

# Puesta en Producción Segura

Unidad . Unidad 1

Infraestructuras de seguridad  
de diferentes lenguajes de  
programación.



*"La seguridad de los sistemas no depende de lo fuerte que sea una sola parte, sino de lo débil que sea su eslabón más frágil."*

— Bruce Schneier

# Objetivos

- Comparar las diferentes infraestructuras de seguridad que adoptan diferentes lenguajes de programación.



# Contenidos

**01** Infraestructuras de seguridad en lenguajes de programación (Python)

**02** Comparación de infraestructuras de seguridad en diferentes lenguajes.



01

# Infraestructuras de seguridad en lenguajes de programación (Python)



# Python y Cpython

- CPython es interfaz abstracta de Python o dicho de otra forma, la implementación por defecto de ejecución de lenguaje de programación Python.
- Escrito en C++.



# Python: ¿Lenguaje interpretado o compilado?

- CPython compila el código fuente de Python a bytecode.
- Genera archivo .pyc con bytecode.
- Luego ejecuta bytecode en MV de CPython.



# Acceso ilegal a memoria

- Se trata de acceder a zonas de memoria que no pertenecen al programa en ejecución.
- Es uno de los problemas de seguridad más frecuentes en lenguajes como C++, en el que se basó Java inicialmente.
  - Un tipo de ataque que explota esto es el desbordamiento de buffer (**buffer overflow**).

**Veamos soluciones propuestas:**



# Acceso ilegal a memoria

- **Eliminación de la aritmética de punteros:**

- Lenguajes antiguos como Pascal, C o C++ permiten acceder a posiciones de memoria aleatorias mediante un mecanismo llamado puntero.
- Estas posiciones no tenían por qué ser del propio programa, permitiendo acceder a segmentos de los demás.
- Al acceder fuera de las direcciones del programa
  - Si el SO es seguro, obtendremos un error de violación de segmento y el programa termina.
  - Si no, podremos modificar los otros programas en ejecución o incluso acceder a la memoria de la tarjeta gráfica, etc.

# Acceso ilegal a memoria en Python

- Gestión automática de memoria:
  - Python usa un **heap** gestionado y **garbage collector** → no permite acceso directo a direcciones de memoria arbitrarias, y en caso de que se produjera, lanza una excepción de alto nivel (*IndexError*, *KeyError*, *AttributeError*, etc.).
  - Esto evita vulnerabilidades clásicas como **buffer overflow** o **dangling pointers**.
- Tipado dinámico y seguro:
  - No puedes escribir fuera de un array o acceder más allá de los límites de una lista.

# Verificación de tipos

- Para la verificación de tipos, los lenguajes compilados como **Java**, realiza esa comprobación en tiempo de compilación.
- En el caso de lenguajes interpretados, se realiza en tiempo de ejecución por lo que es más fácil que se pueda producir una situación inesperada.



# Verificación de tipos en Python

- Python es **tipado dinámico fuerte**: no hay *type confusion* en operaciones básicas.
- Un **int** no se suma con un **str** sin conversión explícita → se lanza **TypeError**.
- Esto previene ataques basados en mezclar tipos para forzar interpretaciones erróneas.



# Niveles de acceso y visibilidad de variables

- ¿Podemos acceder a las atributos y métodos de las clases desde cualquier lugar?
- En java los atributos y métodos tienen niveles de acceso que permiten controlar su visibilidad: **Public, protected, package y private.**
- Pero esto, en java, se puede saltar con el mecanismo de reflexión, por lo que Java incorpora el modificador **final**, para que no se pueda modificar.



# Niveles de acceso y visibilidad de variables

- En **Python** o hay *modificadores* como **private** o **protected** (C++/Java).
- Python usa **convenciones**:
  - **\_variable** → uso interno.
  - **\_\_variable** → *name mangling* (se renombra a **\_Clase\_\_variable**) para evitar accesos accidentales.
- Aunque no es seguridad estricta, sí proporciona cierto **encapsulamiento**.



# Medidas en entorno de ejecución

La ejecución es diferente en lenguajes **interpretados y compilados**, debido a que los compilados se ejecutan por lo general en una máquina virtual.

