Resumen Teórica 8 : Autenticación

Tomás F. Melli

September 2025

Índice

1 Introducción

La autenticación constituye uno de los pilares fundamentales de la seguridad informática. Su propósito es establecer una asociación confiable entre una identidad y un objeto dentro de un sistema.

- **Identidad**: corresponde a una entidad externa, como puede ser una persona, un dispositivo o un proceso cliente que busca interactuar con el sistema.
- **Objeto**: representa la entidad informática interna con la cual se establece el vínculo, generalmente un identificador único de usuario (UID), que permite al sistema reconocer y distinguir a cada entidad autenticada.

Este proceso asegura que los recursos y servicios del sistema solo sean accesibles por quienes realmente están autorizados, constituyendo así la base para implementar otros mecanismos de seguridad como la autorización, el control de acceso y la trazabilidad de acciones.

1.1 Establecimiento de la identidad

El establecimiento de la identidad es un proceso que permite verificar que una entidad que solicita acceso es efectivamente quien dice ser. Este mecanismo no se limita al ámbito informático: lo utilizamos de manera constante en nuestra vida diaria, aun sin darnos cuenta.

Existen distintos factores mediante los cuales puede comprobarse la identidad:

- **Lo que la entidad sabe**: se refiere a información secreta conocida únicamente por el usuario legítimo, como una contraseña, un PIN de celular o una clave de alarma antirrobo.
- **Lo que la entidad tiene**: consiste en un elemento físico que la persona posee, por ejemplo un documento de identidad, una tarjeta magnética, una llave de auto o incluso algo más simple como una rosa en el ojal que sirve de señal acordada.
- **Lo que la entidad es**: corresponde a características biométricas únicas del individuo, tales como huellas dactilares, patrones de retina, rasgos faciales o la voz.
- **Dónde la entidad está**: hace referencia a la localización desde donde se realiza la autenticación, por ejemplo una terminal específica, una dirección IP determinada o la red desde la cual se establece la conexión.

La combinación de estos factores se conoce como **autenticación multifactor (MFA)**. Su propósito es incrementar la seguridad al exigir más de una prueba de identidad, reduciendo así el riesgo de suplantación o accesos no autorizados.

1.2 Sistema de Autenticación

Un **sistema de autenticación** puede representarse formalmente a través de un conjunto de cinco componentes:

- **A**: la información que prueba la identidad (por ejemplo, una contraseña que conoce el usuario).
- **C**: la información almacenada digitalmente y utilizada para validar la identidad (por ejemplo, un registro en una base de datos).
- **F**: la función que transforma los elementos de A en elementos de C (por ejemplo, una función hash que transforma una contraseña en su valor codificado).
- **L**: la función que prueba la identidad, verificando si los datos proporcionados coinciden con los almacenados.
- **S**: las funciones que permiten crear o modificar la información en A y C, como el registro de un nuevo usuario o el cambio de una contraseña.

Este modelo ofrece un marco conceptual para comprender cómo se construyen y gestionan los sistemas de autenticación en distintos contextos.

Ejemplo de (A, C, F, L, S)

Un ejemplo sencillo es un **sistema de claves almacenadas en texto plano y de forma transparente**:

- **A**: conjunto de cadenas de texto (*strings*) que representan las claves de los usuarios.
- **C = A**: la información almacenada es exactamente la misma que la ingresada, ya que no se aplica ninguna transformación.

- **F**: función de identidad (es decir, no se modifica el dato).
- **L**: función de comparación de igualdad de cadenas de texto, que valida si la clave ingresada coincide con la almacenada.
- **S**: funciones que permiten editar directamente el archivo donde se guardan las claves.

Este caso, aunque ilustrativo, es altamente inseguro en la práctica, ya que el almacenamiento en texto plano expone las credenciales de los usuarios sin ningún tipo de protección frente a accesos indebidos.

2 Claves y su Almacenamiento

En los sistemas de autenticación, las **claves** son elementos fundamentales para verificar la identidad de un usuario. Estas pueden adoptar diferentes formas:

- **Secuencia de caracteres:** combinaciones de letras y dígitos elegidos por el usuario, típicamente conocidas como contraseñas tradicionales.
- **Secuencia de palabras (*Pass-Phrases*):** frases completas que resultan más largas y, por ende, más seguras frente a ataques de fuerza bruta.
- **Algoritmos:** sistemas de autenticación basados en algoritmos, como *Challenge-Response* o contraseñas de un solo uso (*one-time passwords*), que no dependen de la memorización de la clave.

2.1 Almacenamiento de Claves

La seguridad de un sistema no depende únicamente de la elección de la clave, sino también de cómo se almacena:

- **Texto transparente:** las claves se guardan tal como son. Si el archivo es comprometido, todas las claves quedan expuestas.
- **Archivo cifrado:** las claves se cifran, necesitando claves adicionales en memoria para su descifrado. Esto puede generar problemas similares a los del almacenamiento en texto plano si las claves de cifrado se ven comprometidas.
- **One-way Hash:** las claves se almacenan mediante funciones hash unidireccionales. Incluso si el archivo es comprometido, un atacante debe adivinar la contraseña o invertir la función hash, aumentando significativamente la seguridad.

2.2 Ejemplo: UNIX Passwords (originales)

El sistema UNIX implementa un mecanismo de almacenamiento de claves basado en **hashes**:

- La clave se transforma mediante una de 4096 funciones de HASH, generando un string de 11 caracteres.
- Expresado formalmente:
  - **A:** conjunto de strings de 8 o menos caracteres ($\approx 6.9 \times 10^{16}$ combinaciones).
  - **C:** hash de la contraseña concatenado con 2 letras de ID del hash ($\approx 3.0 \times 10^{23}$ combinaciones).
  - **F:** 4096 versiones del DES modificado.
  - **L:** funciones de login como `login` o `su`.
  - **S:** funciones de cambio de contraseña como `passwd`, `nispasswd`, `passwd+`.

