GTO
Grandeza de México

Secretaría de Educación de Guanajuato

GUANAJUATO 200 AÑOS DE GRANDEZA

2024
200 AÑOS DE GRANDEZA
GUANAJUATO COMO ENTIDAD FEDERATIVA, LIBRE Y SOBERANA

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

UTyP
DIRECCIÓN GENERAL DE UNIVERSIDADES TECNOLÓGICAS y POLITÉCNICAS

Universidad Tecnológica del Norte de Guanajuato
Organismo Público Descentralizado del Gobierno del Estado
"Educación y progreso para la vida"

29 ANIVERSARIO
UTNG
Educación y progreso para la vida

Norma Mexicana en Igualdad Laboral y No Discriminación

ASIGNATURA:

SEGURIDAD EN EL DESARROLLO DE APLICACIONES

PROFESOR: SERGIO HUMBERTO VAZQUEZ BARRIENTOS
sergiovazquez@utng.edu.mx

Importante antes de iniciar …

examen de diagnostico

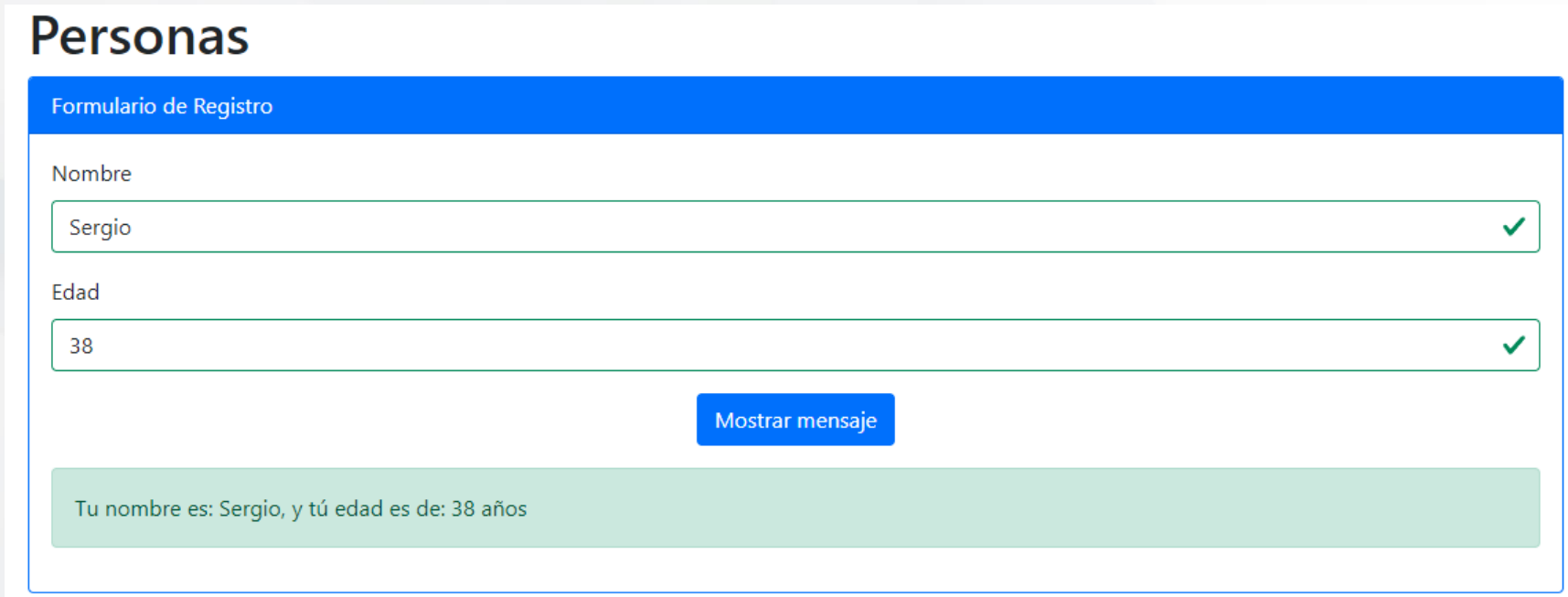
Comencemos a conocer la asignatura…

SEGURIDAD EN EL DESARROLLO DE APLICACIONES

Objetivo de Aprendizaje:

El alumnado integrará mecanismos de seguridad en el desarrollo de aplicaciones de software mediante el uso de técnicas, herramientas y pruebas para proteger los datos de los usuarios y las organizaciones.

Ejercicio


Personas
Formulario de Registro
Nombre
Sergio
Edad
38
Mostrar mensaje
Tu nombre es: Sergio, y tú edad es de: 38 años

UNIDADES DE LA ASIGNATURA:

Unidades de Aprendizaje
I. Principios de codificación segura
II. Aplicaciones seguras

¿Que es la seguridad?

Cualidad de estar protegido y libre de riesgos, de peligro o de daños.

¿Que es la seguridad en el desarrollo de aplicaciones?

El desarrollo seguro es el conjunto de técnicas, normas y conocimientos que permiten crear programas que no puedan ser modificados de forma ilegítima con fines maliciosos y que estén carentes de fallos que puedan comprometer la seguridad del resto de elementos del sistema con el que interactúan

¿Que es la seguridad en el desarrollo de aplicaciones?

El desarrollo seguro es la práctica de escribir código, en cualquier lenguaje de programación, para prevenir problemas de seguridad en la app o sitio web. El objetivo es **minimizar vulnerabilidades**.

Cualquier tarea de programación se puede llevar a cabo de diferentes maneras, con distintos niveles de complejidad. Algunas soluciones son más seguras que otras. Los estándares o buenas prácticas de código seguro guían a los desarrolladores para que elijan el camino más seguro, aunque no sea el camino más rápido o eficiente.

Ejemplo

Un ejemplo sencillo de entender es la recomendada **denegación por defecto** a la hora de permitir accesos. Los desarrolladores que utilizan técnicas de desarrollo seguro crean código que impide el acceso a recursos sensibles, a menos que el usuario pueda demostrar que tiene autorización. El trabajo extra de dar privilegios a los usuarios de forma personalizada —por ejemplo, según si eres usuario registrado o no, o según el estatus dentro de la organización— es necesario. Solo así garantizamos que la persona que accede a un dato tiene permiso para hacerlo en razón de su responsabilidad en la empresa.

"Según OWASP, el objetivo del desarrollo seguro es «mantener la confidencialidad, integridad y disponibilidad de los recursos de información con el fin de facilitar el éxito de las operaciones de las empresas»"

TEMAS DE LA UNIDAD I

Temas
1. Buenas prácticas en el desarrollo de software seguro.
2. Protección de vulnerabilidades.

