

- Confidencialidad
- Integridad : de datos, origen y garantía
- Disponibilidad

### Control de Acceso:

- Discrecional: queda a discreción de una entidad ext.  $\Rightarrow$  reglas arbitrarias + mecanismos puntuales
- Mandatorio: reglas prefijadas + mecanismos del sistema  $\Rightarrow$  NO alterados

Bell-Lapadula : Confidencialidad (integridad viene segundo)

READ $l(o) \leq l(s)$ + permisos disc $\Rightarrow$ condición de seg. simple	WRITE $l(s) \leq l(o)$ + permisos disc $\Rightarrow$ condición de cierre	} MAC } DAC
--	--	----------------

Flujo de info hacia arriba

### MATRIZ DE ACCESO

Completo: con comportamientos

[Comportamiento] = (Level, {Categories})  $\Rightarrow$   $(L, \{C\}) \text{ dom } (L', \{C\}) \Leftrightarrow L' \leq L \text{ y } C' \leq C$

- Garantiza que  $\nexists$  flujo de info en el sentido de la cond. simple
- Cond. de cierre garantiza que  $\nexists$  caminos indirectos que violen la cond. simple

Problema

A  $\rightarrow$  B un msj

B contesta PERO B dom A  $\Rightarrow$  no puede escribir msj

Solución: disminuir nivel

Teorema: si un sist. comienza en estado seguro y sus transiciones cumplen cond. simple y de cierre  $\Rightarrow$  todos los estados del sist. son seguros

Principio de tranquilidad: NO cambiar niveles luego de ser creados

¿Qué pasa si subo de nivel un objeto? Info leída x usuarios de menor nivel  $\Rightarrow$  viola cond. ss

¿Qué pasa si bajo de nivel un objeto? Problema de desclasificación  $\Rightarrow$  viola principio de cierre

Modelos de Biba : Integridad

A mayor nivel, mayor confianza de que un dato es preciso y lo fiable.

Low Water Mark Policy

READ Todo PERO $l(s) = \min(l(s), l(o))$	WRITE $l(o) \leq l(s)$
---	---------------------------

READ Todo PERO $l(s) = \min(l(s), l(o))$	WRITE $l(o) \leq l(s)$
---	---------------------------

flujode info

$\hookrightarrow$  PROBLEMA: niveles de integridad de los sujetos decaen al el uso del sist  
 $\Rightarrow$  eventualmente nadie puede acceder o generar objetos de alto nivel de integridad

Solución? Modificar niveles de objs en vez de sujetos  $\Rightarrow$  mismo problema, objs se degradan

## Ring Policy

READ

Todo

WRITE

$l(o) \leq l(s)$

Flujo de

info



PROBLEMA: Considera solo el prob de la modificación directa de objs  $\Rightarrow$  modificaciones indirectas? Complementar con otro mecanismo

Strict integrity (al revés de Bell-Lapadula)

READ

$l(s) \leq l(o)$

WRITE

$l(o) \leq l(s)$

Flujo de

info



$\Rightarrow$  mismos problemas que Bell-Lapadula

Muralla China: Confidencialidad e Integridad

Problema de conflicto de intereses  $\Rightarrow$  agrupar entidades que entran en COI.

- Objetos relacionados a misma empresa se agrupan en CD's (Company Dataset)
- COIs contienen CD's con conflicto de intereses

⚠ Cada obj pertenece a exactamente 1 CD y 1 COI

READ (cond. ss)

Al principio TODO

Una vez que lee CD de un COI, NO puede leer otro CD del mismo COI

S lee o si:

- $\exists o'$  leido /  $CD(o') = CD(o)$
- $\forall o'$  leido,  $COI(o') \neq COI(o)$
- o es público

$\Rightarrow$  depende de historial de s

WRITE (cond. de cierre)

S escribe o si

- S puede leer o
- $\forall o'$  no publico, si s puede leer o'  $\Rightarrow CD(o') = CD(o)$
- $\exists p$  que s escriba o, es necesario que todo o' accesible por s pertenezca al mismo CD

## Composición de sistemas

### Políticas iguales

- Si se puede cambiar la política de componentes  $\Rightarrow$  composi° trivial
- Si no se puede  $\Rightarrow$  mostrar que composi° cubre los requerimientos de las políticas de los componentes DIFÍCIL

### Políticas diferentes

#### Guia:

- Cualquier acceso permitido en una debe estar permitido en la composi° = autonomía
  - Cualquier acceso prohibido en una debe estar prohibido en la composi° = seguridad
- $\Rightarrow$  composi° satisface seguridad de políticas de las componentes

#### ¿Caso no explícitamente permitido o prohibido?

- Modelo de Gong y Quim: permitirlo
- Principio de denegación por defecto: prohibirlo

⚠政治ica segura + politica segura = politica segura? NO necesariamente



## Principios de diseño

Bases p/ sist. robusto y seguro.

