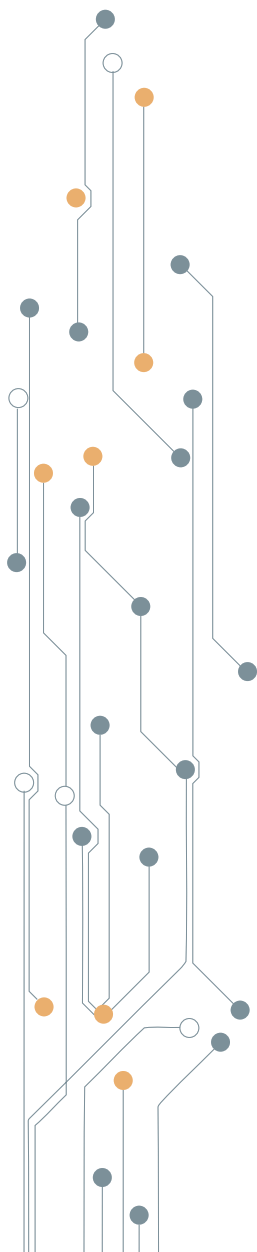




Recolección de información

Índice



1 Introducción	3
2 Extracción de información de servidores DNS	4
3 Extracción de información de un dominio mediante WHOIS	6
4 Geolocalización de IPS	8
5 Integración con Apis Web	11
5.1 Google	12
5.2 Shodan	18
5.3 Twitter	26
5.4 Pastebin	45
6 Recolección de Data Leaks	47
7 Análisis de Metadatos	54

1. Introducción

Una de las fases más importantes de un Pentesting o auditoría de seguridad informática es la fase de recolección de información (Information Gathering), donde se recopila gran cantidad de información de un sistema objetivo y la cual nos será de gran utilidad en fases posteriores.

Posiblemente esta sea una de las fases que más tiempo demande en la realización de un Pentest, por lo que es recomendable automatizar lo máximo posible este proceso. En los próximos apartados se detallará diversas herramientas y librerías que permiten automatizar este proceso mediante scripts en Python y que facilitan mucho esta labor.



2. Extracción de información de servidores DNS

Es posible extraer información valiosa de una compañía realizando consultas a los servidores DNS. En los sistemas Linux existen varios comandos nativos que permiten realizar consultas a un servidor DNS:

- Nslookup
- Dig

Pero si usamos estos comandos desde un script de Python, habría que tratar el resultado de estas consultas ya que se devuelve en texto plano. Por este motivo se va a utilizar la librería DNSPython que abstrae de toda esa complejidad y facilita el tratamiento de los resultados.

Esta librería se podrá descargar de su página oficial: <http://www.dnspython.org/>

Para instalarla habrá que descargarse la última versión estable. Una vez descomprimido en la máquina, se ejecutará:

```
# Python setup.py install
```

En el siguiente ejemplo se va a realizar consultas al servidor DNS preguntando por los siguientes registros DNS:

- A (IPv4)
- AAAA (IPv6)
- NS (NameServers)
- MX (MailServers)

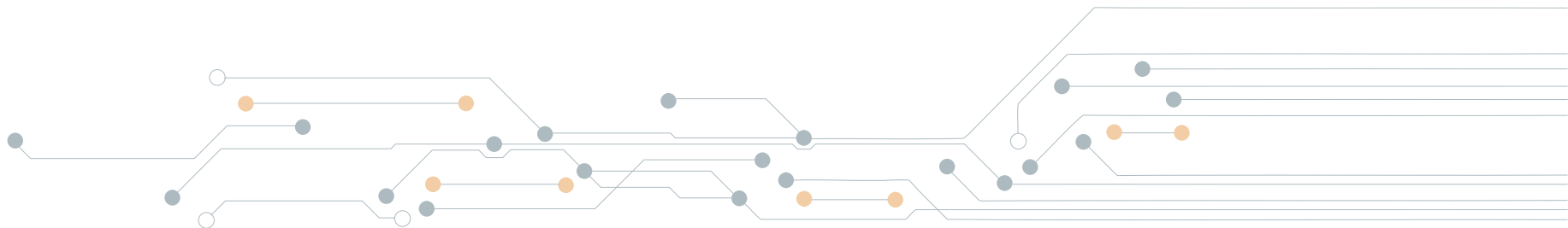
```
import dns
import dns.resolver

registroA = dns.resolver.query('wikipedia.org','A')
registroAAAA = dns.resolver.query('wikipedia.org','AAAA')
registroNS = dns.resolver.query('wikipedia.org','NS')
registroMX = dns.resolver.query('wikipedia.org','MX')
```

Y se obtendrá como respuesta la siguiente información (mostrada solo la del NS NameServer):

```
print(registroNS.response.to_text())

id 24197
opcode QUERY
rcode NOERROR
flags QR RD RA
;QUESTION
wikipedia.org. IN NS
;ANSWER
wikipedia.org. 86336 IN NS ns1.wikimedia.org.
wikipedia.org. 86336 IN NS ns2.wikimedia.org.
wikipedia.org. 86336 IN NS ns0.wikimedia.org.
;AUTHORITY
;ADDITIONAL
ns2.wikimedia.org. 2400 IN A 91.198.174.239
ns0.wikimedia.org. 342 IN A 208.80.154.238
ns1.wikimedia.org. 1777 IN A 208.80.153.231
```



3. Extracción de información mediante un dominio WHOIS

Whois es un protocolo que permite realizar consultas relativas a un dominio determinado con el objetivo de obtener información detallada del mismo como puede ser el propietario, fecha de creación, caducidad, teléfono de contacto, dirección, código postal, etc.

Para realizar estas consultas se utilizará la librería PythonWhois la cual abstrae la complejidad de realizar la búsqueda y el parseo de los resultados. De esta forma se tendrá como resultado una estructura de datos fácil de utilizar.

Esta librería se puede descargar del repositorio:

<https://pypi.python.org/pypi/pythonwhois/>

Para instalarla habrá que descomprimirla y ejecutar el siguiente comando:

```
# Python setup.py install
```

El primer paso será importar la librería y usarla para comprobar el servidor raíz para un dominio determinado:

```
import pythonwhois
pythonwhois.net.get_root_server('wikipedia.org')
u'whois.pir.org'
```

En el siguiente ejemplo se va a utilizar esta librería para obtener información sobre el dominio "wikipedia.org".

Además, con el método `get_whois()` se obtendrá la información relativa a un dominio determinado. Al ejecutar la consulta, sobre el resultado se mirará que atributos consultar:

```
result = pythonwhois.get_whois('wikipedia.org')
result.keys()

['status', 'updated_date', 'contacts', 'nameservers', 'expiration_date', 'raw', 'registrar', 'creation_date', 'id']
```

Para el ejemplo se consultarán los datos de contacto que se informaron al registrar el dominio:

```
result.get('contacts')
```

```
{'admin': {'city': u'San Francisco', 'fax': u'+1.4158820495', 'handle': u'mmr-116560', 'name': u'Domain Admin', 'state': u'CA', 'phone':  
u'+1.4158396885', 'street': u'149 New Montgomery Street\nThird Floor', 'country': u'US', 'postalcode': u'94105', 'organization':  
u'Wikimedia Foundation, Inc.', 'email': u'dns-admin@wikimedia.org'}, 'tech': {'city': u'San Francisco', 'fax': u'+1.4158820495', 'handle':  
u'mmr-116560', 'name': u'Domain Admin', 'state': u'CA', 'phone': u'+1.4158396885', 'street': u'149 New Montgomery Street\nThird  
Floor', 'country': u'US', 'postalcode': u'94105', 'organization': u'Wikimedia Foundation, Inc.', 'email': u'dns-admin@wikimedia.org'},  
'registrant': {'city': u'San Francisco', 'fax': u'+1.4158820495', 'handle': u'mmr-116560', 'name': u'Domain Admin', 'state': u'CA', 'phone':  
u'+1.4158396885', 'street': u'149 New Montgomery Street\nThird Floor', 'country': u'US', 'postalcode': u'94105', 'organization':  
u'Wikimedia Foundation, Inc.', 'email': u'dns-admin@wikimedia.org'}, 'billing': None}
```

```
result = pythonwhois.get_whois('wikipedia.org')  
result.keys()
```

```
['status', 'updated_date', 'contacts', 'nameservers', 'expiration_date', 'raw', 'registrar', 'creation_date', 'id']
```



4. Geolocalización de IPS

Existen en internet varias bases de datos públicas que relacionan millones de IPs con localizaciones geográficas de donde se encuentran.

Con la información aportada por estas bases de datos se podrá obtener nueva información sobre un sistema objetivo o validar la información que ya se tiene del mismo y que se ha obtenido con otras herramientas.

Existen varios sitios web y librerías que ayudan con esta tarea, pero en este curso se utilizará la librería PyGeolIP. Esta librería hace uso de las bases de datos de GeoLite, las cuales se actualizan frecuentemente y están especializadas por tipo de información.

Habrá que descargar la última versión de la librería PyGeolIP de los repositorios de Python:

<https://pypi.python.org/pypi/pygeoip>

Se instalará la librería como en ocasiones anteriores:

```
# Python setup.py install
```

Una vez instalada la librería se tendrá que descargar las bases de datos GeoLite. El proveedor ofrece unas bases de datos gratuitas desde su página web:

<https://dev.maxmind.com/geoip/legacy/geolite/>

Los tipos de bases de datos que se pueden descargar son los siguientes:

- GeoLite Country
- GeoLite Country IPv6
- GeoLite City
- GeoLite City IPv6
- GeoLite ASN
- GeoLite ASN IPv6

Además, el proveedor ofrece una nueva versión de estas bases de datos mucho más precisas, pero son de pago:

<https://www.maxmind.com/en/geoip2-databases>

Para el siguiente ejemplo, habrá que descargar las bases de datos GeoLite Country ('GeoIP.dat'), GeoLite City ('GeoLiteCity.dat') y GeoLite ASN ('GeoIPASN.dat'). Lo primero se importará la librería y se comprobará qué información nos proporciona la base de datos GeoLite Country ('GeoIP.dat'):

```
>>> import pygeoip
>>> gip = pygeoip.GeoIP('GeoIP.dat')
>>> gip.country_code_by_name('wikipedia.org')
'NL'
>>> gip.country_name_by_name('wikipedia.org')
'Netherlands'
>>> gip.country_code_by_addr('91.198.174.192')
'NL'
>>> gip.country_name_by_addr('91.198.174.192')
'Netherlands'
```

Como se puede observar, se puede obtener el país tanto de un nombre de dominio como de una dirección IP. Además, ofrece la posibilidad de obtener el nombre completo o las siglas del país.

En el siguiente ejemplo se comprobará que información obtenemos con la base de datos GeoLite City ('GeoLiteCity.dat').

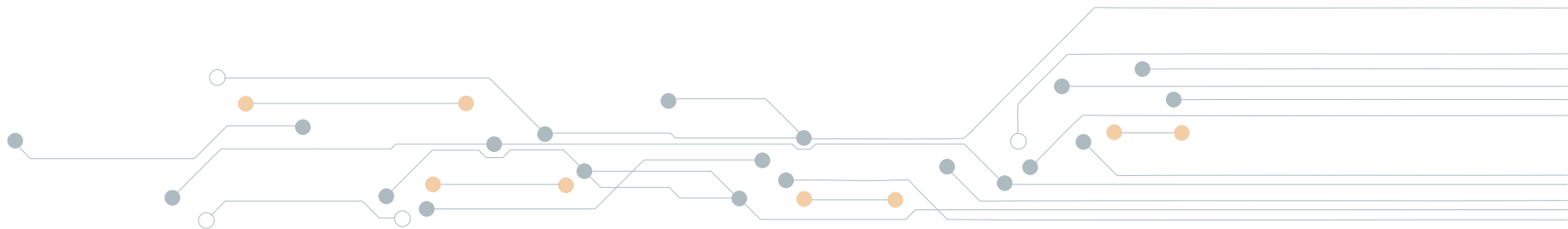
Se utilizará la librería de Python "pprint" para mostrar de forma más clara los resultados:

```
>>> import pprint
>>> gip = pygeoip.GeoIP('GeoLiteCity.dat')
>>> pprint.pprint(gip.record_by_name('wikipedia.org'))
{'area_code': 0,
 'city': None,
 'continent': 'EU',
 'country_code': 'NL',
 'country_code3': 'NLD',
 'country_name': 'Netherlands',
 'dma_code': 0,
 'latitude': 52.38239999999999,
 'longitude': 4.899499999999989,
 'metro_code': None,
 'postal_code': None,
 'region_code': None,
 'time_zone': 'Europe/Amsterdam'}
>>> gip.region_by_name('wikipedia.org')
{'region_code': None, 'country_code': 'NL'}
>>> gip.time_zone_by_name('wikipedia.org')
'Europe/Amsterdam'
>>> pprint.pprint(gip.record_by_name('91.198.174.192'))
{'area_code': 0,
 'city': None,
 'continent': 'EU',
 'country_code': 'NL',
 'country_code3': 'NLD',
 'country_name': 'Netherlands',
 'dma_code': 0,
 'latitude': 52.38239999999999,
 'longitude': 4.899499999999989,
 'metro_code': None,
 'postal_code': None,
 'region_code': None,
 'time_zone': 'Europe/Amsterdam'}
>>> gip.region_by_addr('91.198.174.192')
{'region_code': None, 'country_code': 'NL'}
>>> gip.time_zone_by_addr('91.198.174.192')
'Europe/Amsterdam'
```

Como se puede observar, con esta base de datos se puede obtener información relativa a la ciudad donde está alojado el servidor correspondiente a esa IP, además de obtener unas coordenadas geográficas las cuales podríamos visualizar en un mapa.

