UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Puesta en producción segura

- Tema 1: Prueba de aplicaciones web y para dispositivos móviles

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Índice

- Objetivos
- Fundamentos de programación
- Elementos principales de un programa
- Pruebas software
- Entornos de ejecución (sandboxes)
- Seguridad de los lenguajes de programación

Objetivos

- Definir qué es un lenguaje de programación así como los distintos tipos de lenguaje.
- Comprender la diferencia entre lenguajes compilados e interpretados.
- Reconocer los elementos básicos de un programa así como entender su significado.
- Conocer los diferentes tipos de prueba existentes en el ciclo de desarrollo del software.
- Conocer los diferentes riesgos y vulnerabilidades existentes.

Fundamentos de programación
Lenguajes de programación

- Es un conjunto de instrucciones y reglas (lógica) que forman un lenguaje formal y proporcionan la capacidad de escribir órdenes que controlan el comportamiento de un ordenador y sus programas.

- Se descomponen en:
  - **Sintaxis**: Estructura gramatical del lenguaje. “Que lo que escribes sea gramaticalmente correcto”.
  - **Semántica**: Se refiere al significado del código. Un programa puede ser sintácticamente correcto pero semánticamente incorrecto. “Compila, pero no hace lo que quiero”.

Fundamentos de programación
Paradigmas de programación

- Es un estilo o filosofía que adoptan los lenguajes de programación para resolver un problema. Define el lenguaje de programación y cómo funciona éste.

- Tipos:
  - **Imperativa**: declaran un secuencia de operaciones a realizar (cómo hacerlo) → PHP, Python, Java…
  - **Declarativa**: se describe el resultado final deseado → lenguajes SQL, CSS, Haskell…

Fundamentos de programación
Paradigmas de programación

- Tipos:
  - **Orientada a objetos**: creación de modelos de objeto (clases) que contienen atributos y funcionalidades propias → Java, Python, PHP…
  - **Dirigida por eventos**: el flujo del programa (bucle principal) se determina según los eventos que ocurren → librerías para Java, C, C#...

Fundamentos de programación
Tipado de los lenguajes

- Forma en la que un lenguaje de programación gestiona los tipos de dato (valor/contenido de las variables). Depende de cuándo se realiza la comprobación del tipo:
  - **Tipado estático**: comprobación en tiempo de compilación.
    - Permite detectar errores antes (1 + “1” = “11” de JavaScript).
    - Ejecución más consistente.
    - C, C++, Go…
  - **Tipado dinámico**: comprobación en tiempo de ejecución.
    - Sistema de tipos más sencillo.
    - Menos restricciones → Código más flexible.
    - Python, PHP, JavaScript..

Fundamentos de programación
Lenguajes compilados

- Traducen previamente el código fuente a código máquina para generar el ejecutable del programa (C, C++, Go, Rust).

- Ventajas:
  - Son más rápidos y eficientes.
  - A priori, no se tiene acceso al código fuente (ingeniería inversa).
- Inconvenientes:
  - Necesidad de compilar el programa cada vez que se cambia el código fuente → ciclo de desarrollo más lento.
  - Deben ser compilados específicamente para cada plataforma.

Fundamentos de programación
Lenguajes compilados

- Arquitecturas (instruction sets) soportadas por Go:

`amd64`, `386`
The x86 instruction set, 64- and 32-bit.
`arm64`, `arm`
The ARM instruction set, 64-bit (AArch64) and 32-bit.
`mips64`, `mips64le`, `mips`, `mipsle`
The MIPS instruction set, big- and little-endian, 64- and 32-bit.
`ppc64`, `ppc64le`
The 64-bit PowerPC instruction set, big- and little-endian.
`riscv64`
The 64-bit RISC-V instruction set.
`s390x`
The IBM z/Architecture.
`wasm`
WebAssembly.