Basados en 2 ideas:

- Simplicidad: menos inconsistencias, menos cosas pueden salir mal, facil entender y verificar
- Restriccion: minimizar acceso y comunicacion

### Menor privilegio

Sujeto = solo privilegios necesarios p/ completar su tarea

⇒ privilegios por función, NO por identidad



Si tarea necesita adicionales,  
se asignan y desp se sacan

### Valores iniciales seguros

Por defecto: acceso denegado a cualquier objeto

Si una acción falla x seguridad, el sist. debe volver al estado de seguridad inicial

### Economia de mecanismos

Mecanismos de seguridad simples.

⇒ menos cosas salen mal y si algo sale mal, mas facil de corregir

### Mediación completa

TODOS los accesos a objs deben ser verificados.

Prob: en contra de eficiencia y difícil de implementar

### Diseño abierto

Seguridad NO debe depender de que el diseño o implementación sea privado.

Si se viola ⇒ seguridad x oscuridad.

Esto NO aplica a claves, sino a algoritmos.

### Separación de privilegios

Un sist no debe asignar permisos basados en una sola condición.

### Mecanismos exclusivos

Mecanismos de seguridad NO deben compartirse ⇒ promueve aislación (VMs, Sandboxing)

↳ Canales ocultos, variables compartidas

### Aceptación psicológica

Mecanismos de seguridad NO deben dificultar el acceso al recurso.

⇒ muy difícil / imposible

## Flujo de información

### Control de acceso

- limita operaciones sobre objetos
- limite acceso a info° de objetos

¿ACls sirven? Si, pero como son mecanismos abiertos deben ser complementados.

### Entropía

Mide la incertidumbre a la hora de determinar el valor de una variable.

Ej. Un texto en español no podemos deducir exacto° la sig letra y eso lo hace 'aleatorio'  $\Rightarrow$  entropía se encarga de medir esa aleatoriedad.

Sea  $X$  una VAD q toma valores  $x_1, \dots, x_n \Rightarrow H(X) = -\sum_{1 \leq j \leq n} P(x_j) \cdot \log_2 P(x_j)$

- Max. entropia:  $P(x_j) = 1/n$  i.e distribución uniforme
- Min. entropia:  $P(x_i) = 1$  y  $P(x_j) = 0 \quad \forall j \neq i \Rightarrow$  evento único (Ej. imagen de un color)

### Entropía condicional:

- Sea  $X$  una VAD que toma valores  $x_1, \dots, x_n$
  - Sea  $Y$  una VAD que toma valores  $y_1, \dots, y_m$
- $$\Rightarrow H(X/Y) = \sum_{1 \leq j \leq m} P(y_j) \cdot H(X/Y=y_j) \quad \text{donde } H(X/Y=y) = -\sum_{1 \leq i \leq n} P(x_i|y) \cdot \log_2 P(x_i|y)$$

### Flujo de información

Sea  $c$  una secuencia de comandos que lleva el sist de un estado  $s$  a un estado  $t$ . Sean  $x_s$  e  $y_t$  valores de objetos en el estado  $s$  e  $y_t$  valor de  $y$  en el estado  $t$ .

### Hay flujo de info° si:

- $H(x_s/y_t) < H(x_s/y_s)$  si  $\exists y_s$
- $H(x_s/y_t) < H(x_s)$  sino

Es decir, hay flujo de info° si a partir de  $y_t$  puedo deducir  $x_s$ .

Ejemplo: if  $x=0$  then  $y=1$  else  $y=0$

Considerando  $P(x=0) = P(x=1) = 0.5 \Rightarrow H(x) = 1$  y  $H(x/y) = 0 \Rightarrow H(x/y) < H(x) \Rightarrow$  hay traspaso de info°.

