Tareas

- ~~Consultar la API para extraer los escaneos disponibles~~
- ~~Extraer un top 5 de vulnerabilidades~~
- ~~Generar un gráfico sectorial~~

A continuación se describen cada una de las tareas en sus propios apartados:

Consultar los escaneos disponibles

> Busca las vulnerabilidades del escaneo sobre el que tienes visibilidad con el usuario facilitado (API Keys) usando la siguiente función:
>
> *https://cloud.tenable.com/scans/{scan_id}*

Usando los métodos descritos en el apartado List scans de la documentación, usé el snippet `curl` que indican para obtener los resultados, añadiendo las claves API al header de la consulta (`X-ApiKeys`).

```
curl --request GET \
     --url https://cloud.tenable.com/scans \
     --header 'X-ApiKeys: accessKey=ca357f77…; secretKey=b8541c26…;' \
     --header 'accept: application/json'
```

El único escaneo disponible es ***Escaneo-VM-Metaexploitable*** **con ID 144.**


```
{
  "scans": [
    {
      "control": true,
      "creation_date": 1742299944,
      "enabled": false,
      "id": 144,
      "last_modification_date": 1742300650,
      "legacy": false,
      "name": "Escaneo-VM-Metaexploitable",
      "owner": "mssp-admin-580924d13e7d@telefonica.com",
      "policy_id": 143,
      "read": true,
      "schedule_uuid": "template-a674002d-55b7-f4f0-0947-99a2056f574391d58a10e4b62f99",
      "shared": true,
      "status": "completed",
      "template_uuid": "ad629e16-03b6-8c1d-cef6-ef8c9dd3c658d24bd260ef5f9e66",
      "has_triggers": false,
      "type": "remote",
      "permissions": 16,
      "user_permissions": 16,
      "uuid": "2e7776d2-eea5-4253-ab87-de35c125bed8",
      "wizard_uuid": "ad629e16-03b6-8c1d-cef6-ef8c9dd3c658d24bd260ef5f9e66",
      "progress": 100,
      "total_targets": 1,
      "status_times": {
```

```
          "initializing": 686,
          "pending": 65279,
          "processing": 217,
          "publishing": 60225,
          "running": 580387
        }
      }
    ],
    "folders": [
      {
        "id": 175,
        "name": "My Scans",
        "type": "main",
        "custom": 0,
        "unread_count": 0,
        "default_tag": 1
      },
      {
        "id": 176,
        "name": "Trash",
        "type": "trash",
        "custom": 0,
        "unread_count": 0,
        "default_tag": 0
      }
    ],
    "timestamp": 1749809451
  }
```


Concluyo que esto satisface la primera tarea, ya que he usado la URL descrita para obtener los escaneos disponibles.

Extraer el top 5 de vulnerabilidades

> Saca un Top 5 de vulnerabilidades por plugin family para aquellas vulnerabilidades con Severidad mayor o igual que 1.
>
> Pista: es necesario que en el código incluyas como obtener el `scan_id`.

Una vez obtenido el ID, uso la misma consulta `curl` de la tarea anterior -pero esta vez filtrando por el escaneo específico-, con el fin de revisar el contenido del JSON:

```
curl --request GET \
     --url https://cloud.tenable.com/scans/144 \
     --header 'X-ApiKeys: accessKey=ca357f77…; secretKey=b8541c26…;' \
     --header 'accept: application/json'
```


```
{
  "info": {
    "owner": "mssp-admin-580924d13e7d@telefonica.com",
    "name": "Escaneo-VM-Metaexploitable",
    "no_target": false,
    "folder_id": null,
    "control": true,
```

```
      "user_permissions": 16,
      "schedule_uuid": "template-a674002d-55b7-f4f0-0947-99a2056f574391d58a10e4b62f99",
      "edit_allowed": false,
      "scanner_name": "DESKTOP-VT6SBV3",
      "policy": "Advanced Network Scan",
      "shared": true,
      "object_id": 144,
      "acls": null,
      "hostcount": 1,
      "uuid": "2e7776d2-eea5-4253-ab87-de35c125bed8",
      "status": "completed",
      "scan_type": "remote",
      "targets": "192.168.1.182",
      "alt_targets_used": false,
      "pci-can-upload": false,
      "scan_start": 1742299944,
      "timestamp": 1742300650,
      "is_archived": false,
      "reindexing": false,
      "scan_end": 1742300650,
      "haskb": true,
      "hasauditrail": false,
      "scanner_start": null,
      "scanner_end": null
    },
    "hosts": [...],
    "vulnerabilities": [...],
    "comphosts": [],
    "compliance": [],
    "filters": [...],
    "history": [...],
    "notes": [],
    "remediations": {...}
  }
```


La información a procesar se vuelve más compleja, por lo que me pongo a pensar cómo afrontar el problema y, leyendo la documentación, observo los ejemplos de interacción con la API usando Python, donde se procesan las peticiones usando `requests`.

