Unidad 4

Autenticación

Contenido

- **Introducción**
- **Sistema de Autenticación**
- **Claves**
- **Biométricos**
- **Combinaciones**
- **Mecanismos de configuración**
- **Mecanismos de almacenamiento de claves**

Introducción

- **Autenticación: Es la asociación entre una identidad y un objeto**
  - Identidad es la de la entidad externa (persona, proceso cliente)
  - objeto es la entidad “informática” (UID)

Establecimiento de la Identidad

- **Que es lo que la entidad sabe (clave)**
- **Que es lo que la entidad tiene (documento, tarjeta)**
- **Que es lo que la entidad es (dactilares, retina)**
- **Donde es que la entidad está (terminal local, remota)**

Establecimiento de la Identidad

- **Continuamente usamos, independientemente de los sistemas informáticos.**

- **En base a que sabemos:**
  - PIN de celulares
  - clave de alarma antirobo

- **En base a que tenemos**
  - rosa en el ojal
  - llave de auto

Sistema de autenticación

- **(A, C, F, L ,S)**
  - A es la información que prueba la identidad.
  - C información almacenada digitalmente usada para validar la información de identidad.
  - F Función que transforma de A en C.
  - L Función que prueba la identidad.
  - S funciones que permiten crear o alterar información en A y C.

Ejemplo de (ACFLS)

- **Sistema de claves almacenadas en texto plano y transparente**
  - A conjunto de strings que forman claves
  - C = A
  - F función de identidad
  - L función de comparación de igualdad de strings
  - S funciones de edición del archivo de claves.

Claves

- **Secuencia de caracteres**
  - Combinación de letras y dígitos elegidos por el usuario.
- **Secuencia de Palabras**
  - Pass-Phrases

- **Algoritmos**
  - Challenge Response, one-time passwords

Almacenamiento de Claves

- **Texto transparente**
  - Si el archivo es comprometido, **todas** las claves lo están
- **Archivo cifrado**
  - Necesita claves para cifrado/descifrado en memoria
  - terminamos en problema anterior
- **One-way Hash de la clave**
  - Si el archivo es comprometido, el atacante aún debe adivinar las claves o invertir la función.

Unix Passwords (originales)

- **UNIX utiliza un hash de la clave**
  - Convierte utilizando una de 4096 funciones de HASH la clave en un string de 11 caracteres.

- **Expresado como sistema:**
  - *A* = { strings de 8 o menos caracteres, 6.9×10^{16} }
  - *C* = { 2 letras de id de hash || 11 letras de hash en sí, 3.0×10^{23}}
  - *F* = { 4096 versiones de DES modificado}
  - *L* = { *login*, *su*, … } EQ
  - *S* = { *passwd*, *nispasswd*, *passwd*+, … }

Unix Password, detalle

El algoritmo original:

- **Cifra mediante DES un bloque de 64 bits en 0, utilizando la password como clave.**
- **El proceso se repite 25 veces, cifrando en cada paso el resultado del anterior.**
- **Para complicar el proceso, se agrega un *SALT,* que, basado en la hora del dia, selecciona una de 4096 leves variantes de DES.**
- **La “Salt” se almacena sin cifrar junto con la password cifrada.**

Salt

- En criptografía, el **salt** comprende bits aleatorios que son usados como una de las entradas en una función derivadora de claves. La otra entrada es habitualmente una contraseña. La salida de la función derivadora de claves se almacena como la versión cifrada de la contraseña. La sal puede también ser usada como parte de una clave en un cifrado u otro algoritmo criptográfico. La función de derivación de claves generalmente usa una función hash.

Anatomía de un ataque

- **Objetivo: encontar $a \in A$ tal qué**
  - para cierto $f \in F$, $f(a) = c \in C$
  - c está asociado a una entidad.

- **Dos maneras de lograrlo:**
  - Directamente
  - Indirectamente: como $I(a)$ es correcto iff $f(a) = c \in C$ para algún c asociado a una entidad, entonces pruebo $I(a)$

Prevención de ataques

- **Ocultar la mayor cantidad de variables de la ecuación. (*a, f* o *c*)**
  - por ejemplo linux, actualmente utiliza esquemas de shadow password, ocultando c.

- **Bloquear acceso a $l \in L$ o el resultado de *l*(*a)***
  - Ej: anular login desde la red

Ataques de Diccionario

- Un ataque de diccionario es un método de cracking que consiste en intentar averiguar una contraseña probando todas las palabras de una lista (ej: un diccionario).
  - Off-line: sabiendo *f* y *c* aplicamos *f* a múltiples $a \in A$ hasta que coincida con *c*.
    - *John-the-ripper, hashcat*
  - On-Line: mediante acceso a funciones de *L*, intentamos con múltiples $a \in A$ hasta que algún *l(a)* sea exitoso.
    - intentando logins. THC hydra, Medusa, ncrack

Ataques de fuerza Bruta

Se denomina **ataque de fuerza bruta** a la forma de recuperar una clave probando todas las combinaciones posibles hasta encontrar aquella que permite el acceso.