Detalle del algoritmo original

- La contraseña se utiliza como clave para cifrar un bloque de 64 bits en cero mediante DES.
- Este proceso se repite 25 veces, cifrando cada vez el resultado anterior.
- Se añade un **salt**, un valor basado en la hora del día, que selecciona una de 4096 variantes leves de DES para complicar el proceso.
- El *salt* se almacena en texto claro junto con la contraseña cifrada.

2.2.1 Salt

En criptografía, el **salt** son bits aleatorios utilizados como entrada adicional en una función derivadora de claves, junto con la contraseña. La salida de esta función se almacena como la versión cifrada de la contraseña.

- Incrementa la seguridad frente a ataques de precomputación (*rainbow tables*).
- Permite que contraseñas idénticas generen hashes diferentes.
- Puede ser usado como parte de una clave en cifrados u otros algoritmos criptográficos.
- Generalmente, la función derivadora de claves emplea una **función hash**, garantizando que incluso si se compromete el almacenamiento, la recuperación de la contraseña original sea extremadamente difícil.

3 Anatomía de un Ataque

En términos generales, un ataque sobre un sistema de autenticación tiene como **objetivo** encontrar un elemento $a \in A$ tal que:

$$\text{para cierto } f \in F, \quad f(a) = c \in C,$$

donde c está asociado a una entidad específica.

Existen dos formas principales de lograr este objetivo:

- **Directamente:** intentar determinar a de manera explícita.
- **Indirectamente:** dado que $l(a)$ es correcto si y solo si $f(a) = c \in C$ para algún c asociado a una entidad, se puede probar $l(a)$ repetidamente hasta encontrar un valor exitoso.

3.1 Prevención de Ataques

Para reducir el riesgo de comprometer el sistema, se utilizan varias estrategias:

- **Ocultar la mayor cantidad de variables** de la ecuación (a, f o c). Por ejemplo, sistemas Linux modernos emplean esquemas de *shadow passwords*, que ocultan c y evitan su acceso directo.
- **Bloquear el acceso a las funciones de verificación** ($l \in L$) o al resultado de $l(a)$. Un ejemplo es deshabilitar el acceso al *login* desde la red, reduciendo la superficie de ataque.

3.2 Ataques de Diccionario

Un **ataque de diccionario** consiste en probar todas las palabras de una lista predefinida (como un diccionario) para adivinar la contraseña:

- **Off-line:** conociendo f y c, se aplica f a múltiples $a \in A$ hasta que coincida con c. Herramientas típicas: *John-the-Ripper*, *Hashcat*.
- **On-line:** mediante acceso a las funciones L, se prueba repetidamente con diferentes $a \in A$ hasta que $l(a)$ sea exitoso. Herramientas: *THC Hydra*, *Medusa*, *ncrack*.

3.3 Ataques de Fuerza Bruta

Un **ataque de fuerza bruta** intenta recuperar una clave probando todas las combinaciones posibles hasta encontrar la correcta. Las variables que influyen en su eficacia incluyen:

- Longitud de la clave.
- Conjunto de caracteres utilizado.

Este método suele ser menos eficiente que un ataque de diccionario, aunque muchas veces se combinan ambas técnicas para aumentar la probabilidad de éxito.

3.4 ¿Por qué un Salt?

El uso de un **salt** mejora significativamente la seguridad de las contraseñas:

- Sin salt, cada clave genera un único hash, lo que facilita identificar cuando dos usuarios usan la misma contraseña.
- El espacio de contraseñas posibles, especialmente las fáciles de recordar, es reducido, permitiendo construir diccionarios de hashes rápidamente.
- Con un salt, el espacio de claves se amplía, complicando los ataques de diccionario, ya que es necesario generar el hash de cada contraseña para cada valor de salt posible.

3.5 Agregando Condimentos: Pepper

Además del salt, se puede introducir un **pepper**, un valor adicional que se almacena por separado:

- De esta forma, incluso si un atacante logra acceder a los hashes de las contraseñas mediante un ataque como SQL injection, no dispondrá de la información completa para reconstruir las claves.
- Combinando salt y pepper, se refuerza la resistencia del sistema frente a intentos de cracking.

4 Elección de Claves

La elección adecuada de claves es un factor crítico para la seguridad de cualquier sistema de autenticación. Existen varias estrategias para generar o seleccionar claves:

- **Selección aleatoria:** cada clave de A es equiprobable, garantizando máxima imprevisibilidad.
- **Claves pronunciables:** diseñadas para ser más fáciles de recordar, manteniendo cierto grado de aleatoriedad.
- **Claves elegidas por los usuarios:** el propio usuario selecciona la clave, lo que puede afectar la seguridad si se eligen opciones predecibles.

4.1 Claves Aleatorias

Las claves aleatorias son típicamente asignadas por el sistema:

- En algunos casos especiales pueden generarse claves cortas o con caracteres repetidos. Eliminar estas combinaciones reduce el espacio de búsqueda para un atacante.
- La principal desventaja es que resultan difíciles de recordar.
- La seguridad depende de la calidad del generador de números aleatorios utilizado.

4.2 Claves Pronunciables

Este enfoque busca generar secuencias que puedan pronunciarse fácilmente:

- Se basan en unidades de sonido (fonemas), por ejemplo combinaciones como cv, vc, cvc, vcv.
- Ejemplos de claves pronunciables: helgoret, pirtela.
- Secuencias aleatorias sin pronunciabilidad (asdlkj, aerje) no se consideran pronunciables y son menos memorables.
- El número de combinaciones posibles es relativamente bajo, por lo que su seguridad puede ser limitada frente a ataques sofisticados.

