# Criptografía y Seguridad

## Seguridad en aplicaciones: Control de acceso

# Matriz de control de acceso

- Modelo más simple
  - Una fila por usuario
  - Una columna por objeto
  - Acciones permitidas en la intersección

| | Obj1 | Obj2 | Obj3 | ... | ... | ... | Obj n |
|---|---|---|---|---|---|---|---|
| Usuario1 | r | r | r x | | | | r w x |
| Usuario2 | | o x | | | | | |
| ... | | | | | | | |
| ... | | | | | | | |
| Usuario n | o x | | | | | | |

La O significa Ownership. Desde el punto de vista teorico nosotros podriamos representar todas las interacciones de un sistema a partir de una matriz, como vimos en Bell-LaPadula (condicion simple + condicion de cierre pero ademas aceptaba permisos discrecionales).
En Bell-LaPadula, como los permisos discrecionales solo sirven para restringir mas a los permisos mandatorios, se la conoce como matriz complementaria.
Sin embargo, en un sistema es raro que veamos una matriz de control de accesos, porque son muy ineficientes (dado que tiene tamaño NxM, enumera por extension a todos los sujetos y objetos de un sistema). Ademas ocurre que se arman islas en el espectro de usuarios y espectro de objetos, en donde muy pocos lugares tienen valores. Entonces lo que se hace es modelar esta misma informacion pero de manera distinta, como un ACL.

# Características

- Permite implementar cualquier política
- Rápido crecimiento
  - Ejemplo: 100.000 archivos, 500 usuarios
  - = 50.000.000 de entradas
- Desperdicio de espacio
- Facilidad para determinar accesos
- Complejidad para soportar altas y bajas
  - Cambia la estructura de la matriz
- Administración compleja (elemento a elemento)

# Listas de Control de Acceso (ACL)

- Representan columnas de una matriz de acceso

| | Obj1 | Obj2 | Obj3 | ... | ... | ... | Obj n |
|---|---|---|---|---|---|---|---|
| Usuario1 | r | r | r x | | | | r w x |
| Usuario2 | | o x | | | | | |
| ... | | | | | | | |
| ... | | | | | | | |
| Usuario n | o x | | | | | | |

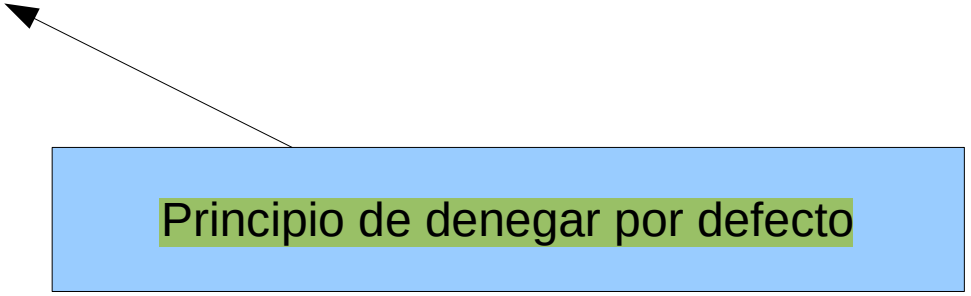
**ACLs:**

Obj1: { (usuario1, r), (usuario n, ox) }
Obj2: { (usuario1, r), (usuario2, ox) }
Obj3: { (usuario1, rx) }
...

Cada objeto guarda todos los usuarios que pueden accederlos de alguna forma. Cuando la matriz de control de accesos es muy dispersa, las listas de control de accesos son mas eficientes. Cuando un objeto no tiene a un usuario, entonces este usuario no tiene ningun tipo de permiso sobre este objeto.

# Definición

- Sean
  - S conjunto de sujetos
  - O conjunto de objetos
  - R conjunto de acciones (derechos)
- ACL(o) = { $(s_i, r_i)$ / s ε S, r ε R }
  - $(s_i, r_i)$ implica que $s_i$ accede a o con cualquier derecho de $i_r$
- Si un sujeto no tiene entrada en ACL(o) no tiene ningún derecho