Obs: La info°, en este caso, fluye de manera indirecta i.e  $x$  e  $y$  no aparecen en la misma asigna°.

### Tipos de flujos

- Explícito: asigna° de tipo  $y = f(x)$
- Implícito: verifíca° de flujo sin asignaciones  $\Rightarrow$  difíciles de encontrar y controlar

### Políticas de control de flujo deben cumplir 2 principios:

#### REFLEXION

$a, b$  en misma clase

$a$  puede r/w  $0_1 \Rightarrow b$  puede r/w  $0_1$

$\hookrightarrow$  info° fluye libre° entre miembros de una misma clase

#### TRANSITIVIDAD

$a, b$  en  $\neq$  clases

$a$  puede w  $0_1$  y r  $0_2$  y  $b$  puede r  $0_2 \Rightarrow b$  puede r  $0_2$

$\hookrightarrow$  Si  $a$  r  $0_2$ , puede escribir en  $0_2$  info° de  $0_2$  y si  $b$  puede r  $0_2$  entonces puede r info° de  $0_2$

### Mecanismos

Estaticos: chequea que el flujo de info° a lo largo del programa sea autorizado

- Determina si los FIs en un prog pueden violar alguna política

- No es preciso  $\Rightarrow$  flujos seguros pueden estar marcados como q violan la política
- Es seguro  $\Rightarrow$  todo FI no autorizado va ser detectado
- Se analiza cmd x cmd
- Solo comandos certificados i.e. si el FI dentro de los comandos no viola la política

### Dinámicos: prevenir FIs que violen la política

- Se asignan etiquetas a lo info° contenido
- Cada zona tiene un conj de etiquetas requeridas y prohibidas

### Problema del confinamiento (aislacion)

Prevenir q un servidor revele info° que el user del servidor considere confidencial.

Aislacion total: proceso no puede comunicarse c/ otros procesos y no puede ser observado

$\Rightarrow$  proceso no revela info°

Problema: procesos usan memoria, ciclos de CPU, espacio de disco, ancho de banda i.e. recursos observables

$\Rightarrow$  crea un canal oculto de info°

### Canal oculto

Canal de comunicac° que no fue diseñado p/ ello.

- Espacial: utiliza atributos de recursos compartidos
- Temporal: utiliza info° temporal o de orden en el acceso a rec. compartidos

### Atributos:

- Ruido: capacidad de interferencia no premeditada de 3° partes
- Ancho de banda: tasa de transm°

### Side channel attacks: hacen uso de un canal oculto p/ ganar info°

### Métodos de aislación

#### (1) Ambiente q se comporte como computadora q solo corre procesos aislados

$\hookrightarrow$  no requiere modificar los sistemas

#### Maquinas virtuales: programas q simulan el HW de una maquina

- Puede ser una maquina real o abstracta
- Correr SOs si modifica°
- Núcleo de la VM = agente q provee seguridad
  - $\Rightarrow$  VMs = sujetos
  - $\Rightarrow$  recursos = objetos

#### (2) Correr procesos en un ambiente q analiza las acciones y detecta fugas de info°

$\hookrightarrow$  requiere modificar sistemas

#### Sandboxes: ambiente donde las acciones estan limitadas x cierta politica

Dos formas  $\Rightarrow$  se modifica el ambiente  $\Rightarrow$  kernel / SO modificado p/ imponer restricciones y programas NO modificados

$\Rightarrow$  se modifica el programa  $\Rightarrow$  se agregan llamadas a pts de control

## Autenticación

Asociar identidad a un principal:

- Identidad corresponde a entidad externa
- Principal = representante interno del sistema

Pasos:

- (1) Obtener info<sup>o</sup> de identidad
- (2) Analizarla
- (3) Determinar si corresponden a la entidad i.e. si la info<sup>o</sup> es suficiente para autenticar al usuario  
⇒ Necesario almacenar info<sup>o</sup> de cada entidad y tener mecanismos para procesar la info<sup>o</sup>

## Sistemas de autenticación

- A = info<sup>o</sup> de autenticación provista por una entidad externa (Ej. huella)
- C = info<sup>o</sup> complementaria que guarda el sistema para validar autenticación (Ej. patrón de huella)
- F = funciones de complementación que generan la info<sup>o</sup> complementaria ⇒ F: A → C
- L = funciones de autenticación determinan si un par (A,C) es una asociación válida ⇒ L: A × C → {0,1}
- S = funciones de selección permiten crear y/o actualizar A y C

## Información de autenticación

Provista por entidad ext. y posee grados de confianza.

