

Autenticación

Víctor Ponz



Institut Educació Secundària

El Caminàs

Curso de especialización
en Ciberseguridad

Contenidos

1 Autenticación HTTP	3
1.1 OAuth0	6
1.2 Single Sign-On (SSO)	12
1.3 Contraseñas de un sólo uso (One-time password)	13
1.4 FIDO (Fast IDentity Online)	14
1.5 JSON web tokens	16
1.6 Autenticación en dos factores	18
1.7 Cómo almacenar de forma segura la contraseña	18
1.7.1 Hashing vs. Encriptación	19
1.7.2 Cómo los atacantes descifran los hash de las contraseñas	19
1.7.3 Contraseñas de texto sin formato	20
1.7.4 Contraseñas encriptadas	21
1.7.5 Funciones hash obsoletas	21
1.7.6 Benchmark para recuperar contraseñas Md5	23
1.7.7 Función hash inapropiada	24
1.7.8 Mejorando SHA512	25
1.7.9 Salting	25
1.7.10 Pimentar	27
1.7.11 Algoritmos modernos	30
1.7.12 Conclusión	32

1 Autenticación HTTP

Práctica 1 Realiza y documenta este punto<

También merece la pena conocer el tipo de autenticación HTTP que provee de un mecanismo sencillo para acceder a recursos protegidos por usuario y contraseña.

Según la [Wikipedia](#) En el contexto de una transacción HTTP, la **autenticación de acceso básico** es un método diseñado para permitir a un [navegador web](#), u otro programa cliente, proveer credenciales en la forma de **usuario** y **contraseña** cuando se le solicita una [página](#) al [servidor](#).

Ha sido diseñado con el fin de permitir a un navegador web o programa **aportar credenciales** basadas en nombre de usuario y [contraseña](#), que le permitan autenticarse ante un determinado servicio. El sistema es muy sencillo de implementar, pero sin embargo no está pensado para ser utilizado sobre líneas públicas, debido a que las credenciales que se envían desde el cliente al servidor, aunque no se envían directamente en texto plano, se envían únicamente codificadas en [Base64](#), lo que hace que se puedan obtener fácilmente debido a que es perfectamente reversible, es decir, una vez que se posee el texto codificado es posible obtener la cadena original sin ningún problema, por lo que la información enviada no es cifrada ni segura.

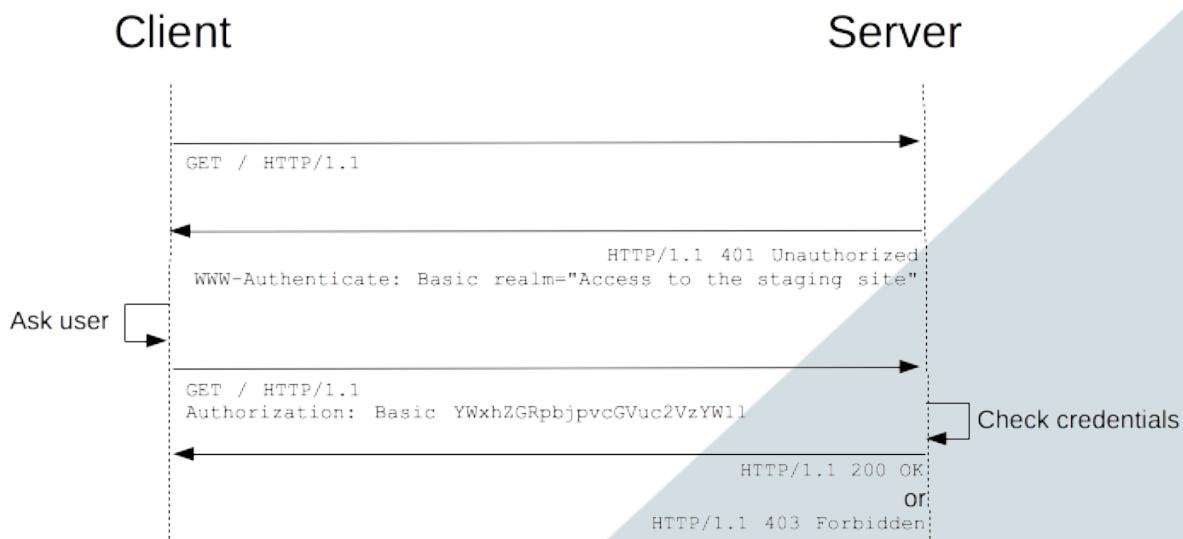


Figura 1: Proceso de autenticación

Veamos un ejemplo de autenticación básica en PHP

```
<?php

$valid_passwords = array ("mario" => "qwerty");
$valid_users = array_keys($valid_passwords);

$user = $_SERVER['PHP_AUTH_USER'];
$pass = $_SERVER['PHP_AUTH_PW'];

$validated = (in_array($user, $valid_users)) && ($pass ==
    $valid_passwords[$user]);

if (!$validated) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Not authorized");
}

// If it arrives here, it is a valid user.
echo "<p>Welcome $user.</p>";
echo "<p>Congratulation, you are into the system.</p>";
```

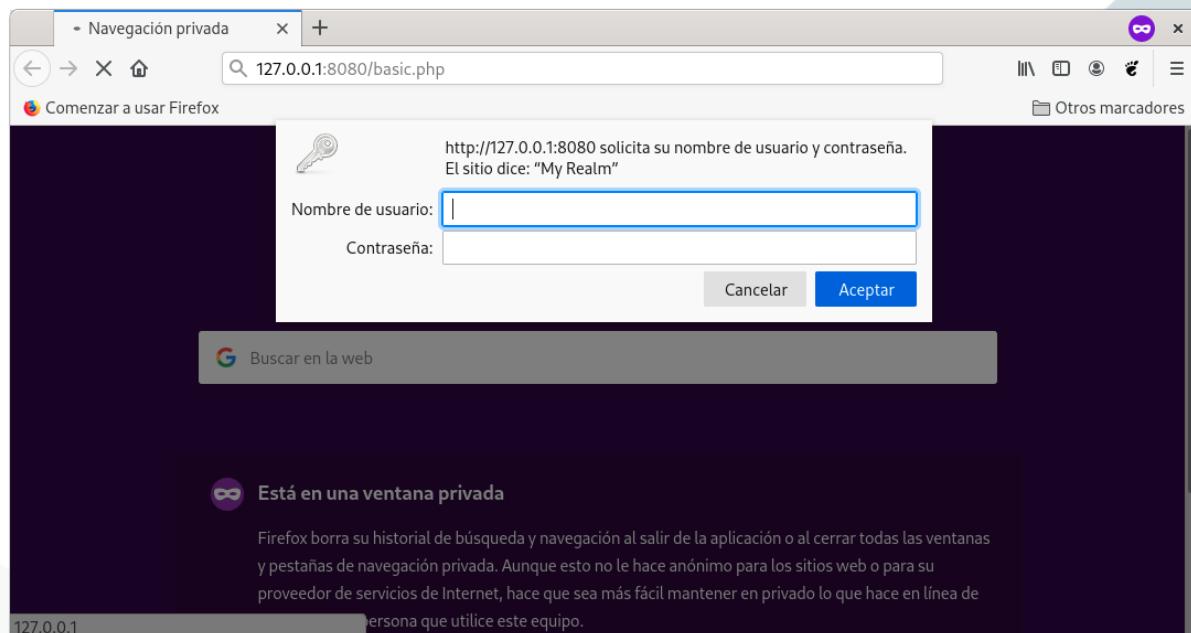


Figura 2: Petición de credenciales

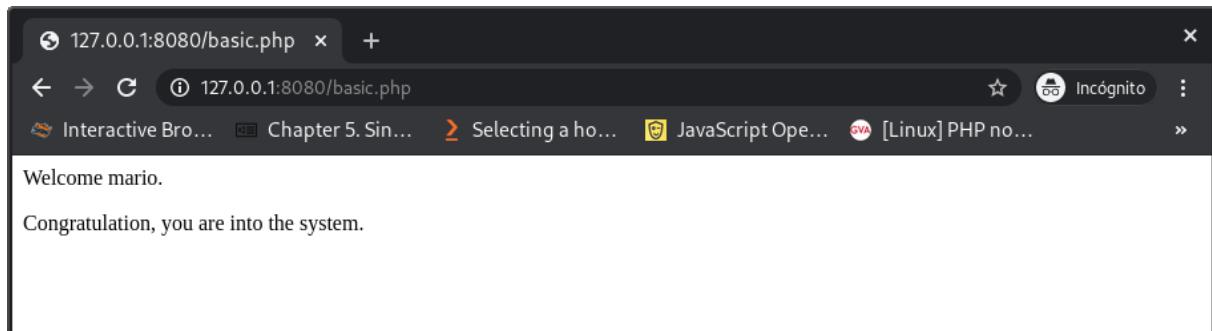


Figura 3: Autenticado con éxito

A partir del momento en que se produce la autenticación, cliente y servidor se intercambian las credenciales mediante la cabecera `Authorization` con el valor `Basic bWFyaW86Y2FyYm9uZWxs`. Como está codificado en base64, es muy fácil obtener las credenciales si estas se envían en texto plano. Por ello es muy importante que el **protocolo de la página sea HTTPS** para que de esta forma las credenciales viajen encriptadas.

