Scrapper

Abraham Luna

Diciembre 2020

Contents

1 Introducción

Web scraping es el raspado de datos que se utiliza para extraer datos de sitios web. El término se refiere a procesos automatizados implementados mediante un bot. Es una forma de copia, en la que se recopilan y copian datos específicos de la web, generalmente en una base de datos local central u hoja de cálculo, para su posterior recuperación o análisis.
Hay métodos que utilizan algunos sitios web para evitar el web scraping, como detectar y evitar que los robots vean sus páginas o utilizar captchas.
Nuestro scraper utiliza los siguientes buscadores para la obtención de datos:

- Google
- Yahoo
- Bing
- Ecosia
- DuckDuckGo
- Aol

2 Modulo de anonimización

El software desarrollado utiliza tor para enviar las peticiones a los buscadores y cambia de ip cuando se detecta que los buscadores implementan un captcha o algún mecanismo propio de estos para evitar el uso de consultas automatizadas. El siguiente bloque de código de nuestra herramienta define los headers que se usarán en las peticiones *Get* a los buscadores, el puerto donde esta escuchando tor y por el que mandaremos las peticiones, y la función que nos permite cambiar la ip que estamos usando para las peticiones.

```
controller = Controller.from_port() #controlador para interactuar con el socket de tor
proxie = {'https':'socks5://127.0.0.1:9050'} #tor
headers_Get = {
        'User-Agent': UserAgent().random,
        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
        'Accept-Language': 'en-US,en;q=0.5',
        'Accept-Encoding': 'gzip, deflate',
        'DNT': '1',
        'Connection': 'keep-alive',
        'Upgrade-Insecure-Requests': '1'
    }

def new_ip():
    controller.authenticate(password = 'hola123.,')# autenticación de acuerdo a lo establecido en /etc/tor/torrc
    controller.signal(Signal.NEWNYM)#renovación de ip
```

Cabe mencionar que la autenticación a tor que se esta usando es con contraseña pues esta forma no requiere privilegios elevados como la autenticación por cookie. El hash de esta contraseña está en el archvio */etc/tor/torrc* y se añade de forma automatica con nuestro script de instalación.

Aqui se demuestra el funcionamiento de este bloque:

3 Módulo de búsqueda

Para el módulo de búsqueda se implementó de manera que se recibiera el query y el buscador con el cual se va harán las peticiones. Como las peticiones a cada buscador son distintas, se incluyeron if donde se encuentran las distintas opciones de navegadores con los parámetros que necesitan estos.
Para la obtención de las 50 páginas se realiza un ciclo while donde termina una vez que la longitud de output sea mayor a 50 o cuando hayamos intentado paginar 7 veces y aún no tengamos 50 resultados. Una vez que nos banea un buscador(la funcion banned() lo comprueba) llamamos a la función new_ip() y damos un sleep de 3 segundos para hacer una nueva petición de modo que no sobre carguemos a tor.
Finalmente, con la función get_urls(), obtenemos los resultados por buscador.

```python
def search(query,engine):
    aux=0
    page=0
    page_ecosia=0
    output = []
    query = query.replace(' ', '+')
    print(f"Busqueda utilizando {engine}")
    if engine=="duckduckgo":
        url = f'https://html.duckduckgo.com/html/?q={query}'
    while len(output) < 50 and page < 100:
        paging=engine_paging[engine]
        if engine_paging[engine]=='p=':
            paging+=str(page_ecosia)
        else:
            paging+=str(page)
        if engine=="google" or engine=="bing":
            url = f'https://www.{engine}.com/search?q={query}&{paging}'
        if engine=="yahoo" or engine=="aol":
            url = f'https://search.{engine}.com/search?q={query}&{paging}'
        if engine=="ecosia":
            url = f'https://www.ecosia.org/search?q={query}&{paging}'
        response = requests.get(url, headers=headers_Get,proxies=proxie)#petición con headers definidos a traves de tor
        while banned(response.text): #baneado
            print("Ip rechazada")
            new_ip()
            time.sleep(3) #para no sobrecargar tanto a tor
            print(requests.get('https://httpbin.org/ip', proxies=proxie).text)#ip usada en nuestras peticiones
            response = requests.get(url, headers=headers_Get,proxies=proxie)
        print("---------------Changing page----------------")
        page+=10#cambio de página de resultados
        page_ecosia+=1
        for site in get_urls(engine,response.text):
            output.append(site)
    return output
```