# Modifcación de ACLs

- Derecho de pertenencia (own)
  - Puede ser asignado a quien crea el objeto
  - Puede ser asignado según el tipo de objeto
- Transferencia de permisos
  - Traspaso: cambiar el sujeto de una entrada de ACL
    - (Usuario 1, t rwx) → (Usuario 2, t rwx)
  - Delegación (El mecanismo se llama ownership, es usado por herramientas como google drive para multiples usuarios)
    - (U1, t rx) → (U1, t rx) , (U2, rx)
    - (U1, t rx) → (U1, t rx) , (U2, r)

Opcion 1: se permite asignarle a otro usuario ownership manteniendo el atributo original (permitiendo mas de un owner)
Opcion 2: otorgar el ownership a otro usuario, te quita el ownership (permitiendo un solo owner)

(por ejemplo, a veces existe un administrador que puede cambiar los permisos del resto. En otros casos, se pueden cambiar los permisos de un objeto desde dentro del sistema. Esto se modela con la relacion de OWN, que permite al sujeto cambiar el ACL de ese objeto)

Dentro de los paradigmas de delegacion, hay dos opciones:
1. El owner puede asignarse todos los permisos sobre el objeto que quiera (aunque no los tenga), y se los puede asignar a otro usuario
2. El owner solo puede delegar permisos que tiene en el momento

# Usuarios privilegiados

- Usuarios a los cuales no aplican los ACLs
  - Ejemplo: root en Linux → tiene todos los accesos

Sirven para recuperar el sistema. Sin embargo, el root esta fuera del esquema de permisos, por lo que si un atacante obtiene acceso a root, entonces es muy dificil arreglar esto con esquemas de seguridad.

- Usuarios con ACLs especiales
  - Ejemplo: administrator en Windows 200x
  - Tiene el derecho (take ownership) sobre todos los objetos

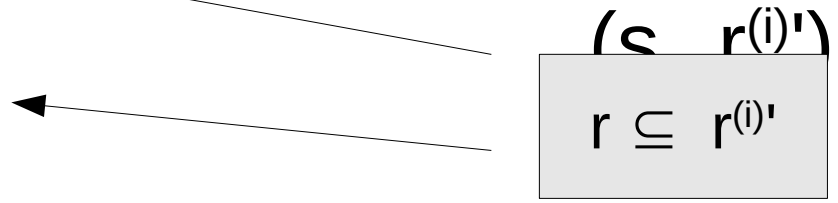
La diferencia con esto es que los usuarios que pertenezcan al grupo de administracion tienen el derecho take-ownership a todos los objetos del sistema. Esto significa que todos los objetos siguen estando regulados por el mismo mecanismo de seguridad.
Como ocurre todo dentro del sistema, todo es auditable (contrario al caso en linux).
La desventaja es que esto es mas dificil de modelar que el sistema de linux.

Cualquier sistema tiene alguna de estas dos opciones, para salvarse ante casos como transferencia de permisos accidentales y otras cosas.

# Manejo de volumen: Grupos

- Aunque menos que la matriz, los ACLs pueden crecer mucho
- En la práctica, se utilizan grupos (roles)
  - No pertenecen al modelo clásico de ACLs
- $g = \{ s_1, s_2, s_i / s \in S \}$
- $(g, r) \subset ACL(o) \Leftrightarrow (s_1, r') \subset ACL(o) \wedge \ldots \wedge$

Se agrupa a los sujetos en roles, y se pueden asignar permisos a nivel rol en vez de a nivel usuario. Ejemplo: grupo de administradores, grupo de contadores, etc. El objetivo de trabajar con grupos es la eficiencia, entonces en vez de tener 100 ACLs iguales para 100 objetos, entonces tenemos un solo rol.




$\subset ACL(o)$

- Reducen tamaño de ACLs
- Agregan una nueva dimensión de complejidad

# Grupos y usuarios - Conflictos

- Más de un AC pueden aplicar al mismo tiempo
- Escenario
  - ACL(o) = { (Pablo, rw), (Profesores, r) }
  - Pablo ∈ Profesores

Yo tengo acceso de escritura a un objeto, y mi rol no lo tiene

- Grant-All
  - Requiere que todos los ACs otorguen el derecho

En el ejemplo, Pablo puede leer, pero no escribir o

- First-Rule
  - El primer ACL encontrado es utilizado

Requiere además, definir un orden de evaluación de ACLs

El orden estandarizado es que primero se analizan los ACLs de usuario y luego los de su rol, pero esto puede cambiar.

