CCN-CERT

BP/28

Recomendaciones sobre desarrollo seguro

INFORME DE BUENAS PRÁCTICAS

ENERO **2023**

ccn-cert centro criptológico nacional

CCN centro criptológico nacional

Edita:

Paseo de la Castellana 109, 28046 Madrid

Fecha de edición: enero de 2023

LIMITACIÓN DE RESPONSABILIDAD

El presente documento se proporciona de acuerdo con los términos en él recogidos, rechazando expresamente cualquier tipo de garantía implícita que se pueda encontrar relacionada. En ningún caso, el Centro Criptológico Nacional puede ser considerado responsable del daño directo, indirecto, fortuito o extraordinario derivado de la utilización de la información y *software* que se indican incluso cuando se advierta de tal posibilidad.

AVISO LEGAL

Catálogo de Publicaciones de la Administración General del Estado
https://cpage.mpr.gob.es

Índice

Índice

1. Introducción

Objetivos

Este documento pretende ayudar a los equipos de desarrollo a comprender los controles de seguridad más comunes que deben aplicarse durante el ciclo de vida del desarrollo de *software*. Las guías de desarrollo seguro proporcionan a los desarrolladores una serie de recomendaciones a seguir que les permiten construir aplicaciones con técnicas de seguridad.

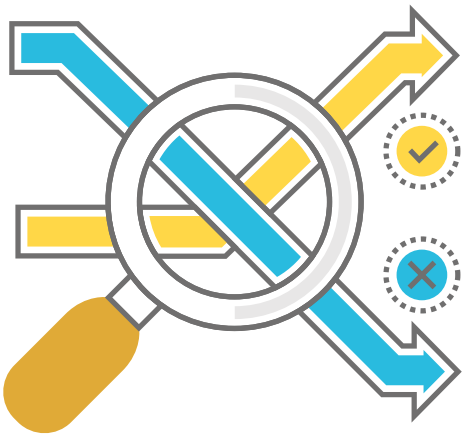
Establecer medidas de seguridad durante el desarrollo de la aplicación en las fases de diseño y codificación no sólo asienta las bases de seguridad contra las vulnerabilidades más comunes, sino que también reduce los costes futuros, ya que solucionar los problemas en fases posteriores conlleva un mayor coste.

Las guías de desarrollo seguro proporcionan a los desarrolladores una serie de recomendaciones a seguir que les permiten construir aplicaciones con técnicas de seguridad

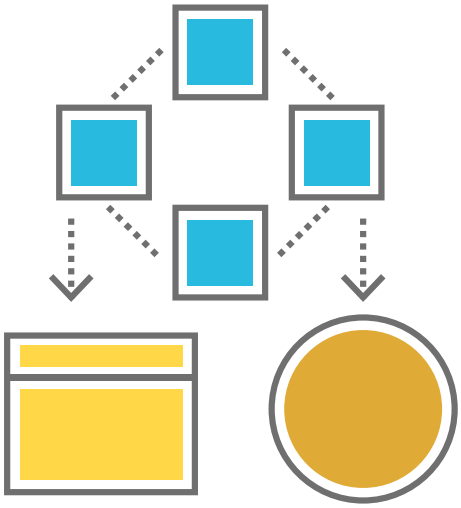
Alcance

Independientemente de la metodología de desarrollo utilizada, la definición de los controles de seguridad debe comenzar en la etapa de diseño o incluso antes, y continuar durante todo el ciclo de vida de la aplicación para garantizar que la aplicación seguirá teniendo las medidas de seguridad pertinentes en caso de sufrir cambios respecto al acuerdo inicial debido a las necesidades del negocio.

Construir un *software* seguro implica actividades a muchos niveles, no sólo para cumplir con las políticas de seguridad internas, sino también para seguir la ley y las regulaciones externas como HIPAA, PCI o GDPR. El *software* resultante debe implementar características de seguridad que cumplan con esos requisitos.

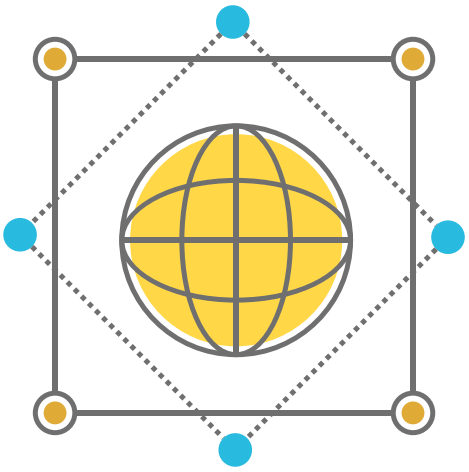


Además, durante el proceso de modelado de amenazas de un proyecto, cuando se combina adecuadamente con la ingeniería de requisitos de seguridad y los principios de diseño seguro, permite al equipo de desarrollo identificar las características de seguridad que son necesarias para garantizar la integridad, la confidencialidad y la disponibilidad de los datos involucrados en el proyecto.



La guía se ha clasificado de la siguiente forma:

- **Recomendaciones de seguridad para los siguientes aspectos:**
  - Arquitectura
  - Autorización
  - Autenticación
  - Gestión de sesiones
  - Validación de parámetros de entrada y salida
  - Gestión de errores
  - Registro seguro
  - Criptografía
  - Gestión segura de archivos
  - Seguridad en las transacciones
  - Seguridad en las comunicaciones
  - Protección de datos
- **Indicaciones complementarias para Python**
- ***Checklist* de Controles de Seguridad**
- **Vulnerabilidades y Controles de Seguridad**
- **Medidas de seguridad ENS y Controles de Seguridad**
- **Anexos al final del documento con Controles de Seguridad Básicos y Avanzados**



2. Arquitectura segura

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.

2.1. Riesgos potenciales

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

- **Se utilizan componentes con vulnerabilidades conocidas.**
- **Se utilizan componentes anticuados, obsoletos o no soportados.**
- **Se utilizan componentes que no han sido identificados.**
- **Se mantienen abiertos puertos prescindibles.**

2.2. Recomendaciones de seguridad

- **Identificar todos los componentes de la arquitectura.**
  - Aquellos componentes que no hayan sido identificado correctamente son riesgos potenciales de seguridad.

- **Realizar una revisión del bastionado de cada dispositivo *hardware* desde el punto de vista de la seguridad:**
  - Revisión de configuraciones. Asegurarse que las configuraciones son las más seguras: sin opciones de depuración activadas, sin usuarios y contraseñas por defecto, etc.
  - Revisión de puertos. Asegurarse que sólo se tienen abiertos los puertos de comunicación que sean estrictamente imprescindibles.
  - Estado de la última actualización del sistema.
  - Identificación de todos los componentes del sistema: Librerías, Módulos, *Frameworks*, Servicios, etc.
  - Para cada componente del sistema, realizar la misma revisión de bastionado en cuanto a las configuraciones y los estados de actualización.

- **Del resultado de la revisión anterior, obtener un informe de los componentes en los cuales existen vulnerabilidades detectadas paras las que no existe actualmente un parche de seguridad y analizar su nivel de riesgo dentro de la aplicación. Podría ser que ciertas vulnerabilidades detectadas no supusieran un riesgo real sobre la aplicación o que no tuviesen un impacto relevante.**

- **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.**

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

Asegurarse que las configuraciones son las más seguras: sin opciones de depuración activadas, sin usuarios y contraseñas por defecto, etc.

- **Mejorar la seguridad perimetral lógica mediante la instalación de Firewalls, dispositivos IDS o similares, o mediante la segmentación de la red.**
- **Garantizar que los datos quedan protegidos por mecanismos de autorización entre entornos mediante la segregación física o lógica y mediante copias de respaldo que aseguren su disponibilidad.**
- **Usar la versión más reciente del lenguaje de programación.**
- **Usar un Entorno Virtual como zona de trabajo del proyecto si aplica según el lenguaje de programación**
- **Importación correcta de paquetes según lenguaje de programación. Paquetes instalados e importados comprobando exhaustivamente la seguridad de los paquetes a instalar.**
- **Desactivar todas las opciones de depuración en Producción que evite fuga de información en los mensajes de error detallados.**
- **Utilizar herramientas en el IDE que realicen análisis semánticos y de seguridad básicos.**

2.3. Referencias

Patrones de Diseño:
https://refactoring.guru/es/design-patterns [1]

OWASP. Design Secure Web Applications:
https://owasp.org/www-pdf-archive/APAC13_Ashish_Rao.pdf [2]

Ámbitos de la Seguridad Nacional. Protección de Infraestructuras Críticas:
file:///Users/lagor/Downloads/BOE-400_Ambitos_de_la_Seguridad_Nacional_Proteccion_de_Infraestructuras_Criticas.pdf [3]

3. 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.

Es la principal área de control de seguridad, por lo que el tipo y método de autenticación debe formar parte del diseño.

La autenticación es la verificación de la identidad de un usuario o dispositivo con el fin de otorgar acceso a sus recursos o información

3.1. 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.
- **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.

3.2. Métodos de autenticación

3.2.1. Autenticación básica

La autenticación básica es un método de **autenticación de red**. En este método de autenticación, el usuario proporciona un nombre de usuario y contraseña en forma de texto en claro (en base64) por lo que se considera una forma insegura de autenticación. Se recomiendan utilizar otros métodos, como la autenticación de dos factores o la autenticación basada en certificados digitales.

RIESGOS

- **Ataque MitM (Man-in-the-Middle):** la información de autenticación se envía en forma de texto claro, lo que significa que podría ser fácilmente interceptada y leída por cualquier persona con acceso a la información descifrada de la red.
- **Reutilización de claves:** si se utiliza la misma contraseña para varios sitios o sistemas, un atacante que obtenga la contraseña a través de la autenticación básica puede utilizarla para acceder a otros recursos protegidos.
- **Ataque de fuerza bruta o diccionario:** este tipo de autenticación no ofrece una protección adecuada contra estos ataques, en los que un atacante intenta adivinar las contraseñas mediante el uso de programas automatizados.
- **Autenticación débil:** no permite a un usuario demostrar su identidad de manera segura y fiable impidiendo la implementación de medidas de seguridad más fuertes, como la autenticación de dos factores o la autenticación basada en certificados digitales.

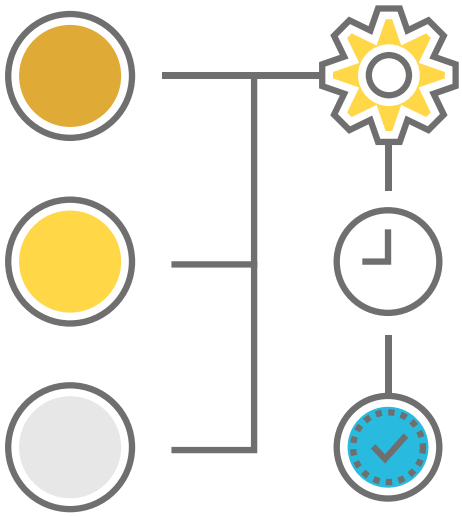
3.2.2. Autenticación mediante formularios

La autenticación mediante formularios es un tipo de **autenticación basada en contraseñas** para un sitio web. En este método de autenticación, el usuario proporciona su nombre de usuario y contraseña mediante un formulario en una página web. El servidor web verifica la información de autenticación recibida y si es correcta, permitirá al usuario su acceso.

La autenticación de formularios puede ser una forma segura de autenticación si se utilizan medidas adecuadas para proteger la información de autenticación, como la cifrado de la información en tránsito y el uso de contraseñas seguras. Sin embargo, también presenta algunos riesgos, como la posibilidad de que un atacante obtenga la información de autenticación mediante técnicas de ingeniería social o mediante el uso de programas de fuerza bruta.

RIESGOS

- **Ataque MitM (Man-in-the-Middle):** se podría interceptar la información de autenticación y utilizarla para acceder al recurso protegido si no se utilizan técnicas adecuadas para proteger la información como el cifrado de las comunicaciones o el uso de contraseñas seguras.
- **Reutilización de claves:** si se utiliza la misma contraseña para varios sitios o sistemas, un atacante que obtenga la contraseña a través de la autenticación de formularios puede utilizarla para acceder a otros recursos protegidos.
- **Ataque de fuerza bruta o diccionario:** tampoco ofrece una protección adecuada contra estos ataques, en los que un atacante intenta adivinar las contraseñas mediante el uso de programas automatizados.
- **Autenticación débil:** no permite a un usuario demostrar su identidad de manera segura y fiable impidiendo la implementación de medidas de seguridad más fuertes, como la autenticación de dos factores o la autenticación basada en certificados digitales.



3.2.3. Autenticación implícita

El usuario no tiene que proporcionar explícitamente credenciales para acceder a un recurso protegido. En lugar de eso, se utilizan otras formas de identificación, como la dirección IP del usuario, la información almacenada en una *cookie* o un *token*. Es un tipo de **autenticación de red** o **basada en *tokens*** dependiendo de la forma elegida para proporcionar esta identificación.

La forma más segura de utilizar este método es utilizar como contraseña el uso de un *hash* criptográfico basada en ésta con el que obtener un *token* suficientemente robusto y largo como para que sea inútil utilizar un ataque de fuerza bruta contra él. El servidor compara el *token* recibido con el *token* que tiene calculado y almacenado para el usuario que requiere su acceso.

RIESGOS

- **Acceso no autorizado:** si se utiliza información como la dirección IP del usuario para determinar si tiene acceso a un recurso protegido, un atacante puede falsificar esta información y obtener acceso no autorizado al recurso.
- **Acceso no autorizado:** si se utilizan *cookies* para almacenar la información de autenticación, un atacante puede obtener acceso a la *cookie* y utilizarla para acceder al recurso protegido.
- **Ataque de fuerza bruta o diccionario:** cuando no se hayan utilizado algoritmos de cifrado suficientemente robustos, como MD5, podría ser vulnerable contra estos ataques.
- **Autenticación débil:** no permite a un usuario demostrar su identidad de manera segura y fiable impidiendo la implementación de medidas de seguridad más fuertes, como la autenticación de dos factores o la autenticación basada en certificados digitales.

3.2.4. Autenticación de cliente HTTP

Es un tipo de autenticación **basada en red** en la que se utiliza un par de claves criptográficas, una clave pública y una clave privada, para verificar la identidad de un usuario o dispositivo. La clave pública se utiliza para cifrar la información y la clave privada se utiliza para descifrarla. El usuario envía un certificado digital que contiene su clave pública para que el servidor cifre el contenido y luego utiliza su clave privada para descifrarlo.

La autenticación mediante certificado de clave pública es considerada una de las formas más seguras de autenticación, ya que permite a un usuario demostrar su identidad de manera confiable y segura. Esto es así siempre que el algoritmo de cifrado sea suficientemente robusto para que este método de autenticación sea realmente seguro. Normalmente este método se utiliza para comunicaciones HTTPS (HTTP sobre SSL/TLS). Lo único negativo a este método es que puede ser más complicada y costosa de implementar que otras formas de autenticación.

El certificado de clave pública del cliente es emitido por una entidad de confianza, como una Autoridad de Certificación (CA) que además proporciona una identificación para el portador.