Esto no me parece mal, pero aunque es una forma asequible, pienso que debería existir algún módulo de Python para este tipo de situaciones, al igual que con otras tecnologías.

Buscando en Internet encuentro la documentación para el módulo pyTenable (1.7.5), disponible en https://github.com/tenable/pyTenable y perteneciente a la organización de Tenable, verificada por GitHub[1], por lo que **considero que es oficial y seguro usar `pyTenable` para desarrollar un script de Python** con el que procesar la información.

- https://github.com/tenable
- https://developer.tenable.com/docs/introduction-to-pytenable

Para no complicarme demasiado, decidí hacer el script poco a poco y con todas las instrucciones de forma secuencial, para **más tarde mejorarlo y hacerlo más legible**;

lo único que sí se tuvo en cuenta desde el inicio es que fuera genérico, pudiéndose usar para todos los escaneos de un usuario.

Hasta este punto, el script[2] era el siguiente:

```python
# Módulos necesarios
import os

from collections import defaultdict
from dotenv import load_dotenv
from tenable.io import TenableIO

# Cargar variables del entorno virtual '.env'
load_dotenv()

ACCESS_KEY = os.getenv('ACCESS')
SECRET_KEY = os.getenv('SECRET')

# Instanciar el acceso a Tenable
tenable = TenableIO(access_key=ACCESS_KEY, secret_key=SECRET_KEY)

# Buscar todos los escaneos a los que tengo acceso
for scan in tenable.scans.list():
    scan_id = scan["id"]

    print(f'Procesando #{scan_id}: {scan["name"]}...')

    # Obtener los resultados de los escáneres
    results = tenable.scans.results(scan_id)

    # Extraer las vulnerabilidades del resultado
    vulns = results.get('vulnerabilities', [])

    # Filtrar las vulnerabilidades
    filtered = [ v for v in vulns if 1.0 ⩽ v.get('severity', 0) ]

    family_counts = defaultdict(int)

    # Contar las vulnerabilidades por el campo 'plugin_family'
    for vuln in filtered:
        family = vuln.get('plugin_family', '-')
        count = vuln.get('count', 0)

        family_counts[family] += count

    # Ordenar y obtener el Top 5 de plugin families con más vulnerabilidades
    top5 = sorted(family_counts.items(), key=lambda x: x[1], reverse=True)[:5]

    print(f"Escaneo #{scan_id} procesado.\n")

    for i, (family, total) in enumerate(top5, 1):
        print(f"\t{i}. {family}: {total} vulnerabilidades")
```

Empleé un fichero `.env` para ocultar las claves API, y se añadiría a un `.gitignore` en un supuesto repositorio para evitar filtraciones.

El módulo funciona de forma muy similar a las peticiones a las URL, y usa la ruta `https://cloud.tenable.com` por defecto, por eso no especifico ninguna en el objeto `TenableIO()`, solo las claves.

Mi objetivo fue crear una estructura de datos como un diccionario, que contuviera:

- Clave: el campo `plugin_family`.
- Valor: la cantidad de vulnerabilidades asociadas.
  - Usando el campo `count`[3] del campo `vulnerabilities`.

Empleé `defaultdict(int)` porque creaba claves automáticamente, con valor 0 por defecto, de forma que evitaba controlar el primer caso de cada nuevo `plugin_family` y dejaba el código menos verboso.

Por último, solo era necesario ordenar la estructura anterior de mayor a menor cantidad de vulnerabilidades, para extraer las 5 primeras `plugin_family`.

