Hacking con Python

Whoami

David Cortez

- Backend Developer.
- Security Researcher.
- Entusiasta de la IA.

@davcortez

Tabla de contenidos

Pre-Requisitos

- Python 3x
- Git
- VirtualBox
- IDE
- Subsistema de Windows para Linux (WSL)
- Dirección de correo electrónico temporal

Enfoque y alcance

- Python desde el punto de vista ofensivo.
- Prototipado y creación de scripts.
- Uso de herramientas usadas en Equipos de seguridad.

Introducción a Python

¿Qué es Python?

- Lenguaje de programación interpretado.
- Fácil de usar.
- De tipado dinámico.
- Inventado por Guido Van Rossum en los 80s.
- Permite el modo interactivo.
- Funciona en múltiples sistemas operativos.

¿Por qué Python?

- Multiplataforma
- Facilidad para prototipar y realizar pruebas de concepto
- Buena documentación
- Muchas librerías y herramientas enfocadas en seguridad

El Zen de Python

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Casos de Uso

Python puede ser usado en:

- Automatización y Scripting
- Criptografía
- Análisis Forense
- Análisis de Malware
- Pruebas de seguridad (Pentesting)

Descargar e instalar Python

Depende de nuestro sistema operation

Enlace para descargar: https://www.python.org/downloads/

En Windows es importante:

- Agregar python.exe al PATH
- Darle permisos de administrador para la instalación

Hello World en Python

Ejecutar modo interactivo (shell de python)

```
$ python
```

REPL (Read, Evaluate, Print, Loop)

Sintaxis básica

Identación: útil para definir bloques de código

No usa {} Ni ;

No usar nombres reservados

```python
age = 22

def check_age(*, age: int):
    """Age checker

    Params
    ------------
    age
        the age of a given person
    """
    # Check if the age is greater than 18
    if age > 18:
        print("You are an adult")

```

Variables

Apunta a un dato almacenado (enteros, numeros reales, boleanos, cadenas, y datos + complejos como listas o diccionarios.) en una dirección de memoria.

age = 20

name = “Juan”

colores = [“Azul”, “Negro”, “Rojo”]

estudiante = {“name”: “Carlos”, “age”: 22}

Nombres reservados

Keywords

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

Soft Keywords

```
match      case       _
```

Constantes

Nos sirve para representar valores que no cambiarán

PI = 3.1415926535897793

MAX_AMOUNT = 2000

PEP8 Referencia: https://peps.python.org/pep-0008/

Duck typing

Concepto de programación que aplica a ciertos lenguajes orientados a objetos.

En python:

- Le da igual los tipos de los objetos, lo único que le importan son los métodos.
- No es necesario especificar el tipo de datos.
- Se puede reasignar diferentes tipos de valores a la misma variable.

Duck typing

```
some_var  = 1000

some_var = “Dynamic typing”

some_var = [1, 2, 3, 4, 5]
```

Tipos de datos

Datos primitivos

- Int
- Boolean
- String
- Float

Tipos de datos

Otros tipos

- Dict
- Set
- Tuple
- List

Sentencias de control

- Las sentencias de control gestionan el flujo de un programa.
- Deciden qué código ejecutar bajo diferentes condiciones.

Sentencias de control

Existen diferentes tipos:

- Sentencias Condicionales: if, elif, else
- Sentencias de Bucle: for, while
- Alteración del Flujo de Control: break, continue, pass

Sentencias de control

Sentencias condicionales:

- Sentencia if: Ejecuta un bloque de código si una condición es verdadera.
- Sentencia elif: Verifica otra condición si las anteriores son falsas.
- Sentencia else: Se ejecuta si ninguna de las condiciones anteriores es verdadera.

Sentencias de control

```python
if condition:
    # código a ejecutar si la condición es verdadera
    print("x")
elif another_condition:
    # código a ejecutar si otra_condición es verdadera
    print("y")
else:
    # código a ejecutar si ninguna de las condiciones anteriores es verdadera
    print("z")
```

Sentencias de control

Sentencias de bucle:

- Bucle for: Itera sobre una secuencia (como una lista, tupla o cadena).
- Bucle while: Se repite mientras una condición sea verdadera.

Sentencias de control

```python
# ejemplo de bucle for
for i in range(5):
    print(i)  # Imprime los números del 0 al 4

# ejemplo de bucle while
contador = 0
while contador < 5:
    print(contador)
    contador += 1  # Incrementa el contador en 1
```

Funciones

- Permite organizar el código en bloques de código.
- Empiezan con la palabra clave def.
- Para llamar una función se usa el nombre seguido de un paréntesis, ejemplo: my_func().
- Puede tener parámetros.

