Gestión de la Información en la Web
Curso 2021-22
Práctica 9 – Autenticación & TOTP

Fecha de entrega: domingo 28 de noviembre de 2021, 23:55h

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 número de grupo. Este ZIP constará de un fichero `autenticacion.py` con el código del servidor web, cuyo esqueleto se puede descargar del Campus Virtual. Además del servidor web, el fichero ZIP contendrá las plantillas y otros ficheros necesarios para mostrar los datos adecuadamente (si habéis utilizado).

Lenguaje de programación
Python 3.8 o superior.

Calificación
El apartado 1 aporta el 40 % de la nota, y el apartado 2 aporta el 60 % restante.

Declaración de autoría e integridad
Todos los ficheros entregados contendrán una cabecera en la que se indique la asignatura, la práctica, el grupo y los autores. Esta cabecera también contendrá la siguiente declaración de integridad:

> *(Nombres completos de los autores)* declaramos que esta solución es fruto exclusivamente de nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona ni hemos obtenido la solución de fuentes externas, y tampoco hemos compartido nuestra solución con nadie. Declaramos además que no hemos realizado de manera deshonesta ninguna otra actividad que pueda mejorar nuestros resultados ni perjudicar los resultados de los demás.

No se corregirá ningún fichero que no venga acompañado de dicha cabecera.

En esta práctica implementaremos distintas acciones de gestión de usuarios (creación, acceso y cambio de contraseña) teniendo como prioridad **almacenar de la forma 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.

Los datos de los usuarios se almacenarán usando MongoEngine en la base de datos `giw_auth`. Para realizar la práctica debéis descargar el esqueleto básico del servidor web desde el Campus Virtual y usarlo como base. Este esqueleto contiene la definición de una clase `User` con el esquema que debéis utilizar para almacenar los usuarios. Además, este fichero 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). Para probar fácilmente el funcionamiento de vuestra solución podéis descargar distintos formularios HTML muy básicos para abrir en el navegador y realizar peticiones POST al servidor web *Flask* en `http://localhost:5000`. Por favor, revisad que vuestra práctica funciona correctamente con dichos formularios, ya que son los mismos que luego uso para la corrección.

1. Autenticación básica mediante contraseñas [4pt]

El fichero `autenticacion.py` debe contener un comentario al inicio de este apartado en el que se **describa y explique detalladamente** el mecanismo escogido para el almacenamiento de contraseñas y se explique razonadamente por qué es seguro.

1. `/signup`
   Esta petición sirve para dar de alta a un usuario. Recibe los siguientes parámetros POST:
   - `nickname`: identificador del usuario (único).
   - `full_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 coincide.

   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 identificador de usuario ya existe en nuestro sistema no realizará ninguna modificación en la base de datos y devolverá una página web con el mensaje `"El usuario ya existe"`.
   - En otro caso insertará el usuario en la base de datos y devolverá una página web con el mensaje `"Bienvenido usuario <name>"`, donde `<name>` es el nombre completo del usuario.

2. `/change_password`
   Actualiza la contraseña de un usuario. Recibe como parámetros POST:
   - `nickname`: identificador del usuario.
   - `old_password`: contraseña antigua.
   - `new_password`: contraseña nueva.

   En respuesta a esta petición el servidor hará lo siguiente:

- Si el identificador del usuario no existe en nuestra base, o si `old_password` no coincide con la contraseña almacenada entonces 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`: identificador del usuario.
   - `password`: contraseña.

   En respuesta a esta petición el servidor hará lo siguiente:

   - Si el identificador del usuario no existe en nuestra base de datos, o si `password` no coincide con la contraseña almacenada entonces 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.

2. Autenticación con TOTP [6pt]

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 inusual y varias veces al año se producen ataques a grandes webs que terminan con la publicación de inmensos listados de usuarios y contraseñas (puedes comprobar si estás afectado en `https://haveibeenpwned.com`). En este apartado vamos a incorporar un segundo factor de autenticación al servidor web mediante TOTP con Google Authenticator[1].

2.1. Alta de usuarios

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

1. Generar un secreto aleatorio representado en **Base32**[2].

2. Almacenar este secreto junto con la información 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 en el momento de la autenticación.

3. Comunicar el secreto 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 el secreto. Este proceso es proclive a fallos ya que hay que introducir manualmente la cadena de texto que representa el secreto.
   - A partir de una URL convenientemente formada[3] que Google Authenticator y aplicaciones similares analizan y de la que extraen la información necesaria para registrar la cuenta.

[1] También se puede usar *Latch*, *FreeOTP Authenticator*, `http://gauth.apps.gbraad.nl` o la biblioteca `pyotp` de Python.

[2] `https://es.wikipedia.org/wiki/Base32`

[3] `https://github.com/google/google-authenticator/wiki/Key-Uri-Format`

Como esta URL tiene que ser procesada por la app del móvil, la opción más cómoda es generar un código QR[4] 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 generar este código QR existen dos alternativas:

a) Utilizar algún servicio externo para la generación de códigos QR a partir de URLs como `http://goqr.me/api/`.

b) Utilizar alguna biblioteca Python para generar el código QR en local e incrustarlo en la página HTML.

2.2. Rutas a definir

Para este apartado de la práctica será necesario implementar las siguientes rutas del servidor web:

1. `/signup_totp`
   Esta petición sirve para dar de alta a un usuario. Recibe los siguientes parámetros POST:
   - `nickname`: identificador del usuario (único).
   - `full_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 coincide.

   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 identificador de usuario ya existe en nuestro sistema no realizará ninguna modificación en la base de datos y devolverá una página web con el mensaje `El usuario ya existe`.
   - En otro caso insertará al usuario en la base de datos y devolverá una página web con el **código QR** para configurar Google Authenticator. Esta página web contendrá también el **nombre de usuario** y el **secreto** generado por si el usuario quiere configurar Google Authenticator manualmente.

2. `/login_totp`
   Autentica un usuario utilizando dos factores. Recibe como parámetros POST:
   - `nickname`: identificador del usuario.
   - `password`: contraseña.
   - `totp`: código TOTP generado por Google Authenticator o cualquier aplicación similar.

   En respuesta a esta petición el servidor hará lo siguiente:
   - Si el identificador del usuario no existe en nuestra base de datos, la contraseña no coincide o el código TOTP 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 comprobar la validez del código temporal enviado por el usuario nuestro servidor web usará la biblioteca onetimepass (ver transparencias).

[4] `https://es.wikipedia.org/wiki/Codigo_QR`