Por último, se utilizará la última base de datos GeoLite ASN ('GeoIPASNum.dat') para obtener información relativa al número ASN (Autonomous System Number) el cual identifica una colección de IPs bajo el control de una entidad:

```
>>> gip = pygeoip.GeoIP('GeoIPASNum.dat')
>>> gip.org_by_name('wikipedia.org')
u'AS43821 WIKIMEDIA-EU'
>>> gip.org_by_name('91.198.174.192')
u'AS43821 WIKIMEDIA-EU'
```



5. Interacción con Apis Web

Una de las fuentes de información más importantes de donde se puede obtener gran cantidad de datos muy útiles son las páginas web de internet. En estas páginas puede haber información relevante asociada a la organización que se está analizando y que puede servir de mucha utilidad en fases posteriores de un Pentest.

La principal fuente donde se puede obtener una gran cantidad de información son los motores de búsqueda, los cuales suelen indexar muchísima información sobre las organizaciones. Algunos de estos buscadores son:

- Google
- Bing
- Yahoo!
- Shodan
- MrLooker

Pero también se puede obtener información muy importante de otros sitios como pueden ser las redes sociales o páginas de publicación de DataLeaks y Pastes similares a PasteBin.

Es muy importante que, con la información obtenida por uno de los métodos de los puntos anteriores, realizar una búsqueda en los distintos buscadores web para poder obtener más información sobre los mismos.

Muchas de estas páginas ofrecen a los usuarios un API con la cual interaccionar con ella por medio de programas o scripts. En estos casos recuperar la información se convierte en una tarea mucho más tediosa, porque habrá que realizar peticiones HTTP normales para traernos la página web completa, y posteriormente tratar esta información para obtener datos concretos que nos sirvan de utilidad. En este punto es muy recomendable el estudio y uso de las expresiones regulares las cuales facilitan mucho esta labor.

En otros casos, los servicios de internet cuentan con APIs que permiten interaccionar con ellas obteniendo resultados estructurados con los que poder trabajar. En algunos casos, estas APIs se pueden invocar de forma pública, pero en otros es necesario el registro en la plataforma para obtener una "Developer Key" con la cual autenticar las peticiones.



5.1 | Google

Google es el motor de búsqueda más utilizado en todo el mundo. Está enfocado en búsquedas de propósito general siendo una de las fuentes más importantes de información sobre sitios web de Internet.

Este motor, aparte de indexar información relativa a las páginas web, también indexa muchísima información sobre organización. Donde en muchos casos, también se indexa información que no debería ser expuesta de forma pública por lo que supone una fuga de información que puede beneficiar a un posible atacante.

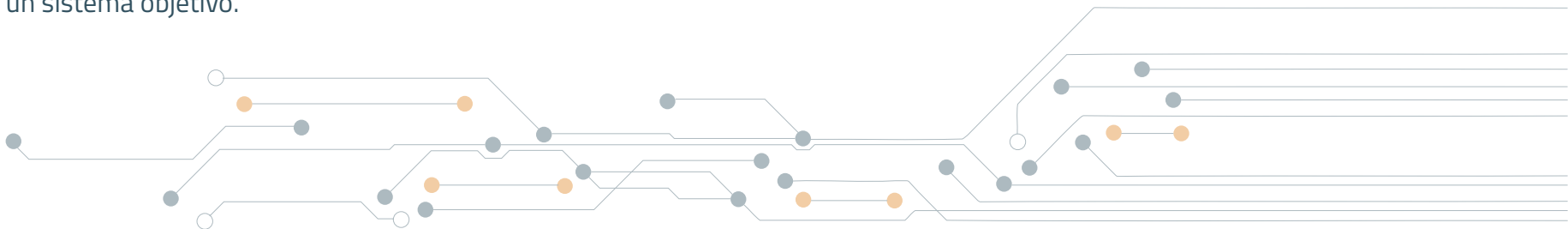
Estas fugas son provocadas muchas veces por malas configuraciones en los servidores o simplemente por algún fallo o despiste. Estas fugas pueden ofrecer información sobre el sistema objetivo: nombre de máquinas, aplicaciones y versiones concretas, nombres de usuario, nombres de empleado, cuentas de correo, identificadores internos, etc.

Para poder recuperar información concreta sobre un determinado sitio en internet, Google ofrece una gran cantidad de filtros los cuales ayudan a realizar búsquedas muy específicas. Estas búsquedas se convierten en una herramienta muy potente para la obtención de datos relativos a un sistema objetivo.

Lo primero que hay que conocer son los operadores básicos en la definición de las búsquedas.

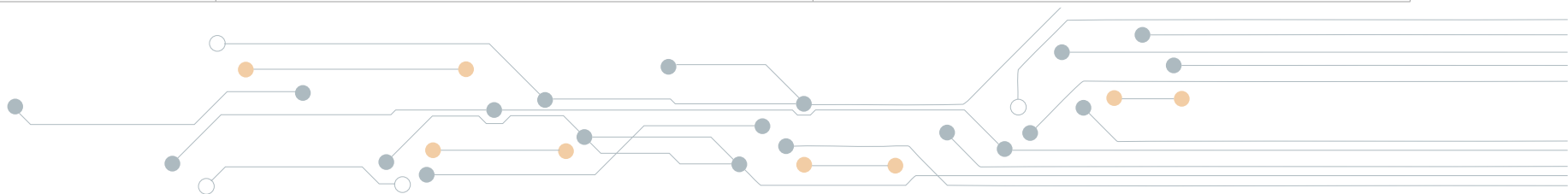
Estos operadores permiten realizar búsquedas mucho mas específicas sin utilizar filtros avanzados. Estos operadores son:

- Comillas (" "): permite buscar las páginas donde aparece el término exacto entrecomillado.
- Operadores lógicos (AND, OR, NOT): sirve para realizar búsquedas con varias palabras o frases
- Operador + y - : permite incluir y excluir palabras de la búsqueda. Ej.: buscar "jaguar -coches", busca la palabra "jaguar", pero omite las webs con la palabra "coches".
- Operador *: representa cualquier palabra en la búsqueda (solo 1 palabra)
- Operador . (punto): representa una o varias palabras en la búsqueda.



Aparte de los operadores básicos, existen los filtros de búsquedas avanzados que son los que realmente permiten realizar búsquedas muy específicas. Los filtros más importantes son:

OPERADOR	DESCRIPCIÓN	EJEMPLO
Site	Sirve para restringir la búsqueda a un sitio web concreto	site: www.wikipedia.org
Inurl	Especifica palabras que únicamente aparecen en la url del website	inurl: pentesting
Intitle	Busca palabras determinadas en el título de la página	intitle: pentesting
Intext	Sirve para localizar páginas webs que contengan en su texto el termino de búsqueda deseado	intext: pentesting
Ext / filetype	Sirve para especificar un formato concreto de los documentos de la búsqueda	pentesting filetype: pdf
Related	Sirve para localizar sitios web similares a la dirección url que has solicitado	related: www.wikipedia.org
Link	Sirve para localizar las páginas web que tienen enlace dirigidos a una página determinada	link: www.wikipedia.org
Cache	Sirve para visualizar una determinada página la última vez que el bot de rastreo de Google la indexó.	cache: www.wikipedia.org



Todas estas cadenas de búsqueda que combinan varios operadores y filtros para obtener información muy detallada sobre un tipo de sistema objetivo se las ha denominado Google Dorks. Algunos ejemplos interesantes de Google Dorks:

- `site:pastebin.com intext:@gmail.com | @yahoo.com | @hotmail.com`
- `inurl:wp-content/uploads filetype:xls | filetype:xlsx password`
- `inurl:"ftp" intext:"user" | "username" | "userID" | "user ID" | "logon" | "login" intext:"password" | "passcode" filetype:xls | filetype:xlsx`
- `filetype:log intext:password | pass | pw`

Existen varias páginas que sirven como repositorio para estas cadenas de búsqueda y que son administradas por la comunidad añadiendo nuevas cadenas diariamente. Una de las más importantes es la denominada Google Hacking Database (GHDB)

<https://www.exploit-db.com/google-hacking-database/>

Este repositorio está estructurado y ordenado por diversas categorías que facilita la búsqueda de los Dorks útiles para un sistema objetivo. Algunas de las categorías de esta base de datos que contienen más Dorks son:

- Footholds
- Directorios sensibles
- Ficheros vulnerables
- Servidores vulnerables
- Mensajes de error
- Ficheros con nombres de usuarios
- Ficheros con passwords
- Páginas con portal de login
- Etc.

Para poder realizar todas estas búsquedas mediante scripts de Python, Google pone a disposición de la comunidad de una librería con la cual interactuar con el motor de búsquedas. Esta librería se encuentra alojada en el proyecto Github:

<https://github.com/google/google-api-python-client>



Esta librería tiene bastantes dependencias externas por lo que se recomienda su instalación usando la herramienta PIP:

```
# pip install --upgrade google-api-python-client
```

Una vez instalada la librería, el siguiente paso para poder realizar un programa que interactúe con Google es el registrar una aplicación en la consola de desarrolladores.

Para ello es necesario disponer de una cuenta de Google válida y autenticarse en cualquiera de los servicios de Google. Desde la consola de administración para los desarrolladores se podrá crear un nuevo proyecto para realizar nuestras pruebas desde la librería. Para ello hay que acceder a la a la página de desarrolladores: <https://console.developers.google.com>

Habrá que seleccionar la opción de "Create Project" y le asignaremos un nombre del proyecto y un identificador único (si no especificamos ninguno Google lo crea por nosotros):



Nuevo proyecto

Nombre del proyecto ⓘ

My Project

El ID del proyecto será the-condition-146616 ⓘ Editar

[Mostrar las opciones avanzadas...](#)

CANCELAR CREAR

Una vez se ha creado el proyecto, es necesario activar los servicios de Google que se desean utilizar. Google dispone de muchos servicios diferentes con los que poder interactuar (Drive, Calendar, etc).

Para realizar búsquedas en el motor de búsquedas de Google, se tendrá que habilitar la API denominada "Compute Engine API". Para la versión gratuita de la cuenta esta API tendrá una serie de restricciones, pero para la mayoría de los casos es más que suficiente. La principal limitación es que solo permite realizar 1000 búsquedas diarias.

Para habilitarla, desde la página principal de la consola de desarrolladores (<https://console.developers.google.com>), habrá que seleccionar el menú "Biblioteca" y en la nueva sección habrá que seleccionar el API "Custom Search API".

Una vez se accede a la nueva pantalla seleccionar la opción de Habilitar. Si no se ha utilizado previamente los servicios de Google Cloud Platform, se solicitará el registro de la cuenta para facturación. Aunque en el proceso de registro se solicita una tarjeta de crédito, este registro es gratuito y no cobrarán nada salvo que se habilite más servicios de Google que sean de pago.