Variables que entran en juego:

- Longitud de la clave
- Set de caracteres utilizado

Menos eficiente que el ataque de diccionario. Muchas veces se combinan.

¿Por qué un Salt?

- Si no tenemos entonces a cada clave le corresponde un único hash:
  - Fácil identificar cuando dos usuarios utilizan la misma clave.
  - El espacio de contraseñas posibles (fáciles de recordar por el usuario) son pocas, por ende un diccionario de hashes es pequeño y rápido de construir.

- El SALT amplía el espacio de claves, dificultando ataques de diccionario. Es necesario construir el hash de las claves para cada SALT posible.

Agregando condimentos

- Pepper
- Además de usar salt, puedo usar un segundo valor que se almacene en otro lado.
- De esta manera, si un atacante logra por ejemplo hacer un SQL injection y roba los hashes de las claves almacenados en la base, no va a tener la información necesaria para intentar recuperar las claves.

Elección de claves

- **Selección aleatoria**
  - cualquier clave de *A* es equiprobable
- **Claves pronunciables**
- **Claves elegidas por los usuarios**

Perdón por el olor. Tengo todas mis contraseñas tatuadas
entre los dedos de los pies…

Claves Aleatorias

- **Asignadas por el sistema:**
  - Casos especiales: claves cortas, caracteres repetidos. Eliminarlos reduce el espacio de búsqueda.
  - Dificultad de recordarlas.
  - Calidad del generador aleatorio.

Claves Pronunciables

- **Generación aleatoria de Fonemas** **(traducción incorrecta)**
  - unidades de sonido: cv, vc, cvc, vcv
  - Ejemplo: helgoret, pirtela.
    asdlkj o aerje no

- **Son pocos**

Elegidas por el usuario

- **Problema: la gente usa claves “fáciles”**
  - basadas en nombres, lugares.
  - Palabras de diccionario (con variantes conocidas como mayúsculas incorrectas, $ por s, 1 por l ,etc)
  - Cortas, solo letras, solo números
  - Acrónimos
  - datos personales (trabajo, hobby, apodo, mascota)
  - Palabra de diccionario con mayúscula al principio y número al final.

Chequeo Proactivo

- **Forzar el chequeo de eficiencia de las claves**
  - siempre
  - detectar y rechazar claves “débiles”
  - Discriminar por sitio, usuario.
  - Hacer búsqueda de expresiones regulares.
  - ejecutar programas externos (spell)
  - Fácil de configurar

Ejemplo: *passwd+*

- **Provee un lenguaje de script para chequear proactivamente**
  - test length(“$p”) < 6
    - si la clave tiene menos de 6 caracteres, rechazarla
  - test infile(“/usr/dict/words”, “$p”)
    - si la clave está en /usr/dict/words, rechazarla
  - test !inprog(“spell”, “$p”, “$p”)
    - Si la clave no está en la salida del programa *spell*, rechazarla

Windows passfilt.dll – requisitos de complejidad (opcional)

"Su contraseña debe tener al menos *x* caracteres, ser diferente de sus *x* contraseñas anteriores, tener como mínimo *x* días de antigüedad, contener mayúsculas, números o signos de puntuación y no contener su nombre de cuenta o nombre completo. Escriba una contraseña diferente. Escriba una contraseña que cumpla con estos requisitos en ambos cuadros de texto. "

La contraseña debe contener al menos tres de los cuatro grupos de caracteres siguientes:

- Mayúsculas (de la A a la Z)
- Minúsculas (de la a a la z)
- Números (del 0 al 9)
- Caracteres no alfabéticos (como !, $, #, %)

Acceso vía *L* (on-line)

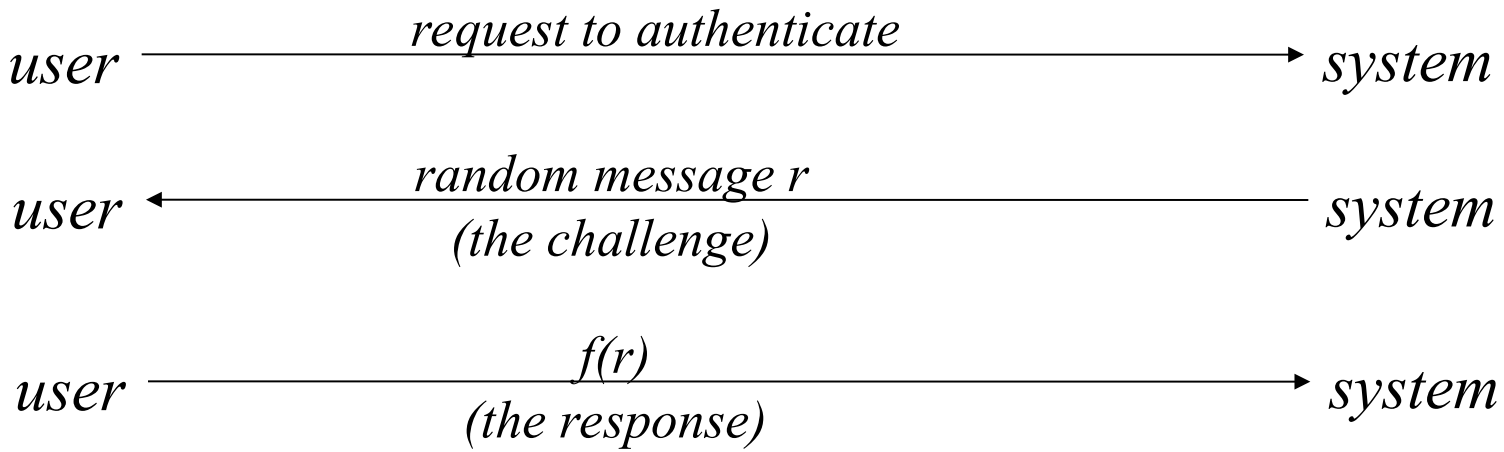
- **Inevitable**
  - sino los usuarios reales no pueden entrar
- **Complicarlo en lo posible, enlentecerlo**
  - Backoff o tarpit
  - Desconexión
  - Deshabilitación (Cuidado con root)
  - Jail