La función get_urls() utiliza BeautifulSoup para parsear el contenido de la respuesta y asi poder movernos sobre las etiquetas de una manera más cómoda. Cada buscador utiliza un nombre de clase para div diferente y todos estos estan contemplados aqui, una vez que se asigna la clase en función del buscador, se procede a extraer las urls encontradas.
Para aol y yahoo se debe hacer un tratamiento a las cadenas obtenidas pues estas no contienen solo la url.

```python
def get_urls(engine,response):
    soup = BeautifulSoup(response, "html.parser")
    tag="div"
    class_dict={}
    output=[]
    if engine=='google':
        class_dict['class']='r'
    if engine=='bing':
        tag="li"
        class_dict['class']='b_algo'
    if engine=='duckduckgo':
        class_dict['class']='links_main links_deep result__body'
    if engine=='ecosia':
        class_dict['class']='result-firstline-container'
    if engine=='aol':
        class_dict['class']='compTitle options-toggle'
    if engine=='yahoo':
        class_dict['class']='dd algo algo-sr relsrch Sr'
    for tag_div in soup.find_all(tag, class_dict):
        tag_a = tag_div.find_all('a')#obtemos las etiquetas <a href>
        if tag_a:
            output.append(tag_a[0]['href'])#obtenemos la url
    return output
```

La funcion banned busca palabras clave en las respuestas de las peticiones que indican la presencia de un captcha o algún tipo de error/baneo.

```python
def banned(text):
    if 'CAPTCHA' in text: #Google y ecosia
        return True
    if 'If this error persists, please let us know: error-lite@duckduckgo.com' in text: #Duckduckgo
        return True
    if 'Verizon' in text: #aol y yahoo
        return True
    return False
```

Para los operadores mail e ip, se hacen pequeños ajustes a las querys dado que estas funcionan diferente en los buscadores. Google no soporta ip y no se encontró forma de emularlo, asi que se evita la ejecución de este operador en dicho buscador. Mail solo es soportado por google, pero los otros buscadores muestran resultados si se buscan *mail dominio* con un *and* implicito.

```python
if arguments.ip:
    query=f"ip:{arguments.ip}"
else:
    query=arguments.query

if "mail:" in query:
    index = query.find('mail:')
    alter_query = query[:index+4] + ' ' + query[index+5:]
    mail=True
    print(query)
```

Buscar correo en los resultados se hace con una experesión regular que extrae cadenas de la respuesta

```
mail_regex = r"[a-zA-Z0-9_.+-áéíóú]+@[a-zA-Z0-9-áéíóú]+\.[a-zA-Z0-9-.áéíóú]+"
results={}
alter_query=""
mail=False

def get_mails(description):
    return re.findall(mail_regex, description, re.I)
```

La función limpieza_url hace uso del modulo urlparse en la libreria urllib.parse. La función limpieza se ocupa en dos situaciones especificas: los buscadores *Yahoo* y *Aol* devulven los resultados de forma extraña y con mucha basura innecesaria, si la bandera **p** o **params** se encuentra habilitada, los parametros de la url deberán ser eliminados del reporte y solo se documentará la url encontrada.
El funcionamiento es sencillo, se parsea la url se recorta con divisores identificados y constantes en las peticiones, se traducen los caracteres url a texto plano y se devuelven las cadenas. Para la segunda opción solamente se pasa a través de la función y se regresan las seciones de la url necesarias.

```
def limpieza_url(url):
    resultados = []
    if type(url) == list:
        for u in url:
            url_limpia, url_estructurada = "", ""
            for dot in urlparse(u).path.split("RU=")[1].split('%3a'):
                url_limpia += dot + ":"
            for slash in url_limpia.split('%2f'):
                url_estructurada += slash + "/"
            resultados.append(url_estructurada.split("RK")[0])
        return resultados
    else:
        url_limpia = urlparse(url)
        return url_limpia.scheme + "://" + url_limpia.netloc + "/"
```

Por último, notemos que programa seguirá cambiando de ip hasta poder tener un request exitoso.

```
Busqueda utilizando yahoo
Ip rechazada
{
  "origin": "51.77.135.89"
}

Ip rechazada
{
  "origin": "209.141.41.103"
}
```

