shodan_scan.py

Este archivo contiene una función que consulta a la api de shodan.

```python
import shodan

# API KEY SHodan
api_key= 'gzZ0HHGQtnZ32bjCteGvN4Zz1QAlusHo'

api = shodan.Shodan(api_key)
def consultar_shodan(ip):
    try:
        # Usamos el método host para obtener la información de la IP
        host = api.host(ip)

        # Almacenar la información en un diccionario
        resultado = {
            'ip': host['ip_str'],
            'organizacion': host.get('org', 'N/A'),
            'sistema_operativo': host.get('os', 'N/A'),
            'ubicacion': {
                'ciudad': host.get('city', 'N/A'),
                'pais': host.get('country_name', 'N/A')
            },
            'puertos_abiertos': []
        }
        # Añadir los puertos abiertos de la IP
        for item in host['data']:
            resultado['puertos_abiertos'].append({
                'puerto': item['port'],
                'banner': item['data']
            })
        return resultado

    except shodan.APIError as e:
        print(f"Error al obtener información de la IP {ip}: {e}")
        return None
```

Usamos la librería shodan para interactuar con la api de una forma más cómoda. Haciendo la consulta con el método host podremos dar con los resultados de una IP dada, que almacenaremos en una variable que será devuelta.

crt_sh.py

Este archivo contiene las funciones necesarias para hacer el reconocimiento de subdominios mediante los certificados SSL:

```python
def crt_sh(domain):
    subdomains = []
    target = clear_url(domain)

    req = requests.get("https://crt.sh/?q=%.{d}&output=json".format(d=target))

    if req.status_code != 200:
        print("[X] Information not available!")
        exit(1)

    for value in req.json():
        subdomains.append(value['name_value'])

    # Eliminar duplicados y ordenar los subdominios
    subdomains = sorted(set(subdomains))
    movida = []

    for subdominio in subdomains:
        lista_subdominios = subdominio.split('\n')
        for sub in lista_subdominios:
            movida.append(sub.strip())
    movida = sorted(set(movida))

    with open(domain + ".log", "w") as file:
        for elem in movida:
            file.write(elem + '\n')

    return (movida)
```

Primero creamos una lista para almacenar los subdominios, y recogemos la url y se pasa por una función por si se ha puesto www.x.com eliminar las tres primeras www. Se lanza una petición a crt.sh?q=%.{dominio}Coutput=json para recoger la información devuelta de la página crt.sh. Añadimos todos los nombres de dominio de los certificados a la lista creada previamente y eliminamos los duplicados de la lista.

Podríamos devolver esta lista pero nos daría muchos subdominios iguales por el formato en el que están almacenados, por tanto creamos una nueva lista para devolver sólo los que no estén repetidos.

Para tratar los resultados tenemos que separar los subdominios que están concatenados con '\n' y agregarlos a la lista 'movida' y eliminar espacios y ya los agregamos a la lista.

Luego almacenamos los resultados en un archivo .log para poder ver el output posteriormente.

Sublister encuentra 20 dominios

```
[-] Searching now in PassiveDNS..
[!] Error: Virustotal probably now is blocking our requests
[-] Total Unique Subdomains Found: 20
www.bcnsoluciona.com
aepso.bcnsoluciona.com
www.aepso.bcnsoluciona.com
asfconsultores.bcnsoluciona.com
www.asfconsultores.bcnsoluciona.com
asfconsultors.bcnsoluciona.com
www.asfconsultors.bcnsoluciona.com
autodiscover.bcnsoluciona.com
cpanel.bcnsoluciona.com
cpcalendars.bcnsoluciona.com
cpcontacts.bcnsoluciona.com
ngcybersecurity.io.bcnsoluciona.com
www.ngcybersecurity.io.bcnsoluciona.com
mail.bcnsoluciona.com
staging.bcnsoluciona.com
www.staging.bcnsoluciona.com
webdisk.bcnsoluciona.com
webmail.bcnsoluciona.com
websegura.bcnsoluciona.com
www.websegura.bcnsoluciona.com
```

```
2   aepso.bcnsoluciona.com
3   asfconsultores.bcnsoluciona.com
4   asfconsultors.bcnsoluciona.com
5   autodiscover.bcnsoluciona.com
6   bcnsoluciona.com
7   cpanel.bcnsoluciona.com
8   cpcalendars.bcnsoluciona.com
9   cpcontacts.bcnsoluciona.com
10  mail.bcnsoluciona.com
11  ngcybersecurity.io.bcnsoluciona.com
12  staging.bcnsoluciona.com
13  webdisk.bcnsoluciona.com
14  webmail.bcnsoluciona.com
15  websegura.bcnsoluciona.com
16  www.aepso.bcnsoluciona.com
17  www.asfconsultores.bcnsoluciona.com
18  www.asfconsultors.bcnsoluciona.com
19  www.bcnsoluciona.com
20  www.ngcybersecurity.io.bcnsoluciona.com
21  www.staging.bcnsoluciona.com
22  www.websegura.bcnsoluciona.com
```