Vencimiento de Claves

- **Forzar el cambio de claves luego de cierto tiempo**
  - ¿Cómo forzar la no reutilización?
    - guardar las previas
    - bloquear cambios por cierto tiempo.
  - permitir a los usuarios pensar
    - avisar con anticipación.
    - no forzar en el momento de logon
    - **Hay estudios que indican que forzar el cambio de claves es contraproducente.**

Challenge-Response

- El usuario y el sistema comparten una función secreta *f* (en realidad una función conocida pero con parámetros ocultos, por ejemplo una clave criptográfica).



Ataques a Challenge-response

- **Igual que para claves fijas**
  - si el atacante conoce el challenge *r* y la respuesta *f(r)* y la *f,* puede probar diferentes claves.
    - En ciertas oportunidades alcanza con saber la forma de la respuesta, ya que *r* es generado aleatoriamente combinando datos conocidos.
    - En kerberos en particular, conociendo parte de los datos (nombre hora, etc) fue posible probar claves sobre sectores conocidos de la respuesta

Intercambio de claves cifradas

Evita los ataques off-line.

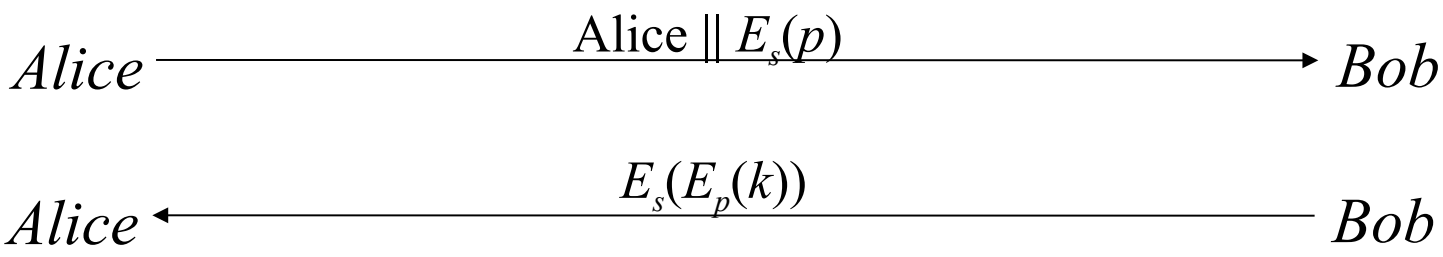
El challenge r es cifrado para evitar que el atacante pueda verificar el correcto desciframiento del mismo.

Alice & bob comparten una clave secreta *s*

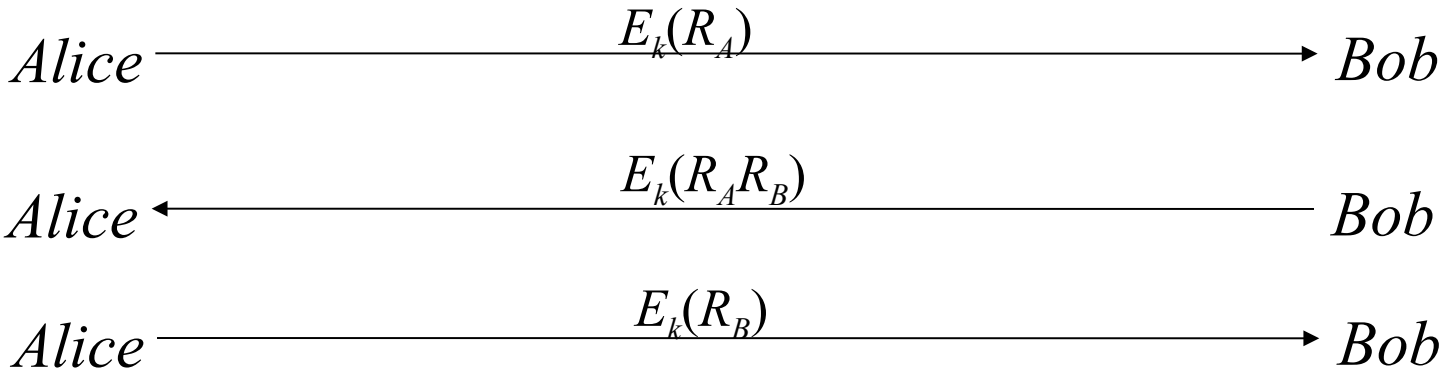
Alice debe generar una claves *p* aleatoria

