OAUTH 2.0

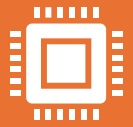
El marco de autorización para la ciberseguridad moderna (RFC 6749)

Por: Jhonattan Barriga, PhD, CEH

Las aplicaciones...

- Hoy en día las aplicaciones necesitan esquemas que faciliten el acceso a la información.
- ¿Confiarían en una app de terceros llamada IA-Image-Free que tiene 5 estrellas de calificación en la tienda de Google Play y necesita acceder a sus fotografías para mejorar la calidad y para ello el proceso les obliga a que entreguen su usuario de correo electrónico y su contraseña?

¿Qué es OAuth 2.0? – Propósito y objetivos



Marco de Autorización: No es un protocolo de autenticación. Permite a una aplicación de terceros obtener acceso limitado a un servicio HTTP protegido.

Acceso Delegado: Permite a un usuario otorgar acceso a sus recursos protegidos sin revelar sus credenciales a largo plazo (usuario y contraseña).

Tokens de Acceso: El cliente obtiene un token de acceso (cadena con alcance, vida útil) emitido por un servidor de autorización con la aprobación del propietario del recurso. Sugerencia de Visual: Un icono de un candado abierto con una flecha que apunta a un recurso, y un token en la flecha.

Importancia en la Ciberseguridad

- **Seguridad de API Robusta:** Utiliza tokens de acceso con alcance definido, asegurando que las aplicaciones solo tengan los permisos necesarios.
- **Principio de Privilegio Mínimo:** Los tokens son limitados y temporales, reduciendo el impacto de un compromiso.
- **Minimiza Riesgos:** Externaliza la gestión de credenciales a proveedores de identidad especializados (Google, Facebook, Okta), reduciendo la carga y el riesgo para las aplicaciones.

Oauth 2.0 vs. OpenID vs. SAML

OAuth 2.0: Se enfoca en la autorización (¿qué puedes hacer?). Delegar acceso a recursos protegidos (ej. app de fotos accede a Google Photos).

OpenID Connect (OIDC): Extiende OAuth 2.0 para la autenticación de usuario (¿quién eres?) y SSO. Introduce tokens de ID con información de usuario.

SAML: Estándar de autenticación para SSO. Se enfoca en la identificación, a diferencia de OAuth que se enfoca en permisos.

Roles clave de Oauth 2.0

Propietario del Recurso: Usuario final

Cliente: Aplicación que solicita acceso a recursos

Servidor de Recursos: Aloja recursos protegidos (API)