Es muy fácil decodificar base64. Por ejemplo en <https://www.base64decode.org/>

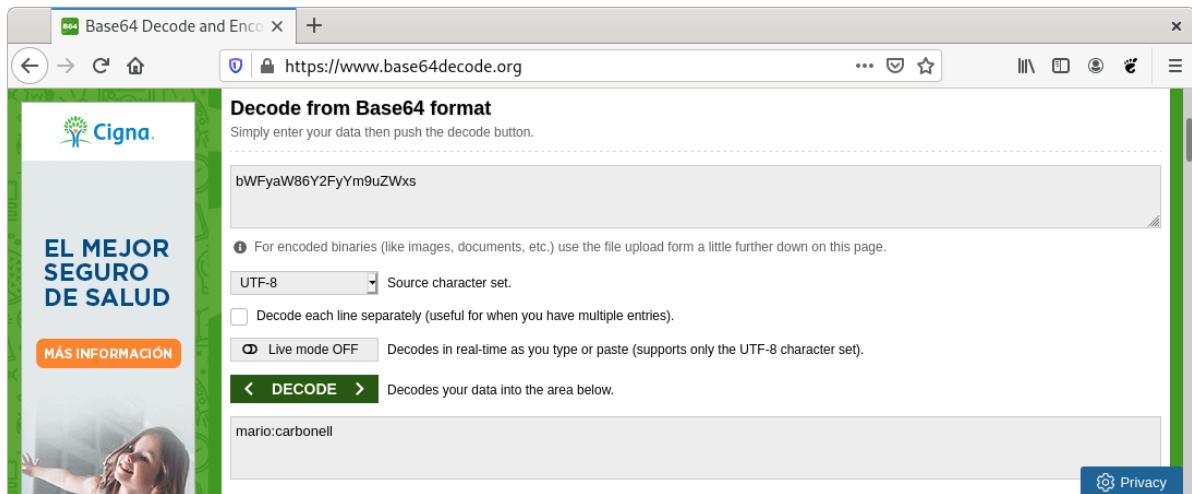


Figura 4: Decodificación

Este método se puede emplear para una intranet o para una parte de la aplicación en la que sea necesario iniciar sesión, añadiendo una capa más de seguridad, porque se deben realizar dos autorizaciones: la primera basada en HTTP y la segunda, como vimos anteriormente, mediante **sesiones**

Práctica 2 Configura apache para que al directorio /protegido sólo se pueda acceder mediante un usuario y contraseña siguiendo las instrucciones detalladas en Password protect a directory using basic authentication. Documenta la configuración e instalación con una entrada en tu blog

1.1 OAuth

OAuth es un estándar para permitir delegar la autenticación a terceras partes. Lo usamos en servicios del día a día cuando, por ejemplo hacemos login en un servicio con las credenciales de Google, como muestra la imagen siguiente.

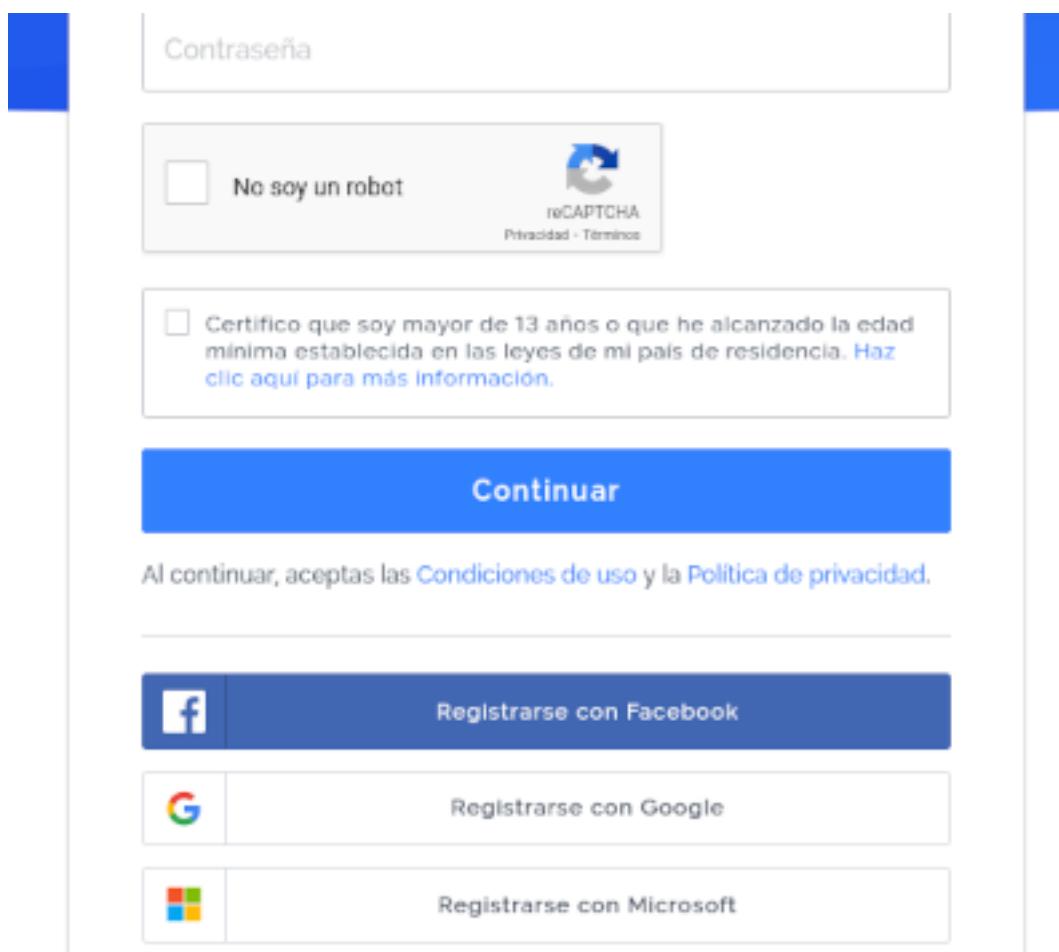


Figura 5: oauth

Práctica 3 Realiza este punto. Como resultado debes realizar una entrada de blog y un repositorio en GitHub

El flujo de autorización es el siguiente: Nuestra aplicación redirecciona a los usuarios para pedir permiso y acceder a parte de su información. Si los usuarios aceptan, este servicio nos devuelve un token de acceso que podemos utilizar para consumir la información protegida de nuestros usuarios.

Muchas compañías y servicios populares (Google, Twitter, Github, Spotify y muchos otros) aprovechan muy bien este tipo de mecanismos para compartir sus APIs con otras aplicaciones o incluso para otros proyectos de las mismas marcas.

Crear una aplicación

Vamos a crear una aplicación web que use la autorización OAuth de Google.

Antes de que pueda comenzar el proceso de OAuth, primero debes registrar una nueva aplicación con el servicio. Desde aquí puedes generar las [credenciales](#) para tu aplicación.

Al registrar una nueva aplicación, normalmente registra información básica como el nombre de la aplicación, el sitio web, un logotipo, etc. Además, debes registrar una URI de redireccionamiento que se utilizará para redirigir a los usuarios al servidor web, el navegador o las aplicaciones móviles.

Redirigir URI

The screenshot shows the Google Cloud Platform interface for creating a new application. At the top, there's a search bar labeled 'Buscar API y servicios'. Below it, a sidebar on the left lists 'Credenciales' (Credentials), 'Claves de API' (API keys), and 'ID de clientes OAuth 2.0'. The main content area is titled 'Credenciales' and contains three sections: 'Clave de API' (API key), 'ID de cliente de OAuth' (OAuth client ID), and 'Cuenta de servicio' (Service account). A button '+ CREAR CREDENCIALES' is at the top right, and a 'Borrar' (Delete) button is at the bottom right. A note 'Recuerda configurar' (Remember to configure) is visible next to the 'Clave de API' section. On the far right, there's a 'Restricciones' (Restrictions) column.

Figura 6: Crear una aplicación

Escribid <http://localhost:8080/oauth2callback.php> en el apartado URI. Esta es la página a la que redireccionará Google cuando haya finalizado el proceso de login.

The screenshot shows the configuration page for a new OAuth 2.0 client. At the top, there's a search bar labeled "Buscar API y servicios". Below it, a navigation bar includes "First Project ▾", "ID de cliente para Aplicación web" (with a back arrow), "DESCARGAR JSON", "RESTABLECER SECRETO", and "BORRAR".

Nombre *: Cliente web 1

El nombre de tu cliente de OAuth 2.0. Este nombre solo se usa para identificar al cliente en la consola y no se mostrará a los usuarios finales.

Orígenes autorizados de JavaScript

Para usar con solicitudes de un navegador

+ AGREGAR URI

URI de redireccionamiento autorizados

Para usar con solicitudes de un servidor web

URI: http://localhost:8080/oauth2callback.php

+ AGREGAR URI

GUARDAR CANCELAR

On the right side, there's a summary table:

ID de cliente	388368801510-nf1oC
Secreto del cliente	zu08LULyYvk9RFtx3p
Fecha de creación	21 de febrero de 202

