Tema 3

Un **usuario** rellena datos en un formulario puede intentar realizar **acciones no deseadas,** es fundamental **validar los datos** que cumplimenta el usuario antes de mandar las peticiones a las bases de datos de los servidores.

Servicios hiperescalares:

- Debe cumplir con las **normativas** de donde se **distribuye**
- **Ejemplos:** Google, Amazon o Azure

Web segura:

- Si es segura primero hay que entender **donde esta**, que **arquitectura** tiene y saber la **tecnología** que usa.
- Aplicar la **lógica del aplicativo** frente a su arquitectura para **encontrar vulnerabilidades** que estén basados en la arquitectura
- Ejemplos: U**na capa**/ **mala validación** de datos/C**uello de botella** en una BBDD

Estándares de Autenticación y Autorización

Se presenta el problema de la multitud de sistemas de autenticación de usuarios y de acceso a los recursos y datos personales.

•OAuth → Es un protocolo abierto, permite a los usuarios autorizar a aplicaciones de terceros a acceder a sus datos y recursos sin compartir sus credenciales.

•OpenID OAuth Hybrid Protocol → OpenID se encarga de la autenticación del usuario, mientras que OAuth se encarga de la autorización

•Facebook Connect (obsoleta) → Permitía a los usuarios iniciar sesión en aplicaciones de terceros con su cuenta de Facebook y fue relanzada por Facebook login

•OpenID Connect (Usa OAuth2.0) → Los sitios web y las aplicaciones móviles autentiquen a los usuarios y obtengan información de perfil de los proveedores de identidad.

Robo de sesión

Es la **explotación de una sesión** de ordenador válida (también llamada llave de sesión) para obtener **acceso no autorizado** a información o servicios en un sistema computacional.

En particular, suele referirse al **robo de una cookie** mágica utilizada para autenticar un usuario a un servidor remoto.

Las cookies de HTTP son utilizadas para mantener una sesión en muchos sitios web pueden ser fácilmente **robadas** por un atacante utilizando un ordenador de intermediario o al acceder a cookies guardadas en el ordenador de la víctima.

Vulnerabilidades Web

Un agujero de seguridad o vulnerabilidad es un **fallo en un sistema** de información que se puede **explotar para violar la seguridad** del sistema web.

Almacenamiento de contraseñas

Las **contraseñas** y otros **datos confidenciales** se tienen que **almacenar de forma segura**, es **evitar** el almacenamiento de **datos confidenciales que no nos sirvan** así evitamos disgustos.

Aplicar criptografía a los datos confidenciales, el uso de **hash** como criptografía **simétrica** o **asimétrica**.

Contramedidas

- **HTTP Strict Transport Security o HTTP con Seguridad de Transporte Estricta)**:
  - Es una **política de seguridad** web establecida para **evitar** ataques que puedan **interceptar comunicaciones**, **cookies**, etc.

- **Content Security Policy (CSP):**
  - Un **estándar** de seguridad informática introducido para **evitar secuencias de comandos** en sitios cruzados, **ataques de clic** y otros **ataques de inyección de código** resultantes de la ejecución de contenido malicioso en el contexto de la página web confiable.

- **CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart):**
  - P**ruebas desafío-respuesta** controladas por máquinas que son utilizadas para **determinar** cuándo el usuario es un **humano o un programa automático**.

- **Tesseract:**
  - Motor de reconocimiento óptico de caracteres de Google (OCR)

- **WAF (Web Application firewall).**
  - El WAF va antes del balanceador de carga y tiene la utilidad de dar seguridad, disponibilidad y monitorización

Nos basamos en…

- **PSP: (Personal Software Process):**
  - **Proceso** diseñado para **ayudar** a los **ingenieros** de software a **controlar**, **manejar** y **mejorar** su trabajo. PSP está basado en una motivación: La **calidad** de software **depende** del trabajo **de cada uno** de los ingenieros de software.

- **TSP:**
  - Se basa en PSP, y se fundamenta en que el **software**, en su mayoría, es **desarrollado por equipos**, por lo que los ingenieros de software **deben** primero saber **controlar su trabajo**, y después saber **trabajar en equipo**. ISO 27001 / ISO 27005

- **CMMI: (Capability Maturity Model Integration):**
  - Es un **“framework”**, un conjunto de **buenas prácticas** organizadas por capacidades críticas de negocio con el objetivo de **mejorar su rendimiento**.