k es una clave de sesión aleatoria y *Ra* y *Rb* son challenges aleatorios

Protocolo EKE



Alice & bob comparten una clave de session secreta aleatoria k

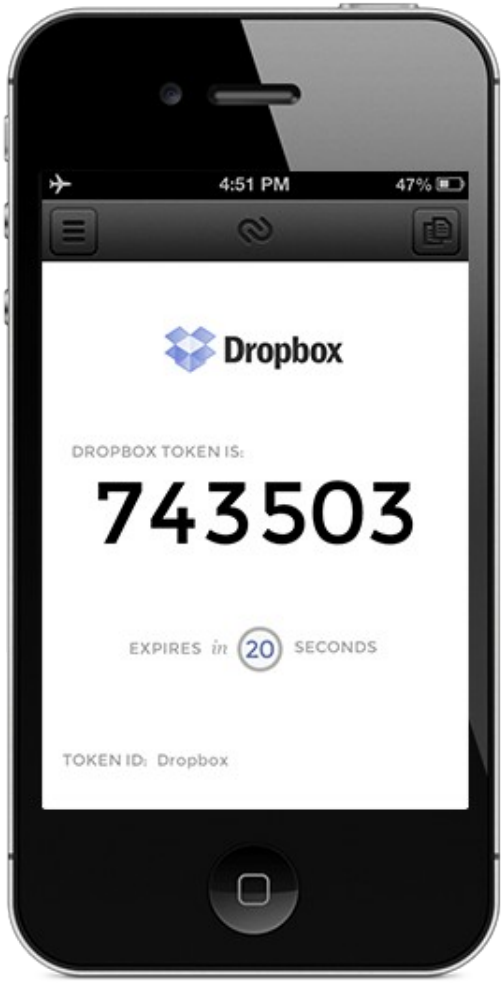


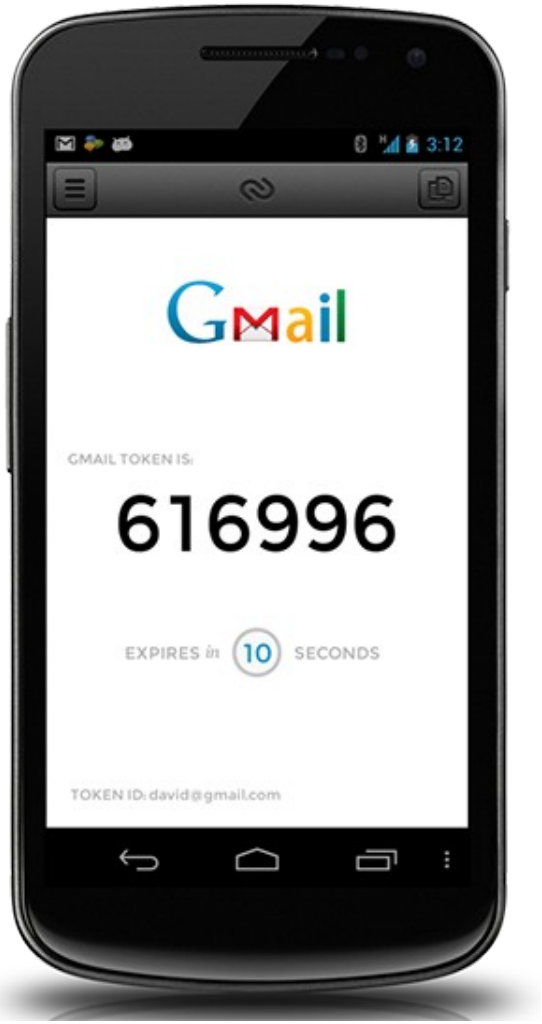
One-Time Passwords

- **Pueden ser usadas una sola vez**
  - se invalida automáticamente luego de su uso y/o vencen muy rápido.
- **Problemas**
  - Sincronización de partes
  - Generación de buenas claves pseudo-aleatorias
  - Distribución de claves.

Tokens OTP

Aplicaciones para celulares





TOPT y HOPT

RFC 6238 - TOTP: Time-based One-time Password Algorithm

https://tools.ietf.org/html/rfc6238

RFC4226 - HOTP: An HMAC-Based One-Time Password Algorithm

https://tools.ietf.org/html/rfc4226

Biometría

- **Medición automática de características biológicas o de comportamiento de un individuo.**
  - Huellas digitales: óptica o eléctricamente
    - mapea en un grafo, y compara en una base de datos
    - no es preciso, así que se usan algoritmos de aproximación.
  - Voces: verificación o reconocimiento
    - Verificación: técnicas estadísticas de matcheo
    - Reconocimiento: chequea el contenido de las respuestas (independiente del emisor)

Biometría (cont)

- **Ojos: los patrones del iris son únicos**
  - muy intrusivo
- **Cara: imagen o características especificas**
  - matcheo de imágenes
  - detección de patrones (largo de labio, ángulo de cejas)
- **Secuencias de tipeo: se creen únicas.**
  - presión sobre las teclas
  - cadencia


Ctrl

Localización

- **Sabiendo donde esta el usuario, validar si efectivamente está ahí**
  - Requiere hard específico (GPS)
  - el usuario envía continuamente información de locación.