Algo que conozco: basan la identificación en secretos compartidos

- Ej. clave, frase, preguntas secretas
- ⇒ almacenar de manera segura
  - ⇒ depende de confidencialidad del secreto
  - ⇒ mecanismo + empleado y sirve de base para otros mecanismos

Algo que tengo: elemento físico

- Ej. smart card, celular, generador de claves, tarjetas de coordenadas
- ⇒ requiere cuidado del elemento
  - ⇒ el elemento puede ser copiado/clonado
  - ⇒ info<sup>o</sup> en tránsito puede ser suplantado

Algo que soy: características intrínsecas de la entidad

- Ej. huellas, retina, voz, cara
- ⇒ características no modificables fáciles
  - ⇒ no 100% eficaz pq asume q el dispositivo de lectura no puede ser manipulado

Contexto: contexto de la info<sup>o</sup>

- Ej. red de origen / geolocalización, host/terminal/DEVICE de origen, dia y hora de acceso
- ⇒ Si un usuario accede desde otro país hay menos chances de q sea quien dice ser
  - ⇒ Factor complementario

## Claves

- Secuencia de caracteres: con restricciones, graft aleatoria, x el usuario o de manera asistida
- Secuencia de palabras (pass-phrases)
- Algorítmicas: pregunta-respuesta y claves de uso único (OTP)

## Almacenamiento

- Texto plano : archivo o BD, accedido x fuero del sist y no garantiza confidencialidad de claves
- Archivo cifrado : requiere una clave p/ acceder a los claves
  - ↳ clave en config file INO), mismo ejecutable del sist (NO), un dispositivo criptografico especializado

## Ataques a un sistema de autenticación

Encontrar  $a \in A / \exists f \in F. f(a) = c$ ,  $c$  asociado a una entidad.

- Ataques offline: probar varios  $a$ , computar  $f(a)$  y comparar con  $c$
- Ataques online: intentar autenticar vía  $\ell(a) \in L$

## Prevenciones

- Esconder info<sup>o</sup> p/ que  $a, c$  o  $f$  no sean conocidas simultanea<sup>t</sup>. General<sup>t</sup>,  $f$  conocido y + facil proteger  $c$  que  $a$ .
- Prevenir el uso de fun<sup>o</sup> de autentico<sup>t</sup>: tiempos l ante fallas, deshabilitar principales, captchas, Jailing/Honeypot.
- Subir complejidad de claves
- Formula de Anderson:  $P \geq \frac{IG}{N} = \frac{\text{tiempo dedicado a pruebas}}{\text{tamaño de espacio de claves}} \times \text{nº de pruebas realizadas por seg.}$   
 ↳ probabilidad de adivinar una clave

## Selección de claves

Selección aleatoria: ideal PERO difícil de memorizar

Claves pronunciables: palabras sin sentido pero formadas x fonemas (Ej. helgoret) ↳ facil de memorizar PERO espacio de claves reducido

Usuario elige: claves faciles de adivinar

## Revisión proactiva de claves

Analizar clave al momento de crearla.

- Rechazar claves faciles
- Realizar búsqueda de patrones
- Aplicar reglas sobre claves
- Realizar búsquedas en diccionarios

## Expiración de claves

Forzar cambio de clave pasado tiempo o evento.

- Evitar reuso de claves ↳ recordar y bloquear ultimas
- Evitar q se pueda cambiar muchas veces en un periodo de tiempo
- Dar a users tiempo p/ pensarlo: avisar antes

## Salting

Introducir una perturba<sup>o</sup> p/ aumentar tiempo requerido p/ adivinar claves.

$$\Rightarrow f(a) = c \parallel f'(a, s)$$

## PBKDF 2

(1) pass

(2) Calcular salt  $s$

(3) Fun<sup>o</sup> pseudoaleatoria (PRF):

- $U_1 = PRF(\text{pass}, s)$
- $U_2 = PRF(\text{pass}, U_1)$
- ...
- $U_j = PRF(\text{pass}, U_{j-1})$

$$\Rightarrow U_1 \text{ xor } U_2 \text{ xor } \dots \text{ xor } U_j$$

↳ A partir de una pass genera una clave.