Fundamentos de programación
Lenguajes interpretados

- **Interpretados**: el código fuente es traducido a código máquina a medida que va siendo leído por el intérprete durante la ejecución (Python, PHP, JavaScript).

- Ventajas:
  - Multiplataforma.
  - El ciclo de desarrollo es más rápido que en los lenguajes compilados.
  - Lenguajes más flexibles y sencillos de probar.
- Inconvenientes:
  - En general son más lentos y consumen más recursos que los lenguajes compilados.
  - Se requiere del intérprete para ejecutar el código.

Fundamentos de programación
Lenguajes mixtos

- Proceso de compilación a un código intermedio (bytecode) que luego es interpretado.

  - **Java**: el código fuente es compilado a bytecode y ejecutado a través de la máquina virtual de java (JVM).
  - **.NET**: similar a java, su código se compila a un lenguaje intermedio llamado “CIL” que después en tiempo de ejecución es convertido a código nativo.
  - **Solidity**: lenguaje para la ejecución de smart contracts en la plataforma Ethereum. Se compila a bytecode y se ejecuta en la Ethereum Virtual Machine (EVM).

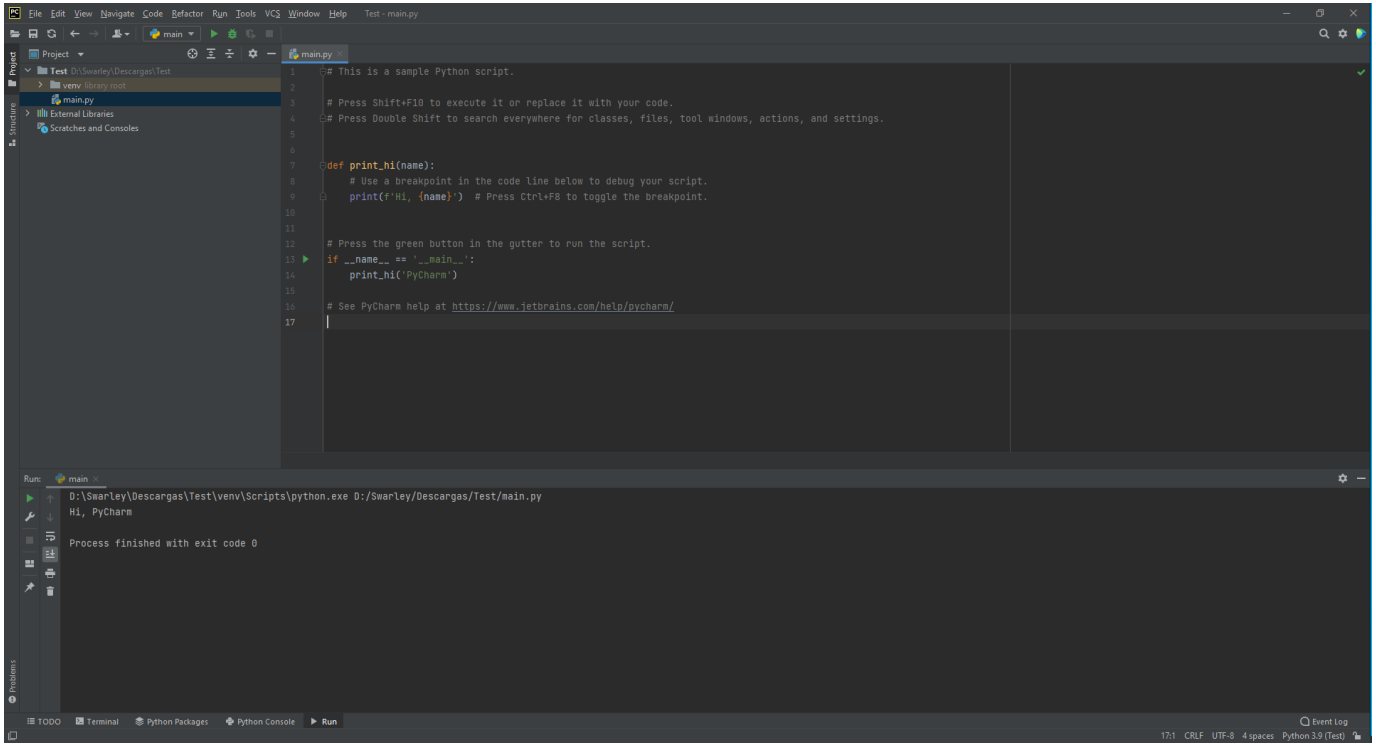
Fundamentos de programación
Entornos de desarrollo integrado (IDE)

