Prueba de aplicaciones web y para dispositivos móviles

Caso práctico

StockSnap. *Portatil, programación código* (Licencia de Pixabay)

Julián es un joven programador que se ha incorporado a una empresa de desarrollo de software. Conoce varios lenguajes de programación, pero se siente más cómodo con PHP, si bien ha tenido experiencia programando algunas aplicaciones con C y C++.

Sabe que muchas veces los requisitos del cliente respecto al software condicionarán el lenguaje elegido, ya sea porque se necesita tener el software cuanto antes, necesite tener un alto rendimiento o tenga que comunicarse con alguna otra pieza de software o hardware y, por tanto, deba de ir en un lenguaje concreto. Es decir, es posible que, por requisitos técnicos, tenga que abandonar su lenguaje favorito y adaptarse a las necesidades del proyecto.

Julián, en uno de sus primeros proyectos recibe el encargo de desarrollar una aplicación bastante sencilla donde el rendimiento no es relevante y tiene dudas sobre qué lenguaje de programación usar, ya que, según el tipo que sea, tiene unas ventajas e inconvenientes. Valorando los tipos de lenguaje se encuentra con lo siguiente:

- **Rendimiento**: un lenguaje de programación compilado tiene mayor rendimiento que un lenguaje de programación interpretado, pero sólo funciona para la plataforma para la que se ha compilado.
- **Facilidad:** los lenguajes de programación interpretado, por lo general, son más sencillos de programar en ellos debido a un mayor nivel de abstracción.

Finalmente decide hacer el programa en PHP, que es un lenguaje de programación interpretado. Sabe que programar en PHP será más sencillo y por tanto le requerirá menos tiempo de programación, pero a cambio necesitará que la máquina donde se ejecute tenga el intérprete de PHP instalado. Como la aplicación es bastante sencilla y el rendimiento no es prioritario a Julián no le importa las desventajas en cuanto a rendimiento de un lenguaje interpretado.

En un mundo que ha sufrido un importante proceso de transformación digital cada vez hay más aplicativos desarrollados con el fin de interactuar con los seres humanos, si estos aplicativos no están correctamente programados un atacante puede hacer uso mal intencionado del software y controlar el flujo de ejecución, pudiendo alterar la información y/o robarla.

Antes de nada, haremos una distinción fundamental entre hardware y software

- **Hardware**: todos los componentes físicos de un ordenador
- **Software**: programas e instrucciones para ejecutar tareas en un ordenador

Una vez tenemos claro la diferencia, en este curso nos centraremos en el software, y en una de sus unidades fundamentales: los programas

Definiríamos **programa** como:

1.- Secuencia de instrucciones
2.- Entendible por el ordenador
3.- Tienen un objetivo o tarea concreta

Boskampi (Licencia de Pixabay)

En esta unidad estudiaremos:

- Fundamentos básicos de la programación.
- La diferencia entre lenguajes de programación interpretados y compilados.
- Código fuente y entornos de desarrollo.
- Modos de ejecución de software.
- Los principales elementos dentro de un programa.
- Pruebas de ejecución de software.
- Seguridad en los lenguajes de programación y sus entornos de ejecución ("sandboxes").

1.- Fundamentos de Programación

Los ordenadores son máquinas eléctricas que sólo entienden de 0 y 1, siendo:

- 0 no hay corriente
- 1 hay corriente

Este es el único lenguaje que entiende el ordenador, llamado **lenguaje binario**.

geralt (Licencia de Pixabay)

Un **programa** es un conjunto de instrucciones que se crea para realizar una tarea específica en una computadora. Estas instrucciones son escritas en un lenguaje de programación y se traducen a un lenguaje que la computadora puede entender y ejecutar. Los programas pueden ser simples, como una calculadora, o muy complejos, como un sistema operativo. En esencia, un programa es como una receta que le dice a la computadora qué hacer y cómo hacerlo. Los programadores crean programas para automatizar tareas, procesar datos, resolver problemas y realizar una amplia variedad de funciones en la computadora.

Como ya hemos dicho para crear un programa y que la computadora lo interprete y ejecute, las instrucciones deben escribirse en un lenguaje de programación. En los primeros tiempos de la computación se programaba directamente en **código máquina**. Escribir programas así resultaba demasiado complicado, también era difícil entenderlos y mantenerlos una vez escritos. Con el tiempo, se fueron desarrollando herramientas para facilitar el trabajo y aparecieron los lenguajes de programación, que podemos dividir en dos grupos:

- **Lenguajes de bajo nivel**: son los más cercanos al binario, pero son difíciles de programar
- **Lenguajes de alto nivel**: necesitan ser traducidos antes de llegar al ordenador, pero son fáciles de programar ya que son más cercanos al lenguaje natural de las personas

Algunos ejemplos:

- **Lenguajes bajo nivel**: lenguaje máquina
- **Lenguajes alto nivel**: C, C++, Java, PHP, Python, Go, Rust, Ruby

En el desarrollo de software hay muchas condicionantes que marcan que elección de lenguaje se usará en el proyecto:

- Requisitos técnicos marcados por otras piezas de software/hardware
- Tiempo de entrega del software
- Necesidad de rendimiento

Programar viene a ser el proceso de crear un software fiable mediante la escritura en un lenguaje (**code** en inglés), prueba (**test**), depuración (**debug**), compilación o interpretación, y mantenimiento (**maintenance**) del código fuente de dicho programa informático.