4 Módulo de reporte

El módulo de reporte esta compuesto por dos funciones: la función **reporte** y la función **crea_archivo**. La función **reporte** hace uso de 5 parametros, que no

son forzosos para la ejecución de la herramienta, los parametros son necesarios dependiento de la tarea que vaya a realizar la herramienta.
3 parametros son necesarios para el formato del reporte (xml, html y txt) y los otros dos son necesarios para la inicialización y termino del archivo.

```
def reporte(formato=str, datos=str, option=str, end=False, start=False):
    if start == True:
        reporte = "<!DOCTYPE html>\n<html>\n<body>\n" if formato == "html" else ""
        reporte = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<scrapper>\n" if formato == "xml" else ""
    else:
        reporte = ""
```

La funcionalidad de cada parametro es la siguiente:

- **formato:** Es de tipo cadena y se le pasa el formato deseado del reporte.
- **datos:** Es la información que se va a formatear, parámetro de tipo cadena.
- **option:** Es la jerarquía seleccionada para la información. En este caso tenemos tres jerarquías (header, section y parrafo), este parámetro es de tipo cadena.
- **end:** Le indica a la función que el archivo esta completo y se debe llamar a la función crea_archivo, es de tipo booleano y por defecto su valor es *False*.
- **start:** Le indica a la función si se comienza un archivo nuevo, se inicializa la variable con los inicializadores de archivo para cada formato. Parámetro de tipo boolenado, valor por defecto False.

EL formato de cadenas se hace de la siguiente manera, dependiendo del tipo de archivo a los datos se le concatenan las etiquetas y se retorna la cadena de datos ya formateada.Si el formato seleccionado no es valido, manda una advertencia.

```
if formato == "xml":
    if option == "header" and start == True:
        reporte += "\t<query nombre=\"" + datos + "\">"
    elif option == "header":
        reporte += "\t</query>\n\t<query nombre=\"" + datos + "\">"
    elif option == "section":
        reporte += "\t\t<url direccion=" + datos + ">"
    elif option == "parrafo":
        reporte += "\t\t\t" + datos + "\n\t\t</url>"
elif formato == "html":
    if option == "header":
        reporte += "<h1>" + datos + "</h1>"
    elif option == "parrafo":
        reporte += "<p>" + datos + "</p>"
    elif option == "section":
        reporte += "<h3>" + datos + "</h3>"
elif formato == "txt":
    if option == "header":
        reporte += datos.upper()
    elif option == "section":
        reporte += "\t" + datos
    elif option == "parrafo":
        reporte += "\t\t" + datos
else:
    print("Formato seleccionado incorrecto!\n\tFomatos permitidos: xml, html, txt\n\tNo se crea el archivo")
    exit
```

Finalmente se evalua si se desea terminar el archivo, se manda llamar la función *crea_archivo* y se termina la ejecución.

```
    if end == False:
        return reporte
    else:
        reporte += datos
        if formato == "xml":
            reporte += "\t</query>\n</scrapper>"
            crea_archivo(reporte, formato)
        elif formato == "html":
            reporte += "</body>\n</html>"
            crea_archivo(reporte, formato)
        else:
            crea_archivo(reporte, formato)

def crea_archivo(datos=str, formato=str):
    nombre = "scrapper_report." + formato
    with open(nombre, "w") as f:
        f.writelines(datos)
```

Un ejemplo de ejecución sería el siguiente:

```
if __name__ == '__main__':
    formato = "xml"
    r = ""
    r += reporte(formato, "Hola", "header", start=True)
    r += reporte(formato, "Primer Programa en reporte", "section")
    r += reporte(formato, "Se puede agregar cuanta información sea necesaria", "parrafo")
    r += reporte(formato, "Se pueden crear varios headers con contenido", "header")
    r += reporte(formato, "Se puede agregar un descriptor por cada header", "section")
    r += reporte(formato, "Una explicación breve de lo que sucede", "parrafo")
    r += reporte(formato, "Se puede agregar otro header", "header")
    r += reporte(formato, "Más información dentro del header", "section")
    r += reporte(formato, "O vacío", "parrafo")
    reporte(formato, r, "", end=True)
```