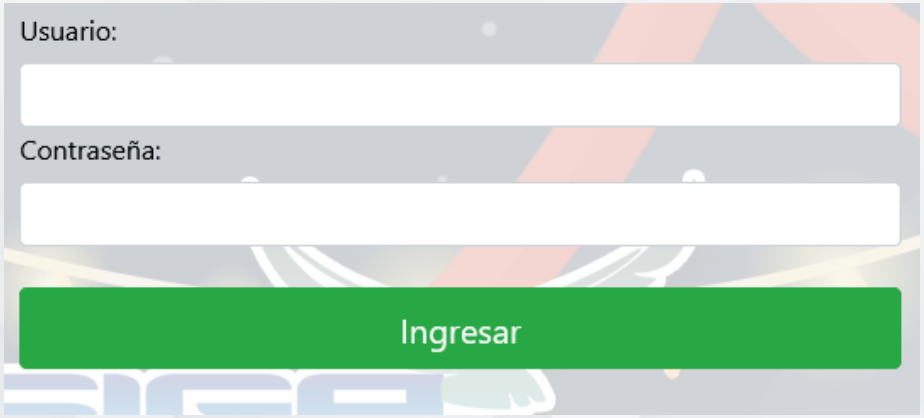
Tema 1: Buenas prácticas en el desarrollo de software seguro.

Buenas prácticas en el desarrollo de software seguro

Las buenas prácticas para un desarrollo seguro evitan vulnerabilidades y previenen ataques. Para esto revisaremos las buenas prácticas de codificación segura establecidas por OWASP (Open Web Application Security Project), una entidad internacional sin ánimo de lucro que trabaja para mejorar la seguridad del software.

Ejemplo

¿Cuantos criterios de seguridad se deberían utilizar?



Ejemplo

¿Cuantos criterios de seguridad se deberían utilizar?

39

¿Por qué?

Parámetro usuario, validación para los nombres de usuario predeterminados comunes, la enumeración de la cuenta, el forzamiento bruto, la inyección LDAP y SQL, y XSS.
Del mismo modo el parámetro de contraseña debe incluir contraseñas comunes, longitud de contraseña, inyección de bytes nulos, eliminación del parámetro, XSS y más.

Buenas prácticas en el desarrollo de software seguro

Los puntos débiles de cualquier aplicación en términos de seguridad tienen que ver con la forma en que se ha escrito su código. Así que, durante todo el proceso de desarrollo, desde que se comienza el proyecto hasta que se entrega la app al cliente, los programadores deben seguir determinadas pautas para garantizar la máxima seguridad del código.

Cuando los desarrolladores empiezan a construir una nueva aplicación, deben tener en cuenta los requisitos del cliente y deben hacer un código eficiente y seguro.

¿Por qué son importantes las buenas prácticas para un código seguro?

Los incidentes de seguridad pueden tener graves consecuencias para negocios y personas. Un código creado sin seguir buenas prácticas de seguridad puede provocar daños económicos y robos de identidad, entre otros perjuicios. En sectores específicos, como finanzas, salud, energía y transporte, los efectos negativos pueden ser mayores.

En los últimos años hemos visto que incluso grandes organizaciones sufren graves filtraciones de datos, a pesar de haber podido dedicar recursos para minimizar las vulnerabilidades.

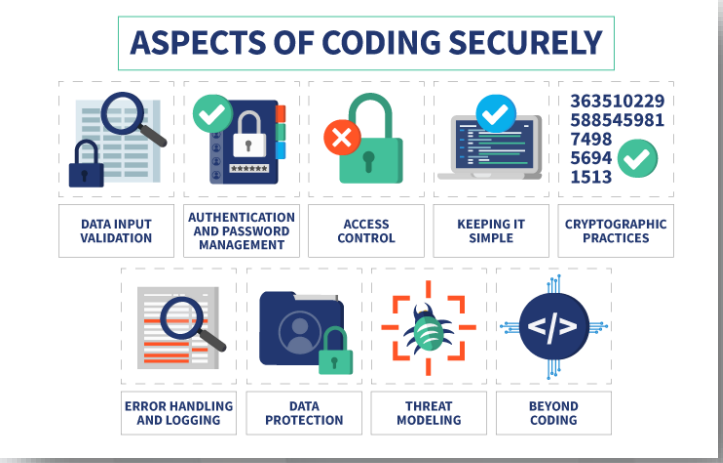
¿Por qué son importantes las buenas prácticas para un código seguro?

Las compañías que se relacionan con sus clientes a través de webs y aplicaciones móviles pueden perder la confianza de estos por problemas de seguridad. Una de las peores cosas que pueden pasar es poner en peligro a los clientes. Por este motivo, consolidar buenas prácticas de desarrollo es una prioridad.

Desde un punto de vista tecnológico, el daño que se causa a una web o app puede afectar a otros sistemas, como por ejemplo la base de datos del backend o el sistema del usuario.

Buenas prácticas en el desarrollo seguro

Cuando se trata de hacer desarrollos, no se puede reinventar la rueda. Así, los expertos han alcanzado consensos sobre qué prácticas garantizan un desarrollo seguro.



Buenas prácticas en el desarrollo seguro

Recomendaciones de seguridad para los siguientes aspectos:

1. Arquitectura
2. Autorización
3. Autenticación
4. Gestión de sesiones
5. Validación de parámetros de entrada y salida
6. Gestión de errores
7. Registro seguro
8. Criptografía
9. Gestión segura de archivos
10. Seguridad en las transacciones
11. Seguridad en las comunicaciones
12. Protección de datos

Arquitectura

Una arquitectura segura sólida es fundamental para la construcción del software pues es la base donde se sustenta y se desarrolla el software.

Para conseguir este propósito hay que identificar los componentes utilizados, asegurarse de que no existen vulnerabilidades conocidas y que están correctamente actualizados.

La arquitectura siempre debe estar diseñada teniendo presente los requisitos de seguridad para evitar o limitar las potenciales amenazas de seguridad.

Riesgos potenciales

Las amenazas potenciales de una aplicación pueden ser múltiples y de alta probabilidad cuando:

1. Se utilizan componentes con vulnerabilidades conocidas.
2. Se utilizan componentes anticuados, obsoletos o no soportados.
3. Se utilizan componentes que no han sido identificados.
4. Se mantienen abiertos puertos prescindibles.

Recomendaciones de seguridad

1. Identificar todos los componentes de la arquitectura.

2. Para aquellas vulnerabilidades que supongan un riesgo real, mantener una estrecha vigilancia sobre los componentes vulnerables con el fin de que sean actualizados lo antes posible.

3. Realizar un estudio de cómo podrían evitarse o mitigarse los problemas de seguridad que crearían estas vulnerabilidades de riesgo mediante sistemas alternativos de seguridad.

4. Usar la versión más reciente del lenguaje de programación.

5. Desactivar todas las opciones de depuración en Producción que evite fuga de información en los mensajes de error detallados.

6. Utilizar herramientas en el IDE que realicen análisis semánticos y de seguridad básicos.

Autenticación

La autenticación es la verificación de la identidad de un usuario o dispositivo con el fin de otorgar acceso a sus recursos o información. Esta tarea suele requerir la presentación de credenciales, como un nombre de usuario y contraseña, para comprobar que el usuario es efectivamente quien dice ser.

Tipos de autenticación

- **Autenticación de red**: se verifica la identidad de un usuario o dispositivo mediante el uso de credenciales de red, como un nombre de usuario y una contraseña, o mediante la verificación de un certificado digital.

- **Autenticación basada en contraseñas**: el usuario proporciona un nombre de usuario y una contraseña para acceder a un sistema o recurso.

