En clases se mencionaron ciertas consecuencias negativas que conlleva el autentificar y autorizar a usuarios en la Web en base a sus nombres de usuario (o correos electrónicos) y contraseñas.

A. Explique en detalle cuáles son estas consecuencias, y describa casos hipotéticos que evidencien que son negativas.

Las consecuencias negativas varían según cómo se implementa cada autenticación en base a nombre-contraseña. Por lo que se detalla para cada caso:

En primer lugar, si ingreso mi nombre de usuario-contraseña y el servidor **guarda la contraseña en texto plano**, cualquier usuario que pueda leer la base de datos tendrá acceso a la información nombre/contraseña por lo que podría "impersonarme" o hacerse pasar por mi y tener autorización o permisos a las acciones que como usuario logueado debería tener.

Por otra parte, si ingreso mi nombre de usuario-contraseña y el **servidor guarda mi contraseña como un hash de la contraseña original**. También está la posibilidad de que se vulnere la autentificación fácilmente. Un primer caso hipotético es que el usuario use la misma contraseña con dos web distintas, asociadas a dos (o más) servidores distintos alguien que se comunique o tenga acceso a esos dos servidores podría darse cuenta y relacionar fácilmente esta contraseña hasheada al mismo usuario. En relación esto un segundo caso hipotético es el uso de "rainbow tables", que son tablas que almacenan contraseñas frecuentes pre-hasheadas, en este caso el atacante puede hacer uso de esos hashes precomputados y si encuentra una coincidencia hacerse pasar por otro usuario teniendo autorización para las acciones permitidas para ese usuario respectivo.

Un tercer escenario, que consiste en la solución clásica es cuando ingreso mi nombre-contraseña y se las envío al **servidor y este guarda la contraseña como un hash salteado es decir h(pass|| salt)** utilizando distintos salt. Si el salt es muy grande es difícil precomputar, y al ser grandes y aleatorios se lograría resolver el problema de las "rainbow tables" mencionadas en el caso anterior. A partir de esto mismo, hay soluciones más elaboradas como **PBKDF2** en las que de igual manera a partir de la contraseña se genera una llave segura aplicando 2048 o 4096 veces una función de hash.

Pese a que en la práctica el último escenario planteado funciona sigue teniendo la dificultad de que se basa en una confianza "ciega" en el servidor a quien le envío mi información, ya que, confío en que este está haciendo bien su trabajo al guardar la contraseña. Desde esta perspectiva le **entrego mi secreto o clave secreta a un tercero y luego confío**, por lo que sigue existiendo un riesgo de que se vulnere mi autentificación.
Esta posible vulneración de la autentificación al entregar nombre-contraseña a un tercero trae a su vez otras consecuencias. La primera es la **auditabilidad**, ya que cualquiera que tenga acceso a mi nombre- contraseña se ve exactamente igual, no hay una firma y desde esta perspectiva se confía plenamente en el server. Por ejemplo, si el banco (server) hiciera una transferencia usando mi nombre/contraseña se vería igual a una realizada por mí y yo no podría probar que no fue realizada por mi, porque se ve exactamente igual. Este problema está presente incluso en PBKDF2. Y tal como se mencionó anteriormente otro problema en sí mismo es que "**mando mi secreto**" lo que da paso a que cualquiera se vea igual.

Es importante mencionar que estos problemas se producen aunque luego se utiliza un token aleatorio como identificación de sesión para otorgar los permisos, ya que, la vulneración ocurre un paso atrás y es al hacer el login, ya que entregó la clave secreta y por tanto entregó información que permite que un tercero se haga pasar por mi.

B. Diseñe una alternativa para autentificar y autorizar usuarios en la web que no traiga consigo las consecuencias negativas mencionadas. Explique cómo funcionaríla su sistema de autentificación/autorización, y qué problemas prácticos podría traer consigo. Finalmente, explique qué medidas tomaría para prevenir dichos problemas.