Los archivos resultantes para cada formato son los siguientes:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scrapper>
    <query nombre="Hola">
        <url direccion=Primer Programa en reporte>
            Se puede agregar cuanta informaci□n sea necesaria
        </url>
    </query>
    <query nombre="Se pueden crear varios headers con contenido">
        <url direccion=Se puede agregar un descriptor por cada header>
            Una explicaci□n breve de lo que sucede
        </url>
    </query>
    <query nombre="Se puede agregar otro header">
        <url direccion=M□s informaci□n dentro del header>
            O vac□o
        </url>
    </query>
</scrapper>
```

Figure 1: scrapper_report.xml

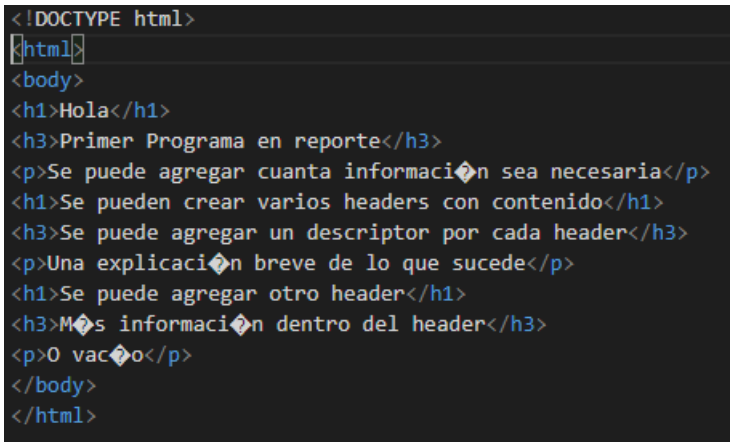


Figure 2: scrapper_report.html

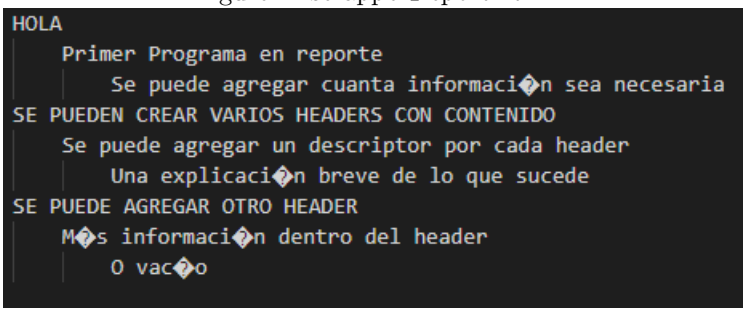


Figure 3: scrapper_report.txt

5 Manual de instalación

Para la intalanción del programa es necesario la ejecución de prerequisitos ya que con este código se instalarán las dependencias necesarias.

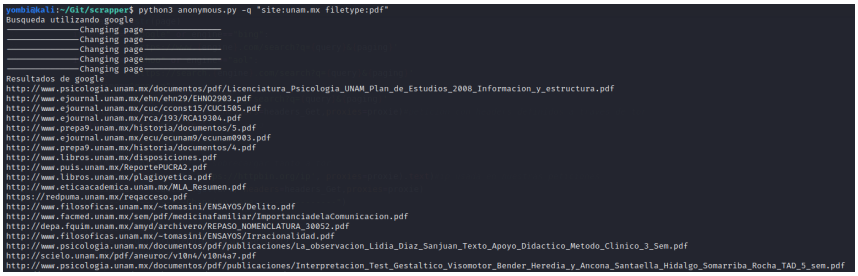
```
yombi@kali:~/Git/scrapper$ cat requisitos.py
#!/usr/bin/python3

import subprocess

"""
Función para instalar los requisitos del programa

"""

def install():
    subprocess.call('sudo apt update', shell=True)
    subprocess.call('sudo apt install python3-pip -y', shell=True)
    subprocess.call('pip3 install requests', shell=True)
    subprocess.call('sudo apt install tor -y', shell=True)
    subprocess.call('pip3 install fake_useragent', shell=True)
    subprocess.call('sudo apt install build-essential libssl-dev libffi-dev python-dev -y', shell=True)
    subprocess.call('pip3 install stem', shell=True)
    subprocess.call('pip3 install pysocks', shell=True)
    subprocess.call('pip3 install bs4', shell=True)

