- Oye, Luis:
- ¿Le pediste permiso al cliente para explotar la vulnerabilidad en producción?
- **NO, ¿Por qué?**
- Afuera está la Policía...

Ciclo de Vida de un software

1. ANALISIS:

En esta fase, se recopilan y documentan los requisitos del software a desarrollar. Esto implica entender las necesidades del cliente y definir claramente qué debe hacer el software.

2. DISEÑO

Los requisitos se utilizan para crear un diseño detallado del sistema. Esto incluye el diseño de la arquitectura, interfaces, bases de datos y otros componentes del software.

3. IMPLEMENTACIÓN

Durante esta fase, se escribe el código del software según el diseño elaborado. Los programadores desarrollan las funciones y características del software.

4. PRUEBAS

Después de la implementación, se realizan pruebas para asegurarse de que el software funcione correctamente y cumpla con los requisitos. Esto incluye pruebas de unidad, integración y sistema.

5. DESPLIEGUE

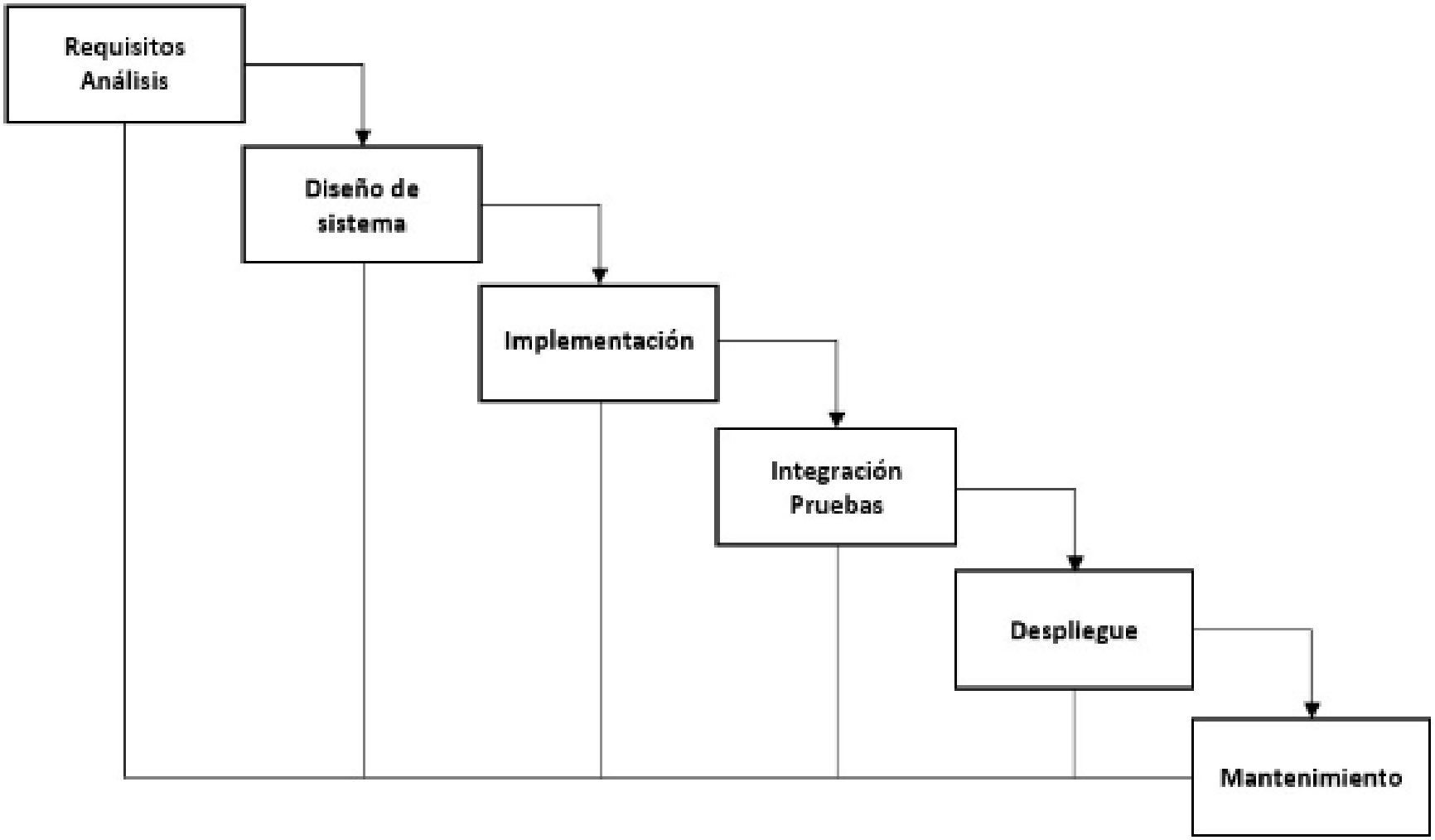
Una vez que el software ha pasado las pruebas y está listo para su uso, se implementa en el entorno de producción.