Figura 7: ID de cliente

Cuando crees el Cliente, anota el ID de Cliente y el secreto

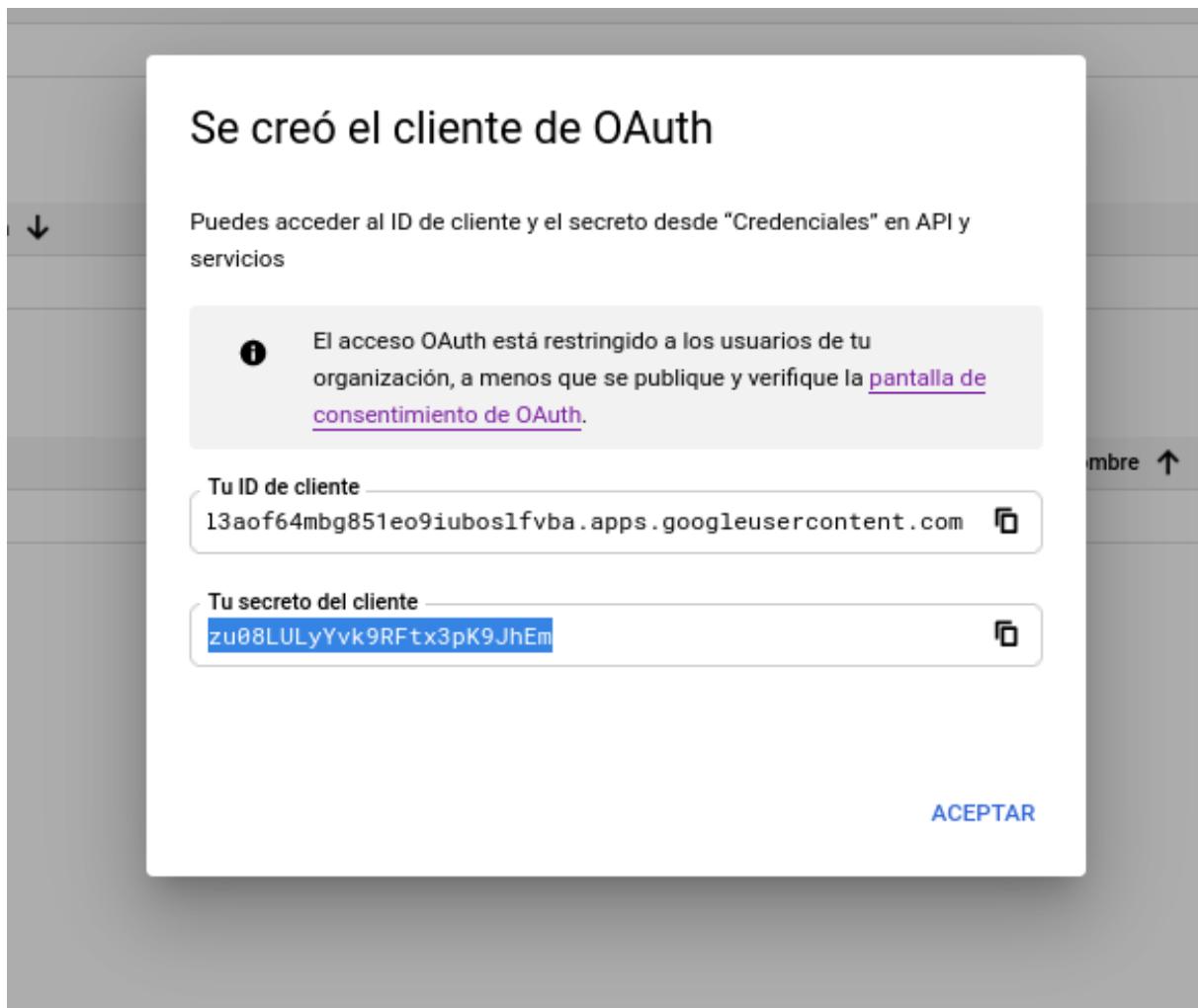


Figura 8: Client Secret

Instala Composer desde la página de [instalación](#).

Ahora ya puedes descargarte este proyecto de [GitHub](#). Si todo ha ido bien, ejecuta composer install. De esta manera ya tienes instalado el paquete [Google API Client](#). Además debes renombrar el archivo client_secrets-sample.json por client_secrets.json e introducir tus credenciales.

ATENCIÓN.

Nunca subas un fichero a GitHub con las credenciales reales!

Ya solo nos resta **habilitar el API de Drive** desde este enlace <https://console.developers.google.com/apis/api/drive.googleapis.com>

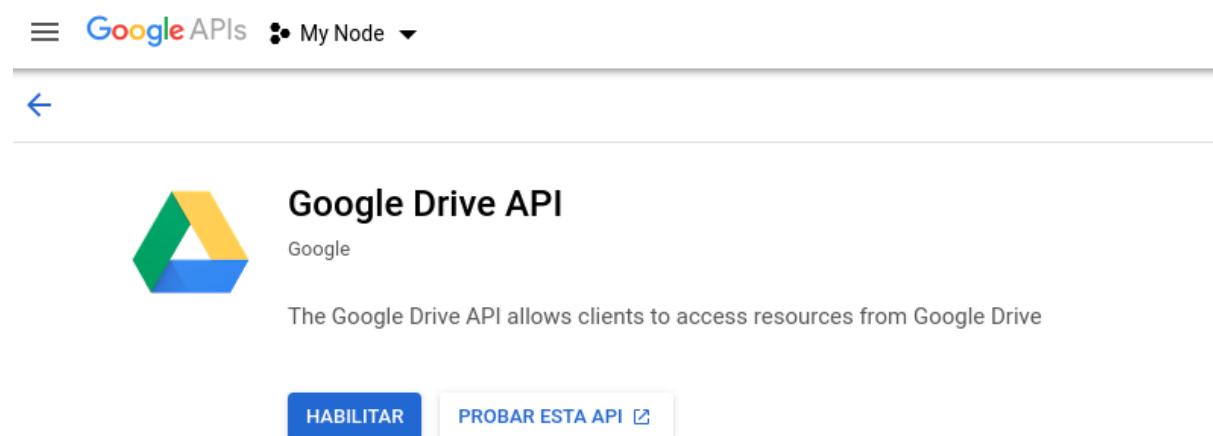


Figura 9: Google Drive API

En el directorio donde has descargado el repositorio ejecuta

```
php -S localhost:8080
```

Y te pedirá tus credenciales de Google para continuar.