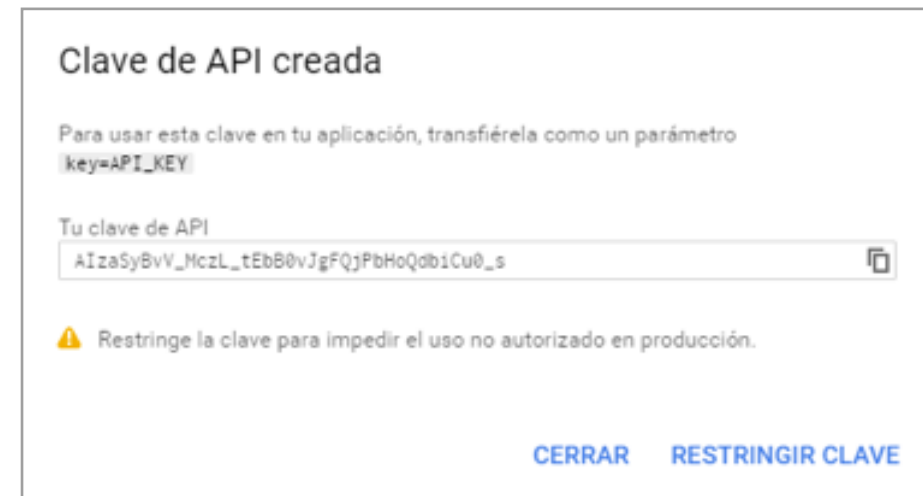
Una vez habilitada la librería "Compute Engine API" se deberán de crear las claves de autenticación para poder consumir el API desde Python. Desde el menú "Credenciales" pulsando el botón de "Crear credenciales" se abre la siguiente ventana:



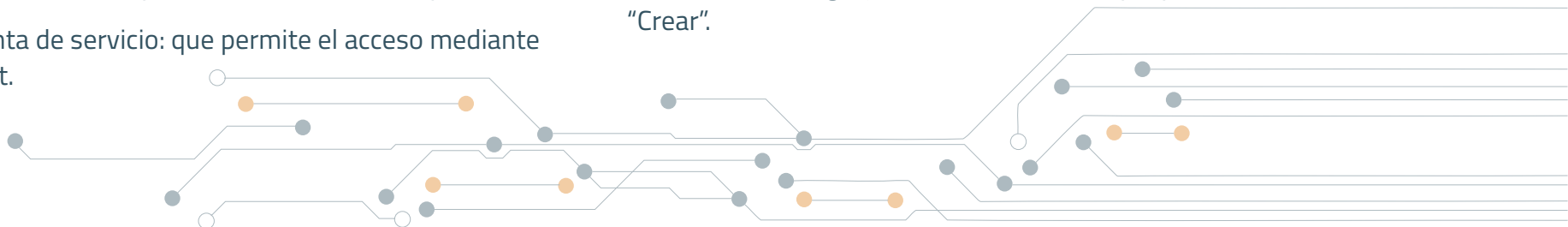
Como se muestra en la imagen anterior, Google ofrece 2 formas diferentes para gestionar la autenticación de las aplicaciones desarrolladas:

- **API Key:** se realiza una autenticación básica comprobando que el token utilizado es válido.
- **Credenciales OAuth:** con estas credenciales se puede especificar en detalle los permisos de acceso y autorización, siendo de gran utilidad para desarrollos que involucren a terceras personas.
- **Clave de cuenta de servicio:** que permite el acceso mediante cuentas robot.

Para realizar búsquedas mediante "Compute Engine API" no es necesario disponer de unas credenciales OAuth por lo que se creará un token básico "API Key":



El siguiente paso es crear y configurar un motor de búsqueda. Para ello hay que acceder a la página <https://cse.google.com/cse/all> y pulsar sobre el botón "Add". En la nueva ventana se tendrá que configurar el motor de búsqueda, en el campo "Sitios donde buscar" habrá que introducir una URL cualquiera ya que se va a eliminar en el siguiente paso (por ejemplo www.misitio.com), también habrá que seleccionar el lenguaje. Finalmente habrá que pulsar sobre el botón "Crear".



El siguiente paso es volver a la página <https://cse.google.com/cse/all> y seleccionar el motor de búsqueda que se acaba de crear. En la nueva ventana habrá que seleccionar en el desplegable "Buscar en toda la web, pero enfatizar los sitios incluidos". El siguiente paso es eliminar el sitio que se ha introducido antes "www.misitio.com" de la lista de abajo. Por último, pulsar sobre el botón "ID de motor de búsqueda" y copiar el código que será necesario utilizarlo desde el script de Python.

En este punto ya se han realizado todas las tareas previas al uso de la librería de Google.

En el siguiente código se muestra un ejemplo simple donde se realiza una búsqueda contra el API del buscador:

```
from googleapiclient.discovery import build

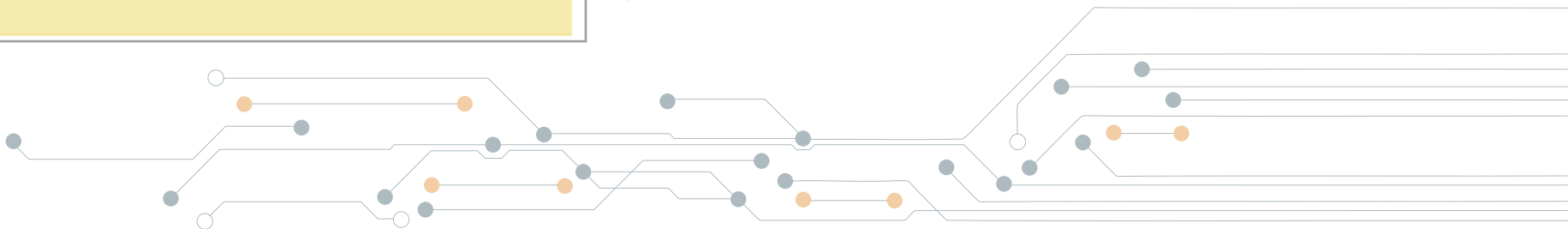
my_api_key = "XXXXXXXXXXXXXXXXXX"
my_cse_id = "XXXXXXXXXXXXXXXXXX"

querie = 'stackoverflow site:en.wikipedia.org'

service = build("customsearch", "v1", developerKey=my_api_key)
res = service.cse().list(q= querie , cx=my_cse_id, num=10).execute()
for result in res['items']:
    print(result)
```

De esta forma sencilla se pueden realizar búsquedas en Google de una manera rápida, pudiendo utilizar el resultado de estas búsquedas en otras tareas automáticas en los scripts.

La Querie utilizada en el ejemplo es muy simple, pero se puede utilizar los Dorks vistos anteriormente para realizar búsquedas muy potentes.



5.2 | Shodan

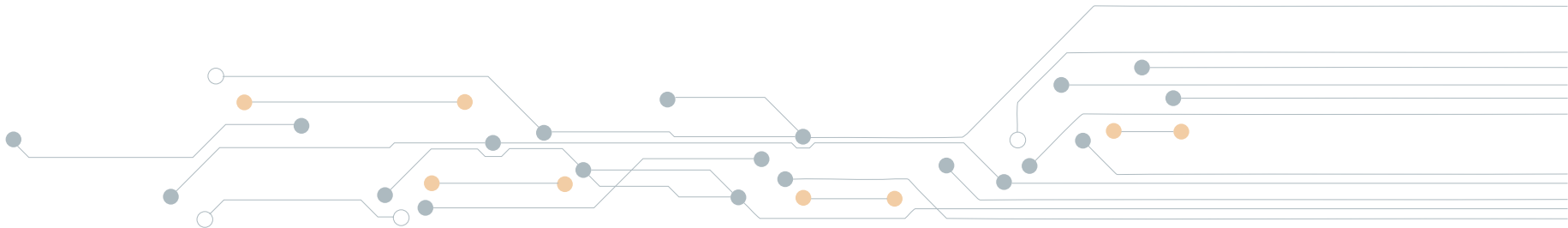
Shodan (<https://www.shodan.io/>) es un motor de búsquedas que difiere mucho de los buscadores habituales. A diferencia de buscadores como Google o Bing donde indexan en sus bases de datos información relativa a las propias páginas web que va rastreando y sus hipervínculos, Shodan cambia este modelo y su objetivo es rastrear e indexar toda la información disponible sobre los servicios que están ejecutándose en cada uno de los dispositivos y servidores detrás de esas IPs. Pudiendo ser un servidor de aplicaciones, un sistema SCADA, una cámara IP, una impresora, etc.

Para obtener toda esta información, Shodan escanea todas las IPs existentes realizando peticiones a estos dispositivos. Recolecta los datos de cabeceras de la respuesta de la petición y los banners que devuelven cada uno de los servicios que están ejecutándose y los almacena en sus BBDD.

Con toda esa información almacenada, Shodan permite realizar búsquedas muy potentes y muy diversas, ya que se te permite especificar un producto concreto, una versión del software, un sistema operativo, una localización geográfica, una IP concreta, etc, devolviendo como resultado todos los servidores que se ajustan a la búsqueda deseada.

Shodan dispone de varias modalidades:

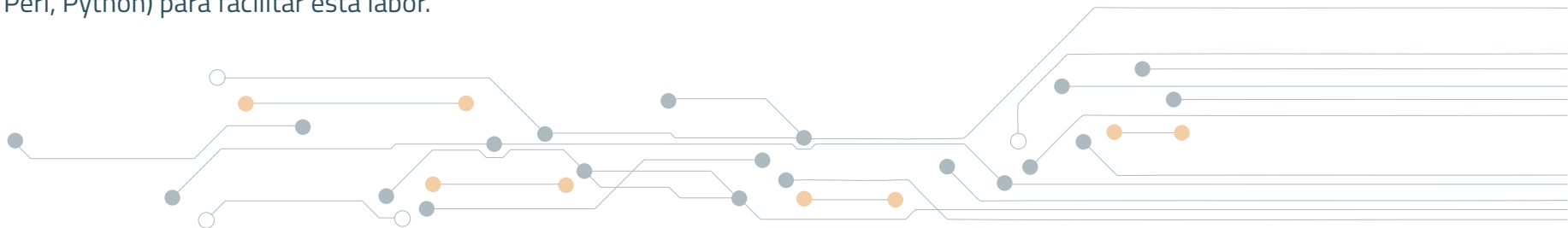
- **Sin registro:** esta versión está bastante limitada. No permite la utilización de filtros en las búsquedas y el resultado de las búsquedas está limitado a 10 resultados.
- **Registro gratuito:** Shodan permite registrarse gratuitamente a través de cuentas de Twitter, Facebook, Google, Windows Live o creando cuenta en Shodan directamente. Esta versión permite utilizar filtros en las búsquedas, y los resultados, aunque siguen limitados se amplían hasta 50.
- **Registro Premium:** Shodan tiene una versión de pago la cual desbloquea todas las funcionalidades del buscador. En esta versión se dispone de más filtros de búsqueda, el número de resultados no está limitado, y permite la integración de los resultados con sus herramientas de geolocalización y de mapas.



Los filtros disponibles en la versión gratuita son:

FILTRO	DESCRIPCIÓN	EJEMPLO
Country	Permite filtrar la búsqueda a un país en concreto	Apache country:ES
City	Permite filtrar la búsqueda a una ciudad concreta	Apache country:Madrid
Port	Permite especificar un puerto concreto	Port:21
Net	Permite realizar búsquedas por una IP concreta o por un rango de IP's	Net:91.198.174.192/24
Hostname	Permite buscar por texto en el hostname del servidor	Hostname:Camera
Os	Permite buscar por un determinado sistema operativo	microsoft-iis os:"windows 2003"
Geo	Permite centrar la búsqueda a una localización determinada por los puntos de longitud y latitud	

Además de las búsquedas propias de la propia web, Shodan cuenta con un API con el cual interaccionar con todas sus funcionalidades de forma programática. También dispone de diversas librerías en varios lenguajes (Ruby, Perl, Python) para facilitar esta labor.