Tipos de autenticación

- **Autenticación basada en tokens**: el usuario recibe un token físico o digital que debe proporcionar para acceder a un sistema o recurso.

- **Autenticación de dos factores**: se requiere que el usuario proporcione dos (2) formas de verificación de su identidad, como una contraseña y un código enviado a su teléfono móvil.

- **Autenticación biométrica**: se utilizan características físicas del usuario, como su huella dactilar o su voz, para verificar su identidad.

Riesgos potenciales

Las amenazas y riesgos más comunes por falta de controles de autenticación de seguridad en las aplicaciones o por utilizar métodos de autenticación insuficientemente seguros son:

- **Ataques de fuerza bruta**: utilizando programas que generan y prueban combinaciones de caracteres para adivinar las contraseñas de los usuarios y acceder a sus cuentas. Si las contraseñas son débiles o fáciles de adivinar, estos ataques podrían tener éxito.

- **Ataques de diccionario**: utilizando programas que intentan adivinar contraseñas mediante el uso de combinaciones comunes de letras y números. Si las contraseñas son débiles o fáciles de adivinar, estos ataques podrían tener éxito.

Riesgos potenciales

- **Ataques de MitM**: si se intervienen las comunicaciones se podrían obtener las credenciales, el token o el ticket de acceso (ataque de “replay”). Si, además, las comunicaciones son inseguras o se utilizan formas de autenticación de texto en claro, los ataques podrían tener más éxito.

- **Suplantación de identidad**: obteniendo las credenciales suplantando la identidad del servidor de autenticación, haciendo creer a la víctima que se encuentra en el sitio correcto donde indicar sus credenciales para tener acceso. El servicio podría incluso reenviar la información al servidor real y redirigirle autenticado para que la víctima no percibiese el engaño.

- **Enumeración de usuarios**: el atacante podría encontrar usuarios registrados en el sistema probando identidades y analizando las respuestas del servidor, ya sea por el tiempo de respuesta o por los mensajes de error.

Recomendaciones de seguridad

- Garantizar que las contraseñas no se almacenan en un formato legible, de modo que, si el sistema o el recurso que contiene las contraseñas se ve comprometido, el usuario malintencionado sigue sin poder utilizarlas. Un buen método podría ser el uso de funciones que garanticen la irreversibilidad de la operación, como el uso de funciones hash fuertes.

- Hay que asegurar que cada página de la aplicación tiene un enlace de desconexión, que la sesión expira cuando el usuario se desconecta y que la sesión expira cuando pasa un tiempo prudente de uso sin actividad.

- Nunca exponer las credenciales en la URL.

- Al usar formularios, utilizar métodos POST para el envío de información entre el cliente y el servidor.

Recomendaciones de seguridad

- Utilizar la autenticación multi-factor para implementar múltiples capas de seguridad para aplicaciones sensibles.

- Desactivar el atributo “autocompletar” del campo de contraseña en la aplicación, ya que está activado por defecto.

- Hacer cumplir los requisitos de complejidad de las contraseñas establecidos por las políticas o la normativa, y asegurarse que estos requisitos siguen unas reglas mínimas de seguridad como:
  - Debe tener un mínimo de ocho (8) caracteres y un máximo prudente.
  - Debe contener, al menos, tres (3) de los siguientes caracteres:
    - Una letra mayúscula.
    - Una letra minúscula.
    - Un número.
    - Un carácter especial.

Ejemplo

Este ejemplo crea un hash y salt para una contraseña llamando a get-SaltedHash (String password). Este método devolverá una cadena que contiene el salt y el hash separado por un | (pipe).

Para verificar si una contraseña dada es correcta, llama a check (String password, String stored), pasando la contraseña que se está verificando junto con el hash/salt almacenado. Este método devolverá true si la contraseña es correcta.

Ver Código

Autorización

La autorización se encarga de determinar qué acciones le está permitido realizar a un usuario o sistema, por lo tanto, la autenticación es un requisito previo para la autorización, ya que es necesario verificar la identidad de un usuario o sistema antes de determinar qué acciones les está permitido realizar.

Es el segundo control de seguridad después de la autenticación y está muy vinculado a la autorización sobre las funciones de negocio de la aplicación.

Riesgos potenciales

Son varias las vulnerabilidades que pueden aparecer en ausencia de controles de seguridad adecuados a la hora de implementar la autorización.

- **Acceso no autorizado**: si la autorización no está implementada de manera adecuada, puede dar lugar a que usuarios no autorizados tengan acceso a recursos y operaciones a los que no debería tener permiso.

- **Escalada vertical de privilegios**: un usuario puede acceder a la funcionalidad de otro usuario con mayores privilegios.

- **Escalada horizontal de privilegios**: un usuario puede acceder a la funcionalidad de otro usuario con el mismo nivel de acceso.

- **Revelación de información sensible**: una aplicación da más información de la necesaria al usuario y podría ser utilizada por un atacante con fines maliciosos.

Riesgos potenciales

- **Violación de la privacidad**: si el atacante acaba teniendo acceso a datos relativos a la privacidad de otros usuarios.
- **Robo de identidad**: es cuando el atacante obtiene la identidad de la víctima y la utiliza para realizar algunas acciones.
- **Robo de datos**: es cuando un atacante obtiene ilegítimamente datos, ya sean confidenciales o no.
- **Disponibilidad del servicio**: un acceso no autorizado a elementos críticos de la aplicación podría permitir a un atacante interrumpir o dañar el servicio.
- **Manipulación de datos**: un atacante es capaz de interceptar y manipular los datos intercambiados entre el cliente y el servidor.

Recomendaciones de seguridad

- Garantizar la implementación del principio de mínimo privilegio: los usuarios tienen acceso restringido sólo a las funciones y datos que realmente necesitan para realizar su trabajo con normalidad.

- Asignar los permisos y privilegios a roles de aplicación, nunca directamente a los usuarios. Los usuarios lo que deben tener son roles y sus privilegios son tomados de éstos.

- Garantizar que las reglas de control de acceso se aplican en el lado del servidor.

- Registrar todas las operaciones sobre datos sensibles en un registro específico de seguridad donde conste, al menos: la fecha/ hora de la operación, la operación (lectura, creación, borrado, actualización), el nombre del dato, el proceso, función o servicio que generó la operación, el usuario, el rol del usuario que tiene el privilegio para la operación, el resultado de la operación (si fue exitosa o no) y un mensaje opcional sobre el resultado de la operación (en caso de error).

Ejemplo

Utilizando el framework de seguridad Spring Security que proporciona un conjunto completo de características de seguridad, incluyendo autenticación y autorización basada en roles. Se utilizan anotaciones para controlar el acceso a ciertas partes de tu aplicación.