6. MANTENIMIENTO

Después del despliegue, se realiza el mantenimiento del software para corregir errores, agregar nuevas características y mejorar el rendimiento según las necesidades cambiantes del usuario.

Ciclo de Vida de un software

CASCADA



Ciclo de Vida de un software

ESPIRAL

Ciclo de Vida de un software

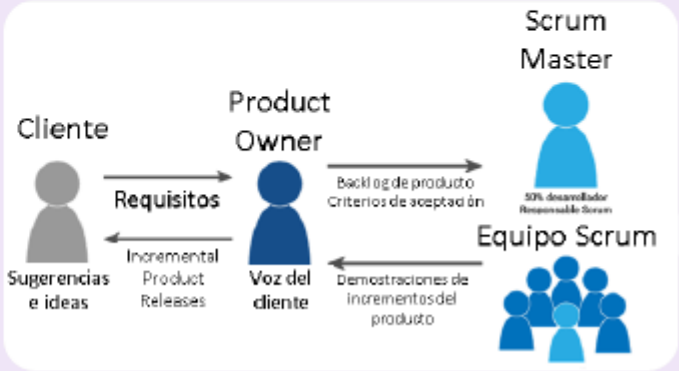
AGIL

¿Qué es?

Son metodologías basadas en el desarrollo iterativo e incremental, donde los requisitos y soluciones van evolucionando según las necesidades del proyecto [1]

SCRUM

Los equipos desarrollan los productos en **sprints**, permitiendo obtener feedback, adaptaciones y una mejora continua



DESARROLLO SEGURO

El software que se desarrolla teniendo en cuenta la seguridad, debe contar con una serie de propiedades a lo largo de su ciclo de vida (desde su desarrollo) que se interpreten y limiten con base a los requerimientos reales, tales como definir cuál es el nivel requerido de seguridad; cuáles son los aspectos más críticos a lo que se deben atender; y qué acciones resultan aptas para el costo y la programación del proyecto. (IATAC & DACS, 2007). De acuerdo con Castellaro et al (2009), estas propiedades se pueden dividir en dos conjuntos: fundamentales, las que conforman la base de una aplicación segura (seguridad de la información) y conducentes, aquellas que permiten caracterizar cuán seguro es el software y están influenciadas por el tamaño, la complejidad y la trazabilidad del software

CARACTERISTICAS

Confidencialidad:
El software debe garantizar que sus características, los activos de información que administra y su contenido sean accesibles sólo para las entidades autorizadas e inaccesibles para el resto.

Integridad: El software debe ser resistente y flexible a las modificaciones no autorizadas del código, los activos administrados, la configuración o el comportamiento por parte de entidades no autorizadas.

Disponibilidad: El software debe estar accesible siempre que sea requerido; y funcionar con un rendimiento adecuado para que los usuarios puedan realizar sus tareas en forma correcta y dar cumplimiento a los objetivos de la organización que lo utiliza.