## Challenge . Response

Problema: autenticar requiere enviar claves  $\Rightarrow$  requiere canal seguro  $\Rightarrow$  descubrir una comunica<sup>o</sup> antigua puede comprometer la clave

Solucion: NO enviar clave  $\Rightarrow$  Servidor manda challenge

## EKE - Encrypted key Exchange

Impedir que atacante capture clave en red  $\Rightarrow$  dialogo en canal encriptado. Atacante no sabe si la clave es correcta.

## One time passwords

Claves de uso unico. A y B comparten clave k y contador C  $\Rightarrow$  (k, C)  $\Rightarrow$  OTP  $\Rightarrow$  (otp, C+1)

### Variantes:

- HOTP (k, C) = Truncate (HMAC-SHA1(k, C))
  - $\hookrightarrow$  extrae 32 bits de los 160 del mac
  - $\Rightarrow$  Contador incrementa en cada uso
  - $\Rightarrow$  Requiere sincroniza<sup>o</sup> (look-ahead)
  - $\Rightarrow$  Usar throttling
  - $\Rightarrow$  Genera codigos de 1 a 8 digitos (al menos 6)
- TOTP (k) = HOTP (k, T) Siendo T = (current unix time - To) / x
  - $\hookrightarrow$  # segs de vida de cada codigo (usual<sup>+</sup> x = 30)
  - $\Rightarrow$  No requiere counter
  - $\Rightarrow$  Usar throttling
  - $\Rightarrow$  Usar un drift window = # de ticks defasados aceptados
  - $\Rightarrow$  Genera codigos de 1 a 8 digitos (al menos 6)

## Multi-factor authentication

Combinar 2 factores de 2 tipo  $\Rightarrow$  + complejo y - usable.

- Algo q conozco + tengo
- Algo q conozco + soy
- Algo que tengo + soy
- Algo que soy + contexto

## Autenticacion remota (SSO)

Delegar autentica<sup>o</sup> a sist. externo  $\Rightarrow$  rela<sup>o</sup> de confianza.

### Tecnologias abiertas:

- OpenID 1 y 2 Obsoletos
- OpenID Connect basado en OAuth2

## Control de Acceso

### Matriz de control de acceso

	obj1	Obj2	...	objm	
User 1	rw	r		rwx	
User 2	rx	ox			
...					
User n	ox	rw		x	CAP
					ACL

- Permite implementar cualquier política
- Rapido crecimiento
- Desperdicio de espacio
- Facilidad p/ determinar acceso
- Difícil administrar y dar altas/bajas (cambia estructura de matriz)

### Lista de Control de Acceso

Sean S conj de sujetos, R conj de acciones y O un objeto  $\Rightarrow \text{ACL}(O) = \{(S_i, r_i) / S_i \in S \wedge r_i \in R\}$

Principio de denegar por defecto: si un sujeto no tiene entrada en  $\text{ACL}(O)$ , no tiene ningún derecho

### Modificación de ACLs

#### Derecho de pertenencia (own):

- Asignado a quien crea el objeto  $\Rightarrow$  puede cambiar permisos
- Asignado según el tipo de objeto

#### Transferencia de permisos.

- Traspaso: cambiar el sujeto de una entrada de ACL  $\Rightarrow$  Si1 deja de tener permisos y los tiene S2
- Delegación: S1 delega a S2 SIN perder sus permisos

### Usuarios privilegiados

- A los cuales ACLs no aplican i.e. tienen todos los permisos (Ej. root en Linux)
- Con ACLs especiales (Ej. admin en Windows tienen derecho sobre todos los objetos)

### Grupos

Los ACLs pueden crecer mucho  $\Rightarrow$  grupos (roles):  $g = \{S_1, S_2, \dots, S_n / S_i \in S\}$

- Reducen tamaño de ACLs
- Agregan nueva dim de complejidad

### Conflictos: + de un AC pueden aplicar al mismo tiempo

Ej.  $\text{ACL}(O) = \{(u_1, rw), (g_1, w)\} \wedge u_1 \in g_1$

#### Soluciones:

- Grant-All: requiere que todos los ACs otorguen el derecho  $\Rightarrow$  U1 solo puede r
- First-Rule: se define un orden de evaluación y el 1º AC que se encuentra se usa  $\Rightarrow$  U1 puede rw xq esta antes

### Derechos por defecto

Permiten definir ACs sobre sujetos p/ los cuales NO hay un AC en la lista.

