Gestión de la Información en la Web
Autenticación y almacenamiento de claves

Fecha de entrega: martes 26 de enero de 2016, 13:55h

Entrega de la práctica
La entrega de la práctica se realizará a través del Campus Virtual de la asignatura mediante un fichero **grupoXX.zip** donde **XX** es el numero de grupo. Este fichero constará de un fichero **autenticacion.py** con el código del servidor web, cuyo esqueleto se puede descargar del Campus Virtual. El fichero **autenticacion.py** comenzará con un comentario indicando los autores, el grupo y una **declaración de integridad** expresando que el código es fruto del trabajo de sus miembros. Además del servidor web, el fichero **ZIP** contendrá las vistas/plantillas necesarias para mostrar los datos adecuadamente (si las habéis utilizado).

Objetivos mínimos
Almacenar y gestionar de manera segura las contraseñas en una aplicación web.

Objetivos opcionales
Incorporar un segundo factor de autenticación mediante *Time-based One-Time Passwords*.

Calificación
El apartado A aporta el 40% de la nota, y el apartado B aporta el 60% restante.

Otras consideraciones
El código entregado será resultado exclusivamente del trabajo de sus miembros. No se permite ningún tipo de colaboración entre grupos. Cualquier fragmento de código que aparezca repetido en distintas prácticas o copiado de alguna fuente externa implicará **automáticamente** una calificación de cero y se trasladará dicha situación a las autoridades académicas competentes.

En esta práctica implementaremos distintas peticiones de gestión de usuarios (creación, acceso y cambio de contraseña) teniendo como prioridad **almacenar de la manera más segura las contraseñas** de los usuarios. De esta manera las contraseñas de nuestros usuarios estarán protegidas aunque un atacante acceda a nuestra base de datos.

Todos los datos de los usuarios se almacenarán en MongoDB, utilizando para ello la librería *pymongo* como en prácticas anteriores. En esta ocasión tenéis libertad para escoger el esquema que prefiráis, pero cada usuario se deberá almacenar como un único documento en la colección **users** de la base de datos **giw** ***(es imprescindible respetar el nombre de la base de datos y de la colección).***

Para realizar la práctica debéis descargar el esqueleto básico del servidor web desde el Campus Virtual y usarlo como base. Este esqueleto incluye 2 apartados con algunas funciones y distintas rutas en las que el servidor web debe responder a peticiones **POST** (no podéis cambiar las rutas, el método HTTP ni añadir nuevas rutas).

Apartado A: Autenticación básica mediante contraseñas

1.- /signup
Esta petición sirve para dar de alta a un usuario. Recibe los siguientes parámetros **POST**:

- **nickname**: alias del usuario (único).
- **name**: nombre completo del usuario (incluye apellidos).
- **country**: país de residencia del usuario.
- **email**: correo electrónico del usuario.
- **password**: contraseña escogida por el usuario.
- **password2**: contraseña repetida para comprobar que está bien escrita.

En respuesta a esta petición el servidor web hará lo siguiente:

- Si las 2 contraseñas no coinciden no realizará ninguna modificación en la base de datos y devolverá una página web con el mensaje "*Las contraseñas no coinciden*".
- Si el alias de usuario ya existe en nuestra colección no realizará ninguna modificación en la base de datos y devolverá una página web con el mensaje "*El alias de usuario ya existe*".
- En otro caso insertará el usuario en la colección **users** y devolverá una página web con el mensaje "*Bienvenido usuario* <***name***>", donde <***name***> es el nombre completo del usuario.

2.- /change_password
Cambia la contraseña de un usuario. Recibe como parámetros **POST**:

- **nickname**: alias del (usuario).
- **old_password**: contraseña antigua.
- **new_password**: contraseña nueva.

En respuesta a esta petición el servidor hará lo siguiente:

- Si el alias del usuario no existe en nuestra base de datos devolverá una página web con el mensaje "*Usuario o contraseña incorrectos*".
- Si **old_password** no coincide con la contraseña actual devolverá una página web con el mensaje "*Usuario o contraseña incorrectos*".
- En otro caso actualizará la contraseña en la base de datos y devolverá una página web con el mensaje "*La contraseña del usuario* <***nickname***> *ha sido modificada*".

3.- /login
Autentica un usuario. Recibe como parámetros **POST**:

- **nickname**: alias del usuario (único).
- **password**: contraseña.

En respuesta a esta petición el servidor hará lo siguiente:

- Si el alias del usuario no existe en nuestra base de datos devolverá una página web con el mensaje "*Usuario o contraseña incorrectos*".
- Si **password** no coincide con la contraseña actual devolverá una página web con el mensaje "*Usuario o contraseña incorrectos*".
- En otro caso devolverá una página web con el mensaje "*Bienvenido* <***name***>" donde <***name***> es el nombre completo del usuario.