- Software o marco de trabajo que integra diferentes utilidades para facilitar el proceso de desarrollo a los programadores.

- Principales características de los IDEs:
  - Editor/visualizador de código.
  - Depurador de código.
  - Compilador/intérprete de código.
  - Autocompletado de código.
  - Soporte para uno o varios lenguajes de programación.
  - Gestor de paquetes/librerías/plugins.

Fundamentos de programación
Entornos de desarrollo integrado (IDE)

- Ejemplos: **Eclipse, NetBeans, Visual Studio, PyCharm, PhpStorm, Android Studio, Xcode…**



- https://www.jetbrains.com/
- https://www.jetbrains.com/es-es/community/education/#students

Fundamentos de programación
Entornos de desarrollo integrado (IDE)

- También existen editores/visualizadores de código más livianos:
  - **Sublime Text**.
  - **Visual Studio Code**.
  - **Atom Editor**.
  - **Notepad++**.

Elementos principales de un programa

Introducción a Python

- Lenguaje de propósito general.
  - Scripting, desarrollo web, móvil y de escritorio.
- Lenguaje simple, fácil de aprender.
- Multiplataforma.
  - Windows, Linux, macOS.
- Multiparadigma.
  - Orientado a objetos, imperativo.
- Tipado dinámico.
- Lenguaje interpretado.

Elementos principales de un programa

Introducción a Python

- Instalación de Python:
  - Mediante un IDE como PyCharm (desarrollo).
  - Manualmente (PoCs, pruebas rápidas, enseñanza).

- Instalación del paquete Jupyter:
  - Notebook, aplicación web que permite desarrollar y ejecutar código Python (cheatsheet).
  - *pip install jupyter*
  - *jupyter notebook*

Elementos principales de un programa

Variables y tipos de datos

- No se declaran variables, directamente se les asignan valores.
- El tipo se asigna de manera automática (tipado dinámico).

```
b = 5
print(type(b))
c = 6.0
print(type(c))
a = "esto es un string"
print(type(a))
a = 5
print(type(a))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'int'>
```

Elementos principales de un programa

Variables y tipos de datos

- Las variables solo pueden ser accedidas dentro de su scope (funciones, clases…).
- Keyword “global” para hacerla accesible desde fuera de su scope.

```
def funcion():
    global x
    x = "Carlos"

funcion()

print("Hola, me llamo " + x)
```

Elementos principales de un programa

Variables y tipos de datos

- **Casting de variables**: conversión de un tipo de datos a otro.
  - Especifica el tipo de dato (ojo, no es una declaración).
  - str(), int(), float()…

- Ejemplo:

```
nombre = "Carlos"
edad = 27
print(nombre + ", " + edad + " años")
```

- ¿Qué creéis que ocurriría?

Elementos principales de un programa

Sentencias de control de flujo

```python
dia = 7

if dia == 1:
    print('lunes')
elif dia == 2:
    print('martes')
elif dia == 3:
    print('miércoles')
elif dia == 4:
    print('jueves')
elif dia == 5:
    print('viernes')
elif dia == 6:
    print('sábado')
elif dia == 7:
    print('domingo')
else:
    print('no existe ese día')
```

```go
dia := 7

switch dia {
case 1:
    fmt.Println("lunes")
case 2:
    fmt.Println("martes")
case 3:
    fmt.Println("miércoles")
case 4:
    fmt.Println("jueves")
case 5:
    fmt.Println("viernes")
case 6:
    fmt.Println("sábado")
case 7:
    fmt.Println("domingo")
default:
    fmt.Printf("no existe ese día")
}
```

- No existe el switch en Python (decisión de los creadores).