Servidor de Autorización: Autentica al propietario y emite tokens de acceso

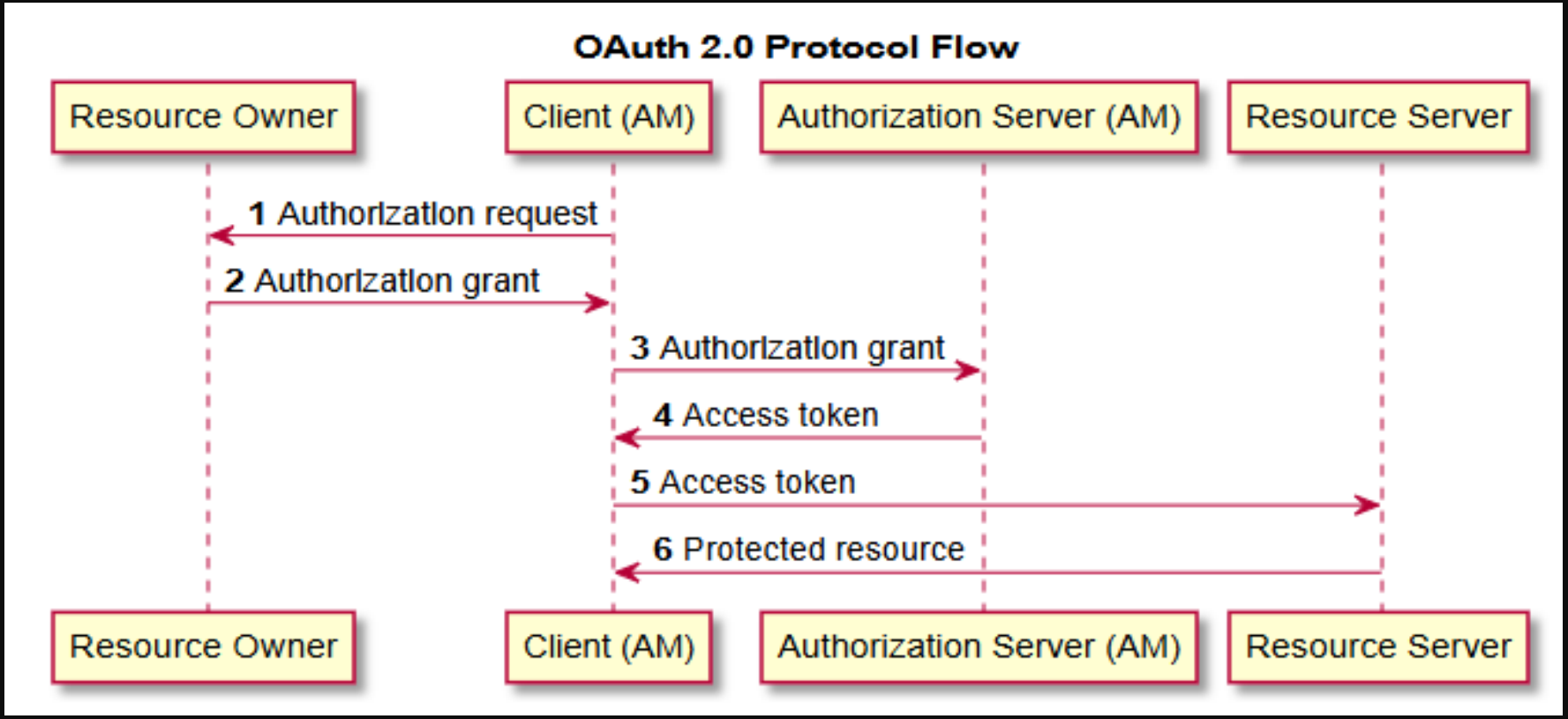
Terminología clave de Oauth 2.0

Concesión de OAuth 2.0 (Grant Type): La autorización otorgada al cliente por el usuario. Define el "flujo" para obtener un token.

Token de Acceso (Access Token): Credencial temporal emitida por el servidor de autorización para acceder a recursos protegidos.

Token de Actualización (Refresh Token): Token opcional de larga duración para obtener nuevos tokens de acceso sin que el usuario se vuelva a autenticar