En el siguiente ejemplo, el método listUsers sólo será accesible para usuarios que tengan el rol ROLE_ADMIN

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
@GetMapping("/admin/users")
public String listUsers(Model model) {
// Código para listar a los usuarios
}
```

Herramientas a utilizar

- Java Jdk 17
- Spring Tools
- Mysql o MariaDB
- Gestor de Base de datos(HeidiSQL, MysqlWorkbench)
- Postman

Gestión de sesiones

Una sesión es una conexión activa entre un usuario y el sistema. La gestión de sesiones consiste determinar acciones sobre las mismas que sirvan para garantizar la seguridad de la autorización, integridad y privacidad de la información entre el usuario y el sistema. Cada acceso autenticado y autorizado en el sistema crea una nueva sesión en éste que permite almacenar información temporal acerca de las operaciones y la lógica de negocio exclusivas del usuario en la aplicación durante el tiempo que dure su acceso. Cada sesión queda identificada por un identificador único.

Tipos de Gestión

Mediante cookies: es el mecanismo más común e inseguro. Se genera un identificador único en la cookie del navegador que es utilizado para identificar la sesión del usuario en el servidor como una sesión activa y autenticada. Es más inseguro porque requiere delegar la responsabilidad de la persistencia de las cookies a las implementaciones más o menos seguras de los navegadores.

Mediante tokens: es equivalente al sistema anterior donde el token es un identificador de acceso que permite acceder al sistema al haberse asociado dicho token con la ID de la sesión correspondiente. Este token suele viajar en las cabeceras de la petición. Es más seguro porque permite modificar los tokens de acceso cada cierto tiempo, mediante un intercambio de tokens, sin necesidad de modificar el ID de la sesión que se mantendría siempre a salvo en el lado servidor.

Riesgos potenciales

- **Predicción de sesión**: se centra en la predicción de valores de ID de sesión que permiten a un atacante evitar el esquema de autenticación de una aplicación.

- **Secuestro de sesión**: el ataque de secuestro de sesión consiste en explotar el mecanismo de control de la sesión web, normalmente gestionado por un token de sesión.

- **Fijación de sesión**: consiste en obtener un ID de sesión válido (por ejemplo, estableciendo una conexión con la aplicación), inducir a un usuario a autenticarse con ese ID de sesión y luego secuestrar la sesión validada por el usuario para conocer el ID de sesión utilizado.

- **Suplantación de sesión**: cuando el atacante gana la identidad de otra entidad para cometer algún tipo de fraude. Por ejemplo, un atacante que genera un sitio web malicioso con la apariencia de un banco legítimo para engañar a sus víctimas mediante el phishing.

Recomendaciones de seguridad

- Es crucial asegurarse de que el ID de sesión nunca se exponga en el tráfico no cifrado.
- Implementar cabeceras seguras con directivas como cachecontrol o strict-transport security.
- Compruebe que las sesiones se invalidan cuando el usuario cierra la sesión.
- Las sesiones deben expirar después de un tiempo de inactividad específico.
- Garantizar que todas las páginas que requieren autenticación tienen un acceso fácil y amigable a la funcionalidad de cierre de sesión.

Recomendaciones de seguridad

- Verificar que el ID de la sesión no aparezca nunca en las URL, en los mensajes de error o en los registros.
- Garantizar que toda autenticación y re-autenticación exitosa genera una nueva sesión y un nuevo ID de sesión, destruyendo el anterior.
- Compruebe que el ID de sesión almacenado en las cookies se define utilizando los atributos HttpOnly y Secure.
- Comprobar que la aplicación realiza un seguimiento de todas las sesiones activas y permite al usuario finalizar las sesiones de forma selectiva o global desde su cuenta

Ejemplo

En Java, la gestión de sesiones se puede realizar mediante el uso de la interface javax.servlet.http.HttpSession. Esta interface proporciona métodos para almacenar y recuperar atributos de sesión, establecer y obtener el tiempo de expiración de la sesión e invalidar la sesión.

```
HttpSession session = request.getSession();
session.setAttribute(“user”, “John”);
String user = (String) session.getAttribute("user");
session.setMaxInactiveInterval(3600);
session.invalidate();
```

Ejemplo

Es importante tener en cuenta que, para garantizar la seguridad de la gestión de sesiones, se deben utilizar cookies HTTPS seguras y generar identificadores de sesión únicos e impredecibles. También hay que asegurarse de validar la solicitud y el estado de la sesión en cada solicitud para evitar la suplantación de sesión.

```
SecureRandom random = new SecureRandom();
byte[] bytes = new byte[32];
random.nextBytes(bytes);
MessageDigest digest = MessageDigest.getInstance("SH A-256");
byte[] hashedSessionId = digest.digest(bytes);
String sessionId = Base64.getEncoder().encodeToString(hashedSessionId);