Trazabilidad:
Todas las acciones relevantes efectuadas por los usuarios del software, se deben registrar con el fin de establecer responsabilidades.

No Repudio:
Prevenir que un usuario del software desmienta o niegue la responsabilidad de acciones que ha realizado. Asegura que no se pueda subvertir la propiedad de Trazabilidad.

Exactitud (Correcto):
Asegurar que el software siempre opere de la manera esperada sin defectos ni debilidades. La intencionalidad y el impacto resultante en caso de un ataque determinan si un defecto o una debilidad realmente constituyen una vulnerabilidad que incrementa el riesgo de la seguridad.

Previsibilidad:
Las funcionalidades, comportamientos y condiciones (ambiente, entradas) del software siempre serán los que se espera que sean; así, el software nunca se desviará de su operación correcta bajo condiciones previstas. Asimismo, esta propiedad se extiende a la operación bajo condiciones no previstas, en las que si los atacantes intentan explotar fallas en el software o en su ambiente no obtendrán respuesta positiva del mismo.

Confiabilidad:
Preservar la ejecución predecible y correcta del software a pesar de la existencia de defectos no intencionales y otras debilidades, y de cambios de estado no predecibles en el ambiente.

Protección:
Capacidad de que el software se detenga o quede parcialmente operativo en un estado seguro de tal forma que persista su confiabilidad y se eviten daños mayores como pérdida de activos valiosos.

Análisis de Riesgos

El análisis de riesgos tiene en cuenta los siguientes elementos:

- Activos: elementos del sistema de información que soportan la misión de la organización. Cualquier cosa que tenga valor para la organización. (NTC-ISO/IEC 27001).
- Amenazas: Causa potencial de un incidente no deseado, que puede ocasionar daño a un sistema o a la Organización. (CELSIA, 2013).
- Vulnerabilidad: Debilidad de un activo o grupo de activos, que puede ser aprovechada por una o más amenazas. (CELSIA, 2013)
- Controles: Medidas de protección desplegadas para que las amenazas no causen (tanto) daño al explotar las vulnerabilidades.

Análisis de Riesgos

Establecer el contexto permite determinar los parámetros y condicionantes externos e internos que orientan la política para gestionar los riesgos, en esta etapa cabe la identificación de los activos. La identificación de los riesgos busca una relación de las vulnerabilidades y amenazas partiendo de sus posibles causas.

El análisis de los riesgos busca calificar los riesgos identificados, cuantificando sus consecuencias (análisis cuantitativo) u ordenando su importancia relativa (análisis cualitativo). La evaluación de los riesgos es más profunda que el análisis y transforma las consecuencias a términos del negocio. Aquí se decide cuales riesgos se aceptan y cuáles no, así como en qué circunstancias se pueden o no aceptar un riesgo y/o trabajar en su tratamiento. El tratamiento de los riesgos busca modificar la situación del riesgo. En cuanto a la comunicación y consulta, se debe buscar el equilibrio entre la seguridad y el soporte a la productividad de la organización contando para ello con todas las partes interesadas (Stakeholders).

Entre tanto para el seguimiento y revisión es importante resaltar que se debe mantener en un entorno de crecimiento y mejora continua

Fuente:
https://alejandria.poligran.edu.co/bitstream/handle/10823/1040/Trabajo%20de%20grado.pdf?sequence=1&isAllowed=y