Flujo OAuth 2.0

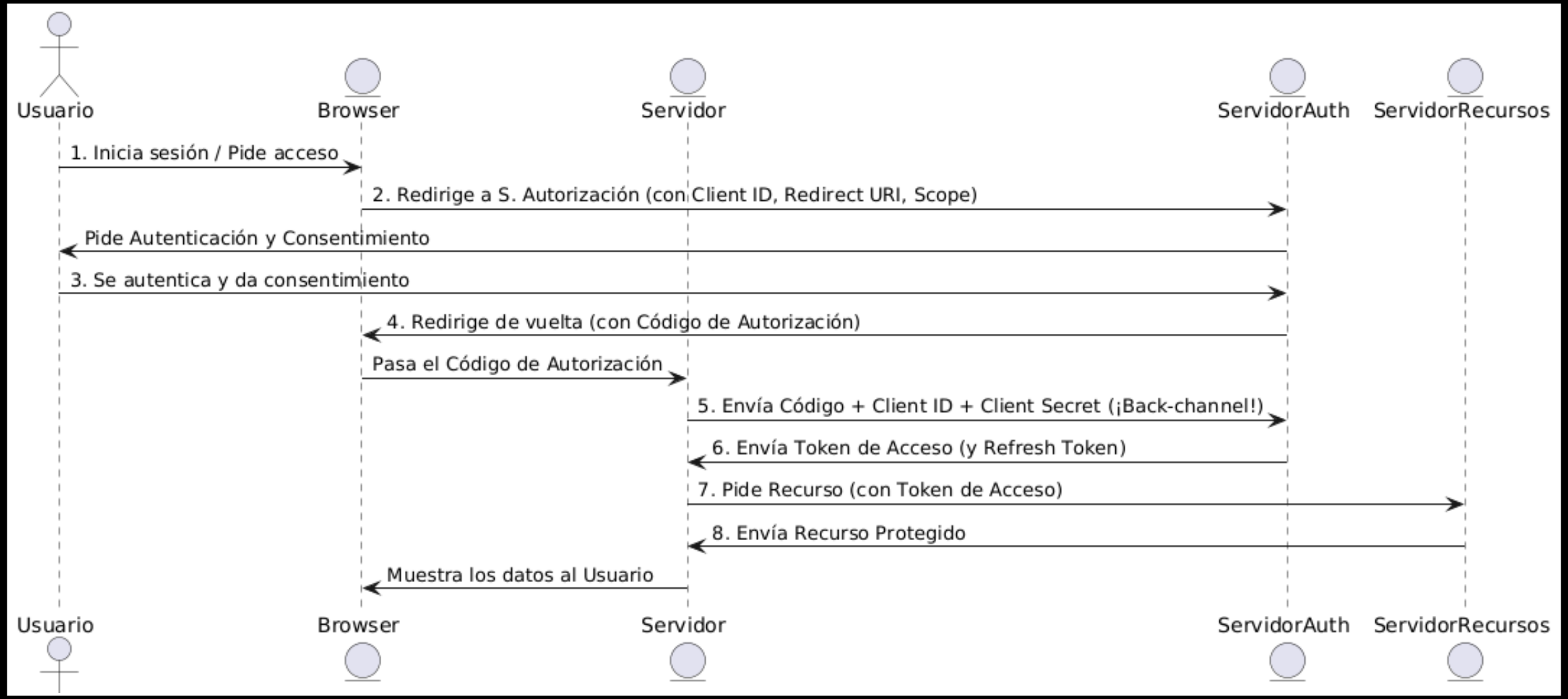


Flujos de Autorización (Tipos de Concesión)

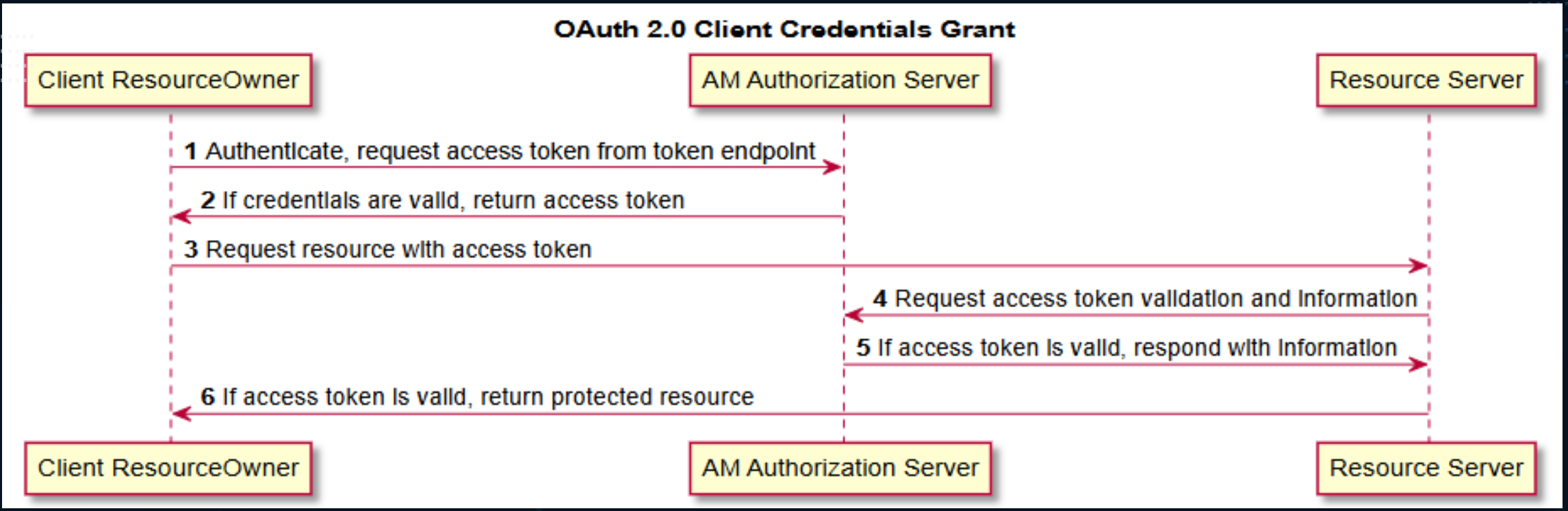
- OAuth 2.0 define varios "tipos de concesión" para obtener un token.
- La elección depende del tipo de aplicación y sus requisitos de seguridad.