install()
```

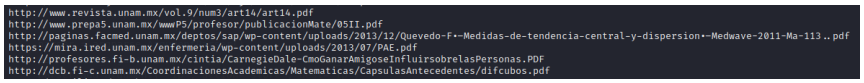
Para la confuguración de tor se utiliza el programa tor_config.sh. Este creará el hash de la contraseña a usar para autenticarse en tor y hará que el archivo de configuración de tor esté listo para interactuar con nuestro programa. Este se debe ejecutar con sudo pues se requieren permisos elevados para modificar el archvio /etc/tor/torrc

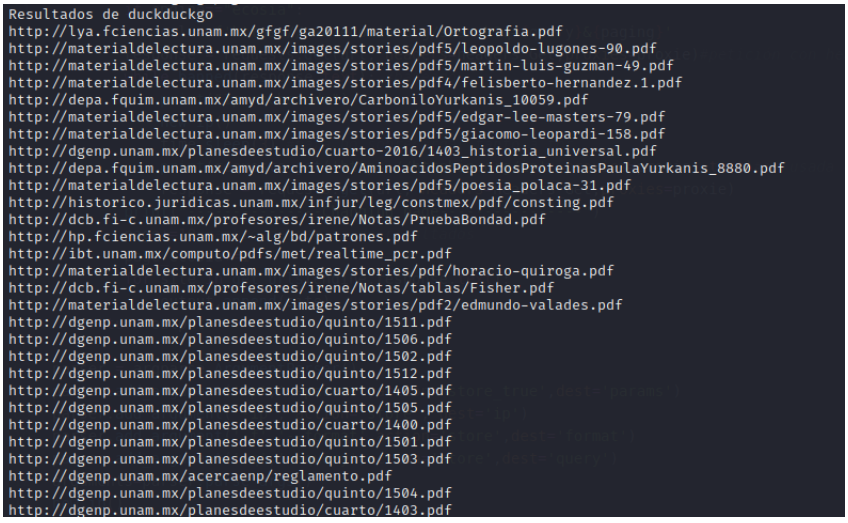
```
yombi@kali:~/Git/scrapper$ cat tor_config.sh
#!/bin/bash
echo "Contraseña para tor"
cat /etc/tor/torrc >> /etc/tor/torrc.back
read password
hash=$(tor --hash-password "$password")
hash=$(echo $hash|awk 'NF>1{print $NF}')
while read line
do
  if [[ $line == "#ControlPort 9051" ]]; then
    line="ControlPort 9051"
  elif [[ $line == *"#HashedControlPassword"* ]]; then
    line="HashedControlPassword "$hash
  fi
  echo $line >> new
done < /etc/tor/torrc
cat new > /etc/tor/torrc
rm new
echo "Reinice el servicio tor para que los cambios surtan efecto"
```

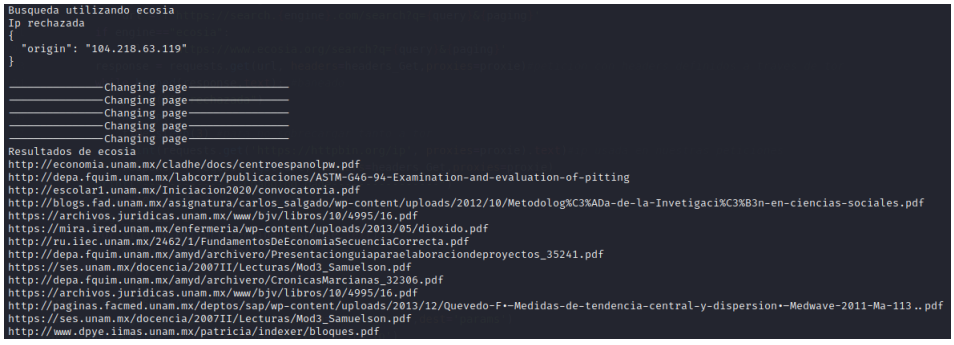
6 Manual de usuario

Una vez configurado todo, solo debemos usar la sentencia python3 scraper.py -q query o alguna otra bandera que deseemos y el programa empezara el *scraping*









Las banderas disponibles son las siguientes:

- –params o -p para incluir los parámetros de la URL en los resultados
- –formato" o -f formato se especifica que los resultados deben mostrarse en el formato especificado:
  - xml
  - html
  - txt
- -ip extrae todos los nombres de dominio asociados a la direccion que se mande con esta bandera
- -q o –query indica la busqueda a realizar