Funciones

```python
# Declarando una función
def greet():
    print('Hello World!')


# Llamando una función
greet()



# Declarando una función con parámetros
def greet(name):
    print("Hello", name)


# llamando una función con un argumento
greet("John")
```

Operadores

Operadores de comparación:

- == Igual al
- != No igual a
- < Menor que
- > Mayor que
- <= Menor o igual que
- >= Mayor o igual que

Operadores

Operadores booleanos:

- True and True
- True and False
- False or True
- False or False

Programación orientada a Objetos

- También conocida como OOP en inglés o POO en español.
- Paradigma de programación introducido en los 70s.
- Permite organizar el código en torno a datos u objetos, en lugar de funciones y lógicas. Es decir, un objeto puede ser definido como un campo de datos que tiene atributos y comportamientos únicos.

Programación orientada a Objetos

La programación orientada a objetos está basada en 6 principios:

- Encapsulación.
- Herencia.
- Abstracción.
- Polimorfismo.

Programación orientada a Objetos

La programación orientada a objetos incluye:

- Clases
- Objetos
- Métodos
- Atributos

Excepciones

Nos permiten controlar el comportamiento de un programa cuando se produce un error.

Las principales excepciones definidas en Python son:

- TypeError
- ZeroDivisionError
- OverflowError
- KeyError
- FIleNotFoundError

Módulos

- Un módulo es un archivo que contiene definiciones y declaraciones en Python.
- Cada módulo tiene su propio espacio de nombres privados.
- Los módulos pueden importar otros módulos.
- Nos permiten reutilizar código y organizarlo.

Módulos

Creando mi módulo

```python
# calculator.py
def suma(a, b):
    return a + b

def resta(a, b):
    return a - b
```

Importando mi módulo

```python
# print_results.py
import calculator

print(calculator.suma(4, 3))   # 7
print(calculator.resta(10, 9)) # 1
```

Importando ciertas funciones de mi módulo

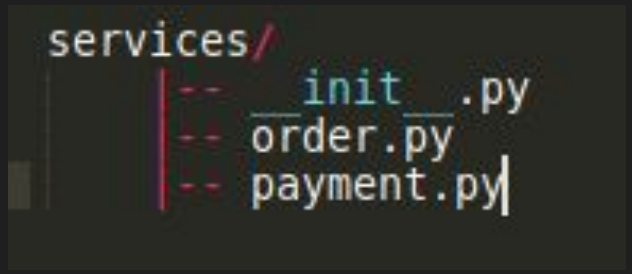
```python
from calculator import suma, resta

print(suma(4, 3))   # 7
print(resta(10, 9)) # 1
```

Paquetes

Un paquete es una carpeta que contiene varios módulos.

Siempre debe contener un archivo __init__.py para que se entienda que se trata de un paquete y no de una carpeta.



Entornos virtuales

Un entorno virtual (virtual environment) es:

- Usado para contener un intérprete específico de python, paquetes, binarios que serán usadas en un proyecto (aplicación o librería).
- Por defecto están aislados de otros entornos virtuales
- Son simples de eliminar y recrear desde cero.
- Permiten instalar y gestionar facilmente paquetes para un proyecto requerido.

Entornos virtuales

python -m venv env -> Crear entorno virtual

source env/bin/activate -> Activar entorno virtual

deactivate -> Desactivar entorno virtual

Gestores de contexto

- Permiten asignar o liberar recursos de una forma expresa. Es decir, se usa un gestor de contexto para que los recursos necesarios del programa se creen y se limpien correctamente.
- Nos permiten escribir código más limpio y legible.
- Permiten una gestión de excepciones simplificada.
- Proporcionan seguridad en el manejo de recursos.

Gestores de contexto

```python
# Usando un context manager
with open('file.txt', 'w') as fichero:
    fichero.write('Hello!')
```

```python
# Sin usar un context manager
fichero = open('file.txt', 'w')
try:
    fichero.write('Hello!')
finally:
    fichero.close()
```

Escribir y Leer archivos

¿Por qué Leer y Escribir Archivos?

- Para almacenar y recuperar datos de forma persistente.
- Para manejar datos de entrada y salida en aplicaciones.

Escribir y Leer archivos

Tipos de Operaciones con Archivos:

- Lectura de Archivos: Abrir y leer el contenido de un archivo.
- Escritura de Archivos: Crear o modificar el contenido de un archivo.