Practicas

Requerimientos	Diseño	Desarrollo	Pruebas	Despliegue	Mantenimiento

N.º	Fase	Descripción	Fase ciclo SW
1	Monitorizar los parámetros de planificación del proyecto	Ciclo de desarrollo seguro, cuánto vale el trabajo, cronogramas, días, perfiles del equipo, el dinero, gestión general de proyecto	**Todas las fases**
2	Asegurar el alineamiento entre el trabajo del proyecto y los requisitos	Identificar inconsistencias y cambios que deberían realizarse en los planes y hacer las correcciones necesarias.	**Todas las fases**
3	Llevar a cabo las revisiones del progreso	Comunicar el estado de las actividades, documentar mediciones, cambios y resultados y gestionar peticiones de cambio. (Las primeras son más generales y luego son más específicas)	**Todas las fases**
4	Llevar a cabo las revisiones de hitos	Revisar hitos, realizar revisión de compromisos, identificar temas relevantes y su impacto, documentar los resultados y gestionar las acciones hasta su cierre	**Todas las fases**
5	Almacenar los datos y los resultados	Revisar Como se almacenan datos del aplicativo que vamos a implementar, como se accede al código.	**Todas las fases**
6	Separación de las instalaciones de desarrollo, ensayo y operación	Ver si se tienen entornos separados para desarrollo, pruebas y producción, con configuraciones para mantener la consistencia y evitar cambios no deseados en producción.	**Todas las fases**
7	Establecer un sistema de gestión de configuración	Definir metodología para diferentes niveles de gestión, implementar PAM, IAM	**Todas las fases**
8	Análisis y especificación de los requisitos de seguridad	Para nuevas implementaciones de sistemas o actualización se deben especificar los requerimientos de seguridad necesarios.	**Toma de requerimiento**
9	Integridad del mensaje	Identificar y establecer los requisitos para garantizar la veracidad e integridad de los mensajes en la aplicación, implementando los controles necesarios.	**Toma de requerimiento**
10	Gestión de privilegios	Se deben controlar y gestionar la asignación y uso de los privilegios de usuarios en el sistema.	**Diseño**

11	Registro de Usuarios	Se deben establecer procedimientos formales para el registro y eliminación de usuarios en los sistemas de información	**Diseño** **Desarrollo**
12	Controles contra códigos maliciosos	Establecimiento de controles para la identificación y prevención de ataques por código malicioso y procedimientos para su recuperación.	**Desarrollo**
13	Gestión del Cambio	Evidenciar si se tiene control de los cambios efectuados en los sistemas de procesamiento de información conservando y administrando log de cambios	**Desarrollo**
14	Control de procesamiento interno	Definir controles de validación para identificar cualquier corrupción de la información derivada de fallos en su procesamiento o acciones malintencionadas.	**Desarrollo**
15	Revisión personal del código fuente para encontrar defectos	Revisar el código personal para aprender a gestionar defectos. PSP (Individuo) y TSP (grupo).	**Desarrollo**
16	Validación de los datos de entrada	Validación de la correctitud e integridad de la información ingresada en la aplicación. Sanitizado el código de entrada evitando malas prácticas, programando triggers	**Desarrollo** **Pruebas**
17	Validación de los datos de salida	Se deben validar los datos de salida de la aplicación, para garantizar que el procesamiento de la información almacenada en la aplicación sea correcto	**Desarrollo** **Pruebas**
18	Aceptación del sistema	Se deben establecer criterios de aceptación de nuevas actualizaciones existentes y realizar las respectivas pruebas de funcionamiento durante su desarrollo, previo a su aprobación.	**Desarrollo** **Pruebas** **Despliegue** **Mantenimiento**
19	Protección de los datos de prueba del sistema	Los datos de prueba se deben pasar por un proceso idóneo de selección según las especificaciones del sistema, estos datos deben ser controlados y asegurados.	**Pruebas**
20	Perfil de calidad	Mide los datos de proceso de un módulo contra los estándares de calidad, a través de cinco dimensiones (datos para el diseño, revisiones de diseño, revisión de código, defectos de compilación, defectos de prueba de unidad) para identificar problemas de calidad.	**Despliegue** **Mantenimiento**