La autenticación mediante certificado de clave pública es considerada una de las formas más seguras de autenticación, ya que permite a un usuario demostrar su identidad de manera confiable y segura

RIESGOS

- **Acceso no autorizado:** si un atacante logra obtener la clave privada de un usuario, puede utilizarla para acceder a los recursos protegidos de ese usuario de manera no autorizada.

Si se emite un certificado falso con una clave pública falsa, se podría utilizar para acceder a recursos protegidos de manera no autorizada.

- **Ataque MitM:** interceptando un certificado en tránsito, podría utilizarse para acceder a los recursos protegidos de manera no autorizada.
- **Manipulación de un certificado SSL:** la implementación de certificados que no hayan sido verificados por una CA de confianza o de certificados auto-firmados que podrían ofrecer una falsa sensación de seguridad.

3.2.5. Autenticación de Windows

Es un tipo de **autenticación basada en contraseñas** en el cual se verifica la identidad de un usuario que intenta acceder a un equipo o recurso protegido por MS Windows. Se hace mediante el uso de un nombre de usuario y una contraseña, que son verificadas contra la información almacenada en un servidor de autenticación o en un directorio local. MS Windows utiliza dos (2) protocolos de autenticación Kerberos y NTLM.

La autenticación es la verificación de la identidad de un usuario o dispositivo con el fin de otorgar acceso a sus recursos o información

Kerberos utiliza un servidor de autenticación centralizado y claves cifradas para verificar la identidad de los usuarios de manera segura y NTLM utiliza el sistema de archivos local del equipo.

En un entorno de dominio Windows NT o de Directorio Activo, la autenticación de usuario se lleva a cabo mediante el uso de un servidor de autenticación centralizado. Cuando un usuario intenta acceder, su equipo envía una solicitud de acceso al servidor de autenticación y verifica la identidad del usuario utilizando las credenciales almacenadas en el Directorio Activo. Si las credenciales son válidas, el servidor de autenticación emite un ticket de acceso que permite al usuario el acceso.

Este método de autenticación más adecuado en entornos empresariales (Intranet) donde se necesita un control centralizado sobre el acceso a los recursos de la red. Además, al utilizar un Directorio Activo, es posible administrar de forma centralizada las cuentas de usuario y sus permisos de acceso a los diferentes recursos de la red.

Existen bastantes riesgos de seguridad con este método de autenticación, muchos de ellos dependientes de las políticas de seguridad que se hayan establecido por el administrador del servidor de MS Windows.

RIESGOS

- **Ataques de diccionario:** se basan en adivinar contraseñas mediante el uso de programas que intentan combinaciones comunes de letras y números.
- **Ataques de fuerza bruta:** intentan adivinar la contraseña de un usuario mediante el uso de programas que generan y prueban combinaciones de caracteres hasta encontrar la correcta.
- **Ataques de MitM:** intervienen la comunicación entre el usuario y el servidor de autenticación con el fin de obtener las credenciales del usuario.
- **Ataques de "replay":** capturan y reutilizan tickets de acceso válidos emitidos por el servidor de autenticación lo que permitiría a un atacante obtener acceso sin tener que conocer la contraseña del usuario. Los sistemas operativos MS Windows incluyen medidas de seguridad que impiden la reutilización de tickets de acceso.
- **Ataques de *phishing*:** se basan en enviar correos electrónicos falsos que parecen provenir de una fuente confiable con mensajes que utilizan técnicas de ingeniería social, con el fin de obtener las credenciales de un usuario.
- **Vulnerabilidades de *software*:** los sistemas operativos MS Windows y otros programas utilizados en la autenticación contienen vulnerabilidades que pueden ser identificadas y explotadas por atacantes.
- **Fallos de seguridad humana:** errores o descuidos de los usuarios, como utilizar contraseñas débiles, compartir sus credenciales o permitir que se descubran.

3.2.6. Autenticación Passport

Microsoft Passport es un servicio de autenticación utilizado por algunos sistemas Windows y aplicaciones de Microsoft para verificar la identidad de los usuarios. Este servicio utiliza una cuenta de Microsoft, como una cuenta de Outlook o una cuenta de Skype, para verificar la identidad del usuario.

Se basa en el uso de credenciales de un sólo uso, que se envían al servidor de autenticación en lugar de la contraseña del usuario. Esto evita que los atacantes puedan adivinar la contraseña del usuario mediante el uso de técnicas de fuerza bruta o de diccionario. Además, utiliza cifrado para protegerse del riesgo de ataques de MitM.

A partir de MS Windows 10, el sistema de Passport ha evolucionado utilizando autenticación de dos factores. Uno de ellos es el registro del dispositivo y el otro una autenticación biométrica mediante un gesto (Windows Hello) o un PIN.

RIESGOS

- **Dependencia de una cuenta de Microsoft:** para utilizar Microsoft Passport, es necesario tener una cuenta de Microsoft.
- **Vulnerabilidades de *software*:** Microsoft Passport puede tener vulnerabilidades que podrían ser explotadas por atacantes.
- **Riesgos de *phishing* y de error humano:** los mismos problemas indicados en la autenticación por MS Windows.

A partir de MS Windows 10, el sistema de Passport ha evolucionado utilizando autenticación de dos factores. Uno de ellos es el registro del dispositivo y el otro una autenticación biométrica o un PIN

3.3. Riesgos potenciales

En resumen, 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 listas de palabras más comúnmente usadas como contraseñas utilizando variaciones con letras y números. Si las contraseñas son débiles o típicas, estos ataques podrían tener éxito en menos tiempo.

- **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.

- **Ataques de "replay":** una vez que un atacante haya conseguido un ticket de acceso, podría ser reutilizado para obtener acceso.

- **Vulnerabilidades de *software*:** los sistemas operativos y las aplicaciones utilizadas en la autenticación pueden tener vulnerabilidades que pueden ser explotadas por atacantes.

- **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.

- **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.

- ***Bypass* de autenticación:** incluye las técnicas que permiten la autenticación evitando las medidas de seguridad asociadas. Por ejemplo, el acceso a una cuenta de usuario mediante una inyección de código.

- **Robo de identidad:** es cuando el atacante obtiene la identidad de la víctima y la utiliza para realizar algunas acciones. Esto puede hacerse, por ejemplo, robando una *cookie* de sesión de la víctima.

3.4. 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.

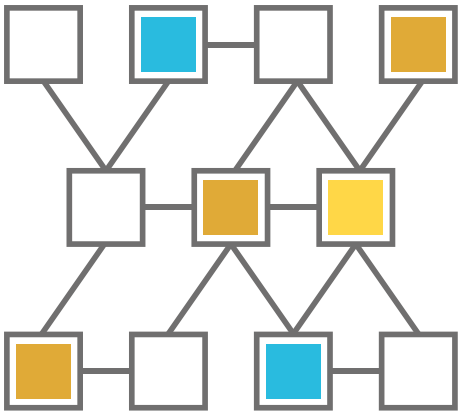
- Utilizar la autenticación multi-factor para aumentar la seguridad en operaciones con acceso a recursos sensibles o en aplicaciones accesibles públicamente.

- Usar doble factor de autenticación al usuario para funciones críticas en la aplicación, como en el cambio de contraseña o en el acceso a recursos especialmente sensibles.

- Implementar el bloqueo de la cuenta después de tres (3) intentos fallidos de conexión y una forma de ponerse en contacto con el administrador para desbloquearla.

- Implementar CAPTCHA para mitigar los ataques de fuerza bruta en aplicaciones expuestas en internet.

- Evitar la enumeración de usuarios, proporcionando mensajes de error genéricos en caso de fallo de autenticación, como “El usuario y/o la contraseña proporcionada no son correctos”.



- Evitar la enumeración de usuarios por ofrecer tiempos distintos de respuesta en caso de usuario inexistente o contraseña incorrecta. Se requiere incluir tiempos de espera aleatorios en ambos casos como para que sea imposible reconocer los casos midiendo los tiempos de respuesta.

- Evitar la enumeración de usuarios en los procedimientos de recuperación de contraseña que requieren introducir el nombre de usuario.

- 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.

- No almacenar de forma persistente la *cookie* de autenticación en el ordenador del cliente, y no utilizarla para otros fines como la personalización.

- Asegurarse de que la sesión es diferente cada vez un usuario se ha conectado con éxito a la aplicación.

- Registrar todos los intentos exitosos o fallidos de autenticación sin exponer la clave utilizada.

- Registrar en un registro específico de seguridad los intentos maliciosos de acceso: Detección de múltiples intentos fallidos de autenticación, detección de intentos de inyección, como SQL o LDAP, detección de múltiples usuarios para mismas IPs, etc.

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

3.5. Ejemplo

Este ejemplo crea un *hash* y *salt* para una contraseña llamando a **getSaltedHash (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.

Es importante utilizar **SecureRandom** para generar el *salt* de manera segura, aleatoria y única para cada contraseña. Se utiliza un número suficientemente alto de iteraciones para el cálculo del *hash* (10000 en este ejemplo) para aumentar la seguridad y hacer más complicado para los atacantes de fuerza bruta calcular el *hash* de una contraseña dada. También es importante utilizar una función de *hash* segura como SHA-512, que es resistente a ataques de colisión y es relativamente rápida de calcular.

```
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.Arrays;
import java.util.Base64;

public class SecureAuth {

   private static final int ITERATION_COUNT = 10000;
   private static final int KEY_LENGTH = 512;

   public static String getSaltedHash(String password) throws IllegalStateException {
      byte[] salt = SecureRandom.getInstanceStrong().generateSeed(KEY_LENGTH / 8);
      return Base64.getEncoder().encodeToString(salt) + "|" + hash(password, salt);
   }

   public static boolean check(String password, String stored) throws IllegalStateException{
      String[] saltAndPass = stored.split("\\|");
      if (saltAndPass.length != 2) {
         throw new IllegalStateException("Use '<salt>|<hash>'");
      }
      byte[] salt = Base64.getDecoder().decode(saltAndPass[0]);
      String hashOfInput = hash(password, salt);
      return hashOfInput.equals(saltAndPass[1]);
   }

   private static String hash(String password, byte[] salt) throws IllegalStateException {
      MessageDigest md = MessageDigest.getInstance("SHA-512");
      md.update(salt);
      byte[] hashedPassword = md.digest(password.getBytes("UTF-8"));
      int iterations = ITERATION_COUNT;
      while (iterations-- > 0) {
         hashedPassword = md.update(hashedPassword);
      }
      hashedPassword = md.digest(hashedPassword);
      return Base64.getEncoder().encodeToString(hashedPassword);
   }
}
```

3.6. Referencias

Autenticación segura en Java:
https://docs.oracle.com/javaee/5/tutorial/doc/bncbe.html#bncbn [4]

Python. Librería para implementar OTP:
https://pypi.org/project/pyotp/ [5]

4. 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.

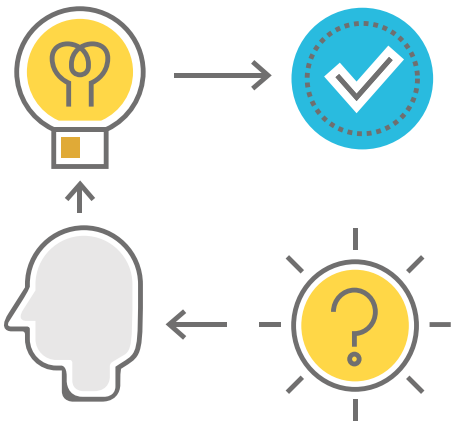
4.1. Riesgos potenciales

Son varias las vulnerabilidades que pueden aparecer en ausencia de controles de seguridad adecuados a la hora de implementar la autorización. A continuación, se detallan algunos de los principales y comunes riesgos identificados en aplicaciones por falta de controles de seguridad en cuanto a 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.
- **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.
- **Modificación de registros:** cuando un atacante consigue acceder y editar los registros de la aplicación o del sistema, comprometiendo la integridad de la trazabilidad de los registros.
- ***Path transversal*:** el atacante se mueve a través del sistema de archivos del servidor, más allá del alcance en el que se supone que debe actuar y fuera del contexto que requeriría una autorización.
- **Lógica de negocio:** si la aplicación no fue bien diseñada podría ocurrir que, si un usuario no sigue la lógica normal de la aplicación para sus funciones de negocio, podría encontrar comportamientos inesperados que podría aprovecharse para realizar operaciones que no deberían haberse permitido.

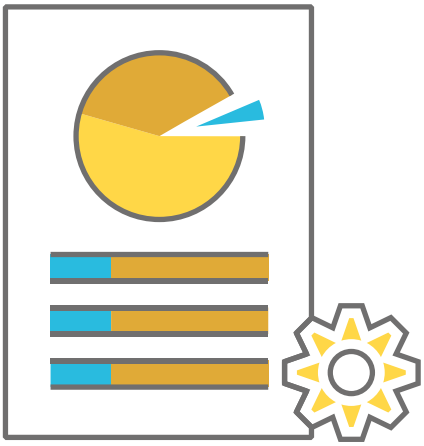


4.2. Recomendaciones de seguridad

- 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.
- Comprobar que el acceso a los registros confidenciales está protegido, de modo que sólo se puede llegar a los objetos o datos autorizados y accesibles para cada usuario (por ejemplo, protegiendo contra los usuarios que manipulan un parámetro para ver o modificar la cuenta de otro usuario).
- Verificar que la navegación por los directorios esté deshabilitada, a menos que se habilite deliberadamente. Además, las aplicaciones no deben permitir el descubrimiento o la divulgación de archivos o directorios, como las carpetas Thumbs.ds, .DS_Store, .git o .svn.
- Garantizar que las reglas de control de acceso se aplican en el lado del servidor.
- Verificar que todos los atributos de los usuarios, los datos y la información de las políticas utilizadas por los controles de acceso no pueden ser manipulados por los usuarios finales a menos que estén específicamente autorizados.
- Verificar que existe un mecanismo centralizado (incluyendo bibliotecas que llaman a servicios de autorización externos) para proteger el acceso a cada tipo de recurso protegido.
- Asegurarse de que la aplicación utiliza *tokens* aleatorios fuertes anti-CSRF o que implementa otro mecanismo de protección de transacciones.

Comprobar que el acceso a los registros confidenciales está protegido, de modo que sólo se puede llegar a los objetos o datos autorizados y accesibles para cada usuario

- 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).