El Flujo Más Seguro: Código de Autorización con PKCE

- **Recomendado:** El más seguro y ampliamente utilizado para clientes confidenciales y públicos (móviles, SPA).
- **Mecanismo:** El cliente obtiene un código de autorización (vía redirección del usuario), que luego se intercambia por un token de acceso en un canal seguro (servidor a servidor).
- **PKCE (Proof Key for Code Exchange):** Protege contra la inyección de código de autorización. El cliente genera un verificador de código aleatorio que se valida al intercambiar el código.



El Flujo Más Seguro: Código de Autorización con PKCE



Flujo de Credenciales de Cliente

- Uso: Comunicación máquina a máquina.
- El cliente se autentica directamente con el servidor para obtener un token.

Flujos a Evitar: Riesgos de Seguridad

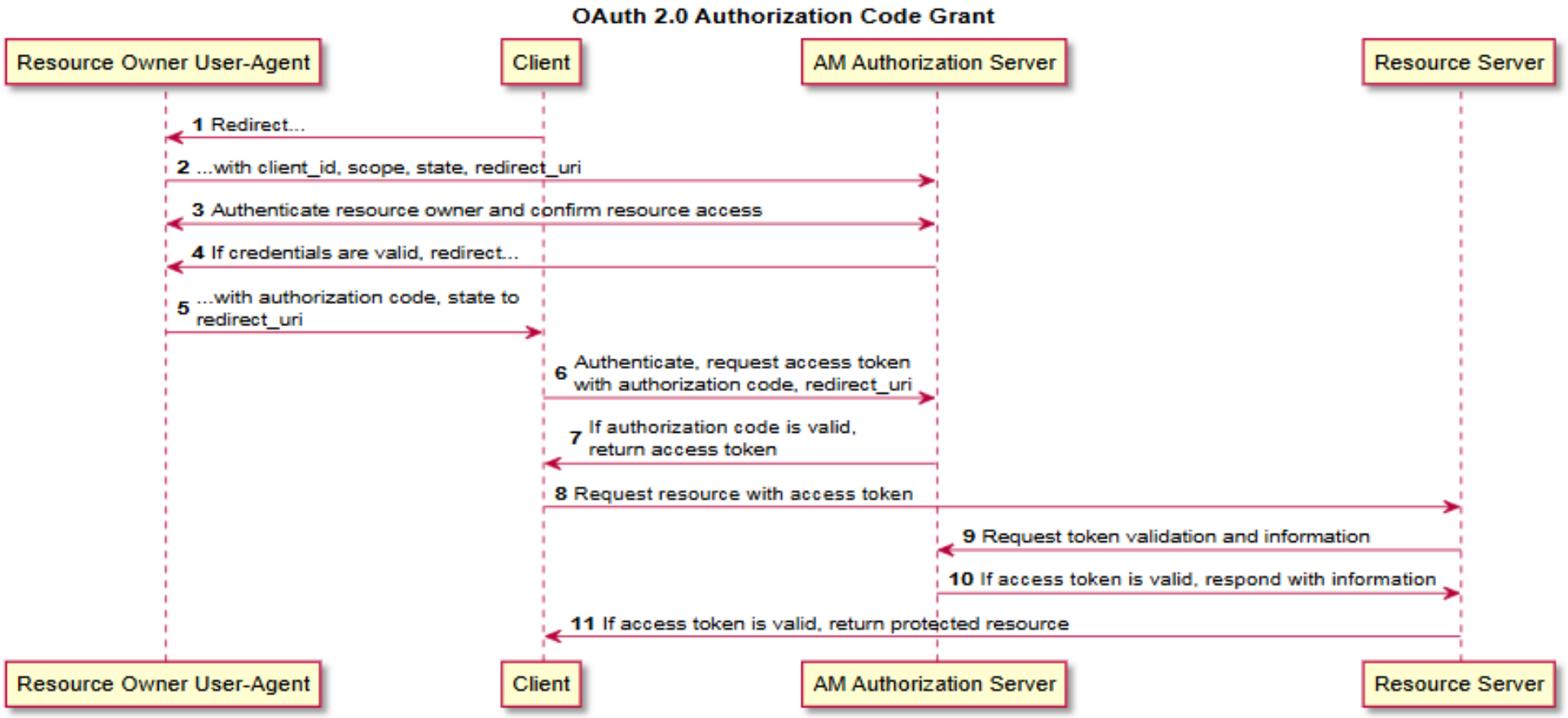
- Flujo Implícito: Obsoleto y menos seguro, expone tokens en URLs.
- Credenciales del Propietario: Cliente maneja directamente las credenciales del usuario.