leak_email_hunter.py

```python
from intelxapi import intelx

# API KEY
api_key_intelx = '07b06e38-3038-4125-9ba7-a110b5e0beb7'

#  API de Hunter.io
api_key = '759f15f447a35bb8f3c236c6588d487f5f6dde30'

intelx = intelx(api_key_intelx)
def searchLeak(correo):
    resultados = intelx.search(correo)
    output=[]
    if len(resultados['records']) == 0:
        return None
    else:
        for resultado in resultados['records']:
            if resultado['bucket'] == "leaks.private.general":
                output.append(((resultado['name']), (resultado['date'])))

        if len(output) == 0:
            return None
        else:
            return (output)
```

Primero de todo tenemos esta función searchLeak que va a interactuar con la api de intelx, buscando mediante el correo. Después filtra los resultados por el bucket leaks.private.general que son las entradas que devuelve que están relacionadas con fugas de datos. Añadimos una tupla al output con el nombre y la fecha, ya que estamos limitados por la API gratuita.

```python
def recogerEmails(domain):
    # URL base de la API de Hunter.io
    url = f'https://api.hunter.io/v2/domain-search?domain={domain}&api_key={api_key}'

    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        emails = data.get('data', {}).get('emails', [])

        correos = []
        for email in emails:
            correos.append((email['value'], email['confidence']))
        return (correos)
    else:
        print(f"Error: {response.status_code} - {response.text}")
```

Luego esta esta función, recogerEmails que mediante hunter.io buscara el dominio y recogerá los correos que encuentre en la página y su nivel de confianza en una tupla. Estas se devolverán en una lista de tuplas.

```python
def main(domain):
    lista = recogerEmails(domain)
    for elem in lista:
        resultados = searchLeak(elem[0])
        if resultados:
            for resultado in resultados:
                print(f"Correo: {elem[0]} Leak: {resultado[0]} Fecha: {resultado[1]}")
        else:
            print(f"No se han encontrado filtraciones para {elem[0]}.")
```

Luego tenemos la función main que lanzará las dos por orden, recogerá los resultados de recogerEmails (correos del dominio) y hará una petición a intelx para ver si están presentes en algún leak.

terminal_commands.py

```python
import subprocess

#Ejecuta un comando en la terminal y devuelve el resultado
def run_command(command):
    result = subprocess.run(command, shell=True, capture_output=True, text=True)
    return result.stdout


def whois_scan(domain):
    command = f"whois {domain}"
    output = run_command(command)
    with open(f"{domain}-resultados/whois.txt", "w") as f:
        f.write(output)

def nuclei_fuzzer(domain):
    command = f"nf -d {domain}"
    output = run_command(command)
    with open(f"{domain}-resultados/nuclei_fuzzer.txt", "w") as f:
        f.write(output)

def nuclei(domain):
    command = f"nuclei -target {domain}"
    output = run_command(command)
    with open(f"{domain}-resultados/nuclei.txt", "w") as f:
        f.write(output)
```

Como se puede ver en la imagen lo que contiene este archivo son funciones que básicamente lanzan un comando y almacenan el resultado en la carpeta de resultados.

En este archivo se lanza el whois, el nuclei y el nucleifuzzer.

Virustotal_scan.py

Este archivo contiene la llamada a la API de virustotal para comprobar si un dominio es malicioso o no.

```python
def scan_dominio(target):
    url = f"https://www.virustotal.com/api/v3/domains/{target}"

    headers = {
        "accept": "application/json",
        "x-apikey": config.api_key_virustotal
    }

    response = requests.get(url, headers=headers)
    datos = json.loads(response.text)

    analisis = datos["data"]["attributes"]["last_analysis_results"]
    atributos = datos["data"]["attributes"]
    maliciosos = {motor: detalles for motor, detalles in analisis.items() if detalles["category"] == "malicious"}
    sospechosos = {motor: detalles for motor, detalles in analisis.items() if detalles["category"] == "suspicious"}

    guardar_resultados_virustotal(datos)

    print(f"Dominio: {datos['data']['id']}")
    print("Análisis de resultados:")
    print(f"- Total servicios que reportan como malicioso: {len(maliciosos)}")
    print(f"- Total servicios que reportan como sospechoso: {len(sospechosos)}")
```

Tiene dos funciones principales; escanear el dominio y guardar los resultados.