4.3. 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
}
```

4.4. Referencias

Java. Autorización Segura:
https://docs.oracle.com/en/java/javase/19/security/java-authentication-and-authorization-service-jaas1.html [6]

Python. Librería segura de autorización simple:
https://pypi.org/project/python-authorization/ [7]

5. 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.

A partir de aquí, la forma con la que el usuario se identifica con el servidor para el resto de las operaciones puede variar:

- **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.

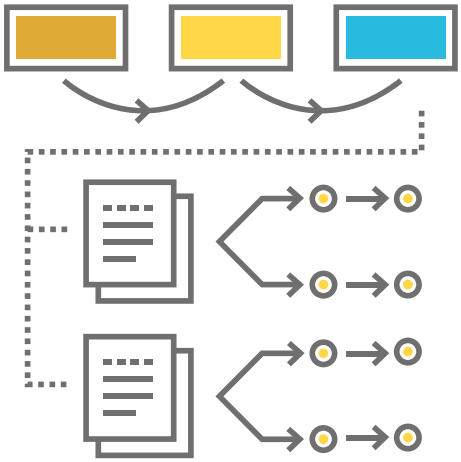
La gestión de sesiones consiste determinar acciones sobre las mismas que sirvan para garantizar la seguridad de la autorización, integridad y privacidad de la información entre el usuario y el sistema

5.1. Aspectos de la seguridad

Los primeros aspectos básicos que considerar en la seguridad de la gestión de sesiones son:

- **Autenticar de manera segura a los usuarios para asegurarse de que solo personas autorizadas tengan acceso al sistema.**
- **Autorizar adecuadamente al usuario que ha obtenido el acceso.**
- **Cifrar con un algoritmo robusto las comunicaciones entre el cliente y el servidor.**

5.1.1. Sesión del lado cliente



La sesión del lado cliente es algo a lo que se suele prestar poca o nula atención y es tan importante o más que la sesión que se crea en el servidor. Si no existe esta sesión podrían crearse las siguientes situaciones de inseguridad a modo de ejemplo, entre otras:

- Un usuario acaba su tarea en la aplicación, abre otra pestaña y cambia de tarea. La sesión del servidor finaliza, pero el cliente no sabe que la sesión ha finalizado mientras no genera actividad en la ventana que dejó olvidada. Esta pestaña podría exponer información sensible que cualquiera que manipulase el dispositivo podría ver y copiar sin necesidad de realizar actividad alguna.
- Un usuario debe rellenar un formulario con bastante información que requiere tiempo en ser completado, incluso podría ir rellenándolo a partir de consultas a otras fuentes que requieren su atención. Si el tiempo de expiración de una sesión de servidor se encontrase en 30 minutos por inactividad, después de 2 horas rellenando el formulario, el usuario pulsaría enviar y como la sesión caducó, todo su trabajo quedó perdido y se encontró con un *login* en la pantalla pidiendo las credenciales de un nuevo acceso. El cliente tuvo actividad durante las 2 horas, pero el servidor no lo pudo percibir.

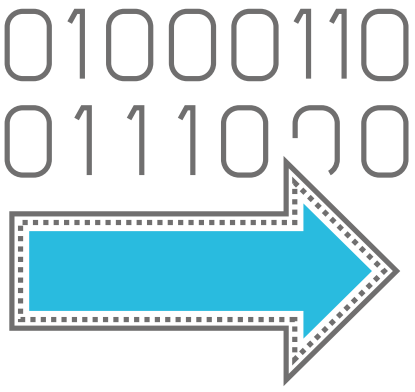
Además, implementar el concepto de sesión en el cliente, permitiría mostrar avisos de expiración tiempo antes de producirse. La sesión del lado cliente, por tanto, debe cumplir algunos requisitos mínimos de seguridad:

- La información persistente del ID de sesión de cliente, debe quedar guardada asociada a la pestaña, para que cuando desaparezca la pestaña, desparezca toda la información asociada a esta sesión. Esto puede implementarse mediante código dentro de la propia página o mediante el objeto **sessionStorage**.
- La información persistente de la sesión de cliente debe borrarse siempre en la pantalla de *login* y crearse en el momento haberse creado la sesión en el servidor, no antes.
- Permitir que la sesión se cliente pueda hacer transacciones vacías contra el servidor simplemente para indicarle que la sesión debe seguir activa y que no la expire. Este comportamiento debería ser bajo demanda en aquellas pantallas que requieran mucho tiempo de atención del lado cliente, nunca que sea como un comportamiento por defecto.
- Como en las sesiones de servidor, éstas deben controlar el tiempo máximo de inactividad con un tiempo similar o inferior al de éstas. En caso de inactividad en el lado cliente, la sesión del cliente debería lanzar una petición de cierre de sesión al servidor y redirigirle a la pantalla de *login*. Establecer un tiempo máximo absoluto de la sesión también podría ser recomendable.
- La información guardada en estas sesiones de cliente deben ser las mínimas y necesarias para la lógica de navegación, y no deberían contener información sensible. Podría contener el *token* de acceso al servidor para evitar arrastrarlo en todas las peticiones (en su caso).
- Tiempo de espera de inicio de sesión, para evitar ataques de fijación de sesión.
- Forzar el cierre de sesión de cliente y servidor en los eventos de cierre de la ventana del navegador o de la pestaña.

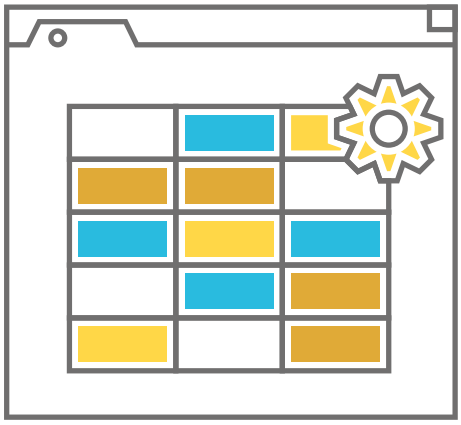
5.1.2. ID de sesión

Los controles mínimos de seguridad asociados al ID de sesión son:

- Todo identificador de sesión debe ser único, suficientemente aleatorio y con una longitud adecuada como la que ofrece un *hash* criptográficamente seguro a partir de un número aleatorio, con una longitud de clave importante.
- El generador de números aleatorios para crear el ID de sesión debe ser un generador de números aleatorios seguro.



- Los ID deben ser validados por el servidor para garantizar que tienen el formato válido y que forman parte de sesiones activas y válidas.
- Los identificadores de sesión no deben ser registrados. En caso de necesidad por trazabilidad de las sesiones, utilizar un ID diferente, nunca usar el ID utilizado para la identificación de la sesión en el tráfico cliente-servidor.
- Se debe generar un nuevo ID de sesión en cada iniciar la sesión o cuando hay un cambio en el nivel de privilegios del usuario, para prevenir ataques de fijación de sesión.
- El almacenamiento y monitorización de los ID de sesión activos debe ser seguro para evitar que pueda ser consultado por personal no autorizado.



5.1.3. Cierre de sesión

Cuando una sesión de servidor se cierra se debe tener presente los siguientes controles de seguridad:

- Redirigir al usuario siempre a la página de *login*.
- Asegurarse que el cliente borrará toda la información de la *cookie* o del *token* de acceso.
- Asegurarse que el cliente borrará toda la información de la sesión cliente (en su caso).
- Registrar los cierres de sesión en el registro de seguridad

5.1.4. Caducidad de la sesión

Los controles mínimos de seguridad sobre la caducidad de la sesión son:

- Toda sesión debe tener un tiempo máximo de vida prudente en caso de inactividad para evitar que la sesión permanezca permanentemente activa.
- Cuando la sesión del servidor caduca debe borrar toda la información relativa a esta sesión.
- Registrar las expiraciones de sesión en el registro de seguridad
- Establecer un tiempo máximo absoluto de duración de una sesión.

5.1.5. Gestión de sesiones en aplicaciones móviles

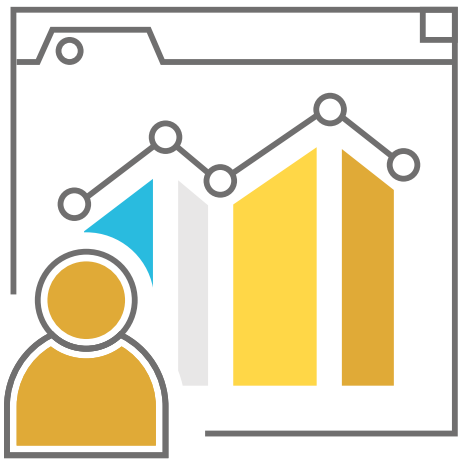
Por cuestiones de usabilidad, las aplicaciones móviles suelen requerir que las sesiones duren más que las aplicaciones web. Las recomendaciones para gestionar las sesiones en las aplicaciones móviles son:

- Utilizar *tokens* que puedan ser retirados en caso de pérdida o robo del dispositivo.
- El tiempo de espera de la sesión de la aplicación móvil debe configurarse y expirar en función de la sensibilidad de la aplicación.
- Utilizar un almacén de datos basado en el servidor para facilitar el uso de la sesión en varias páginas.
- No utilizar nunca un identificador de dispositivo como *token* de sesión.

Utilizar un almacén de datos basado en el servidor para facilitar el uso de la sesión en varias páginas

5.2. 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*.



5.3. 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 **cache-control** 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.
- 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**.
- Configurar adecuadamente el atributo **Path** de las *cookies* de sesión para evitar el acceso a otros dominios.
- 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.
- En el caso de las aplicaciones de alto valor, asegúrese de que se exige al usuario que cierre todas sus sesiones activas si acaba de cambiar la contraseña con éxito.

Es crucial asegurarse de que el ID de sesión nunca se exponga en el tráfico no cifrado

- Limite el acceso a las URLs protegidas, a las funciones, a los datos de la aplicación, a los atributos del usuario y a los datos de configuración de acceso sólo a los usuarios autorizados.
- Registrar en un registro de seguridad todas las sesiones que se crean, se cierran manualmente o por expiración desde el cliente o desde el servidor identificadas por el nombre de usuario y el ID de sesión interno del servidor (que no se comparte con el cliente) incluyendo la fecha/hora del evento.

5.4. 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.

Para utilizar la interface **HttpSession**, primero se debe obtener una instancia:

```
HttpSession session = request.getSession();
```

Luego, se pueden utilizar los métodos de la interface para llevar a cabo la gestión de sesiones:

```
session.setAttribute("user", "John");
```

Para recuperar un atributo de sesión:

```
String user = (String) session.getAttribute("user");
```

Para establecer el tiempo de expiración de la sesión (en segundos):

```
session.setMaxInactiveInterval(3600);
```

Para invalidar la sesión:

```
session.invalidate();
```

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.

```
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.Arrays;
import java.util.Base64;
...