En el caso de **Java**, en este aspecto, podemos decir que es un lenguaje de programación seguro:

- Utiliza **Certificados** para verificar el **autor del código** y su **integridad** ( herramientas jar, keytool y jarsigner).
- **Sistema de control de permisos basado en políticas**, para los programas.



# Medidas en entorno de ejecución en Python

- **Python no implementa separación de privilegios** para reducir la superficie de ataque.
  - El atacante toma el control total del sistema.
- **Mecanismo de excepciones**: evita ejecución errónea que comprometa la aplicación.
- **Virtualenv**: aislamiento de dependencias → reduce riesgo de conflictos o librerías maliciosas.
- **Sandboxing (limitado)**: antiguamente existía **rexec** y **Bastion** para ejecutar código restringido, pero se eliminaron (inseguros).
- Hoy en día, el aislamiento se consigue mejor con **contenedores (Docker, venv, etc.)**.

# Certificados y permisos de ejecución en Python

- **SSL/TLS integrado** en el módulo `ssl`:
  - Verificación de certificados de servidores.
  - Creación de sockets seguros.
- **Módulo subprocess**: permite controlar permisos y privilegios al ejecutar procesos externos (evita inyección de comandos con `shell=True` si se usa mal).
- Python puede integrarse con **SELinux**, **AppArmor** o **permisos del SO** para limitar su ejecución.

# Seguridad con clases

En **Java** tenemos:

- **Cargador de clases** para gestionar los bytecodes de las diferentes clases.
- Verificador de archivos de clases que comprueba estructura interna de clases e integridad de bytecode en carga y en ejecución, lanzando excepción si hay algo que no fuera bien.
- **Gestor de seguridad** (SecurityManager) que determina las **políticas de seguridad activa** y **verificaciones de control de acceso**.



# Seguridad con bytecode y clases en Python

- Python compila a **bytecode (.pyc)** → no contiene metadatos críticos (pero puede ser descompilado fácilmente, no es ofuscación segura).
- No hay un **gestor de seguridad centralizado** (como el *SecurityManager* de Java).
- Existen librerías de **ofuscación y empaquetado** (ej. *pyarmor*, *cython*) para proteger código.



# APIs de seguridad en la librería estándar

En los diferentes lenguajes de programación nos solemos encontrar diferentes APIs (interfaz de programación de aplicaciones) para mejorar los servicios y características en aspectos como:

- Criptografía.
- Seguridad en comunicación.
- Autenticación y autorización.
- Gestión de permisos y sistema.
- Seguridad en la red.

Así nos podemos encontrar:



# APIs de seguridad en la librería estándar en Python

Python incluye módulos muy útiles para ciberseguridad:

- **Autenticación y autorización (indirectamente)**
  - No hay API de auth en la stdlib, pero se integra fácilmente con frameworks ([Flask-Login](#), [Django Auth](#), [OAuthlib](#), [PyJWT](#) (JSON Web Tokens)).
- **Gestión de permisos y sistema**
  - [os.chmod](#), [os.setuid](#), [os.environ](#) → gestión de permisos, usuarios y variables de entorno.
- **Ciberseguridad ofensiva: [scapy](#) (red), [pwntools](#) (exploits), [paramiko](#) (SSH).**



# APIs de seguridad en la librería estándar en Python

- Criptografía y hashing
  - **Hashlib**, **PyNaCl**: hashing y sal.
  - **hmac**: firmas con clave secreta.
  - **secrets**: generación de contraseñas y tokens seguros (mejor que **random**).
  - **crypt**: hash de contraseñas en Unix.
  - **Cryptography**: servicios criptográficos.
- Seguridad en red
  - **ssl**: para sockets seguros TLS.
  - **http.server + ssl**: servidores web seguros.
  - **imaplib**, **smtplib** con TLS.