CODIFICAR ⇒ PROBAR ⇒ DEPURAR ⇒ COMPILAR/INTERPRETAR ⇒ MANTENER (y vuelta a empezar)

Cada cierto tiempo (hoy en día puede que incluso cada hora) el código debe ser modificado para añadir o corregir alguna funcionalidad y de nuevo se deben hacer los mismos pasos, por eso es tan importante desde el punto de vista de la seguridad automatizar todo lo que se pueda del proceso para evitar vulnerabilidades y errores. En la unidad última se verán técnicas y herramientas para poder **automatizar** todo el proceso.

Debes conocer

Un **algoritmo** es una secuencia ordenada de pasos o reglas que se siguen para resolver un problema o realizar una tarea. Los algoritmos son independientes de cualquier lenguaje de programación específico y son la base de la programación. Los programadores utilizan algoritmos para diseñar la lógica subyacente de un programa antes de implementarla en código. Los algoritmos son como planes detallados que describen el proceso paso a paso para alcanzar un objetivo. Pueden ser representados de diversas formas, como diagramas de flujo, seudocódigo o simplemente una descripción en lenguaje natural. Un buen algoritmo es eficiente y garantiza que la tarea se realice de manera precisa y en un tiempo razonable.

Un **programa** es la implementación concreta de un algoritmo en un lenguaje de programación específico, mientras que un algoritmo es una serie de pasos lógicos que se diseñan previamente para resolver un problema o realizar una tarea.

2.- Modelos de Ejecución Software

Caso práctico

Julian sabe que durante su carrera profesional va a trabajar con diferentes lenguajes, que tienen diferentes modos de ejecución, pero dependiendo de las necesidades de los proyectos pueden ser más o menos óptimos.

StockSnap (Licencia de Pixabay)

El trabajo de **Julián** se centra en el desarrollo y puesta en producción segura de aplicaciones web y móviles por lo tanto ha investigado como funcionan estas aplicaciones para entender mejor como protegerlas. Tras esta investigación esta empezando a darse cuenta que al desarrollar las aplicaciones web y móviles se enfrenta una serie de desafíos y problemáticas en el ámbito de la seguridad.

Podemos clasificar los lenguajes de programación, según su modo de ejecución, en 3 tipos distintos:

Lenguaje compilado

- Nuestro código fuente se transforma en un binario mediante compilación
- El ordenador ejecuta el binario
- No necesitamos por tanto ningún programa adicional

Lenguaje interpretado

- Nuestro código fuente es leído en tiempo real por un programa (intérprete) que lo traduce a código máquina (objeto binario)
- El ordenador ejecuta ese binario
- Necesita por tanto un programa adicional, llamado intérprete, que hace de traductor entre las instrucciones y el código máquina

Lenguaje intermedio

- El código fuente se compila a un lenguaje intermedio
- Este lenguaje intermedio se ejecuta en una máquina virtual

Debes conocer

Un **compilador** no es más que un software que se encarga de transformar código fuente en lenguaje binario que es más fácil de entender por los ordenadores. Cuando compilamos un programa elegimos el tipo de plataforma y arquitectura en la que queremos que el binario funcione según el compilador que estemos usando.

Hay lenguajes como **Golang** que permiten hacer un compilado multiplataforma especificando sobre qué plataforma y arquitectura queremos que nuestro binario funcione cuando lo estamos compilando.

Para saber más

Dos de los lenguajes de programación más utilizados a nivel mundial son:

Python se ha convertido en uno de los lenguajes más usados a nivel mundial para el desarrollo de software. Es un lenguaje de alto nivel de programación interpretado multiparadigma y multiplataforma. Si bien no tiene el rendimiento que podrían tener lenguajes como C o C++ es sencillo de programar, tiene una sintaxis clara, tiene numerosas librerías disponibles y, además, ¡resulta muy divertido programar en Python!

Si te interesa existe un "zen" o mejores prácticas de Python que seguro te resultará interesante. Aquí el link

PHP es un lenguaje de programación ampliamente utilizado en el desarrollo web. Su flexibilidad y potencial para crear aplicaciones web dinámicas lo convierten en una herramienta esencial. Para garantizar que tu código sea eficiente y seguro, es crucial seguir las mejores prácticas de programación en PHP. Consulta este enlace para obtener directrices técnicas sobre cómo escribir un código PHP óptimo y seguro.

Autoevaluación

Identifica si las siguientes frases son verdaderas o falsas:

1. Python es un lenguaje compilado de alto rendimiento.

○ Verdadero ○ Falso

Falso

Python es un lenguaje que necesita un intérprete, que hace un trabajo de compilado en tiempo real, es decir es un lenguaje NO compilado y por tanto su rendimiento será menor.

2. Ruby es un lenguaje interpretado, es decir necesita un intérprete y por tanto NO es compilado.

○ Verdadero ○ Falso

Verdadero

Los lenguajes interpretados tienen, valga la redundancia, un intérprete que los compila y ejecuta en tiempo real. Es decir no ha sufrido un proceso de compilación previo.

3. Golang es un lenguaje compilado y por tanto si rendimiento será mayor.

○ Verdadero ○ Falso

Verdadero

Golang o go es un lenguaje que esta adquiriendo gran popularidad porque es sencillo de programar como Python pero tiene el rendimiento de un lenguaje compilado.