Múltiples Métodos

- **Se pueden utilizar múltiples métodos simultáneamente:**
  - Donde estás y que tenés (GPS+token)
- **Se pueden utilizar diferentes métodos para diferentes tareas (niveles)**
  - también controles de acceso (hora, terminal)
- **Cada vez se usa más, especialmente combinado con teléfonos celulares**

PAM - Pluggable Authentication Modules

Mecanismo flexible para la autenticación de usuarios. PAM permite el desarrollo de programas independientes del mecanismo de autenticación a utilizar. Así es posible que un programa que aproveche las facilidades ofrecidas por PAM sea capaz de utilizar desde el sencillo /etc/passwd hasta dispositivos hardware, pasando por servidores LDAP o motores de bases de datos.

Permite distintas políticas de autenticación para cada servicio.

PAM

- **Sistema configurable de autenticación**
- **Chequea un repositorio de métodos a utilizar**
- **Librería: *pam_authenticate***
  - accede a archivo con igual nombre del programa en */etc/pam.d*
- **Módulos independientes efectúan el chequeo**
  - *sufficient*: Aceptado si modulo acepta
  - *required:* falla si el modulo falla, pero ejecuta todos los requeridos antes de reportar falla
  - *requisite:* igual a required, pero no chequea el resto
  - *optional:* solo es invocado si todos fallan

Ejemplo PAM

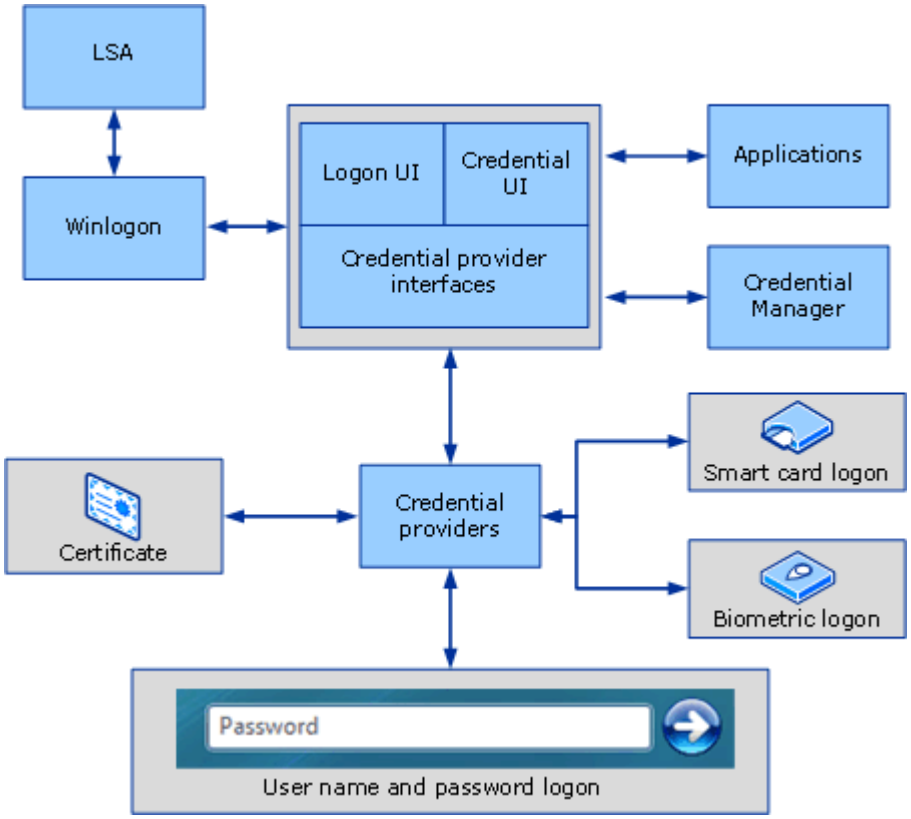
rlogin

auth required /lib/security/pam_nologin.so
auth required /lib/security/pam_securetty.so
auth required /lib/security/pam_env.so
auth sufficient /lib/security/pam_rhosts_auth.so
auth required /lib/security/pam_stack.so service=system-auth

 # Autenticacion SMTP contra un server LDAP (ver /etc/ldap.conf)
auth required /lib/security/pam_ldap.so debug
auth required /lib/security/pam_nologin.so
account sufficient /lib/security/pam_ldap.so debug
account required /lib/security/pam_unix_acct.so