Apartado B: Autenticación usando un segundo factor de autenticación

Muchas personas repiten su nombre de usuario y contraseña en múltiples aplicaciones web, por lo que su revelación puede comprometer la seguridad de todas ellas. Lamentablemente esta situación no es tan remota y varias veces al año se producen ataques a grandes *webs* que terminan con la publicación de inmensos listados de usuarios y contraseñas (¿alguna vez has echado un vistazo a https://haveibeenpwned.com o a https://hesidohackeado.com?). Para impedir que la revelación del nombre de usuario y contraseña ponga en peligro todas las demás aplicaciones que comparten esas credenciales, es muy conveniente proporcionar un **segundo factor de autenticación**. Este mecanismo usualmente combina los credenciales usuales (nombre y contraseña, *algo que sabes*) con otro mecanismo que usualmente es *algo que tienes* (el móvil, una tarjeta de coordenadas). De esta manera a la hora de acceder a un servicio se te solicita el usuario y la contraseña **y además** un código temporal de un solo uso que, en el caso del móvil, te llega vía SMS o es generado por una aplicación concreta como **Google Authenticator**. Aunque tu usuario y contraseña hubieran sido comprometidos nadie podría acceder a tu cuenta (en principio) ya que no tendría acceso al dispositivo generador de claves temporales.

En este apartado vamos a incorporar un segundo factor de autenticación a un servidor web mediante la *app* **Google Authenticator**[1], disponible para Android, iOS y Blackberry (https://support.google.com/accounts/answer/1066447?hl=es). Esta *app* genera *Time-based One-Time Passwords* (TOTP), es decir, contraseñas de un solo uso basadas en la hora actual y que cambian cada 30 segundos. Básicamente utiliza un algoritmo de generación de números pseudo-aleatorios que a partir de una **semilla** y la **hora actual** genera un código de manera determinista. Para incorporar estos TOTPs a nuestro servidor web es necesario que tanto el móvil como el servidor web estén sincronizados: ambos deben conocer la semilla y tener el reloj en hora para generar los mismos códigos en los mismos instantes de tiempo.

Para integrar TOTPs mediante Google Authenticator en nuestro servidor web tendremos que modificar ligeramente el proceso de alta a un usuario:

1. Generar una semilla. Esta semilla debe ser una **cadena de 16 caracteres** tomados aleatoriamente entre las 26 letras mayúsculas del inglés y los dígitos 2, 3, 4, 5, 6 y 7.
2. Almacenar la semilla en el documento del usuario dentro la base de datos para que el servidor web pueda generar el código temporal actual y compararlo con el que proporciona el usuario.
3. Comunicar la semilla al usuario para que añada una nueva cuenta a *Google Authenticator*. La *app* soporta dos maneras de añadir una nueva cuenta:
    - Manualmente a partir de un nombre de cuenta y la semilla. Este proceso es proclive a fallos ya que hay que introducir manualmente la cadena de 16 caracteres.
    - A partir de una URL convenientemente formada que *Google Authenticator* analiza y de la que extrae la información necesaria para añadir la cuenta. El formato de esta URL, que se explica en https://github.com/google/google-authenticator/wiki/Key-Uri-Format, es el siguiente:

      otpauth://totp/<**USERNAME**>?secret=<**SECRET**>&issuer=<**APP_NAME**>

1 Si algún alumno no utiliza *Google Authenticator* y no quiere instalar esta *app* podrá realizar las pruebas con el generador de TOTPs online disponible en http://gauth.apps.gbraad.nl o con la propia librería **onetimepass**.

donde **<USERNAME>** es el nombre del usuario, **<SECRET>** la cadena de 16 caracteres generada automáticamente que sirve como semilla y **<APP_NAME>** el nombre del servidor web tal y como lo mostrará *Google Authenticator*. Un ejemplo de URL sería:

otpauth://totp/**pepe_lopez**?secret=**JBSWY3DPEHPK3PXP**&issuer=**GIW_grupo89**

Como esta URL tiene que ser procesada por la *app* del móvil, la opción más cómoda es generar un código QR (https://es.wikipedia.org/wiki/C%C3%B3digo_QR) que codifique dicha URL. De esta manera el usuario solo tendrá que escanear el código en su móvil y la cuenta se añadirá de manera automática a *Google Authenticator*.