nmap_scan.py

```python
import nmap
from lxml import etree
nm = nmap.PortScanner()

def solo(target, ruta):
    print("\nIniciando escaneo de puertos con versiones...")
    nm.scan(target, arguments='-p- -sS -sV -Pn -n')
    guardar_resultado_xml(f"{ruta}/resultado.xml")
    transform_xml_to_html(f"{ruta}/resultado.xml", "nmap.xsl", f"Resultados nmap escaneo con versiones.html")
    mostrar_resultados()

def main(target, ruta):
    while True:
        print("\nOpciones de escaneo:")
        print("1. Escaneo de puertos con versiones")
        print("2. Escaneo de puertos con scripts y versiones")
        print("3. Escaneo de puertos UDP")
        print("4. Salir")
        opcion = input("Selecciona una opción: ")

        if opcion == "1":
            print("\nIniciando escaneo de puertos con versiones...")
            nm.scan(target, arguments='-p- -sS -sV -Pn -n')
            guardar_resultado_xml(f"{ruta}/resultado_versiones.xml")
```

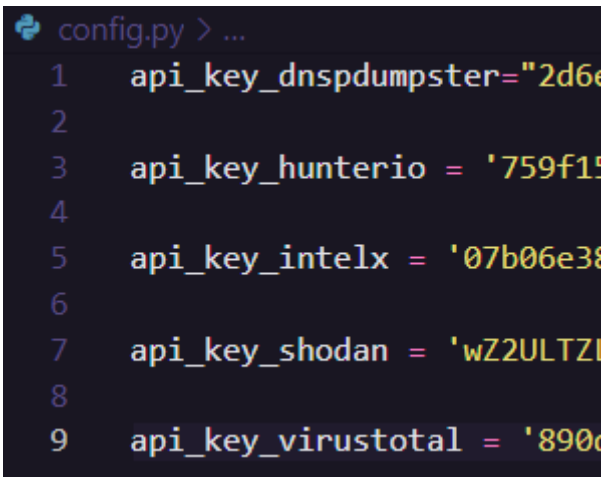
Este archivo tiene 3 funciones, una para lanzar el escaneo directamente, para que no te aparezca el menú, la función main que entra en un menú donde se puede elegir diferentes modos de escaneo (UDP, con scripts y sin scripts) y la otra función es para parsear los resultados que devuelve nmap y hacerlo en un HTML estructurado y legible.

dnsdumpster.py

Este archivo contiene una llamada a la API de dnsdumpster la cual nos devolverá información relevante sobre el dominio, como los NS, MX registros de hosts… Esta información la guarda y hay una función que muestra los datos de una forma legible en la consola y el log se guarda en json.

```python
import requests
import json
import config

api_key = config.api_key_dnspdumpster
def obtener_datos_dominio(dominio):
    url = f"https://api.dnsdumpster.com/domain/{dominio}"

    headers = {
        'X-API-Key': api_key
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        with open (f"{dominio}-resultados/dnsdumpster-{dominio}.json", "w") as file:
                file.write(json.dumps(response.json(), indent=3))
        return response.json()
    else:
        print(f"Error {response.status_code}: No se pudo obtener los datos del dominio.")
        return None


def mostrar_datos(data):
    print("\n--- Registros A ---")
```

config.py



Este archivo de Python contiene las api keys necesarias en algunos de los otros archivos programados, para tenerlas centralizadas en un mismo archivo.

main.py

```python
def obtener_ip(dominio):
    try:
        ip = socket.gethostbyname(dominio)
        return ip
    except socket.gaierror:
        print(f"No se pudo obtener la IP para el dominio: {dominio}")

def ejecutar_todo(target_dominio, target_ip):
    sys.argv = ['cloud_enum.py', '-k', target_dominio, '-k', target_dominio.split('.')[0], '-l', "cloud_enum-"+target_dominio+".log"]
    with concurrent.futures.ThreadPoolExecutor() as executor:
        executor.submit(dnsdumpster.obtener_datos_dominio, target_dominio)
        executor.submit(crt_sh.main, target_dominio)
        executor.submit(email_leak_hunter.main, target_dominio)
        executor.submit(shodan_scan.scan, target_ip)
        executor.submit(cloud_enum.main)
        executor.submit(terminal_commands.whois_scan, target_dominio)
        executor.submit(virustotal_scan.scan_dominio, target_dominio)
        executor.submit(terminal_commands.nuclei_fuzzer)

def main():
    target = input("Introduce el dominio: ")
    target_ip = obtener_ip(target)
    carpeta_resultados = f"{target}-resultados"
    if os.path.exists(carpeta_resultados):
```

Todos estos archivos están centralizados en main.py, creando un menú con todas las opciones que al ser lanzadas se ejecuten las funciones que hemos visto previamente. También hay una función que es obtener_ip para no tener que introducir la dirección IP en caso de que sea necesario para ejecutar x función, como por ejemplo la de shodan, y que solo tengas que introducir el dominio.

También está la función de ejecutar_todo, que como bien indica la función sirve para ejecutar todas las funciones a la vez.

Cuando lanzamos el programa e introducimos un dominio se creará la carpeta **dominio-resultados** que es donde se almacenarán todos los resultados de las funciones que ejecutemos.

Además está integrada el programa cloud_enum, que enumerará (en caso de que hayan) buckets de cloud relacionados al dominio.