Amenazas de Ciberseguridad

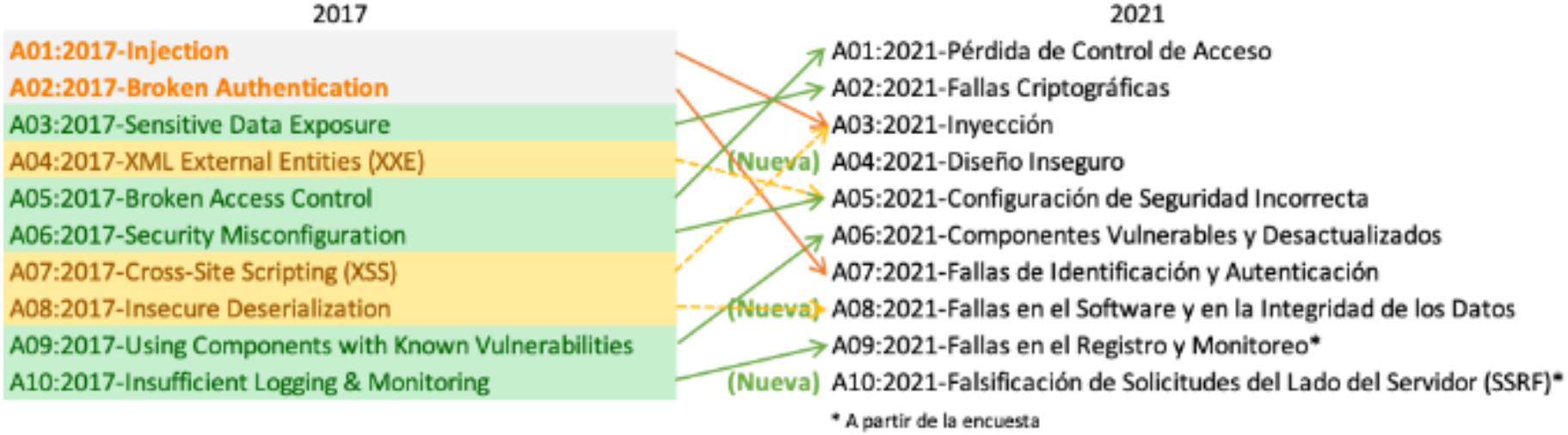
Ejemplos Amenazas de Ciberseguridad

La inyección de SQL es una vulnerabilidad en la cual un atacante puede manipular las consultas SQL de una aplicación enviando datos maliciosos a través de formularios web u otros canales de entrada. Esto puede llevar a la revelación de información confidencial o la eliminación no autorizada de datos en una base de datos.

La divulgación de información sensible ocurre cuando se exponen datos confidenciales, como contraseñas, claves de API o datos personales, a personas no autorizadas. Esto puede suceder debido a configuraciones incorrectas de permisos, errores de programación o fallos de seguridad en la aplicación.

La falta de validación de datos puede permitir que los usuarios ingresen datos maliciosos o incorrectos en una aplicación. Esto puede conducir a errores de procesamiento, ataques de inyección de código y otros problemas de seguridad.

OWASP TOP 10


2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Pérdida de Control de Acceso
A02:2021-Fallas Criptográficas
A03:2021-Inyección
(Nueva) A04:2021-Diseño Inseguro
A05:2021-Configuración de Seguridad Incorrecta
A06:2021-Componentes Vulnerables y Desactualizados
A07:2021-Fallas de Identificación y Autenticación
(Nueva) A08:2021-Fallas en el Software y en la Integridad de los Datos
A09:2021-Fallas en el Registro y Monitoreo*
(Nueva) A10:2021-Falsificación de Solicitudes del Lado del Servidor (SSRF)*
* A partir de la encuesta

Software Assurance Maturity Model (SAMM)

SAMM, integra prácticas de seguridad en cada fase de desarrollo, desde la planificación hasta el despliegue

Modelo de madurez que es una guía que ayuda a a evaluar, mejorar y construir procesos de seguridad en el ciclo de vida del desarrollo de software.

Fue creado por OWASP (Open Web Application Security Project), una comunidad dedicada a mejorar la seguridad del software.

Software Assurance Maturity Model (SAMM)

Figura 2. Bases Modelo SAMM. Fuente: Fundación OWASP (s.f.-b).