Ej.  $\text{AC}(O) = \{(u_1, x), (*, r)\}$

- Override: si sujeto tiene AC, utilizarlo sino default  $\Rightarrow$  U1 puede x pero no r
- Augment: partir del default y agregar ACs explícitos  $\Rightarrow$  U1 puede rx

### Revocación

Dueño quita al sujeto del ACL o quita el/los derecho(s) necesario(s) del AC.

Si hay transferencia: borrar en cascada permisos delegados

### Lista de Capacidades

Sean O conj de objetos, R conj de acciones y S sujeto  $\Rightarrow \text{CAP}(S) = \{(O_i, r_i) / O_i \in O \wedge r_i \in R\}$

$\Rightarrow$  Si accede a Oi ci cualquier derecho en ri

## Principio de denegar por defecto: S no tiene derechos sobre objetos q no estan en CAP(s)

- Indica que permisos tiene S sobre los objetos
- ≠ ACLs, el sist no controla estos datos => implementar mecanismos de protección p/ evitar q S altere/cree capacidades

### Implementación

- Tags: marcas de bits controladas x HW
  - => Set = proceso
  - => unset = proceso puede leer y modificar
- Paging/segmentos protegidos: almacenar CAPs en un segmento de memoria solo de lectura => procesos acceden indirectamente mediante punteros sino se podrían copiar
- Criptografía: asociar cada CAP a un hash criptográfico cifrado cl una clave conocida solo x el sist. => al presentar una CAP, el sist verifica el hash

### Controles

- Control de copia: copy flag asociada a los CAPs => proceso no puede copiar CAPs a otro proceso a menos q el flag este seteado
- Amplificación: capacidades extendidas temporales (Ej. user mode vs. kernel mode)

### Revocación

Implica revisar todas las listas de CAPs => costoso y a veces imposible.

#### Solución: In dirección

=> CAPs = índices en una tabla NO accesible a los procesos donde se invalida la entrada correspondiente cuando se revocan las CAPs

### ACLs vs CAPs

#### Equivalentes teóricos<sup>†</sup>

- ACLs: dado un obj. ¿quienes pueden usarlo y como?
- C-lists: dado un suj. ¿que objs puede acceder y como?

### Secretos compartidos

Implementación de políticas de separación de privilegios.

Método: (t,n)-threshold => n partes (sombras) de las cuales t partes son suficientes p/ acceder al obj. Si k < t NO permite acceso al obj.

Implementación: control vía sistema o threshold cryptography

### Método de Shamir

- Armazón polinomial de grado t:  $P(x) = a_0 + a_1x + \dots + a_{t-1}x^{t-1} + a_tx^t \mod p$ ,  $p > s$ ,  $p > n$  y  $a_0 = s$
- El secreto se reconstruye haciendo una interpolación de Lagrange =>  $P(0) = s$

Ejemplo: (1, 4), (2, 0), (3, 6), (4, 2), (5, 4) son las sombras y esquema (3, 5) mod 11

$$\text{Tomamos } (2, 0), (3, 6) \text{ y } (5, 4) \Rightarrow P(x) = 0 \frac{(x-3)(x-5)}{(2-3)(x-2)} + 6 \frac{(x-2)(x-5)}{(3-2)(x-3)} + 4 \frac{(x-2)(x-3)}{(5-2)(x-5)} = 5x^2 + 3x + 7 \mod 11$$