Tema 4

Definiciones

- **RiskTool:** Malware que daño tanto software como hardware embebido en app móvil (símil a un troyano)
- **Torjan-Dropper:** Malware que elude detecciones para que las herramientas de seguridad y las stores no lo denuncien
- **AdWare:** Malware introduce publicidad, a veces, maliciosa
- **Trojan-Banker:** Malware especializado en el robo de datos de carácter bancario
- **Trojan-SMS:** Malware sectorial con el objetivo
- **Trojan-Ransom:** Malware que tiene como objetivo bloqueo y cifrado de información. Puede acarrear extorsión.
- **Trojan-Spy:** Malware que tiene como objetivo monitorizar al usuario
- **Backdoor:** Malware que tiene como objetivo tomar el control del dispositivo, p.ej. Spade.
- **APK (Android Application Package):** Se trata de un archivo ejecutable que contiene todos los datos que se necesitan para instalar y hacer funcionar una aplicación Android.
- **SandBox:** Es una simulación virtual de un ordenador que se utiliza con el fin de ejecutar un malware y observar cómo se comporta allí, sin poner en riesgo de infección a la máquina anfitriona.
- **EOL: End of life**: Es el tiempo cuando un producto o software llega al final de su vida útil o de soporte

Componentes de un APK

- **META-INF:** En este directorio nos encontramos varios archivos:
  - **MANIFEST.MF:** Es el manifiesto de la aplicación.
  - **CERT.RSA:** Se trata del certificado de la aplicación.
  - **CERT.SF:** Se trata de una lista de recursos necesarios y su clave sha-1

- **LIB:** En este directorio nos encontramos código compilado específico del software o procesador que contiene directorios como armeabi, x86 o mips entre otros.

- **RES:** Directorio que contiene recursos no compilados en resources.arsc

- **ASSETS:** Contiene recursos de aplicaciones, los assets de la aplicación.

- **AndroidManifest.xml:** Manifiesto adicional, archivo que contiene en formato .xml la información de las clases, versión, nombre, librerías o derechos de acceso.

- **Classes.dex:** Contiene las clases compiladas, el código java, en formato dex que puede ser interpretado por la máquina virtual Dakvik o bien por Android Runtime.

- **Resources.arsc:** Contiene los recursos precompilados.

Tema 5

Planificación

Se centra en **identificar el alcance** del proyecto **y sus requisitos** para su posterior análisis.

El objetivo de esta etapa consiste en entender el **propósito del proyecto** y las **necesidades del cliente** y conocer sus resultados deseados.

Se determinarán **objetivos SMART**:

- Específicos (Specific)
- Medibles (Measurable)
- Alcanzables (Achievable)
- Realistas (Realistic)
- De duración limitada (Time-bound).

Diseño

Se determinarán los pasos a seguir para alcanzar ese objetivo - **el "cómo" de completar un proyecto.** Se elaborará un modelo abstracto del sistema a construir basado en los requerimientos planteados en la fase anterior.

Proporciona **detalles sobre el software a desarrollar**, incluyendo las **arquitecturas** del sistema, las **estructuras de datos**, el **diseño de la interfaz de usuario** y otros aspectos técnicos necesarios para implementar el sistema.

Se establecerá una **asignación de costes, plazos e hitos, y materiales y documentación necesarios.**

Esta etapa también implica el cálculo y la **previsión de riesgos**, puesta en marcha de procesos de cambio y definición de protocolos.

Implementación

Puesta en marcha del producto de software, es decir, donde se generará el código fuente en el lenguaje de programación escogido adecuado a nuestro proyecto.

Pruebas

Consiste en **testear el diseño implementado**.

En esta etapa ponemos a **prueba los errores** que hayan podido aparecer en las etapas anteriores. Es una fase de **corrección, eliminación y perfeccionamiento de posibles fallos**, la ejecución de los siguientes procesos: (Pruebas unitarias/integración/sistema/aceptación).

Despliegue

Procedemos a su **implementación en el entorno deseado**, ya sea un **entorno pre-productivo o productivo**. Esto dependerá de la casuística del cliente y asociado a ella existirá un flujo de trabajo.

Mantenimiento

El software ya está en **funcionamiento**. Con el tiempo alguna función puede quedar obsoleta, implica realizar correcciones de errores, **actualizaciones** periódicas del software, o **implementación y despliegue de mejoras.**

Importancia DevOps

La metodología DevOps es un enfoque de desarrollo de software que busca **integrar la colaboración y comunicación** entre los equipos de **desarrollo** (Dev) y **operaciones** (Ops) para lograr una entrega de **software más rápida, eficiente y confiable.**

Mejoras

Colaboración: Se hace **partícipe** a todo el **equipo** en cada una de las **etapas**.

Automatización: **S**e busca **eliminar la acción humana en procesos repetitivos** con ello conseguimos que la **confiabilidad en procesos críticos**

Integración continua (CI): tiene como objetivo **automatizar y facilitar la integración** frecuente de cambios **en el código fuente** de un proyecto en un repositorio compartido. En esencia, busca garantizar que el **código nuevo** o modificado se **incorpore** al código existente **de manera regular**, para que los **errores**, tanto de calidad de código como de **seguridad**, y posibles **conflictos** puedan **detectarse y solucionarse rápidamente**.