Cada función es una categoría de actividades

Las prácticas de seguridad son un área de actividades de seguridad que construyen las actividades de aseguramiento para las funciones de negocio con la que se relaciona

Gobierno:

Estrategias y métricas: Dirección estratégica global

Política y cumplimiento: Estructura de control y auditoria de seguridad y cumplimiento de regulaciones.

Educación y orientación: Aportar en conocimiento de seguridad de personal de desarrollo.

Construcción

Evaluación de Amenazas: identifica y caracteriza ataques potenciales para comprender mejor los riesgos y facilitar su gestión

Requisitos de seguridad: Incluye necesidades de seguridad durante el desarrollo de software.

Arquitectura de seguridad: Fortalecer proceso de diseño para promover diseños con seguridad y marcos de trabajo

Software Assurance Maturity Model (SAMM)

Verificación:

Revisión de diseño: inspeccionar artefactos para asegurar la provisión de mecanismos de seguridad.

Revisión de código: Evaluación de código fuente para identificar vulnerabilidades

Pruebas de seguridad: Probar software en ambiente de ejecución

Implementación:

Administración de vulnerabilidades: Administrar reportes internos o externos de vulnerabilidades

Fortalecimiento de Ambientes; Controlar ambiente operativo que rodea los programas para reforzar la seguridad de las aplicaciones implementadas.

Habilitación operativa: identificar y capturar información relevante a la seguridad que se necesitan para configurar, instalar y correr los programas.

Metodología de desarrollo seguro SDL (Secure Development Lifecycle

Enfoque estructurado para incorporar seguridad a las etapas del ciclo de vida del software. Creado por microsoft.

Componentes:

- Entrenamiento: Capacitar y concientizar en seguridad a todos los miembros sel equipo (buenas prácticas, identificación, prevención y comprensión de políticas y procedimientos.

- Requisitos: Identificar y documentar los requisitos de seguridad del proyecto.
  - Análisis de riesgos
  - Documentación de requisitos
  - Verificar implementación y cumplimiento de requisitos de seguridad.

- Diseño: Desarrollar una arquitectura segura y resistente a ataques, identifica y mitiga riesgos de seguridad.
  - Revisión de diseño de seguridad para identificar y mitigar posibles riesgos.
  - Utilizar patrones de diseño seguro (ejemplo mínimo privilegio)
  - Incorporación de controles de seguridad en la arquitectura del sistema (ejemplo autenticación, cifrado)

Metodología de desarrollo seguro SDL (Secure Development Lifecycle

- Implementación: Aplicar mejores prácticas de codificación segura (ejemplo validación de entradas, sanitización de datos, manejo de errores)
  - Herramientas de análisis de seguridad para identificar vulnerabilidades de código)
  - Aplicar buenas prácticas de codificación
  - Revisión de código por pares.

- Verificación: Pruebas para identificar y corregir vulnerabilidades (ejemplo: penetración, análisis de código)
  - Pruebas de penetración
  - Herramientas de análisis de código

- Liberación: Revisión final de software para verificar cumplimiento de requisitos de seguridad.

- Respuesta: Gestión de incidentes y respuesta (preparación para responder a futuros incidentes)
  - Desarrollo y documentación
  - Formación de personal de respuesta
  - Simulacros de respuesta