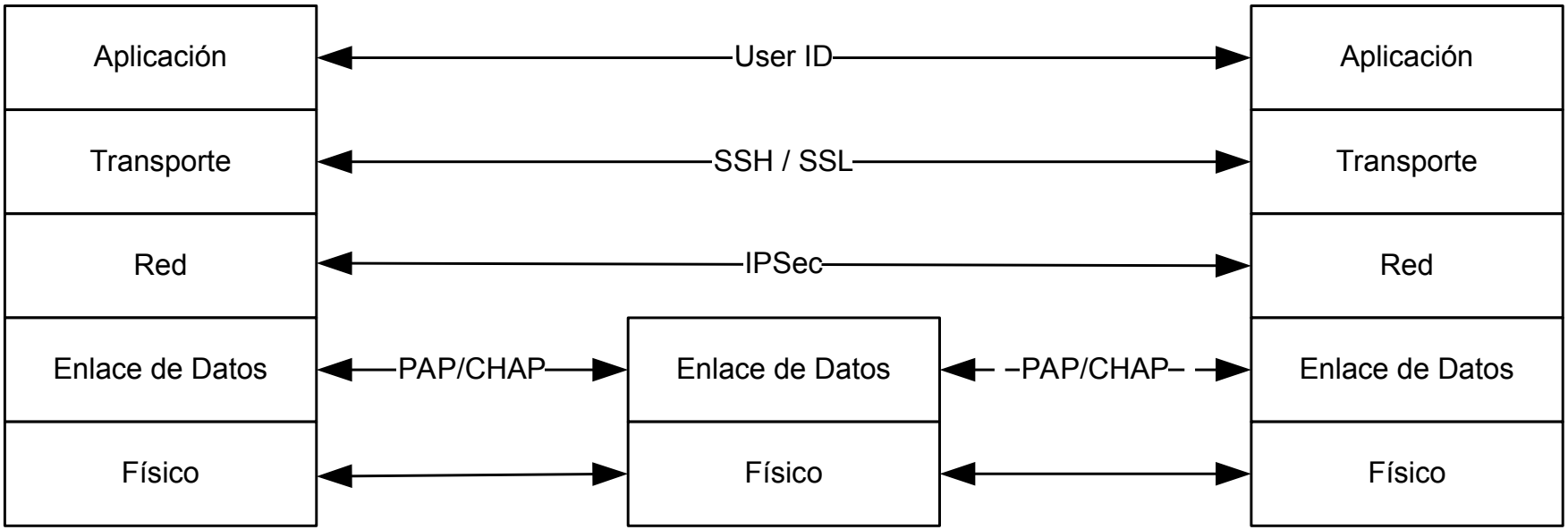
MS Credential Providers

- **Mecanismo implementado a partir de Vista/2008**



Autenticación

No solo se autentican usuarios y no solo es a nivel de aplicación, la autenticación puede realizarse en varios niveles:



Gestión de usuarios y contraseñas

Identity Management

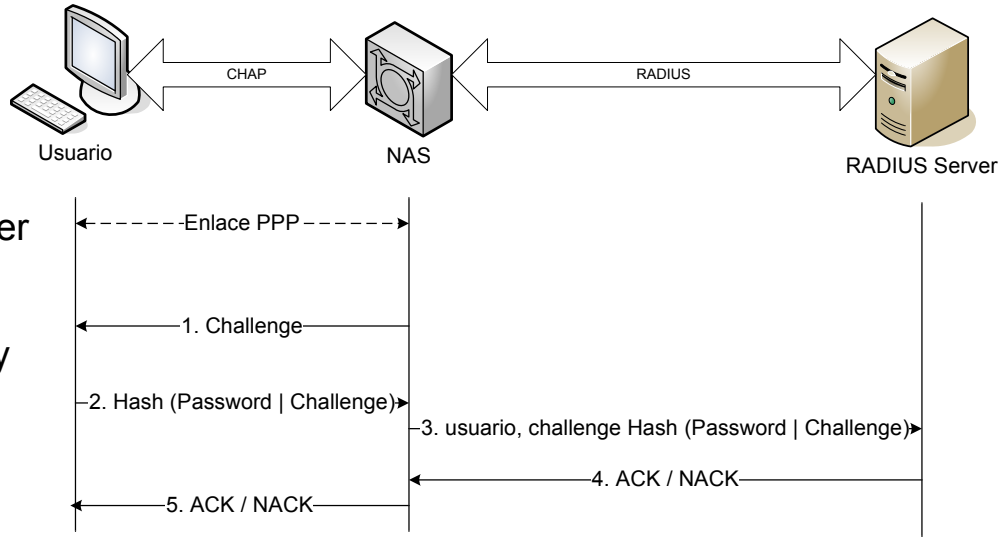
Sistema integrado de políticas y procesos organizacionales que pretende facilitar y controlar el acceso a los sistemas de información y a las instalaciones.

Privileged Identity Management

Orientado a manejo de contraseñas de administración de la infraestructura IT de una organización.