La salida del script anterior, usando mis claves, era:

```
Procesando #144: Escaneo-VM-Metaexploitable...
Escaneo #144 procesado.

    1. General: 17 vulnerabilidades
    2. Misc.: 10 vulnerabilidades
    3. Service detection: 8 vulnerabilidades
    4. DNS: 5 vulnerabilidades
    5. Gain a shell remotely: 4 vulnerabilidades
```

Concluyo que esto satisface la segunda tarea, pues el script muestra cómo se obtiene el escaneo, cómo se filtran las vulnerabilidades y cómo se genera el top 5.

Generar un gráfico sectorial

> Genera un gráfico sectorial usando la librería `matplotlib` que muestre las vulnerabilidades del escaneo sobre el que tiene visibilidad con el usuario facilitado.
>
> - Críticas, severidad 9.0 → 10.0
> - Altas, severidad 7.0 → 8.9
> - Medias, severidad 4.0 → 6.9
> - Bajas, severidad 0.1 → 3.9

Para esta última tarea empleé parte de un código que usé anteriormente en un pequeño caso de uso de `matplotlib`; aunque conocía el módulo, no había trabajado mucho con él, solo en algunas prácticas de la universidad y casos concretos de cursos.

Al script anterior, le añadí este código:

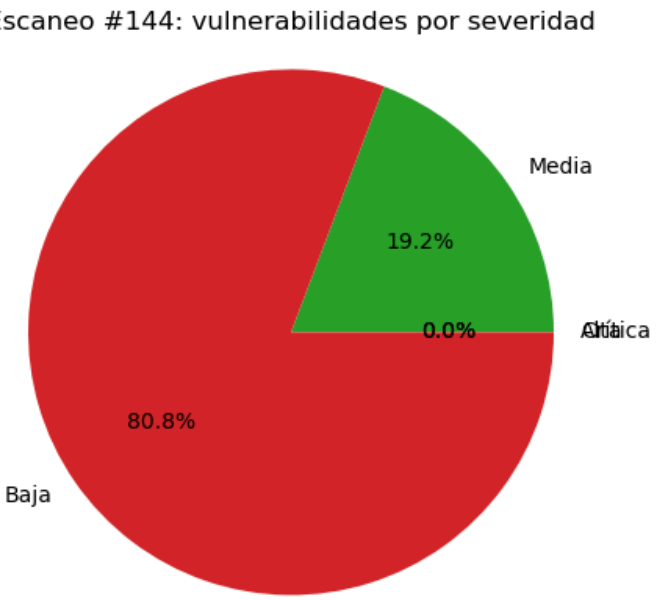
```
import matplotlib.pyplot as plt
```

```
    """
    [...] Contenido del script anterior [...]
    """

        # Categorizar según severity
        categories = {'Crítica': 0, 'Alta': 0, 'Media': 0, 'Baja': 0}

        for v in vulns:
            cnt = v.get('count', 0)
            sev = v.get('severity', 0)

            if 9.0 ⩽ sev ⩽ 10.0: categories['Crítica'] += cnt
            elif 7.0 ⩽ sev < 9.0: categories['Alta'] += cnt
            elif 4.0 ⩽ sev < 7.0: categories['Media'] += cnt
            elif 0.1 ⩽ sev < 4.0: categories['Baja'] += cnt

        print('\n\tCategorías de vulnerabilidades:\n')

        for cat, cnt in categories.items():
            print(f'\t{cat}: {cnt}')

        categories_filtered = {k: v for k, v in categories.items() if v > 0}

        # Configuración del gráfico
        labels = list(categories_filtered.keys())
        sizes = list(categories_filtered.values())

        plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=0)
        plt.axis('equal')
        plt.title(f'Escaneo #{scan_id}: vulnerabilidades por severidad')
        plt.show()
```

Lo único que hice fue crear un diccionario, esta vez usando `{}` porque sí conocía las claves, para usarlo para configurar los valores del gráfico.

Usando los campos `count` y `severity` del campo `vulnerabilities` de la consulta -que no está filtrada, al contrario que la tarea anterior-, añado al diccionario el conteo de cada tipo de vulnerabilidad.

Por último, antes de generar el gráfico, decidí descartar las categorías de la gráfica con valor 0, para evitar cosas como esta:



Busqué en la documentación de `matplotlib` algún parámetro que hiciera esto automáticamente, pero no terminé de encontrar nada, por lo que decidí dejar mi opción.