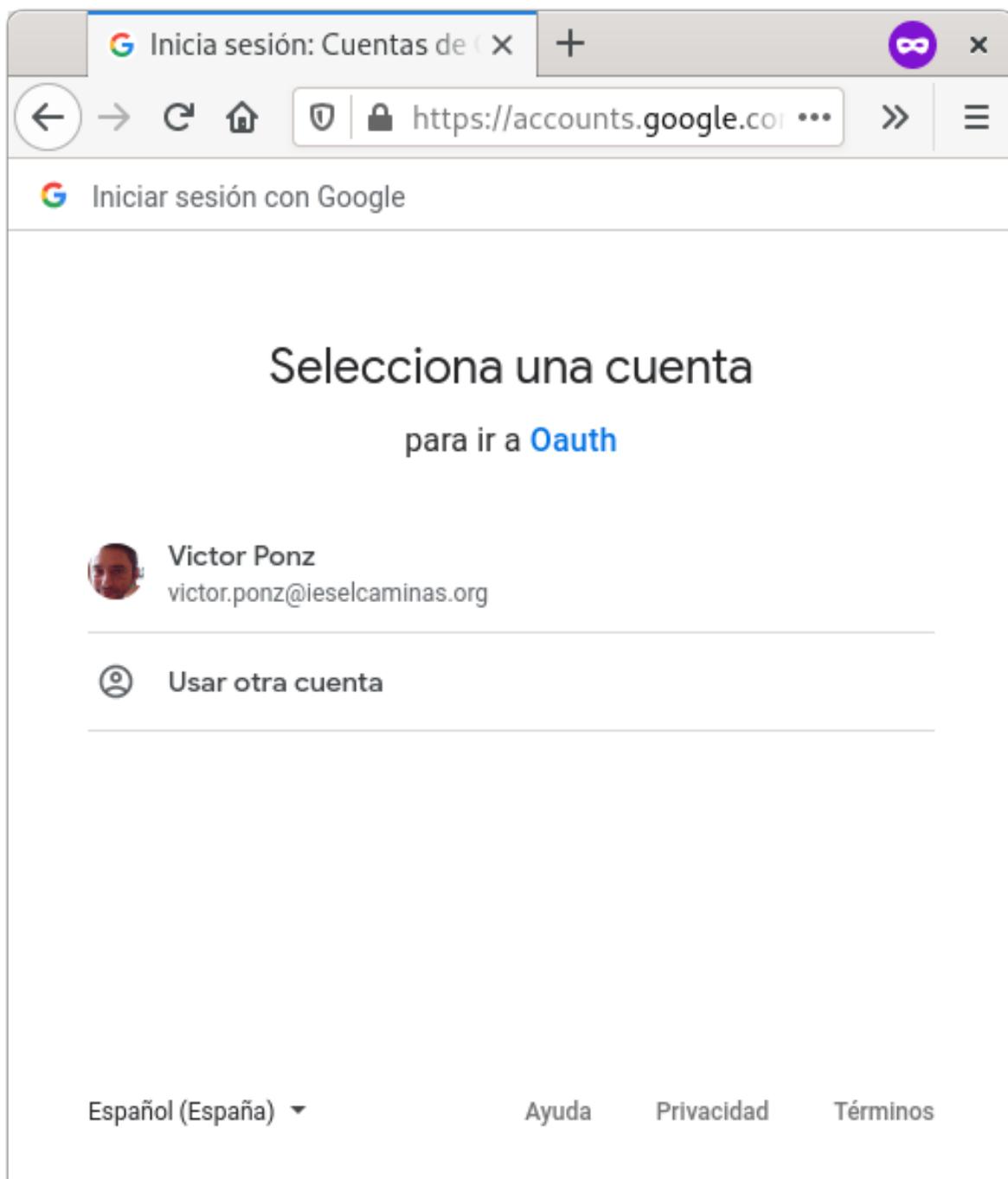


Figura 10: Credenciales

Si todo ha ido bien, verás un listado con todos tus documentos de Drive

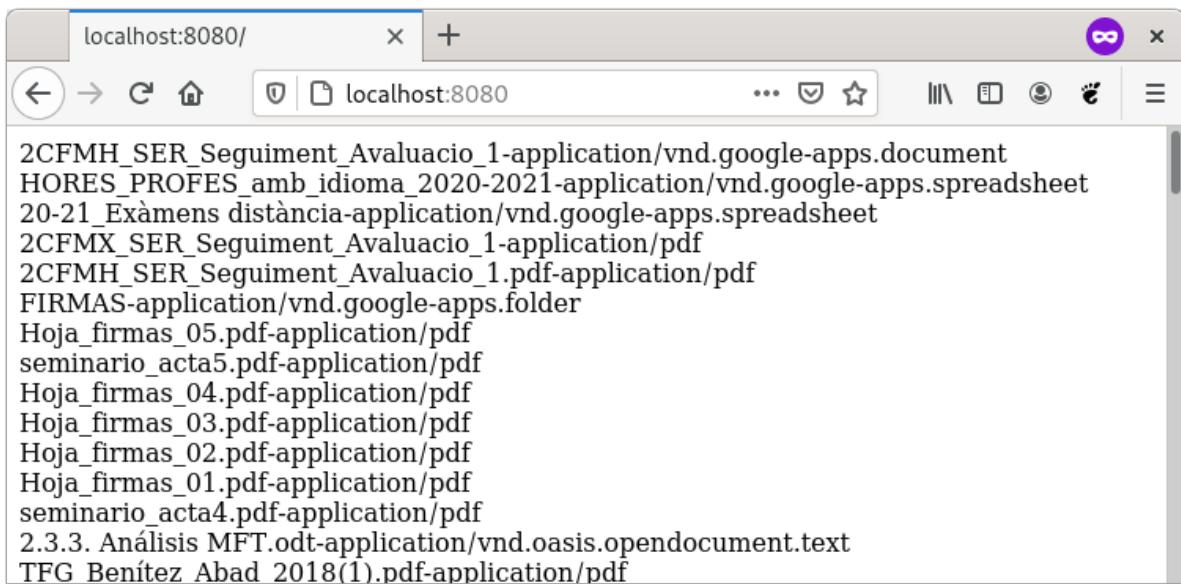


Figura 11: Documentos de Drive

Mas información en:

<https://aaronparecki.com/oauth-2-simplified/>

https://jordisan.net/proyectos/Autent_y_auth-J_Sanchez.pdf

<https://auth0.com/learn/how-auth0-uses-identity-industry-standards/>

<https://platzi.com/blog/aprende-estandares-de-seguridad-y-oauth/>

1.2 Single Sign-On (SSO)

Según la [Wikipedia](#)

El «Inicio de Sesión Único» o «Inicio de Sesión Unificado» (**Single Sign-On, SSO**) es un procedimiento de autenticación que habilita a un [usuario](#) determinado para acceder a varios sistemas con una sola instancia de identificación. Su traducción literal es «autenticación única» o «validación única».

Un ejemplo real es el uso de los distintos servicios de Google ya que una vez hemos iniciado sesión en uno de ellos, estamos logeados en todos los demás servicio (Gmail, Maps, Docs, Drive, ...).

Ventajas	Desventajas
Acelera el acceso de los usuarios a sus aplicaciones	Utilizar una única combinación aumenta las probabilidades de vulnerabilidad de contraseñas
Reduce la carga de memorizar diversas contraseñas	Al fallar SSO se pierde acceso a todos los sistemas relacionados
Fácil de implementar y conectar a nuevas fuentes de datos	Suplantación de identidades en los accesos externos de los usuarios

Fuente: <https://www.chakray.com/es/que-es-el-single-sign-on-sso-definicion-caracteristicas-y-ventajas/>

1.3 Contraseñas de un sólo uso (One-time password)

Según la [Wikipedia](#)

Una **contraseña de un solo uso** u **OTP** (del inglés *One-Time Password*) es una [contraseña](#) válida solo para una [autenticación](#). La OTP soluciona una serie de deficiencias que se asocian con la tradicional (estática) [contraseña](#). La deficiencia más importante que se aborda en las OTP es que, en contraste con contraseñas estáticas, no son vulnerables a [ataques de REPLAY](#). También hace al sistema más resistente frente ataques de [fuerza bruta](#), ya que cada vez que cambia la contraseña, los intentos realizados anteriormente para romper la contraseña anterior son inútiles y hay que empezar de nuevo. Esto significa que un posible intruso que logre registrar una OTP que ya se utiliza para iniciar sesión en un servicio o realizar una transacción no será capaz de abusar de ella, puesto que ya no será válida. En el lado negativo, las OTPs son difíciles de utilizar en los seres humanos, debido a que un ser humano no puede memorizar todas. Por lo tanto, se requiere de una tecnología adicional para funcionar.

Una de las implementaciones más comunes es mediante Google Authenticator que asigna un código de 6 dígitos que el usuario debe proporciona además de su usuario y contraseña. Estos códigos One-time son generados haciendo uso de standards abiertos desarrollados por [The Initiative for Open Authentication \(OATH\)](#) (no guarda relación con OAuth).

Mediante correo electrónico

En este método el inicio de sesión sin contraseña está basado en correo electrónico. Cuando nos registramos recibimos un correo con un enlace que incluye la contraseña generada aleatoriamente

por el servidor. Este enlace se invalida cuando iniciamos sesión y también se fija una ventana temporal para su uso. De esta forma es complicado suplantar al usuario ya que los ataques por fuerza bruta sólo serían válidos durante dicha ventana temporal.

1.4 FIDO (Fast IDentity Online)

La autenticación FIDO (Fast IDentity Online) es un conjunto de estándares para una autenticación rápida, simple y sólida.

Estos estándares son desarrollados por FIDO Alliance, una asociación de la industria con representantes de una variedad de organizaciones, incluidas Google, Microsoft, Mozilla y Yubico. Los estándares permiten la autenticación multifactor, sin contraseña y resistente al phishing. Mejoran la experiencia de usuario en línea al hacer que la autenticación sólida sea más fácil de implementar y usar.

Algunas de las herramientas y aplicaciones más populares de la web ya utilizan la autenticación FIDO, incluidas las cuentas de Google, Dropbox, GitHub, Twitter y Yahoo Japón.

- Los usuarios ganan. Los usuarios se benefician de los flujos de autenticación que son rápidos y seguros.
- Los desarrolladores ganan. Los desarrolladores de aplicaciones y web pueden utilizar API simples para autenticar a los usuarios de forma segura.
- Las empresas ganan. Los propietarios de sitios y proveedores de servicios pueden proteger a sus usuarios de manera más eficaz.

¿Cómo funciona la autenticación FIDO?

Cuando un usuario se registra en un servicio «online» que emplea el estándar FIDO, el sistema genera una pareja de **claves criptográficas**, de forma que **la clave privada se conserva en el «hardware» del dispositivo y la clave pública se guarda en el servicio «online»**. Para realizar la autenticación, el dispositivo del cliente debe demostrar al servicio «online» que dispone de la clave privada realizando una verificación matemática. Además, la clave privada del cliente únicamente se podrá utilizar una vez que el usuario la haya desbloqueado de forma local en el dispositivo. Este desbloqueo puede realizarse mediante una acción segura y fácil como por ejemplo introduciendo su huella dactilar, utilizando su voz o introduciendo un PIN.

De esta forma se consigue **proteger la privacidad del usuario y sus credenciales de acceso**, consiguiendo que los usuarios no se vean obligados a elegir entre mejor seguridad o mejor experiencia de usuario, sino que pueden disponer de ambas.

En un flujo de autenticación FIDO, una parte que confía usa API para interactuar con el autenticador de un usuario.

- Relying party La parte que confía en su servicio, compuesto por un servidor back-end y una aplicación front-end.
- Aplicación

Durante un flujo de autenticación o registro, la aplicación utiliza API del lado del cliente como WebAuthn y FIDO2 para Android para crear y verificar las credenciales de usuario con el autenticador.

Esto implica pasar un desafío criptográfico del servidor al autenticador y devolver la respuesta del autenticador al servidor para su validación

- Servidor El servidor almacena la información de la cuenta y la credencial de clave pública del usuario.

Durante un flujo de autenticación o registro, el servidor genera un desafío criptográfico en respuesta a una solicitud de la aplicación. Luego evalúa la respuesta al desafío.

La Alianza FIDO mantiene una lista de productos de terceros certificados, incluidas las soluciones de servidor. También están disponibles varios servidores FIDO de código abierto; consulte WebAuthn Awesome para obtener más información.

Autenticador

Un autenticador FIDO genera credenciales de usuario. Una credencial de usuario tiene un componente de clave pública y privada. La clave pública se comparte con su servicio, mientras que el autenticador mantiene en secreto la clave privada.

Un autenticador puede ser parte del dispositivo del usuario o una pieza externa de hardware o software.

El autenticador se utiliza en dos interacciones básicas: registro y autenticación.

Registro

En un escenario de registro, cuando un usuario se registra para obtener una cuenta en un sitio web, el autenticador genera un nuevo par de claves que solo se puede utilizar en su servicio. La clave pública y un identificador de la credencial se almacenarán con el servidor.

Autenticación

En un escenario de autenticación, cuando un usuario regresa al servicio en un nuevo dispositivo, o después de que expira su sesión, el autenticador debe proporcionar prueba de la clave privada del usuario. Lo hace respondiendo a un desafío criptográfico emitido por el servidor.

Para verificar la identidad del usuario, algunos tipos de autenticadores utilizan datos biométricos como huellas dactilares o reconocimiento facial. Otros usan un PIN. En algunos casos, se utiliza una

contraseña para verificar al usuario y el autenticador solo proporciona autenticación de segundo factor.

Más información:

<https://developers.google.com/identity/fido>

<https://codelabs.developers.google.com/codelabs/webauthn-reauth/#0>

<https://www.bbva.com/es/que-es-fido-el-nuevo-estandar-de-autenticacion-online/>

1.5 JSON web tokens

Según la [Wikipedia](#)

JSON Web Token (abreviado **JWT**) es un [estándar abierto](#) basado en [JSON](#) propuesto por [IETF \(RFC 7519\)](#) para la creación de [tokens de acceso](#) que permiten la propagación de identidad y privilegios o *claims* en inglés. Por ejemplo, un servidor podría generar un token indicando que el usuario tiene privilegios de administrador y proporcionarlo a un cliente. El cliente entonces podría utilizar el token para probar que está actuando como un administrador en el cliente o en otro sistema. El token está firmado por la clave del servidor, así que el cliente y el servidor son ambos capaz de verificar que el token es legítimo. Los JSON Web Tokens están diseñados para ser compactos, poder ser enviados en las URLs -*URL-safe*- y ser utilizados en escenarios de [Single Sign-On \(SSO\)](#). Los privilegios de los JSON Web Tokens puede ser utilizados para propagar la identidad de usuarios como parte del proceso de [autenticación](#) entre un [proveedor de identidad](#) y un [proveedor de servicio](#), o cualquiera otro tipo de privilegios requeridos por procesos empresariales

La autorización se logra cuando el usuario ingresa sus credenciales con éxito, entonces se genera un JSON Web Token que es retornado al cliente, quien tiene que guardarlo localmente, en vez del modelo tradicional de crear una sesión en el servidor y retornar una [cookie](#).