$$\Rightarrow \text{Secreto} = P(0) = 7 \mod 11 = 7$$

### OAuth2

#### Open Standard for Authorization

=> Permite al dueño de un recurso delegar en una aplicación el acceso al mismo

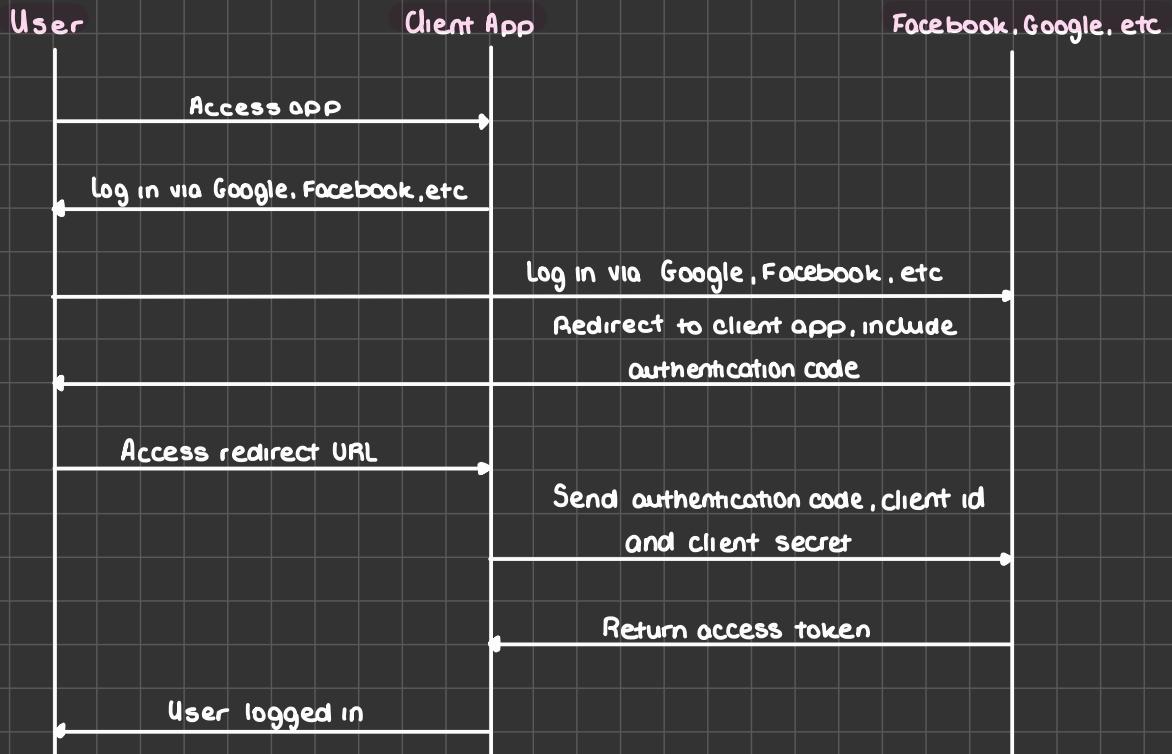
### Tipos de clientes

- Confidencial: app en un server, puede mantener secreto compartido al servidor (Client-Secret)
- Publico: desktop/mobile app o app que corre en un navegador → no puede mantener secreto de forma confiable

### Información del cliente:

Clients deben registrarse x adelantado, lo cual requiere:

- Client ID
- Client secret = solo pl confidenciales
- Redirect URL(s) = lista de direcciones validas a donde redirigir los accesos



### Grant Types

- Authorization Code: Server retorna code y cliente consigue access token al el código, su client id y su client secret
- Implicit: Servidor de autoriza° retorna access token
- Resource Owner Password Credentials: en lugar de redirigir pedido, el cliente lo captura y envia user y pass
- Client credentials: autoriza° a nivel cliente (via client secret)

### OpenID connect

Agrega autentica° a OAuth 2: se basa en un bearer token que es un JSON firmado digitalmente x el provider.

## Análisis de Vulnerabilidades

- Confianza : sist. cumple requerimientos
  - Aseguramiento : confianza obtenida a través de técnicas específicas
- ↳ adhesión a estándares, documentación, revisión x expertos (costoso y complejo)

$$\text{Riesgo} = \text{amenazas} \times \text{vulnerabilidades} \times \text{bienes}$$

Política → requerimientos q definen las expectativas de Seguridad



Aseguramiento → justifico° de q mecanismo sigue la política, a través de evidencia



Mecanismo → ejecutables diseñados e implementados para hacer cumplir políticas

Evidencia: enunciados (informal), pseudocódigo (semiformal), métodos matemáticos (formal)

### Fuentes de problemas

1. Requerimientos incompletos, incorrectos o faltantes
2. Fallos en diseño
3. Fallos en implementación en HW
4. Fallos en implementación en SW
5. Errores de uso x errores de operación
6. Uso indebido del sist.
7. Fallos de los equipos o medio de comunicación
8. Casos de fuerza mayor, desastres
9. Errores al actualizar, mantener o decomisar

### Análisis de requerimientos

Amenazas: evento q tiene como consecuencia un efecto no deseado en el sist.