# Librerías externas muy usadas en Java

En Java tenemos diferentes APIs para, :

- **Criptografía:**
  - Arquitectura Criptográfica de Java (**JCA**).
  - Extensión Criptográfica de Java (**JCE**).
- **Seguridad en la Comunicación:**
  - Extensión de Sockets Seguros (**JSSE**).
  - **Apache WSS4**: Seguridad en servicios web SOAP.
  - **Jose4j**: Trabajo con JSON Web Tokens y firma/cifrado.
- **Autenticación y Autorización:**
  - Servicio de autoautenticación y autorización(**JAAS**).
  - **Spring Security** (OAuth2, JWT, etc)
  - **Apache Shiro** similar a Spring security pero más simple.

# Python

- Python no implementa separación de privilegios para reducir la superficie de ataque.
  - El atacante toma el control total del sistema.
- CPython no verifica si el *bytecode* es seguro.
- CPython tiene una gran superficie de ataque.
  - Construir un *sandbox* sea muy difícil, habría que meter todo CPython en una *sandbox* mejor.
- Tiene muchas funciones y módulos inseguros.





02

# Comparación de infraestructuras de seguridad en diferentes lenguajes.

# Java

- El lenguaje de programación Java tiene varias medidas de protección:
  - Contra acceso ilegal a memoria.
  - Verificación de tipos.
  - Niveles de acceso a miembros, aunque se puede saltar mediante el mecanismo de reflexión.
  - El modificador final.



# Java

- La seguridad de entorno de ejecución de la JVM se resume en:
  - Cargador de clases jerárquico
  - Archivos JAR firmados y bytecodes firmados y verificados al ejecutar.
  - El gestor de seguridad controla permisos para los programas, basado en políticas de seguridad.
  - Juego de APIs de criptografía (JCA y JCE).
  - Extensión de sockets seguros (JSSE).
  - Autenticación y autorización con JAAS.



# .NET

- Los lenguajes Visual Basic® .NET y C# tienen las mismas medidas de protección que Java
  - Contra acceso ilegal a memoria.
  - Verificación de tipos.
  - Niveles de acceso a miembros, aunque se puede saltar mediante el mecanismo de reflexión.
  - El modificador final .
- Después, se compilan a Microsoft Intermediate Language (MSIL), similar a bytecode de Java.



# .NET

- El *Common Language Runtime (CLR)*, equivalente al JRE, tiene una política de seguridad llamada *Code Access Security*, basada en el origen del código, no qué usuario lo ejecuta:
  - Todos los llamantes deben tener permiso suficiente.
  - Por defecto sólo el código local tiene permiso.
- .NET Cryptographic Framework es como JCA y JCE • Windows Communication Foundation (WCF) implementa mecanismos similares a JAAS y JSSE.



# PHP

- Montado en un Apache, hereda los permisos del usuario que lo corre.
  - Por eso no se debe lanzar Apache como root.
- La configuración le permite limitar los ficheros y directorios a los que tiene acceso, pero por defecto no se hace
  - Tiene un módulo OpenSSL que debería ser equivalente a JSSE, JCA y JCE



# PHP

- Montado en un Apache, hereda los permisos del usuario que lo corre.
  - Por eso no se debe lanzar Apache como root.
- La configuración le permite limitar los ficheros y directorios a los que tiene acceso, pero por defecto no se hace
  - Tiene un módulo OpenSSL que debería ser equivalente a JSSE, JCA y JCE