Siempre que el usuario quiere acceder a una ruta protegida o recurso, el cliente tiene que enviar el JWT, generalmente en el encabezado de Authorization utilizando el esquema Bearer. El contenido del encabezado HTTP se ve de la siguiente forma:

Authorization: Bearer eyJhbGci...<snip>...yu5CSpyHI

Este es un mecanismo de autenticación sin estado - *stateless*- ya que la sesión del usuario nunca se guarda en el proveedor de identidad o en el proveedor del servicio. Los recursos protegidos siempre comprobaran si existe un JWT válido en cada pedido de acceso. Si el token esta presente y es válido,

el proveedor del servicio otorga accesos a los recursos protegidos. Como los JWTs contienen toda la información necesaria en sí mismos, se reduce la necesidad de consultar la base de datos u otras fuentes de información múltiples veces.

A continuación se muestra un ejemplo de token, que está formado por tres partes: un encabezado o *header*, un contenido o *payload*, y una firma o *signature*.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4

que decodificado se convierte el siguiente json:

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYLOAD: DATA

```
"sub": "1234567890",  
"name": "John Doe",  
"iat": 1516239022  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

The screenshot shows a web-based tool for decoding JWTs. It has three main sections: 'HEADER: ALGORITHM & TOKEN TYPE' containing the JSON { "alg": "HS256", "typ": "JWT" }; 'PAYLOAD: DATA' containing { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }; and 'VERIFY SIGNATURE' containing the HMACSHA256 formula with a placeholder for the secret key. A checkbox for 'secret base64 encoded' is present but unchecked. The entire interface is set against a light gray background with a large white triangle on the right side.

Figura 12: JSON Web Token

IMPLEMENTACIÓN EN PHP

<https://coderwall.com/p/8wrxfw/goodbye-php-sessions-hello-json-web-tokens>

1.6 Autenticación en dos factores

La Autenticación de Dos Factores es una herramienta que ofrecen varios proveedores de servicios en línea, que cumple la función de agregar una capa de seguridad adicional al proceso de inicio de sesión de tus cuentas de Internet. La mecánica es simple: cuando el usuario inicia sesión en su cuenta personal de algún servicio online, esta herramienta le solicita que autentifique la titularidad de su cuenta, proporcionando dos factores distintos. El primero de estos, es la contraseña. El segundo, puede ser varias cosas, siempre dependiendo del servicio. En el más común de los casos, suele tratarse de un código que se envía a un teléfono móvil vía SMS o a una cuenta de email. La esencia fundamental de esta herramienta se reduce a que, si quieras logearte a una de tus cuentas personales, debes «saber algo» y «poseer algo». Así, por ejemplo, para acceder a la red virtual privada de una compañía, es posible que necesites de una clave y de una memoria USB. La autenticación de dos factores esencialmente logra que un atacante tenga que no sólo descifrar la contraseña de los usuarios, sino también acceder a un segundo factor, mucho más difícil de conseguir, y que implicaría robar un teléfono celular o comprometer un correo electrónico.

1.7 Cómo almacenar de forma segura la contraseña

Como la mayoría de los usuarios reutilizarán sus contraseñas entre diferentes aplicaciones, es importante almacenar las contraseñas de una manera que impida que sean obtenidas por un atacante, incluso si la aplicación o la base de datos están comprometidas. Como en la mayoría de las áreas de la criptografía, hay muchos factores diferentes que deben ser considerados, pero afortunadamente, la mayoría de los lenguajes y frameworks modernos proporcionan una funcionalidad incorporada para ayudar a almacenar las contraseñas, que maneja gran parte de la complejidad.

Debido a que las tablas son vulnerables al robo, el almacenamiento de la contraseña en texto llano es peligroso. Por lo tanto, la mayoría de las bases de datos almacenan un hash criptográfico de la contraseña del usuario en la base de datos. En un entorno así, nadie, incluyendo la propia autenticación de sistema puede determinar cual es la contraseña del usuario simplemente observando el valor almacenado en la base de datos. En cambio, cuando un usuario introduce su contraseña para autenticarse, se calcula el hash de la contraseña introducida y se compara con el valor almacenado para ese usuario (que fue hash antes de ser almacenado). Si los dos valores hash coinciden, se concede el acceso.

Esta hoja de trucos proporciona orientación sobre las diversas áreas que deben ser consideradas en relación con el almacenamiento de contraseñas. En resumen:

- **Utilizar Bcrypt a menos que tenga una buena razón para no hacerlo.**
- **Establecer un factor de trabajo razonable para su sistema.**
- **Utilizar un salt (los algoritmos modernos lo hacen automáticamente).**
- **Considerar el uso de una pimienta para proporcionar una defensa adicional en profundidad (aunque por sí sola no proporciona ninguna característica de seguridad adicional).**

1.7.1 Hashing vs. Encriptación

Hashing y encriptación son dos términos que a menudo se confunden o se utilizan incorrectamente. La diferencia clave entre ellos es que el hashing es una función **unidireccional** (es decir, no es posible «desencriptar» un hash y obtener el valor original), mientras que el cifrado es una función bidireccional.

En casi todas las circunstancias, las contraseñas deberían ser sometidas a un hash en lugar de ser encriptadas, ya que esto dificulta o imposibilita que un atacante obtenga las contraseñas originales a partir de los hashes.

El cifrado sólo debería utilizarse en casos extremos en los que sea necesario poder obtener la contraseña original. Algunos ejemplos de los casos en los que esto puede ser necesario son:

- Si la aplicación necesita usar la contraseña para autenticarse contra un sistema externo heredado que no soporta SSO.
- Si es necesario recuperar caracteres individuales de la contraseña.

La posibilidad de descifrar las contraseñas representa un grave riesgo para la seguridad, por lo que debería evaluarse completamente el riesgo. Siempre que sea posible, se debe utilizar una arquitectura alternativa para evitar la necesidad de almacenar las contraseñas de forma cifrada.

Esta hoja de trucos se centra en el hashing de contraseñas; para más información sobre el cifrado de contraseñas, consulta la [hoja de trucos sobre almacenamiento criptográfico](#).

1.7.2 Cómo los atacantes descifran los hash de las contraseñas

Aunque no es posible «descifrar» los hashes de contraseñas para obtener las contraseñas originales, en algunas circunstancias es posible «crackear» los hashes. Los pasos básicos son:

- Seleccionar un candidato probable (como «contraseña»).

- Calcular el hash de la entrada.
- Compararlo con el hash objetivo.

Este proceso se repite para un gran número de posibles contraseñas candidatas hasta encontrar una coincidencia. Hay un gran número de métodos diferentes que se pueden utilizar para seleccionar las contraseñas candidatas, incluyendo:

- Fuerza bruta (probar todas las candidatas posibles).
- Diccionarios o listas de palabras de contraseñas comunes
- Listas de contraseñas obtenidas de otros sitios comprometidos.
- Algoritmos más sofisticados, como las [cadenas de Markov](#) o [PRINCE](#)
- Patrones o máscaras (como «1 mayúscula, 6 minúsculas, 1 número»).

El proceso de descifrado no tiene garantía de éxito, y el porcentaje de éxito dependerá de varios factores:

- La fuerza de la contraseña.
- La velocidad del algoritmo (o factor de trabajo para los algoritmos modernos).
- El número de contraseñas a las que se apunta (suponiendo que tengan sales únicas).

Las contraseñas fuertes almacenadas con algoritmos hash modernos deberían ser efectivamente imposibles de descifrar para un atacante.

1.7.3 Contraseñas de texto sin formato

Supongamos que un atacante logra recuperar una base de datos de una aplicación web y extrae de ella la pareja <login, Password>

Login	Password
admin	azerty
toto	matrix
billy	yep59f\$4txwrr
tata	matrix
titi	freepass

En este caso, el atacante posee directamente las contraseñas de todos los usuarios en texto sin formato. Incluso Billy, que tiene una contraseña segura, no está protegido.

Almacenar contraseñas en texto plano NO es una solución segura. Nadie, incluidos los administradores de sitios web / bases de datos, debe tener acceso a la contraseña de texto sin formato del usuario.

1.7.4 Contraseñas encriptadas

En algunos casos, las contraseñas se almacenan en una base de datos después de haber sido encriptadas por un algoritmo reversible (rot13, mask encryption...). Como el algoritmo es reversible, no cumple con las reglas del ENS.

De hecho, recomienda que cualquier contraseña sea transformada por una función criptográfica no reversible.

Dado que el atacante conoce su contraseña en formato de texto sin formato / cifrado, puede adivinar la lógica del cifrado e intentar revertirla. Si tiene éxito, todas las contraseñas se recuperarán tan rápido como estaban en texto sin formato, independientemente de la complejidad del algoritmo.

1.7.5 Funciones hash obsoletas

En muchos casos, las contraseñas se almacenan con funciones criptográficas irreversibles desactualizadas (md5, sha1...). Por ejemplo, el sitio de LinkedIn solía almacenar parte de sus contraseñas con sha1, y después de las filtraciones de hash en 2012, solo se tardó tres días recuperar el 90% de las contraseñas.

Tomemos la siguiente base de datos (las contraseñas son las mismas que antes)

Login	Password
admin	ab4f63f9ac65152575886860dde480a1
toto	21b72c0b7adc5c7b4a50ffcb90d92dd6
billy	47ad898a379c3dad10b4812eba843601
tata	21b72c0b7adc5c7b4a50ffcb90d92dd6
titi	5b9a8069d33fe9812dc8310ebff0a315

A destacar:

- En nuestro caso, todas las contraseñas (excepto la de Billy) son contraseñas de uso muy frecuente y se encuentran entre las contraseñas más utilizadas (por ejemplo, en la lista de 10 millones de contraseñas top-1000.txt)

- También es interesante hacer notar que dado que los hash no tienen una noción de aleatoriedad, todo y tata comparten el mismo hash, ya que tienen la misma contraseña.

Una simple búsqueda del hash del administrador en Internet permite recuperar directamente sus contraseñas.

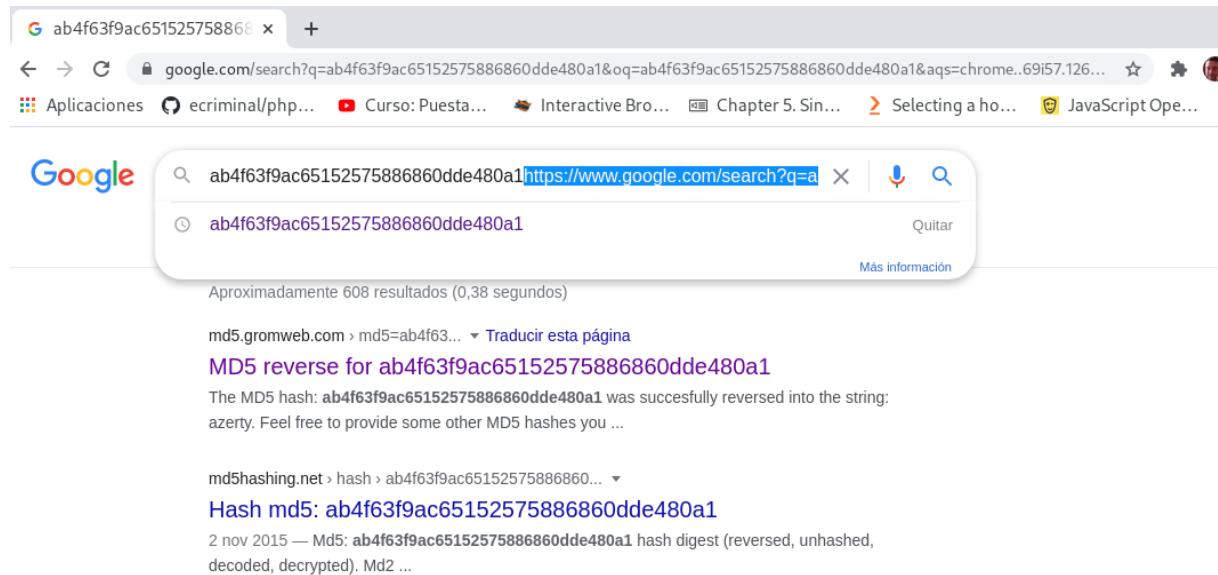


Figura 13: Búsqueda hash

A excepción del usuario **billy**, que tiene una contraseña compleja, es posible recuperar todos los hashes directamente mediante consultas en un motor de búsqueda.

Si los hash no se encuentran directamente en un motor de búsqueda, el atacante tiene otros métodos:

Fuerza bruta

La fuerza bruta es la acción de probar todas las posibilidades siguiendo iterativamente una regla de generación. Es como cuando intentamos abrir un candado enumerando todas las posibilidades desde 0000 hasta 9999 hasta que se abre el candado.

Diccionario

Un ataque de diccionario es un ataque en el que se prueban todos los términos de una lista de palabras. Se pueden imaginar varios tipos de diccionarios:

- Un diccionario de idiomas
- Una clasificación de las contraseñas más utilizadas
- Una lista adaptada a un contexto dado

Si tomamos la imagen del candado, podemos imaginar una lista contextualizada como esta:

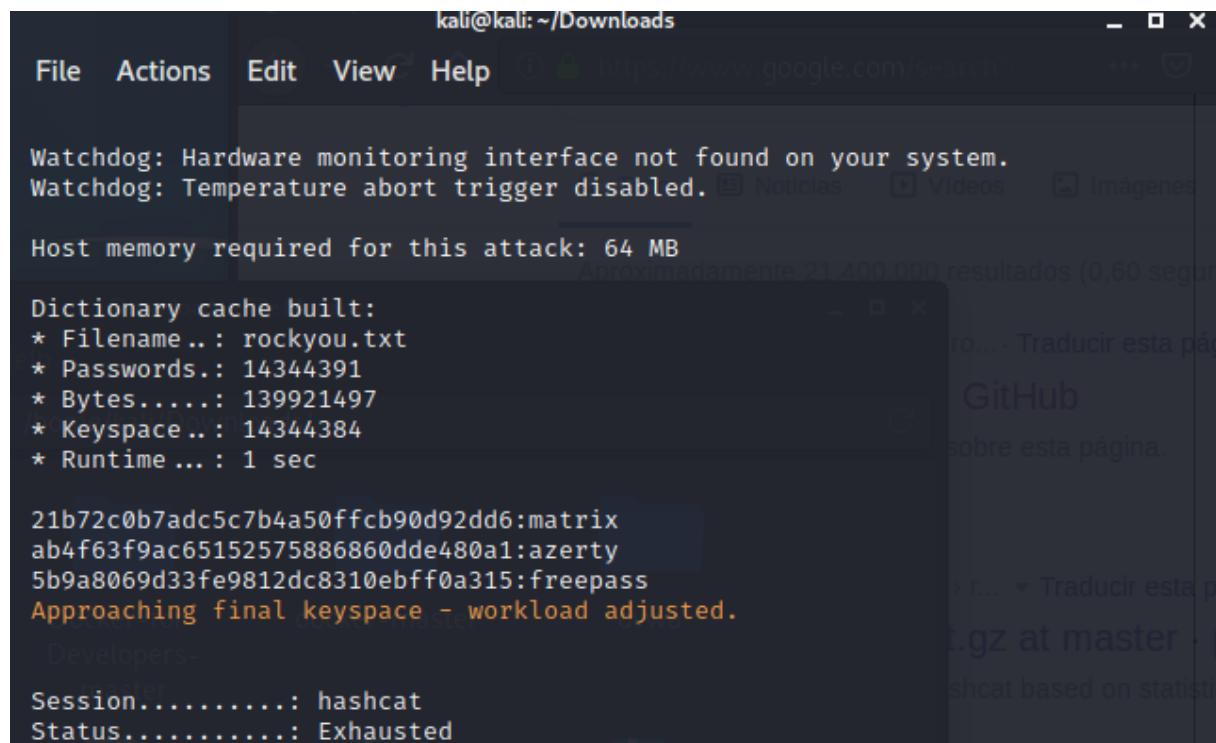
Sabemos que el candado le pertenece a «tutu», que le encanta el número 42, y que nació el 26 de noviembre de 2001. Entonces podemos suponer que el candado posiblemente puede contener el número 42, 26, 11 y 01 y así genere una lista de acuerdo con estos criterios.

Tabla rainbow

Las tablas de arcoiris son un tema que merece un artículo por sí solo. Rápidamente, es una estructura de datos que permite recuperar contraseñas con un buen compromiso de almacenamiento / tiempo. Esta estructura tiene una lista de hashes precalculados y hace posible recuperar un hash en un tiempo aceptable. Muchas tablas de arcoiris están disponibles en línea.

1.7.6 Benchmark para recuperar contraseñas Md5