Para este apartado será necesario implementar las siguientes funciones y rutas del servidor web:

1.- def gen_secret():
Genera una cadena aleatoria de 16 caracteres a escoger entre las 26 letras mayúsculas del inglés y los dígitos 2, 3, 4, 5, 6 y 7. Ejemplo:

```
>>> gen_secret()
'7ZVVBSKR22ATNU26'
```

2.- def gen_gauth_url(app_name, username, secret):
Genera la URL para insertar una cuenta a *Google Authenticator* a partir de sus fragmentos. Ejemplo:

```
>>> gen_gauth_url( 'GIW_grupoX', 'pepe_lopez', 'JBSWY3DPEHPK3PXP')
'otpauth://totp/pepe_lopez?secret=JBSWY3DPEHPK3PXP&issuer=GIW_grupoX'
```

3.- def gen_qrcode_url(gauth_url):
Para la generación del código QR a partir de la URL utilizaremos el servicio externo **goqr.com** ***(sería preferible que nuestro propio servidor web generase los códigos QR para evitar compartir las semillas con terceros, pero aquí usaremos goqr.com para simplificar la práctica)***. Este servicio nos proporciona una API que genera imágenes con códigos QR a partir de los datos que queremos codificar. Por ejemplo para generar el código QR con contenido "GIW" realizaríamos una petición a https://api.qrserver.com/v1/create-qr-code/?data=GIW. Podéis encontrar más información sobre el resto de parámetros para la generación de códigos QR en http://goqr.me/api/doc/create-qr-code/.

A partir de la URL de *Google Authenticator*, la función **def gen_qrcode_url** genera la URL de la petición a goqr.com que generará el código QR. Por ejemplo (las cadenas de texto deben aparecer en la misma línea, pero aquí han sido separadas por limitación de espacio):

```
>>> gen_qrcode_url('otpauth://totp/pepe_lopez?
secret=JBSWY3DPEHPK3PXP&issuer=GIW_grupoX')
'https://api.qrserver.com/v1/create-qr-code/?data=otpauth%3A%2F
%2Ftotp%2Fpepe_lopez%3Fsecret%3DJBSWY3DPEHPK3PXP%26issuer
%3DGIW_grupoX'
```

***Nota**: Es importante codificar los parámetros adecuadamente, ya que formarán parte de una URL.*

4.- /signup_totp
Esta petición sirve para dar de alta a un usuario. Recibe los siguientes parámetros **POST**:

- **nickname**: alias del usuario (único).
- **name**: nombre completo del usuario (incluye apellidos).
- **country**: país de residencia del usuario.
- **email**: correo electrónico del usuario.
- **password**: contraseña escogida por el usuario.
- **password2**: contraseña repetida para comprobar que está bien escrita.

En respuesta a esta petición el servidor web hará lo siguiente:

- Si las 2 contraseñas no coinciden no realizará ninguna modificación en la base de datos y devolverá una página web con el mensaje "*Las contraseñas no coinciden*".
- Si el alias de usuario ya existe en nuestra colección no realizará ninguna modificación en la base de datos y devolverá una página web con el mensaje "*El alias de usuario ya existe*".
- En otro caso insertará al usuario en la colección **users** y devolverá una página web con el código QR para configurar *Google Authenticator*. Esta página web contendrá también el nombre de usuario y la semilla generada por si el usuario quiere configurar *Google Authenticator* manualmente o utilizar otro generador TOTP.

5.- /login_totp
Autentica un usuario utilizando un segundo factor. Recibe como parámetros **POST**:

- **nickname**: alias del usuario (único).
- **password**: contraseña.
- **totp**: código temporal generado por *Google Authenticator*.

En respuesta a esta petición el servidor hará lo siguiente:

- Si el alias del usuario no existe en nuestra base de datos, la contraseña no coincide o el código temporal no es válido devolverá una página web con el mensaje "*Usuario o contraseña incorrectos*".
- En otro caso devolverá una página web con el mensaje "*Bienvenido* <***name***>" donde <***name***> es el nombre completo del usuario.

Para poder comprobar la validez del código temporal enviado por el usuario nuestro servidor web debe ser capaz de generarlos a partir de la semilla y la hora actual. Para ello utilizaremos la biblioteca **onetimepass** (https://github.com/tadeck/onetimepass) que debéis descargar desde el Campus Virtual y situar en la misma carpeta que el servidor web[2]. Esta biblioteca os proporciona 2 funciones para generar y validar TOTPs generados a partir de la hora actual y la semilla:

```
>>> get_totp('2PIKOQJWBVXKGLMB', as_string=True)
'761183'

>>> valid_totp('761183', '2PIKOQJWBVXKGLMB')
True

(30 segundos más tarde)
>>> valid_totp('761182', '2PIKOQJWBVXKGLMB')
False
```

2 Esta es la manera de instalarlo en los laboratorios de la Facultad pues no tenemos permisos de escritura. En cualquier otra máquina solo necesitaríamos ejecutar **sudo pip install onetimepass**.