uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

Aplicaciones Web (OpenCourseWare 2023)
Jesús Arias Fisteus

Tema 4: Otros aspectos del desarrollo de aplicaciones Web

1.- Una aplicación ejecuta código JavaScript en el lado del cliente para validar los valores introducidos por los usuarios en un formulario. ¿Debe la aplicación validar esos valores de nuevo en el lado del servidor cuando recibe los datos del formulario?

(a) Solo si la aplicación funciona sobre HTTP en lugar de HTTPS.
(b) No, porque es redundante.
(c) *** Sí, porque los atacantes podrían saltarse la validación de datos en el lado del cliente, incluso aunque la aplicación funcionase sobre HTTPS.

2.- El siguiente fragmento de código Python que construye una consulta a enviar a la base de datos, donde las variables `name` y `password` provienen de un formulario de autenticación:

```
query = f"""
SELECT id, name, fullName, balance FROM Users
WHERE name='{name}'
AND password='{password}'
"""
```

(a) Es seguro.
(b) *** Puede ser explotado para saltarse el proceso de autenticación.
(c) Puede ser explotado para realizar un ataque de *cross-site scripting*.

3.- Las contraseñas de los usuarios deben ser cifradas:

(a) *** Con una función *hash*.
(b) Con un algoritmo de cifrado simétrico.
(c) Con un algoritmo de clave pública.

4.- Una de las razones por las que el *cross-site scripting* almacenado es más peligroso que el *cross-site scripting* reflejado es que:

(a) Es más fácil enviar a las víctimas el enlace maligno que desencadena el ataque.

(b) Los desarrolladores no tienen forma de defender sus aplicaciones contra ataques de *cross-site scripting* almacenado.

(c) *** Las víctimas están normalmente autenticadas en la aplicación cuando sufren el ataque.

5.- La API *XmlHttpRequest*:

(a) No permite enviar datos en los mensajes de petición.

(b) Solo permite enviar los datos de las peticiones y respuestas en formato XML.

(c) *** Permite enviar los datos de las peticiones y respuestas en múltiples formatos, como JSON, XML y texto plano.

6.- Construir una consulta SQL concatenando datos recibidos desde el lado del cliente de una aplicación Web:

(a) Es normalmente seguro si la aplicación se sirve a través de HTTPS.

(b) *** Con frecuencia hace que la aplicación sea vulnerable a ataques de inyección de SQL.

(c) A menudo hace que la aplicación sea vulnerable a ataques de *cross-site scripting*.

7.- Los *tokens* de sesión en aplicaciones Web:

(a) Deben incluir el nombre de usuario o su dirección de correo electrónico para que las aplicaciones sepan quién es el propietario de la sesión solo con inspeccionar el *token*.

(b) Se diseñan de forma que el hecho de que un tercero los lea no suponga ningún problema de seguridad.

(c) *** Deben ser mantenidos en secreto.

8.- Si un fragmento de texto puede ser introducido por un usuario de una aplicación Web, almacenado en la base de datos y mostrado posteriormente a otros usuarios de la aplicación:

(a) No hay riesgo si la aplicación toma precauciones contra ataques de inyección de SQL.

(b) No hay riesgo si el fragmento de texto es siempre enviado a través de TLS.

(c) *** La aplicación debe tomar precauciones con dicho fragmento de texto para evitar ataques de *cross-site scripting*.

9.- Guardar datos en *caché* en una aplicación Web:

(a) Solo puede aplicarse a aplicaciones que no usen base de datos.

(b) *** Ayuda a reducir la carga en la base de datos.

(c) No contribuye a hacer una aplicación Web escalable.

10.- Cuando se usa la API *XMLHttpRequest* de JavaScript:

(a) La página Web se bloquea siempre desde el instante en que se envía la petición HTTP hasta el instante en que se recibe la respuesta HTTP. No reacciona a las acciones del usuario durante dicho periodo.

(b) *** Los datos se representan frecuentemente en formato JSON.

(c) Los datos deben ser siempre representados en formato XML.

11.- La forma más segura de entre las siguientes de cifrar una contraseña de usuario en la base de datos de una aplicación web es:

(a) *** Aplicar una función de *hash* que preferiblemente suponga un tiempo de cómputo relativamente alto a una mezcla de la contraseña y un valor aleatorio (`salt`) generado en el momento.

(b) Aplicar a la contraseña una función de *hash* que suponga un tiempo de cómputo tan pequeño como sea posible.

(c) Aplicar un algoritmo de cifrado simétrico a la contraseña. La clave simétrica empleada para cifrar debe residir en un lugar seguro dentro del servidor.

12.- Un ejemplo de buen *token* de sesión es:

(a) El identificador del usuario al cual pertenece la sesión combinado con la fecha y hora en que dicha sesión se ha creado.

(b) Un número secuencial que se incremente para cada nueva sesión que se cree.

(c) *** Un número aleatorio grande e impredecible.

13.- Un motivo importante por el que se prefiere el envío de peticiones *XMLHttpRequest* asíncronas es:

(a) Que la programación mediante peticiones síncronas es considerablemente más compleja en el lado del navegador.

(b) Que la programación mediante peticiones síncronas es considerablemente más compleja en el lado del servidor.

(c) *** Que si la petición fuese síncrona la pestaña se bloquearía y dejaría de responder hasta que llegase y hubiese sido procesado el mensaje de respuesta.