4. Scala es un lenguaje intermedio puesto que el código se compila a un lenguaje intermedio para más tarde ser ejecutado en un entorno controlado o máquina virtual.

○ Verdadero ○ Falso

Verdadero

Los lenguajes intermedios sufren de un proceso de compilación y ejecución en un entorno virtual o controlado.

5. PHP es un lenguaje de alto nivel muy habitual en las aplicaciones web que se interpreta en el navegador.

○ Verdadero ○ Falso

Falso

PHP es un lenguaje de alto nivel muy habitual en las aplicaciones web pero para la parte backend. El lenguaje que los navegadores interpretan es Javascript.

2.1.- Aplicaciones web y móviles

Habitualmente, tanto en las aplicaciones web como en las aplicaciones móviles (apps), existe una parte de la aplicación que se ejecutará en el cliente (**frontend**) y otra parte que se ejecutará en el servidor (**backend**). Cada parte tiene sus lenguajes específicos y, en relación con la seguridad, su problemática.

Aplicaciones web

En una aplicación web el cliente es la interfaz de usuario, típicamente un navegador web, que solicita información (páginas html, archivos css, archivos javascript, datos…) al servidor y presenta información al usuario. La parte servidora es una aplicación que almacena, procesa y entrega datos al cliente. Estos componentes se comunican utilizando diferentes protocolos, siendo el más habitual el Protocolo de Transferencia de Hipertexto (HTTP)

- El **navegador web** es una aplicación de software que actúa como un **intérprete de protocolos web y lenguajes de marcado**, como **HTML, CSS** y **JavaScript**. Su función principal es solicitar, recibir y procesar recursos en línea, interpretando y renderizando páginas web y otros contenidos. Los navegadores interpretan el código HTML de una página web para representar su estructura y contenido visualmente, aplican hojas de estilo (CSS) para dar formato y diseño, y ejecutan código JavaScript para habilitar la interactividad y el dinamismo. Además, gestionan cookies, sesiones de usuario y almacenan en caché recursos para acelerar la carga de páginas subsiguientes. En última instancia, un navegador permite a los usuarios interactuar con la World Wide Web, mostrando el contenido web de manera legible y permitiendo la navegación y la ejecución de aplicaciones en línea.
- La **parte servidora** almacena la lógica de la aplicación, la base de datos y los recursos. Recibe solicitudes del cliente, gestiona la sesión, procesa datos, realiza operaciones en la base de datos y envía respuestas al cliente. Además, pueden hacer uso de otros servicios ubicados en otros servidores. Pueden estar desarrolladas con diferentes lenguajes, siendo los más utilizados PHP, Java, C#, Python, NodeJS, Ruby…

Aplicaciones móviles

Una aplicación móvil, comúnmente conocida como "app", es un software diseñado específicamente para ser ejecutado en dispositivos móviles, como teléfonos inteligentes y tabletas. Estas aplicaciones están optimizadas para funcionar en las limitaciones de hardware y tamaño de pantalla de los dispositivos móviles.

- La parte que se ejecuta en el dispositivo se desarrolla en lenguajes de programación apropiados para las plataformas móviles, como iOS (utilizando Swift o Objective-C) o Android (utilizando Java o Kotlin).

- La parte servidora es similar a cualquier aplicación web.

Hoy en día es muy habitual construir las aplicaciones web y móviles utilizando **APIs** (Interfaz de Programación de Aplicaciones web, de sus siglas en ingles Application Programming Interface):

- Son un conjunto de reglas y protocolos que permiten la comunicación y la interacción entre sistemas de software a través de Internet.
- Están diseñadas para que una aplicación pueda solicitar y acceder a datos o servicios de otra aplicación o sistema, generalmente a través de solicitudes **HTTP** (Protocolo de Transferencia de Hipertexto por sus siglas en inglés), que es un protocolo de comunicaciones que se utiliza para la transmisión de información en la World Wide Web).
- Los datos se suelen transmitir en formatos estándar como **JSON** (Notación de Objetos JavaScript por sus siglas en inglés JavaScript Object Notation) o **XML** (Lenguaje de Marcado Extensible por sus siglas en inglés eXtensible Markup Language). Son esenciales para la integración de sistemas, permitiendo que aplicaciones y servicios se comuniquen de manera estructurada y segura a través de la web, lo que facilita el intercambio de información y la construcción de aplicaciones más completas.

A nivel de seguridad, la problemática en el desarrollo de aplicaciones web y móviles abarca diversos desafíos que los equipos de desarrollo deben enfrentar para crear sistemas efectivos, eficientes y seguros:

- En primer lugar, la diversidad de dispositivos, navegadores y sistemas operativos utilizados por los usuarios introduce vulnerabilidades potenciales, ya que es necesario garantizar la compatibilidad y seguridad en una amplia gama de plataformas.
- Además, la creciente dependencia de servicios en la nube y la comunicación constante entre dispositivos a través de redes públicas aumenta la exposición a amenazas como ataques de robo de sesión. La gestión de contraseñas y la autenticación segura son preocupaciones adicionales, dado que las violaciones de datos y la filtración de credenciales son riesgos comunes.
- Por último, la rápida evolución tecnológica y las actualizaciones frecuentes de las aplicaciones también presentan desafíos para mantener la seguridad de manera constante. Los profesionales de seguridad deben abordar estos problemas de manera proactiva para proteger la privacidad y los datos de los usuarios en un entorno en constante cambio.