Escribir y Leer archivos

Cómo Leer Archivos en Python:

- Abrir un Archivo para Leer:
  - Modo 'r' para leer (read).
- Métodos Comunes:
  - read(): Lee todo el contenido del archivo.
  - readline(): Lee una línea del archivo.
  - readlines(): Lee todas las líneas y las devuelve como una lista.

Escribir y Leer archivos

```python
# Leer todo el contenido de un archivo
with open('archivo.txt', 'r') as archivo:
    contenido = archivo.read()
    print(contenido)

# Leer línea por línea
with open('archivo.txt', 'r') as archivo:
    for linea in archivo:
        print(linea.strip())
```

Escribir y Leer archivos

Cómo Escribir Archivos en Python:

- Abrir un Archivo para Escribir:
  - Modo 'w' para escribir (write), borra el contenido existente.
  - Modo 'a' para agregar (append), añade al final del archivo.
  - Modo 'r+' para leer y escribir (read and write).
- Escribir en un Archivo:
  - write(): Escribe una cadena en el archivo.
  - writelines(): Escribe una lista de cadenas en el archivo.

Escribir y Leer archivos

```python
# Escribir en un archivo (modo 'w' borra el contenido existente
with open('archivo.txt', 'w') as archivo:
    archivo.write('Hola, Mundo!\n')

# Agregar al final de un archivo (modo 'a')
with open('archivo.txt', 'a') as archivo:
    archivo.write('Esta es una nueva línea.\n')
```

Estructura básica de un programa

```python
# This is a basic Python script to greet the user and calculate the length of their name.

# Function to greet the user
def greet_user():
    print("Hello! Welcome to the Name Length Calculator.")

# Function to get user input
def get_user_name():
    name = input("Please enter your name: ")
    return name

# Function to calculate the length of the name
def calculate_name_length(name):
    length = len(name)
    return length

# Function to display the result
def display_result(name, length):
    print(f"Hello, {name}! Your name has {length} characters.")

# Main function
def main():
    greet_user()  # Greet the user
    user_name = get_user_name()  # Get user input
    name_length = calculate_name_length(user_name)  # Calculate name length
    display_result(user_name, name_length)  # Display the result

# Entry point of the program
if __name__ == "__main__":
    main()  # Call the main function
```

Entry point

Nos permite ejecutar código cuando un archivo se ejecuta como un script, pero no cuando es importado como un módulo.

```
def echo_text(user_input):
    print(user_input)

if __name__ == "__main__":
    user_input = input("insert text here:  ")
    echo_text(user_input=user_input)
```

Usando paquetes de 3ros

Trabajando con PIP:

pip —version -> ver la versión de pip instalada

pip install <package-name> -> instalar un paquete usando pip

pip install -r requirements.txt -> instalar varios paquetes

pip uninstall <package-name> -> desinstalando un paquete

pip freeze -> ver todas las dependencias del proyecto

Recomendaciones

- Instalar solo paquetes confiables (respaldo comunidad).
- Corroborar la reputación del autor.
- Comprobar que el proyecto tengo repositorio público.
- Validar que no se trate de typosquatting.

Trabajando con APIs

Trabajando con el paquete REQUEST:

pip install requests

Referencia: https://pypi.org/project/requests/

Trabajando con APIs

Petición GET

```python
import requests

url = 'https://api.example.com/data'
response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    print(data)
else:
    print(f'Error: {response.status_code}')
```

Trabajando con APIs

Petición POST

```python
import requests

url = 'https://api.example.com/data'
payload = {'key1': 'value1', 'key2': 'value2'}
headers = {'Content-Type': 'application/json'}

response = requests.post(url, json=payload, headers=headers)

if response.status_code == 200:
    data = response.json()
    print(data)
else:
    print(f'Error: {response.status_code}')
```

Ahora a practicar!

