REDES DE COMUNICACIONES II

Práctica 2 - Seguridad y criptografía

Volver a: Prácticas

2 Funcionalidad

2.4 Descripción del API

En esta sección se describe con detalle las distintas funciones que proporciona el API del servidor, que deberán ser llamadas adecuadamente por el cliente para implementar la funcionalidad requerida.

Como ya se ha comentado, el API se ha diseño en forma de API REST. Asegúrate de entender bien en qué consiste, y sus principales ventajas antes de continuar [1][2].

Autenticación

En las aplicaciones Web tradicionales, la autenticación de los usuarios se realiza normalmente con esquemas de usuario/contraseña, y almacenando luego el ID del usuario en una sesión (que suele implementarse con cookies en el lado del cliente). Cuando el usuario visita de nuevo una página que necesita autenticación, el navegador envia la cookie al servidor, éste busca la ID del usuario en la BD y, si su sesión no ha expirado, permite el acceso sin tener que volver a introducir las credenciales.

Con un API, puesto que suelen estar diseñadas para ser llamadas desde aplicaciones, y no desde un navegador, el uso de sesiones y esquemas de usuario/contraseña no es la mejor opción. La solución más común es utilizar autenticación basada en tokens.

El funcionamiento es muy sencillo: el usuario se autentica una única vez con su usuario/contraseña, o un mecanismo similar, y la aplicación devuelve un token único para ese usuario que deberá ser enviada en cada petición al API posterior. Un estándar muy común que implementa este método, utilizado hoy en día por prácticamente todas las APIs, se llama OAuth, específicamente su versión 2.

Para empezar a utilizar las funciones del API de la práctica, deberás, por tanto, solicitar un token de autenticación primero. Para ello dirígete a la URL:

```
https://vega.ii.uam.es:8080
```

Allí se solicitarán unos datos, que deberás rellenar. Recibirás entonces en el correo especificado el token, que deberás incluir en una cabecera `HTTP Authorization` de todas las llamadas al API. Concretamente, si el token que has recibido es AaFBe2d7894C1D63, la cabecera HTTP a incluir quedaría:

```
Authorization: Bearer AaFBe2d7894C1D63
```

Endpoints

Las diferentes funciones o métodos que proporciona un API para que sean llamados reciben el nombre de *endpoints*. En el API de SecureBox, existen los siguientes:

Funciones relacionadas con la gestión de identidades

- */users/register* - registra un usuario en el sistema
- */users/getPublicKey* - obtiene la clave pública de un usuario
- */users/search* - obtiene datos de un usuario por nombre o correo electrónico
- */user/delete* - borrar un usuario

Funciones relacionadas con la gestión de ficheros

- */files/upload* - sube un fichero al sistema
- */files/download* - descarga un fichero
- */files/list* - lista todos los ficheros pertenecientes a un usuario
- */files/delete* - borra un fichero

Todas estas funciones cuelgan de la URL *https://vega.ii.uam.es:8080/api*, de forma que la llamada para subir un fichero, por ejemplo, quedaría de la siguiente forma:

```
https://vega.ii.uam.es:8080/api/files/upload
```

Ten en cuenta, también, que todos los endpoints del API esperan recibir los datos en formato JSON, así que asegúrate de enviarlos correctamente. Por ejemplo, si utilizas la librería *requests* de Python, puedes utilizar un código similar al siguiente:

```
url = 'https://vega.ii.uam.es:8080/api/users/getPublicKey'
args = {'idUser': 123}
r = requests.post(url, json=args)
print(r.text)
```

Gestión de identidades

Estas funciones sirven para la gestión de las identidades de los usuarios que utilicen *SecureBox*. A grandes rasgos, por tanto, para empezar a poder utilizar una identidad el cliente deberá:

- Obtener un token de autenticación
- Crear un par de claves público/privada
- Registrar el usuario en el sistema, con su nombre, correo y clave pública

Registro de usuarios - /users/register

Argumentos:

- **nombre**: nombre completo del usuario, para que pueda ser buscado después por otros usuarios.
- **email**: correo electrónico.
- **publicKey**: clave pública del usuario, que será utilizada por otros usuarios para enviarle ficheros cifrados. Deberá utilizarse el formato PEM.

Respuesta

Como todos los endpoints, la respuesta del API será en formato JSON, con los siguientes parámetros:

- **nombre**: nombre completo del usuario.
- **ts**: marca de tiempo (timestamp), en formato UNIX, de registro del usuario.

Posibles errores: USER_ID2

Ejemplo

Petición

```
POST https://vega.ii.uam.es:8080/api/users/register HTTP/1.1
Content-Type: application/json
Content-Length: 102
Authorization: Bearer AaFBe2d7894C1D63

{'nombre': 'Antonio Chicharro', 'email': 'antonio.chicharro@estudiante.uam.es', 'publicKey': '---B
EGIN ....'}
```

Respuesta

Si la petición es correcta, el servidor devuelve un código HTTP `200 OK` junto con información adicional:

```
HTTP/1.1 200
Content-Type: application/json
{'nombre': 'Antonio Chicharro', 'ts': '1513781531.16802'}
```

Obtener clave pública - /users/getPublicKey

Obtiene la clave pública de un usuario. Típicamente esta función se llamará cuando sea necesaria encontrar esta clave para poder enviar a un usuario concreto un fichero cifrado.

Argumentos:

- **userID**: identificador único del usuario cuya clave pública solicitamos.

Respuesta

- **publicKey**: clave pública solicitada

Posibles errores: USER_ID2

Búsqueda de usuarios - /users/search

Busca los metadatos de un usuario a partir de su nombre o correo electrónico.