Debes conocer

Tanto las aplicaciones móviles como las aplicaciones web comparten similitudes y diferencias significativas en términos de ciberseguridad. En cuanto a las similitudes, ambas pueden ser vulnerables a amenazas comunes, como ataques de inyección (por ejemplo, SQL injection), ataques de denegación de servicio (DDoS), y vulnerabilidades en la gestión de sesiones. Además, ambas requieren medidas de seguridad, como la autenticación y la autorización, para proteger los datos y las funciones críticas.

Sin embargo, también existen diferencias clave. Las aplicaciones móviles, al estar instaladas directamente en el dispositivo del usuario, pueden enfrentar desafíos específicos, como la ingeniería inversa de la aplicación para descubrir

vulnerabilidades o el robo de datos almacenados en el dispositivo. Por otro lado, las aplicaciones web son accesibles a través de navegadores y se ejecutan en servidores remotos, lo que las hace vulnerables a amenazas web específicas, como ataques de Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF).

3.- Elementos básicos

Caso práctico

thedigitalartist. *Seguridad e internet* (Licencia de Pixabay)

Julián ha trabajado ya con varios lenguajes y sabe que cuando tiene que aprender uno nuevo debe conocer los elementos esenciales del mismo junto con sus reglas. La comprensión de los elementos de un lenguaje de programación es esencial para los profesionales en el campo de la ciberseguridad y, en general, para todos los programadores. Un lenguaje de programación permite a los desarrolladores crear sus aplicaciones, pero también representa un terreno clave para la detección y mitigación de posibles vulnerabilidades. Al entender los elementos fundamentales, como variables, estructuras de control, funciones y manejo de errores, los expertos en ciberseguridad pueden evaluar y fortalecer la seguridad de las aplicaciones y sistemas. La familiaridad con la sintaxis y la semántica de un lenguaje les permite identificar posibles puntos débiles, prevenir errores comunes y desarrollar estrategias efectivas para proteger el código fuente contra amenazas potenciales.

Como ejemplo de lo anterior, en el contexto de la ciberseguridad, el control de excepciones es fundamental para garantizar la integridad, confidencialidad y disponibilidad de los sistemas de información. Es esencial desde una perspectiva técnica debido a su capacidad para detectar, informar y mitigar situaciones inesperadas o vulnerabilidades que puedan surgir durante la operación de sistemas y aplicaciones. Al capturar y gestionar errores de seguridad de manera efectiva, se pueden prevenir brechas de seguridad, minimizar el impacto de ataques y garantizar la integridad de la infraestructura.

Julián se ha dado cuenta que el conocimiento sobre código fuente y la capacidad de analizarlo es esencial para evaluar posibles vulnerabilidades en aplicaciones, y los entornos de desarrollo son herramientas vitales para escribir y analizar código, así como para realizar pruebas y auditorías de seguridad. Hoy en día la mayoría de los entornos de desarrollo permiten realizar el análisis estático de código mientras el desarrollador escribe el código y **Julián** sabe que debe instalar en sus entornos estas opciones o extensiones.

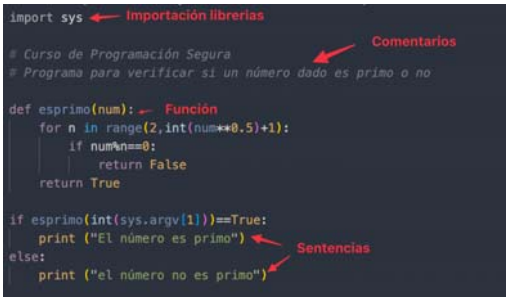
En los lenguajes de programación, se identifican varios elementos esenciales, que incluyen el léxico característico del lenguaje, así como las reglas sintácticas y semánticas. En el léxico, se emplean símbolos y palabras con funciones específicas dentro del lenguaje. Estas palabras, a menudo adoptadas del inglés, se conocen como "**palabras reservadas**" y no pueden ser utilizadas de manera diferente.

Una particularidad adicional de los lenguajes de programación es la capacidad de los programadores para incorporar comentarios en su código. Estos comentarios consisten en

frases o párrafos que carecen de funcionalidad en el programa, siendo descartados por los compiladores o intérpretes, y están diseñados exclusivamente para ser leídos por seres humanos. De esta manera, se pueden proporcionar explicaciones que faciliten la comprensión del código a quienes lo revisen. **A nivel de seguridad es muy importante que esos comentarios no incluyan información sensible, como direcciones de servidores, códigos de usuario, contraseñas…**

El código de un programa es lo que se conoce como **código fuente**. Se refiere al conjunto de instrucciones escritas por un programador en un lenguaje de programación específico. Estas instrucciones son comprensibles para los humanos y se utilizan como base para crear programas y aplicaciones de software.