Para poder hacer uso de este API es necesario tener una cuenta registrada (gratuita o premium). Para ello será necesario acceder a la sección de registro del sitio web de Shodan <https://account.shodan.io/register>

Una vez se dispone de una cuenta válida hace falta obtener el API Key de la cuenta para poderla utilizar desde las librerías de Python. Esta Key se encuentra en el apartado 'My Account': <https://account.shodan.io/>

Para la instalación de la librería de Shodan para Python se puede realizar de 3 formas diferentes:

- Utilizando easy_install:

```
# easy_install shodan
```

- Utilizando PIP:

```
# pip install shodan
```

- Descargando la librería del repositorio Github (<https://github.com/achillean/shodan-python>) y ejecutando el siguiente comando:

```
# python setup.py install
```

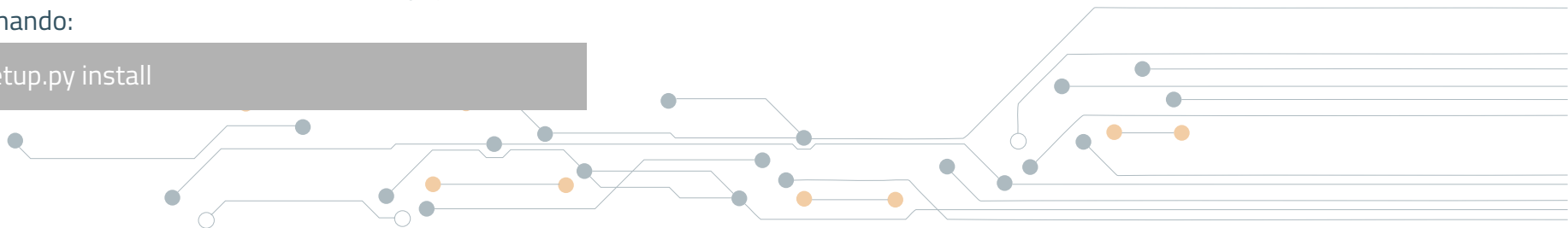
Para empezar a utilizar esta librería en los scripts de Python lo primero es crear una instancia de la clase shodan.Shodan pasándole como parámetro el API Key de la cuenta de usuario:

```
import shodan
SHODAN_API_KEY = "XXXXXX"
api = shodan.Shodan(SHODAN_API_KEY)
```

Una vez instanciada la clase Shodan, se puede obtener todos los datos de configuración relacionados con la cuenta de usuario de una forma muy simple, se usará el método info() de la instancia de shodan que se acaba de crear:

```
info = api.info()
info

{'https': False, 'unlocked_left': 0, 'query_credits': 0, 'plan': 'oss',
 'scan_credits': 0, 'telnet': False, 'unlocked': False}
```



También se pueden realizar búsquedas de forma muy sencilla utilizando el método *search()* de esa instancia:

```
>>> r = api.search('apache')
>>> r.keys()
dict_keys(['matches', 'total'])
>>> print('Numero de resultados de la búsqueda:', r['total'])
Numero de resultados de la búsqueda: 14777024
>>> type(r['matches'])
<class 'list'>
>>> len(r['matches'])
100
```



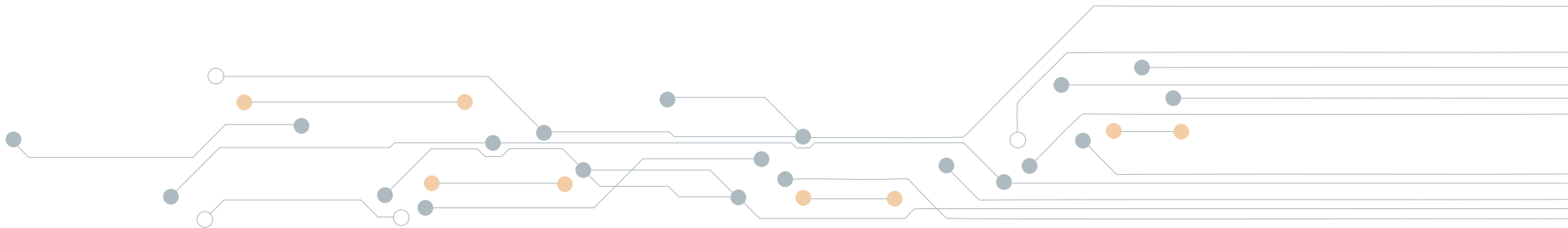
Los resultados de la búsqueda se encuentran en el registro 'matches' del diccionario obtenido como resultado de la búsqueda. Este registro es del tipo lista, se puede iterar sobre ella para realizar acciones con los datos. Con la cuenta de usuario gratuita este número de resultados está limitado a 100. En el siguiente ejemplo se mostrará los datos obtenidos para uno de los resultados:

```
>>> result = r['matches'][0]
>>> result.keys()
dict_keys(['domains', 'http', 'transport', 'asn', 'os', 'timestamp', 'isp', 'hash', '_shodan', 'title', 'ip_str', 'org', 'ip', 'hostnames', 'location', 'port', 'data', 'deprecated'])
>>> result.values()
dict_values(['t-ipconnect.de', {'components': {}, 'html': '<HEAD><TITLE>401 Authorization Required</TITLE></HEAD>\n<BODY><H1>401 Authorization Required</H1>\nBrowser not authentication-capable or \nauthentication failed.\n</BODY>\n', 'title': '401 Authorization Required', 'host': '91.44.235.226', 'html_hash': 1220172246, 'redirects': [], 'location': '/', 'robots': None, 'sitemap': None, 'server': 'Apache'}, 'tcp', 'AS3320', None, '2016-10-31T13:18:34.634129', 'Deutsche Telekom AG', -1382997051, {'id': 'fa2b46ad-0b13-48d6-8925-4328faeeadab', 'module': 'http-simple-new', 'options': {}, 'crawler': '97b9d37f0484f45ce645307121c5c1ce0b3db578'}, '401 Authorization Required', '91.44.235.226', 'Deutsche Telekom AG', 1529670626, ['p5B2CEBE2.dip0.t-ipconnect.de'], {'country_name': 'Germany', 'city': 'Stuttgart', 'country_code3': 'DEU', 'dma_code': None, 'area_code': None, 'country_code': 'DE', 'region_code': '01', 'longitude': 9.183300000000003, 'latitude': 48.766699999999986, 'postal_code': '70173'}, 7547, 'HTTP/1.1 401 Unauthorized\r\nDate: Mon, 31 Oct 2016 13:18:26 GMT;\r\nServer: Apache\r\nPragma: no-cache\r\nCache-Control: max-age=0, must-revalidate\r\nConnection: close\r\nContent-type: text/html\r\nWWW-Authenticate: Digest realm="DSLForum CPE Management", algorithm=MD5, qop=auth, stale=FALSE, nonce="68a7b97c0e65f377fe0a27585226b703", opaque="5ccc069c403ebaf9f0171e9517f40e41"\r\n\r\n', {'html': {'eol': '2016-07-01', 'new': 'http.html'}, 'opts.pem': {'eol': '2016-07-01', 'new': 'ssl.chain'}, 'title': {'eol': '2016-07-01', 'new': 'http.title'}, 'opts.sitemap': {'eol': '2016-07-01', 'new': 'http.sitemap'}, 'opts.robots': {'eol': '2016-07-01', 'new': 'http.robots'}}])
```



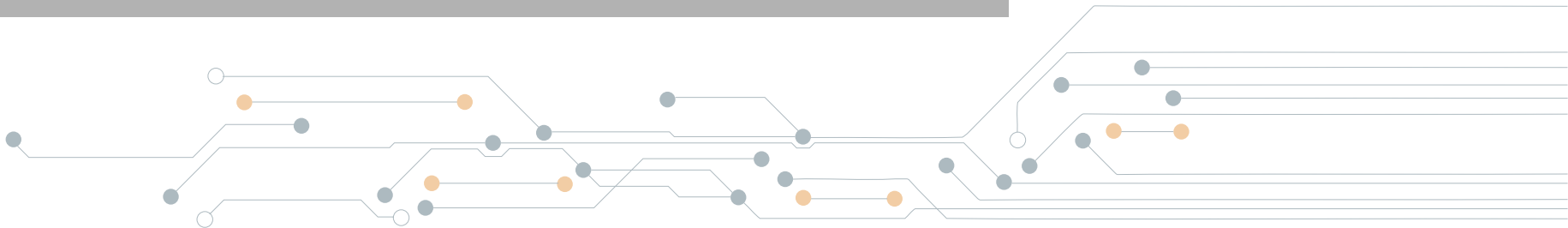
También se pueden realizar búsquedas de forma muy sencilla utilizando el método *search()* de esa instancia:

```
>>> r = api.search('apache')
>>> r.keys()
dict_keys(['matches', 'total'])
>>> print('Numero de resultados de la búsqueda:', r['total'])
Numero de resultados de la búsqueda: 14777024
>>> type(r['matches'])
<class 'list'>
>>> len(r['matches'])
100
```



Los resultados de la búsqueda se encuentran en el registro 'matches' del diccionario obtenido como resultado de la búsqueda. Este registro es del tipo lista, se puede iterar sobre ella para realizar acciones con los datos. Con la cuenta de usuario gratuita este número de resultados está limitado a 100. En el siguiente ejemplo se mostrará los datos obtenidos para uno de los resultados:

```
>>> result = r['matches'][0]
>>> result.keys()
dict_keys(['domains', 'http', 'transport', 'asn', 'os', 'timestamp', 'isp', 'hash', '_shodan', 'title', 'ip_str', 'org', 'ip', 'host-names', 'location', 'port', 'data', 'deprecated'])
>>> result.values()
dict_values([['t-ipconnect.de'], {'components': {}, 'html': '<HEAD><TITLE>401 Authorization Required</TITLE></HEAD>\n<BODY><H1>401 Authorization Required</H1>\nBrowser not authentication-capable or \nauthentication failed.\n</BODY>\n', 'title': '401 Authorization Required', 'host': '91.44.235.226', 'html_hash': '1220172246', 'redirects': [], 'location': '/', 'robots': None, 'sitemap': None, 'server': 'Apache', 'tcp': 'AS3320', None, '2016-10-31T13:18:34.634129', 'Deutsche Telekom AG', -1382997051, {'id': 'fa2b46ad-0b13-48d6-8925-4328faeeadab', 'module': 'http-simple-new', 'options': {}, 'crawler': '97b9d37f0484f45ce645307121c5c-1ce0b3db578'}, '401 Authorization Required', '91.44.235.226', 'Deutsche Telekom AG', 1529670626, ['p5B-2CEBE2.dip0.t-ipconnect.de'], {'country_name': 'Germany', 'city': 'Stuttgart', 'country_code3': 'DEU', 'dma_code': None, 'area_code': None, 'country_code': 'DE', 'region_code': '01', 'longitude': 9.183300000000003, 'latitude': 48.766699999999986, 'postal_code': '70173'}, 7547, 'HTTP/1.1 401 Unauthorized\r\nDate: Mon, 31 Oct 2016 13:18:26 GMT\r\nServer: Apache\r\nPragma: no-cache\r\nCache-Control: max-age=0, must-revalidate\r\nConnection: close\r\nContent-type: text/html\r\nWWW-Authenticate: Digest realm="DSLForum CPE Management", algorithm=MD5, qop=auth, stale=FALSE, nonce="68a7b97c0e65f377fe0a27585226b703", opaque="5ccc069c403ebaf9f0171e9517f40e41"\r\n\r\n', {'html': {'eol': '2016-07-01', 'new': 'http.html'}, 'opts.pem': {'eol': '2016-07-01', 'new': 'ssl.chain'}, 'title': {'eol': '2016-07-01', 'new': 'http.title'}, 'opts.sitemap': {'eol': '2016-07-01', 'new': 'http.sitemap'}, 'opts.robots': {'eol': '2016-07-01', 'new': 'http.robots'}}])
```



Obteniendo como resultado:

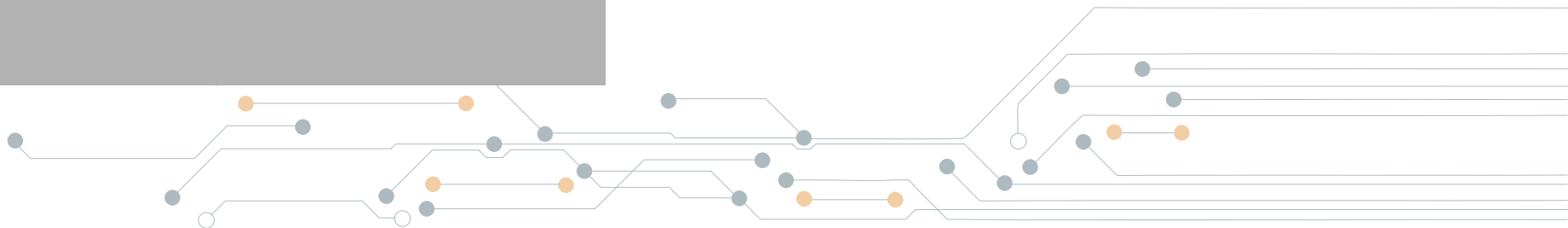
Top 10 Organizations
Deutsche Telekom AG: 1772345
Telmex: 1260197
Amazon.com: 794287
GoDaddy.com, LLC: 387910
Ecommerce Corporation: 318649

Top 10 Domains
t-ipconnect.de: 1731827
prod-infinitum.com.mx: 1183875
amazonaws.com: 1157831
secureserver.net: 390755
unifiedlayer.com: 146344

Top 10 Ports
80: 11544137
443: 6090993
7547: 1743083
8080: 662076
8443: 183799

Top 5 Countries
US: 7135462
DE: 2987033
MX: 1293119
CN: 1225594
JP: 1204702

Top 10 Autonomous Systems
as3320: 1774781
as8151: 952387
as26496: 350564
as32392: 318649
as16276: 249701



5.3 | Twitter

Twitter es una de las redes sociales más famosas de la actualidad, teniendo en torno a 310 millones de usuarios activos alrededor del mundo. En esta red social se genera una gran cantidad de información diariamente, donde la mayoría de ella se puede acceder de forma pública.