14.- Se dice que una aplicación web es escalable si:

(a) *** Es capaz de dar servicio a una carga de trabajo creciente o puede ser ampliada para ello.

(b) Es conforme a los estándares más recientes de HTML, CSS y JavaScript.

(c) Se programa en un lenguaje de programación eficiente, como es el caso de Java o C++.

15.- ¿Por qué es buena práctica cifrar contraseñas en la base de datos con un valor de *salt* distinto para cada usuario?

(a) Porque es mucho más eficiente en términos de uso de memoria que utilizar el mismo valor de *salt*.

(b) *** Para que los atacantes necesiten probar cada contraseña una vez por cada usuario al que deseen atacar.

(c) Porque usar el mismo valor de *salt* facilita los ataques de *cross-site scripting*.

16.- Una aplicación web que utilice HTTPS:

(a) No necesita que se guarden las contraseñas cifradas en la base de datos, dado que será imposible para un atacante acceder a dicha base de datos.

(b) Es totalmente segura siempre que se use HTTPS en todas las comunicaciones entre cliente y servidor.

(c) *** Evita o dificulta algunos tipos de ataques, pero podría contener pese a ello numerosas vulnerabilidades.

17.- Una aplicación de tienda Web en la cual se emplea un formulario como el siguiente para que el cliente añada el producto mostrado en la página actual a su carro de la compra:

```
<form action="add-to-cart" method="post">
  <input type="hidden" name="product-id" value="76af5662">
  <input type="hidden" name="price" value="699.95">
  <input type="submit" value="Añadir al carro">
</form>
```

(a) Presenta un problema de seguridad: no es seguro recoger en el servidor los valores de `product-id` ni `price`. Deben eliminarse ambos controles del formulario.

(b) Es segura si en el servidor se utiliza `PreparedStatement` para acceder a la base de datos con los datos recibidos desde el formulario.

(c) *** Presenta un problema de seguridad: no es seguro recoger en el servidor el valor de `price`. Debe eliminarse este control del formulario.

18.- El siguiente fragmento de código JavaScript, que usa JQuery:

```
$.get("userProfile", {id: 776})
  .done(function(data) {
    $("body")
      .append("Name: " + data.name)
      .append("Age: " + data.age);
  });
```

(a) Lee las propiedades `name` y `age` de la variable `data`, y envía sus valores al servidor en una petición con método GET.

(b) Registra un manejador de eventos para ser ejecutado no antes de que se haya cargado por completo el elemento con clase `userProfile` e identificador 776.

(c) *** Envía una petición con método GET al servidor pasando un parámetro llamado `id`, y actualiza el contenido de la página con los datos de la respuesta.

(d) Actualiza el contenido de la página con los datos obtenidos del contenido del elemento con clase `userProfile` e identificador 776.

19.- Indica cuál de las siguientes afirmaciones es correcta:

(a) *** El balanceo de carga mediante servidor de *front-end* (o *proxy* inverso) permite ajustar dinámicamente el reparto de peticiones a los distintos servidores dependiendo del estado actual de carga de cada uno de ellos.

(b) Replicar los gestores de bases de datos con el paradigma maestro–esclavo es efectivo cuando la aplicación realiza mayoritariamente operaciones de escritura, pero resulta poco efectivo o incluso contraproducente cuando la mayoría de las operaciones son de lectura.

(c) El balanceo de carga mediante DNS permite ajustar dinámicamente el reparto de peticiones a los distintos servidores dependiendo del estado actual de carga de cada uno de ellos.

(d) *Memcached* es un ejemplo de sistema de balanceo de carga mediante servidor de *front-end* (o *proxy* inverso).

20.- Una aplicación *web* gestiona las sesiones de usuario mediante un *token* de sesión almacenado en una *cookie*. Si la autenticación se realiza mediante HTTPS y el resto de peticiones a la aplicación se transmiten sobre HTTP:

(a) Un atacante que tenga acceso al tráfico intercambiado en una sesión de usuario podrá ver el *token* de sesión y, con él, suplantar al usuario iniciando nuevas sesiones hasta que este cambie su contraseña.

(b) *** Un atacante que tenga acceso al tráfico intercambiado en una sesión de usuario podrá ver el *token* de sesión y, con él, suplantar al usuario hasta que se cierre su sesión.

(c) Un atacante que tenga acceso al tráfico intercambiado en una sesión de usuario podrá ver su contraseña y, con ella, suplantar al usuario iniciando nuevas sesiones hasta que este cambie su contraseña.

(d) Un atacante que tenga acceso al tráfico intercambiado en una sesión de usuario no podrá pese a ello suplantarlo, dado que solo es necesario transmitir el *token* durante el proceso de autenticación, el cual se lleva a cabo sobre HTTPS.

21.- La capacidad de un atacante para ejecutar código JavaScript arbitrario en el navegador de otros usuarios de una aplicación *web*:

(a) *** Es lo que se pretende conseguir en un ataque de *cross–site scripting*.

(b) No supone ningún peligro por las estrictas restricciones de seguridad que los navegadores imponen al código JavaScript que ejecutan.

(c) No supone ningún problema si la aplicación utiliza HTTPS en todas sus comunicaciones.

(d) Es lo que se pretende conseguir en un ataque de inyección de SQL reflejado horizontal.