Cookie sessionCookie = new Cookie("SESSIONID", sessionId);
sessionCookie.setHttpOnly(true);
sessionCookie.setSecure(true);
sessionCookie.setMaxAge(3600); // expira en 1 hora
response.addCookie(sessionCookie);
```

Validación de parámetros de entrada y salida

Es una de las más importantes áreas críticas de seguridad en las aplicaciones. La mayoría de las vulnerabilidades de las aplicaciones surgen de una incorrecta o insuficiente validación de los datos de entrada o salida que es aprovechada para realizar ataques como Cross-site scripting, inyecciones en general, exposición de datos sensibles o DoS.

Técnicas de validación

Sanitización: Es un proceso para convertir los datos que tienen más de una representación posible en un formato estándar, canónico o normalizado, reduciendo la entrada/salida a una única forma fija convertida y/o reducida. Esta técnica por sí sola ya evita muchos problemas de validación y muchos tipos de ataques.

1. **Sanitización de rutas**: Todas las rutas de archivos o directorios son normalizadas
2. **Sanitización de espacios**: Todos los parámetros de entrada eliminan los espacios, caracteres de tabulación u otros caracteres blancos (160) por delante y por detrás del dato
3. **Sanitización de charset**: Se verifica que todos los caracteres introducidos en el dato corresponden al charset esperado
4. **Sanitización de case**: Todos los caracteres son transformados a mayúsculas o minúsculas, según lo requiera el parámetro definido

Técnicas de validación

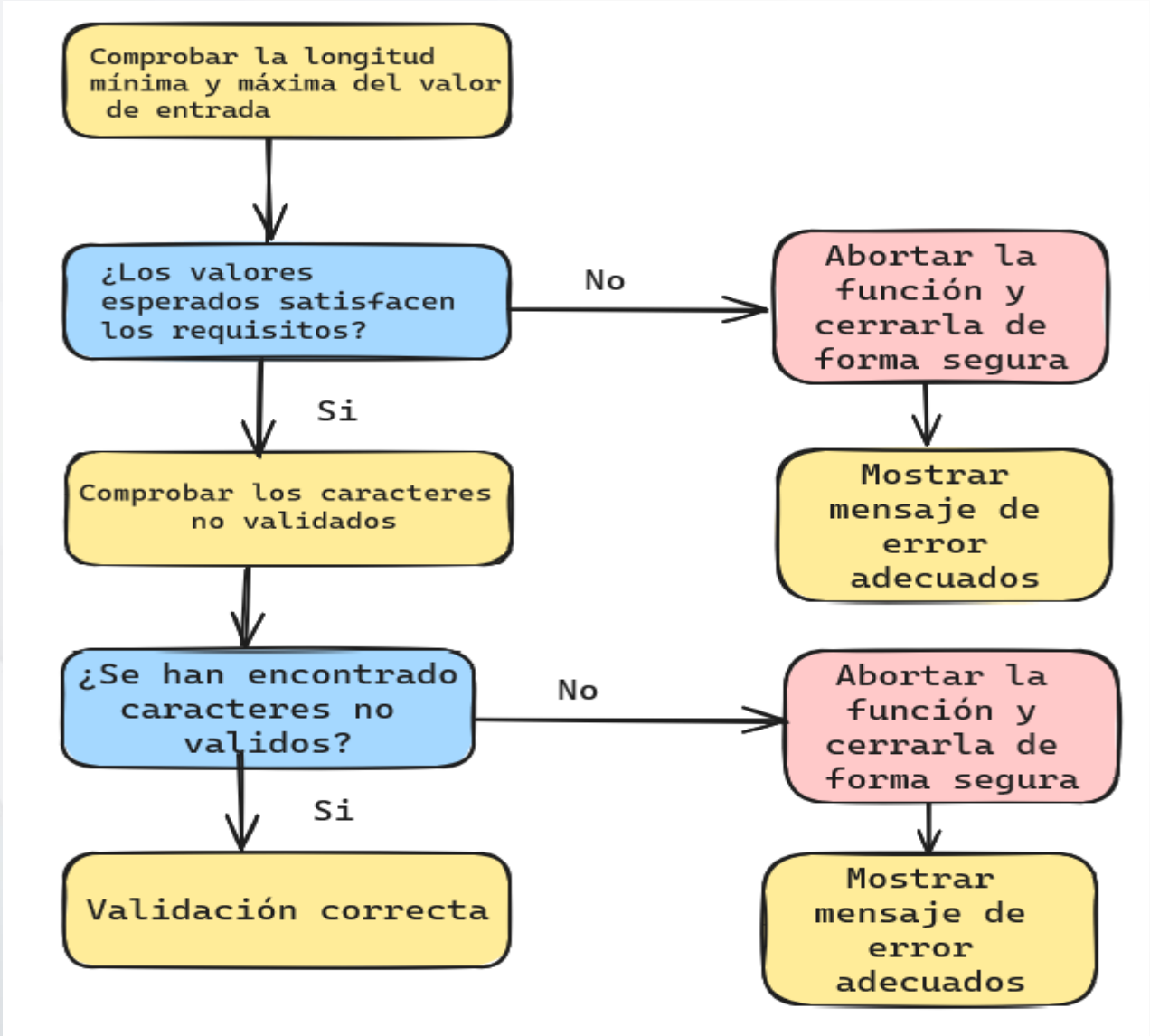
Tipo de datos: Valida que el dato recibido es del tipo esperado con el formato adecuado. Los tipos de datos pueden ser simples o complejos en función de las definiciones que se hayan implementado en la aplicación. Los tipos de datos simples serían, por ejemplo: entero, decimal, booleano, cadena de caracteres, y los complejos, e-mail, NIF, dirección, nombre, CP, fecha, hora, URL, etc.

- Formato
- Tamaños mínimos y máximos
- Valores mínimos y máximos
- Lista blanca
- Lista negra

Diagrama de flujo

El siguiente diagrama es un ejemplo para ilustrar el flujo de trabajo para validar valores de texto libre. Incluye cuatro 4 valores de entrada que se comprueban con algunos requisitos sencillos. Cuando un valor no satisface ningún requisito, los datos de entrada deben ser rechazados

Ejercicio


Comprobar la longitud mínima y máxima del valor de entrada
¿Los valores esperados satisfacen los requisitos?
No
Abortar la función y cerrarla de forma segura
Si
Mostrar mensaje de error adecuados
Comprobar los caracteres no validados
¿Se han encontrado caracteres no validos?
No
Abortar la función y cerrarla de forma segura
Si
Mostrar mensaje de error adecuados
Validación correcta

Riesgos potenciales

La validación de datos es la fuente de muchas vulnerabilidades que un atacante puede explotar. A continuación, se detallan algunas de las vulnerabilidades relacionadas con la insuficiencia o falta de validación de entradas:

- **Cross-Site Scripting**: es un tipo de ataque que permite a un usuario malicioso inyectar código en el navegador web de las víctimas.

- **Inyección SQL**: aprovecha los fallos de programación de la aplicación a nivel de validación de entradas para realizar operaciones sobre una base de datos de forma ilegítima.

- **Inyección LDAP**: consiste en inyectar consultas LDAP arbitrarias para acceder a datos prohibidos o incluso obtener privilegios adicionales.

Riesgos potenciales

- **Inyección de Log**: consiste en inyectar comandos de ejecución en el sistema o cualquier otro tipo de problema utilizando el sistema de registros que registran datos a partir de información parámetros de entrada.

- **Inyección XEE**: una inyección XPATH consiste en la inyección de código XML arbitrario con la intención de acceder a datos a los que no se debe acceder o para obtener información sobre la estructura del árbol XML.

- **Bomba XML**: este ataque trata de sobrecargar el XML excediendo los recursos de memoria de una aplicación para provocar una denegación de servicio.

- **DoS, DDoS o ReDoS**: ataques de denegación de servicio por error del sistema al procesar entradas sin validar que causan fallas en el sistema.

Recomendaciones de seguridad

- Las validaciones siempre deben realizarse en el lado del servidor. Las validaciones del lado del cliente también pueden ser útiles, pero son sólo recomendadas.

- Utilizar esquemas de validación de datos.

- En su defecto, utilizar mecanismos de validación de entrada estándar proporcionados por las bibliotecas específicas de la tecnología (Spring Validator, etc.)

- Cubrir completamente la validación de los datos mediante esquemas de validación o mecanismos estándar que aseguren la entrada de datos mediante: sanitizaciones, tipo de dato, formato, longitudes, valores, listas blancas, listas negras, etc.

Recomendaciones de seguridad

- Verificar que los datos no estructurados se sanean para imponer medidas de seguridad genéricas, como los caracteres permitidos y la longitud, y evitar los caracteres potencialmente dañinos.

- Asegurarse de que toda la entrada no fiable se sanea adecuadamente utilizando una biblioteca de sanitización.

- Evitar mostrar información sensible como consecuencia de un error de validación de algún parámetro recibido.

- Verificar que los errores de validación de entrada en el lado del servidor den como resultado el rechazo de la solicitud.

- Asegurarse de que todas las consultas a la base de datos están protegidas utilizando consultas parametrizadas para evitar la inyección SQL.

Ejemplo

En Java Spring, se utilizan anotaciones para validar los parámetros de entrada a los métodos y los valores permitidos en los miembros de una clase:

```
@PostMapping("/users")
public ResponseEntity<User> createUser(@Valid @RequestBody User user) {
...
return ResponseEntity.ok(user);
}
public class User {
@NotBlank
private String name;
@Email
private String email;
...
```

Gestión de errores

La gestión de errores trata sobre cómo evitar mostrar relevante o sensible a los usuarios que pudieran ser utilizados para lanzar otros tipos de ataques sofisticados contra la aplicación y sobre cómo manejar los errores no controlados dentro de la aplicación para ofrecer salidas seguras que no permitan situaciones inesperadas que puedas ser aprovechadas por usuarios maliciosos.

Confidencialidad de los mensajes

Las aplicaciones si no han implementado una correcta gestión de errores, podrían revelar, de forma involuntaria, información sobre su configuración, su estado interno, mensajes de depuración, datos sensibles, o incluso violar la privacidad.

Otras formas de revelar información son, por ejemplo, el tiempo que tardan en procesar determinadas operaciones, u ofreciendo códigos diferentes a distintas entradas.

Toda esta información podría aprovecharse para lanzar, o incluso automatizar, ataques muy potentes, lo que hace imprescindible una buena gestión de errores.

Riesgos Potenciales

Los siguientes son los posibles riesgos más comunes si el manejo de errores no se implementa adecuadamente dentro de la aplicación.

- **Fuga de información sensible**: versión del servidor, motor de la base de datos, documentos sensibles, estructura de archivos, etc.
- **Denegación del servicio**: cuando los errores forzados mediante abuso pueden causar caída del sistema.
- **Cross-site scripting**: cuando los mensajes de error muestran parámetros de entrada que no han sido correctamente escapados.

Errores no controlados

Cuando no se han contemplado aquellas situaciones donde podría producirse excepciones o errores en puntos de la aplicación, y se dan, la aplicación causaría una salida inesperada de la lógica de negocio que podría dejarla en un estado vulnerable a las siguientes actividades del usuario.

Por tanto, es imprescindible que todas las operaciones estén perfectamente analizadas en cuanto a las distintas excepciones que pudieran producirse y tratar estas salidas excepcionales de forma adecuada.

Riesgos potenciales

Los siguientes son los posibles riesgos más comunes si el manejo de errores no se implementa adecuadamente dentro de la aplicación.

- **Denegación del servicio**: cuando los errores forzados mediante abuso pueden causar caída del sistema.

- **Saltarse la lógica de negocio**: cuando se producen salidas inesperadas de la lógica de negocio por forzar excepciones o errores no controlados.

Recomendaciones de seguridad

- Utilizar mensajes de error genéricos que no den pistas a los usuarios finales sobre ningún aspecto sensible de la aplicación.
- Utilizar un control de excepciones centralizado.
- La aplicación debe manejar los errores sin depender de los mensajes de error del servidor que se muestran a los usuarios.
- Cualquier lógica de control de acceso que conduzca a un error debe denegar el acceso por defecto.
- Analizar con detalle todas las excepciones que pueden aparecer por el uso de librerías del sistema o de terceros en la aplicación, tratarlas adecuadamente y ofrecer una salida segura a la aplicación.
- Utilizar try/catch/finally para asegurar el cierre de todos los recursos en caso de error.

Ejemplo

El siguiente ejemplo sería el bloque de código típico que debería encontrarse en todo método donde se pretendan controlar los errores que puedan aparecer de un bloque de código.
Cualquier excepción no controlada dentro del bloque try será capturada por catch desde donde se lanzará una excepción controlada. Y, en cualquier caso, se ejecutará el bloque finally para cerrar recursos abiertos antes del try o después del try.

```
try {
...
// código que puede lanzar una excepción
...
} catch (Exception e) {
// manejo de la ex cepción
log.error("ERROR", e);
throw new CustomException("ERROR", e);
} finally {
// cierre o liberación de recursos abiertos
...
}
```

Registro seguro

Consiste en registrar la actividad de las aplicaciones y de los eventos del sistema, relacionados con la seguridad, en cuanto a autenticación, autorización, integridad o confidencialidad, como, por ejemplo, los intentos de conexión fallidos de conexión, la autorización adquirida por un usuario, los accesos a datos sensibles, etc., así como las posibles amenazas detectadas que puedan ser controladas desde la aplicación: inyecciones, enumeración de usuarios, ataques por fuerza bruta, etc. Esto último es especialmente importante cuando se desean realizar análisis forenses sobre incidentes de seguridad ocurridos.

Registro seguro

Los registros de seguridad deben estar especialmente protegidos a nivel de autenticación, integridad, confidencialidad, disponibilidad y trazabilidad:

- **Autenticación/Autorización**: sólo las personas identificadas y autorizadas pueden tener acceso al registro

- **Integridad**: el registro mantiene una firma de integridad que garantiza que no ha sido manipulado a nivel de registro. Esta firma se actualiza en cada nuevo apunte incluido en el registro.

Registro seguro

- **Confidencialidad**: los datos sensibles del registro son cifrados para evitar que pueda accederse al registro mediante métodos distintos a los implementados para acceder a ellos con garantía de autorización.

- **Disponibilidad**: los registros son guardados con redundancia y copias de respaldo.

- **Trazabilidad/Auditabilidad**: deben ser almacenados de forma segura durante un tiempo de retención para cuestiones de auditoría.

Riesgos potenciales

- **Fuga de información**: los registros pueden contener información sensible sobre la aplicación o el sistema y no haberse protegido adecuadamente contra grietas en la autorización del acceso al mismo.

- **Falsificación de registros**: un usuario no autorizado podría modificar los archivos de registro, lo que puede suponer la pérdida de trazabilidad en la aplicación o incluso permitir al usuario malicioso ejecutar código en el sistema.

- **Eliminación de registros**: un usuario malicioso podría eliminar los registros para evitar dejar las huellas de sus delitos.

Recomendaciones de seguridad

- No registrar información sensible como contraseñas, información financiera, tarjetas de crédito, datos personales, etc. En estos casos, utilizar tokens, anonimización de la información o cifrado.

- Validar los parámetros de los componentes variables que conformarán el apunte de registro para evitar inyecciones y comportamientos inesperados.

- Realizar apuntes suficientemente precisos para la seguridad:
  - Los intentos de autenticación, especialmente los fallos.
  - Los accesos concedidos con los roles asociados al usuario.
  - El acceso a los datos sensibles y las acciones realizadas sobre ellos, qué rol es el que se ha utilizado y de qué usuario.
  - Los errores de validación de las entradas

- El registro de seguridad debe ser independiente de otros registros y tener sus propias protecciones de seguridad en el sistema

Ejemplo

El registro seguro en Java se puede llevar a cabo utilizando la librería java.util.logging.

```
import java.util.logging.Logger;
public class MyClass {
	private static final Logger log = Logger.getLogger(MyClass.class.getName());