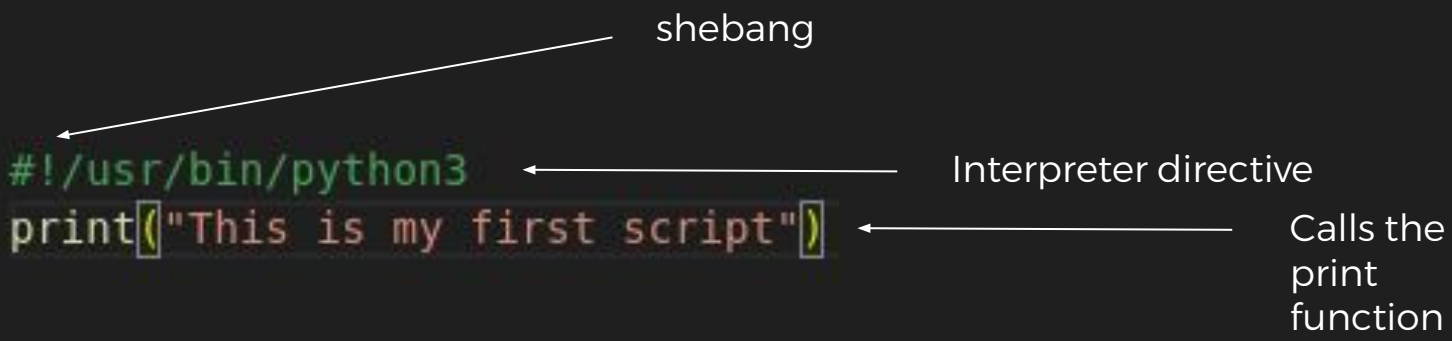
Recurso:

https://pokeapi.co/

Tareas:

- Instalación y configuración inicial.
- Obtener datos específicos.
- Listar habilidades de un Pokémon.
- Manejo de errores.
- Guardar resultados en un archivo.

Empezando con el scripting



Ideas de Proyectos

- Verificador de robustez de contraseña.
- Gestor de claves.
- Scaner de puertos con detección de servicios.
- Cifrado y descifrado de mensajes.
- Generador de hashes

Ideas de Proyectos

- Monitor de integridad de archivos.
- Herramienta para detectar phishing.
- Webspider

Ideas de Proyectos

- Cifrado de archivos
- Simulador de ataques de fuerza bruta
- Fuzzer (básico) para encontrar vulns en aplicaciones web.

Manipulación de datos y automatización de tareas

Módulo OS

- Proporciona funciones para interactuar con el sistema operativo.
- Permite realizar operaciones como manejar archivos y directorios.
- Nos permite automatizar tareas del sistema.
- Es de gran ayuda para manejar archivos y directorios de forma eficiente.

Módulo OS

Funciones básicas del módulo os..

- os.getcwd() -> devuelve el directorio actual de trabajo
- os.chdir(path) -> cambia el directorio de trabajo actual
- os.listdir(path) -> Lista los archivos y directorios en el path especificado
- os.walk(path) -> recorre directorios y subdirectorios de forma eficiente, y permite generar los nombres de los archivos en un árbol de directorios,

Módulo OS

Manipulación de archivos y directorios:

- os.mkdir(dirname) -> crea un nuevo directorio
- os.remove(namefile) -> elimina el archivo especificado
- os.rmdir(dirname) -> elimina el directorio especificado (debe estar vacío)
- os.rename(old_name, new_name) -> renombra un archivo o directorio

Módulo OS

Ejercicios:

- Listar archivos sensibles (db, sql, key, pem)
- Crear un backup de archivos importantes (ejemplo: .log)

Módulo Argparse

- Permite analizar argumentos de línea de comandos en Python
- Facilita la creación de interfaces de líne de comandos (CLI)
- Mejora la usabilidad y la accesibilidad de nuestro script/programa

Módulo Argparse

```python
import argparse

parser = argparse.ArgumentParser(description='Descripción del programa')
parser.add_argument('archivo', type=str, help='Nombre del archivo a procesar')
parser.add_argument('--modo', choices=['lectura', 'escritura'], default='lectura', help='Modo de operación')
args = parser.parse_args()

print(f"Archivo: {args.archivo}")
print(f"Modo: {args.modo}")
```

Módulo Argparse

Ejercicios:

- Listar posibles archivos maliciosos (ps1, dll, bat, vbs, scr)
- Verificar la robustez de una clave
  - Re
  - Import re

Ataques de fuerza bruta

- Un ataque de fuerza bruta es un método para encontrar una contraseña o clave probando todas las combinaciones posibles hasta encontrar la correcta.
- Es una técnica básica pero efectiva para comprometer la seguridad de sistemas.

Ataques de fuerza bruta

- Un ataque de fuerza bruta es un método para encontrar una contraseña o clave probando todas las combinaciones posibles hasta encontrar la correcta.
- Es una técnica básica pero efectiva para comprometer la seguridad de sistemas.

Ataques de fuerza bruta

Y bueno, el objetivo es?

- Acceder a cuentas protegidas por contraseñas o claves.
- Descifrar mensajes encriptados o archivos protegidos.