En el ejemplo, Pablo puede leer y escribir

# Derechos por defecto

- Permiten definir ACs a utilizar sobre sujetos para los cuales no hay un AC en la lista para usuarios nuevos
- Escenario
  - ACL(o) = { (Pablo, x) (*, r) }

Por ejemplo, todos los usuarios (antiguos y nuevos) pertenecen a un grupo de todos, que tienen algunos permisos default. En este caso, el por defecto permite que pablo lea el archivo O.

- Forma 1: Override
  - Si el sujeto no tiene un AC, utilizarlo. Sino el default Cualquier AC en particular overriadea al AC por defecto

En el ejemplo, Pablo puede ejecutar o

- Forma 2: Augment Permisos especificos modifican a los por defecto
  - Partir del default y agregar ACs explicitos

En el ejemplo, Pablo puede leer y ejecutar o

# Revocación

- ¿Como quitar derechos de un sujeto a un objeto?
  - El dueño quita el AC del sujeto del ACL
  - O quita el/los derechos necesarios dentro del AC
- Si hay transferencia de permisos
  - Mucho más complejo
  - Puede requerir borrar en cascada permisos delegados

En la revocacion, hay muchos casos borde como que pasa si un administrador borra a todos los administradores del sistema incluido a si mismo, o si borra todos los ACL

En la mayoria de los sistemas, la revocacion deberia surgir en cascada

S1 delega en S2. S2 delega en S3
S1 revoca a S2. Acto seguido S3 delega en S2

# Ejemplo: Windows (archivos)

- Derechos: leer, escribir, ejecutar, borrar cambiar permisos, tomar control

Los SOs tienen los sistemas de control de accesos mas complejos de todos, porque tienen que servir para usos totalmente distintos (como gaming y servidores).
Esta lista de permisos no estan todos al mismo nivel, sino que hay algunas que son mas importantes que otras: como cambiar derechos vs leer

- ACLs:

Notar que tomar control y cambiar permisos son cosas distintas, esto hace que surjan nuevos casos borde

  - Incluyen usuarios y grupos
  - Tres posibles estado por derecho
    - Otorgado (+)
    - No asignado ( )
    - Revocado (-)

Windows decidio complejizar el sistema para que se pueda implementar tanto GRANT ALL como FIRST RULE.
Por esto es que se puede tanto otorgar como revocar permisos.

  - Los derechos están agrupados: No access (todos-), read (leer+, ejecutar+), change(leer+,escribir+,ejecutar+,borrar+), full control (todos +)

# Ejemplo: Windows (archivos)

- Acceso a un archivo
  1. Buscar todos los ACs en el ACL del archivo que referencien al usuario y a sus grupos
  2. No hay ningún AC → denegado
  3. Acceso revocado en un AC → denegado
  4. Al menos un AC permite el derecho → aceptado
  5. denegado

# Listas de Capacidades

- Representan filas de una matriz de acceso

| | Obj1 | Obj2 | Obj3 | ... | ... | ... | Obj n |
|---|---|---|---|---|---|---|---|
| Usuario1 | r | r | r x | | | | |
| Usuario2 | | o x | | | | | |
| ... | | | | | | | |
| ... | | | | | | | |
| Usuario n | o x | | | | | | |

**CAPs:**

usuario1: { (obj1, r), (obj2, r), (obj3, rx) }
usuario2: { (obj2, ox) }
usuario n: { (obj1, ox) }
...

En capacidades, es mucho mas natural la transferencia de permisos.
Esto se debe a que se habla de capacidades sobre un objeto, y se las puede mover facilmente de un lado a otro.
El protocolo open-auth nos permite autorizar a un usuario para que tenga capacidades de un sistema externo tambien.

# Definición

- Sean
  - S conjunto de sujetos
  - O conjunto de objetos
  - R conjunto de acciones (derechos)
- CAP(s) = { $(o_i, r_i)$ / o $\in$ O, r $\subseteq$ R }
  - $(o_i, r_i)$ implica que s accede a $o_i$ con cualquier derecho de $i_r$
- s no tiene derechos sobre objetos que no estén en CAP(s)

Principio de denegar por defecto

# Funcionamiento

- Las capacidades funcionan como entradas
  - La posesión indica derechos del sujeto sobre el objeto
  - Ejemplo: La capacidad (o1, rwx) puede ser presentada por diferentes sujetos
- A diferencia de los ACLs, el sistema no controla estos datos
- Deben implementarse mecanismos de protección
  - Evitar que un usuario altere Capacidades
  - Evitar que un usuario cree Capacidades

# Implementación

Hay 3 mecanismos de implementacion

- Tags

(etiquetas) Esta medio en desuso. Se puede etiquetar una zona de hardware (donde estan las capacidades por ejemplo) tal que el resto de los usuarios no pueda leerlo. Necesitamos hardware especializado para hacer esto.