Entrega continua (CD): A**mplía el concepto de integración continua** al **automatizar** el **despliegue** de las aplicaciones en el **entorno deseado**, una vez **pasadas las pruebas** satisfactoriamente. En caso de que estas **pruebas sean negativas**, el proceso de entrega continua **quedará bloqueado** hasta que estos resultados cumplan con los requisitos marcados. Este aspecto es fundamental para lograr un **control exhaustivo sobre el código**

Escalabilidad y flexibilidad: la **metodología DevOps** se adapta bien a entornos de desarrollo **ágiles** y sistemas distribuidos permitiendo la **adaptabilidad según necesidades** y el despliegue **automatizado** de aplicaciones en entornos complejos.

Monitorización: DevOps promueve la **observabilidad** y **revisión constante** del rendimiento de las **aplicaciones** y los **servidores** donde se encuentran alojadas. El objetivo de este proceso es **recoger toda la información** posible del sistema para poder **adelantarse** a la aparición de futuros **problemas reales**.

Herramientas

Cada empresa puede utilizar una combinación de herramientas opensource/empresariales que den cobertura a todo el ciclo de vida de desarrollo del código, entre las más comúnmente nos encontramos:

- **Herramienta de ticketing**: I**nventario de tareas**, las cuales describan de manera clara las diferentes **acciones a realizar y el estado** en el que se encuentran.

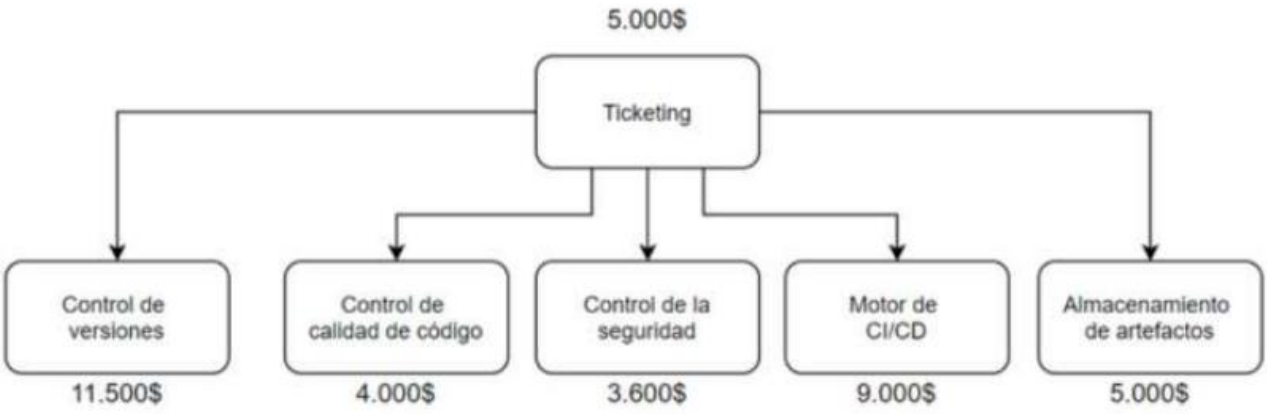
- **Control de versiones de código**: Herramienta esencial en el desarrollo de software, su función principal es **rastrear y administrar las diferentes versiones de archivos y código fuente** a lo largo del tiempo.

- **Control de calidad de código:** Diseñado para **analizar y evaluar el código fuente de un programa** de software con el **objetivo de identificar problemas**, aplicar **buenas prácticas** de programación y mejorar la **calidad** general del código.

- **Control de la seguridad:** soluciones diseñadas para **proteger sistemas informáticos, redes, aplicaciones y datos** contra **amenazas** y **vulnerabilidades**.

- **Motor de CI/CD:** Herramienta o plataforma que **automatiza y facilita el proceso de integración, pruebas y entrega** de código en el desarrollo de software.

- **Almacenamiento de artefactos:** **Repositorio centralizado donde se almacenan y gestionan los componentes de software** generados durante el proceso de desarrollo, compilación y construcción de un proyecto



Metodologías de Desarrollo Seguro de Software SDLC

- **Desarrollo Seguro de Software (SSD):** El enfoque SSD coloca **la seguridad como un requisito central en todas las fases** del SDLC. Desde el diseño hasta la implementación y el mantenimiento, se **busca identificar, mitigar y prevenir vulnerabilidades**.