ZipCracker

- Objetivo: Extraer todo el contenido de un archivo .zip protegido por clave
- Script: crack_zip.py

Requests

- Biblioteca que simplifica el envío de solicitudes HTTP
- Facilita la comunicación con servicios web y APIs
- Útil para interactuar con servicios web y APIs de terceros

Requests

Cómo realizar solicitudes HTTP

- requests.get(url) -> realizar solicitud GET
- requests.post(url, data) -> realizar una solicitud POST con datos
- requests.get(url, params=params) -> pasar parametros como un diccionario para solicitudes GET

Requests

Manejo de respuestas

- response.txt ->acceder al contenido de la respuesta como texto
- response.json() -> acceder al contenido de la respuesta como JSON
- Response.status_code -> para verificar el código de estado de la respuesta
- requests.exceptions.RequestException -> manejar excepciones como para errores de red.

Trabajando con APIs

Petición GET

```python
import requests

url = 'https://api.example.com/data'
response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    print(data)
else:
    print(f'Error: {response.status_code}')
```

Trabajando con APIs

Petición POST

```python
import requests

url = 'https://api.example.com/data'
payload = {'key1': 'value1', 'key2': 'value2'}
headers = {'Content-Type': 'application/json'}

response = requests.post(url, json=payload, headers=headers)

if response.status_code == 200:
    data = response.json()
    print(data)
else:
    print(f'Error: {response.status_code}')
```

Ahora a practicar!

Recurso:

https://pokeapi.co/

Tareas:

- Instalación y configuración inicial.
- Obtener datos específicos.
- Listar habilidades de un Pokémon.
- Manejo de errores.
- Guardar resultados en un archivo.

Beautifulsoup

- Biblioteca de python para analizar documentos HTML y XML
- Permite extraer datos de manera sencilla a partir de documentos HTML/XML
- Proporciona herramientas de navegación y extracción de información de páginas web

Beautifulsoup

Uso básico:

- Creación de un Objeto Beautiful Soup:
    - Utilizar BeautifulSoup(html, 'html.parser') para crear un objeto Beautiful Soup.
- Buscar Elementos en el Documento:
    - Utilizar métodos como find() y find_all() para buscar elementos por etiquetas, atributos, etc.

Beautifulsoup

Uso básico:

- Acceder a los Atributos y Contenido de los Elementos:
  - Utilizar .text para acceder al contenido de un elemento.
  - Utilizar .get('atributo') para acceder al valor de un atributo.

Beautifulsoup

Ejercicios

- Obtener todos los enlaces en una página web
- Webspider simple

Usar el siguiente dominio `http://testphp.vulnweb.com/`

Beautifulsoup

Ejercicios

- Obtener todos los enlaces en una página web
- Webspider simple

Usar el siguiente dominio `http://testphp.vulnweb.com/`

Selenium

Ejercicio

- Leer documentación de selenium para Python.
- Crear herramienta para capturar imagen de un sitio web.

Ethical Hacking

- Práctica de evaluar la seguridad de sistemas y redes utilizando las mismas habilidades y herramientas que los atacantes, pero con el permiso del propietario del sistema.
- Objetivo: Identificar y corregir vulnerabilidades antes de que sean explotadas por atacantes.

Ethical Hacking

NO

- Acceso No Autorizado:
  - Nunca accedas a sistemas sin permiso explícito.
  - Ejemplo: Realizar un escaneo de red en una empresa sin su conocimiento.
- Exceder el Alcance Acordado:
  - No pruebes más allá del alcance definido en el acuerdo con el cliente.
  - Ejemplo: Explorar sistemas adicionales no incluidos en el contrato.

Ethical Hacking

- Uso de Herramientas Maliciosas:
  - No utilices herramientas diseñadas para causar daño.
  - Ejemplo: Instalar malware en sistemas del cliente.

Introducción a las Etapas de Hacking

- **Reconocimiento**: Recopilación de información sobre el objetivo.
- **Escaneo**: Identificación de puertos y servicios abiertos.
- **Enumeración**: Enumeración de servicios y usuarios.
- **Explotación**: Aprovechamiento de vulnerabilidades para obtener acceso.
- **Mantenimiento** del Acceso: Asegurar el acceso continuo.
- **Cobertura de Huellas:** Eliminar rastros de la intrusión.

DISCLAIMER!

ESTE CONTENIDO ES PARA PROPÓSITOS EDUCACIONALES

Recursos

Libros

Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers de TJ O'Connor.

Black Hat Python: Python Programming for Hackers and Pentesters de Justin Seitz.

Automate the Boring Stuff with Python, 2nd Edition: Practical Programming for Total Beginners de Al Sweigart.

Python Documentation de Python Software Foundation