Elementos principales de un programa

Sentencias de control de flujo - Condicionales

- 1)

```
a = 5
b = 6

if a > b:
    print(a + " es mayor que " + b)
elif a < b:
    print(a + " es menor que " + b)
else:
    print(a + " y " + b + " son iguales")
```

- 2)

```
if True:
    print("Hola mundo")
print("Que esta pasando")
else:
    print("No va a entrar nunca")
```

- ¿Daría una salida correcta? **Identación!**

Elementos principales de un programa

Sentencias de control de flujo - Bucles

- **While**:

```
import string

digits = string.digits
index = 0

while index < len(digits):
    print(digits[index])
    index += 1
```

- **For**:
  - 1)

```
ascii_letters = string.ascii_letters
for i in ascii_letters:
    print(i)
```

  - 2)

```
ascii_letters = string.ascii_uppercase
for i in range(0,len(ascii_letters)):
    print(ascii_letters[i])
```

https://docs.python.org/3/library/string.html

Elementos principales de un programa

Entrada y salida de datos

- **Entrada por teclado**:
  - Se almacena como una cadena de texto, si queremos un número entero hay que hacer la conversión con int().

```
num1 = input('introduce un número por teclado:')
print(type(num1))
```

- **Salida por pantalla**:

```
print('El número introducido es: ' + str(num1))
print('El número introducido es: {}'.format(num1))
print('El número es : {:04d}'.format(num1))
print('Formateando decimales {:.9f}'.format(5.312313))
print('Numero decimal %i y numero en flotante %.3f' % (num1,2.1234))
```

- Placeholders en Python:
  - https://www.w3schools.com/python/ref_string_format.asp

Elementos principales de un programa

Entrada y salida de datos

- **Lectura de ficheros:**

```
f = open("archivo.txt","r")
data = f.readline()
f.close()
```

- **Escritura de ficheros:**

```
f = open("archivo.txt","a")
f.write("Escribiendo en el archivo de texto")
f.close()
```

Elementos principales de un programa

Funciones

- Bloques de código con nombre que reciben parámetros como entrada y devuelven valores.
- Modularización y reutilización del código.
- Son declaradas a través de la keyword “def”.
- Las instrucciones que formen parte de la función deben encontrarse identadas.

```
def funcion1(x, y, z):
    print(x + y + z)

def funcion2(x, y):
    return x + y
```

Elementos principales de un programa

Funciones

- **Parámetros por omisión**: definición de una función con parámetros por defecto que serán los utilizados en caso de no ser especificados al invocar a la función.

```
def funcion3(x = 1, y = 1):
    return x + y

print(funcion3(3, 3))
print(funcion3())
```

Elementos principales de un programa

Funciones

- **Múltiples parámetros**: permite definir un número desconocido de parámetros.

```
def mi_suma(*args):
    suma = 0
    for i in args:
        suma += i
    return suma

total = mi_suma(6,7,10,5,1)
print(total)
```

- ¿Qué pasaría si realizamos la llamada “mi_suma()”?
- ¿Y si la llamada es “mi_suma(‘1’,’2’,’3’,’4’)”?

Elementos principales de un programa

Estructuras de datos

- **Listas**: colección simple de datos heterogéneos y mutables (arrays).

```
mi_lista = [1]
mi_lista.append(5)
mi_lista.append("hola")
mi_lista.append([5.0,3])
print(mi_lista)
mi_lista[2] = 'Adiossss'
print(mi_lista)
```

Se usan corchetes!