# Comparativa de seguridad de diferentes lenguajes

Aspecto	Python	Java	.NET (C#)	PHP
<b>Gestión de memoria</b>	Automática (GC), sin acceso directo → evita <i>buffer overflows</i> .	JVM controla memoria, verificación de bytecode, sin punteros directos.	CLR (Common Language Runtime) con GC, evita corrupción de memoria.	Manejo automático básico, pero históricamente vulnerable (ej. uso inseguro de funciones nativas).
<b>Verificación de tipos</b>	Tipado dinámico fuerte → lanza excepciones ( <code>TypeError</code> ).	Tipado estático fuerte → chequeo en compilación y ejecución.	Tipado estático fuerte → chequeo en compilación y ejecución.	Débilmente tipado (aunque con <code>strict_types</code> se mejora). Riesgo de confusión de tipos.
<b>Control de acceso / visibilidad</b>	Convenciones ( <code>var</code> , <code>var</code> ), no estrictos.	<code>public, private, protected, default.</code>	<code>public, private, protected, internal.</code>	Visibilidad de funciones/variables limitada; más flexible, menos estricto.
<b>Entorno de ejecución</b>	Virtualenv, aislamiento en contenedores.	JVM con <i>sandboxing</i> (limitado hoy en día).	CLR con <i>Code Access Security</i> (CAS).	Depende del servidor web y configuraciones ( <code>open_basedir</code> , <code>safe_mode</code> – obsoleto).
<b>Certificados y comunicación segura</b>	<code>ssl, cryptography, secrets.</code>	JSSE, KeyStore, Bouncy Castle.	<code>System.Security.Cryptography</code> , soporte nativo TLS.	<code>openssl, libsodium, wrappers de OpenSSL.</code>
<b>Autenticación y autorización</b>	Frameworks externos (Flask-Security, Django Auth, OAuthlib).	JAAS, Spring Security, Apache Shiro.	ASP.NET Identity, Windows Authentication, OAuth2/JWT integrados.	Frameworks (Laravel Auth, Symfony Security, JWT).
<b>Criptografía</b>	<code>hashlib, hmac, cryptography, PyNaCl.</code>	JCA/JCE, Bouncy Castle, Google Tink.	<code>System.Security.Cryptography</code> , BCrypt, AES, RSA.	<code>password_hash(), libsodium, openssl.</code>
<b>Protecciones extra</b>	Excepciones controladas, <code>secrets</code> para entropía fuerte.	Verificación de bytecode, políticas de seguridad, fuerte tipado.	Integración con Windows (Active Directory, ACLs), CAS.	Filtros de entrada/salida, <code>filter_input()</code> , validaciones en frameworks.
<b>APIs externas destacadas</b>	Scapy, Paramiko, PyJWT, cryptography.	OWASP ESAPI, Jose4j, Apache WSS4J.	IdentityServer, OWIN Security.	OWASP Enterprise Security API (ESAPI-PHP), Firebase JWT.



Ver tabla comparativa de seguridad en los lenguajes de programación en  
[https://docs.google.com/document/d/1ilnkvjshb5KUCsYIUiDR\\_wzXNyKxiLyy5bf\\_0C50Xmg/edit?usp=sharing](https://docs.google.com/document/d/1ilnkvjshb5KUCsYIUiDR_wzXNyKxiLyy5bf_0C50Xmg/edit?usp=sharing)

# Bibliografía y Webgrafía

- Comparación de infraestructuras de seguridad - Rafael López García.
- Oracle: "Java Security Libraries" -  
<https://www.oracle.com/java/technologies/javase/javase-tech-security.html>
- Oracle: "Estructura Arquitectónica de Java ES" -  
<https://docs.oracle.com/cd/E19528-01/820-3091/aauav/index.html>
- Seguridad en Java –  
<https://www.uv.es/~sto/cursos/seguridad.java/html/sjava-1.html>
- Security in .NET – <https://learn.microsoft.com/en-us/archive/msdn-magazine/2002/september/net-security-the-security-infrastructure-of-the-clr>

# Bibliografía y Webgrafía

- Windows Communication Foundation –  
<https://learn.microsoft.com/es-es/dotnet/framework/wcf/whats-wcf>
- Practical experience with the .NET crypto API –  
<https://lirias.kuleuven.be/retrieve/34370/>
- Seguridad en Python –  
<https://python-security.readthedocs.io/security.html>
- About half of Python libraries in PyPI may have security issues” –  
[https://www.theregister.com/2021/07/28/python\\_pypi\\_security/](https://www.theregister.com/2021/07/28/python_pypi_security/)
- Manual de PHP –  
<https://www.php.net/manual/es/index.php>



# Gracias!

¿Alguna pregunta?



[informatica.iesvalledeljerteplasencia.es](http://informatica.iesvalledeljerteplasencia.es)



[coordinacion.cenfp@iesvp.es](mailto:coordinacion.cenfp@iesvp.es)



C/ Pedro y Francisco González, s/n  
10600, Plasencia (Cáceres)



927 01 77 74