Esta red social cuenta con un API Rest que permite la interacción desde cualquier tipo de aplicación (móvil, web, scripts, etc) ya que se utiliza el protocolo HTTP y es compatible para cualquier lenguaje o plataforma. Desde este API se puede recoger toda la información de la página, y además una gran cantidad de información que la plataforma no muestra en el interfaz web.

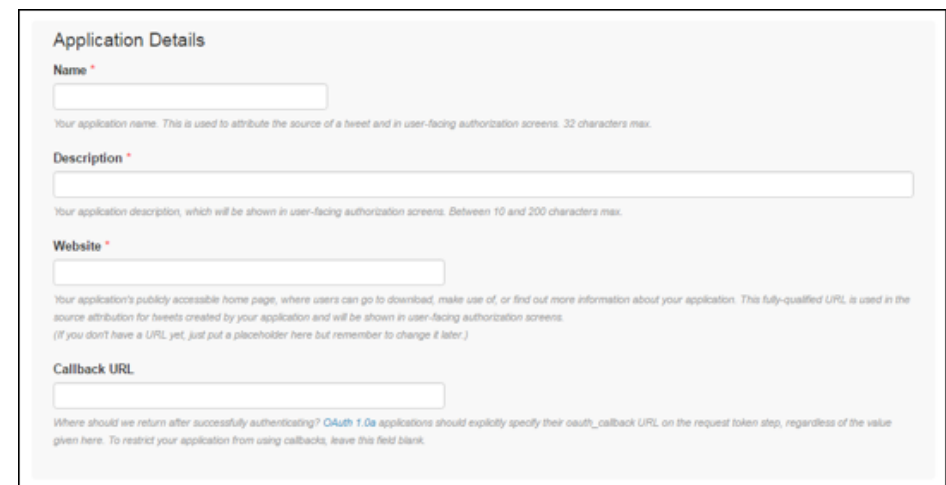
Toda la documentación del funcionamiento de la API Rest se encuentra disponible en la página para desarrolladores:

<https://dev.twitter.com/docs>

Desde la versión 1.1 del API es necesario contar con una cuenta de usuario válida para poder interaccionar con ella. Además, será necesario la creación de tokens de autenticación OAuth que se deberán usar en los scripts creados para poder realizar peticiones de forma autenticada.

El primer paso será el crear una cuenta de usuario en la red social y acceder a la página <https://apps.twitter.com/> con el usuario y

contraseña de la cuenta. En la nueva pantalla (en la parte superior derecha) habrá que pulsar sobre “Create new App”. Lo cual abrirá la siguiente pantalla:



The screenshot shows the 'Application Details' form on the Twitter developer portal. It contains four main sections, each with a text input field and a descriptive note:

- Name ***: A text input field. Below it, a note states: "Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max."
- Description ***: A text input field. Below it, a note states: "Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max."
- Website ***: A text input field. Below it, a note states: "Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)"
- Callback URL**: A text input field. Below it, a note states: "Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank."

En la pantalla de configuración de la nueva aplicación será necesario especificar un nombre para la aplicación, una pequeña descripción, y una URL, además de aceptar las condiciones de uso.

Una vez creada la nueva aplicación habrá que entrar en la configuración de la misma para especificar los mecanismos de autenticación y los permisos.

En la pestaña “Keys and Access Token” se administran todas las formas de autenticación disponibles para interactuar con el API Rest. Existen dos tipos disponibles:

- Con credenciales OAuth del usuario
- Con credenciales de aplicación

La diferencia entre los dos métodos es que en el primero autentica como si fuese el usuario, por lo que podrá realizar muchas más acciones que necesitan un contexto de usuario respecto a la autenticación por aplicación (p.j. publicar un Tweet, mandar un mensaje privado, etc). Y el segundo método autentica la aplicación.

Otra característica entre las dos son los límites de velocidad de invocación a cada método del API, siendo diferentes por cada método y por cada mecanismo de autenticación utilizada.

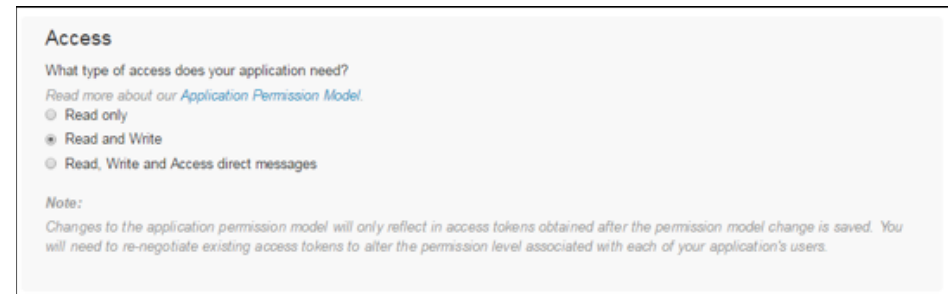
Para poder realizar la autenticación OAuth del usuario se deberá tomar nota de los dos campos:

- Costumer API (API KEY)
- Consumer Secret (API Secret)

Para autenticarse con credenciales de aplicación se deberá generar unos nuevos Tokens pulsando sobre la opción “Create my Access token” lo cual generará un par de Tokens nuevos:

- Access Token
- Access Token Secret

En la pestaña “Permissions” se podrá configurar los permisos que tendrán las credenciales de autenticación y estos limitarán la operativa que se podrá realizar desde los Scripts o programas desarrollados:



The screenshot shows a web interface titled "Access". It asks "What type of access does your application need?" and provides a link to "Read more about our Application Permission Model." There are three radio button options: "Read only", "Read and Write" (which is selected), and "Read, Write and Access direct messages". Below the options is a "Note:" section stating: "Changes to the application permission model will only reflect in access tokens obtained after the permission model change is saved. You will need to re-negotiate existing access tokens to alter the permission level associated with each of your application's users."

Después de estas configuraciones ya está todo preparado para poder interactuar con el API desde los diversos desarrollos que se deseen realizar.

Si no queremos utilizar ninguna librería de terceros, podemos interactuar con el API utilizando peticiones HTTP estándar. Para ello primero será necesario gestionar el proceso de autenticación por lo que será necesario el disponer de librerías de Python que lo manejen. En este caso será necesaria la instalación de la librería `request_oauthlib` que se puede encontrar en su Github oficial: <https://github.com/requests/requests-oauthlib>

Para la instalación será necesario descargar la rama Master del repositorio. Una vez descomprimida la carpeta se ejecutará en consola:

```
# python setup.py install
```

Búsqueda simple

El siguiente ejemplo muestra una ejecución sencilla donde se interactúa con el API gestionando las credenciales OAuth. Habrá que colocar los tokens obtenidos en la pestaña de métodos de autenticación vista previamente:

```
import requests
from requests_oauthlib import OAuth1

CONSUMER_KEY = "XXXX"
CONSUMER_SECRET = "XXXX"
OAUTH_TOKEN = "XXXXX"
OAUTH_TOKEN_SECRET = "XXXX"

oauth = OAuth1(CONSUMER_KEY,
               client_secret=CONSUMER_SECRET,
               resource_owner_key=OAUTH_TOKEN,
               resource_owner_secret=OAUTH_TOKEN_SECRET)

query = "%40twitterapi"
r = requests.get(url="https://api.twitter.com/1.1/search/tweets.
json?q=" + query, auth=oauth)
```

En este ejemplo se ha utilizado la funcionalidad de búsqueda de Twitter, donde podemos pasarle una Query como parámetro de entrada. Hay que destacar que esta Query debe de estar en un formato URLEncoder, por lo que habrá que cambiar el formato antes de realizar la petición.

Las búsquedas en Twitter también disponen de una serie de operadores y filtros. Las principales son:

- Operadores AND, OR, +, -
- From: permite especificar la cuenta origen de los Tweets
- To: permite especificar los Tweet con destino a una cuenta
- # y @: operadores propios de Twitter que corresponden con Hastag y Mentions.
- Filter: Twitter permite la especificación de filtros relativos al contenido del Tweet. Por ejemplos:
 - Filter: media - contiene una imagen o un video.
 - Filter: image - contiene una imagen.
 - Filter: links - contiene un link con una URL.
 - Filter: periscope - contiene una URL de Periscope.

Se pueden consultar el listado completo en la documentación del método Search: <https://dev.twitter.com/rest/public/search>

Tweepy

Para la realización de los siguientes ejemplos nos centraremos en el uso de la librería Tweepy. El primer paso es la descarga e instalación de la misma. Se puede instalar descargando la rama Master desde el repositorio oficial ubicado en Github o mediante la herramienta PIP:

```
#pip install tweepy
```

En el siguiente ejemplo se van a importar las librerías necesarias y se va a instanciar la librería completa sobre la que se realizará el resto de interacciones con el API de Twitter:

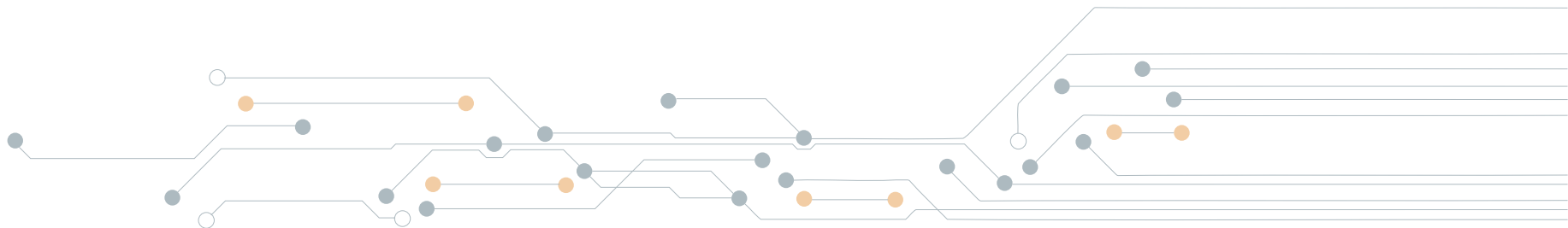
```
import tweepy

consumer_key = 'YOUR-CONSUMER-KEY'
consumer_secret = 'YOUR-CONSUMER-SECRET'
access_token = 'YOUR-ACCESS-TOKEN'
access_secret = 'YOUR-ACCESS-SECRET'

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)

api = tweepy.API(auth)
```

Sobre el objeto 'api' que se acaba de instanciar se utilizarán todas las funcionalidades que nos ofrece la librería Tweepy.



TimeLine de un usuario

Para el primer ejemplo de uso de esta librería se va a consultar el Time Line de una determinada cuenta de usuario (la propia u otra). Para ello se usará el método `user_timeline()` de la librería, el cual devuelve como salida una lista de objetos del tipo "Status". Estos objetos contienen toda la información relativa sobre el Tweet y sobre su autor. Los principales atributos son:

- **id:** identificador único del Tweet.
- **created_at:** fecha de creación
- **text:** texto del Tweet
- **source:** atributo que nos indica con qué herramienta se envió el Tweet (Twitter Web, Twitter iPhone, Twitter Android, Twitter Ads, TweetDeck, Flipboard, etc).
- **geo, place, coordinates:** atributos relativos a la geolocalización del Tweet si en el momento de ser enviado estaba activa.
- **user:** información completa sobre el autor del Tweet. Este atributo es del tipo "User" y contiene un montón de atributos más.
- **entities:** información estructurada sobre el contenido del Tweet (symbols, Hashtag, user_mentions, URLS).
- **reTweet_count, favorite_count:** número de Retweet y de favoritos.
- **in_reply_to_user_id, in_reply_to_status_id:** si el Tweet se trata de un "reply", identifica al usuario del Tweet inicial y al Tweet en concreto.