Tipos de Concesión

Tipo de Concesión	Caso de Uso Ideal	Consideraciones de Seguridad
Código de Autorización (con PKCE)	Web, Móvil, SPA	**Altamente recomendado.**
Credenciales de Cliente	Máquina a máquina	Adecuado para servicios confiables.
Implícito	**Obsoleto/Desaconsejado**	Vulnerable a intercepción de tokens.
Credenciales de Contraseña	**Desaconsejado**	Cliente maneja credenciales sensibles.

Flujo de Autorización

1. **Solicitud de Autorización:** Cliente redirige al usuario al Servidor de Autorización.
2. **Usuario Otorga Autorización:** Servidor de Autorización responde con un código de autorización.
3. **Intercambio de Código por Token:** Cliente intercambia el código por un token de acceso (servidor a servidor).
4. **Servidor de Autorización Emite Token:** Servidor de Autorización entrega el token de acceso.
5. **Cliente Accede a Recursos:** Cliente usa el token de acceso para interactuar con el Servidor de Recursos.



Flujo de Autorización

Flujo de Autorización Detallado (I)

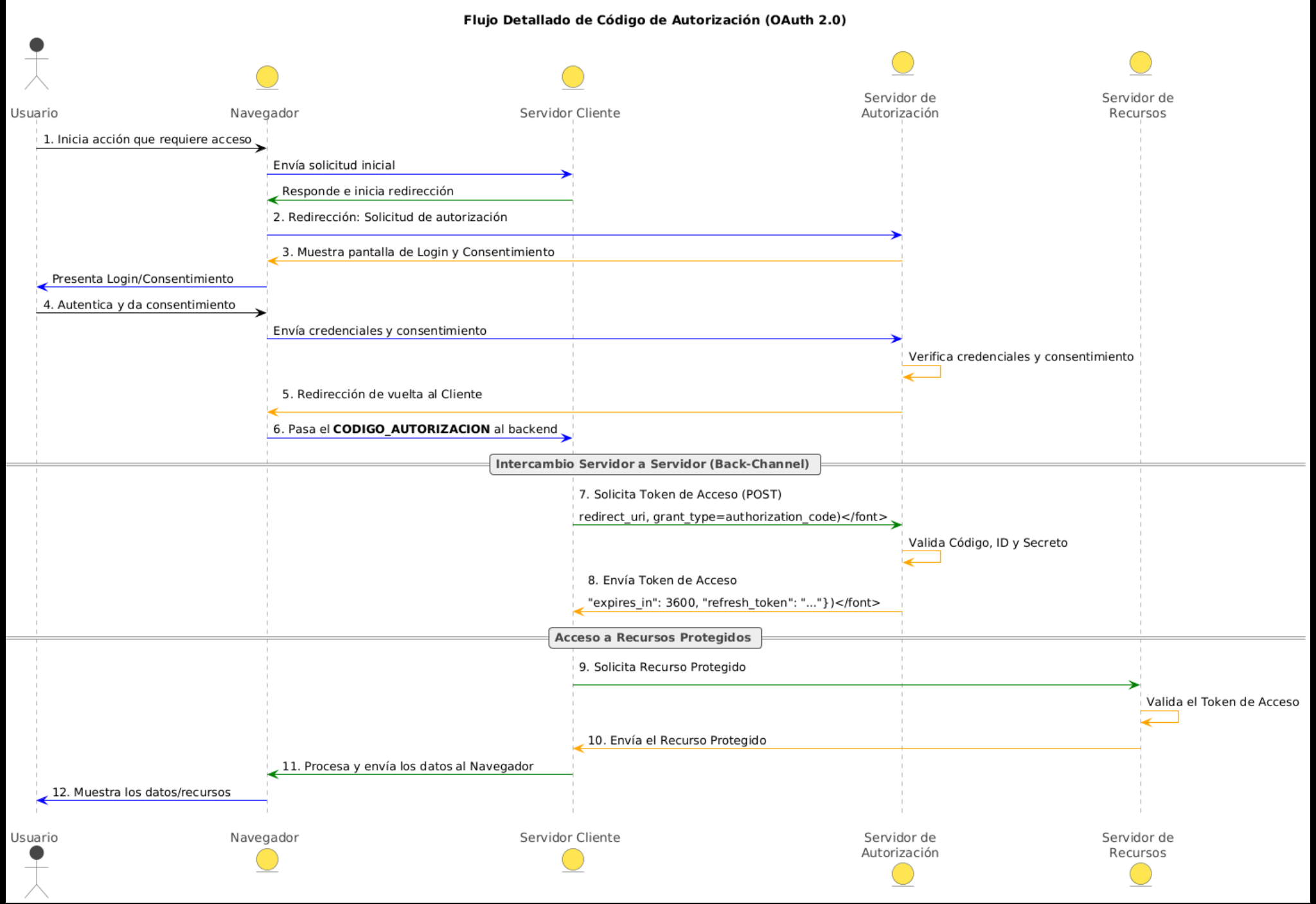
1. La aplicación cliente redirige el navegador del usuario al punto final /authorize del Servidor de Autorización. 4
2. Incluye client_id, redirect_uri, response_type=code, scope, state y code_challenge (PKCE). 4
3. El Servidor de Autorización muestra la página de inicio de sesión y la pantalla de consentimiento.
4. Si el usuario aprueba, el Servidor de Autorización emite un código de autorización temporal.
5. El Servidor de Autorización redirige el navegador del usuario de vuelta a la redirect_uri del cliente, incluyendo el código de autorización y el state.