  - Marcas de bits controladas por hardware que impiden la modificación de registros en procesos de bajo privilegio

- Paging / segmentos protegidos

Cuando nosotros abrimos un archivo con fopen(), nos devuelve un puntero a una estructura (FILE *) con ciertos datos de un archivo, como un file descriptor y otros datos. El file descriptor es un indice (un numero que indica un lugar en una tabla). Ese FD termina siendo una capacidad, ya que teniendo ese FD podemos acceder al archivo. Entonces, el syscall OPEN crea una capacidad

  - Las capacidades se almacenan en un segmento de memoria marcado como de solo lectura
  - Los procesos acceden indirectamente a dichas capacidades
    - Sino se podrían copiar las capacidades
  - Ejemplo: Descriptores de archivo en Linux

# Implementación

- Criptografía
  - Asociar a cada capacidad un hash criptográfico cifrado con una clave conocida solo por el sistema
  - Al presentar la capacidad, el sistema verifica el hash

# Controles

- Control de copia

  Generalmente se le agrega metadata a la capacidad, que permita asociarla a un usuario o una transaccion.

  - Como la tenencia de una capacidad implica acceso, se debe restringir su copia
  - Acceso indirecto a las capacidades
  - Copia controlada por el sistema
- Amplificación

  Es un problema comun en sistemas operativos. Por ejemplo, abrir un archivo requiere ciertos pasos de lectura que un usuario normal no deberia poder hacer.

  - Posibilidad de contar con capacidades extendidas temporalmente
  - Al ejecutar ciertas funciones
    - Ejemplo: User mode vs Kernel mode

# Revocación

- Implica revisar todas las listas de capacidades
  - Demasiado costoso
  - A veces imposible (sistemas remotos)
- En la práctica: indirección
  - Las capacidades son indices dentro de una tabla que no es accesible a los procesos
  - Se invalida la entrada correspondiente en la tabla

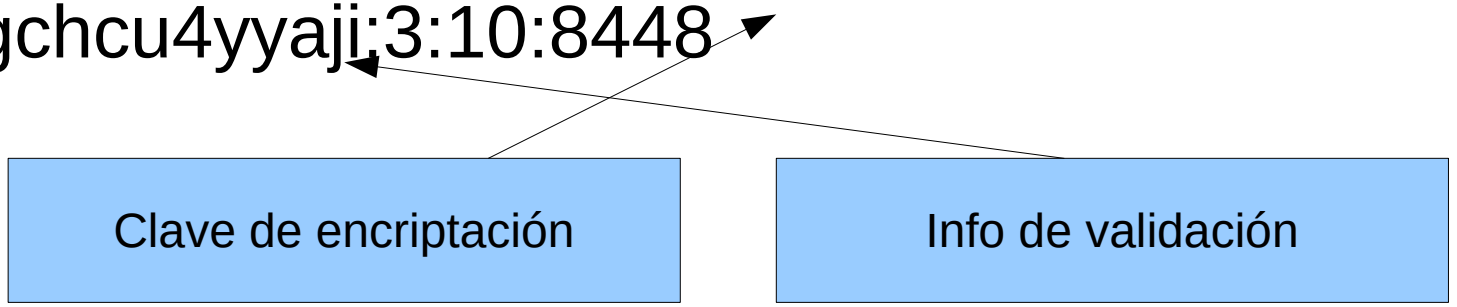
En estos casos a veces tenemos que mantener listas de revocacion, o tener capacidades con cierto lifetime

# Ejemplo de Capacidades

- Sistema distribuido de archivos Tahoe
- Para acceder a un archivo se requiere una capacidad
- Hay capacidades de escritura, lectura y verificación:

Cada archivo que se guarda en el filesystem, termina teniendo una URI como esta

  - URI:CHK:6hwdguhr5dvgte3qhosev7zszq:lgi66a5s6gchcu4yyaji:3:10:8448



# ACLs y Capacidades

- Ambos modelos son teoricamente equivalentes
- ACLs: dado un objeto ¿Quienes pueden usarlo y como? Es lo mas comun que nos vamos a encontrar
  - Asociado a procesamiento imperativo
  - Historicamente, el más desarrollado
  - Ejemplo: Windows / Linux
- Capacidades: dado un sujeto ¿Que objetos puede acceder y como?
  - Asociado a procesamiento declarativo
  - Ejemplo: IDSs - Sistemas de respuesta de incidentes Aparece mucho en herramientas de seguridad, como firewalls

# Secretos compartidos

- Implementación de políticas de separación de privilegios
- Metodo (t,n)-threshold
  - Se crean n partes o n capacidades (sombras)
  - Cualesquiera t permiten acceder al objeto
  - Cualesquiera k < t NO permiten acceder al objeto
- Implementación
  - Control via sistema
  - Criptografía: threshold cryptography

Un secreto compartido es una operacion tan critica que no queremos que un solo usuario la pueda ejecutar (como lanzar misiles nucleares, que requiere de dos personas).
Esto se puede hacer con reglas discrecionales, pero tambien se puede hacer con criptografia (con el metodo t,n threshold)

# Metodo de Shamir

- Construcción de un método (t,n) threshold
- Principio:
  - Un polinomio de grado t puede ser especificado mediante su evaluación en t puntos diferentes
- Construcción:
  - Armar $P(x) = a_t x^t + a_{t-1} x^{t-1} + .. + a_1 x + a_0 \mod p$
  - Sea s el secreto a compartir.
  - $p > s$, $p > n$, $a_0 = s$
  - P(1), P(2), … P(n) son las sombras

Si yo tengo un polinomio de grado t, lo puedo definir univocamente evaluandolo en t puntos distintos. La idea del metodo de shamir, es que para armar un criptosistema de umbrales, tengo que trabajar con un polinomio de grado t. Lo invento con coeficientes aleatorios (por ejemplo 1, 2, 3, 4, 5, ...) y obtengo n pares. Mantengo en secreto el coeficiente constante, que representa el secreto a proteger.
Con estos pares de valores (llamado sombras), puedo usar el metodo de interpolacion de lagrange para reconstruir un polinomio a traves de una serie de puntos. Si este polinomio lo evaluamos en 0, recuperamos el secreto (que era el coeficiente constante). Sin embargo, si tengo menos de n puntos, entonces hay infinitos polinomios que cumplen, y no hay manera de recuperar el coeficiente secreto. Este sistema es seguro porque tampoco me da informacion parcial.

# Metodo de Shamir

- Reconstrucción del secreto
  - Sean ($s_{i1}$, $s_{i2}$, $s_{it}$) / $s_{i1}$ = P(i1) , t sombras cualesquiera
  - Se interpola el polinomio (metodo de Lagrange)

$$P(x) = \sum_{a=1}^{t} \left( s_{i_a} \prod_{b=1, b \neq s}^{t} \frac{x - i_b}{i_a - i_b} \right)$$

  - Se evalúa P(0) = s

# Ejemplo

- Secreto a compartir: s= 7, esquema (3,5)
- $P(x) = 5x^2+3x+7 \bmod 11$
- Sombras:
  - P(1) = 5 + 3 + 7 mod 11 = 4
  - P(2) = 20 + 6 +7 mod 11 = 0
  - P(3) = 45 + 9 + 7 mod 11 = 6
  - P(4) = 80 + 12 + 7 mod 11 = 2
  - P(5) = 125 + 15 +7 mod 11 = 4
  - Sombras: (1,4), (2,0), (3,6), (4,2), (5,4)

# Ejemplo

- Sombras: (1,4), (2,0), (3,6), (4,2), (5,4)
- Tomando tres cualesquiera:
  - (2,0) (3,6) (5,4)

$$P(x)=\sum_{a=1}^{t}\left(s_{i_a}\prod_{b=1,b\neq s}^{t}\frac{x-i_b}{i_a-i_b}\right)$$

$$P(x)=\left[0\frac{(x-3)(x-5)}{(2-3)(2-5)}+6\frac{(x-2)(x-5)}{(3-2)(3-5)}+4\frac{(x-2)(x-3)}{(5-2)(5-3)}\right]$$

$$P(x)=\left[0-3(x^2-7x+10)+(4/6)(x^2-5x+6)\right] mod\ 11$$

$$P(x)=5x^2+3x+7$$

$s=P(0)=7$

Este ejemplo es corto, pero con polinomios mas grandes, este secreto (7) puede ser por ejemplo la clave criptografica necesaria para autorizar cierta capacidad