Dentro de nuestro programa o código fuente podemos distinguir varios **componentes**:



David Conde (Dominio público)

- **Comandos de pre-procesamiento**: importación de módulos/librerías.
- **Sentencias y Expresiones:** los programas consisten en una secuencia de instrucciones que se ejecutan en orden. Esto puede incluir tareas como cálculos, toma de decisiones y manipulación de datos.
- **Declaración de variables:** Las variables son contenedores que almacenan datos, como números, texto o valores booleanos. Los datos pueden ser modificados y utilizados en el programa.
- **Estructuras de Control:** Las estructuras de control, como bucles y condicionales, permiten a los programadores tomar decisiones y repetir acciones según sea necesario.
- **Funciones y métodos:** son bloques de código que realizan tareas específicas. Esto promueve la modularidad y la reutilización de código.
- **Estructuras de Datos:** Las estructuras de datos, como arreglos (**array** en inglés), pilas, listas y diccionarios, permiten organizar y manipular datos de manera eficiente.
- **Orientación a Objetos (OOP):** es un paradigma de programación que se basa en el concepto de "objetos". Los objetos son instancias de clases, que son plantillas que definen la estructura y el comportamiento de los objetos. La idea fundamental es la reutilización del código.
- **Comentarios:** son notas explicativas en el código que ayudan a otros programadores (y a ti mismo en el futuro) a entender el propósito de un fragmento de código.

En la imagen de la derecha se puede ver un pequeño programa desarrollado en Python en el que se muestra la estructura básica que sigue el código fuente de un programa. En violeta se muestran las palabras reservadas del lenguaje como **def, if, else, for, return**...

Citas Para Pensar

"Si la depuración es el proceso de eliminar errores, entonces la programación debe ser el proceso de introducirlos". Edsger W. Dijkstra

3.1.- Entornos de desarrollo

Los entornos de desarrollo, comúnmente conocidos como **IDE** (Entorno de Desarrollo Integrado, por sus siglas en inglés Integrated Development Environment), son herramientas que los programadores utilizan para escribir, depurar y compilar su código. Algunos aspectos importantes de los entornos de desarrollo incluyen:

Imagen de Freepik. *experiencia-programacion-persona-que-trabaja-codigos-computadora*
(Licencia propia de Freepik)

- **Editor de Código:** Los IDEs incluyen editores de código que ofrecen resaltado de sintaxis, sugerencias de autocompletado y otras características para mejorar la eficiencia de escritura.
- **Depuración:** Los entornos de desarrollo proporcionan herramientas de depuración que permiten a los programadores detectar y corregir errores en su código de manera eficiente.
- **Gestión de Proyectos:** Los IDEs suelen ofrecer funciones para organizar proyectos, gestionar bibliotecas y dependencias, y facilitar la colaboración en equipo.
- **Integración con Herramientas:** Pueden integrar herramientas de control de versiones, compiladores, analizadores estáticos y otros recursos que hacen que el desarrollo sea más efectivo.
- **Personalización:** Los entornos de desarrollo suelen ser altamente personalizables para adaptarse a las preferencias del programador.
- **Soporte Multiplataforma:** Muchos IDEs se pueden utilizar en sistemas operativos como Windows, macOS y Linux.

Existen multitud de entornos de desarrollo y te recomendamos probar alguno porque te puede ayudar y simplificar las tareas que se van a desarrollar durante el curso. Algunos ejemplos son:

IDE	Descripción	Características Destacadas
Visual Studio Code	Un editor de código fuente gratuito y altamente personalizable desarrollado por Microsoft.	Extensiones, depuración integrada, control de versión
Atom	Desarrollado por GitHub, Atom es un editor de código fuente de código abierto y altamente personalizable.	Paquetes, integración con Git, interfaz moderna
WebStorm	Un IDE específicamente diseñado para el desarrollo web y JavaScript, desarrollado por JetBrains.	Autocompletado inteligente, depuración, prueba
Brackets	Un editor de código fuente moderno y ligero, enfocado en el diseño web.	Vista en vivo, preprocesadores, extensible
Eclipse	Un entorno de desarrollo de código abierto ampliamente utilizado que admite varios lenguajes, incluido Java.	Extensibilidad, comunidad grande, soporte multiplataforma

IntelliJ IDEA	Un IDE desarrollado por JetBrains que ofrece un excelente soporte para Java y otras tecnologías.	Refactorización, análisis estático, integración con frameworks
NetBeans	Un IDE de código abierto que soporta múltiples lenguajes y es especialmente popular para el desarrollo Java.	Integración con Maven

El poder analizar el código fuente es una práctica fundamental para mejorar la calidad, seguridad y eficiencia del desarrollo de software. Ayuda a prevenir errores, garantizar el cumplimiento de estándares y a mantener la integridad del código a lo largo del ciclo de desarrollo. Los entornos de desarrollo son herramientas vitales para escribir y analizar código, así como para realizar pruebas y auditorías de seguridad. Hoy en día la mayoría de los entornos de desarrollo permiten realizar el análisis estático de código mientras el desarrollador escribe el código.

Debes conocer

Para acelerar en el desarrollo de una aplicación y no tener que partir de cero es habitual utilizar librerías y/o frameworks:

- Una **librería** es un conjunto de funciones o rutinas predefinidas que se pueden utilizar en una aplicación para realizar tareas específicas. Estas bibliotecas generalmente se proporcionan como archivos independientes que se pueden vincular a una aplicación durante la compilación o tiempo de ejecución. Su objetivo es proporcionar funcionalidades comunes y reutilizables.
- Un **framework**, por otro lado, es un conjunto más amplio de herramientas, componentes y librerías que proporcionan una estructura y una arquitectura predefinida para el desarrollo de aplicaciones. Permiten a los desarrolladores centrarse en la lógica de la aplicación, ya que gran parte de la infraestructura y la funcionalidad común ya están implementadas.