Flujo de Autorización Detallado (II)

1. El backend de la aplicación cliente realiza una solicitud POST directa (servidor a servidor) al punto final /oauth/token del Servidor de Autorización.
2. Esta solicitud incluye el código de autorización, client_id, client_secret (para clientes confidenciales) y code_verifier (PKCE).
3. Este intercambio del lado del servidor es crucial para la seguridad, evitando la exposición del token en el navegador.
4. Si la solicitud es válida, el Servidor de Autorización responde con un token de acceso. [3]
5. Opcionalmente, se incluye un token de actualización (para acceso sin conexión) y un token de ID (si se usa OIDC).
6. Los tokens de acceso tienen una vida útil limitada.

Flujo de Autorización Detallado (III)

1. La aplicación cliente utiliza el token de acceso para realizar solicitudes autenticadas a las API del Servidor de Recursos. 3
2. El token de acceso se incluye típicamente en el encabezado Authorization de las solicitudes HTTP.


Flujo Detallado de Código de Autorización (OAuth 2.0)
Usuario
Navegador
Servidor Cliente
Servidor de Autorización
Servidor de Recursos
1. Inicia acción que requiere acceso
Envía solicitud inicial
Responde e inicia redirección
2. Redirección: Solicitud de autorización
3. Muestra pantalla de Login y Consentimiento
Presenta Login/Consentimiento
4. Autentica y da consentimiento
Envía credenciales y consentimiento
Verifica credenciales y consentimiento
5. Redirección de vuelta al Cliente
6. Pasa el CODIGO_AUTORIZACION al backend
Intercambio Servidor a Servidor (Back-Channel)
7. Solicita Token de Acceso (POST)
redirect_uri, grant_type=authorization_code)
Valida Código, ID y Secreto
8. Envía Token de Acceso
"expires_in": 3600, "refresh_token": "..."})
Acceso a Recursos Protegidos
9. Solicita Recurso Protegido
Valida el Token de Acceso
10. Envía el Recurso Protegido
11. Procesa y envía los datos al Navegador
12. Muestra los datos/recursos
Usuario
Navegador
Servidor Cliente
Servidor de Autorización
Servidor de Recursos

Aplicaciones comunes de OAuth 2.0

- **Integraciones con Redes Sociales:** "Iniciar sesión con Google/Facebook/Twitter".
- **Pasarelas de Pago y Comercio Electrónico:** PayPal, Stripe.
- **Acceso a Aplicaciones Empresariales:** Acceso seguro a herramientas internas y de terceros (ej. Google Workspace).
- **Autorización de Dispositivos IoT:** Gestión de dispositivos en hogares inteligentes (termostatos, cámaras).
- **Aplicaciones de Atención Médica:** Intercambio seguro de datos de salud (ej. apps de fitness con EHR).