En el ejemplo se va a mostrar la llamada a este método, y sobre la lista de resultados se va a obtener el nombre del autor, la fecha de creación y el texto del Tweet:

```
timeline = api.user_timeline('wikipedia')

for tweet in timeline:
    print("Nombre de la cuenta: ", tweet.user.screen_name)
    print("Fecha de creación: ", tweet.created_at)
    print("Texto del Tweet: ",tweet.text)
    print(".....")
```

Con este ejemplo se pueden observar claramente las diferencias de procesamiento del resultado respecto a la interacción con el API Rest mediante peticiones HTTP normales, donde había que Parsear manualmente el JSON.

Información de un usuario

Para obtener información sobre un perfil de usuario se puede utilizar el método `get_user()` de la librería. Este método devuelve como resultado un objeto del tipo `User`. Este tipo contienen toda la información relativa sobre la cuenta de un usuario. Los principales atributos son:

- **id:** identificador único del usuario.
- **screen_name, name:** nombre de la cuenta y nombre a mostrar.
- **created_at:** fecha de creación de la cuenta.
- **followers_count:** número de Followers.
- **favourites_count:** número de Tweets favoritos.
- **description:** descripción del perfil del usuario.
- **profile_image_url_https:** URL de la imagen del usuario.
- **statuses_count:** número de Tweets de la cuenta.
- **entities:** información estructurada sobre el contenido de la cuenta (URLs, etc).
- **geo_enabled:** si la cuenta de usuario tiene la geolocalización activada.

Para mostrar el funcionamiento de este método, se va a mostrar como salida los campos más representativos:

```
user = api.get_user('wikipedia')

print("Identificador único de la cuenta: ", user.id)
print("Nombre: ", user.screen_name)
print("Nombre mostrado: ", user.name)
print("Descripción: ", user.description)
print("Fecha de creación: ", user.created_at)
print("Followers: ", user.followers_count)
print("Numero de tweets: ", user.statuses_count)
print("¿Geolocalización activa?: ", user.geo_enabled)
```

Obteniendo como resultado de la ejecución:

```
Identificador único de la cuenta: 86390214
Nombre: Wikipedia
Nombre mostrado: Wikipedia
Descripción: The official Twitter account for the sum of all
knowledge, Wikipedia. Overseen by @jeffelder.
Fecha de creación: 2009-10-30 20:29:14
Followers: 346296
Numero de tweets: 8766
¿Geolocalización activa?: True
```

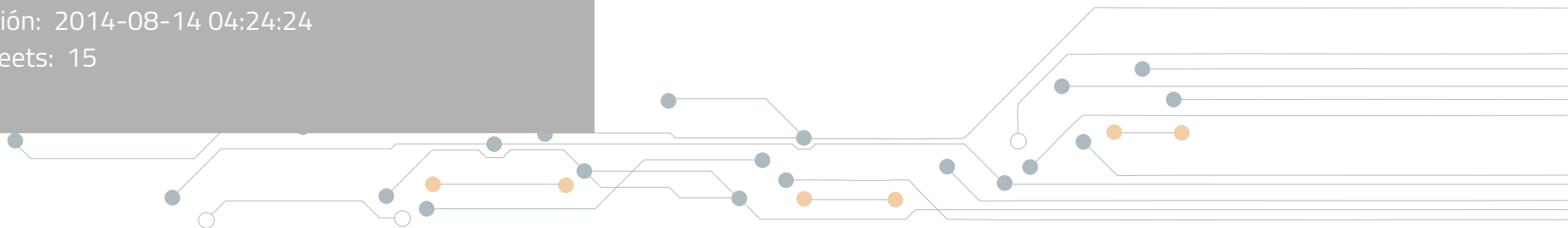
Followers

Para obtener el listado de followers a los que sigue una determinada cuenta de usuario (la propia u otra) se puede usar el método *followers()* de la librería. En el ejemplo se mostrará por pantalla los el nombre de la cuenta, la fecha de creación, y el número de Tweets enviados:

```
for follower in followers:
    print("Nombre de la cuenta: ", follower.screen_name)
    print("Fecha de creación: ", follower.created_at)
    print("Número de Tweets: ", follower.statuses_count)
    print(".....")
```

Obteniendo como resultado:

```
Nombre de la cuenta: jenn_haufler
Fecha de creación: 2009-09-28 12:53:21
Número de Tweets: 4101
.....
Nombre de la cuenta: r3z01v
Fecha de creación: 2016-09-17 14:55:23
Número de Tweets: 579
.....
Nombre de la cuenta: maomaoleba
Fecha de creación: 2014-08-14 04:24:24
Número de Tweets: 15
...
```



Friends

Del mismo modo se pueden obtener el listado de cuentas a la que sigue una determinada cuenta de usuario (la propia u otra). Para ello se usará el método *friends()* de la librería. Del mismo modo que el ejemplo anterior, se mostrará de salida el nombre de la cuenta, la fecha de creación, y el número de Tweets enviados:

```
friends = api.friends('wikipedia')

for friend in friends:
    print("Nombre de la cuenta: ", friend.screen_name)
    print("Fecha de creación: ", friend.created_at)
    print("Número de Tweets: ", friend.statuses_count)
    print(".....")
```

Obteniendo como resultado:

```
Nombre de la cuenta: WikiEval
Fecha de creación: 2014-05-22 23:36:59
Número de Tweets: 2318
.....
Nombre de la cuenta: pt
Fecha de creación: 2007-10-21 01:50:51
Número de Tweets: 13028
.....
Nombre de la cuenta: dhacker615
Fecha de creación: 2007-11-20 05:17:35
Número de Tweets: 8730
...
```



Paginación

Hasta este momento se ha podido observar que el número de resultados obtenidos en los diferentes ejercicios están limitados a 20 respuestas. Esto es por una limitación de la API de Twitter que divide los resultados en Páginas. Para poder obtener el resto de resultados es necesario manejar la paginación.

Para manejar la paginación en las peticiones realizadas directamente contra la API Rest de Twitter (sin utilizar librerías de terceros) se pueden realizar de dos formas distintas:

- Utilizando `since_id` y `max_id`
- Utilizando cursores

La primera forma utilizando los parámetros `max_id` y `since_id` es la más antigua y más manual de las 2. Los parámetros `max_id` y `since_id` se agregan a la URL de petición indicando a Twitter cual fue el último Tweet ID que se ha manejado. De esta forma Twitter elige qué Tweets tiene que devolver en la siguiente respuesta:

- Con el parámetro `max_id` se le indica a Twitter que devuelva los Tweets anteriores al Tweet indicado (los Tweet más viejos).
- Con el parámetro `since_id` se le indica a Twitter que devuelva los Tweets más nuevos al Tweet indicado.

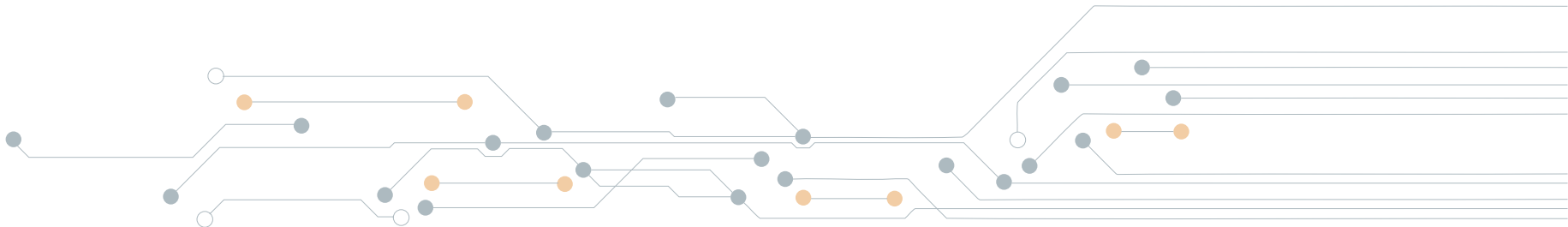
Los cursores es una técnica desarrollada por Twitter en la cual separa el resultado de una búsqueda en diferentes páginas (del tamaño que se ha definido al realizar la primera consulta mediante el parámetro "count") y devuelve unos parámetros para poder mover la búsqueda adelante y atrás entre las diferentes páginas.

Este movimiento de páginas se realiza mediante los parámetros denominados "previous_cursor" y "next_cursor", y que devueltos en el resultado de las peticiones.

Para poder realizar esta técnica, es necesario empezar la primera búsqueda pasándole como entrada el parámetro "cursor" con valor a -1. Este valor indica a Twitter que se va a utilizar cursores en esta búsqueda. El resultado de esta búsqueda devolverá los parámetros anteriores indicando cual es la siguiente página y la previa.

En las siguientes peticiones de búsqueda, el parámetro "cursor" deberá de tomar el valor de la página a la que se desea mover ("previous_cursor" o "next_cursor").

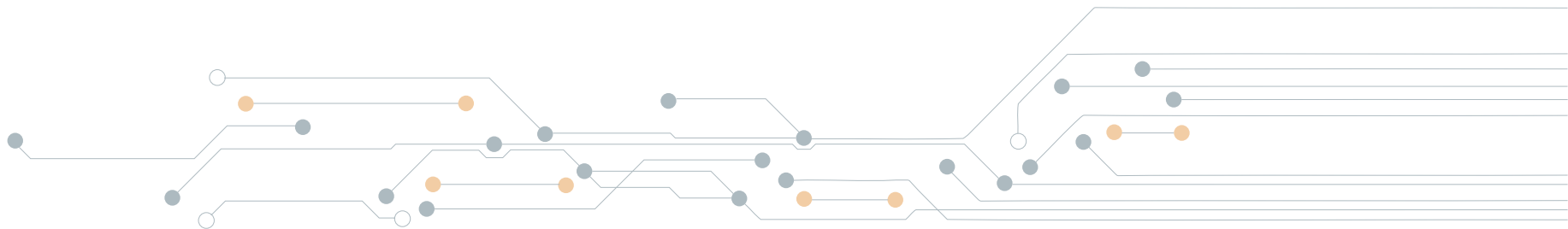
Cuando se ha navegado hasta la última página de resultados, el valor de `next_cursor` se tornará a 0, indicando que no existen más páginas.



En el siguiente ejemplo se va a realizar una búsqueda de todos los Followers de una cuenta de Twitter concreta. Se va a configurar el parámetro "count" para que devuelva el máximo de respuestas posibles por cada petición (limitado por Twitter a 100). Para obtener todas las páginas con las respuestas, se realizará un bucle While que esté iterando hasta que el resultado del parámetro "next_cursor_str" tenga un valor de 0. Se irá mostrando por pantalla el valor de los 2 cursores:

```
cursor = -1

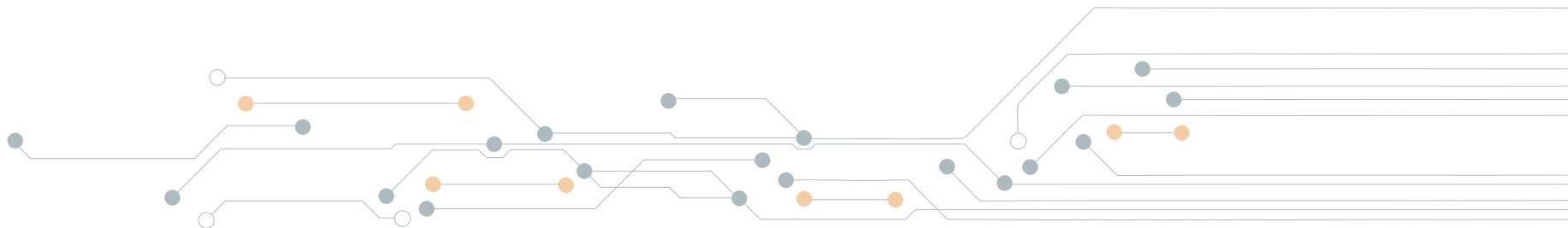
while cursor != 0:
    urls = "https://api.twitter.com/1.1/followers/ids.json?"
    user = "twitterapi"
    r = requests.get(url= urls + "count=100" + "&screen_name=" + user + "&cursor=" + str(cursor),
auth=oauth)
    cursor = int(r.json()['next_cursor_str'])
    previous_cursor = int(r.json()['previous_cursor_str'])
    print("Next cursor: ", cursor, " -- Previous cursor: ",
previous_cursor)
```



Como resultado de la ejecución de este ejemplo se obtiene:

```
Next cursor: 1550325529014068265 -- Previous cursor: 0
Next cursor: 1550316600223167665 -- Previous cursor: -1550325526812869836
Next cursor: 1550308253790304614 -- Previous cursor: -1550316496050981732
Next cursor: 1550298775435923739 -- Previous cursor: -1550308243524008931
Next cursor: 1550287685502372272 -- Previous cursor: -1550298765007081415
Next cursor: 1550280381859416072 -- Previous cursor: -1550287615633950600
```

Se puede observar como el valor del cursor previo en la primera petición tiene un valor 0, indicando que no hay más páginas previa a esta.