Intentar con (1,4), (3,6), (4,2)

# Oauth (2.0)

- Open Standard for Authorization
- Comienza en 2006 por necesidades de Twitter
- Oauth 1.0 – Publicado en 2010
  - RFC 5849
  - Estructura similar a OpenID
- Oauth 2.0 – Publicado en octubre 2012
  - Mucho mas simple que la versión anterior
  - No es compatible
  - Es el estándar de facto para sitios públicos

Oauth2 es mucho mas simple de oauth1, ya que este ultimo requeria de un canal seguro, y oauth2 simplemente le delega este aspecto a HTTPS

# Oauth 2.0

- Permite al dueño de un recurso delegar en una aplicación el acceso al mismo
- Roles:

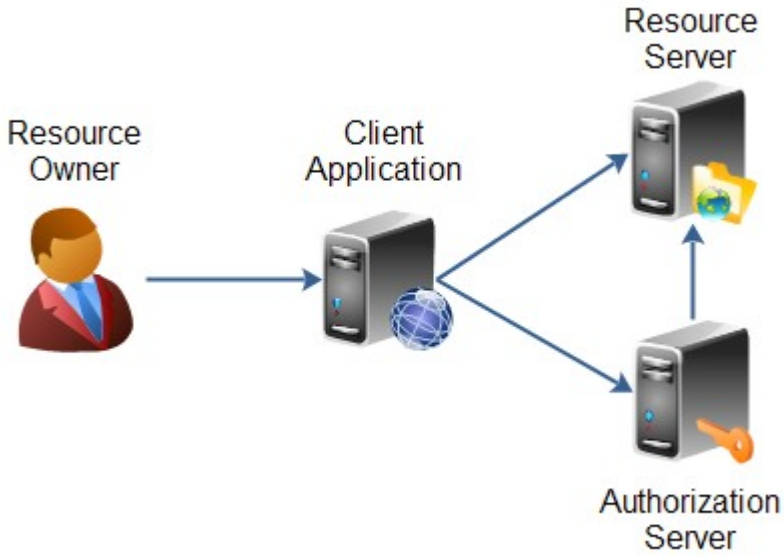



Imagen obtenida de: http://tutorials.jenkov.com/oauth2/

# Oauth 2.0 – Dance

Hacer un sistema de control de accesos es tan complicado que generalmente se usan herramientas de terceros

Nosotros tenemos un cliente que se comunica con una aplicacion. La aplicacion redirige al usuario a una pagina especial del proveedor. En esta pagina, el cliente se autentica. Luego el proveedor redirige al usuario a la aplicacion con un access token. Cuando la aplicacion presenta el access token al proveedor, tiene que ser valido. Finalmente el cliente ya esta logueado.