SecureRandom random = new SecureRandom();
byte[] bytes = new byte[32];
random.nextBytes(bytes);
MessageDigest digest = MessageDigest.getInstance("SHA-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);
```

5.5. Referencias

OWASP. Gestión de Sesiones Seguras:
https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html [8]

Java: Framework SpringSession:
https://www.baeldung.com/spring-session [9]

6. Validación de los datos 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.

Aunque la validación de los datos de salida no es una actividad muy común entre los desarrolladores es igualmente importante, y podría ser explotada por usuarios maliciosos si no se incluyeron medidas de seguridad adecuadas. La salida devuelta por una aplicación podría utilizarse para realizar los mismos tipos de ataque, aunque de una forma más sofisticada.

Implementar validaciones de todos los datos en la entrada y en la salida no garantiza que la aplicación quede libre de vulnerabilidades pues dichas validaciones podrían ser insuficientes y no haber tenido en cuenta otro tipo de cuestiones funcionales o de la lógica de negocio que podrían afectar al comportamiento normal de la aplicación. Un ejemplo de esto sería validar un dato de entrada numérico donde se comprueba que es número, que es positivo y que no es 0. Sin embargo, si la aplicación lo utiliza dentro de un bucle iterativo, podría ser llamado con un valor enorme y meter a la aplicación en un bucle casi infinito; lo que sería un ataque DoS.

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

6.1. Técnicas de validación

6.1.1. 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.

Existen librerías para cada tipo de lenguaje que ya incorporan métodos de sanitización.

6.1.1.1. SANITIZACIÓN DE RUTAS

Todas las rutas de archivos o directorios son normalizadas.
Por ejemplo, una ruta UNIX como **/home/usuario/miArchivo.txt**, podría definirse como **/var/log/../../home/user/miArchivo.txt**. Esto podría ser utilizado por un usuario malicioso para navegar por el sistema de archivos y obtener información no autorizada. Para evitar esto se canonicaliza la ruta a una forma fija o directamente se eliminan las subcadenas **"../"** y **"./"** de la ruta.

6.1.1.2. 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.
Hay quien decide eliminar también todos los caracteres blancos dentro del dato y convertirlos directamente a un código de espacio (32).

6.1.1.3. SANITIZACIÓN DE CHARSET

Se verifica que todos los caracteres introducidos en el dato corresponden al *charset* esperado. Aquellos que no lo son, se eliminan o se transforman a un espacio blanco. Se tienen en cuenta los caracteres que pueden llegar codificados como "0xA0", "/xA0", "%A0", etc. para que sean decodificados antes de ser procesados.

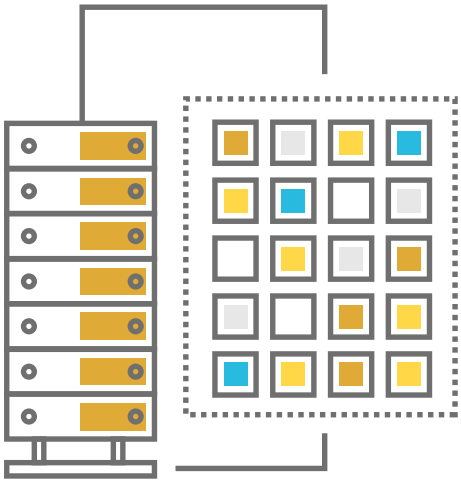
6.1.1.4. SANITIZACIÓN DE CASE

Todos los caracteres son transformados a mayúsculas o minúsculas, según lo requiera el parámetro definido.

6.1.2. 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.

6.1.3. Formato



Permite validar datos con formatos más o menos complejos y variables pero que pueden perfectamente definirse dentro de una expresión regular. Muchos de los tipos de datos complejos validados del punto anterior probablemente se realicen utilizando esta técnica.

Lo único que se debe tener presente es crear una expresión regular tan compleja que pueda dar lugar a ser vulnerable a un taque ReDoS. Para asegurase de que esto no ocurra, se debería validar la expresión regular una herramienta que nos garantice su seguridad e idoneidad como: **RegEx 101** o **RegEx Testing**.

6.1.4. Tamaños mínimos y máximos

Valida que el tamaño (longitud) del dato supere un mínimo de caracteres o no supere un máximo. Esta verificación evitaría que enviasen datos tan grandes que podrían dejar a la aplicación "colgada" simplemente procesando la entrada.

6.1.5. Valores mínimos y máximos

A diferencia del punto anterior esta validación aplica sobre valores numéricos que deben encontrarse dentro de un rango de máximo y mínimo.

6.1.6. Lista blanca

Valida que el dato se encuentra dentro de una lista de datos prefijados. Esta validación se utiliza mucho sobre datos que forman parte de enumeraciones.

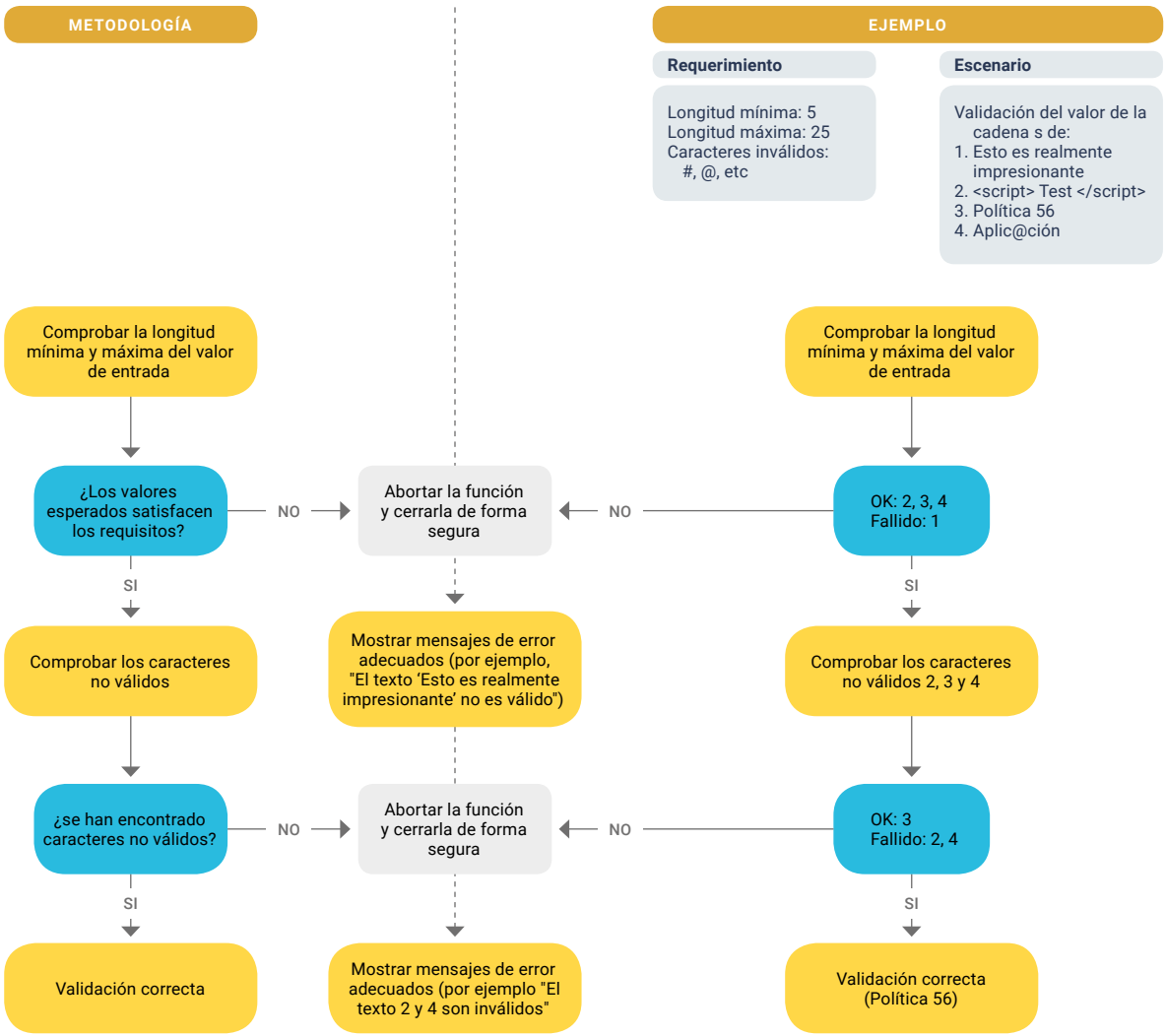
6.1.7. Lista negra

Valida que el dato se no encuentra dentro de una lista de datos prefijados. Este tipo de validación suele ser insuficiente y peligrosa porque difícilmente puede cubrir todas las posibilidades maliciosas.

Sin embargo, puede resultar útil cuando se deciden rechazar textos que contienen palabras ofensivas que existen dentro de una lista negra.

6.2. 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.



6.3. 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.
- **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.

> **La validación de datos es la fuente de muchas vulnerabilidades que un atacante puede explotar**

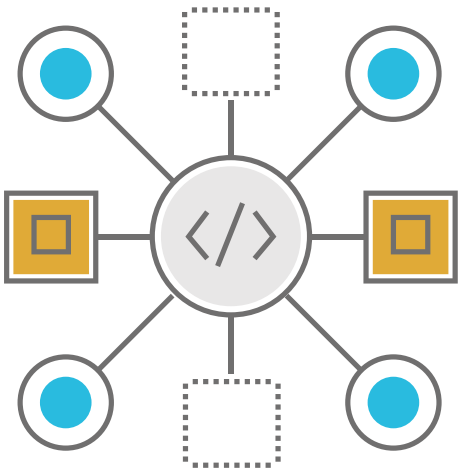
6.4. Recomendaciones de seguridad

- Las validaciones siempre deben realizarse en el lado del servidor. Las validaciones del lado del cliente también pueden ser útiles, pero son sólo recomendadas.
- 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.
- Comprobar que los datos estructurados están fuertemente tipificados y validados según un esquema definido, incluyendo los caracteres permitidos, la longitud y el patrón, por ejemplo, números de tarjeta de crédito o de teléfono, o validando que dos campos relacionados son razonables, como validar suburbios y códigos postales.
- 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.
- Aceptar sólo los datos esperados en cada punto de entrada de la aplicación que proceda del usuario, del proceso final de todos los campos de entrada, formularios, URLs, *cookies* de la aplicación, etc. Cualquier dato inesperado debe ser rechazado.

Las validaciones siempre deben realizarse en el lado del servidor

- 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.
- Comprobar que la aplicación no es susceptible de inyección de comandos.
- Verificar que todas las variables de cadena ubicadas dentro del código HTML u otro código del cliente web se codifican correctamente de forma manual en el contexto o que utilizan plantillas que codifican el contexto de forma automática para garantizar que la aplicación no es susceptible a *Cross-Site Scripting* (XSS) reflejado, almacenado o DOM.
- Compruebe que la aplicación no contiene vulnerabilidades de asignación masiva de parámetros (AKA unión automática de variables). Asegúrese de que todos los datos de entrada son validados, no sólo los campos de los formularios HTML, sino todas las fuentes de entrada, como las solicitudes REST, los parámetros de consulta, las cabeceras HTTP, las *cookies*, los archivos por lotes, las fuentes RSS, etc., utilizando listas blancas, formas de validación menores como las listas grises (que eliminan las cadenas malas conocidas) o las listas negras (que rechazan las entradas malas).
- Verificar que la aplicación restringe los analizadores XML para utilizar sólo la configuración más restrictiva posible y garantizar que las funciones peligrosas, como la resolución de entidades externas, están desactivadas.
- Verificar que la deserialización de datos no confiables se evita o se protege ampliamente cuando la deserialización no se puede evitar.

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



6.5. 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;

  ...
}
```

Se utiliza la anotación **@Valid** junto con un objeto de validación para validar los parámetros de entrada de un método de controlador.

Las anotaciones **@NotBlank** y **@Email** para indicar que el campo **name** no debe estar en blanco y que el campo **email** debe tener un formato de dirección de correo electrónico válido.

6.6. Referencias

OWASP. Validación de Parámetros de entrada: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html [10]

OWASP. Validación de Datos – Librería ESAPI Parámetros de entrada: https://owasp.org/www-project-enterprise-security-api/ [11]

Validación de Formularios Web en Javascript: https://www.tutorialspoint.com/javascript_form_validation_web_application/index.asp [12]

Java. Spring Boot Validation: https: //www.baeldung.com/spring-boot-bean-validation [13]

Prevención XSS: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html [14]

Python. Validación Colander: https://pypi.org/project/colander/ [15]

Python. Validación Cerberus: https://docs.python-cerberus.org/en/stable/ [16]

Python. Validación Schematics: https://schematics.readthedocs.io/en/latest/ [17]

Python. Validación Schema: https://pypi.org/project/schema/ [18]

Python. Validación JSON Schema: https://pypi.org/project/jsonschema/ [19]

7. 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.

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

7.1. 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.

7.2. 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.

Otra forma de aprovecharse de los errores no controlados por usuarios maliciosos es cuando no se cierran correctamente los recursos pudiendo causar con el tiempo una denegación de servicio por realizar un consumo excesivo de recursos no cerrados como consecuencia de forzar errores en la aplicación. Para evitar esta vulnerabilidad es imprescindible asegurar el cierre de todos los recursos cuando dejan de usarse independientemente de los errores que pudieran aparecer durante las operaciones. Se recomienda utilizar las sentencias **try / finally** para asegurar el cierre de los recursos.

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.

7.3. 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.
- Registrar todas las excepciones inesperadas en un registro específico donde indicar, entre otras, la fecha/hora del fallo, el usuario que lo causó y el método donde se produjo el fallo, así como información de la excepción. Este registro debería encontrarse en un entorno seguro sólo accesible a usuarios autorizados.

7.4. 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 excepción
   log.error("ERROR", e);
   throw new CustomException("ERROR", e);
} finally {
   // cierre o liberación de recursos abiertos
   ...
}
```

7.5. Referencias

Java. Try - Catch - Finally:
https://www.w3schools.com/java/java_try_catch.asp [20]

Java. Exception Handling:
https://www.baeldung.com/java-exceptions [21]

Error Handling Strategies:
https://dzone.com/articles/error-handling-strategies [22]

Python. Errores y excepciones:
https://docs.python.org/es/3/tutorial/errors.html#errors-and-exceptions [23]

8. 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.

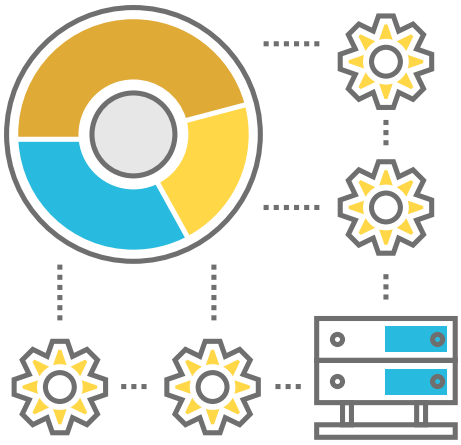
Los registros de seguridad deben estar especialmente protegidos a nivel de autenticación, integridad, confidencialidad, disponibilidad y trazabilidad:

- **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 ni a nivel de apunte. Esta firma se actualiza en cada nuevo apunte incluido en el registro.
- **Confidencialidad:** los datos sensibles del registro son *tokenizados*, anonimizados o 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.

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

8.1. 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.



8.2. 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

- Las excepciones del sistema.
- Las posibles amenazas o intentos de amenaza detectados que pueda controlar la aplicación: inyecciones, ataques de fuerza bruta, enumeración de usuarios, fallos de la lógica de negocio, intentos de operaciones sin privilegios, intentos de *path transversal*, etc.

- Utilizar funciones *hash* para asegurar la integridad de los datos de los registros.

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

8.3. 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());
    }
  }
}
```

8.4. Referencias

Security Log. Best Practices for Logging and Management:
https://www.dnsstuff.com/security-log-best-practices [24]

Java. Logging:
https://docs.oracle.com/javase/7/docs/technotes/guides/logging/index.html [25]

9. 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.

La criptografía se puede utilizar de diferentes maneras: como cifrado de datos sensibles para proteger la información o como firma digital para garantizar la integridad de los datos.

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

9.1. Uso del crifrado

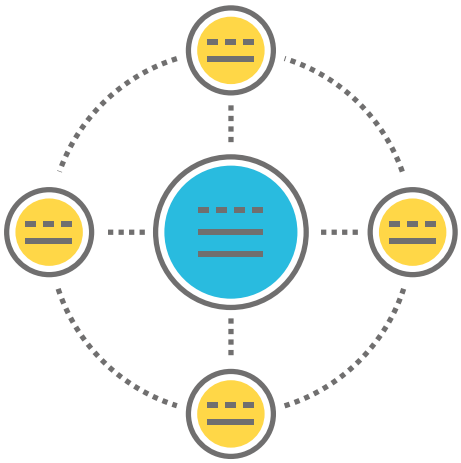
9.1.1. Funciones *hash*

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.

La potencia del *hash* criptográfico es que la probabilidad de generar los mismos datos de salida, en el mismo orden, a partir de una entrada diferente es bajísima, muy improbable. Si esto ocurriese se produciría lo que se llama "colisión", y este efecto descubierto en una de estas funciones matemáticas, podría aprovecharse por usuarios maliciosos para algunos ataques contra la seguridad.

Una función de *hash* se utiliza comúnmente para verificar la integridad de los datos, ya que cualquier cambio en la entrada original se reflejará en un cambio significativo en el *hash* resultante. Cuando se alimenta la función de *hash* con los datos que queremos asegurar, los datos de salida de la función nos ofrecen el *hash* de verificación. Este *hash* sólo se podría volver a obtenerse con la misma función, con los mismos datos de entrada y en el mismo orden.

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.



9.1.2. 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.

9.1.3. 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.

9.2. 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*.
- **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.

9.3. 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.
- 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.

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

9.4. Ejemplo

En este ejemplo se utiliza la criptografía simétrica para cifrar un mensaje utilizando una clave secreta y un vector de inicialización (IV) para proteger contra ataques de "reutilización de clave".

```
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;

public class AdvancedCryptographyExample {
    public static void main(String[] args) throws Exception {
        // Generamos una clave secreta para la criptografía simétrica
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(256); // utilizamos una clave de 256 bits
        SecretKey secretKey = keyGenerator.generateKey();

        // Generamos un vector de inicialización (IV) para la criptografía simétrica
        byte[] iv = new byte[16]; // los IV típicamente tienen un tamaño de 16 bytes
        SecureRandom secureRandom = new SecureRandom();
        secureRandom.nextBytes(iv);
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);

        // Ciframos un mensaje utilizando la clave secreta y el IV
        String message = "Este es un mensaje secreto";
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivParameterSpec);
        byte[] encryptedMessage = cipher.doFinal(message.getBytes());

        // Guardamos la clave secreta de manera segura
            (por ejemplo, utilizando una clave de cifrado)
        saveSecretKey(secretKey);

        // Guardamos el IV y el mensaje cifrado de alguna manera
            (por ejemplo, en una base de datos)
        saveIvAndEncryptedMessage(iv, encryptedMessage);
    }

    private static void saveSecretKey(SecretKey secretKey) {
        // Guardamos la clave secreta de alguna manera
            (por ejemplo, utilizando una clave de cifrado)
    }

    private static void saveIvAndEncryptedMessage(byte[] iv, byte[] encryptedMessage) {
        // Guardamos el IV y el mensaje cifrado de alguna manera
            (por ejemplo, en una base de datos)
    }
}
```

9.5. Referencias

OWASP. Criptografía segura en Java:
https://www.owasp.org/index.php/Using_the_Java_Cryptographic_Extensions [26]

Función de derivación de clave scrypt:
https://www.tarsnap.com/scrypt.html [27]

Python. Funciones criptográficas bcrypt:
https://pypi.org/project/bcrypt/ [28]

10. 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.

10.1. 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.

10.2. 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.
- 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.
- No utilizar información sensible para nombrar los archivos.
- Hay que asegurar que el control de acceso esté configurado como de sólo lectura.
- Limitar el número de archivos subidos por el usuario.
- Almacenar los *hashes* de los archivos subidos para garantizar su integridad.