4.3 Claves Elegidas por el Usuario

Cuando los usuarios eligen sus propias claves, a menudo optan por opciones predecibles:

- Basadas en nombres, lugares o palabras de diccionario, incluyendo variantes simples como mayúsculas incorrectas, reemplazos de caracteres (pors$, 1por$l$).
- Claves cortas, que solo contienen letras o números.
- Acrónimos o datos personales (trabajo, hobbies, apodos, mascotas).
- Combinaciones comunes: palabra de diccionario con mayúscula inicial y número al final.

4.4 Chequeo Proactivo de Claves

Para mejorar la seguridad, muchos sistemas implementan un **chequeo proactivo de eficiencia de claves**, que permite:

- Detectar y rechazar automáticamente claves "débiles".
- Aplicar discriminación según sitio, usuario o política de seguridad.
- Realizar búsquedas mediante expresiones regulares o ejecutar programas externos, como *spell*, para verificar que la clave no sea fácilmente adivinable.
- Configuraciones flexibles que se adaptan a las necesidades del sistema.

4.4.1 Ejemplo: passwd+

Proporciona un lenguaje de script para verificar proactivamente la seguridad de las claves:

- `test length("$p") < 6` → si la clave tiene menos de 6 caracteres, se rechaza.
- `test infile("/usr/dict/words", "$p")` → si la clave está en el diccionario de palabras, se rechaza.
- `test !inprog("spell", "$p", "$p")` → si la clave no aparece en la salida del programa *spell*, se rechaza.

Este tipo de medidas reduce drásticamente la probabilidad de que los usuarios elijan claves predecibles, aumentando la seguridad general del sistema.

4.5 Requisitos de Complejidad y Gestión de Claves en Windows ('passfilt.dll')

En los sistemas Windows, la biblioteca `passfilt.dll` permite establecer requisitos opcionales de complejidad para las contraseñas. Esto ayuda a garantizar que las contraseñas sean suficientemente robustas frente a ataques de diccionario o fuerza bruta.

- Mensaje típico de la política de contraseñas:

  > "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 (A-Z)
  - Minúsculas (a-z)
  - Números (0-9)
  - Caracteres no alfabéticos (como !, $, #, %)

4.6 Acceso vía L (On-line)

El acceso en línea mediante la función L es **inevitable**, ya que los usuarios legítimos deben poder autenticarse. Para complicar posibles ataques:

- **Backoff o tarpit:** ralentizar las respuestas a intentos de autenticación fallidos.
- **Desconexión:** cortar la sesión después de múltiples intentos fallidos.
- **Deshabilitación:** bloquear temporalmente la cuenta (atención con cuentas críticas como `root`).
- **Jail:** restringir la sesión a un entorno limitado, evitando acceso a recursos críticos.

Estas técnicas permiten que los usuarios legítimos accedan, pero dificultan y enlentecen los ataques automatizados.

4.7 Vencimiento de Claves

Forzar cambios periódicos de contraseñas es otra estrategia de seguridad:

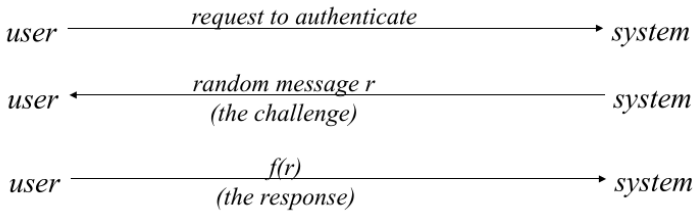
- Obliga a los usuarios a cambiar sus contraseñas después de cierto tiempo.
- Para evitar la **reutilización**, se pueden:
  - Guardar las contraseñas previas.
  - Bloquear cambios por un periodo mínimo.
- Permitir que los usuarios piensen en su nueva contraseña:
  - Avisar con anticipación.
  - No forzar el cambio justo en el momento de logon.
- Nota: algunos estudios indican que forzar cambios frecuentes puede ser contraproducente, ya que los usuarios tienden a elegir contraseñas más simples o predecibles.

5 Challenge-Response

El mecanismo de **Challenge-Response** es un esquema de autenticación que permite al sistema verificar la identidad de un usuario sin transmitir la contraseña o clave secreta directamente.

- Tanto el usuario como el sistema comparten una función secreta f. En la práctica, f puede ser una función conocida, pero con parámetros ocultos, como una clave criptográfica.
- Proceso de autenticación:
  1. El usuario solicita autenticarse ante el sistema.
  2. El sistema genera un **mensaje aleatorio** r, conocido como *challenge*, y se lo envía al usuario.
  3. El usuario aplica la función secreta f sobre r y envía el resultado $f(r)$ de vuelta al sistema. Este resultado se conoce como la *response*.
  4. El sistema verifica que la respuesta recibida coincide con la que él mismo calcula usando f y r. Si coincide, la autenticación es exitosa.
- Este esquema garantiza que la contraseña o clave secreta nunca se transmite por la red, reduciendo el riesgo de intercepción directa.

Esquema



5.1 Ataques a Challenge-Response

Aunque Challenge-Response protege la transmisión de claves, existen vulnerabilidades:

- **Ataques similares a los de claves fijas:** si un atacante conoce el challenge r y la respuesta $f(r)$, además de la función f, puede probar diferentes claves.
- **Vulnerabilidades por la forma de la respuesta:** en algunos casos, basta con conocer la estructura de la respuesta, ya que r se genera combinando datos parcialmente conocidos.
- **Ejemplo en Kerberos:** conociendo parte de la información usada para generar r (como nombre de usuario, hora, etc.), fue posible probar claves sobre sectores específicos de la respuesta, reduciendo la complejidad del ataque.