Paginación con Tweepy

La librería Tweepy también tiene 2 formas diferentes de gestionar la paginación. Cualquiera de las 2 formas es mucho más simple que las peticiones HTTP manuales, ya que abstrae toda la complejidad de las peticiones y del manejo de las respuestas.

Las 2 alternativas disponibles son:

- Utilizando el parámetro "pages" en los métodos
- Utilizando los cursores

La primera alternativa que nos ofrece la librería es la más antigua y es necesario manejar las iteraciones entre las diferentes páginas manualmente.

Esto implica la realización de un bucle que vaya incrementando el valor del parámetro "pages". Este parámetro se pasará como entrada a los diferentes métodos de búsqueda de la librería para que devuelva en la respuesta los resultados correspondientes a las diferentes páginas.

Se puede saber en qué momento se han terminado las páginas de resultados porque el resultado del método no contendrá ningún valor:

```
page = 1
while True:
    statuses = api.user_timeline('wikipedia', page=page)
    if statuses:
        for status in statuses:
            print("Nombre de la cuenta: ", status.user.screen_name)
            print("Fecha de creación: ", status.created_at)
            print("Texto del Tweet: ", status.text)
            print(".....")
    else:
        break
    page += 1
```

Como se puede observar, esta alternativa requiere una gran cantidad de líneas de código para poder manejar la paginación y comprobar cuando no existen más páginas.

La segunda alternativa es el uso de la Clase Cursor, la cual abstrae toda la complejidad de la alternativa anterior.

```
tweepy.Cursor(api.user_timeline)
```



Si el método de búsqueda tiene parámetros de entrada, estos los podemos agregar añadiéndolos como entrada en la creación del cursor:

```
api.user_timeline(id="wikipedia")
tweepy.Cursor(api.user_timeline, id="wikipedia")
```

Las respuestas se manejan de forma muy sencilla incorporando el Cursor en un bucle for-in. Es posible iterar tanto por Items como por Páginas:

```
for status in tweepy.Cursor(api.user_timeline).items():
    process_status(status)

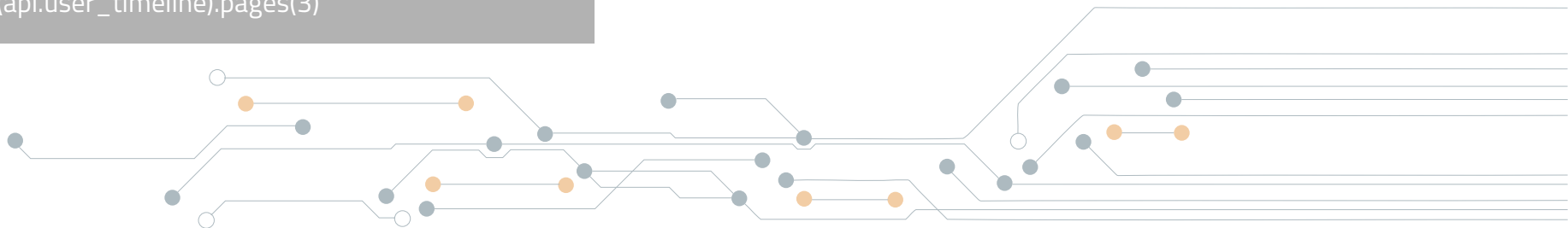
for page in tweepy.Cursor(api.user_timeline).pages():
    process_page(page)
```

Además, permite especificar límites en el mismo Cursor sobre el número de respuestas que se devuelve. De esta forma el Cursor solo buscará y devolverá la información que se necesita:

```
tweepy.Cursor(api.user_timeline).items(200)
tweepy.Cursor(api.user_timeline).pages(3)
```

En el ejemplo se va a mostrar los primeros 200 Tweets del Timeline del usuario "Wikipedia":

```
for status in tweepy.Cursor(api.user_timeline, id="wikipedia").
items(200):
    print("Nombre de la cuenta: ", status.user.screen_name)
    print("Fecha de creación: ", status.created_at)
    print("Texto del Tweet: ", status.text)
    print(".....")
```



Relaciones entre usuarios

Una de las funcionalidades más interesantes de la librería es el método `show_friendship()`. Este método devuelve información relativa a la relación entre 2 cuentas de usuario, como por ejemplo si se siguen entre sí, si la cuenta origen tiene bloqueada a la cuenta destino, si la tiene silenciada, si la tiene reportada como Spam, etc.

El resultado de este método es una Tupla con 2 objetos del tipo `Relationship`. Cada objeto muestra información relativa a uno de los sentidos de la relación. La información más interesante que se puede obtener es:

- De la cuenta origen:
 - `following`: si sigue la cuenta destino
 - `followed_by`: si es seguido por la cuenta destino
 - `blocking`: si tiene bloqueada a la cuenta destino
 - `muting`: si tiene silenciada a la cuenta destino
 - `marked_spam`: si tiene marcada como Spam a la cuenta destino
 - `can_dm`: si puede enviarle mensajes privados
- De la cuenta destino:
 - `following`: si sigue la cuenta origen
 - `followed_by`: si es seguido por la cuenta origen

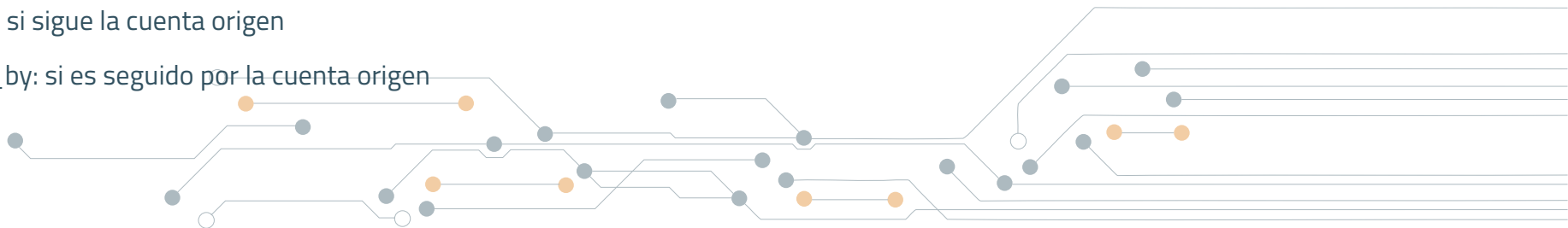
En el ejemplo se va a mostrar el funcionamiento de este método:

```
source, target = api.show_friendship(source_screen_name='wikipedia', target_screen_name='jeffelder')
```

El resultado de las dos relaciones es:

```
>>> source
Friendship(want_retweets=None, following_requested=None,
following=True, id=86390214, id_str='86390214', _api=<tweepy.
api.API object at 0x034F8950>, live_following=False,
followed_by=True, muting=None, following_received=None,
all_replies=None, screen_name='Wikipedia', can_dm=True,
blocking=None, marked_spam=None, notifications_
enabled=None, blocked_by=None)

>>> target
Friendship(following_requested=None, following=True,
id=14478614, id_str='14478614', _api=<tweepy.api.API object
at 0x034F8950>, followed_by=True, following_received=None,
screen_name='JeffElder')
```



Geolocalización

Otra de las funcionalidades más interesantes que se pueden realizar con la API de Twitter es la relativa a la geolocalización. Como hemos visto anteriormente, mediante el método `get_user()`, se obtienen datos relativos a un User, incluyendo si tiene la geolocalización activada o no.

Dependiendo del programa con el que envíe el Tweet, si este tiene acceso a algún tipo de geolocalización (por ejemplo en móviles), el Tweet creado contará con información de la geolocalización del punto donde se ha enviado el Tweet.

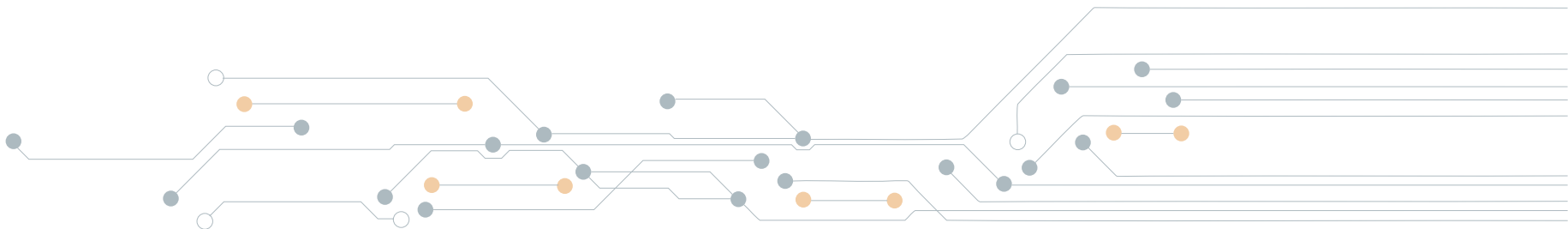
Esto ofrece una información muy valiosa sobre el posible objetivo. Se puede determinar si un usuario se encuentra de viaje o de vacaciones por lo que se encuentra fuera de casa. Se puede determinar cuál es su vivienda habitual o su lugar trabajo analizando la geolocalización de todos los Tweet y visualizar desde qué punto se envían gran cantidad de los mismos.

Para ello se utilizará el método `user_timeline()` visto previamente, pero en este caso se mostrará los atributos 'place' y 'coordinates' del objeto User. Además de mostrar también el atributo 'source' que mostrará la herramienta con qué se envió el Tweet.

Una de las cuentas famosas que tiene habilitada la geolocalización es la de "Steve Wozniak", con lo que se utilizará para realizar el ejemplo:

```
user = api.get_user('stevewoz')
print("¿Geolocalización activa?: ", user.geo_enabled)

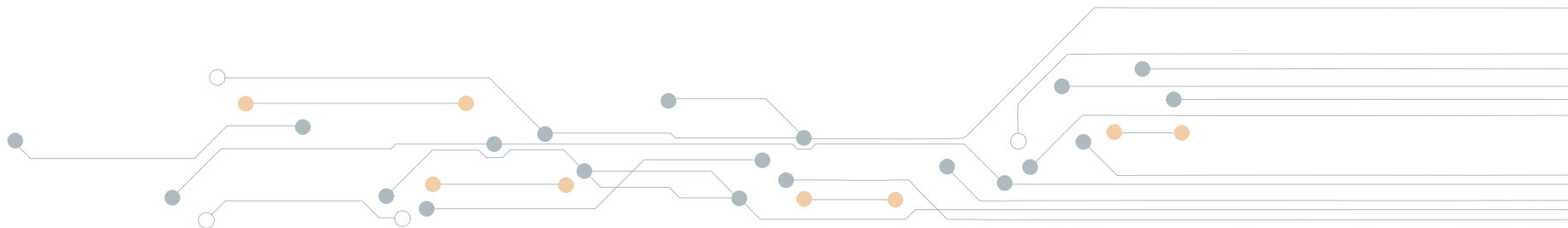
timeline = api.user_timeline('stevewoz')
for tweet in timeline:
    print("Aplicación de envío: ", tweet.source)
    print("Coordenadas: ", tweet.coordinates['coordinates'])
    print("Nombre del lugar: ", tweet.place.name)
    print("Tipo del lugar: ", tweet.place.place_type)
    print("Nombre completo: ", tweet.place.full_name)
    print("País: ", tweet.place.country)
    print(".....")
```



Y se obtiene como salida:

```
Aplicación de envío: Foursquare
Coordenadas: [-56.01661316, -34.83683178]
Nombre del lugar: Canelones
Tipo del lugar: admin
Nombre completo: Canelones, Uruguay
País: Uruguay
.....
Aplicación de envío: Foursquare
Coordenadas: [-58.54228305, -34.81608353]
Nombre del lugar: Buenos Aires
Tipo del lugar: admin
Nombre completo: Buenos Aires, Argentina
País: Argentina
.....
Aplicación de envío: Foursquare
Coordenadas: [2.57624253, 49.0018188]
Nombre del lugar: Le Mesnil-Amelot
Tipo del lugar: city
Nombre completo: Le Mesnil-Amelot, France
País: Francia
```

Como se puede observar, "Steve Wozniak" viaja bastante, y con un simple Script de Python es posible monitorizar su Time Line identificando en todos los puntos que se encuentra en cada momento pudiendo hacer un seguimiento del mismo.