	public void myMethod() {
		// algunas operaciones que pueden lanzar una excepción
		try {
			// código que puede lanzar una excepción
		} catch (Exception e) {
			Log.severe("Ocurrió un error grave: " + e.getMessage());
		}
	}
}
```

Criptografía

La criptografía es una técnica utilizada para proteger la información y comunicaciones mediante el uso de códigos o claves secretas, y es una herramienta esencial para proteger la información y las comunicaciones.

Se utiliza para garantizar la confidencialidad, integridad y autenticidad de la información y las comunicaciones: la confidencialidad protegiendo la información para que solo pueda ser accedida por las personas autorizadas, la integridad protegiendo la información contra la alteración no autorizada y la autenticación verificando la identidad de las personas o sistemas que acceden a la información.

Criptografía

Funciones hash: Un hash criptográfico es una función matemática que, a partir de un conjunto de datos, produce una cantidad fija de datos que se denominan “resumen” o “hash”. En función de la función utilizada el número de datos obtenido puede variar, pero siempre va a generar el mismo número de datos para la misma función, independientemente del número de datos utilizado en la entrada.

Las funciones hash se utilizan para el almacenamiento de contraseñas o para comprobar la integridad de los datos con el fin de garantizar que no han sido modificados.

Criptografía

Cifrado simétrico: El cifrado simétrico es un tipo de cifrado en el que se utiliza la misma clave tanto para cifrar como para descifrar la información. Se utiliza para proteger la confidencialidad de la información durante su transmisión o como almacenamiento en un dispositivo.

La principal ventaja del cifrado simétrico es que es rápido y fácil de implementar. Su principal desventaja es que ambas partes deben compartir la clave secreta para poder comunicarse de manera segura. Esto puede ser un problema en entornos distribuidos, ya que requiere un mecanismo seguro para compartir la clave de manera confiable.

Criptografía

Cifrado asimétrico: El cifrado asimétrico es un tipo de cifrado en el que se utilizan dos (2) claves diferentes, conocidas como clave pública y clave privada, para cifrar y descifrar la información. La clave pública se utiliza para cifrar la información y puede ser compartida sin problemas, mientras que la clave privada se utiliza para descifrarla y debe mantenerse en secreto.

Las ventajas del cifrado asimétrico es que no se necesita compartir la clave privada para establecer una comunicación segura. La clave pública se puede compartir sin problemas y se utiliza para cifrar la información, mientras que la clave privada se utiliza para descifrarla. Sin embargo, el cifrado asimétrico es más lento que el cifrado simétrico y puede ser más difícil de implementar.

Riesgos potenciales

La información que no está encriptada o que no está correctamente cifrada está expuesta a los siguientes riesgos:

- **Exposición de información sensible**: los datos sensibles estarían en claro para cualquier persona no autorizada.

- **Robo de credenciales y suplantación de identidad:** con la información sensible relativa a contraseñas robada podrían suplantar la identidad de otro usuario y realizar otros ataques como el spoofing.

Riesgos potenciales

- **Fuga de datos personales**: son datos protegidos por regulaciones del país cuya violación de la privacidad podría ocasionar sanciones económicas.
- **Pérdida de reputación de la empresa**: como consecuencia de todo lo anterior.
- **Ataques MitM (Man-in-the-Middle)**: si las comunicaciones no están bien aseguradas, podrían interceptarse las comunicaciones y descifrar todo el flujo de datos para obtener información valiosa que podría servir para otros ataques.

Recomendaciones de seguridad

- Toda la información sensible de una organización como contraseñas, datos personales, repositorios de registro o cualquier otro tipo de información etiquetada como confidencial o superior para la empresa debe ser almacenada de forma ilegible (cifrada) para garantizar la confidencialidad de la empresa.

- Comprobar que todos los números aleatorios, nombres de archivos aleatorios, GUID aleatorios y cadenas aleatorias son generados por un generador de números aleatorios aprobado por el módulo criptográfico.

- Comprobar que existe una política explícita sobre el manejo de las claves criptográficas (como la generación, distribución, revocación y desactualización).

- Comprobar que el ciclo de vida de las claves criptográficas se aplica correctamente.

Recomendaciones de seguridad

- Garantizar que las contraseñas confidenciales o la información crítica que residan en la memoria se sobrescriban con ceros en cuanto dejen de utilizarse para mitigar los ataques de volcado de memoria.

- Comprobar que los números aleatorios se crean con un nivel de entropía adecuado, incluso cuando la aplicación está sometida a una gran carga.

- Comprobar que no se utilizan algoritmos criptográficos obsoletos o débiles como el algoritmo DES de clave simétrica o funciones hash como MD5 o SHA-1 debido a la imposibilidad de garantizar la confidencialidad.

- Utilizar las librerías criptográficas existentes y en todo caso utilizar algoritmos o implementaciones criptográficas personalizadas o creadas por el usuario.

Gestión segura de archivos

Es un proceso que involucra la protección de los datos almacenados en archivos para asegurar su integridad, confidencialidad y disponibilidad, incluyendo medidas como la criptografía, la autenticación, la autorización, el control de acceso y la copia de seguridad.

Este proceso debe considerarse durante la fase de diseño de una aplicación y luego implementarse durante el desarrollo. La mayoría de las aplicaciones cuentan con archivos internos para poder funcionar y, además, si se permite la carga de archivos por parte de los usuarios, se deben establecer los controles de seguridad apropiados.

Riesgos potenciales

Si no hay una buena gestión de archivos podría existir los siguientes riesgos:

- Acceso no autorizado a los archivos, lo que resultaría:
  - Revelación de datos.
  - Pérdida de información sensible.
  - Manipulación de datos.
  - Borrado de datos.
- Carga de archivos maliciosos, lo que resultaría:
  - Ejecución remota de archivos.
  - Ataque de denegación de servicio.
  - Infección de malware.
- Falta de copias de respaldo:
  - Falta de disponibilidad del servicio (DoS).
  - Pérdida de datos de usuario.
  - Pérdida de datos de valor para la empresa

Recomendaciones de seguridad

- Autentificar y autorizar al usuario antes de cargar o descargar cualquier archivo, especialmente si los datos son sensibles.
- No utilizar la entrada proporcionada por el usuario para nombrar archivos o directorios.
- Validar los tipos de contenido no sólo por la extensión, sino también compruebe los tipos MIME para verificar los archivos.
- No permitir la subida de archivos ejecutables a la aplicación.
- Limitar el tamaño de los archivos al mínimo que el servidor pueda manejar sin causar problemas de disponibilidad e impacto a las funcionalidades de la aplicación.

Recomendaciones de seguridad

- Antes de procesar los archivos para el servidor, un escáner antivirus verificar que los archivos no tienen malware o virus.
- Desactivar los privilegios de ejecución en los directorios en los que los usuarios pueden subir archivos.
- No utilizar rutas absolutas cuando se proporcione un enlace de descarga al usuario.
- No almacenar los archivos con sus nombres de forma secuencial.

Seguridad en las transacciones

Es un conjunto de medidas destinadas a proteger las operaciones financieras y de pago de posibles fraudes o ataques. Estas medidas incluyen:

- **Autenticación de usuario**: solo personas autorizadas tengan acceso a las transacciones y cuentas financieras. La autenticación de usuario suele incluir contraseñas seguras y, mejor, autenticación de dos factores.

- **Criptografía**: ayuda a proteger la información confidencial durante las transacciones.

- **Validación de transacciones**: para asegurarse de que son legítimas y que no son parte de un fraude o ataque cibernético.

- **Monitoreo de transacciones**: para ayudar a detectar y prevenir posibles fraudes o ataques en tiempo real.

- **Copias de respaldo**: un plan de recuperación de datos en caso de un ataque o por pérdida accidental de datos garantiza la disponibilidad de estos

Riesgos potenciales

Hay varios riesgos potenciales asociados a las transacciones:

- **Explotación de la vulnerabilidad Payment Bypass**: debido a una inadecuada configuración del sistema de pago. Permite a un atacante manipular los parámetros que se intercambian entre el cliente y el servidor tratando la respuesta antes de ser enviada a la pasarela de pago y eludiendo el sistema de pago en general.

- **Fraude**: haciendo uso de información falsa o manipulando transacciones para obtener beneficios ilícitos.

- **Robo de información confidencial**: para ser usada para hacer daño comercial o para futuros ataques

- **Interrupción del proceso de transacciones**: mediante ataques DoS/DDoS.

- **Pérdida de datos**: por fallos en el sistema, ataques o desastres naturales con consecuencias graves para las transacciones, pudiendo afectar la confidencialidad y la integridad de la información.

Recomendaciones de seguridad

Para evitar la vulnerabilidad Payment Bypass:

- La autorización y confirmación de una compra tiene que realizarse en el lado del servidor.
- Hay que validar que las firmas utilizadas sean correctas durante el proceso de comunicación con la pasarela de pago.
- Validar que el precio está correctamente fijado en el lado del servidor.
- Validar que no se reutilicen pagos.
- El servidor de pago debe comprobar en cada momento en qué fase de la transacción se está.

Recomendaciones de seguridad

- Incluir un tiempo de expiración de la autorización para cada transacción relativamente corto.
- Garantizar la trazabilidad de las transacciones.
- Cifrar las comunicaciones con algoritmos robustos asimétricos.
- Registrar con detalle todas las operaciones anonimizando la información sensible.

Seguridad en las comunicaciones

La seguridad en las comunicaciones de las aplicaciones es esencial para proteger la confidencialidad, la integridad y la disponibilidad de la información transmitida a través de ellas. Esto es especialmente importante en el contexto de las aplicaciones móviles, donde la información puede ser transmitida a través de redes inseguras o públicas.

Riesgos potenciales

Una aplicación sin medidas de seguridad en las comunicaciones está expuesta a varias amenazas potenciales:

- Robo de información confidencial.
- Interrupción del servicio.
- Suplantación de identidad.
- Modificación o destrucción de datos.
- Propagación de software malicioso.

Recomendaciones de seguridad

- Cifrar siempre los canales de comunicación mediante:
  - **TLS**: es un protocolo criptográfico que permite cifrar los canales de comunicación. Este protocolo, aplica privacidad, autenticación e integridad de datos como propiedades del canal de comunicación.
  - **WebSocket**: es una tecnología que proporciona un canal de comunicación bidireccional (en ambos sentidos, enviar y recibir) y full-dúplex (simultáneamente) sobre el mismo socket TCP.

- **Utiliza criptografía robusta**: suficientemente fuerte como para que cualquier intento en descifrar sea en vano.

- **Utiliza protocolos de seguridad**: como HTTPS o FTPS.

Protección de datos

Los componentes en los que se centra la seguridad para proteger los datos son los siguientes:

- **Autenticación**: el usuario que accede al dato es quien dice ser y se le proporciona unos roles o privilegios de acceso.

- **Autorización**: los privilegios de acceso o los roles del usuario permiten realizar sólo cierto tipo de operaciones sobre sólo ciertos datos.

- **Confidencialidad**: los datos deben estar protegidos de la observación o divulgación no autorizada en tránsito y cuando se almacenan.

Protección de datos

- **Integridad**: los datos deben estar protegidos en caso de que sean creados, modificados o eliminados maliciosamente por atacantes no autorizados.

- **Disponibilidad**: los datos deben estar disponibles para los usuarios autorizados siempre que sea necesario (políticas de backups).

Esta norma supone que la protección de los datos se aplica en un sistema de confianza que ha sido bastionado con suficientes protecciones de seguridad.

Riesgos potenciales

Cualquier vulnerabilidad de seguridad que sea explotada sobre los datos de una aplicación podría ocasionar:

- Incumplimiento de la normativa y legislación en materia de tratamiento de datos personales pudiendo ser causa de sanciones económicas o interrupción del servicio.
- Compromiso o pérdida de información sensible de la empresa.
- Compromiso o pérdida de información sensible de terceros que podría ser causa de litigios y pérdidas económicas.
- Pérdida de imagen empresarial.
- Pérdida de certificaciones como consecuencia del incumplimiento relativo a la protección de datos.

Recomendaciones de seguridad

- Comprobar que toda la información confidencial o que tenga datos personales y que vaya a ser tratada por la aplicación esté identificada, y que exista una política explícita que especifique cómo se debe controlar el acceso a la misma, procesarla y, cuando se almacene, encriptarla adecuadamente siguiendo las directrices correctas sobre protección de datos, en cumplimiento de la normativa y la legislación local.

- Asegurarse de que todos los datos confidenciales se envían al servidor en el cuerpo del mensaje HTTP o en las cabeceras, evitando enviar datos confidenciales a través de los parámetros de la URL.

Recomendaciones de seguridad

Comprobar que los canales de comunicación utilizados para el envío de datos confidenciales son seguros mediante algoritmos de cifrado robusto.

- Comprobar que los datos confidenciales almacenados están cifrados con algoritmos de cifrado robusto.

- Comprobar que la aplicación establece suficientes cabeceras contra el almacenamiento en caché, de modo que cualquier información confidencial no se almacene en la caché de los navegadores modernos (por ejemplo, visitar sobre la caché para revisar la caché del disco).

Ejemplo

Un ejemplo de uso de sockets SSL en Java:

```
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;

// Crear una factoría de sockets SSL
SSLSocketFactory sslSocketFactory = (SSLSocketFactory)
SSLSocketFactory.getDefault();

// Crear un socket SSL y establecer la conexión
SSLSocket sslSocket = (SSLSocket)
sslSocketFactory.createSocket("www.example.com", 443);
```

Conclusión:

Las buenas prácticas de seguridad se deben seguir durante todo el proceso de creación de código, desde las primeras líneas de código hasta la finalización del proyecto. Además, hay que utilizar herramientas automatizadas para ir evaluando la calidad del código y detectar fallos tan pronto se produzcan. Lo que nunca podemos hacer es ponernos a pensar en los aspectos seguridad del proyecto a mitad del proyecto de desarrollo.

Dar prioridad a un código seguro sale siempre a cuenta. Es cierto que el desarrollo puede perder velocidad y ganar complejidad. Sin embargo, no hay mayor tranquilidad que saber que la aplicación difícilmente va a sufrir problemas de seguridad en el futuro que puedan tener un impacto negativo en el negocio. ¿Qué costes podría tener que asumir la empresa si un ciberataque explotase una vulnerabilidad con éxito y hubiese una filtración de información sensible de usuarios?

Conclusión:

Poner en práctica unas normas de desarrollo seguro desde las primeras etapas del ciclo de desarrollo es más fácil y más económico que corregir problemas detectados a través del testing en fases más avanzadas o cuando la aplicación ya está en producción, es decir, cuando es utilizada por usuarios reales.

Además de seguir buenas prácticas de desarrollo seguro, hay que mantener el diseño tan simple y mínimo como sea posible. Un diseño muy complejo es una desventaja, ya que aumenta la probabilidad de errores de todo tipo, también de seguridad.

sergiovazquez@utng.edu.mx

GTO
Grandeza de México

Secretaría
de Educación
de Guanajuato

GUANAJUATO
200
AÑOS DE GRANDEZA

2024
200 AÑOS DE GRANDEZA
GUANAJUATO COMO ENTIDAD FEDERATIVA, LIBRE Y SOBERANA