Varios

- **Radius:** Remote Authentication Dial In User Service
  - Diseñado para transmitir información de autenticación entre NAS y un server de autenticación central.
  - Utiliza PAP o CHAP.
  - Soporta Autenticación, Autorización y Accounting (pero viajan todas las funciones en los mismos paquetes)

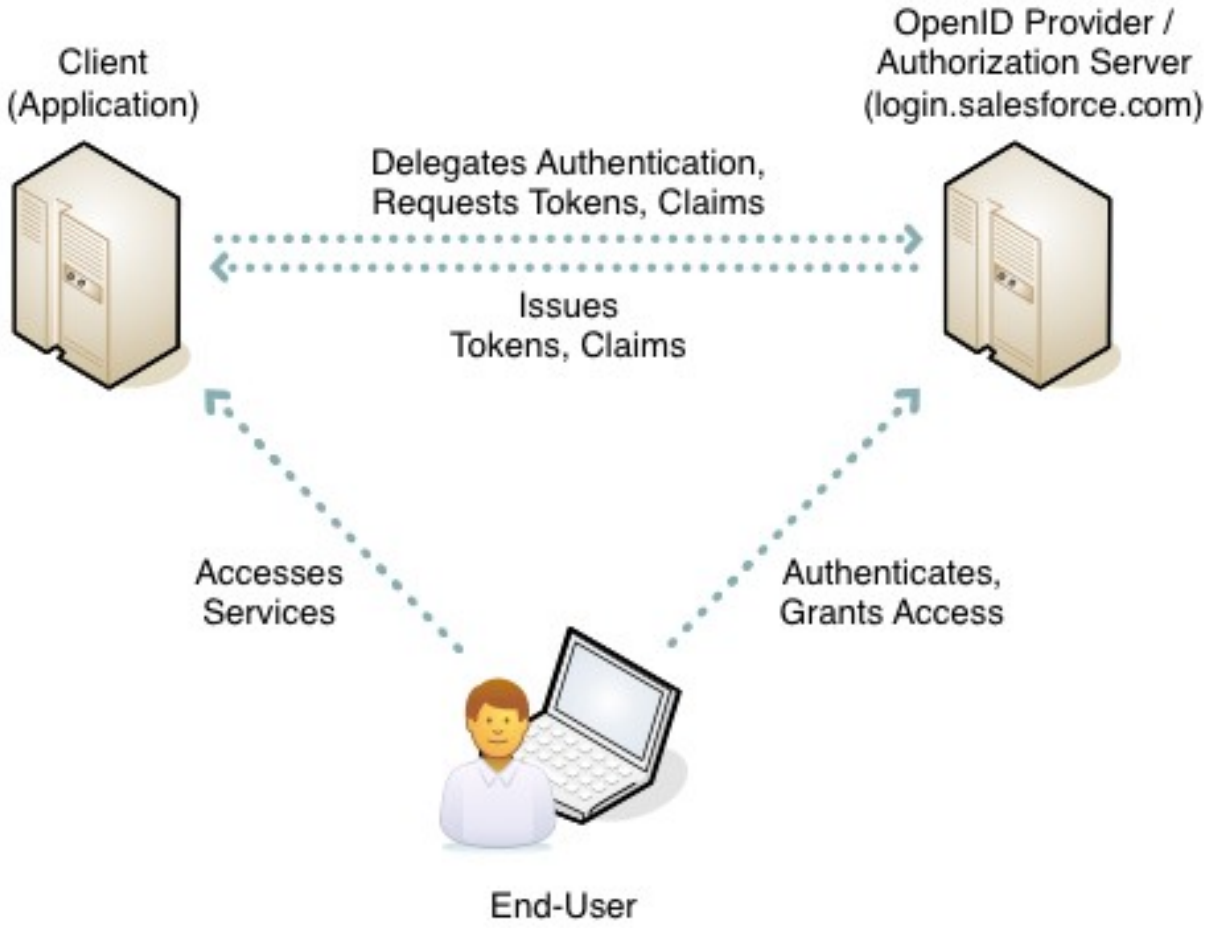


- **SSO Single Sign On**
  - Login único y centralizado a todos los recursos

- **Herramientas de gestión segura de claves**
  - almacenamiento y acceso seguro a claves e información personal

https://www.keepassx.org/

Open ID Connect



Ver https://www.argentina.gob.ar/autenticar

WebAuthN

- W3c y Fido alliance
- Apoyado por Google, Microsoft y Mozilla
- Standard de autenticación para aplicaciones web, utilizando distintos mecanismos segun las capacidades del dispositivo cliente.

https://www.w3.org/TR/webauthn/



Passkeys

- Usar TPM o TEE para almacenar los pares de claves del usuario para cada sitio. Se pueden sincronizar en la nube, si no uso un dispositivo tipo yubikey.

https://fidoalliance.org/passkeys/

https://www.yubico.com/resources/glossary/what-is-a-passkey/

https://docs.yubico.com/hardware/yubikey-guidance/best-practices/all-faq-passkeys.html

https://www.yubico.com/blog/passkeys-are-winning-but-security-leaders-must-raise-the-bar/

Almacenamiento de claves de usuarios. Mecanismos utilizados en Windows y Linux

Lan Manager Hash

- **Se convierte todo a mayusculas antes de generar el hash.**
- **El set de caracteres es:**
  ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 %!@\#$%^&*()_-=+‘~[]\{}|\:;"’<>,.?/