Las buenas prácticas de desarrollo seguro

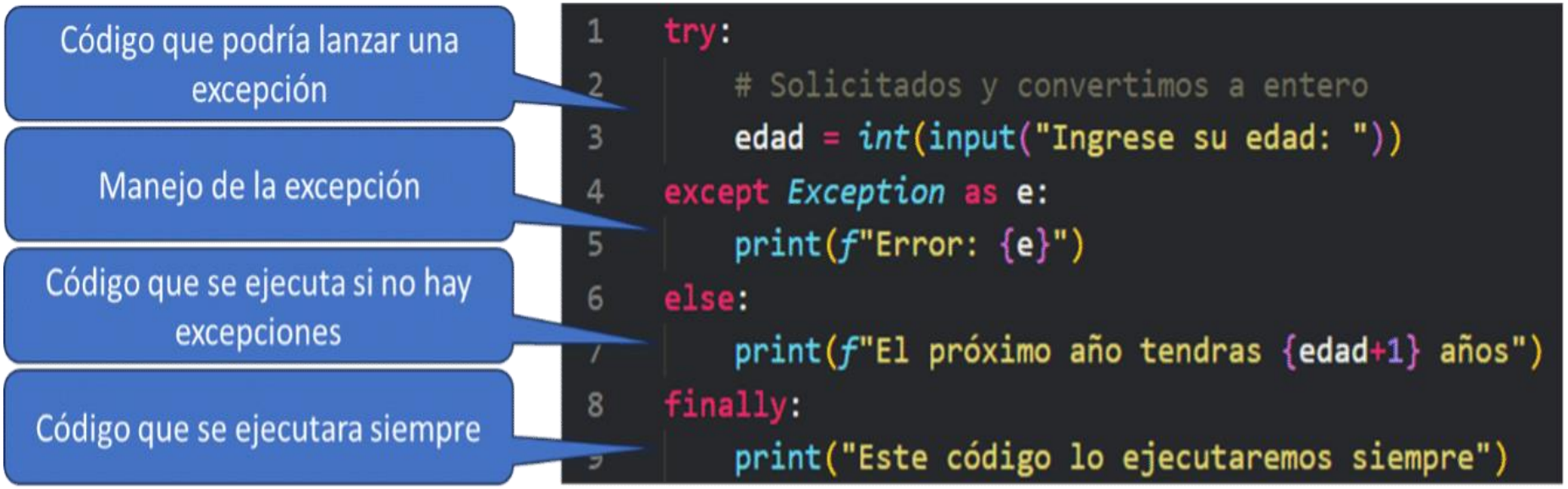
- Validación y sanitización de datos de entrada
- Autentificación y autorización sólidas que solo usuarios autorizados tengan acceso a los recursos protegidos . mecanismos sólidos de autenticación
- Protección de datos sensibles
- Prevenir accesos no autorizados y garantizar la privacidad de los usuarios. Cifrado de datos . algoritmos hash.
- Anonimización de datos. Eliminar información que pueda identificar a un individuo en un conjunto de datos: Agregar datos para ocultar valores individuales, encriptación o el reemplazo de valores con valores ficticios.
- Validación de datos: Validar estos datos para garantizar integridad y seguridad: Verificación de tipos de datos, rangos de valores.

MANEJO DE ERRORES

Los errores, fallas y excepciones son situaciones en las que el programa no puede continuar su ejecución normal debido a problemas en el código. Estos pueden ocurrir por diversos motivos, como errores de sintaxis, problemas lógicos o situaciones excepcionales durante la ejecución.

Existen excepciones, que son situaciones anormales que pueden ocurrir durante la ejecución de un programa, como divisiones entre cero, acceso a índices inválidos en listas, etc.

Python proporciona un sistema de manejo de excepciones para capturar y manejar estas situaciones, evitando que el programa se detenga abruptamente.




```
Ingrese su edad: treinta
Error: invalid literal for int() with base 10: 'treinta'
Este código lo ejecutaremos siempre
```

```
Ingrese su edad: 30
El próximo año tendras 31 años
Este código lo ejecutaremos siempre
```

Fuente: Recursos Inacap

Ejemplo excepciones:

TypeError : Operación o función de tipo inapropiado.

•ZeroDivisionError : Dividir por cero.

•OverflowError : Cálculo excede el límite para un tipo de dato numérico.

•IndexError : Acceder a una secuencia con un índice que no existe.

•KeyError : Acceder a un diccionario con una clave que no existe.

•FileNotFoundError : Acceder a un fichero que no existe.

•ImportError : Importación de un módulo que no existe.

APLICAR EJEMPLO A EXCEPCIONES