```
hashcat -m0 password.txt rockyou.txt
```



```
kali㉿kali:~/Downloads
File Actions Edit View Help https://www.google.com/search?...  ... 📧
Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled. 📰 Noticias 🎥 Videos 🖼 Imágenes
Host memory required for this attack: 64 MB
Dictionary cache built:
* Filename..: rockyou.txt
* Passwords.: 14344391
* Bytes.....: 139921497
* Keyspace ..: 14344384
* Runtime ...: 1 sec
21b72c0b7adc5c7b4a50ffcb90d92dd6:matrix
ab4f63f9ac65152575886860dde480a1:azerty
5b9a8069d33fe9812dc8310ebff0a315:freepass
Approaching final keyspace - workload adjusted.
Developers...
Session.....: hashcat
Status.....: Exhausted
```

Figura 14: Benchmark md5

Se realizó una evaluación comparativa en la base de datos con la lista rockyou que contiene 14.341.564 contraseñas únicas. La prueba se realizó en una máquina virtual, que no es óptima para

romper contraseñas. Al observar los resultados, vemos que a excepción de la contraseña de Billy, que no está en la lista de rockyou, se han encontrado las tres contraseñas y solo fue necesario 1 segundo para que el software calcule todos los hashes presentes en rockyou.

1.7.7 Función hash inapropiada

Después de ver los malos ejemplos anteriores, es tentador utilizar funciones irreversibles seguras como sha256, sha512 o sha3. Sin embargo, el propósito de estas funciones es que se utilicen para calcular un resumen criptográfico para verificar la integridad de un archivo, realizar una firma electrónica u optimizar la búsqueda y la indexación. No son adecuados para almacenar contraseñas, porque son rápidos de calcular, como el siguiente punto de referencia prueba:

Login	Password
admin	df6b9fb15cfdbb7527be5a8a6e39f39e572c8ddb943fbc79a943438e9d3d85ebfc2ccf9e0eccd9346026c0b68
toto	11a25e88658143a853d280bf77f81ff391347aaba2db54a3aab0149b265276de419880762a473fc496388bcf7
billy	fe9cb9b07725fd1cc3906232405119fedf9a020436630d3c1e0f632f73909e6ed9e731c149ac22743bbe954188
tata	11a25e88658143a853d280bf77f81ff391347aaba2db54a3aab0149b265276de419880762a473fc496388bcf7
titi	f767036acd951f5ddaf4eed5291c677db060055806dbcce69ca35d95847559dc8abce5011fd2b50833e760ec

```
hashcat -m1700 password-sha512.txt rockyou.txt
```

```

File Actions Edit View Help
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace..: 14344384
11a25e88658143a853d280bf77f81ff391347aaba2db54a3aab0149b265276de419880762a473fc496388bcf70566
d7cf0346c34add40652f8f7b669caf9ec0:matrix
df6b9fb15cfdbb7527be5a8a6e39f39e572c8ddb943fbc79a943438e9d3d85ebfc2ccf9e0eccd9346026c0b6876e0
e01556fe56f135582c05fdbbb505d46755a:azerty
f767036acd951f5ddaf4eed5291c677db060055806dbcae69ca35d95847559dc8abce5011fd2b50833e760eca2d84
d6daf1f078200f42b4fc10b58bad3761c88:freepass
Approaching final keyspace - workload adjusted.