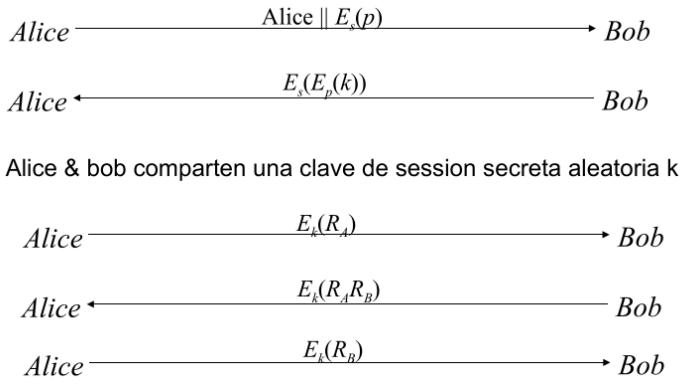
En resumen, Challenge-Response protege la transmisión de la clave secreta, pero su seguridad depende de mantener la **secreción de la función** f y de generar **challenges verdaderamente aleatorios e impredecibles**.

6 Protocolo EKE (Encrypted Key Exchange)

El **Protocolo EKE** permite a dos partes (Alice y Bob) establecer una **clave de sesión secreta** k de manera segura, incluso a través de un canal inseguro.

- Alice y Bob comparten una clave pública o función secreta previamente acordada.
- Alice genera una **clave de sesión aleatoria** k y la cifra usando la función compartida $E_p(k)$, enviándola a Bob.
- Bob descifra la información y, mediante un intercambio de mensajes cifrados con k y desafíos aleatorios (R_A, R_B), ambos verifican que la clave de sesión coincide y que ninguna tercera parte ha intervenido.
- Resultado: Alice y Bob comparten una clave de sesión segura que puede usarse para cifrado posterior de comunicaciones.

Esquema



7 Métodos de Autenticación

7.1 One-Time Passwords (OTP)

Las **contraseñas de un solo uso** son claves que solo pueden usarse una vez y se invalidan automáticamente después de su uso o expiran rápidamente.

Esquema

- **Ventajas:** Previenen ataques de repetición, ya que la misma contraseña no puede reutilizarse.
- **Problemas:**
  - Sincronización entre las partes generadora y verificadora.
  - Generación de claves pseudoaleatorias de buena calidad.
  - Distribución segura de las claves iniciales.
- **Tokens OTP:** dispositivos físicos o aplicaciones móviles que generan estas contraseñas.
- **Aplicaciones móviles:**

  - **TOTP (Time-based One-Time Password, RFC 6238):** genera contraseñas basadas en el tiempo. Referencia: RFC 6238 - TOTP Algorithm
  - **HOTP (HMAC-based One-Time Password, RFC 4226):** genera contraseñas basadas en un contador de eventos. Referencia: RFC 4226 - HOTP Algorithm

7.2 Biometría

La **biometría** se refiere a la medición automática de características biológicas o de comportamiento de un individuo para autenticarlo.

- **Huellas digitales:**
  - Captura óptica o eléctrica.
  - Se mapean en un grafo y se comparan con una base de datos.
  - Se usan algoritmos de aproximación debido a errores de captura.
- **Voz:**
  - *Verificación:* compara patrones estadísticos de la voz del usuario.
  - *Reconocimiento:* identifica al usuario verificando el contenido de las respuestas, independientemente de quién las pronuncie.
- **Ojos:** patrones del iris, muy únicos pero intrusivos de medir.
- **Cara:**
  - Uso de imágenes o características específicas.
  - Detección de patrones, como longitud del labio o ángulo de las cejas.
- **Secuencias de tipeo (keystroke dynamics):**
  - Analiza presión sobre las teclas y cadencia de tipeo.
  - Consideradas únicas por usuario.

La biometría complementa los métodos tradicionales de autenticación (contraseñas, tokens), proporcionando una capa adicional de seguridad basada en características personales difíciles de reproducir por un atacante.

7.3 Localización del Usuario

La autenticación basada en **localización** valida la identidad de un usuario determinando **dónde se encuentra físicamente**.

- Requiere **hardware específico**, como GPS o sistemas de posicionamiento en dispositivos móviles.
- El usuario debe **enviar continuamente información de ubicación** para que el sistema pueda verificar su presencia en el lugar esperado.
- Este método añade una capa adicional de seguridad, asegurando que un usuario solo pueda acceder desde ubicaciones autorizadas.

7.4 Múltiples Métodos de Autenticación

La tendencia actual en seguridad es combinar **varios métodos de autenticación** para aumentar la robustez.

- Ejemplo: **GPS + token**, verificando la ubicación y la posesión de un dispositivo físico.
- Se pueden aplicar diferentes métodos según la **tarea o nivel de acceso**, incluyendo controles basados en **hora, terminal o ubicación**.
- Cada vez se usan más combinaciones con **teléfonos celulares**, aprovechando sensores y conectividad.

7.5 PAM – Pluggable Authentication Modules

PAM es un mecanismo flexible que permite a los sistemas UNIX/Linux autenticar usuarios mediante **módulos independientes**, sin depender de un mecanismo específico.

- Permite utilizar distintos métodos de autenticación: desde archivos simples como `/etc/passwd`, hasta dispositivos de hardware, servidores LDAP o bases de datos.
- Configurable para cada servicio del sistema.
- Revisa un **repositorio de métodos** mediante la librería `pam_authenticate`.
- Los módulos de PAM pueden clasificarse como:
  - `sufficient`: Si el módulo acepta, la autenticación se considera exitosa.
  - `required`: Si falla, la autenticación falla, pero se ejecutan todos los módulos requeridos antes de reportar la falla.
  - `requisite`: Igual que `required`, pero si falla no se ejecutan los demás módulos.
  - `optional`: Solo se invoca si todos los demás fallan.