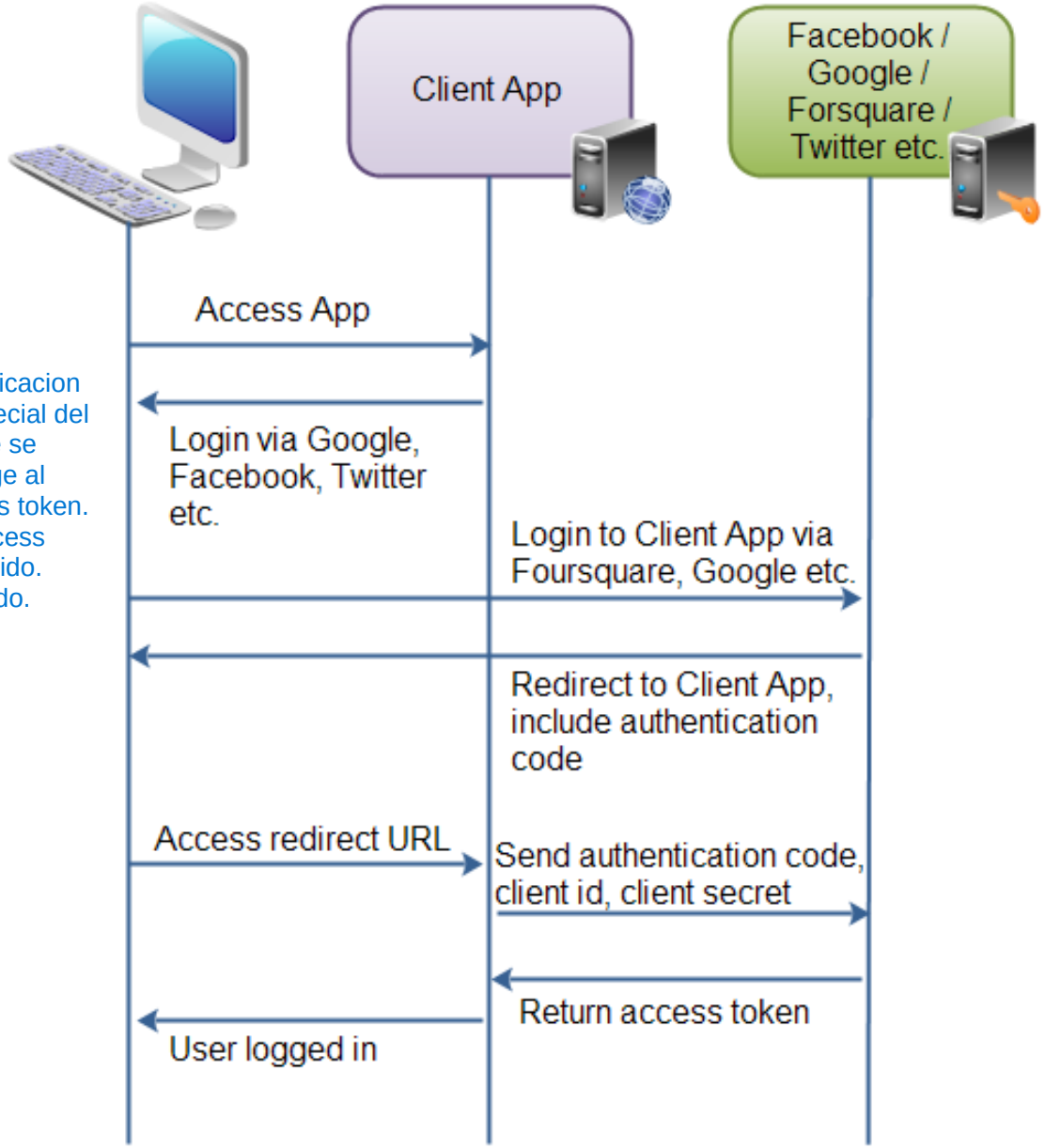



Imagen obtenida de: http://tutorials.jenkov.com/oauth2/

# Oauth 2.0 – Tipos de clientes

- Confidencial
  - Típicamente una aplicación en un servidor
  - Puede mantener un secreto compartido con el servidor Por ejemplo, una aplicacion web
    - Client-secret
- Publico
  - Típicamente una aplicación desktop, movil o que corre en un navegador
  - No puede mantener un secreto de forma confiable Por ejemplo, una aplicacion movil. Esta no puede mantener ningun secreto, ya que se puede desensamblar.

# Información del cliente

- Los clientes deben registrarse por adelantado
- Cada cliente requiere:
  - Client ID – Identificador único
  - Client Secret – Secreto compartido
    - Solo para clientes confidencials
  - Redirect URI(s)
    - Lista de direcciones validas a donde redirigir los accesos

Esto solo tiene sentido si el secreto no puede ser descubierto por el usuario final. Entonces, si hablamos de una aplicacion movil, no podemos usar esto.