- **Ciclo de Vida de Desarrollo de Seguridad (SecDevOps):** La fusión de las **prácticas DevOps con la seguridad** da lugar al SecDevOps. Esta metodología **incorpora** herramientas y prácticas de **seguridad** en el ciclo de **desarrollo continuo**, incluyendo automatización de pruebas y retroalimentación rápida.

- **Desarrollo Seguro por Diseño (SSDLC):** Se enfoca en integrar la **seguridad desde la etapa de diseño**, incorporando controles de seguridad **en los requisitos y la arquitectura del software**

- **Modelo de Madurez de Seguridad (SMM): Evalúa** y mejora la madurez de **la seguridad en el entorno de desarrollo.** Proporciona un marco para identificar el **estado actual de seguridad** y **establecer metas** de mejora.

- **Metodologías Ágiles y Seguridad (Agile Security):** Las **metodologías ágiles,** como Scrum o Kanban, se combinan **con enfoques de seguridad**. Esto se logra mediante **revisiones continuas** que contemplan el **análisis de seguridad** de código, las pruebas de **penetración** y los ajustes en cada **iteración del desarrollo.**

- **Análisis de Amenazas y Modelado (TAM):** El TAM **identifica amenazas** y riesgos desde las **etapas iniciales** del desarrollo mediante técnicas como **análisis de riesgos** y modelado de amenazas.

Claves para adoptar una metodología

- **Flexibilidad y Adaptabilidad**
- **Evaluación Continua y Mejora:** Deben **estar preparados para adaptarse a las amenazas emergentes y los cambios en los requisitos de seguridad.**
- **Educación y Conciencia en Seguridad:** Además de las metodologías, la educación y la conciencia en seguridad son vitales. Los equipos de desarrollo deben **estar capacitados en prácticas de desarrollo seguro y concientizados sobre las últimas amenazas** y vulnerabilidades
- **Colaboración y Comunicación**
- **Adopción de Cultura de Seguridad: La seguridad no debe ser considerada como una capa adicional, sino como un componente inherente** en todas las actividades de desarrollo.
- **Evolución Constante: Mantenerse al día con las tendencias y tecnologías emergentes es esencial** para proteger el software contra las amenazas en evolución.

Puntos clave / Los 10 mandamientos

1. Ningún componente es confiable hasta demostrar lo contrario (Zero trust)

2. Delinear mecanismos de autenticación difíciles de eludir (doble factor, certificados)

3. Autorizar, además de autenticar (El DevOps es el administrador, es el que autoriza los permisos al equipo)

4. Separar datos de instrucciones de control (Hardcoding)

5. Validar todos los datos explícitamente (validar los datos sensibles, que tienen valor, un algoritmo que implementes en una base de datos)

6. Utilizar criptografía correctamente (usar algoritmos no deprecados)

7. Identificar datos sensibles y cómo se los debería gestionar (con que módulos trabajan esos datos sensibles y como lo vas a proteger)

8. Considerar siempre a los usuarios del sistema (UX)

9. La integración de componentes cambia la superficie de ataque (cuantos más componentes más peligrosidad o menos, pero cambia)

10. Considerar cambios futuros en objetos y actores (cambios de programadores o del equipo)

Principios básicos

- Partir siempre de un **modelo de permisos mínimos**, es mejor ir escalando privilegios por demanda de acuerdo a los perfiles establecidos en las etapas de diseño.
- **Nunca confiar en los datos que ingresan a la aplicación, todo debe ser verificado**
- Si se utiliza un lenguaje que no sea compilado, asegurarse de **limpiar el código que se pone en producción**
- **Hacer un seguimiento de las tecnologías utilizadas para el desarrollo**.
- **Todos los accesos que se hagan a los sistemas deben ser validados.**
- Intercambio de información sensible, utilizar protocolos **para cifrar las comunicaciones**, y en el caso de almacenamiento **la información confidencial debería estar cifrada utilizando algoritmos fuertes y claves robustas**
- Cualquier funcionalidad **nueva debe agregarse de acuerdo a los requerimientos de diseño.**
- **La información almacenada en dispositivos móviles debería ser la mínima**
- **Cualquier cambio que se haga debería quedar documentado**
- **Poner más cuidado en los puntos más vulnerables**, no hay que olvidar que el nivel máximo de seguridad viene dado por el punto más débil.

Reducción de superficie de ataque

- **Servicios SAST** (Static Application Security Testing) /SCA (Software composition análisis).
- **Identificación de vulnerabilidades**
- **Análisis DAST** (Dynamic Application Security Testing)
- Ejecución de **auditorías de seguridad** para la identificación de **posibles vulnerabilidades**, facilitando la ejecución del framework de desarrollo seguro de software.