- **El hash se divide en dos bloques de 7 caracteres. Si la clave tiene menos de 14 caracteres, se paddea con null.**
- **Utiliza DES, encriptado con la clave el texto “KGS!@#$%”**
- **El resultado del hash es un valor de 16 bytes.**
- **No se utiliza Salt.**

NT Hash

- Distingue entre mayusculas y minusculas.
- Utiliza MD4.
- Puede tener un largo de hasta 128 caracteres, y la longitud es variable, no hace falta hacer padding.
- No se utiliza salt.
- No es soportado por defecto en win 95,98 y me. Nt4 lo soporta desde el SP4

Linux – MD5

La contraseña se almacena en un archivo sólo accesible por root (/etc/shadow) y la clave se cifra de la siguiente manera:

`$1$O34Nj3aF$3ecTiyOvCOAqhtaEaN0yG0`

La misma está compuesta por 3 campos:

- `$1$`: Indica que se utiliza MD5.
- `$O34Nj3aF$`: El salt utilizado.
- `$3ecTi…`: El hash propiamente dicho.

OpenBSD Bcrypt

- **Implementa un mecanismo por el cual el tiempo computacional necesario para cifrar una password con una función de una vía puede ir variando en el tiempo, a medida que avanza la velocidad del hardware. Las claves de usuarios con mayores privilegios pueden ser configuradas para ser más difíciles de calcular.**

- **Ver paper "A Future-Adaptable Password Scheme" - http://www.usenix.org/event/usenix99/provos/provos.pdf**

Linux – Evolución

Se pueden usar otros algoritmos:

ID	MÉTODO
1	MD5
2a	Blowfish
5	sha-256
6	sha-512

- Se puede configurar el numero de rounds. Por defecto es 5000 para SHA-512.
- Este último es un mecanismo razonable para utilizar en aplicaciones web, con un número más alto de rondas (ej: 100000).

Otros mecanismos

- **PBKDF2**

  Password-Based Key Derivation Function 2

- **Scrypt**

  requiere más memoria, para dificultar ataques mediante hardware de uso más específico.

- **Argon2**

Ganador del password hashing contest. 2 variantes: 1 para dificultar side-channels, otra dificulta ataque con GPU

https://password-hashing.net/

Cracking de claves

John The Ripper

Permite crackear claves Crypt,MD5,Blowfish,LM,etc. ya sea usando diccionarios, utilizando reglas, o por fuerza bruta.

Benchmarking: descrypt, traditional crypt(3) [DES 256/256 AVX2-16]... (4xOMP) DONE
Many salts: 18579K c/s real, 4691K c/s virtual

Benchmarking: md5crypt, crypt(3) 1 [MD5 256/256 AVX2 8x3]... (4xOMP) DONE
Raw: 173184 c/s real, 43513 c/s virtual

Benchmarking: bcrypt ("$2a$05", 32 iterations) [Blowfish 32/64 X2]... (4xOMP) DONE
Speed for cost 1 (iteration count) of 32
Raw: 2970 c/s real, 755 c/s virtual

Benchmarking: LM [DES 256/256 AVX2-16]... (4xOMP) DONE
Raw: 95627K c/s real, 24513K c/s virtual

Hashcat

Logra mejor performance usando la GPU

Hashtype: descrypt, DES (Unix), Traditional DES
Speed.Dev.#1.....: 101.9 MH/s (61.65ms)

Hashtype: md5crypt, MD5 (Unix), Cisco-IOS 1 (MD5)
Speed.Dev.#1.....: 774.0 kH/s (45.76ms)

Hashtype: sha512crypt 6, SHA512 (Unix)
Speed.Dev.#1.....: 14452 H/s (85.97ms)

Hashtype: LM
Speed.Dev.#1.....: 1637.2 MH/s (61.37ms)

Hashcat

En PC gamer. I7 + GeForce GTX1080 Ti

Hashtype: descrypt, DES (Unix), Traditional DES
Speed.Dev.#1.....: 1418.3 MH/s (82.65ms)

Hashtype: md5crypt, MD5 (Unix), Cisco-IOS 1 (MD5)
Speed.Dev.#1.....: 1418.3 MH/s (56.31ms)

Hashtype: sha512crypt 6, SHA512 (Unix)
Speed.Dev.#1.....: 228.8 kH/s (50.48ms)

Hashtype: LM
Speed.Dev.#1.....: 21341.3 MH/s (87.87ms)

Rainbow Tables

Una tabla rainbow es una tabla de busqueda especial. Tiene precomputados los pares “hash - clave texto claro”.

Cuando necesito obtener una clave en texto claro, se hace una busqueda eficiente en las tablas precomputadas.

Ejemplo: ophcrack

- **Demo en: http://lasecwww.epfl.ch/~oechslin/projects/ophcrack/**

Rainbow Tables: ¿Cómo funciona?

- Es un trade-off entre espacio y tiempo.

- Los hashes se almacenan de una forma inteligente de tal manera que encontrar la clave correspondiente a una contraseña es rápido si tenemos la tabla pre-computada.