Antes de entregar el código, tomé la decisión de guardar el fichero como imagen, así que ese sería un cambio que añadiría después.

La salida del script, usando mis claves, era:

```
Procesando #144: Escaneo-VM-Metaexploitable...
Escaneo #144 procesado.


    Top 5 de familias de vulnerabilidades:

    1. General: 17 vulnerabilidades
    2. Misc.: 10 vulnerabilidades
    3. Service detection: 8 vulnerabilidades
    4. DNS: 5 vulnerabilidades
    5. Gain a shell remotely: 4 vulnerabilidades

    Categorías de vulnerabilidades:

    Crítica: 0
    Alta: 0
    Media: 10
    Baja: 42
```

Concluyo que esto satisface la última tarea del ejercicio, pues el gráfico se genera correctamente y se usan todas las vulnerabilidades del escaneo para generarlo.

Resultado final

Esta última sección aborda los cambios introducidos al script que se fue describiendo en los apartados anteriores, así como la ejecución del mismo en aspectos generales.

Script

Tras realizar las tareas anteriores y asegurarme de que funcionaba y cumplía las funciones solicitadas, decidí hacer el script más legible dividiendo el código en 2 funciones principales, una para cada tarea 2 y 3 -ya que la tarea 1 considero que era simplemente poder acceder a los datos mediante la API-.

La función para extraer el top 5, ahora extrae el top n.

La función de generación del gráfico devuelve la ruta donde este se almacena, para que eso pueda usarse como un controlador y detectar posibles errores en su generación.

> También marqué como `# TODO` las líneas que representan las tareas.

main.py

```python
# Módulos necesarios
import matplotlib.pyplot as plt                 # Crear gráficas
import os                                       # Acceder al sistema operativo

from collections import defaultdict             # Manejar colecciones de datos
from dotenv import load_dotenv                  # Cargar variables de entorno virtuali
from tenable.io import TenableIO                # Operar con el endpoint Tenable (API)

# Cargar variables del entorno virtual '.env'
load_dotenv()

ACCESS_KEY = os.getenv('ACCESS')
SECRET_KEY = os.getenv('SECRET')

def top_n(scan_id: int, n: int):
    """
    Muestra por pantalla el Top N de 'plugin_family' en función de
    la cantidad de vulnerabilidades.

    :param scan_id:    El ID del escaneo del que procesar las vulnerabilidades.
    :param n:          El número del top que se quiere generar.
    """
    # Extraer las vulnerabilidades del resultado del escaneo (por defecto: '[]')
    vulns = tenable.scans.results(scan_id).get('vulnerabilities', [])

    # Filtrar las vulnerabilidades con 'severity' de 1 o más
    filtered = [ v for v in vulns if 1.0 ⩽ v.get('severity', 0) ]

    groups = defaultdict(int)

    # Contar las vulnerabilidades por el campo 'plugin_family'
    for vuln in filtered:
        family = vuln.get('plugin_family', '-')     # (por defecto: '-')
        count = vuln.get('count', 0)                # (por defecto: '0')

        groups[family] += count
```

```python

    # Ordenar y obtener el Top N de 'plugin_family' con más vulnerabilidades
    top = sorted(groups.items(), key=lambda x: x[1], reverse=True)[:n]

    print(f"\n\tTop {n} según 'plugin_family':\n")

    for i, (family, total) in enumerate(top, 1):
        print(f'\t{i}. {family}: {total} vulnerabilidades')

def pie_chart(scan_id: int) -> str:
    """
    Genera la imagen de un gráfico de vulnerabilidades cateforizadas por
    severidad y muestra por pantalla los datos del gráfico.

    El gráfico no muestra las categorías con 0 vulnerabilidades.

    :param scan_id:    El ID del escaneo del que procesar las vulnerabilidades.

    :return:    La ruta de la imagen generada (fichero), o '' si hubo un error.
    """
    try:
        # Extraer las vulnerabilidades del resultado del escaneo (por defecto: '[]')
        vulns = tenable.scans.results(scan_id).get('vulnerabilities', [])

        # Categorías (severidad)
        categories = {'Crítica': 0, 'Alta': 0, 'Media': 0, 'Baja': 0}

        # Contar las vulnerabilidades por categoría
        for v in vulns:
            count = v.get('count', 0)
            severity = v.get('severity', 0)

            if 9.0 ⩽ severity ⩽ 10.0: categories['Crítica'] += count
            elif 7.0 ⩽ severity < 9.0: categories['Alta'] += count
            elif 4.0 ⩽ severity < 7.0: categories['Media'] += count
            elif 0.1 ⩽ severity < 4.0: categories['Baja'] += count

        print('\n\tCategorías de vulnerabilidades:\n')

        for cat, cnt in categories.items():
            print(f'\t{cat}: {cnt}')

        # Eliminar las categorías sin vulnerabilidades (mayor legibilidad)
        filtered = {k: v for k, v in categories.items() if 0 < v}

        if not filtered:
            print('\n\tNo hay suficientes datos para generar el gráfico.')
            return ''

        labels = list(filtered.keys())
        sizes = list(filtered.values())

        # Configuración del gráfico
        plt.figure(figsize=(6, 6))
        plt.title(f'Escaneo #{scan_id}: vulnerabilidades categorizadas')
        plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=0)
        plt.axis('equal')
        # plt.show()

        # Ruta del fichero
```

```
            path = os.path.join(os.getcwd(), f'{scan_id}_categorized.png')

            # Guardar la imagen
            plt.savefig(path)
            plt.close()

            print(f"\n\tGráfico guardado en '{path}'.")
            return path

        except Exception as e:
            print(f'Error al general el gráfico: "{e}".')
            return ''

    if __name__ == '__main__':
        # Instanciar el acceso a Tenable
        tenable = TenableIO(access_key=ACCESS_KEY, secret_key=SECRET_KEY)

        # Iterar sobre todos los escaneos a los que se tiene acceso
        for scan in tenable.scans.list():   # TODO: tarea 1/3
            scan_id = scan["id"]

            print(f'#{scan_id}: "{scan["name"]}"')

            top_n(scan_id, 5)   # TODO: tarea 2/3
            pie_chart(scan_id)  # TODO: tarea 3/3
```