Ejemplo de Configuración PAM

Listing 1: Configuración PAM para rlogin y SMTP con LDAP

```
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

# Autenticaci n SMTP contra un server LDAP
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
```

PAM permite definir **políticas distintas para cada servicio**, haciendo que la autenticación sea flexible, modular y fácilmente ampliable.

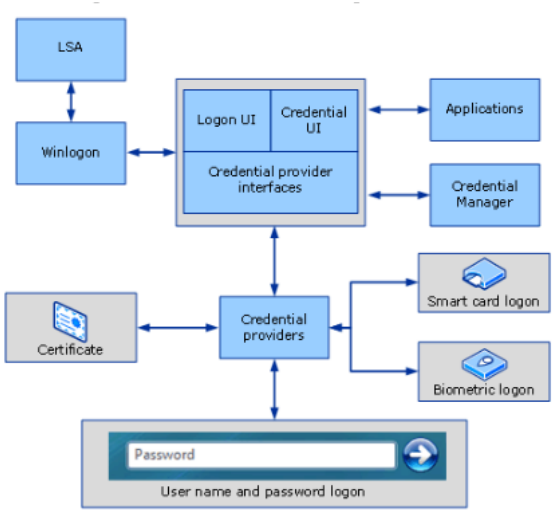
7.6 MS Credential Providers

Los **Microsoft Credential Providers** son un mecanismo de autenticación modular introducido a partir de **Windows Vista** y **Windows Server 2008**, diseñado para reemplazar el antiguo GINA (Graphical Identification and Authentication).

- **Objetivo:** Permitir que diferentes métodos de autenticación se integren de manera uniforme en Windows, proporcionando flexibilidad y extensibilidad para el inicio de sesión de usuarios y servicios.
- **Características principales:**
  - **Modularidad:** Cada método de autenticación se implementa como un provider independiente (contraseñas, PIN, tarjetas inteligentes, biometría, etc.).
  - **Integración uniforme:** Todos los providers interactúan con la interfaz de inicio de sesión de Windows de forma estándar.
  - **Flexibilidad para desarrolladores:** Posibilidad de crear providers personalizados para soportar métodos corporativos, tokens OTP, LDAP, etc.
  - **Compatibilidad:** Funciona en inicio de sesión local, remoto (RDP) y desbloqueo de estaciones de trabajo.
- **Funcionamiento general:**
  1. Windows carga los Credential Providers instalados en el sistema.
  2. Cada provider ofrece uno o más tipos de credenciales al usuario.
  3. El usuario selecciona o introduce sus credenciales.
  4. Windows pasa la información al provider correspondiente, que valida la identidad y devuelve un resultado al sistema.

- **Ventajas sobre GINA:**
  - Mayor seguridad y estabilidad.
  - Permite múltiples métodos de autenticación simultáneos y personalizados.
  - Facilita la adopción de nuevas tecnologías como Windows Hello y autenticación biométrica integrada.

Esquema



8 Control de Acceso

El **control de acceso** es un componente que no solo se limita a autenticar usuarios, sino que también regula **qué recursos pueden usar** y **cómo pueden interactuar con ellos**.

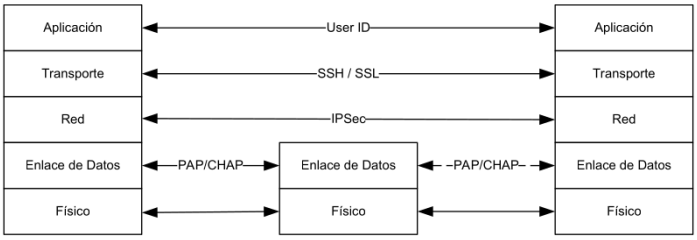
8.1 Autenticación en varios niveles

La autenticación no se realiza únicamente a nivel de aplicación. Puede implementarse en distintos niveles de la infraestructura, aumentando la seguridad mediante capas de protección:

- **Aplicación:** identificación mediante User ID y contraseña o credenciales específicas dentro del software.
- **Transporte:** protocolos como SSH o SSL que autentican y cifran la comunicación entre sistemas.
- **Red:** IPSec para garantizar la autenticidad y confidencialidad de los paquetes que circulan en la red.
- **Enlace de Datos:** protocolos de autenticación como PAP o CHAP que verifican la identidad en conexiones punto a punto.
- **Físico:** control de acceso a instalaciones, hardware o dispositivos críticos, asegurando que solo personal autorizado pueda interactuar con ellos.

Cada nivel añade una capa de protección adicional, dificultando que un atacante logre comprometer el sistema mediante un único punto de fallo.

Esquema



8.2 Gestión de usuarios y contraseñas

El control de acceso requiere políticas y sistemas que **gestionen quién tiene acceso a qué**, de manera segura y eficiente:

- **Identity Management (IDM):** Sistema integrado de políticas y procesos organizacionales que facilita y controla el acceso a los sistemas de información y a las instalaciones físicas. Incluye la creación, modificación y eliminación de cuentas, roles y permisos, asegurando que cada usuario tenga únicamente el acceso necesario.
- **Privileged Identity Management (PIM):** Orientado al manejo de contraseñas y privilegios de administración de la infraestructura IT. Garantiza que las cuentas con privilegios críticos sean gestionadas de forma segura, con seguimiento, auditoría y control de uso, reduciendo el riesgo de comprometer sistemas sensibles.

En conjunto, **autenticación multicapas** y **gestión de identidades** forman la base de un control de acceso robusto, asegurando que los recursos de una organización estén protegidos frente a accesos no autorizados y abusos internos.