Aunque es una gran mejora y una práctica habitual, desde el punto de vista de la ciberseguridad puede exponer a las aplicaciones a una serie de riesgos y amenazas. La revisión y la gestión adecuada de estas librerías y frameworks son fundamentales para garantizar la seguridad de las aplicaciones. Para que entiendas esta problemática te recomiendo que eches un vistazo a la vulnerabilidad provocada por el uso de la librería Log4j en este enlace

Investiga

Visual Studio Code es un editor de código (aunque en muchos sitios ya se considera un IDE) altamente popular y versátil. Es conocido por su rendimiento, capacidad de personalización y una comunidad activa de desarrolladores que crean extensiones. Investiga alguna extensión para hacer análisis estático de

seguridad de código, como por ejemplo SonarLint, CodeRush o PHP Tools for Visual Studio.

4.- Pruebas de software

Caso práctico

StockSnap. *Portatil, programación código* (Licencia de Pixabay)

Julián sabe que las pruebas de software son un elemento crítico para garantizar la calidad de un producto de software. Existen diversas metodologías para el desarrollo y ejecución de pruebas de software, pero definir los pasos para mejorar y controlar las fases del proceso de pruebas y el orden en que estas se implementan es, en general, una tarea difícil.

Habitualmente **Julián** debe realizar pruebas de examen de unidad con el código que desarrolla, junto con las pruebas de análisis estático de código para asegurarse, en la medida de lo posible, que es seguro y de calidad. Para crear las pruebas sabe que es imprescindible definir unos buenos casos de prueba.

De forma general las pruebas de software las podemos clasificar **en función de cómo las ejecutamos**:

1.- Pruebas **manuales**

2.- Pruebas **automáticas**

Boskampi (Licencia de Pixabay)

Otra clasificación se basa **en la visibilidad del código**:

- Pruebas de **caja blanca**: los evaluadores tienen acceso completo al código fuente y conocimiento detallado de la arquitectura del sistema.
- Pruebas de **caja gris**: los evaluadores tienen un conocimiento parcial del código fuente y la arquitectura del sistema. Combinan elementos de pruebas de caja blanca y caja negra.
- Pruebas de **caja negra**: no se tiene acceso al código, los evaluadores no tienen conocimiento del código fuente subyacente, solo interactúan con la aplicación desde una perspectiva de usuario externo.

También podemos clasificarlas basándonos en lo que estamos midiendo y probando, teniendo muchos tipos distintos. las más comunes son:

- **Examen de la unidad o pruebas unitarias:** valida que cada unidad de software funcione como se esperaba. Una unidad es el componente comprobable más pequeño de una aplicación. Normalmente, la unidad más pequeña suele ser una función o método. Se realizan típicamente por desarrolladores durante el proceso de desarrollo.
- **Pruebas de integración:** verifican que distintos módulos que funcionan por separado funcionen de manera conjunta. Pueden ser pruebas de integración horizontal (pruebas entre componentes similares) o pruebas de integración vertical (pruebas entre capas de la aplicación).

- **Pruebas funcionales:** se centran en verificar si el software cumple con sus especificaciones funcionales. Evalúan las funciones (no confundir con las pruebas unitarias) y características del software para garantizar que funcionen como se espera.
- **Pruebas de aceptación:** verifican si todo el conjunto de software se comporta según lo esperado. Pueden ser pruebas de aceptación del usuario (UAT) realizadas por el cliente o pruebas de aceptación del negocio (BAT) realizadas por los analistas de negocio.
- **Pruebas de usabilidad:** estas pruebas confirman la eficacia y facilidad con la que un usuario puede interactuar con un sistema o una aplicación web para llevar a cabo una tarea específica. Ayudan a garantizar que el software sea intuitivo y satisfaga las necesidades del usuario.
- **Pruebas de rendimiento:** prueban el rendimiento del software con diferentes cargas de trabajo. Se utilizan para evaluar el rendimiento en condiciones de carga reales.
- **Pruebas de regresión:** verifican si el código introducido rompe o degrada la funcionalidad del software original. Suelen realizarse cuando hacemos cambios en un programa.
- **Pruebas de Seguridad:** estas pruebas evalúan la robustez del sistema ante posibles amenazas y vulnerabilidades. Garantizan que el software sea resistente a posibles ataques y cumpla con los estándares de seguridad.

Citas Para Pensar

"Ninguna cantidad de pruebas puede probar que un software es correcto, una sola prueba puede probar que un software es incorrecto" - Amir Ghahrai

4.1.- Las pruebas de seguridad

Las **pruebas de seguridad** son esenciales para identificar y mitigar riesgos de seguridad en sistemas y aplicaciones, y son parte fundamental de la ciberseguridad en la protección de datos y activos críticos. Veamos las más típicas:

- **Pruebas de Análisis Estático** (Static Analysis Testing): Analizan el código fuente o binario de una aplicación en busca de vulnerabilidades potenciales sin ejecutar el programa. Identifican problemas de seguridad a nivel de código y los IDEs suelen incluir extensiones para realizar dichas pruebas mientras los desarrolladores crean el código.
- **Pruebas de Análisis Dinámico** (Dynamic Analysis Testing): Evalúan el software en tiempo de ejecución para detectar posibles amenazas y vulnerabilidades mientras interactúa con el sistema.
- **Pruebas de Penetración** (Penetration Testing): También conocidas como "pen tests", implican simular ataques controlados por parte de expertos en seguridad. El objetivo es identificar vulnerabilidades y puntos de entrada para posibles atacantes. Estas pruebas se realizan antes de poner en producción el software y periódicamente en el entorno de producción.
- **Pruebas de Vulnerabilidad** (Vulnerability Assessment): Identifican y evalúan vulnerabilidades conocidas en sistemas y aplicaciones. Utilizan escáneres automatizados o herramientas especializadas para detectar debilidades de seguridad. Proporcionan una lista de vulnerabilidades que necesitan ser corregidas.
- **Pruebas de Seguridad Web** (Web Application Security Testing): Centradas en evaluar la seguridad de aplicaciones web y sitios web. Incluyen pruebas de inyección SQL, cross-site scripting (XSS), cross-site request forgery (CSRF), y otras vulnerabilidades específicas de la web. Ayudan a garantizar que las aplicaciones web sean resistentes a ataques comunes.
- **Pruebas de Seguridad de Aplicaciones Móviles** (Mobile Application Security Testing): Se enfocan en evaluar aplicaciones móviles en busca de vulnerabilidades específicas de plataformas móviles. Verifican la seguridad en aspectos como el almacenamiento de datos, la autenticación, y la comunicación segura.
- **Pruebas de Seguridad de Red** (Network Security Testing): Evalúan la seguridad de la infraestructura de red, como firewalls, routers y conmutadores. Buscan posibles puntos débiles y vulnerabilidades en la red que podrían ser explotados.
- **Pruebas de Autenticación y Autorización**: Evalúan la autenticación y autorización de usuarios y sistemas en una aplicación o red. Aseguran que solo los usuarios autorizados tengan acceso a recursos específicos.
- **Pruebas de Seguridad de APIs** (API Security Testing): Se centran en evaluar la seguridad de las interfaces de programación de aplicaciones (APIs) utilizadas para la comunicación entre sistemas. Verifican que las API sean seguras y resistentes a ataques.
- **Pruebas de Estrés y Carga** (Stress and Load Testing): Evalúan cómo el sistema se comporta bajo cargas de trabajo intensas o condiciones de estrés. Identifican cuellos de botella y problemas de rendimiento que podrían afectar la seguridad.

Veamos algunos ejemplos de herramientas que permiten realizar esas pruebas y que en muchos casos nos van a permitir automatizar el proceso:

Prueba	Herramientas
Pruebas de Análisis Estático	Checkmarx, Fortify, SonarQube

Pruebas de Análisis Dinámico	AppScan, Veracode, Burp Suite
Pruebas de Penetración	Metasploit, Nmap, Burp Suite, OWASP ZAP
Pruebas de Vulnerabilidad	Nessus, OpenVAS, Qualys, Nikto
Pruebas de Seguridad Web	OWASP ZAP, Burp Suite, Acunetix
Pruebas de Seguridad de Aplicaciones Móviles	MobSF, OWASP Mobile Security Testing Guide, Burp Suite
Pruebas de Seguridad de Red	Wireshark, Nmap, Snort
Pruebas de Autenticación y Autorización	No se basan en herramientas específicas, sino en casos de prueba diseñados para evaluar estas funciones
Pruebas de Seguridad de APIs	Postman, Swagger Inspector y OWASP API Security Top Ten
Pruebas de Estrés y Carga	Apache JMeter, LoadRunner y Gatling

Citas Para Pensar

"El aprendizaje autodirigido no es solo una forma de aprender, es la forma de aprender en una época de cambio constante". - Eric Hoffer

4.2.- Recomendaciones a la hora de realizar las pruebas

thedigitalartist, *Seguridad e internet* (Licencia de Pixabay)

Como ya hemos comentado las pruebas del software son una parte muy importante del desarrollo del software y sobre todo para la ciberseguridad. Aquí te damos algunas recomendaciones a la hora de realizar las pruebas:

- Cada caso de prueba debe definir el resultado de salida esperado.
- El programador debe evitar probar sus propios programas, ya que desea (consciente o inconscientemente) demostrar que funcionan sin problemas.
- Se debe inspeccionar a conciencia el resultado de cada prueba, y así, poder descubrir posibles síntomas de defectos.
- Al generar casos de prueba, se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados.
- Las pruebas deben centrarse en dos objetivos (es habitual olvidar el segundo):
  - Probar que el programa hace lo que debe hacer.
  - Probar que el programa no hace lo que no debe hacer.
- Todos los casos de prueba deben documentarse y diseñarse con cuidado.

Además, es recomendable que las **pruebas de software sean**, en la medida de lo posible, **automatizadas** ya que permiten verificar de manera eficiente la funcionalidad de un programa a través de scripts y herramientas de prueba. Por ejemplo, imagina que se estamos desarrollando un sitio web de comercio electrónico y queremos asegurarnos de que el proceso de compra funcione correctamente. Un script de prueba automatizado podría simular el proceso de selección de un producto, agregarlo al carrito, completar el proceso de pago y verificar que se complete con éxito. Estos scripts pueden ejecutarse de manera repetitiva, rápida y consistente, lo que ahorra tiempo y recursos en comparación con las pruebas manuales.