↳ potencial

# vulnerabilidades: permiten que ocurra una amenaza

Bug → vulnerabilidad → amenazas → efecto no deseado

#### Clasificación por consecuencia:

Perdida de confidencialidad

• Perdida de integridad

• Denegación de servicio

### Modelado de amenazas

Trabajo previo a construcción del sist pl. documentar q aspectos de seguridad cubrir.

↳ políticas del sist (i.e req de seguridad) deben eliminar amenazas documentadas

### Perspectivas: centrado en atacante, software o productos

IN: conocimiento de función primaria de aplico°, casos de uso, modelos E/R

OUT: lista de amenazas y vulnerabilidades

(1) Identificar objetivos de seguridad: garantizar C.I y D

## (2) Conceptualizar aplica°

- Esquematizar escenario del deployment : topología, componentes, protocolos, servicios
- Identificar roles
- Identificar tecnologías
- Identificar mecanismos de Seguridad

## (3) Descomponer aplica°

- Identificar zonas donde cambia el nivel de confianza requerido
- Identificar flujo de datos

## (4) Identificar amenazas

- Comenzar con lista de amenazas recurrentes
- Derivar amenazas mediante preguntas
- Explorar amenazas mediante arboles de ataques

### Clasificación STRIDE:

S = spoofing => viola autenticación

T = tampering => viola integridad

R = repudiation => viola non repudiation

I = information disclosure => viola confidencialidad

D = denial of Service => viola disponibilidad

E = elevation of privilege => viola autorización

## (5) Identificar vulnerabilidades (que permiten que ocurran amenazas)

Revisar zonas q se derivan del análisis anterior

## A Iterativo

## Pentesting

Prueba para verificar q un sist cumple con ciertas restricciones.

- Precondiciones: hipótesis sobre el estado del sist y la existencia de una vulnerabilidad
- Poscondiciones: sist comprometido

Consiste en aplicar pruebas p/ intentar mover al sist. del estado inicial al comprometido.

≠ VERIFICACION FORMAL: verifco° matemática de q un sist. cumple c/ ciertas restricciones

⇒ prueba ausencia + existencia de vulnerabilidades

OJO: PT solo prueba existencia

Objetivo: probar eficacia de los controles de seguridad de un sist.

- ⇒ ethical hacking i.e se ejecutan técnicas similares a las de un atacante
- ⇒ prueba al sist. como un todo

### Metodología Informal

- Determinar y cuantificar objetivos
- Encontrar/Buscar vulnerabilidades
- Clasificar vulnerabilidades

### Metodología de Hipótesis de Falla

1. Recolectar info : p/ conocer sist. y su funcionamiento
2. Hipótesis: asumir que 3 vulnerabilidades concretas
3. Prueba: probar vulnerabilidades
4. Generalización: generalizar patrones y hallar otras vulnerabilidades del mismo tipo
5. Eliminación (opcional): determinar posos p/ eliminarlas  
↳ por lo geral, solo se incluyen recomendaciones

#### Paso 1: Recolectar información

- Componer modelo del sist y sus partes
- Documento°, mapeo de privilegios, tipos de cuentas ...
- Conocer ambiente: usuarios / servidores, estructura de red del sist.

#### Paso 2: Hipótesis

Possibles vulnerabilidades examinando políticas y procedimientos, examinando implementaciones, examinando mecanismos, comparando con otros sistemas, verificando Federadores, ...

⇒ Obtenemos lista de vulnerabilidades

#### Paso 3: Prueba

- Priorizar vulnerabilidades
- Definir como probar su existencia
- Diseñar test lo menos intrusivo posible

⚠ Prueba debe ser repetible

Pivoting: uso de un pt de acceso comprometido p/ continuar un ataque

#### Paso 4: Generalizar

Patrones + combinación de vulnerabilidades ⇒ parte + importante de PT