10.3. Ejemplo

En siguiente ejemplo se valida el tamaño del archivo y se comprueba que el *mimetype* coincide con lo esperado antes de procesar el archivo.

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;

public class FileManager {
   private static final long MAX_FILE_SIZE = 10485760; // 10MB
   private static final List<String> ALLOWED_MIME_TYPES = List.of
      ("text/plain", "image/jpeg", "image/png");

   public static void processFile(String filePath) throws IOException {
      // Comprobar que el archivo existe y es legible
      File file = new File(filePath);
      if (!file.exists() || !file.canRead()) {
         throw new IOException("El archivo no existe o no se puede leer");
      }
      // Comprobar el tamaño del archivo
      long fileSize = file.length();
      if (fileSize > MAX_FILE_SIZE) {
         throw new IOException("El archivo es demasiado grande");
      }
      // Comprobar el mimetype del archivo
      Path path = Paths.get(filePath);
      String mimeType = Files.probeContentType(path);
      if (!ALLOWED_MIME_TYPES.contains(mimeType)) {
         throw new IOException("El tipo de archivo no está permitido");
      }
                  // Procesar el archivo
      // ...
   }
}
```

Otra solución es configurar el servidor Apache para limitar este tipo de peticiones. Es óptimo en cuanto a rendimiento y más seguro porque la petición se queda en el servidor Web y no llega siquiera a entrar en la aplicación.

```
LimitRequestBody 10485760
SetEnvIf Request_URI "^.*$" ALLOWED_MIME_TYPE=1
SetEnvIf Request_URI "^.*\.(txt|jpe?g|png)$" ALLOWED_MIME_TYPE=1
```

10.4. Referencias

Servidor Apache. Directivas de Configuración:
https://httpd.apache.org/docs/2.4/mod/core.html#limitrequestbody [29]

Java. Canonización del Path:
https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/io/File.html#getCanonicalPath() [30]

PHP. Canonización del Path:
https://www.php.net/manual/es/function.realpath.php [31]

Python. Mapeo de archivos con sus MimeTypes:
https://docs.python.org/3/library/mimetypes.html [32]

11. 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.

Toda interacción con una estructura de datos compleja y formada por varios procesos aplicados secuencialmente debe realizarse de una sola vez y de forma segura.

Deben cumplir las siguientes propiedades:

Atomicidad	Consistencia	Aislamiento	Durabilidad
Las transacciones deben tener un punto de inicio y un punto de finalización, siempre, y en cualquier caso. Deben ser identificadas de forma única. Y todas las operaciones realizadas por la transacción deben ejecutarse con éxito o, en caso de error, devolver las operaciones al estado inicial (rollback).	Los datos de las transacciones deben estar validados y ser consistentes en cuanto a la integridad de estos y, en caso de causar un cambio de estado, que sea un estado válido y esperado.	Las operaciones de las transacciones pueden realizarse independientemente de las operaciones de otras transacciones sin afectarse unas con otras en sus datos o sus estados particulares.	Las transacciones tiene un tipo de vida máximo, y una vez que se hayan completado, su información es persistente.

11.1. 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.

11.2. 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.

- **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.**

11.3. Ejemplo

Este ejemplo podría servir de guía para evitar la vulnerabilidad por Payment ByPass:

```
import java.math.BigDecimal;

public class PaymentProcessor {

   private static final BigDecimal MIN_PAYMENT_AMOUNT = new BigDecimal("0.01");

   public void processPayment(String tId, BigDecimal amount, String paymentMethod) {
      // Verificar si el ID de transacción es válido
      if (tId == null || !isValidTransactionId(tId)) {
      throw new IllegalArgumentException("ID de transacción inválido");
      }
      // Verificar si el monto y el método del pago es válido
      if (amount == null || amount.compareTo(MIN_PAYMENT_AMOUNT) < 0) {
      throw new IllegalArgumentException("Monto de pago inválido");
      }
      if (paymentMethod == null || !isValidPaymentMethod(paymentMethod)) {
         throw new IllegalArgumentException("Método de pago inválido");
      }
      // Procesar el pago
   }
   private boolean isValidTransactionId(String transactionId) {
      // Verificar si el ID de transacción está en la lista de IDs permitidos
   }
   private boolean isValidPaymentMethod(String paymentMethod) {
      // Verificar si el método de pago está en la lista de métodos permitidos
   }
}
```

11.4. Referencias

WordPress. Plugin para evitar la vulnerabilidad Payment ByPass:
https://www.acunetix.com/vulnerabilities/web/wordpress-plugin-nab-transact-security-bypass-2-1-0/

12. 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.

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

12.1. Riesgos potenciales

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

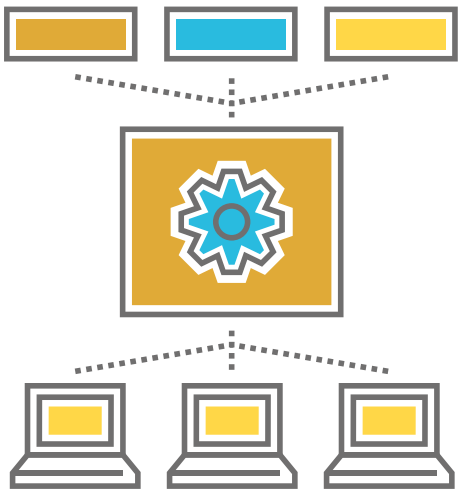
- **Robo de información confidencial.**
- **Interrupción del servicio.**
- **Suplantación de identidad.**
- **Modificación o destrucción de datos.**
- **Propagación de *software* malicioso.**

12.2. 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.

13. 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.
- **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.

13.1. 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.

13.2. 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.

- 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).
- Compruebe que la información confidencial almacenada en memoria se sobrescribe con ceros en cuanto no se necesita, para mitigar los ataques de volcado de memoria.
- Verificar que existe una política de borrado seguro de datos cuando los activos han finalizado su ciclo de vida.

13.3. Ejemplo

Un ejemplo de configuración del servidor Apache Tomcat para asegurar las comunicaciones con TLS sería, en el archivo server.xml:

```
<Connector
   protocol="org.apache.coyote.http11.Http11NioProtocol"
   port="8443" maxThreads="200"
   scheme="https" secure="true" SSLEnabled="true"
   keystoreFile="mykeystore" keystorePass="<password>"
   clientAuth="false" sslProtocol="TLS"/>
```

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);
```

Un ejemplo de uso de conexiones HTTPS en Java:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.URL;
import java.net.URLConnection;

// Crear una conexión HTTPS
URL url = new URL("https://www.example.com");
URLConnection connection = url.openConnection();

// Enviar una solicitud POST a la conexión HTTPS
connection.setDoOutput(true);
OutputStreamWriter out = new OutputStreamWriter(connection.getOutputStream());
out.write("param1=value1&param2=value2");
out.close();

// Leer la respuesta de la conexión HTTPS
BufferedReader in = new BufferedReader(new InputStreamReader(connection.
getInputStream()));
String inputLine;
while ((inputLine = in.readLine()) != null) {
    System.out.println(inputLine);
}
in.close();
```

Un ejemplo de uso de conexiones seguras con WebSockets en Javascript, en el lado cliente:

```
// Crear una conexión HTTPS con WebSockets
const socket = new WebSocket("wss://www.example.com/ws");

// Enviar un mensaje a través de la conexión HTTPS
socket.send("Hello, world!");

// Recibir mensajes de la conexión HTTPS
socket.onmessage = function(event) {
    console.log("Mensaje recibido:", event.data);
};
```

13.4. Referencias

Algoritmos SSL/TLS:
https://docs.oracle.com/en/java/javase/15/docs/specs/security/standard-names.html#sslcontext-algorithms [34]

Java. HttpClient with SSL:
https://www.baeldung.com/java-httpclient-ssl [35]

Python. SSL/TLS:
https://docs.python.org/3/library/ssl.html [36]

Python. WebSockets:
https://pypi.org/project/websockets/ [37]

14. Python: indicaciones complementarias

14.1. Arquitectura

14.1.1. Entorno Virtual

Es aconsejable utilizar un entorno virtual en cualquier proyecto en Python, el cual está equipado para separar el desarrollo de aplicaciones en entornos virtuales.

Un entorno virtual aísla el intérprete de Python, las librerías y los scripts instalados en él. Esto significa que en lugar de usar una versión global de Python y dependencias globales de Python para todos los proyectos que se quieran desarrollar, se puede tener entornos virtuales específicos para cada proyecto y en cada uno tener sus propias versiones de Python, así como sus dependencias.

Los entornos virtuales facilitan el desarrollo, empaquetado y envío de aplicaciones Python seguras. La mayoría de los IDEs tienen funciones incorporadas para cambiar entre entornos virtuales.

A continuación, se puede encontrar un enlace a la librería Python para la creación de entornos virtuales **venv**:

https://docs.python.org/3/library/venv.html [38]

14.1.2. Importación de paquetes

Al trabajar con módulos externos o internos de Python, siempre hay que asegurarse de que se está importando de la manera correcta y utilizando las rutas adecuadas. Existen dos tipos de rutas de importación en Python, absolutas y relativas.

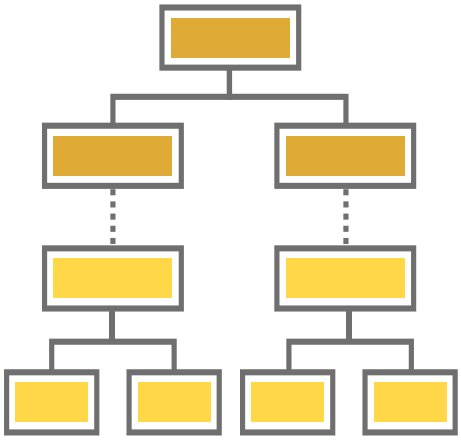
La importación absoluta especifica la ruta del recurso a importar usando su ruta completa desde la carpeta raíz del proyecto mientras que la importación relativa especifica el recurso a importar en relación con la ubicación actual del proyecto donde está la sentencia **import**.

```
/* Absolute Import */

from package1 import module1
from package1.module2 import function1


/* Relative Import */

from .some_module import some_class
from ..some_package import some_function
```



Hay dos (2) tipos de importaciones relativas:

- **Implícita:** las importaciones implícitas no especifican la ruta del recurso en relación con el módulo actual. La importación implícita ha sido eliminada de Python 3, porque si el módulo especificado se encuentra en la ruta del sistema, será importado y eso podría ser muy peligroso. Es posible que un módulo malicioso con el mismo nombre esté en una biblioteca popular de código abierto y encuentre su camino a la ruta del sistema. Si el módulo malicioso se encuentra antes que el módulo real, se importará y podría utilizarse para explotar las aplicaciones que lo tengan en su árbol de dependencias.
  Por lo tanto, se debe usar importación absoluta o la relativa explícita, garantizando la importación del módulo real y el previsto.

```
from safe_module import package, function, class
o
from  ..relative_module import package, function, class
```

- **Explícita:** las importaciones explícitas especifican la ruta exacta del módulo que se quiere importar en relación con el módulo actual.

14.2. Autenticación

El *framework* Django, basado en el lenguaje Python, dispone, en su configuración por defecto, un **sistema de autenticación** y soporta una extensión y personalización de la autenticación. Proporciona tanto la autenticación como la autorización de forma conjunta. Maneja cuentas de usuario, grupos, permisos y sesiones de usuario basadas en *cookies*. Este sistema está formado por:

- **Objetos Users:** son el núcleo del sistema de autenticación. Normalmente representan a las personas que interactúan con la aplicación y se utilizan para permitir cosas como restringir el acceso, registrar perfiles de usuario, asociar contenido con creadores, etc. Sólo existe una clase de usuario en el marco de autenticación de Django, es decir, los usuarios "superusers" o los usuarios "staff" de administración son sólo objetos de usuario con atributos especiales establecidos.

- **Permisos y Autorización:** Django viene con un sistema de permisos incorporado. Proporciona una manera de asignar permisos a usuarios específicos y grupos de usuarios.

- **Autenticación en peticiones web:** Django usa sesiones y middleware para enganchar el sistema de autenticación en los objetos **request**. Estos proporcionan un atributo **request.user** en cada petición que representa al usuario actual. Si el usuario actual no ha iniciado sesión, este atributo será una instancia de **AnonymousUser**, de lo contrario será una instancia de User.

- **Gestión de usuarios en el panel de administración:** cuando se tiene instalado tanto **django.contrib.admin** como **django.contrib.auth**, el panel de administración proporciona un modo.

El *framework* Django, basado en el lenguaje Python, dispone, en su configuración por defecto, un sistema de autenticación y soporta una extensión y personalización de la autenticación

14.3. Gestión de sesiones

Se recomienda hacer uso de las implementaciones por defecto de las protecciones CSRF que existen la gran mayoría de los *frameworks*; como son el ***framework* Django** que por defecto de configuración está activado CSRF middleware, así como tag de plantilla y el ***framework* Flask**.

Ejemplo Python con *framework* Django.

- **Fichero settings.py:** dejar activado el middleware por defecto CSRF en el fichero settings.

```
…
MIDDLEWARE = [
…
'django.middleware.csrf.CsrfViewMiddleware',
…
]
```

- **Fichero plantilla_ejemplo_csrf.html:** añadir la etiqueta **csrf_token** dentro del elemento **<form>** con método POST para una URL interna:

```
…
<form method="post">{% csrf_token %}
…
```

- **Fichero views.py:** se debe usar **RequestContext** en la renderización de la respuesta para que la etiqueta **{% csrf %}** funcione correctamente. A tener en cuenta que si se usa la función **render()**, vistas genéricas apps contrib, esto ya está cubierto ya que en todos ellos se hace uso de **RequestContext:**

```
…
def ejemplo_uso_request_context(request):
    …
return render(request, 'plantilla_ejemplo_csrf.html', {form: form, 'username': username'}))
```

14.4. Validación de parámetros de entrada

14.4.1. Consultas de base de datos parametrizadas

Se recomienda el uso de consultas parametrizadas para cualquier acceso de la aplicación a la base de datos.

Ejemplo de código seguro Python con paso de parámetros de consulta seguro. En el siguiente fragmento de código se define la función **is_admin** que recibe una cadena como parámetro de entrada y devuelve un booleano. Si no existe el usuario devuelve False y en caso de existir el usuario, devuelve el valor de la columna **admin** que puede ser True si es administrador o False si no es administrador.

Al parametrizar la consulta y validar si el resultado de la consulta es **None**, se evita un ataque de vulnerabilidad Inyección SQL:

```
def is_admin(username: str) -> bool:
    with connection.cursor() as cursor:
        cursor.execute("SELECT admin FROM users WHERE username = %(username)s",{'username': username});
        result = cursor.fetchone()

    if result is None:
        # User does not exist
        return False

    admin = result
    return admin
```

```
>>> is_admin('leo')
False
>>> is_admin('irati')
True
>>> is_admin('lemon')
False
>>> is_admin("'; select true; --")
False
```

14.4.2. Protección de Formularios

Se recomienda proteger los formularios en los que existan parámetros cuyo valor pueda ser modificado por los usuarios. Esto se consigue con las siguientes medidas:

- **Codificar adecuadamente la salida de datos:** consiste principalmente en aplicar **HTML encoding** a toda salida que reproduzca datos introducidos por el usuario a la entrada, de modo que no pueda ser interpretado como código por el navegador:
  - Indicar el tipo del contenido de la respuesta en su cabecera Content-type (ver **IANA media-types**[1] [39] para tipos soportados: **application/json, text/html…**), de forma que el lado cliente sepa cómo interpretarlo.
  - Incluir cabecera X-Content-Type-Options=nosniff para evitar el MIME-sniffing por parte de algunos browsers, y por lo tanto la renderización de contenido de forma diferente a la declarada en el Content-type.

- **Filtrar meta-caracteres potencialmente peligrosos:** en las entradas vulnerables. Por ejemplo, los caracteres **"<"**, **">"**, **";"**, **"/"**, **"\\"** y todos los caracteres no imprimibles deberían ser filtrados adecuadamente a la entrada en la aplicación.

- **Aplicar políticas de filtrado de datos de los formularios.**

  Ejemplo de código seguro Python haciendo uso de las funciones **html.escape** (convierte los caracteres &, < y > de la cadena recibida como parámetro de entrada en secuencias seguras para HTML) y **html.unescape** (convierte las) de la librería **html**.