- ¿Qué resultado se obtendría al acceder a “mi_lista[4]”? ¿Y a “mi_lista[-1]”?
- Se puede obtener un subconjunto de valores con “mi_lista[0:4:2]”.

Elementos principales de un programa

Estructuras de datos

- **Tuplas:** colección simple de datos heterogéneos e inmutables, es decir, una vez se genera una tupla no puede modificarse el valor de ninguno de sus elementos.

```
mi_tupla = (5,10,"hola")
```

Se usan paréntesis!

- ¿Qué pasa si intentamos modificar un elemento “mi_tupla[2]= 33”?

Elementos principales de un programa

Estructuras de datos

- **Diccionarios:** estructura de datos del tipo clave-valor. Cada clave es única y a través de ésta se accede a su valor.

```
mi_dict = {}
print(type(mi_dict))
mi_dict = dict()
print(type(mi_dict))
mi_dict['nombre'] = 'mi nombre'
mi_dict['apellidos'] = 'mis apellidos'
mi_dict['direccion'] = 'calle lo que sea'
mi_dict['dni'] = '11111111A'
```

Se usan llaves!

- Obtención de los clave-valor:

```
for i in mi_dict:
    print("clave: " + i + " valor: " + mi_dict[i])
```

```
clave: nombre valor: mi nombre
clave: apellidos valor: mis apellidos
clave: direccion valor: calle lo que sea
clave: dni valor: 11111111A
```

Elementos principales de un programa

Clases y objetos

- Forma de empaquetar datos (atributos) y métodos que trabajan sobre los datos.
- Un objeto es una instancia de una clase y se pueden crear múltiples objetos sobre una clase (plantilla).
- Las clases en Python tienen las mismas características que cualquier lenguaje orientado a objetos:
  - Herencia: para extender la funcionalidad de una clase.
  - Polimorfismo: diferentes clases, mismo método, diferentes operaciones y resultado.

Elementos principales de un programa

Clases y objetos

```
import r2pipe
import json
class radareAnalysis():
    func_list = []
    r2_handler = None

    def __init__(self,filename):
        '''Constructor de la clase
        :param filename: ruta al fichero a analizar.
        '''
        print("init method")
        self.r2_handler = r2pipe.open(filename,flags=['-2'])

    def analyze(self):
        self.r2_handler.cmd('aaa')
        functions = json.loads(self.r2_handler.cmd('aflj'))
        for f in functions:
            self.func_list.append(f['name'])

    def get_functions(self):
        return self.func_list

    def get_disassembly(self,func):
        self.r2_handler.cmd('s ' + func)
        dis = json.loads(self.r2_handler.cmd('pdfj'))
        list_ins = []
        for ins in dis.get('ops'):
            list_ins.append(ins['disasm'])
        return list_ins


    def close(self):
        print("close method")
        try:
            self.r2_handler.quit()
        except AttributeError:
            print("Except: el handler ya había sido cerrado")

    def __del__(self):
        '''Destructor de la clase.
           Este método se ejecuta automáticamente
           cada vez que el objeto se destruye'''
        print("del method")
        self.close()
```

- En Python, los atributos de una clase son públicos.
- "**__init__**": constructor de la clase.
- "**self**": hace referencia a la propia clase (primer parámetro de cada función).
- "**None**": null o nil de otros lenguajes.

Elementos principales de un programa

Algoritmos

- Conjunto de instrucciones o pasos bien definidos para resolver un problema concreto.

- Características:
  - Debe ser claro, preciso y conciso.
  - Debe tener sus entradas y salidas bien definidas.
  - Deben ser finitos.
  - Deben ser independientes de cualquier lenguaje de programación.