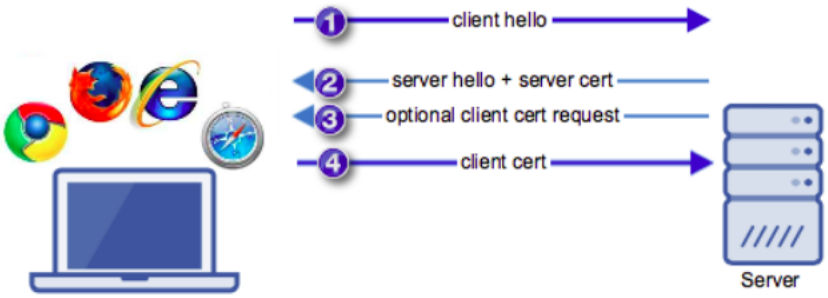
Los problemas planteados surgen a partir de que envío mi nombre y clave secreta al banco/servidor. Por lo tanto, una posible solución se basa en usar una clave privada y una clave pública, donde **el banco solo tenga acceso a la clave pública**.
Este problema soluciona el problema de entregar mi secreto y de la auditabilidad, ya que como solo yo localmente tengo mi llave privada **solo yo podría firmar las acciones** (como post request) realizadas por mi.
Una forma es producir una llave privada y otra pública con una **llave maestra**, y en específico **la llave privada debe guardarse encriptada** en mi computador o servidor. Pudiendo ser desencriptada con la llave maestra para poder **firmar las acciones** a las que estoy autorizada y realizo.
Entonces el sistema podría tener un login basado en la llave pública pero lo importante es que cada vez que realizo una acción esta debe ser firmada con la clave privada. Esto traería dificultades de implementación en al menos 3 aspectos:

(1) Cómo **guardar la llave privada**, que como se explicó anteriormente puede resolverse encriptando con una llave maestra.
(2) El a**lmacenamiento de la llave pública** por parte del otro servidor, ya que esta es la llave que se usará en específico para comunicarse con ese servicio. Este problema puede resolverse encriptando o guardando de manera análoga a como se almacenan las contraseñas en el sistema nombre-contraseña (Con la diferencia que se trata de una llave pública).
(3) La última dificultad es que **es más costoso de realizar, ya que, cada acción debe ser firmada y se debe verificar** esta firma para autorizar la acción. Esta es una consecuencia de diseño que podría no ser ideal, pero no es una limitación como tal para la implementación, sino que podría considerarse como un costo por la mayor seguridad brindada.

En la práctica esta idea se ocupa en los **"Client Certificates"** donde un atacante para poder vulnerar seguridad no le basta con un nombre de usuario + contraseña sino que tendría que tener posesión de algo que solo tiene el cliente, que es su certificado de cliente (secreto o llave privada). Un **certificado digital de cliente** es un archivo, generalmente protegido con una contraseña y cargado en una aplicación de cliente (generalmente como archivos PKCS12 con la extensión .p12, .pfx, .pem). Al comienzo de una sesión SSL o TLS, el servidor pued**e requerir que la aplicación cliente envíe un certificado de cliente para autenticación**. Solo los usuarios que se conecten desde clientes cargados con los certificados correctos pueden conectarse (Este certificado está firmado con la llave privada).

En cuanto a la implementabilidad, según JSCAPE (2021), "los navegadores web populares como Firefox, Chrome, Safari e Internet Explorer pueden admitir fácilmente certificados de cliente. Estos certificados digitales también se pueden cargar en clientes de transferencia de archivos seguros como

AnyClient , así como en otras aplicaciones cliente que admiten protocolos protegidos por SSL / TLS como HTTPS, FTPS, WebDAV y AS2 ”
En cuanto a las dificultades para su uso por parte de los usuarios, se menciona que: (1) Deben instalarse en las máquinas / aplicaciones cliente (Tarea tediosa para los administradores del sistema) y (2) La mayoría de los usuarios finales de los clientes no son técnicos y estas exigencias pueden ser molestas.
La estructura de los certificados digitales de cliente se detalla en la siguiente imagen:



1. First, the client performs a "client hello", wherein it introduces itself to the server and provides a set of security-related information.
2. The server responds with its own "server hello", which is accompanied with its server certificate and pertinent security details based on the information initially sent by the client.
3. This is the optional step that initiates client certificate authentication. This will only be carried out if the server is configured to request a digital certificate from the client for the purpose of authentication.
4. Before this step is performed, the client inspects the server certificate for authenticity. If all goes well, it transmits additional security details and its own client certificate.

Solo después de que cliente y servidor se autentifiquen comienza la transmisión de datos.

Fuentes:
https://www.jscape.com/blog/client-certificate-authentication
https://www.jscape.com/blog/an-overview-of-how-digital-certificates-work