Investiga

Existen múltiples herramientas que permiten crear esas pruebas automatizadas, por ejemplo investiga un poco la herramienta Selenium y en concreto sobre su componente **Selenium WebDriver**. Comprueba en qué lenguajes permite escribir los scripts, para qué navegadores está soportado y cómo funciona.

5.- Evaluación de lenguajes de programación

Evaluar los lenguajes de programación en cuanto a seguridad no es tarea sencilla, ya que la mayoría de veces no depende del lenguaje en sí, sino del programador, de la tecnología que implementa ese lenguaje y sobre todo del volumen de código que hay publicado de ese lenguaje de programación.


Ilustración del concepto de evaluación, con un dedo en medio de una espiral de flechas naranja

Geralt. *Evaluación* (Licencia de Pixabay)

Si tuviéramos que clasificar el lenguaje de programación con más vulnerabilidades sería sin duda el lenguaje C, casi la mitad de las vulnerabilidades de software reportadas los últimos años están relacionadas con el lenguaje C. Esto nos haría pensar que C es un lenguaje menos seguro que los otros lenguajes, pero la explicación de la gran cantidad de vulnerabilidades que hay en C es sencilla:

- El lenguaje C es uno de los lenguajes más antiguos y por tanto se ha usado durante más tiempo, por lo tanto, a más tiempo expuesto más vulnerabilidades encontradas
- Es el lenguaje con mayor volumen de código abierto publicado en Internet
- Muchos sistemas operativos como UNIX/Linux, incluso partes de Windows usan C

Realmente si tenemos que evaluar los lenguajes de programación en cuanto a su seguridad deberíamos de valorar más que el propio lenguaje el ciclo de desarrollo del mismo dentro de un proyecto. En la mayoría de las páginas oficiales de los lenguajes de programación suele existir unas guías de buenas prácticas para desarrollar con seguridad, te recomendamos revisar alguna, por ejemplo, este enlace

En la próxima unidad se verán las fuentes abiertas para el desarrollo seguro y dentro de estas fuentes una muy habitual es "**OWASP Secure Coding Practices**". Se refiere a un conjunto de prácticas y pautas de seguridad recomendadas por el Proyecto de Seguridad de Aplicaciones Web Abiertas (OWASP, por sus siglas en inglés) para ayudar a los desarrolladores a escribir código seguro. El proyecto "OWASP Secure Coding Practices" se enfoca en ofrecer orientación específica para escribir código seguro en una variedad de lenguajes de programación y plataformas. Proporciona directrices detalladas y ejemplos prácticos para abordar vulnerabilidades conocidas, como inyección de SQL, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) y muchas otras. Puedes encontrar más información y recursos específicos sobre las "OWASP Secure Coding Practices" en el sitio web de OWASP y en su documentación oficial.

Debes conocer

Cuando estamos desarrollando y probando software necesitamos disponer de un entorno de desarrollo aislado donde poder hacer nuestras pruebas y verificar si el software concreto que hemos desarrollado se comporta de la manera esperada ante las distintas variables o datos procesados.

La mejor manera de hacerlo es tener un entorno controlado, no influenciado por otras piezas de software o hardware y donde todos los programadores pueden

tener las mismas condiciones y por tanto poder medir y probar el software de manera eficaz y eficiente.

Estos entornos se conocen como s*andbox,* es decir es un entorno cerrado en el que probar los cambios de código de forma aislada, sin comprometer la producción ni la edición del programa en desarrollo. Así, el sandbox protege, controla de forma preventiva y en tiempo real la ejecución del código, evitando cambios que podrían comprometer la estabilidad del programa.

Para saber más

Si bien el término *sandbox* se refiere a un entorno aislado y controlado, dentro de la informática tiene dos usos. A nivel de desarrollo de software, nuestro caso, se podría definir como el entorno donde creamos y probamos nuestro software de manera controlada, mientras que a nivel de ciberseguridad se entiende por *sandbox* el entorno donde se ejecuta de manera aislada el software malicioso o malware para ver cómo se comporta sin suponer un riesgo.

Autoevaluación

Marca las siguientes sentencias como verdaderas o falsas según consideres.

1. El lenguaje C es el más vulnerable puesto que tiene fallos graves de diseño.

○ Verdadero ○ Falso

Falso

Las vulnerabilidades del lenguaje C están más relacionadas con la cantidad de código disponible y con el tiempo que lleva ese código publicado.

2. Python es un lenguaje muy seguro porque tiene librerías para depurar y hacer correcciones de código.

○ Verdadero ○ Falso

Falso

La mayoría de lenguajes modernos incorporan módulos o librerías que permiten revisar y detectar errores de manera automática o semi automática. No podemos marcar lenguajes como seguros o inseguros de manera categórica, sino su diseño, implantación y validación dentro de un proyecto.

3. No tiene sentido analizar los posibles datos de entrada de una función mientras estamos programando porque ya haremos la revisión más adelante de manera automática.

○ Verdadero ○ Falso

Falso

Cuando definimos una función deberemos de entender que datos estamos permitiendo de entrada y realmente revisar si lo podemos limitar lo máximo posible para prevenir situaciones no esperadas.

4. El sistema operativo Linux no es muy seguro porque está escrito mayoritariamente en C.

○ Verdadero ○ Falso

Falso

Que un sistema operativo sea robusto o no no depende tanto del lenguaje de programación, como el hecho de si es un sistema operativo abierto y su código está publico, así como el numero de librerías o módulos de terceros que se incluyen sin revisar.