Por: Jhonattan Barriga, PhD, CEH

Vulnerabilidad: Manipulación de la URI de Redirección

- **Problema**:
  - Una validación defectuosa de la redirect_uri permite a los atacantes redirigir códigos/tokens a sitios maliciosos.
- **Explotación**:
  - Elusión de listas blancas (ej. https://client-app.com/oauth/callback/../../evil.com).
  - Redirecciones abiertas o XSS en páginas de retorno. Sugerencia de Visual: Un diagrama que muestre una redirección maliciosa.

Vulnerabilidad: CSRF en Flujos de OAuth

- **Problema**:
  - Si el parámetro state falta, es predecible o no se valida, un atacante puede engañar a un usuario para que complete un flujo de OAuth sin su consentimiento. .
- **Explotación**:
  - Vinculación forzada de perfiles: La cuenta de la víctima se vincula a la cuenta del atacante.
  - CSRF de inicio de sesión: La víctima es engañada para iniciar sesión en la cuenta del atacante. Sugerencia de Visual: Un diagrama que muestre un ataque CSRF con el parámetro state ausente o incorrecto.

Vulnerabilidad: Almacenamiento inseguro de tokens

- **Problema**:
  - Los tokens de acceso y actualización se almacenan de forma inadecuada (texto plano, acceso público), permitiendo el acceso no autorizado si la aplicación es comprometida.
- **Explotación**:
  - Un atacante explota una vulnerabilidad (ej. inyección SQL) para extraer tokens y suplantar a los usuarios..

Vulnerabilidad: Validación Insuficiente del Alcance

- **Problema**:
  - El servidor de autorización o el servidor de recursos tienen una validación defectuosa, permitiendo a un atacante "actualizar" un token para obtener más permisos de los aprobados.
- **Explotación**:
  - Un atacante añade un parámetro *scope* más amplio a la solicitud de intercambio de tokens.

Por: Jhonattan Barriga, PhD, CEH

Mejores prácticas de seguridad en OAuth 2.0

- ✓ Validación Estricta de redirect_uri: Usar listas blancas estrictas y del lado del servidor.
- ✓ Uso Efectivo del Parámetro state: Generar un token único y aleatorio por solicitud, vincularlo a la sesión.
- ✓ Priorizar Código de Autorización con PKCE: Para todos los clientes, especialmente SPA y móviles.
- ✓ Almacenamiento Seguro de Tokens: Cifrar tokens, usar cookies HttpOnly y Secure.
- ✓ Principio de Privilegio Mínimo: Solicitar solo los permisos mínimos necesarios (scope).
- ✓ Uso de HTTPS en Todas Partes: Cifrar todos los datos en tránsito.
- ✓ Aprovechar Bibliotecas Cliente Oficiales: Reducir errores de implementación.

Vulnerabilidades y Mecanismos de Mitigación

Vulnerabilidad	Mitigación Clave
Almacenamiento Inseguro de Tokens	Cifrar tokens; cookies HttpOnly/Secure.
Manipulación de redirect_uri	Lista blanca estricta; evitar comodines.
CSRF en Flujos de OAuth	Usar y validar parámetro state único.
Validación Insuficiente del Alcance	Aplicar privilegio mínimo; validar alcance en servidor de recursos.
Uso del Flujo Implícito	Preferir Código de Autorización con PKCE.

Por: Jhonattan Barriga, PhD, CEH

Conclusiones

- **OAuth 2.0 es Poderoso:** Marco de autorización esencial para el acceso delegado seguro.
- **La Implementación es Crítica:** La seguridad depende de una implementación correcta y diligente.
- **Priorizar Flujos Seguros:** Siempre Código de Autorización con PKCE.
- **Adherirse a Mejores Prácticas:** Validación estricta, state, almacenamiento seguro, privilegio mínimo, HTTPS.
- **Vigilancia Continua:** Mantenerse informado sobre nuevas amenazas y mejores prácticas

Muchas Gracias

¿Preguntas?

- https://backstage.forgerock.com/docs/am/6/oauth2-guide/
- https://datatracker.ietf.org/doc/html/rfc6749
- https://oauth.net/2/