>
    - Twitter Core si se utiliza Fabric <https://docs.fabric.io/android/twitter/twitter-core.html>



# Autenticación en aplicaciones móviles

## Autorización en Android - Navegador

- Si la aplicación no está instalada se puede hacer uso de una WebView (vista web ofrecida por el sistema) para acceder a la información. El funcionamiento de OAuth puede diferir de un servicio a otro.

```
@Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 String url = URL_DE_OAUTH + "?client_id=" + ID_NUESTRA_APP_EN_SERVICIO_OAUTH;
 WebView webview = (WebView)findViewById(R.id.webview);
 webview.getSettings().setJavaScriptEnabled(true);
 webview.setWebViewClient(new WebViewClient() {
 public void onPageStarted(WebView view, String url, Bitmap favicon) {
 String accessTokenFragment = "access_token=";
 String accessCodeFragment = "code=";
 if (url.contains(accessTokenFragment)) {
 // Capturamos las peticiones para buscar los codigos de autorization y el token
 String accessToken = url.substring(url.indexOf(accessTokenFragment));
 guardarToken(accessToken);
 } else if (url.contains(accessCodeFragment)) {
 // Si es un access code realizamos otra peticion para obtener el token
 String accessCode = url.substring(url.indexOf(accessCodeFragment));
 guardarCodigoAutorizacion(accessCode);
 String query = "client_id=" + ID_NUESTRA_APP_EN_SERVICIO_OAUTH + "&client_secret=" +
 SECRETO_OBTENIDO_DURANTE_EL_REGISTRO_DE_NUESTRA_APP + "&code=" + accessCode;
 view.postUrl(OAUTH_ACCESS_TOKEN_URL, query.getBytes());
 }
 }
 });
 webview.loadUrl(url);
}
```

# Autenticación en aplicaciones móviles

## Autorización en iOS - *Accounts Framework*

- iOS permite la configuración de las cuentas de Facebook y Twitter en los ajustes del propio dispositivo, de tal forma que otras aplicaciones pueden solicitar el acceso a los mismos.
- En este caso las preferencias de privacidad de iOS permiten revocar directamente los *tokens* de acceso.
- A continuación se muestra un ejemplo para el acceso a la cuenta de correo almacenada en la cuenta de Facebook del usuario.

```
ACAccountStore *accountStore = [[ACAccountStore alloc] init];
ACAccountType *accountType = [accountStore
ccountTypeWithAccountTypeIdentifier:ACAccountTypeIdentifierFacebook];
NSDictionary *options = @{@"ACFacebookAppIdKey" : APPID_DEL_CLIENTE_EN_FACEBOOK,
 @"ACFacebookPermissionsKey" : @"email"};

[accountStore requestAccessToAccountsWithType:accountType
options:options
completion:^(BOOL granted, NSError *error){
 if (granted) {
 _currentUser.facebook = [accounts firstObject];
 } else {
 //MOSTRAR MENSAJE DE ERROR
 }
}];
```

# Autenticación en aplicaciones móviles

## Autorización en iOS – SDKs de aplicaciones

- De la misma manera que en Android, es posible que el servidor de recursos acepte peticiones de recursos a través de su propia aplicación instalada en iOS.
- En esos casos, el propio servicio suele ofrecer documentación exhaustiva de cómo utilizar la aplicación para la solicitud de *tokens* de autenticación.
- Ejemplos de este tipo de autorización son:
  - Facebook:  
<https://developers.facebook.com/docs/facebook-login/ios>
  - Foursquare:  
<https://developer.foursquare.com/resources/libraries>
  - Twitter: a través de la API de Fabric:  
<https://docs.fabric.io/ios/twitter/authentication.html>



# Autenticación en aplicaciones móviles

## Autorización en iOS – Vistas web

- En iOS se puede utilizar cualquiera de las tres vistas web disponibles para implementar OAuth.
- Tanto UIWebView como WKWebView permiten acceder a las credenciales que el usuario escribe dentro de la vista por lo que se recomienda la utilización de SafariViewController.
- SafariViewController corre en un proceso separado y además, no requerirá al usuario las credenciales si ya se han utilizado previamente con Safari.
- Existe una librería OAuthSwift con soporte para realizar peticiones OAuth a través de SafariViewControllers escrita en Swift.
  - <https://github.com/mwermuth/OAuthSwift/tree/swift2.0>
- Entre los servicios soportados se pueden encontrar Twitter, Flickr, Github, Instagram, Foursquare, Fitbit, LinkedIn, Dropbox y Salesforce entre otros.