UNIVERSIDAD COMPLUTENSE MADRID

Autenticación delegada

Gestión de la Información en la Web
Enrique Martín - emartinm@ucm.es
Grados de la Fac. Informática

Gestión de claves

- Como hemos visto, **gestionar** de manera adecuada las **claves** que almacenamos en la BD requiere **tiempo y esfuerzo**.
- Además, las leyes de protección de datos imponen limitaciones y exigencias sobre los ficheros que contienen datos personales → **más tiempo y esfuerzo**.
- *¿Cómo podemos evitarlo?*

Autenticación delegada

- Una solución es **delegar** el proceso de autenticación en una **tercera parte**: Google, Facebook, Twitter, etc.
- De esta manera es esa **tercera parte** la que tiene que **gestionar** de manera segura los datos y las claves, y **cumplir** la legislación sobre almacenamiento de datos.

Autenticación delegada

- Usando autenticación delegada **facilitamos** a los potenciales usuarios el **acceso** a nuestra aplicación web:
  - No se tienen que registrar (o deben dar muy pocos datos).
  - Acceden a través de una red social de confianza (Google, Facebook, etc.).
- Adicionalmente podremos **integrarnos** con esa red social: publicar logros, obtener amigos, obtener imagen de perfil, almacenar en la nube.

Autenticación delegada

- En principio **todos ganan** (*win-win*):
  - La red social consigue visibilidad y potenciales nuevos usuarios.
  - Nuestra aplicación se “desentiende” de la autenticación, aprovecha la confianza de la red social y se puede integrar con ella.
- **Ojo**: la autenticación delegada es un servicio, y como tal la red social podría cobrarlo.

Protocolos

- El principal protocolo de autenticación delegada es **OpenID Connect**.
- *OpenID Connect* es la **tercera evolución** de OpenID, tras pasar también por OpenID 2.0.
- Las versiones anteriores **no** gozaron de mucha **aceptación** debido a ser **complicadas** de utilizar.
  - Entre otros puntos requerían un uso intensivo de la firma de mensajes, al poder ser usado sobre HTTP no seguro.

Protocolos

- OpenID Connect **se despliega sobre OAuth 2.0**, el protocolo de autorización delegada con más éxito.
- Por ello la **autenticación** con *OpenID Connect* y la **autorización** con *OAuth 2.0* son procesos muy similares, que comparten muchos de los pasos.
- Todo el proceso se basa en el envío y obtención de **tokens**: uno de **identificación** (*id_token*) y uno de **acceso** (*access_token*).

Autenticación usando Google

- Veamos un **ejemplo** de autenticación utilizando OpenID Connect y la API de Google. *(Usar otra red social será muy parecido)*
- Mostraremos las **ideas principales** del proceso y de las peticiones HTTP involucradas, pero debéis consultar los **detalles** concretos en la documentación oficial de Google:
  - https://developers.google.com/identity/protocols/OpenIDConnect