Elementos principales de un programa

Algoritmos

- Pseudocódigo:

función max(C)

//C es un conjunto no vacío de números//

$n \leftarrow |C|$ //$|C|$ es el número de elementos de C//

$m \leftarrow c_0$

para $i \leftarrow 1$ **hasta** n **hacer**

si $c_i > m$ **entonces**

$m \leftarrow c_i$

devolver m

- Implementación (C++):

```
int max(int c[], int n)
{
    int i, m = c[0];
    for (i = 1; i < n; i++)
        if (c[i] > m) m = c[i];
    return m;
}
```

Elementos principales de un programa

Módulos

- Los módulos/paquetes/bibliotecas agrupan código previamente implementado para poder ser reutilizado en otros programas.
- Los módulos en Python se importan mediante la keyword “import” al principio del fichero.
  - Ejemplos:
    - **import json**
    - **import datetime**
- Existen un gran numero de módulos en Python que pueden instalarse desde su repositorio.
  - Ejemplos: *pip install jupyter, requests, shodan, nmap*

Pruebas software

Funcionales y no funcionales

- Las pruebas de software son un proceso cuya intención es la búsqueda de errores y comprobación de que una aplicación o sistema funciona de la manera que se espera.

- Tipos:
  - **Pruebas funcionales**: pruebas que verifican que el software cumple con los requisitos especificados (hace lo que tiene que hacer) → calidad del software.
  - **Pruebas no funcionales**: se centran en verificar el comportamiento del software sin entrar en aspectos de funcionalidad → rendimiento, escalabilidad, usabilidad…

Pruebas software

Funcionales y no funcionales

- **Pruebas funcionales:**
  - Pruebas unitarias
  - Pruebas de integración
  - Pruebas de regresión
  - Pruebas de humo
  - Pruebas de aceptación
  - …
- **Pruebas no funcionales:**
  - Pruebas de carga
  - Pruebas de estrés
  - Pruebas de compatibilidad
  - Pruebas de usabilidad
  - Pruebas de escalabilidad
  - Pruebas de seguridad
  - …

