UNIVERSIDAD COMPLUTENSE MADRID

Autenticación en aplicaciones web

Gestión de la Información en la Web
Enrique Martín - emartinm@ucm.es
Grados de la Fac. Informática

Autenticación en aps. web

- En la inmensa mayoría de aplicaciones web, la autenticación se realiza mediante **contraseñas** (*algo que el usuario sabe*).
- En algunos servicios críticos, se puede **complementar** con *algo que el usuario tiene*, como el **teléfono móvil** para recibir un PIN temporal (SMS, llamada, *app*) o una **tarjeta de coordenadas** → 2º **factor de autenticación**.
- En cualquier caso, el servidor debe almacenar las contraseñas del usuario.

Contraseñas en el servidor

- La opción más sencilla es **almacenar los secretos directamente** en la base de datos de la aplicación web. Ej:

```
pepe:12345678:1997-05-12
ana:password90:2014-06-01
```

- **Autenticar** a un usuario es comprobar que la contraseña proporcionada **es la misma** que la almacenada en la base de datos.
- La seguridad de las contraseñas será la misma que la de la base de datos.

Contraseñas en el servidor

- Sin embargo, hay muchas situaciones en las que la seguridad de la base de datos no es suficiente:
  - Inyección de SQL (o de comandos en general)
  - Vulnerabilidades de la BBDD o del servidor
  - Administrador malintencionado
- En todos estos casos, el atacante podría obtener **directamente las contraseñas** de los usuarios.

Contraseñas en el servidor

- Esto provoca que el atacante pueda acceder a la aplicación web como cualquier usuario. Pero es **aún peor**...
  - ¿Cuántas **cuentas** habéis creado en aplicaciones web?
  - ¿Cuántas **direcciones e-mail o nombre de usuario diferentes** habéis utilizado?
  - ¿Y cuántas **contraseñas diferentes**?
- Lo más usual es **repetir la misma pareja usuario/contraseña** para muchas aplicaciones diferentes.

Contraseñas en el servidor

- Si almacenamos las contraseñas directamente, un descuido o vulnerabilidad puede **comprometer la seguridad de muchas otras aplicaciones web**.
- Es más seguro **almacenar** en el servidor ***un valor* obtenido a partir de la contraseña**, de tal manera que podamos utilizarlo para autenticar.

Hash criptográfico

- Para mejorar la seguridad a la hora de almacenar contraseñas, se utilizan **funciones hash criptográficas**.
- Estas funciones generan **bloques de tamaño fijo**, de tal manera que cambios en la entrada producirán cambios en la salida.
- No confundir con las funciones hash utilizadas de **tablas hash**, que únicamente son funciones de **dispersión: valor → natural**.

Hash criptográfico

- Una función **hash criptográfica**:
  - Debe ser **sencilla** de calcular (poco cómputo)
  - Debe ser **impracticable** generar un mensaje con un hash dado.
  - Debe ser **impracticable** modificar un mensaje y que siga generando el mismo hash.
  - Debe ser **impracticable** encontrar dos mensajes con el mismo hash.
- Estas funciones se usan también para calcular *checksums* de ficheros.

Hash criptográfico

- Para cifrar las contraseñas en las aplicaciones web se suelen usar algoritmos como:
  - MD5
  - SHA-0, SHA-1, **SHA-256, SHA-512, SHA-3**...
  - **RIPEMD**
  - **WHIRLPOOL**
- Se recomienda **no utilizar MD5 o SHA-1**, ya que se han encontrado formas de calcular **colisiones** (mensajes con el mismo hash).

Hash criptográfico

- Si la contraseña está cifrada con una función hash criptográfica, un atacante no la podrá utilizar **de manera directa** aunque la obtenga:

```
pepe:ef797c8118f02dfb6...cc95c5ed7a898a64f:1997-05-12
ana:ef797c8118f02dfb6...63d532cc95c5ea64f:2014-06-01
```

- Sin embargo...
  - Las funciones hash son deterministas: misma entrada → misma salida
  - La mayoría de los usuarios utiliza palabras comunes (diccionarios) o contraseñas débiles

Hash criptográfico

- Se pueden realizar listados precalculados para una función hash y diccionarios concretos: **rainbow tables**. P.ej con SHA-256:

```
hash("hello")-> 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hbllo")-> 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366
…
hash("waltz")-> c0e81794384491161f1777c232bc6bd9ec38f616560b120fda8e90f383853542
```

- Un atacante puede buscar en estos listados, que están disponibles en Internet:
  - http://project-rainbowcrack.com/table.htm
  - http://www.pwcrack.com/rainbowtables.shtml

Hash criptográfico

- Los *hashes* de las claves débiles aparecerán en los listados, así que el atacante podrá obtener las contraseñas originales.
- ¿Cómo evitarlo?
  - Forzando a los usuarios a utilizar **contraseñas fuertes**: largas, con variedad de caracteres → **mala idea**.
  - **Añadir algo más a la clave antes de encriptarla**, de tal manera que el *hash* resultante de cada clave sea diferente en cada aplicación web.