8.3 Varios Métodos de Autenticación

8.3.1 RADIUS (Remote Authentication Dial-In User Service)

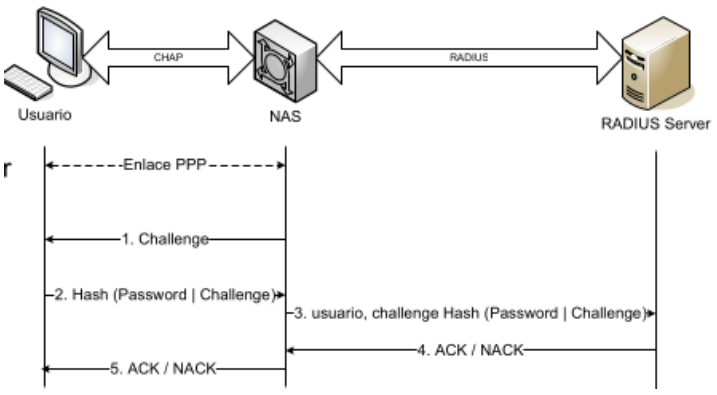
RADIUS es un protocolo diseñado para transmitir información de autenticación entre un **NAS (Network Access Server)** y un servidor central de autenticación.

- Soporta los protocolos **PAP** y **CHAP** para la verificación de credenciales.
- Integra funciones de:
  - **Autenticación:** verificar la identidad del usuario.
  - **Autorización:** determinar los recursos que puede utilizar.
  - **Accounting:** registrar el uso de recursos.
- Todo viaja dentro de los mismos paquetes, simplificando la comunicación pero requiriendo seguridad adicional.

Ejemplo de flujo de autenticación CHAP sobre PPP:

1. El servidor envía un **Challenge** al usuario.
2. El usuario genera un **Hash (Password — Challenge)**.
3. El usuario envía su **usuario** y el **Hash calculado** al servidor.
4. El servidor responde con **ACK** si la autenticación es correcta, o **NACK** si falla.

Esquema



8.3.2 SSO (Single Sign-On)

- Permite que un usuario inicie sesión **una sola vez** y acceda a múltiples recursos sin volver a autenticarse.
- Centraliza la autenticación, facilitando la administración de credenciales y mejorando la experiencia del usuario.

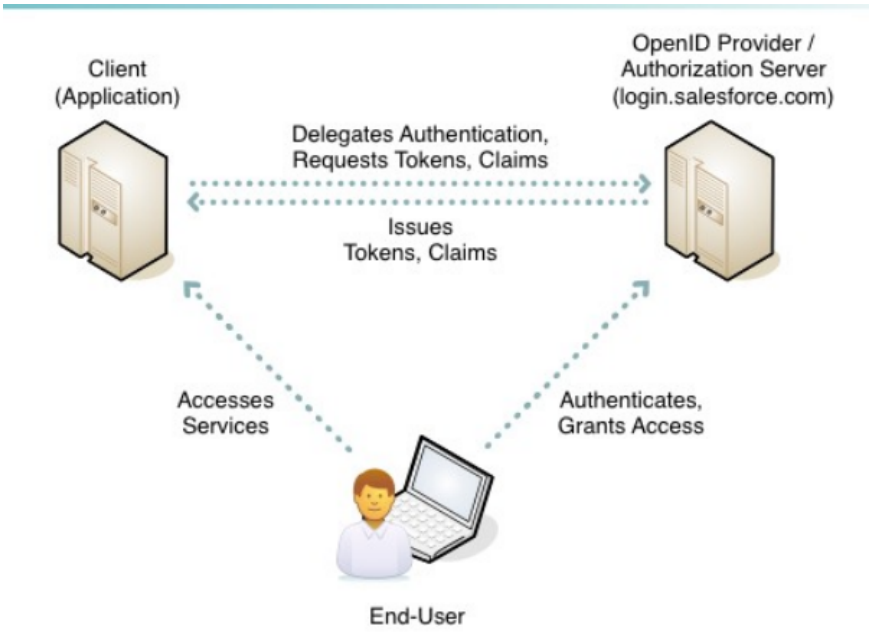
8.3.3 Herramientas de gestión segura de claves

- Permiten almacenar y acceder de manera segura a contraseñas, tokens y otra información sensible.
- Ejemplo: KeePassXC.

8.3.4 OpenID Connect

- Protocolo de autenticación basado en OAuth 2.0, que permite iniciar sesión en aplicaciones web usando un **proveedor de identidad centralizado**.
- Facilita el login único sin necesidad de múltiples contraseñas.
- Ejemplo de implementación en Argentina: Autenticar.gob.ar.

Esquema



8.3.5 WebAuthn

- Estándar promovido por **W3C** y la **FIDO Alliance**, soportado por Google, Microsoft y Mozilla.
- Permite autenticación en aplicaciones web usando mecanismos basados en hardware o software, según las capacidades del dispositivo cliente.
- Mejora la seguridad frente a phishing y robo de contraseñas.
- Más información: W3C WebAuthn.

8.3.6 Passkeys

- Utilizan **TPM** o **TEE** para almacenar de forma segura los pares de claves del usuario para cada sitio.
- Pueden sincronizarse en la nube para usar múltiples dispositivos sin contraseñas.
- Si no se usa la nube, se requieren dispositivos físicos como **YubiKey**.
- Referencias:
  - FIDO Alliance - Passkeys
  - Yubico - Qué es un Passkey
  - Yubico - Mejores prácticas Passkeys
  - Yubico Blog - Passkeys y seguridad

Estos métodos representan la evolución de la autenticación, desde protocolos tradicionales de red como RADIUS hasta estándares modernos sin contraseñas como WebAuthn y Passkeys, mejorando tanto la **seguridad** como la **experiencia del usuario**.

Esquema

9 Almacenamiento de Claves de Usuarios

El almacenamiento seguro de las contraseñas de usuarios es un componente crítico de la seguridad de los sistemas operativos. Tanto **Windows** como **Linux** utilizan distintos mecanismos para proteger las credenciales, cada uno con ventajas, limitaciones y evolución a lo largo del tiempo.