La salida, finalmente, es:

```
#144: "Escaneo-VM-Metaexploitable"

    Top 5 según 'plugin_family':

    1. General: 17 vulnerabilidades
    2. Misc.: 10 vulnerabilidades
    3. Service detection: 8 vulnerabilidades
    4. DNS: 5 vulnerabilidades
    5. Gain a shell remotely: 4 vulnerabilidades

    Categorías de vulnerabilidades:

    Crítica: 0
    Alta: 0
    Media: 10
    Baja: 42

    Gráfico guardado en '/home/galan/Flakes/VASS/144_categorized.png'.
```

Concluyo que esta versión del script **cumple todas las tareas del ejercicio**, descritas anteriormente en cada apartado de la sección *Tareas*.

Ejecución

Yo uso el sistema operativo NixOS, por lo que puedo generar “entornos virtuales” de forma nativa, llamados Nix Flakes, de ahí la existencia de los ficheros `flake.nix` y `flake.lock`; eso es lo que he usado para manejar y aislar las dependecias de este “proyecto”[4].

Eso es equivalente a generar un contenedor Docker

No obstante, el proceso anterior también puede replicarse en entornos de desarrollo convencionales creando un entorno virtual de Python e instalando en él los módulos necesarios.

Suponiendo una distribución Linux convencional

```
python -m venv .venv
source .venv/bin/activate
pip install matplotlib pytenable python-dotenv
```

Documentación

Este documento se generado usando Obsidian, la herramienta que uso para mis notas personales, y que posee una función de exportación de notas Markdown a documentos PDF.

Adjuntos

- Claves API (`.env` oculto)
- Gráfico resultante
- Script final
- Ficheros de Nix Flake
- Esta documentación

Adicionalmente, dejo incluida la carpeta `resources/` con los resultados de las consultas que usé para ir comprobando el funcionamiento de mi script.

---

1. GitHub Enterprise Cloud Docs: Verificar o aprobar un dominio para tu organización ↩
2. Este script no se corresponde con el entregado, sino a una versión previa. ↩
3. Representan ocurrencias de dicha vulnerabilidad. Estoy no es lo mismo que “cantidad de hosts” con esa vulnerabilidad. ↩
4. Como curiosidad: para activar el entorno usando Nix, solo es necesario ejecutar `nix develop`, pero se debe tener instalado Nix en primer lugar con la opción de Flakes habilitada. ↩