# Oauth 2.0 – Grant types

OAuth2 define 4 flujos distintos

- Authorization Code
  - El servidor de autorización retorna un código
  - El cliente consigue el access token con el código, su client id y su client secret.
- Implicit
  - El servidor de autorización retorna directamente el access token

    Esta pensado para clientes publicos. Si no va a haber alguien de por medio que pueda mantener un secreto, le damos directamente el token.
- Resource Owner Password Credentials
  - En lugar de redirigir el pedido, el cliente captura y envía usuario y contraseña

    Hoy en dia por seguridad ya casi no se usa
- Client credentials

  El que se autentica no es el usuario final, sino la aplicacion cliente
  - Autorización a nivel cliente (via cliente secret)

# Oauth 2.0 – Ejemplo de mensajes

- Login redirect (URL) En este paso, el usuario se autentica con el proveedor OAuth

```
https://
login.salesforce.com/
services/oauth2/authorize?
response_type=code&client_i
d=84837569234 65.as.org&redi
rect_uri=https%3A%2F
%2Fwww.example.com%2Fback
```

# Oauth 2.0 – Ejemplo de mensajes

- Callback response (URL que recibe el callback)

  El proveedor OAuth le contesta al cliente

```
https://app.example.com/
oauth_callback?
code=aWekysIEeqM9PiThEfm0C
nr6MoLIfwWyRJcqOqHdF8f9INo
kharAS09ia7UNP6RiVScerfhc4
w%3D%3D
```

# Oauth 2.0 – Ejemplo de mensajes

- Token Request (post al Provider – form url encoding)

La aplicacion cliente le pide el token al provider. Este client_id y client_secret son secretos de la aplicacion cliente, no del usuario final.

```
code=aWekysIEeqM9PiThEfm0Cnr6MoLIfwWyRJcqOqHdF8f9INokharAS09ia7UNP6RiVScerfhc4w==&grant_type=authorization_code&client_id=ffdskhfeihoaw&client_secret=khfeaihdiu38nd&redirect_uri=...
```

# Oauth 2.0 – Ejemplo de mensajes

## Response token (json response body)


```
{
"id":"https://login.salesforce.com/id/
00D5000Z3ZEAW/00550001fg5OAQ",
"issued_at":"1296458209517",
"refresh_token":"5Aep862eWO5D.7wJBuW5aaARbb
xQ83jMRnbFNT5R8X2GUKNA==",
"instance_url":"",
"signature":"0/1Ldval/
TIPf2tTgTKUAxRy44VwEJ7ffsFLMWFcNoA=",
"access_token":"00D50000000IZ3Z!
AQ0AQDpEDKYsn7ioKug2aSmgCjgrPjG9eRLz"
}
```


Nuestro access token expira despues de determinado tiempo

El refresh token tiene un tiempo de expiracion mucho mas largo, lo que nos permite poder seguir usando el sistema sin autenticarnos cada vez

Durante un tiempo, se usaba el access_token como indicador de login. El problema con esto es que en ningun lado este token se vincula a un usuario. Por eso, se hizo una extension llamada OpenID Connect, que tiene un campo mas llamado "id_token", que es un JWT que tiene informacion del usuario y sus permisos.

# Oauth 2.0 – OpenID Connect

- Agrega autenticacion a Oauth2
- Se basa en un tipo especial de token (bearer token) al obtener el access token:

```
“id_token”: “
eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzcyI6ICJodHRw
Oi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5NzYxMDAxIiw
KICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZfV3pBMk1qIi
wKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5NzAKfQ.ggW8hZ
1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6qJp6IcmD3HP9
9Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJNqeGpe-gccM
g4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7TpdQyHE5lcMiKP
XfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoSK5hoDalrcvR
YLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4XUVrWOLrLl0
Nx7RkKU8NXNHq-rvKMzqg
“,
“access_token”: “jdj2V32hkKG”
“token_type”: Bearer”
```


Base 64 - URLSAFE

# Oauth 2.0 – OpenID Connect

- El bearer id token es un documento JSON (JWT – Json Web Token)
  - Firmado digitalmente por el provider

```
{
  "sub"       : "alice",                    <<< Subject
  "iss"       : "https://openid.c2id.com", <<< Issuer
  "aud"       : "client-12345",             <<< Audience – destinatario
  "nonce"     : "n-0S6_WzA2Mj",             <<< Nonce
  "auth_time" : 1311280969,                 <<< Cuando fue autenticado
  "acr"       : "c2id.loa.hisec",           <<< Como fue autenticado (OPT)
  "iat"       : 1311280970,      <<< Issue Time
  "exp"       : 1311281970,      <<< Expiration time
}
```

# Lectura recomendada

Capítulo 15

Computer Security Art and Science
Matt Bishop

OAuth2.0 – RFC 6749