9.1 Windows

9.1.1 Lan Manager Hash (LM Hash)

- Convierte toda la contraseña a **mayúsculas** antes de generar el hash.
- Set de caracteres permitido: `ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 %#$^&*()_=+'~[]{}\|:;"'<>,.?/`
- El hash se divide en dos bloques de 7 caracteres; si tiene menos de 14, se paddea con `null`.
- Utiliza **DES**, encriptando el texto fijo "KGS!@$
- Resultado del hash: 16 bytes.
- No utiliza **salt**.

9.1.2 NT Hash

- Distingue entre **mayúsculas y minúsculas**.
- Utiliza **MD4**.
- Longitud de hasta 128 caracteres, no requiere padding.
- No utiliza **salt**.
- No soportado por defecto en Windows 95, 98 y ME. NT4 lo soporta desde SP4. (Los sistemas Windows 95, 98 y ME no incluyen soporte nativo para el NT Hash. Es decir, en esos sistemas, aunque el NT Hash exista como estándar, el sistema operativo no lo puede generar ni usar para almacenar contraseñas por defecto. En Windows NT 4.0, el soporte para NT Hash fue agregado a partir del Service Pack 4 (SP4). Antes de instalar SP4, NT4 tampoco podía usarlo.)

9.2 Linux

9.2.1 MD5

- Contraseñas almacenadas en archivo solo accesible por root (`/etc/shadow`).
- Ejemplo de clave cifrada: `$1$O34Nj3aF$3ecTiyOvCOAqhtaEaNOyG0`
- Composición:
  - `$1$`: indica que se utiliza MD5.
  - `$O34Nj3aF$`: salt utilizado.
  - `3ecTiyOvCOAqhtaEaNOyG0`: hash propiamente dicho.

9.2.2 OpenBSD Bcrypt

- **Adaptabilidad temporal:** OpenBSD Bcrypt implementa un mecanismo por el cual el tiempo computacional necesario para cifrar una contraseña con una función de una vía puede variar a medida que aumenta la velocidad del hardware. Esto asegura que los hashes sigan siendo resistentes a ataques de fuerza bruta incluso con sistemas más rápidos.
- **Dificultad diferenciada:** Las contraseñas de usuarios con mayores privilegios pueden configurarse para ser más difíciles de calcular, aumentando la protección de cuentas críticas.
- **Resistencia a ataques futuros:** Su diseño adaptativo permite mantener la seguridad de las contraseñas a lo largo del tiempo, evitando que la mejora en hardware reduzca la efectividad del hash.
- **Referencia técnica:** Para más detalles, consultar el paper original: "A Future-Adaptable Password Scheme".

9.2.3 Evolución de algoritmos en Linux

Linux ha incorporado diversos algoritmos de hashing para el almacenamiento de contraseñas, buscando mejorar la seguridad frente a ataques de fuerza bruta y ataques basados en diccionarios. Los algoritmos soportados incluyen:

- **MD5 (ID 1):** Algoritmo clásico de hashing, ampliamente utilizado pero considerado menos seguro frente a ataques modernos.
- **Blowfish (ID 2a):** Introduce un esquema más fuerte que MD5, con mayor resistencia a ataques de fuerza bruta.
- **SHA-256 (ID 5):** Algoritmo de la familia SHA-2, que produce hashes más largos y seguros que MD5.
- **SHA-512 (ID 6):** Versión más robusta de SHA-2, con hashes de 512 bits.

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

Una característica importante de estos algoritmos es que **el número de *rounds* (iteraciones de hashing) se puede configurar**, permitiendo aumentar la dificultad de calcular el hash de manera deliberada. Por defecto, SHA-512 utiliza 5000 rondas, pero para aplicaciones web modernas, donde la seguridad adicional es crítica, se puede incrementar este número significativamente (por ejemplo, hasta 100000 rondas), dificultando enormemente los ataques automatizados.

Esta evolución refleja la necesidad de ajustar los mecanismos de seguridad de acuerdo con el aumento de la capacidad computacional disponible y las amenazas actuales, manteniendo la protección de las contraseñas almacenadas en Linux.

9.3 Otros mecanismos modernos de hashing de contraseñas

Para enfrentar ataques cada vez más sofisticados y aprovechar las capacidades del hardware moderno, se han desarrollado algoritmos de derivación de claves avanzados que superan las limitaciones de los métodos tradicionales como MD5 o SHA-512. Entre los más destacados se encuentran:

- **PBKDF2 (Password-Based Key Derivation Function 2):** Función de derivación de claves que aplica iterativamente un hash criptográfico para aumentar la dificultad de ataques de fuerza bruta. Permite configurar el número de iteraciones para ajustar la seguridad según las necesidades del sistema.
- **Scrypt:** Algoritmo que además de incrementar la complejidad computacional, requiere un uso significativo de memoria, dificultando los ataques realizados mediante hardware especializado (como GPUs o ASICs).
- **Argon2:** Ganador del Password Hashing Competition. Ofrece dos variantes:
  - Una enfocada en dificultar ataques por **side-channels**.
  - Otra diseñada para resistir ataques mediante GPU, aumentando la seguridad frente a ataques masivos y paralelos.

- Más información y comparativas de algoritmos: https://password-hashing.net/.

Estos mecanismos modernos permiten que las contraseñas sean mucho más resistentes a ataques de fuerza bruta, ataques por diccionario y ataques optimizados por hardware, asegurando que incluso con la mejora en velocidad de cómputo, las credenciales sigan siendo seguras.

10 Cracking de Claves

El cracking de claves consiste en intentar recuperar contraseñas a partir de sus hashes utilizando distintas herramientas y técnicas. Las más conocidas incluyen **John the Ripper**, **Hashcat** y **Rainbow Tables**. Estas herramientas permiten probar diccionarios, aplicar reglas, o incluso realizar ataques de fuerza bruta, dependiendo del algoritmo de hashing y del poder computacional disponible.