Sazonando las claves

- Para evitar el problema anterior, se **añade sal** a la contraseña antes de calcular el hash.
- La sal es una cadena que se concatena a la contraseña (p.ej después). Se debe generar de manera **aleatoria**.
- La sal conseguirá que a partir de claves iguales se obtengan *hashes* diferentes:

```
hash("hello")                    = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hello" + "QxLUF1bgIAdeQX") = 9e209040c863f84a31e719795b2577523954739fe5ed3b58a75cff2127075ed1
hash("hello" + "bv5PehSMfV11Cd") = d1d3ec2e6f20fd420d50e2642992841d8338a314b8ea157c9e18477aaef226ab
hash("hello" + "YYLmfY6IehjZMQ") = a49670c3c18b9e079b9cfaf51634f563dc8ae3070db2c4a8544305df1b60f007
```

Sazonando las claves

- Consejos importantes para la sal:
  - Para cada contraseña, hay que generar una **sal nueva y aleatoria**.
  - **No hay que reutilizar la sal**, ni entre usuarios ni dentro del mismo usuario. *Si cambia su contraseña hay que generar una nueva sal*.
  - La **sal** debe ser **larga**. Como norma general tan larga como la salida de la función *hash*.
  - La sal **no tiene por qué ser secreta**, se puede almacenar tal cual en la BD junto con el valor hash obtenido.

Sazonando las claves

- La **sal** se almacenará en nuestra base de datos junto con el resultado de la función hash:

```
pepe:QxLUF1bgIAdeQX:9e209040c863f8...ed1:1997-05-12
ana:YYLmfY6IehjZMQ:6511b2429b4ceb3...05f:2014-06-01
```

- En ocasiones la sal se complementa con **pimienta**, un valor **constante** que está **incrustado** en el código del programa y nunca almacenado.
- Antes de usar la función hash, se concatenan tanto la **sal** como la **pimienta** a la contraseña.
- Se pueden tener varios valores de pimienta en el sistema. Para autenticar se deberían probar todos.

Cuando la sal no es suficiente

- La sal impide/dificulta el uso de tablas precalculadas para obtener las contraseñas cifradas. Esto evita que obtengan muchas contraseñas.
- Pero si un atacante roba tus *hashes* y tiene **mucho interés** en conocer la clave de **un usuario concreto**, puede realizar un ataque de fuerza bruta centrado en él.
- Las funciones *hash* son rápidas, lo que *ayuda* al atacante. *¿Cómo evitarlo?*

Cuando la sal no es suficiente

- Para mitigar los ataques de fuerza bruta se pueden utilizar algoritmos de **ralentizado** como PBKDF2, o funciones de **derivación de claves** como bcrypt, scrypt o Argon2.
- Básicamente **aplican en cadena la función** de *hash* para obtener un resultado derivado.
- Es necesario encontrar un **equilibrio entre velocidad y seguridad**, o puedes sobrecargar el servidor a base de autenticaciones muy costosas.

Almacenamiento de parámetros

- Según aumenta la capacidad de cómputo, será necesario aplicar más iteraciones.
- Es común almacenar varios datos sobre la contraseña: ralentizado aplicado, iteraciones, función hash usada...

  `pepe:pbkdf2-sha256:29000:N2YMIWQsBWBM:9e209040c863f8...ed1:1997-05-12`

- Esto permite aumentar el número de iteraciones o usar una función hash más segura durante una **autenticación con éxito (único momento donde tenemos la clave real)**.

Transmitir la clave

- **Siempre** que un usuario introduzca una clave, debe ser en una página **HTTPS**:
  - Podrá confirmar en qué web está introduciendo la clave.
  - La clave se enviará cifrada con la contraseña de sesión TLS.
- **Siempre** aplicar la **función hash** en el **servidor**. Si quieres puedes aplicar hash **también** en el cliente, pero **nunca sólo en el cliente**.