Argumentos: estructura JSON con los siguientes parámetros:

- **data_search**: cadena de búsqueda, que será contrastada contra el nombre y correo electrónico.

Respuesta: si no se producen errores, la respuseta del API es una estructura JSON con los siguientes parámetros:

Como todos los endpoints, la respuesta del API será en formato JSON, con los siguientes parámetros **userID, nombre, email, publicKey** y **ts**

Posibles errores: USER_ID3

Borrado de usuarios - /users/delete

Elimina un usuario a partir de su identificador.

Argumentos: estructura JSON con los siguientes parámetros:

- **userID:** ID del usuario a ser borrado

Respuesta: si no se producen errores, la respuseta del API es una estructura JSON con los siguientes parámetros:

- **userID**: ID del usuario borrado

Posibles errores: USER_ID2

Gestión de ficheros

Estas funciones gestionan el almacenamiento de ficheros en *SecureBox*. Esencialmente, el flujo de llamadas sería el siguiente:

- Un usuario A desea enviar un fichero a otro usuario B. Para ello, debe comenzar por obtener la clave pública de B.
- Preparar (cifrar y firmar) el fichero para su envío a B a través de SecureBox.
- Subir el fichero a SecureBox, lo que devuelve un identificador, *file_id*.
- Enviar el identificador a B por cualquier medio (correo electrónico, mensajería, etc.), para que éste pueda solicitar su descarga.
- B comprueba la firma del fichero y, si es correcta, descifra el contenido.

Subida de ficheros - /files/upload

Función para subir un fichero al sistema que, como se ha comentado, debe ser previamente cifrado y firmado. Otro usuario podrá, después, descargarlo, descifrarlo y verificar su firma.

Argumentos: esta es la única función de todo el API que no recibe los argumentos JSON, sino como un formulario HTML tipo POST, con nombre *ufile*.

Respuesta: la respuesta, sin embargo, sí se realizará en JSON, con los siguientes parámetros:

- **file_id**: identificador del fichero asignado por el sistema, necesario para solicitar su descarga posterior.
- **file_size**: tamaño del fichero, en bytes. El tamaño aceptado está limitado a 50KB.

Posibles errores: FILE1

Ejemplo

Con un comando como el siguiente:

```
# curl --verbose -H "Authorization: Bearer AaFBe2d7894C1D63" -F "ufile=@/home/pepito/analysis.py" https://vega.ii.uam.es:8080/api/files/upload
```

Si la respuesta es corercta, el servidor responderá con un código de error HTTP `200 OK` , como habitualmente, e información de metadatos:

```
HTTP/1.1 200
Content-Type: application/json

{"file_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxx", "file_size": "12345"}
```

Descarga de ficheros - /files/download

Función para descargar un fichero del sistema.

Argumentos: estructura JSON con los siguientes parámetros:

- **file_id**: identificador del fichero a descargar

Respuesta: si no se producen errores, la respuesta del API es directamente el contenido binario del fichero.

Posibles errores: FILE2 - El fichero no existe

Listado de ficheros - /files/list

Función para listar todos los ficheros pertenecientes a un usuario.

Argumentos: no necesita

Respuesta: si no se producen errores, la respuseta del API es una estructura JSON con los siguientes parámetros:

- **files_list**: lista con los ID de todos los ficheros pertenecientes al usuario.
- **num_files**: número total de elementos de la lista anterior.

Posibles errores: Ninguno

Borrado de ficheros - /files/delete

Función para borrar un fichero del sistema.

Argumentos: estructura JSON con los siguientes parámetros:

- **file_id**: identificador del fichero a borrar

Respuesta: si no se producen errores, la respuseta del API es una estructura JSON con los siguientes parámetros:

- **file_id**: identificador del fichero borrado

Posibles errores: FILE2 - El fichero no existe

Gestión de errores

Todos los *endpoints* del API devuelven los errores en dos niveles diferentes:

- Códigos de error HTTP en la cabecera de la respuesta HTTP
- Un objeto JSON en el cuerpo de la respuesta, con información adicional, como el siguiente ejemplo.

```
{"http_error_code": 401, "error_code": "TOK1", "description": "Token de usuario incorrecto"}
```

Los campos son bastante auto-descriptivos, pero su significado es el siguiente:

- **http_error_code**: código de error HTTP, para un primer filtrado.
- **error_code**: código de error JSON específico del API (tabla con todos los códigos a continuación)
- **description**: descripción del error en lenguaje natural. Puede (y debe) ampliarse para dar al usuario más información sobre las posibles causas del error y qué puede hacer para solucionarlo.

Por último, en la siguiente tabla se recogen todos los posibles códigos de error devueltos por el API y su significado:

Código HTTP	Código JSON	Descripción
401	TOK1	Token de usuario incorrecto.
403	TOK2	Token de usuario caducado, se debe solicitar uno nuevo.
401	TOK3	Falta cabecera de autenticación. No se ha incluido la cabecera o ésta tiene un forma incorrecto.
403	FILE1	Se ha supera el tamaño máximo permitido en la subida de un fichero (50Kb)
401	FILE2	El ID de fichero proporcinado no es correcto (el fichero no existe o el usuario no disponde permisos para acceder a él)
401	FILE3	La cuota máxima de almacenamiento de ficheros ha sido superada (20)
401	USER_ID1	El ID de usuario proporcionado no existe
401	USER_ID2	No se ha encontrado el usuario con los datos proporcionados para la búsqueda con la función `search`.
401	ARGS1	Los argumentos de la petición HTTP son erróneos

Volver a: Prácticas