Session.....: hashcat
Status.....: Exhausted
Hash.Name....: SHA2-512
Hash.Target....: password-sha256.txt
Time.Started....: Sat Feb 20 12:57:04 2021 (10 secs)
Time.Estimated ...: Sat Feb 20 12:57:14 2021 (0 secs)
Guess.Base.....: File (rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 1418.3 KH/s (0.71ms) @ Accel:1024 Loops:1 Thr:1 Vec:4
Recovered.....: 3/4 (75.00%) Digests
Progress.....: 14344384/14344384 (100.00%)
Rejected.....: 0/14344384 (0.00%)
Restore.Point....: 14344384/14344384 (100.00%)
Restore.Sub.#1 ...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: $HEX[206b6d3831303838] → $HEX[042a0337c2a156616d6f732103]

Started: Sat Feb 20 12:57:03 2021
Stopped: Sat Feb 20 12:57:16 2021

```

Figura 15: Benchmark SHA2-512

Aquí nuevamente, a excepción del usuario Billy, se han recuperado todas las contraseñas y solo fueron necesarios 13 segundos para que hashcat finalizara sus operaciones (aunque los tres primeros los recupera en menos de un segundo)

1.7.8 Mejorando SHA512

Incluso si se ha dicho anteriormente que la función SHA512 no estaba optimizada para el almacenamiento de contraseñas, puede ser interesante mostrar cómo optimizarla para comprender el interés de las funciones hash apropiadas para las contraseñas.

1.7.9 Salting

Una sal es una cadena única, generada aleatoriamente, que se añade a cada contraseña como parte del proceso de hashing. Como la sal es única para cada usuario, un atacante tiene que descifrar los hashes de uno en uno usando la sal respectiva, en lugar de poder calcular un hash una vez y

compararlo con cada hash almacenado. Esto hace que descifrar un gran número de hashes sea mucho más difícil, ya que el tiempo requerido crece en proporción directa al número de hashes.

El salado también proporciona protección contra un atacante que precalcule los hashes utilizando tablas rainbow o búsquedas basadas en bases de datos. Por último, el salting significa que no es posible determinar si dos usuarios tienen la misma contraseña sin descifrar los hashes, ya que las diferentes sales darán lugar a diferentes hashes aunque las contraseñas sean las mismas.

Mediante software para descifrar contraseñas (hashcat, Johntheripper...), después de romper un hash, se puede buscar si está presente para otro usuario. Por lo tanto, sin sal, después de descubrir la contraseña de `toto`, la contraseña de `tata` se descubre directamente. Sin embargo, con una sal, el software debe comenzar de nuevo desde cero para cada usuario.

Los algoritmos de hashing modernos, como Argon2 o Bcrypt, salan automáticamente las contraseñas, por lo que no es necesario realizar ningún paso adicional al utilizarlos. Sin embargo, si se utiliza un algoritmo de hash de contraseñas heredado, el salado debe implementarse manualmente. Los pasos básicos para realizarlo son

- Genera una sal utilizando una función criptográficamente segura.- La sal debe tener al menos 16 caracteres.
- Codificar la sal en un conjunto de caracteres seguro como el hexadecimal o Base64.
- Combinar la sal con la contraseña.- Esto puede hacerse utilizando una simple concatenación, o una construcción como HMAC.
- Hacer un hash de la contraseña y la sal combinadas.
- Almacenar la sal y el hash de la contraseña.

La sal es un contaminante para los datos brutos (aquí la contraseña) permitiendo producir dos hashes diferentes a partir de los mismos datos. La sal es única para cada usuario y se compone de una secuencia aleatoria. Aumenta la posibilidad de que una contraseña sea única y, por tanto, la posibilidad de que nunca se haya utilizado un hash. Por ejemplo, con salt, `toto` y `tata` no tendrán el mismo hash en la base de datos.

Las ventajas de la sal son múltiples:

1. Es casi imposible encontrar el hash directamente en Internet si está salado. Sin embargo, la sal debe ser lo suficientemente larga y aleatoria.
2. Las tablas Rainbow no funcionan con hashes con salt.
3. Como se dijo antes, dos usuarios con la misma contraseña no tendrán el mismo hash si se usa salt. Software para descifrar contraseñas (hashcat, Johntheripper...), después de romper un hash, busca si no está presente para otro usuario. Por lo tanto, sin sal, después de descubrir la

contraseña de toto, la contraseña de tata se descubre directamente. Sin embargo, con una sal, el software debe comenzar de nuevo desde cero para cada usuario.

Login	Salt	Hash sha512 with salt
admin	BGdd6d6^ZgvkMhKf@W3R5J9d123bce1aa92331861cf8fd738a58205045123f0e25f0862477cb19d3ee0757cd	
toto	HZBD^@gL*wvoExo6yJ7h0B28830776de6ad7ef1dd8c221e0d53fec4532c623075d0216d937ab82ab284a56a	
billy	wvVndjwcZJy!dwT4fBD@Q847b2605f6a1cd88399e6c9784c0e583799be9485cb128fe5f541f43636559067ec3	
tata	QeNWm9NXqJ8m@m2^F1K5906b69fa2bfcd893bfd86358394406c87c7f7abba891cd10ed9fac887c54d52e	
titi	iQUemgw9M6Gw*&v6RG%847#d6c815c7522ab6197aa319d3ff4cddc2c7effa0f91c1291603f807a47f320324d	

Con esta configuración, el software hashcat tardó 33 segundos en recuperar las contraseñas. Ninguno de los hash de la base de datos está indexado por un motor de búsqueda.

1.7.10 Pimentar

Se puede utilizar una [pimienta](#) además del salado para proporcionar una capa adicional de protección. Es similar a la sal, pero tiene cuatro diferencias clave:

- La pimienta se **comparte entre todas las contraseñas almacenadas**, en lugar de ser **única** como la sal. Esto hace que la pimienta sea predecible y que los intentos de descifrar el hash de una contraseña *sean probabilísticos*. La naturaleza estática de una pimienta también «debilita» la resistencia a las colisiones de hash, mientras que la sal mejora la resistencia a las colisiones de hash al ampliar la longitud con caracteres únicos que aumentan la entropía de la entrada a la función de hash.
- La pimienta **no se almacena en la base de datos**, a diferencia de muchas implementaciones de una sal de contraseña (pero no siempre es cierto para una sal).
- La pimienta no es un mecanismo para hacer que el descifrado de contraseñas **sea demasiado difícil** para un atacante, como pretenden hacer muchas protecciones de almacenamiento de contraseñas (entre ellas la sal).
- Una sal evita que los atacantes compilen tablas de arco iris de contraseñas conocidas, sin embargo una pimienta no ofrece esta característica

El propósito de la pimienta es evitar que un atacante pueda descifrar cualquiera de los hashes si sólo tiene acceso a la base de datos, por ejemplo si ha explotado una vulnerabilidad de inyección SQL u obtenido una copia de seguridad de la base de datos.

La pimienta debe tener *al menos* 32 caracteres y debe generarse aleatoriamente utilizando un generador pseudoaleatorio seguro (CSPRNG). Debe almacenarse de forma segura en una «bóveda de secretos» (no en un archivo de configuración de la aplicación, independientemente de los permisos de los archivos que son susceptibles de SSRF) utilizando las API de acceso seguro, o para un almacenamiento seguro óptimo almacenar la pimienta en un Módulo de Seguridad de Hardware (HSM) si es posible.

La pimienta se utiliza a menudo de forma similar a la sal, concatenándola con la contraseña antes del hash, utilizando una construcción como `hash($pepper . $password)`. Mientras que la concatenación se considera apropiada para una sal, sólo el prefijo se considera apropiado para una pimienta.

Nunca coloque una pimienta como sufijo, ya que esto puede conducir a vulnerabilidades tales como problemas relacionados con el truncamiento y los ataques de extensión de longitud. Prácticamente estas amenazas permiten que el componente de la contraseña de entrada se valide con éxito porque la contraseña única nunca se trunca, sólo la pimienta probabilística se truncaría.

Incrementar el número de iteraciones o work factor

Otra forma de aumentar la seguridad es repetir el número de iteraciones de hash. Aumentar el número de iteraciones significa que vamos a codificar la contraseña varias veces.

A la hora de elegir un factor de trabajo, hay que encontrar un equilibrio entre seguridad y rendimiento. Los factores de trabajo más altos harán que los hashes sean más difíciles de descifrar para un atacante, pero también harán que el proceso de verificación de un intento de inicio de sesión sea más lento. Si el factor de trabajo es demasiado alto, esto puede degradar el rendimiento de la aplicación, y también podría ser utilizado por un atacante para llevar a cabo un ataque de denegación de servicio haciendo un gran número de intentos de inicio de sesión para agotar la CPU del servidor.

No existe una regla de oro para el factor de trabajo ideal: dependerá del rendimiento del servidor y del número de usuarios de la aplicación. La determinación del factor de trabajo óptimo requerirá la experimentación en el servidor o servidores específicos utilizados por la aplicación. Como regla general, el cálculo de un hash debería tardar menos de un segundo, aunque en los sitios de mayor tráfico debería ser bastante menos que esto.

Otra forma de aumentar la seguridad es repetir el número de iteraciones de hash. Aumentar el número de iteraciones significa que vamos a codificar la contraseña varias veces. Por ejemplo, con sha512 tenemos el siguiente bucle:

```
Siempre que la iteración sea mayor que 0  
hash = sha512 (hash)  
Disminuir la iteración
```

Para un usuario que inicia sesión, el cálculo del hash será más largo (aún toma un milisegundo). Pero cuando un usuario pierde unos milisegundos para iniciar sesión, un atacante perderá mucho más tiempo, porque el atacante perderá varios milisegundos por intento, y dado que el atacante realiza millones de intentos, esto dará como resultado horas / días adicionales para recuperar las contraseñas.

El factor de trabajo es esencialmente el número de iteraciones del algoritmo de hashing que se realizan para cada contraseña (normalmente es realmente 2^{work} iteraciones). El propósito del factor de trabajo es hacer que el cálculo del hash sea más caro computacionalmente, lo que a su vez reduce la velocidad a la que un atacante puede intentar descifrar el hash de la contraseña. El factor de trabajo se suele almacenar en la salida del hash.

Fusionar los 3 métodos

Podemos fusionar los tres métodos (sal, pimienta y número de iteraciones) para tener un método para almacenar contraseñas de forma más segura que un simple hash.

```
Function calculation_hash(password,salt,pepper,iteration)
Inputs
password is the user's password in plain text
salt is the unique salt per user and is randomly generated
pepper is the common pepper for all users and is randomly generated.
iteration is the number of iterations

Output:
The password hash
Hash = sha512(salt+password+pepper)
As long as iteration is greater than 0
hash = sha512(hash)
Decrement iteration

return hash
```

Luego, para verificar las contraseñas al iniciar sesión, simplemente se llama a la misma función con la contraseña ingresada por el usuario y compárela con el hash en la base de datos. Si ambos son idénticos, entonces el inicio de sesión es exitoso.

Usar funciones específicas

Escribir código criptográfico personalizado, como un algoritmo hash, es **realmente difícil y nunca** debería **hacerse** fuera de un ejercicio académico. Cualquier beneficio potencial que puedas tener al

usar un algoritmo desconocido o hecho a medida será ampliamente eclipsado por las debilidades que existen en él.

No lo hagas.

1.7.11 Algoritmos modernos

Hay una serie de algoritmos de hashing modernos que han sido diseñados específicamente para almacenar contraseñas de forma segura. Esto significa que deben ser lentos (a diferencia de algoritmos como MD5 y SHA-1, que fueron diseñados para ser rápidos), y su lentitud puede configurarse cambiando el factor de trabajo.

A continuación se enumeran los tres principales algoritmos que deberían considerarse:

1.7.11.1 Argon2id Argon2 es el ganador del [Concurso de Hashing de Contraseñas](#) de 2015. Hay tres versiones diferentes del algoritmo, y la variante Argon2id debería usarse cuando esté disponible, ya que proporciona un enfoque equilibrado para resistir tanto los ataques de canal lateral como los basados en la GPU.

En lugar de un simple factor de trabajo como otros algoritmos, Argon2 tiene tres parámetros diferentes que pueden configurarse, lo que significa que es más complicado ajustarlo correctamente para el entorno. La especificación contiene [una guía para elegir los parámetros adecuados](#), sin embargo, si no estás en condiciones de ajustarlo correctamente, entonces un algoritmo más simple como Bcrypt puede ser una mejor opción.

1.7.11.2 PBKDF2 PBKDF2 está recomendado por el [NIST](#) y tiene implementaciones validadas por FIPS-140. Por lo tanto, debería ser el algoritmo preferido cuando se requiera. Además, es compatible con el marco de trabajo de .NET, por lo que se utiliza comúnmente en aplicaciones ASP.NET.

PBKDF2 puede utilizarse con HMACs basados en un número de algoritmos hash diferentes. El HMAC-SHA-256 está ampliamente soportado y es recomendado por el NIST.

El factor de trabajo para PBKDF2 se implementa a través del recuento de iteraciones, que debe ser de al menos 10.000 (aunque valores de hasta 100.000 pueden ser apropiados en entornos de mayor seguridad).

1.7.11.3 Bcrypt Bcrypt es el algoritmo más ampliamente soportado y debería ser la elección por defecto a menos que haya requisitos específicos para PBKDF2, o conocimientos apropiados para ajustar Argon2.

El factor de trabajo por defecto para Bcrypt es 10, y por lo general debe ser elevado a 12 a menos que se opere en sistemas más antiguos o de menor potencia.

bcrypt es una función hash creada por Niels Provos y David Mazières. Se basa en el algoritmo de cifrado Blowfish y se presentó en USENIX en 1999.

Entre estos puntos positivos además de los mencionados anteriormente encontramos implementaciones en muchos idiomas. Además, dado que este algoritmo se remonta a 1999, ha mostrado su robustez a lo largo del tiempo, donde algunos algoritmos como Argon2 (i) solo existen desde 2015.

El hash calculado por bcrypt tiene una forma predefinida:

```
$2y$11$SXAXZyioy60hbnymeoJ9.ulscXwUFMhbvLaTxAt729tGusw.5AG4C
```

- Algoritmo: Este puede tomar varias versiones dependiendo de la versión de bcrypt (2, 2a, 2x, 2y y 2b)
- El costo: el número de iteraciones en potencia de 2. Por ejemplo, aquí, la iteración es 11, el algoritmo hará 2^{11} iteraciones (2048 iteraciones).
- Sal: en lugar de almacenar la sal en una columna dedicada, se almacena directamente en el hash final.
- La contraseña hash

Dado que bcrypt almacena el número de iteraciones, esto la convierte en una función adaptativa, porque el número de iteraciones se puede aumentar y, por lo tanto, es cada vez más largo. Esto le permite, a pesar de su antigüedad y la evolución de la potencia informática, seguir siendo robusto contra los ataques de fuerza bruta. El siguiente punto de referencia muestra que el hashcat tarda 23 días en calcular la totalidad de los hashes de rockyou.

En un algoritmo sha512, herramientas como hascat o John de ripper pueden hashear hasta 5.000.000 de entradas por segundo (Hahs/Seconds) mientras que con bcrypt (con 8 iteraciones) sólo puede procesar unas 300

```
Dictionary cache hit:  
* Filename...: /home/falcon/tools/wordlists/rockyou.txt  
* Passwords.: 14344386  
* Bytes.....: 139921519  
* Keyspace..: 14344386  
  
$2y$11$IImvzMi6zpWWz/n3LmqD50oawnDr4nIEFsLwpklnhfY.FR2ND4Tb6:matrix  
$2y$11$aDqioJwzYWc.YQdBbVDZP0g3gTj/Ua5w37gjzzYC3rWVNbqymCc0:matrix  
$2y$11$SXAXZyioy60hbnymeoJ9.ulscXwUFMhbvLaTxAt729tGusw.5AG4C:azerty  
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => s  
  
Session.....: hashcat  
Status.....: Running  
Hash.Type....: bcrypt $2*$, Blowfish (Unix)  
Hash.Target...: /home/falcon/hashes/bcrypt.txt  
Time.Started...: Thu Mar 26 11:56:38 2020 (2 hours, 8 mins)  
Time.Estimated...: Sat Apr 18 20:29:00 2020 (23 days, 5 hours)  
Guess.Base....: File (/home/falcon/tools/wordlists/rockyou.txt)  
Guess.Queue....: 1/1 (100.00%)  
Speed.#1.....: 14 H/s (8.60ms) @ Accel:8 Loops:2 Thr:8 Vec:8  
Recovered.....: 3/5 (60.00%) Digests, 3/5 (60.00%) Salts  
Progress.....: 264320/71721930 (0.37%)  
Rejected.....: 0/264320 (0.00%)  
Restore.Point...: 52864/14344386 (0.37%)  
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:868-870  
Candidates.#1...: 220395 -> 092005  
  
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit =>
```

Figura 16: Hashcat

1.7.12 Conclusión

Hemos visto en este artículo la utilidad de una función hash robusta y la ventaja de utilizar una función ya existente. Además, el problema del almacenamiento de contraseñas tiene problemas legales además de problemas de seguridad.

Finalmente, es interesante notar que en todos los casos las contraseñas azerty y matrix se encontraron rápidamente, mientras que la contraseña yep59f\$4txwrr nunca se encontró. De hecho, como éste no está presente en ninguna lista, la única forma de encontrarlo es realizar una fuerza bruta exhaustiva en 13 caracteres, lo que es una operación que requiere mucho tiempo (debido a la gran cantidad de posibilidades de contraseña). Esto también muestra lo importante que es para una aplicación web forzar una política de contraseña segura.

Basado en

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

<https://stackoverflow.com/questions/674904/salting-your-password-best-practices>

<https://security.stackexchange.com/questions/3272/password-hashing-add-salt-pepper-or-is-salt-enough>

<https://www.vaadata.com/blog/how-to-securely-store-passwords-in-database/>

https://es.wikipedia.org/wiki/Tabla_arco%C3%ADris

https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html

<https://md5hashing.net/hash/md5/21b72c0b7adc5c7b4a50ffcb90d92dd6>

<https://platzi.com/blog/introduccion-json-web-tokens/>

<https://www.bbva.com/es/que-es-fido-el-nuevo-estandar-de-autenticacion-online/>