Pruebas software

Ejemplo: pruebas unitarias en Python

```python
import unittest
import mi_modulo

class prueba_modulo(unittest.TestCase):

        def test_suma(self):
                self.assertEqual(mi_modulo.suma(1100,60),1160)
        def test_rest(self):
                self.assertEqual(mi_modulo.resta(1000,600),400)
        def test_multiplicacion(self):
                self.assertEqual(mi_modulo.multiplicacion(25,25),25*25)
        def test_division(self):
                self.assertEqual(mi_modulo.division(10,5),2)
        def test_isnumber(self):
                self.assertTrue(mi_modulo.isNumber(5),True)
                self.assertTrue(mi_modulo.isNumber(1.0),True)
                self.assertFalse(mi_modulo.isNumber('1'),False)

if __name__ == '__main__':
        unittest.main()
```

```
kali@kali:~/UNIT TEST$ python3 -m unittest -v mi_prueba.py
test_division (mi_prueba.prueba_modulo) ... ok
test_isnumber (mi_prueba.prueba_modulo) ... ok
test_multiplicacion (mi_prueba.prueba_modulo) ... ok
test_rest (mi_prueba.prueba_modulo) ... ok
test_suma (mi_prueba.prueba_modulo) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

Entornos de ejecución

- Término que se refiere al entorno software donde se ejecutarán los programas, así como los requisitos necesarios para su correcto funcionamiento → Java Runtime Environment (JRE).

- Principales requisitos:
  - Bibliotecas propias del lenguaje de programación.
  - Intérprete /máquina virtual → Java Virtual Machine (JVM).
  - Variables de entorno (configuración entre despliegues/ejecuciones).
  - Hardware y sistema operativo.
  - …

Entornos de ejecución

Sandboxes

- Entornos de ejecución controlados que se utilizan para ejecutar archivos sin afectar al sistema o plataforma principal.

- Principales usos:
  - Probar software nuevo antes de pasarlo a producción.
  - Analizar software potencialmente malicioso (malware).
  - Para proteger que el software no pueda acceder a datos privados de los usuarios → Navegadores.

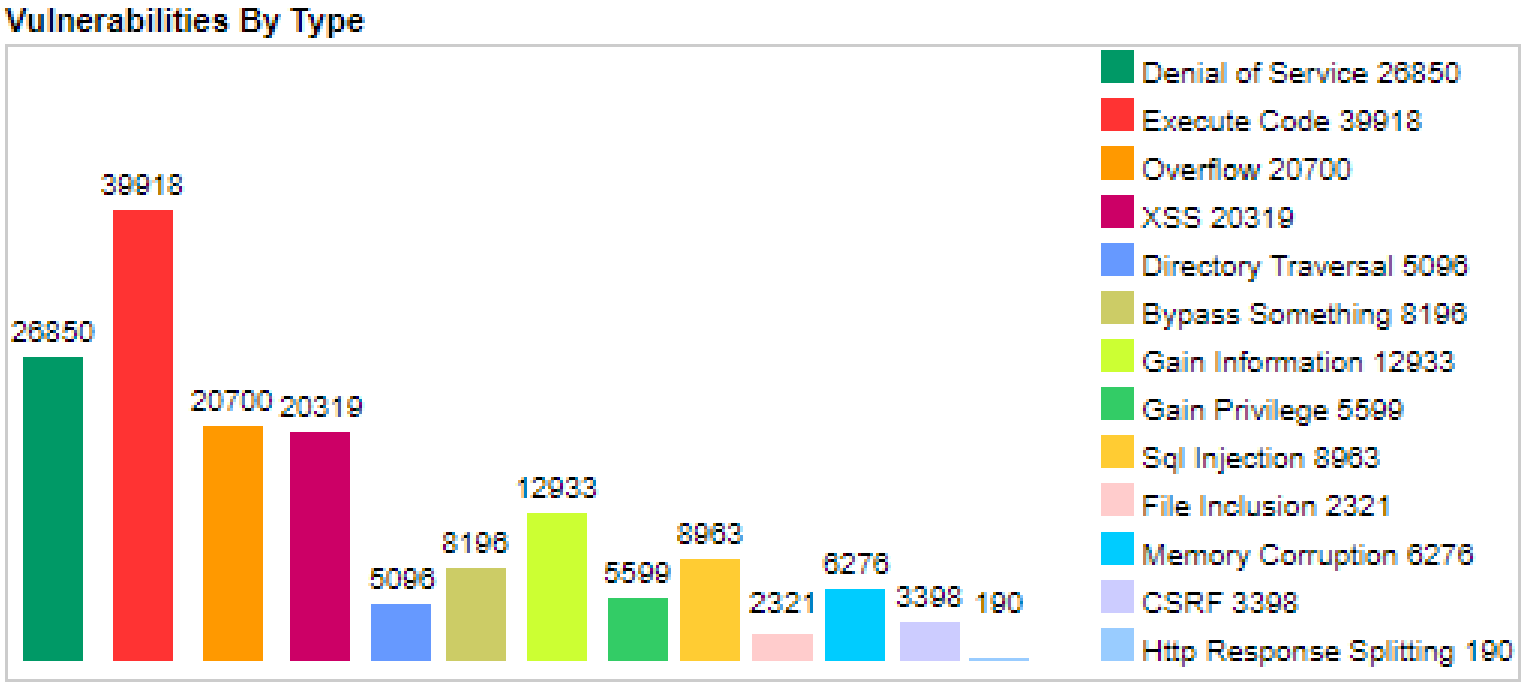
Seguridad de los lenguajes de programación

Buenas prácticas

- Validación de parámetros de entrada: limpieza (1).
- Validación de parámetros de entrada: límites (2).
- Correcto manejo de errores (try-catch).
- Criptografía a nivel de código (cifrado y funciones hash).
- Protección de datos sensibles (contraseñas).
- Autenticación y autorización (cookies de sesión).
- Seguridad en las comunicaciones (HTTPS).
- …

Seguridad de los lenguajes de programación

Principales vulnerabilidades (2021)



Fuente: cvedetails.com