10.1 John The Ripper

John the Ripper es una de las herramientas más conocidas para crackear contraseñas en sistemas Unix, Linux y Windows. Permite recuperar contraseñas usando distintos métodos, incluyendo diccionarios, reglas y fuerza bruta.

10.1.1 Características principales

- Soporta múltiples tipos de hashes: Crypt, MD5, Blowfish, LM, NTLM, entre otros.
- Permite ataques **offline**, usando archivos de hashes extraídos del sistema.
- Optimizado para CPUs modernas y soporta múltiples hilos (multithreading).
- Capacidad de benchmarking para medir la velocidad de crackeo.

10.1.2 Tipos de ataques

- **Ataque de diccionario:** Prueba contraseñas a partir de listas de palabras y puede aplicar reglas de transformación.
- **Ataque de fuerza bruta:** Intenta todas las combinaciones posibles de caracteres.
- **Ataques combinados:** Mezcla diccionario y fuerza bruta, o aplica modificaciones a palabras del diccionario.

10.1.3 Benchmarking

En este contexto, **benchmarking** significa medir el rendimiento de un programa o sistema bajo condiciones controladas. En el caso de **John the Ripper** y **Hashcat**, se refiere a probar qué tan rápido pueden crackear diferentes tipos de hashes usando el hardware disponible (CPU o GPU).

Específicamente:

- Se ejecuta el software sobre un conjunto de hashes de prueba.
- Se mide la velocidad en contraseñas por segundo (c/s) o hashes por segundo (H/s, kH/s, MH/s).
- Permite comparar el rendimiento entre algoritmos (DES, MD5, LM, bcrypt, etc.) y entre distintos equipos o configuraciones.
- Ayuda a estimar cuánto tiempo tomaría un ataque de fuerza bruta o diccionario en condiciones reales.

En resumen, **benchmarking** en cracking de claves es como un test de velocidad para ver qué tan rápido un atacante podría recuperar contraseñas usando un determinado hardware y algoritmo.

Ejemplo

```
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
```

10.2 Hashcat

Hashcat es una herramienta avanzada de recuperación de contraseñas mediante fuerza bruta y ataques basados en diccionarios, optimizada para aprovechar **CPU y GPU** para acelerar el proceso. Es muy popular en auditorías de seguridad y pruebas de penetración.

- Compatible con decenas de tipos de **hashes**: MD5, SHA1, SHA256, SHA512, LM, NTLM, bcrypt, etc.
- Permite ejecutar ataques **offline**, sin interacción con el sistema original.
- Funciona en Windows, Linux y macOS.

10.2.1 Tipos de ataques

- **Ataques de diccionario:** Prueba contraseñas a partir de listas de palabras, aplicando reglas como cambios de mayúsculas/minúsculas, sustitución de caracteres o adición de números.
- **Ataques de fuerza bruta:** Se prueban todas las combinaciones posibles hasta encontrar la contraseña. Es más lento que un diccionario, pero exhaustivo.
- **Ataques combinados e híbridos:** Mezcla diccionarios con fuerza bruta, o aplica reglas a palabras de diccionario para generar nuevas variantes.

10.2.2 Rendimiento y benchmarking

- Hashcat puede aprovechar **GPUs modernas** para alcanzar velocidades de cracking muy superiores a las CPUs.
- El rendimiento depende del tipo de hash, hardware disponible y número de threads o unidades de cálculo (CUDA, OpenCL).

Ejemplo

```
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)
```

En un PC gamer con i7 + GeForce GTX1080 Ti, los rendimientos aumentan aún más:

```
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)
```

10.2.3 Funcionalidades avanzadas

- **Soporte de reglas personalizadas:** modifica palabras de diccionario con gran flexibilidad.
- **Distribución de carga:** Hashcat puede ejecutarse en varios dispositivos al mismo tiempo.
- **Capacidad de reanudación:** permite continuar tareas interrumpidas.
- **Soporte para hashes salted y KDFs:** PBKDF2, bcrypt, scrypt, entre otros.

10.2.4 Casos de uso

- Auditoría de seguridad para detectar contraseñas débiles.
- Recuperación de contraseñas olvidadas.
- Investigación en seguridad de aplicaciones web y sistemas operativos.

10.3 Rainbow Tables

Las **Rainbow Tables** son tablas de búsqueda especialmente diseñadas para acelerar la recuperación de contraseñas a partir de sus hashes. A diferencia de los ataques de fuerza bruta, que prueban todas las combinaciones posibles, las rainbow tables utilizan cálculos precomputados de *hash–clave en texto claro* para encontrar rápidamente contraseñas conocidas.

10.3.1 Funcionamiento

- Se precomputan cadenas de hash a partir de contraseñas posibles, creando una tabla que relaciona el *hash* con la contraseña original.
- Cada hash de contraseña almacenado en el sistema puede ser buscado eficientemente en la tabla para recuperar la contraseña en texto claro.
- Este enfoque es un **trade-off entre espacio y tiempo**: se necesita mucho almacenamiento, pero se reduce el tiempo de búsqueda.
- Las rainbow tables son especialmente útiles contra hashes sin salt.

10.3.2 Ejemplo de uso

- Herramienta: **Ophcrack**.
- Página de referencia y demo: http://lasecwww.epfl.ch/ oechslin/projects/ophcrack/

10.3.3 Ventajas y limitaciones

- **Ventajas:** permite recuperación rápida de contraseñas usando hashes precomputados.
- **Limitaciones:** grandes requerimientos de almacenamiento; ineficaz si los hashes utilizan **salt**, ya que cada salt requiere su propia tabla.
- Se combina frecuentemente con ataques de diccionario y fuerza bruta para cubrir hashes más complejos.

Esquema