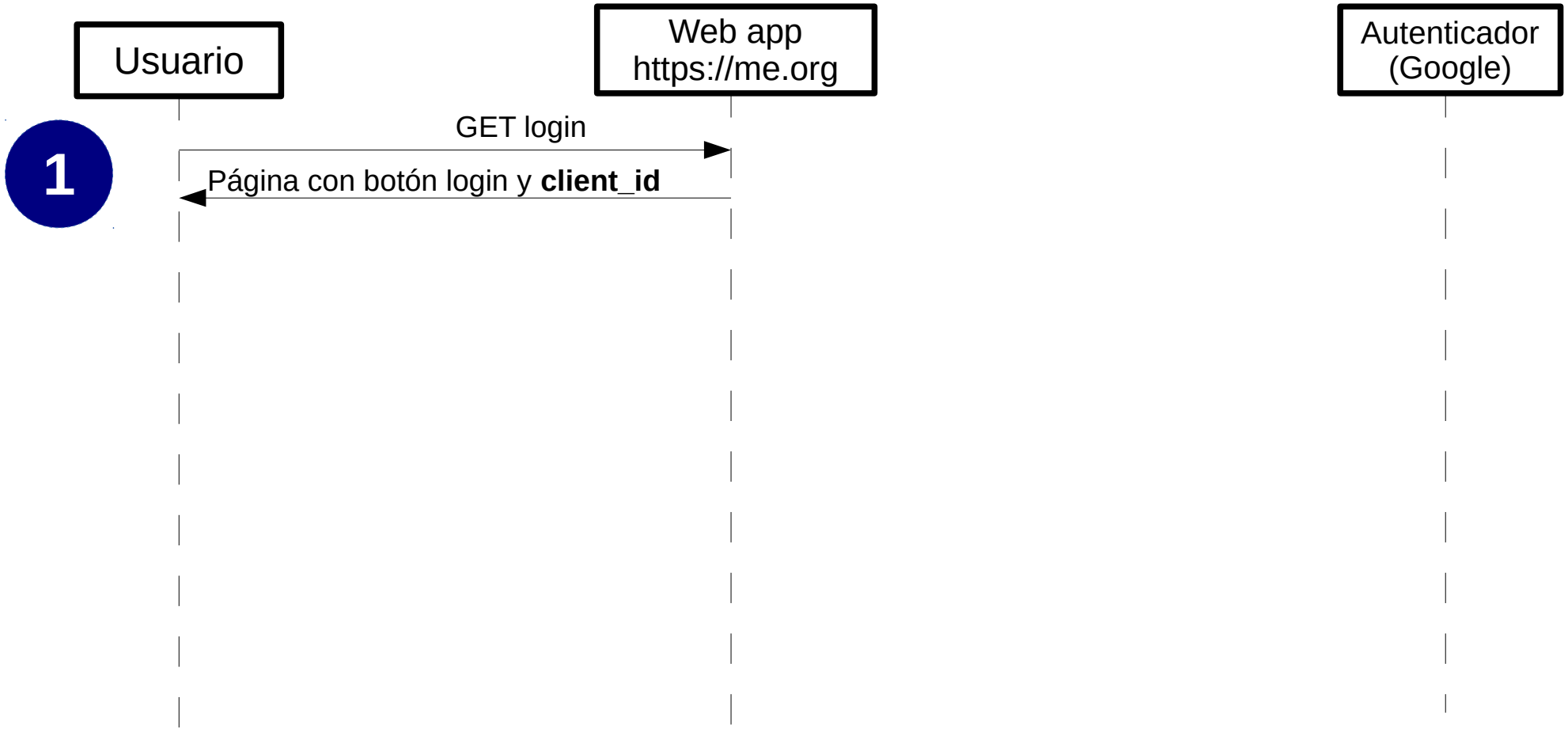
Autenticación usando Google

- Lo primero que hay que hacer es darse de **alta** como **desarrollador** en Google y dar de alta un proyecto.
- Todo esto se hace desde la **consola de desarrolladores** de Google: https://console.developers.google.com

Autenticación usando Google

- También se debe generar los **credenciales** para OAuth. Esto creará (entre otros):
  - **client_id**: identificador de mi aplicación web
  - **secret**: clave que solo conocen mi aplicación web y Google
  - **redirect_uri**: lista de URL de mi aplicación web a las que es legítimo que Google redirija al usuario tras un intento de *login*.

Autenticación: paso 1



Autenticación: paso 1

- Las páginas de la aplicación web que permiten autenticar contendrán un enlace al **punto de autorización** *(authorization endpoint)* de Google.
- La URL de este punto se obtiene del **documento de descubrimiento**. Es un JSON con distintas configuraciones necesarias para autenticar:
  https://accounts.google.com/.well-known/openid-configuration

  (Supondremos que el authorization endpoint es https://accounts.google.com/o/oauth2/v2/auth)

Autenticación: paso 1

- Ese enlace al punto de autorización contendrá varios parámetros:
  - **client_id**: para que Google sepa qué aplicación web es la que quiere solicitar autenticación.
  - **redirect_uri**: una de las dadas de alta en la consola del desarrollador.
  - **scope**: para expresar qué datos se solicitan del usuario, p.ej: **openid email** (solo e-mail). Dependiendo del *scope* Google mostrará diversos mensajes de advertencia sobre privacidad.

Autenticación: paso 1

- Suponiendo que client_id es 1234 y que redirect_uri es https://me.org/token el enlace completo quedaría como:

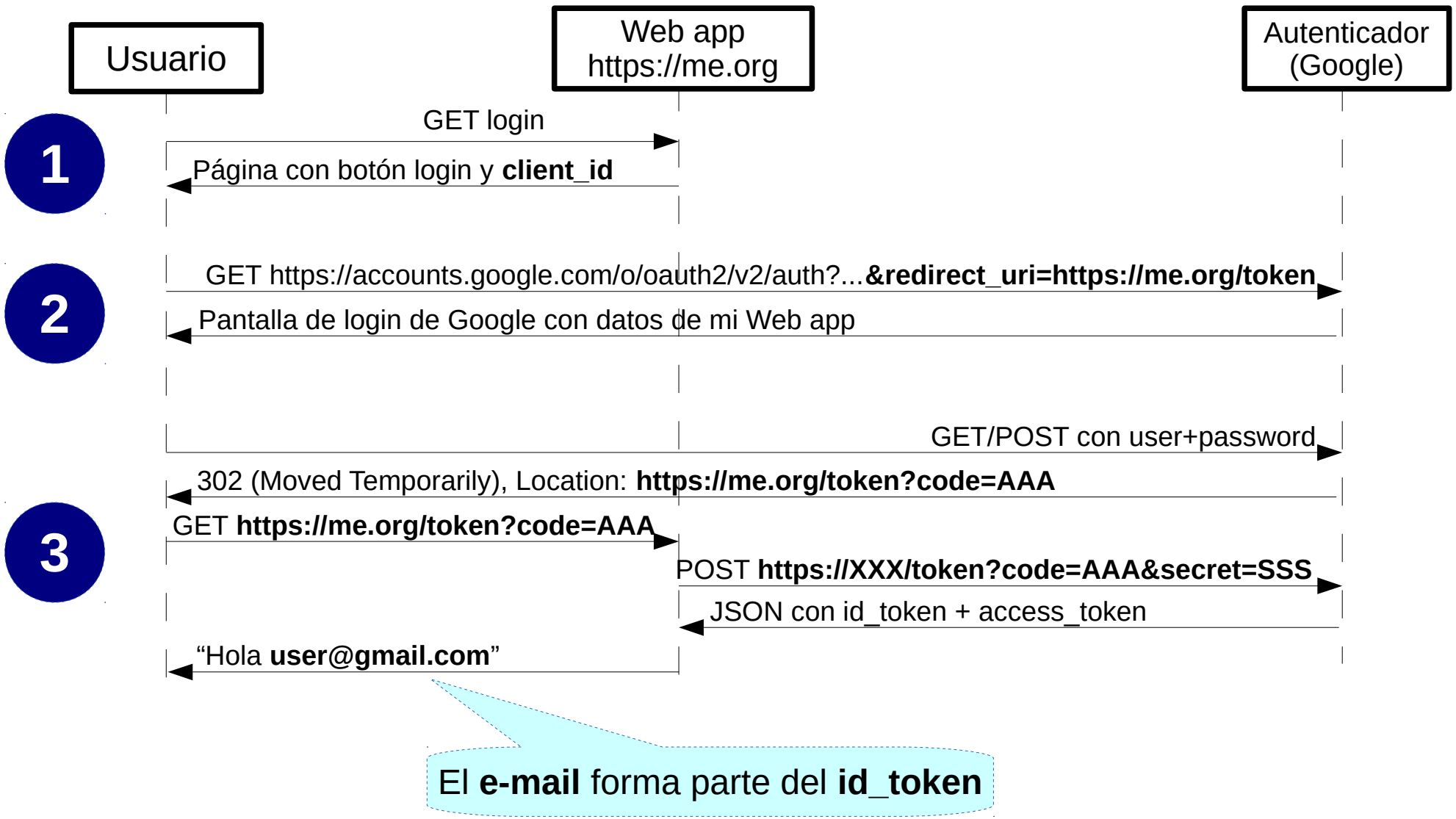
```
https://accounts.google.com/o/oauth2/v2/auth?
  client_id=1234&
  response_type=code&
  scope=openid%20email&
  redirect_uri=https://me.org/token
```

Autenticación: paso 2

Autenticación: paso 2

- El usuario pincha en el enlace, que le lleva a una página de Google
- Aquí tiene que introducir sus credenciales y dar permiso a la aplicación web para que acceda a los datos solicitados en el parámetro *scope*.
- Gracias al *client_id* Google podrá incluir información sobre la aplicación web: título, logotipo, enlace a términos y condiciones, etc.

Autenticación: paso 3



Autenticación: paso 3

- La tercera etapa de autenticación involucra más pasos:

  1) El usuario envía sus credenciales a Google

  2) Google genera un código temporal **AAAA** e informa al navegador que tiene que redirigirse a *redirect_uri* con dicho código **AAAA** como parámetro.

  3) La aplicación web intercambia el código **AAAA** por los dos tokens: *id_token* y *access_token*.

  4) La aplicación extrae la información sobre el usuario del *id_token*.

Autenticación: paso 3

- Es importante darse cuenta de que el **código** temporal **solo sirve para mi aplicación web**
  - Para intercambiar el código por los tokens se necesita proporcionar el secreto que solo mi aplicación web y Google conocen.
- El **id_token** es un objeto JSON **firmado y cifrado** con la clave privada de Google (JSON Web Token, JWT).
  - Para extraer sus datos es necesario usar el certificado digital público de Google.

Autenticación: paso 4

- Para verificar y descifrar id_tokens de manera remota consultad la documentación de Google: https://developers.google.com/identity/protocols/OpenIDConnect#validatinganidtoken
- Requiere conectar con: https://www.googleapis.com/oauth2/v3/tokeninfo

Autenticación: paso 4

- En una aplicación en producción descifraremos lo tokens en el propio servidor usando los certificados de Google:
  - https://github.com/googleplus/gplus-verifytoken-python
  - https://pypi.python.org/pypi/PyJWT
- Certificados de Google: campo “**jwks_uri**” del documento de descubrimiento.

Autenticación: paso 4

- En lugar de descifrar el id_token, también se puede usar el access_token para obtener los datos del usuario.
- Requiere una conexión a **userinfo_endpoint** del documento de descubrimiento, pasando el access_token como **cabecera** de autenticación:

```
Authorization: Bearer access_token
```

Referencias

Referencias

- Documentación oficial sobre autenticación con Google: https://developers.google.com/identity/protocols/OpenIDConnect
- Explicaciones paso a paso sobre las fases de la autenticación con Google (rutas, peticiones y parámetros involucrados): https://developers.google.com/identity/protocols/OpenIDConnect#server-flow
- Consola del desarrollador de Google: https://console.developers.google.com

Referencias

- Descifrado de JWTs de Google: https://developers.google.com/identity/protocols/OpenIDConnect#validatinganidtoken
- Documento de descubrimiento de Google: https://accounts.google.com/.well-known/openid-configuration