```
import html

bp1 = html.unescape('<html><head></head><body><h1>Best Practices</h1></body></html>')
bp2 = html.unescape('x > 2 && x < 7 single quote: \' double quote: "')
bp3 = html.escape('<html><head></head><body><h1>Best Practices</h1></body></html>')
bp4 = html.escape('x > 2 && x < 7 single quote: \' double quote: "')
print("UNESCAPE")
print(bp1)
print(bp2)
print("ESCAPE")
print(bp3)
print(bp4)
```

```
UNESCAPE
<html><head></head><body><h1>Best Practices</h1></body></html>
x > 2 && x < 7 single quote: ' double quote: "
ESCAPE
&lt;html&gt;&lt;head&gt;&lt;/head&gt;&lt;body&gt;&lt;h1&gt;Best Practices&lt;/h1&gt;&lt;/body&gt;&lt;/html&gt;
x &gt; 2 &amp;&amp; x &lt; 7 single quote: &#x27; double quote: &quot;
```

1 Media Types https://www.iana.org/assignments/media-types/media-types.xhtml

- *Cookies* de sesión marcadas con el ***flag* HttpOnly** para evitar el robo de sesiones en caso de explotar una vulnerabilidad XSS.

- **Implementar todas las recomendaciones en el lado del servidor.**

- **Validación de Inputs.** Se recomienda usar siempre un planteamiento de lista blanca o lista negra, siendo la lista blanca más recomendable donde se rechaza todo aquello que no encaje con las especificaciones.

  Estas comprobaciones deben hacerse tanto en el lado del cliente como en el del servidor, como mínimo en el lado del servidor para garantizar que la lógica de negocio trate los datos de forma segura **(CWE-602)**[2] [40].

  Por otro lado, también es necesario en cualquier lenguaje tipado transformar los tipos de datos en el tipo esperado, por ejemplo, si se recibe un **String** y se espera un **Int**, realizar un casting o transformación.

Ejemplo de código seguro Python lista blanca:

```
if tipo_seguro == "opciónA" or tipo_seguro == "opciónB":
    print("Sí soportado")
else:
    print("No soportado")
```

Ejemplo de código seguro Python comprobación de expresión regular:

```
import re

pat = re.compile(r"[A-Za-z0-9]+")
test = input("Introduzca cadena: ")

if re.fullmatch(pat, test):
    print(f"'{test}' Sí es cadena alfanumérica")
else:
    print(f"'{test}' No es cadena alfanumérica")
```

```
>Introduzca cadena : Alfanumerica1
  ' Ejemplo cadena alfanumérica ' Sí es cadena alfanumérica.
>Introduzca cadena : Alfanumerica 2
  ' Ejemplo cadena alfanumérica ' No es cadena alfanumérica.
```

```
#Integer
x = int(1)   # x es 1
y = int(2.8) # y es 2


z = int("3") # z es 3
v = "6"
if not isinstance(v,int):
    print("No es tipo int")
    u = int(v)    # u es 6

#Float
x = float(1)     # x es 1.0
y = float(2.8)   # y es 2.8
z = float("3")   # z es 3.0
w = float("4.2") # w es 4.2
v = "7"
if not isinstance(v,float):
    print("No es tipo float")
    u = float(v)    # u es 7.0

#String
x = str("s1") # x es 's1'
y = str(2)    # y es '2'
z = str(3.0)  # z es '3.0'
v = 2
if not isinstance(v,str):
    print("No es tipo string")
    u = str(v)    # u es '2'
```

Ejemplo de código seguro Python comprobación de tipo y casting.

2 CWE-602 https://cwe.mitre.org/data/definitions/602.html

- **Evitar un mal uso de input del usuario.** Se debe evitar el uso de input de usuario para llamadas al sistema o para proporcionar partes de archivo. A continuación, un ejemplo en Python de cómo se puede restringir el acceso a los archivos dentro de un directorio específico:

```
import os
import sys

def is_safe_path(basedir, path, follow_symlinks=True):
    # resolves symbolic links
    if follow_symlinks:
        matchpath = os.path.realpath(path)
    else:
        matchpath = os.path.abspath(path)
    return basedir == os.path.commonpath((basedir, matchpath))

def main(args):
    for arg in args:
        if is_safe_path(os.getcwd(), arg):
            print("safe: {}".format(arg))
        else:
            print("unsafe: {}".format(arg))

if __name__ == "__main__":
    main(sys.argv[1:])
```

- **Prevenir XEE.** Deshabilitar siempre la resolución de DTDs externas, de forma que sólo se utilicen DTDs locales definidas de manera estática. También se recomienda siempre validar la estructura del documento XML proporcionado por el usuario, usando como base el fichero DTD de definición de formato de manera estática por el servidor.

  Ejemplo de librería segura de Python analizador XML: **defusedxml**[3] [41] es un paquete Python puro con subclases modificadas de todos los analizadores XML stdlib que impiden cualquier operación potencialmente malintencionada. Se recomienda el uso de este paquete para cualquier código de servidor que analice datos XML que no sean de confianza. El paquete también incluye ataques de ejemplo y documentación ampliada sobre más vulnerabilidades XML, como la inyección de XPath.

  La librería **defusedxml** previene ataques XEE porque no permite el uso de XML con declaraciones **<!ENTITY>** dentro de la DTD y lanza la excepción **EntitiesForbidden** cuando se declara una entidad. Por otro lado, no permite ningún acceso a recursos en remoto o local en entidades externas o DTD y eleva la excepción **External ReferenceForbidden** cuando un DTD o una entidad hace referencia a un recurso externo.

  Ejemplo Python con la librería **defusedxml:**

```
from defusedxml import pulldom
data = parse('ejemplo.xml')
```

- **Estandarización.** Utilizar las funciones nativas de Python para la normalización del charset de toda la información que procese el sistema.

  Ejemplo: librería **codecs**[4] [42]:

```
codecs.encode(obj, encoding='utf-8', errors='strict')
codecs.decode(obj, encoding='utf-8', errors='strict')
```

3 Analizador XML https://pypi.org/project/defusedxml/

4 Estandarización Codecs https://docs.python.org/3/library/codecs.html

- **Sanitización de los datos.** Algunos de los lenguajes con sets de caracteres especiales más comunes y los paquetes recomendados para su sanitización son:
  - HTML/URL: **html-sanitizer**[5] [43]
  - XML: **EscapingXML**[6] [44]
  - JSON: **JsonSchema**[7] [45] implementación de la especificación de JSON Schema para Python.

  La mayoría de los *frameworks* vienen con funciones de sanitización: **Flask**[8] [46] y **Django**[9] [47]

- **Formateo de cadenas.** Python tiene uno de los métodos más potentes y flexibles para formatear cadenas y si no se hace un buen uso, podría terminar abriendo una vulnerabilidad de seguridad en el código. Python3 introdujo **f-strings**[10] [48] y **str.format()**[11] [49] como una forma flexible de formatear cadenas y es realmente muy interesante.

  Sin embargo, esto abre una brecha para la explotación de datos cuando se trata de entradas de usuario. Si la aplicación construida en Python permite a los usuarios el control de la cadena de formato, pueden ser mal utilizados para filtrar datos sensibles. Por ejemplo:

```
CONFIG = {
    "API_KEY": "secret_key"
}
class User:
    name = ""
    email = ""
    def __init__(self, name, email):
        self.name = name
        self.email = email
    def __str__(self):
        return self.name

name = "Nombre"
email = "micorreo@gmail.com"
user = User(name, email)
print(f"{user.__init__.__globals__['CONFIG']['API_KEY']}")
/* secret_key */
```

  Con esto, se puede acceder a los datos globales sensibles de un diccionario CONFIG a través del argumento.

  Sin embargo, Python tiene un módulo de cadenas incorporado que puede ser utilizado para arreglar y prevenir esto. Usando la clase Template del módulo string:

```
from string import Template

nombre_plantilla = Plantilla("Hola, mi nombre es $nombre.")
greeting = name_template.substitute(name="Python")
/* Hola, me llamo Python */
```

5 Sanitización HTML https://pypi.org/project/html-sanitizer/
6 Sanitización XML https://wiki.python.org/moin/EscapingXml
7 Sanitización JSON Schema https://python-jsonschema.readthedocs.io/en/latest/
8 Sanitización Flask https://flask.palletsprojects.com/en/2.0.x/api/#flask.escape
9 Sanitización Django https://docs.djangoproject.com/en/4.0/_modules/django/utils/html/
10 Formateo de Cadenas I https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals
11 Formateo de Cadenas II https://docs.python.org/3/library/stdtypes.html#str.format

15. *Checklist* controles de seguridad

ARQUITECTURA	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	ARQ-01	**Identificar componentes**	Aquellos componentes que no hayan sido identificado correctamente son riesgos potenciales de seguridad
	ARQ-02	**Bastionar componentes**	Asegurarse que las configuraciones son las más seguras: Sin opciones de depuración activadas, sin usuarios y contraseñas por defecto, etc. Asegurarse que sólo se tienen abiertos los puertos de comunicación que sean estrictamente imprescindibles. Estado de la última actualización del sistema. Identificación de todos los componentes del sistema: Librerías, Módulos, *Frameworks*, Servicios, etc. Para cada componente del sistema, realizar la misma revisión de bastionado en cuanto a las configuraciones y los estados de actualización.
	ARQ-03	**Analizar riesgos**	Obtener un informe de los componentes en los cuales existen vulnerabilidades detectadas paras las que no existe actualmente un parche de seguridad y analizar su nivel de riesgo dentro de la aplicación.
	ARQ-04	**Vigilancia actualizaciones**	Mantener una estrecha vigilancia sobre los componentes vulnerables con el fin de que sean actualizados lo antes posible.
	ARQ-05	**Mitigaciones alternativas**	Realizar un estudio de cómo podrían evitarse o mitigarse los problemas de seguridad que crearían estas vulnerabilidades de riesgo mediante sistemas alternativos de seguridad.

[ARQUITECTURA]	ARQ-06	**Seguridad perimetral lógica**	Mediante la instalación de Firewalls, dispositivos IDS o similares, o mediante la segmentación de la red.
	ARQ-07	**Asegurar datos sensibles**	Garantizar que los datos quedan protegidos por mecanismos de autorización entre entornos mediante la segregación física o lógica, y mediante copias de respaldo que aseguren su disponibilidad.
	ARQ-08	**Seguridad del lenguaje de programación**	Usar la versión más reciente del lenguaje de programación. Usar un Entorno Virtual como zona de trabajo del proyecto si aplica según el lenguaje de programación Importación correcta de paquetes según lenguaje de programación comprobando exhaustivamente la seguridad los paquetes a instalar.
	ARQ-09	**Desactivar opciones de depuración**	Especialmente en Producción para evitar fuga de información en los mensajes de error detallados.
	ARQ-10	**Entorno de desarrollo**	Utilizar herramientas en el IDE que realicen análisis semánticos y de seguridad básicos.

AUTENTICACIÓN	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	AUT-01	***Hash* para contraseñas**	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.
	AUT-02	**Botón de desconexión**	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.
	AUT-03	**No exponer credenciales**	Nunca exponer las credenciales en la URL.
	AUT-04	**Usar POST**	Al usar formularios, utilizar métodos POST para el envío de información entre el cliente y el servidor.

[AUTENTICACIÓN]	AUT-05	**Usar autenticación multi-factor**	Utilizar la autenticación multi-factor para implementar múltiples capas de seguridad para aplicaciones sensibles. Usar doble factor de autenticación al usuario para funciones críticas en la aplicación, como en el cambio de contraseña o en el acceso a recursos especialmente sensibles.
	AUT-06	**Bloqueo de cuentas**	Implementar el bloqueo de la cuenta después de 3 intentos fallidos de conexión y una forma de ponerse en contacto con el administrador para desbloquearla.
	AUT-07	**Uso de CAPTCHA**	Implementar CAPTCHA para mitigar los ataques de fuerza bruta en aplicaciones expuestas en internet.
	AUT-08	**Prevenir enumeraciones de usuarios**	Proporcionando mensajes de error genéricos en caso de fallo de autenticación, como "El usuario y/o la contraseña proporcionada no son correctos". Por ofrecer tiempos distintos de respuesta en caso de usuario inexistente o contraseña incorrecta. En los procedimientos de recuperación de contraseña que requieren introducir el nombre de usuario.
	AUT-09	**Desactivar "autocompletar"**	Desactivar el atributo "autocompletar" del campo de contraseña en la aplicación, ya que está activado por defecto.
	AUT-10	**Requisitos de complejidad de contraseñas**	Debe tener un mínimo de ocho caracteres y un máximo prudente. Debe contener, al menos, tres de los siguientes caracteres: Una letra mayúscula, una letra minúscula, un número, un carácter especial.
	AUT-11	**No almacenar *Cookie***	No almacenar de forma persistente la *cookie* de autenticación en el ordenador del cliente, y no utilizarla para otros fines como la personalización.
	AUT-12	**Registrar accesos**	Registrar todos los accesos en un registro específico de seguridad donde conste el resultado del intento de acceso (si fue exitosa o no, si está cancelada o si está bloqueada)

AUTORIZACIÓN	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	ATZ-01	**Mínimo privilegio**	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.
	ATZ-02	**Roles y privilegios**	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.
	ATZ-03	**Protección por autorización**	Comprobar que el acceso a los datos confidenciales está protegido, de modo que sólo se puede llegar a los objetos o datos autorizados y accesibles para cada usuario.
	ATZ-04	**Navegación por directorios deshabilitada**	Verificar que la navegación por los directorios esté deshabilitada, a menos que se habilite deliberadamente.
	ATZ-05	**Control de acceso en el servidor**	Garantizar que las reglas de control de acceso se aplican en el lado del servidor.
	ATZ-06	**Manipulación segura de la Información de usuario**	Verificar que todos los atributos de los usuarios, los datos y la información de las políticas utilizadas por los controles de acceso no pueden ser manipulados por los usuarios finales a menos que estén específicamente autorizados.
	ATZ-07	**Manipulación segura de recursos**	Verificar que existe un mecanismo centralizado (incluyendo bibliotecas que llaman a servicios de autorización externos) para proteger el acceso a cada tipo de recurso protegido
	ATZ-08	***Tokens* anti-CSRF**	La aplicación utiliza *tokens* aleatorios fuertes anti-CSRF o que implementa otro mecanismo de protección de transacciones
	ATZ-09	**Registrar control de acceso**	Registrar todas las operaciones sobre datos sensibles en un registro específico de seguridad incluso si han sido denegadas

GESTIÓN DE SESIONES	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	SES-01	**Cabeceras seguras**	Implementar cabeceras seguras con directivas como **cache-control** o **strict-transport security**
	SES-02	**Invalidación de sesión**	Compruebe que las sesiones se invalidan cuando el usuario cierra la sesión.
	SES-03	**Expiración de sesión**	Las sesiones deben expirar después de un tiempo de inactividad específico.
	SES-04	**Cierre de sesión**	Garantizar que todas las páginas que requieren autenticación tienen un acceso fácil y amigable a la funcionalidad de cierre de sesión.
	SES-05	**No exponer ID de sesión**	Verificar que el ID de la sesión no aparezca nunca en las URL, en los mensajes de error o en los registros.
	SES-06	**Nuevas ID de sesión**	Garantizar que toda autenticación y re-autenticación exitosa genera una nueva sesión y un nuevo ID de sesión, destruyendo el anterior.
	SES-07	***Cookie* de sesión**	Compruebe que el ID de sesión almacenado en las *cookies* se define utilizando los atributos **HttpOnly** y **Secure**. Configurar adecuadamente el atributo Path de las *cookies* de sesión para evitar el acceso a otros dominios.
	SES-08	**Seguimiento de sesiones**	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. En el caso de las aplicaciones de alto valor, asegúrese de que se exige al usuario que cierre todas sus sesiones activas si acaba de cambiar la contraseña con éxito.
	SES-09	**Registrar actividad de sesiones**	Registrar en un registro de seguridad todas las sesiones que se crean, se cierran manualmente o por expiración desde el cliente o desde el servidor.

VALIDACIÓN DE LOS DATOS DE ENTRADA Y SALIDA	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	VAL-01	**Validar en el lado servidor**	Las validaciones siempre deben realizarse en el lado del servidor.
	VAL-02	**Esquemas de validación**	Utilizar esquemas de validación de datos como la mejor solución para validar la entrada de datos. En su defecto, utilizar mecanismos de validación de entrada estándar proporcionados por las bibliotecas específicas de la tecnología.
	VAL-03	**Tipificación de datos**	Comprobar que los datos estructurados están fuertemente tipificados y validados según un esquema definido, incluyendo los caracteres permitidos, la longitud y el patrón.
	VAL-04	**Sanitización de datos**	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.
	VAL-05	**Aceptar sólo los datos esperados**	Aceptar sólo los datos esperados en cada punto de entrada de la aplicación. Cualquier dato inesperado debe ser rechazado.
	VAL-06	**Protección inyección SQL**	Asegurarse de que todas las consultas a la base de datos están protegidas utilizando consultas parametrizadas para evitar la inyección SQL
	VAL-07	**Codificación correcta de variables HTML**	Para garantizar que la aplicación no es susceptible a Cross-Site Scripting (XSS) reflejado, almacenado o DOM
	VAL-08	**Restricciones *parser* XML**	Verificar que la aplicación restringe los analizadores XML para utilizar sólo la configuración más restrictiva posible y garantizar que las funciones peligrosas, como la resolución de entidades externas, están desactivadas
	VAL-09	**Deserialización de datos**	Verificar que la deserialización de datos no confiables se evita o se protege ampliamente cuando la deserialización no se puede evitar.

GESTIÓN DE ERRORES	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	ERR-01	**Exposición de información en errores**	Utilizar mensajes de error genéricos que no den pistas a los usuarios finales sobre ningún aspecto sensible de la aplicación.
	ERR-02	**Centralización de errores**	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.
	ERR-03	**Denegación por defecto en caso de error**	Cualquier lógica de control de acceso que conduzca a un error debe denegar el acceso por defecto.
	ERR-04	**Errores controlados**	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.
	ERR-05	**Registrar errores**	Registrar todas las excepciones inesperadas en un registro específico de seguridad.

REGISTRO SEGURO	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	LOG-01	**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.
	LOG-02	**Validar variables de registro**	Validar los parámetros de los componentes variables que conformarán el apunte de registro para evitar inyecciones y comportamientos inesperados
	LOG-03	**Apuntes precisos**	Realizar apuntes suficientemente precisos que permitan conocer las operaciones y actividades del usuario para la seguridad.
	LOG-04	**Integridad del registro**	Utilizar funciones *hash* para asegurar la integridad de los datos de los registros.
	LOG-05	**Aislamiento del registro de seguridad**	El registro de seguridad debe ser independiente de otros registros y tener sus propias protecciones de seguridad en el sistema.

CRIPTOGRAFÍA	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	CRT-01	**Cifrado de información sensible**	Toda la información sensible de una organización debe ser almacenada de forma ilegible (cifrada) para garantizar su confidencialidad.
	CRT-02	**Generador aleatorio seguro**	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 los números aleatorios se crean con un nivel de entropía adecuado, incluso cuando la aplicación está sometida a una gran carga.
	CRT-03	**Política de gestión de claves**	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.
	CRT-04	**Liberación de información sensible**	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.
	CRT-05	**Uso de algoritmos robustos**	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.

ARCHIVOS SEGUROS	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	FIL-01	Autorización de descarga	Autentificar y autorizar al usuario antes de cargar o descargar cualquier archivo, especialmente si los datos son sensibles.
	FIL-02	No permitir renombrar	No utilizar la entrada proporcionada por el usuario para nombrar archivos o directorios.
	FIL-03	Validar tipos MIME	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.
	FIL-04	Limitar tamaño archivos	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.
	FIL-05	Escanear archivos	Antes de procesar los archivos para el servidor, un escáner antivirus verificar que los archivos no tienen malware o virus.
	FIL-06	Privilegios de directorio	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.
	FIL-07	Nombres de archivo seguros	No almacenar los archivos con sus nombres de forma secuencial. No utilizar información sensible para nombrar los archivos.
	FIL-08	Sólo lectura	Hay que asegurar que el control de acceso esté configurado como de sólo lectura.
	FIL-09	Límite de archivos	Limitar el número de archivos subidos por el usuario.
	FIL-10	Integridad de archivos	Almacenar los *hashes* de los archivos subidos para garantizar su integridad.

SEGURIDAD EN LAS TRANSACCIONES	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	TRN-01	**Protección contra 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.
	TRN-02	**Tiempo de expiración de transacción**	Incluir un tiempo de expiración de la autorización para cada transacción relativamente corto.
	TRN-03	**Trazabilidad de transacción**	Garantizar la trazabilidad de las transacciones.
	TRN-04	**Cifrar comunicación entre transacciones**	Cifrar las comunicaciones con algoritmos robustos asimétricos.
	TRN-05	**Registrar la actividad de transacciones**	Registrar con detalle todas las operaciones anonimizando la información sensible.

SEGURIDAD EN LAS COMUNICACIONES	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	COM-01	**Cifrar siempre los canales de comunicación**	TLS, protocolo criptográfico que permite cifrar los canales de comunicación. WebSocket, tecnología que proporciona un canal de comunicación bidireccional sobre el mismo socket TCP.
	COM-02	**Usar criptografía robusta**	Suficientemente fuerte como para que cualquier intento en descifrar sea en vano.
	COM-03	**Usar protocolos seguros**	Utiliza protocolos de seguridad: Como HTTPS, o FTPS.

PROTECCIÓN DE DATOS	ID	CONTROL DE SEGURIDAD	DESCRIPCIÓN
	DAT-01	**Política de protección de datos**	Comprobar que exista una política explícita de protección de datos.
	DAT-02	**Envío de datos sensibles**	Asegurarse de que todos los datos confidenciales se envían al servidor en el cuerpo del mensaje HTTP o en las cabeceras. Comprobar que los canales de comunicación utilizados para el envío de datos confidenciales son seguros. Comprobar que la aplicación establece suficientes cabeceras contra el almacenamiento en caché.
	DAT-03	**Datos almacenados seguros**	Comprobar que los datos confidenciales almacenados están cifrados con algoritmos de cifrado robusto Comprobar que existen políticas de *backup* para disponibilidad de los datos.
	DAT-04	**Liberación datos sensibles**	Compruebe que la información confidencial almacenada en memoria se sobrescribe con ceros en cuanto no se necesita, para mitigar los ataques de volcado de memoria.
	DAT-05	**Borrado de datos**	Existe una política de borrado seguro de datos cuando los activos han finalizado su ciclo de vida.

16. Vulnerabilidades y controles de seguridad

Es importante tener en cuenta las vulnerabilidades más comunes localizadas en el código para saber cómo deben ser abordadas. Estas vulnerabilidades se han cruzado con los controles de seguridad de los capítulos anteriores.

VULNERABILIDAD	CONTROL DE SEGURIDAD
Comunicación Insegura	Autenticación
Enumeración de Nombres de Usuario	
Contraseña Débil	
Falsificación de Petición en Sitios Cruzados - Cross-Site Request Forgery (CSRF)	Autorización
Fallas de Identificación y Autenticación	
Acceso Directo a Objetos	
Control de Acceso	
Secuencia de Comandos en Sitios Cruzados - Cross-Site Scripting (XSS)	Validación de Datos
Inyección SQL	
Desbordamiento de Búfer	
Falsificación de Registros	
SQL Dinámico	
Vulnerabilidad de Redirección Abierta	
Codificación de la Salida	
Revelación de Información	
Gestión Débil de la Sesión	Gestión de la Sesión
Caché de Formularios	
Divulgación de Información	Tratamiento de Errores
Divulgación de Información	Registro
Registro no Existe para Funciones Críticas	
Almacenamiento de Información Sensible en Texto no Cifrado	Criptografía
Criptografía Débil	
Carga Insegura de Archivos	Gestión Segura de Archivos
Divulgación de Información	

17. Medidas de seguridad ENS y controles de seguridad

MEDIDAS DE SEGURIDAD		GUÍA DE DESARROLLO SEGURO
op.exp	**Explotación**	**Arquitectura**
op.exp.4.1, op.exp.4.2, op.exp.7.r4.1		RECOMENDACIONES DE SEGURIDAD
mp.com	**Protección de las comunicaciones**	
mp.com.1.1, mp.com.4		
op.acc	**Control de acceso**	**Autenticación**
op.acc.5.r3.2, op.acc.5.8, op.acc.5.r1.2, op.acc.5.r6.1		RECOMENDACIONES DE SEGURIDAD
mp.sw	**Protección de las aplicaciones**	**Autorización**
mp.sw.1.r1.1		RECOMENDACIONES DE SEGURIDAD
mp.s	**Protección de servicios**	
mp.s.2.1		
op.pl	**Planificación**	
op.pl.2.4		
op.mon	**Monitorización del sistema**	
op.mon.1.r2.1		
op.acc	**Control de acceso**	
