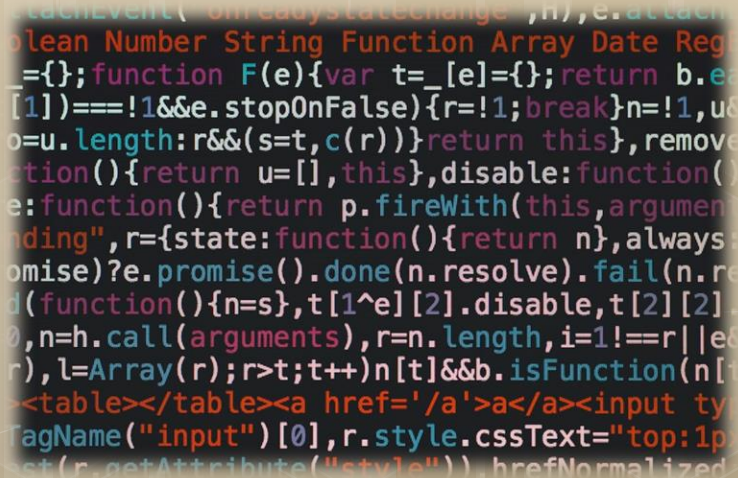

Unidad 6: COMPILADORES Y LENGUAJES DE PROGRAMACIÓN

BLOQUE II – Sistemas, aplicaciones y personas

CONTENIDOS

1. Introducción.
2. Programación y lenguajes de alto nivel.
3. Compilador.
4. Primeras nociones de seguridad en el software.

1. Introducción

- Hasta ahora sabemos que:

1. Introducción

- Ya hemos comentado en esta asignatura que los programadores que trabajaban con primeros computadores, los más sencillos, escribían sus aplicaciones directamente en lenguaje ensamblador.
  - También trabajan así los que desarrollan ciertas partes de los SO.
- Pero al hacerse las arquitecturas y los repertorios más complejos se comenzaron a utilizar lenguajes de alto nivel.
  - Añaden una capa de abstracción que simplifica la programación (se parece más a cómo pensamos los humanos), mejora la portabilidad, etc.

1. Introducción

Lenguajes compilados

- Hay un código fuente (propietario o abierto) y se necesita un compilador que lo transforme en código objeto
- La compilación, costosa, sólo se realiza una vez, luego la aplicación se ejecuta las veces que sea necesario
- C, Java, C++

Lenguajes interpretados

- Las instrucciones en ensamblador se generan en tiempo real, no hace falta compilación
- Esto ocurre en cada ejecución de la aplicación, que es algo más lenta
- Python, JavaScript

2. Programación y lenguajes de alto nivel

Asignaturas completas: **Introducción a la Programación** (estáis en ello) y **Estructuras de Datos**, este mismo curso

3. Compilador

- Es una herramienta esencial hoy en día (porque los lenguajes compilados están muy extendidos).
- Partiendo de un código fuente, analiza la sintaxis/gramática:
  - Lo divide en unidades mínimas de información teniendo en cuenta la sintaxis del lenguaje (tokenization).
  - Lo interpreta teniendo en cuenta la gramática del lenguaje (parsing).
- Si todo es correcto, genera un código objeto que el sistema operativo pueda lanzar y ejecutar sobre la arquitectura objetivo.

- ¿Os acordáis de la práctica 1?

3. Compilador

- Un buen compilador:

Detecta diferentes tipos de errores y ayuda a resolverlos

Genera código eficiente para diferentes arquitecturas objeto

Es capaz de ayudar con la paralelización de diferentes formas

4. Primeras nociones de seguridad en el software

- Ya sabemos que una de las principales fuentes de vulnerabilidades en los sistemas actuales es el software:
  - Las aplicaciones.
  - El sistema operativo.
  - Incluso los drivers, el firmware, etc.
- Pregunta: ¿Por qué aparecen estas vulnerabilidades?

4. Primeras nociones de seguridad en el software

- Comprobar los operandos antes de utilizarlos (para evitar divisiones por 0, por ejemplo).
- Comprobar rangos antes de realizar conversiones de tipos.
- No ignorar valores de retorno ni excepciones.
- Validación de datos de entrada.
  - Comprobar el tamaño de los operandos que se pasan a las funciones y procedimientos.
  - Normalización de cadenas y de datos.
  - Canonización de rutas, rutas que sólo incluyan caracteres alfanuméricos.

4. Primeras nociones de seguridad en el software

- Evitar usar identificadores públicos de clases, interfaces, paquetes, etc.
- No mostrar información sensible en las excepciones.
- Ni en el código, se pueden emplear des-ensambladores y de-compiladores para localizarla.
- Borrar ficheros temporales.
- Liberar recursos cuando no se necesitan.
- Generar números aleatorios “fuertes”.

4. Primeras nociones de seguridad en el software

- Existen diferentes guías que recopilan mejores prácticas para el desarrollo de aplicaciones seguras en diferentes lenguajes de programación:
  - ISO: Programming languages – Guide for the Use of the Ada Programming Language in High Integrity Systems.
  - MISRA: Guidelines for the use of the C language in critical systems.
  - CERT: C Secure Coding Standard, Secure Coding in C and C++ y Oracle Secure Coding Standard for Java.
  - OWASP: Secure Coding Practices Quick Reference Guide.

4. Primeras nociones de seguridad en el software

- Existen analizadores estáticos de código que son capaces de detectar las vulnerabilidades más extendidas:

PMD	FlawFinder	LAPSE (OWASP)
CodeSecure (Armorize)	AppScan (IBM)	Fortify (HP)

4. Primeras nociones de seguridad en el software

- Con el análisis estático o los check-lists que comprueban la aplicación de mejores prácticas, no se ejecuta el código que se está probando.
- Las técnicas de análisis dinámico de código sí que lo ejecutan, por lo que se monitoriza su comportamiento para ver si supera un conjunto de requisitos (verificación formal).
  - La ventaja frente al análisis estático es que se está probando el programa en un escenario de ejecución real, donde pueden aparecer vulnerabilidades/debilidades no detectados en el análisis estático.

4. Primeras nociones de seguridad en el software

Asignaturas completas: **Desarrollo web seguro, Metodologías de desarrollo seguro e Ingeniería del Software**

Para practicar un poco

1. ¿Qué compilador usáis en IP (o cuando programáis para vuestros proyectos personales)? Investiga un poco acerca de su funcionamiento, de las opciones de compilación que ofrece, en concreto de las que tienen que ver con protección y seguridad, etc.
2. Recuerda la base de datos CWE, comienza a familiarizarte con las debilidades más habituales y busca mejores prácticas para evitarlas. Aprovecha y aprende a programar aprendiendo directamente a programar de manera segura.
3. Comienza a familiarizarte con esta página de OWASP acerca del análisis estático de código y prueba alguna de las herramientas con códigos sencillos que ya tengas: https://www.owasp.org/index.php/Static_Code_Analysis

Referencias

- Fotografías
  - https://unsplash.com
- Iconos
  - https://www.flaticon.es/