op.acc.4.1, op.acc.6.r5.1		

mp.eq	**Protección de los equipos**	**Gestión de la Sesión**
mp.eq.2.r1.1		RECOMENDACIONES DE SEGURIDAD
op.pl	**Validación de datos**	**Validación de Parámetro**
op.pl.2.r3.1		RECOMENDACIONES DE SEGURIDAD
mp.s	**Protección de los servicios**	
mp.s.2.3		
op.exp	**Explotación**	**Registro**
op.exp.5.1, op.exp.7.r2.1, op.exp.8.2		RECOMENDACIONES DE SEGURIDAD
mp.s	**Protección de los servicios**	
mp.s.3.r1.1		
op.acc	**Control de acceso**	
op.acc.6.r9.2		
mp.si	**Protección de soportes de información**	**Criptografía**
mp.si.2.1		RECOMENDACIONES DE SEGURIDAD
op.exp	**Explotación**	
op.exp.10		
op.exp	**Explotación**	**Gestión Segura de Archivos**
op.exp.6.3		RECOMENDACIONES DE SEGURIDAD
mp.com	**Protección de las comunicaciones**	**Seguridad en las Comunicaciones**
mp.com.2.r5.1		RECOMENDACIONES DE SEGURIDAD
mp.info	**Protección de la información**	**Protección de Datos**
mp.info.1.1, mp.info.2		RECOMENDACIONES DE SEGURIDAD

18. Glosario

OWASP (Open Web Application Security Project): es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el *software* sea inseguro. La Fundación OWASP es un organismo sin ánimo de lucro que apoya y gestiona los proyectos e infraestructura de OWASP.

La comunidad OWASP está formada por empresas, organizaciones educativas y particulares de todo mundo. Juntos constituyen una comunidad de seguridad informática que trabaja para crear artículos, metodologías, documentación, herramientas y tecnologías que se liberan y pueden ser usadas gratuitamente por cualquiera.

***Cookie*:** archivo de pequeño tamaño enviado por un sitio web y almacenado en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del navegador. Así, es posible identificar al usuario que visita un sitio web y llevar un registro de su actividad en el mismo.

CSRF (Cross Site Request Forgery): es una vulnerabilidad de falsificación de peticiones en sitios cruzados. Consiste en engañar a un usuario legítimo para que ejecute peticiones o acciones sin su consentimiento, sin saber lo que está haciendo.

MFA (Multi-Factor de Autenticación): es un método de control de acceso en el que a un usuario se le concede acceso al sistema solo después de que presente dos o más pruebas diferentes de que es quién dice ser.

XEE/JEE (Xml/Json External Entity): es una vulnerabilidad de inyección de código en una aplicación que analiza datos XML/Json.

DTD (Document Type Definition): es una definición en un documento SGML o XML, que especifica restricciones en la estructura y sintaxis de este.

CWE: es una lista desarrollada por la comunidad de tipos de debilidades de *software* y *hardware*. Sirve como un lenguaje común, una regla de medir para las herramientas de seguridad, y como una línea de base para la identificación de debilidades, la mitigación y los esfuerzos de prevención

19. Referencias

[1] «Patrones de Diseño,» [en línea]. Available: https://refactoring.guru/es/design-patterns

[2] «OWASP: Design Secure Web Applications,» [en línea]. Available: https://owasp.org/www-pdf-archive/APAC13_Ashish_Rao.pdf

[3] «Ámbitos de la Seguridad Nacional: Protección de Infraestructuras Críticas,» [en línea]. Available: file:///Users/lagor/Downloads/BOE-400_Ambitos_de_la_Seguridad_Nacional_Proteccion_de_Infraestructuras_Criticas.pdf

[4] «Autenticación Segura Java,» [en línea]. Available: https://docs.oracle.com/javaee/5/tutorial/doc/bncbe.html#bncbn

[5] «Python: Librería para implementar OTP,» [en línea]. Available: https://pypi.org/project/pyotp/

[6] «Java: Autorización Segura,» [en línea]. Available: https://docs.oracle.com/en/java/javase/19/security/java-authentication-and-authorization-service-jaas1.html

[7] «Python: Librería segura de autorización simple,» [en línea]. Available: https://pypi.org/project/python-authorization/

[8] «OWASP: Gestión de Sesiones Seguras,» [en línea]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[9] «Java: Framework SpringSession,» [en línea]. Available: https://www.baeldung.com/spring-session

[10] «OWASP: Validación de Parámetros,» [en línea]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[11] «OWASP: Validación de Datos – Librería ESAPI,» [en línea]. Available: https://owasp.org/www-project-enterprise-security-api/

[12] «Validación Formularios en Javascript,» [en línea]. Available: https://www.tutorialspoint.com/javascript_form_validation_web_application/index.asp

[13] «Java: Spring Boot Validation,» [en línea]. Available: https://www.baeldung.com/spring-boot-bean-validation

[14] «Prevención XSS,» [en línea]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[15] «Validación Colander,» [en línea]. Available: https://pypi.org/project/colander/

[16] «Validación Cerberus,» [en línea]. Available: https://docs.python-cerberus.org/en/stable/

[17] «Validación Schematics,» [en línea]. Available: https://schematics.readthedocs.io/en/latest/

[18] «Validación Schema,» [en línea]. Available: https://pypi.org/project/schema/

[19] «Validación JSON Schema,» [en línea]. Available: https://pypi.org/project/jsonschema/

[20] «Java: Try - Catch - Finally,» [en línea]. Available: https://www.w3schools.com/java/java_try_catch.asp

[21] «Java: Exception Handling,» [en línea]. Available: https://www.baeldung.com/java-exceptions

[22] «Error Handling Strategies,» [en línea]. Available: https://dzone.com/articles/error-handling-strategies

[23] «Python: Errores y excepciones,» [en línea]. Available: https://docs.python.org/es/3/tutorial/errors.html#errors-and-exceptions

[24] «Security Log: Best Practices for Logging and Management,» [en línea]. Available: https://www.dnsstuff.com/security-log-best-practices

[25] «java: Logging,» [en línea]. Available: https://docs.oracle.com/javase/7/docs/technotes/guides/logging/index.html

[26] «OWASP: Criptografía segura en Java,» [en línea]. Available: https://www.owasp.org/index.php/Using_the_Java_Cryptographic_Extensions

[27] «Función de derivación de clave scrypt,» [en línea]. Available: https://www.tarsnap.com/scrypt.html

[28] «Python: Funciones criptográficas bcrypt,» [en línea]. Available: https://pypi.org/project/bcrypt/

[29] «Servidor Apache: Directivas de Configuración,» [en línea]. Available: https://httpd.apache.org/docs/2.4/mod/core.html#limitrequestbody

[30] «Java: Canonización del Path,» [en línea]. Available: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/io/File.html#getCanonicalPath()

[31] «PHP: Canonización del Path,» [en línea]. Available: https://www.php.net/manual/es/function.realpath.php

[32] «Python: Mapeo de archivos con sus MimeTypes,» [en línea]. Available: https://docs.python.org/3/library/mimetypes.html

[33] «WordPress: Plugin para evitar la vulnerabilidad Payment ByPass,» [en línea]. Available: https://www.acunetix.com/vulnerabilities/web/wordpress-plugin-nab-transact-security-bypass-2-1-0/

[34] «Algoritmos SSL/TLS,» [en línea]. Available: https://docs.oracle.com/en/java/javase/15/docs/specs/security/standard-names.html#sslcontext-algorithms

[35] «Java: HttpClient with SSL,» [en línea]. Available: https://www.baeldung.com/java-httpclient-ssl

[36] «Python: SSL/TLS,» [en línea]. Available: https://docs.python.org/3/library/ssl.html

[37] «Python: WebSockets,» [en línea]. Available: https://pypi.org/project/websockets/

[38] «Entorno Virtual,» [en línea]. Available: https://docs.python.org/3/library/venv.html

[39] «Media Types,» [en línea]. Available: https://www.iana.org/assignments/media-types/media-types.xhtml

[40] «CWE-602,» [en línea]. Available: https://cwe.mitre.org/data/definitions/602.html

[41] «Analizador XML,» [en línea]. Available: https://pypi.org/project/defusedxml/

[42] «Estandarización Codecs,» [en línea]. Available: https://docs.python.org/3/library/codecs.html

[43] «Sanitización HTML,» [en línea]. Available: https://pypi.org/project/html-sanitizer/

[44] «Sanitización XML,» [en línea]. Available: https://wiki.python.org/moin/EscapingXml

[45] «Sanitización JSON Schema,» [en línea]. Available: https://python-jsonschema.readthedocs.io/en/latest/

[46] «Sanitización Flask,» [en línea]. Available: https://flask.palletsprojects.com/en/2.0.x/api/#flask.escape

[47] «Sanitización Django,» [en línea]. Available: https://docs.djangoproject.com/en/4.0/_modules/django/utils/html/

[48] «Formateo de Cadenas I,» [en línea]. Available: https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals

[49] «Formateo de Cadenas II,» [en línea]. Available: https://docs.python.org/3/library/stdtypes.html#str.format

ANEXO A. *Cheatsheet* básica

PRINCIPIOS DE DESARROLLO SEGURO		*CHECKLIST* CONTROLES DE SEGURIDAD	
Mínimo Privilegio	Un actor debe tener el mínimo nivel de permisos sobre recursos del sistema durante el mínimo tiempo posible. Se debe denegar por defecto el acceso a los recursos.	**Sistema de Autorización**	Dividir desde un principio una aplicación en un sistema de autorización, impidiendo a todo agente no autorizado acceder a toda la aplicación. Ej: Sistema RBAC.
Separación de Responsabilidades	Toda tarea compleja o sensible debe requerir la participación de más de un actor con diferentes niveles de rol. Así se evita que un único actor comprometa todo el sistema.	**Parametrizadas**	Uso de consultas parametrizadas para cualquier acceso de la aplicación a la base de datos.
Defensa en Profundidad	Establecer múltiples mecanismos de seguridad en capas, con diferentes niveles de complejidad y factores de control. Se busca evitar el "Single Point ofFailure".	**Protección de Formularios**	Protección de los formularios en los que existan parámetros cuyo valor pueda ser modificado por los usuarios. Esta protección consiste en codificar adecuadamente la salida de datos, filtrar meta-caracteres potencialmente peligrosos en las entradas vulnerables y aplicar políticas de filtrado de datos de los formularios.
Fallo Seguro	En caso de fallo, el sistema debe volver a un estado seguro, minimizando el compromiso de la confidencialidad, integridad o disponibilidad del sistema.	**Validación de Inputs**	Validación de cualquier área de input y de forma eficiente incluido los datos provenientes de internet, de clientes, de proveedores y reguladores.
Economía de Mecanismos	La implementación de funcionalidades y controles de seguridad de un sistema debe ser lo más simple posible. Más simple significa que menos cosas pueden salir mal.	**Método de Transacción Seguro**	• La autorización y confirmación de una compra tiene que realizarse en el lado del servidor. • 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 de comprobar en cada momento en qué fase de la transacción se está. • La autorización de cada transacción debe tener un periodo de expiración relativamente corto.
Mediación Completa	Toda petición de acceso a recursos del sistema debe ser validada, de manera que los controles de autenticación y autorización no puedan ser circunvalados.	**Uso de MFA**	Implementación de un Múltiple Factor de Autenticación en aplicaciones de manejo de datos sensibles expuestas a internet y en accesos de usuarios con privilegios.

Diseño Abierto	Si un diseño es robusto, no es necesario mantenerlo en secreto para garantizar su seguridad. Los detalles de implementación deben ser independientes del diseño. No llevar a cabo "Seguridad por Oscuridad"	**Protección datos Sensibles**	Realizar una serie de acciones desde el inicio de la fase de desarrollo de una aplicación, como pueden ser los siguientes: • Identificar los datos a tratar que sean sensibles respecto a requerimientos regulatorios de privacidad. • Aplicar controles como cifrado en tránsito o reposo. • No almacenar datos sensibles innecesariamente. • Deshabilitar el cacheo de datos sensibles. • Tener cifrados los datos en reposo. • Todo el tráfico de datos debe ser cifrado con métodos como TLS. • Almacenar contraseñas usando algoritmos fuertes de *hashing* con un factor de trabajo que ralentice posibles roturas de los mismos.
Mecanismo Menos Común	Se debe restringir al mínimo los recursos compartidos entre diferentes usuarios del sistema, por cuestiones de confidencialidad y concurrencia.	**Protecciones CSRF**	Incluir implementación por defecto de las protecciones CSRF en el *framework* en uso.
Aceptabilidad Psicológica	Se debe considerar el impacto de los controles de seguridad en la usabilidad del sistema. Lo ideal es que los controles de seguridad sean transparentes para el usuario.	**Prevenir XEE/JEE**	Deshabilitar la resolución de DTDs externas y validar la estructura del documento XML.
Eslabón Más Débil	Se debe identificar el punto principal de compromiso de un sistema e implementar los controles de seguridad necesarios para protegerlo. Un sistema es tan seguro como el eslabón más débil	**Evitar uso de determinados inputs**	Evitar el uso de input de usuario para llamadas al sistema o para proporcionar partes de archivo.
Aprovechar Componentes Existentes	Es recomendable fomentar la reutilización de componentes bien probados y soluciones consolidadas en la arquitectura del sistema.	**Uso versión actualizada del lenguaje**	Usar la versión más reciente del lenguaje de
Para ampliar información sobre los Principios de Desarrollo Seguro véase CCN-CERT - WorkShop - Practical Approch to Secure Application Development v 1.0.		**Uso Entorno Virtual**	Uso de un Entorno Virtual como zona de trabajo del proyecto si aplica según el lenguaje de programación
		Importación correcta de Paquetes	Realizar path de importaciones según lenguaje de programación.
		Uso Formateo Cadenas Seguro	Formateo de Cadenas de entradas de usuario de forma que no permita el control de la cadena de formato y evitar el filtrado de datos sensibles.
		Uso Seguro de Peticiones HTTP	Manejo de las peticiones HTTP con seguridad evitando peticiones a fuentes explotadas que puedan devolver código explotado en las cabeceras o en el cuerpo de la respuesta.
		Paquetes instalados e importados Seguros	Comprobación exhaustiva de la Seguridad los Paquetes que se quieran instalar.
		Deserialización de Datos con Seguridad	Hacer uso de funciones de librerías de deserialización que eviten vectores de ataque.
		Mantener Vulnerabilidades Actualizadas	Mantener al día las Vulnerabilidades de Código abierto en los Paquetes instalados e importados.
		Desactivar Depuración en Producción	Poner a False la Depuración en Producción que evite fuga de información en los mensajes de error detallados.
		Escaneo de código	Usar herramientas en el IDE que realicen análisis semánticos.

ANEXO B. *Cheatsheet* avanzada

PRINCIPIOS DE DESARROLLO SEGURO	*CHECKLIST* CONTROLES DE SEGURIDAD	ENTIDADES DE REFERENCIA		MAPEO DE CONTROLES DE SEGURIDAD	
Mínimo Privilegio	Sistema de Autorización	**ISO**	Promueve la seguridad, claridad y fiabilidad de productos y para diversas industrias publicando estándares aplicables a escala global	**Comunicación Insegura**	Autenticación
Separación de Responsabilidades	Parametrizadas	**NIST**	Laboratorio de ciencias físicas y una agencia no regulatoria del departamento de comercio de los EE.UU.	**Enumeración de Nombres de Usuario**	
Defensa en Profundidad	Protección de Formularios	**OWASP**	Comunidad online que proporciona de forma gratuita artículos, metodologías, documentación, herramientas y tecnologías en el campo de la seguridad de aplicaciones web	**Contraseña Débil**	
Fallo Seguro	Validación de Inputs	**MITRE**	Organización que proporciona ingeniería de sistemas, investigación, desarrollo y soporte sobre tecnologías de la información al gobierno de EE. UU.	**Falsificación de Petición en Sitios Cruzados Cross-Site Request Forgery (CSRF)**	Autorización
Economía de Mecanismos	Método de Transacción Seguro	**CONTROLES PROACTIVOS**	Para ampliar información sobre los Principios de Desarrollo Seguro véase CCN-CERT - WorkShop - Practical Approch to Secure Application Development v 1.0.	**Falla de Identificación y Autenticación**	
Mediación Completa	Uso de MFA	**Definir requisitos de seguridad**		**Acceso Directo a Objetos**	
Diseño Abierto	Protección datos Sensibles	**Aprovechar *frameworks* y librerías de seguridad**		**Control de Acceso**	
Mecanismo Menos Común	Protecciones CSRF	**Acceso seguro a bases de datos**		**Secuencia de Comandos en Sitios Cruzados Cross Site Scripting (XSS)**	Validación de Datos
Aceptabilidad Psicológica	Prevenir XEE/JEE	**Codificar y escapar datos**		**Inyección SQL**	
Eslabón Más Débil	Evitar uso de determinados inputs	**Validar todas las entradas**		**Desbordamiento de Búfer**	
Aprovechar Componentes Existentes	Uso versión actualizada del lenguaje	**Implementar identidad digital**		**Falsificación de Registros**	

Uso Entorno Virtual	**Reforzar controles de acceso**		**SQL Dinámico**	
Importación correcta de Paquetes	**Proteger los datos continuamente**		**Vulnerabilidad de Redirección Abierta**	
Uso Formateo Cadenas Seguro	**Implementar monitorización de seguridad**		**Codificación de la Salida**	
Uso Seguro de Peticiones HTTP	**Manejar todos los errores y excepciones**		**Revelación de información**	
Paquetes instalados e importados Seguros			**Gestión Débil de la Sesión**	Gestión de la Sesión
Deserialización de Datos con Seguridad			**Caché de Formularios**	
Mantener Vulnerabilidades Actualizadas			**Divulgación de Información**	Tratamiento de Errores
Desactivar Depuración en Producción			**Divulgación de Información**	Registro
Escaneo de código			**Registro no Existe para Funciones Críticas**	
			Almacenamiento de información Sensible en Texto no Cifrado	Criptografía
			Criptografía Débil	
			Carga insegura de Archivos	Gestión Segura de Archivos
			Divulgación de Información	

centro criptológico nacional

centro criptológico nacional

www.ccn.cni.es

www.ccn-cert.cni.es

oc.ccn.cni.es


ccn-cert
centro criptológico nacional
CCN
centro criptológico nacional


PUBLICACIONES
Pd
DE DEFENSA


GOBIERNO
DE ESPAÑA
MINISTERIO
DE DEFENSA
SUBSECRETARÍA DE DEFENSA
SECRETARÍA GENERAL TÉCNICA
SUBDIRECCIÓN GENERAL
DE PUBLICACIONES
Y PATRIMONIO CULTURAL