Geolocalización inversa

Del mismo modo, Tweepy permite realizar búsquedas inversas, es decir, dadas unas coordenadas y un radio, devuelve todos los Tweet realizados en esa zona que tienen activa la geolocalización.

Esto es muy útil ya que se puede crear un Script a la escucha apuntando a unas coordenadas concretas e ir monitorizando todo lo que se Twittea desde ese lugar. Si estas coordenadas son relativas a un lugar de trabajo, o un evento con información sensible donde la entrada está restringida, etc, se puede llegar a obtener información importante que se filtra sin querer.

Para el ejemplo se ha elegido las coordenadas de la Puerta del Sol de Madrid (40.4169473,-3.7057172) y de radio 1Km.

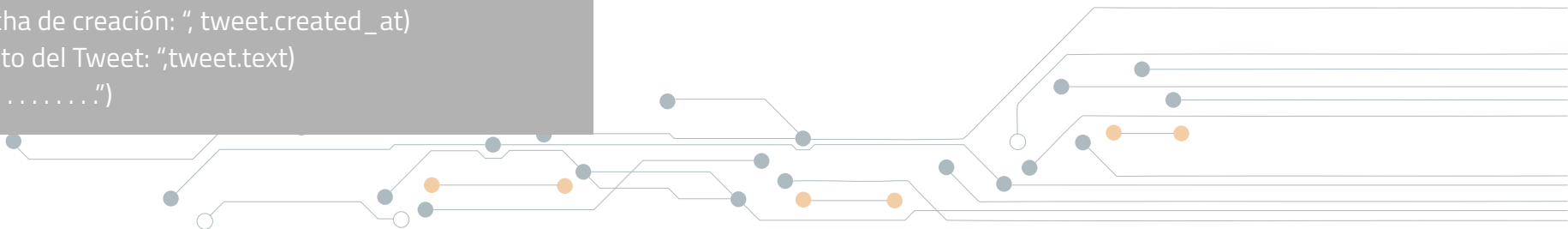
```
tweets = api.search(geocode="40.4169473,-3.7057172,1km")
```

Como resultado se obtiene una lista de Status con la información de los Tweets. Se filtrará solos los atributos que se desean, en este caso el nombre de la cuenta, la fecha de creación del Tweet y el texto del mismo:

```
for tweet in tweets:
    print("Nombre de la cuenta: ", tweet.user.screen_name)
    print("Fecha de creación: ", tweet.created_at)
    print("Texto del Tweet: ",tweet.text)
    print(".....")
```

Obteniendo como resultado:

```
Nombre de la cuenta: proudxpos
Fecha de creación: 2016-11-06 16:23:35
Texto del Tweet: Lali ganó el EMA https://t.co/ykKVGVNNOD
.....
Nombre de la cuenta: kennysullivan
Fecha de creación: 2016-11-06 16:22:51
Texto del Tweet: Fall in Madrid (@ Jardines de Sabatini in Madrid)
https://t.co/dpux0HX3YS https://t.co/pekJJYddJw
.....
Nombre de la cuenta: trendinaliaES
Fecha de creación: 2016-11-06 16:20:53
Texto del Tweet: 'Hermione' acaba de convertirse en TT ocupando
la primera posición en España. Más en https://t.co/K5DFqqcseW
#trndnl https://t.co/ihzGYrGefv
```



Streams

Por último, una de las funcionalidades más interesantes y de la que se le puede sacar más provecho es el uso de los Streams. Los Streams en Twitter es una forma de interaccionar con la API Rest de modo que el Script se queda a la escucha con unos determinados parámetros y filtros hasta que se produce un evento que coincide con la búsqueda deseada y se notifica al instante. De esta forma se obtiene información en tiempo real, y aunque el autor del mismo borre dicho Tweet, el Script ya ha recibido la notificación del mismo.

Para poder utilizar los Streams de la librería, es necesario crear un Listener que herede de la Clase `tweepy.StreamListener`. En este Listener se pueden incluir varias acciones que con la Clase original no se podría realizar:

```
class CustomStreamListener(tweepy.StreamListener):
    def on_status(self, status):
        print("Nombre de la cuenta: ", status.user.screen_name)
        print("Fecha de creación: ", status.created_at)
        print("Texto del Tweet: ", status.text)
        print(".....")

    def on_error(self, status_code):
        print('Se ha encontrado un error con el código:', status_code)
        return True
```

Para poder filtrar el Stream para especificar a qué tiene que estar a la escucha, se utilizará el método `filter()`. Es posible filtrar por varios atributos, pero los 3 más importantes son:

- **follow:** permite filtrar el Stream para que se quede a la escucha de una determinada cuenta de usuario. Pero cabe destacar que no devuelve toda la información relativa a ella:
 - Información que devuelve:
 - Tweets creados por la cuenta
 - Tweets re-Twiteados por el usuario
 - Replies sobre los Tweets creados por la cuenta de usuario
 - Re-Tweets de cualquier Tweet creado por el usuario
 - Información que no devuelve:
 - Tweets donde se mencione al usuario
 - Re-Tweets creados mediante un API (no han sido generados pulsando el botón de re-Tweet)
 - Tweets de usuarios protegidos
- **track:** permite filtrar el Stream para que se quede a la escucha de determinadas palabras que tienen que aparecer en el texto del Tweet. Como filtro se especifica una lista de palabras separada por comas donde se especifica los textos. Cada frase puede estar compuesta de uno o más términos separados por espacios, además de incluir operadores lógicos OR, AND, etc.

- **locations:** permite filtrar el Stream para especificar unas coordenadas geográficas concretas y solo recibir los Tweet generados en dicha área. El filtro recibe como dato una lista separada por comas de coordenadas geográficas (pares longitud/latitud).

En el ejemplo se va a filtrar el Stream para que se quede a la escucha de los Tweets generados en el área de la Puerta del Sol de Madrid:

```
stream.filter(locations=[ -3.705557, 40.415274, -3.701496,  
40.418309])
```

Y se obtendrá como salida del Script una lista creciente de elementos según se van recibiendo:

Nombre de la cuenta: pilartaratoruga
Fecha de creación: 2016-11-06 17:31:26
Texto del Tweet: @cakivi HAZLO

.....

Nombre de la cuenta: Anitaa_mm
Fecha de creación: 2016-11-06 17:31:26
Texto del Tweet: @_belenmartinez pues come.

.....

Nombre de la cuenta: ElGatitoDeShrek
Fecha de creación: 2016-11-06 17:31:26
Texto del Tweet: Muchísimas gracias, pero por favor no me robes Vallecanao <https://t.co/Hi6e1XBYVr>

.....

Nombre de la cuenta: BarnatanMR
Fecha de creación: 2016-11-06 17:31:26
Texto del Tweet: @historiadesal ese plurar de no querer no me incluye.



5.4 | Pastebin

Pastebin es una página web donde se puede almacenar textos de gran tamaño por un periodo de tiempo determinado. Se puede acceder a ellos muy sencillamente a partir de una URL o directamente desde la página web. Básicamente es un portapapeles en la nube orientado a texto.

Todos los pastes que se realizan son públicos, esto significa que en el momento de publicar un paste este aparecerá en la sección de últimos pastes de la página web pudiendo ser visto por todo el mundo.

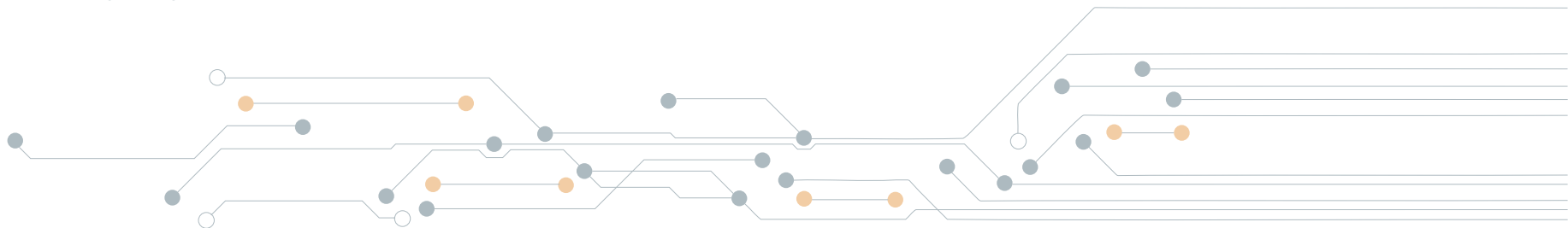
La publicación de un paste se puede realizar de forma anónima o con una cuenta de usuario autenticada. La ventaja de la segunda forma es que se puede administrar los pastes enviados teniendo la posibilidad de borrarlos.

Estos sitios web de pastes son muy famosos en la actualidad, ya que los hackers lo utilizan para filtrar información obtenida de sus ataques. Por ese motivo, las páginas web de pastes se convierten en una fuente de información importante que puede ser utilizada.

Pastebin cuenta con un API Rest (<http://pastebin.com/api>) con la que poder interaccionar para poder obtener información de esta plataforma.

Para poder hacer uso del API es necesario tener una cuenta en la página y generar un API Key que se deberá de utilizar en las invocaciones al API.

Las dos únicas opciones que ofrece el API para obtener información de la página es listar los Trending Pastes, y realizar un Scraping completo de todos los pastes por antigüedad.



Para listar los Trending Pastes habrá que generar una petición POST contra el recurso `api_post.php` del API, añadiendo el parámetro `'api_option'` con el valor de `'trends'`. También será necesario incorporar el API Key generado previamente en el parámetro `'api_dev_key'`.

```
import requests

api_key = "XXXXXX"
api_option = "trends"

url = "http://pastebin.com/api/api_post.php"

r = requests.post(url, data={"api_option":api_option,"api_dev_key":api_key})

print(r.status_code, r.reason)
print(r.text)
```

El resultado de esta petición devuelve un XML con los paste más importantes en este momento. La estructura del resultado (por paste) es la siguiente:

```
<paste>
<paste_key>36Q0yKSM</paste_key>
<paste_date>1478360077</paste_date>
<paste_title>Clinton Underground Scandal PART 1</paste_title>
<paste_size>14486</paste_size>
<paste_expire_date>0</paste_expire_date>
<paste_private>0</paste_private>
<paste_format_short>text</paste_format_short>
<paste_format_long>None</paste_format_long>
<paste_url>http://pastebin.com/36Q0yKSM</paste_url>
<paste_hits>283122</paste_hits>
</paste>
```

La otra forma de obtener información de esta página es hacer un Scraping de todos los paste e ir analizándolos uno por uno para obtener información valiosa de todos ellos. El único problema de esta funcionalidad es que solo está disponible para los usuarios de pago, siendo necesario registrar una IP desde donde se realizará las peticiones. Se pueden consultar los detalles de esta técnica en la página del API que ofrece Pastebin para este cometido:

http://pastebin.com/api_scraping_faq



6. Recolección de Data Leaks

En los últimos años se está escuchando más a menudo de múltiples intrusiones a grandes corporaciones y sitios webs en los cuales les han robado miles de datos. Toda esta información acaba filtrada en internet o en las redes alternativas (DarkNet, Tor).

Los Data Leak es un conjunto de información que se hace pública en internet (o en las redes alternativas: DarkNet, Tor, etc) la cual ha sido obtenida tras un incidente de seguridad ya sea de forma externa o mediante un filtrado interno.

Esta información que se hace pública suele ser muy sensible, teniendo información protegida y confidencial de los usuarios de ese sistema. Entre esta información se encuentran datos financieros, datos bancarios, datos personales, etc.

En los últimos años se ha incrementado el número de Data Leaks que han visto la luz y el impacto de los mismos. Los más populares han sido el de LinkedIn donde se filtraron 164 millones de registros, Adobe con 152 millones, Ashley Madison con 30 millones o Mate 1 con 27 millones.

Uno de los datos más interesantes que se encuentran en los Data Leaks, y que son muy útiles para la realización de los test de intrusión, son las cuentas de correo y las *passwords* asociadas.

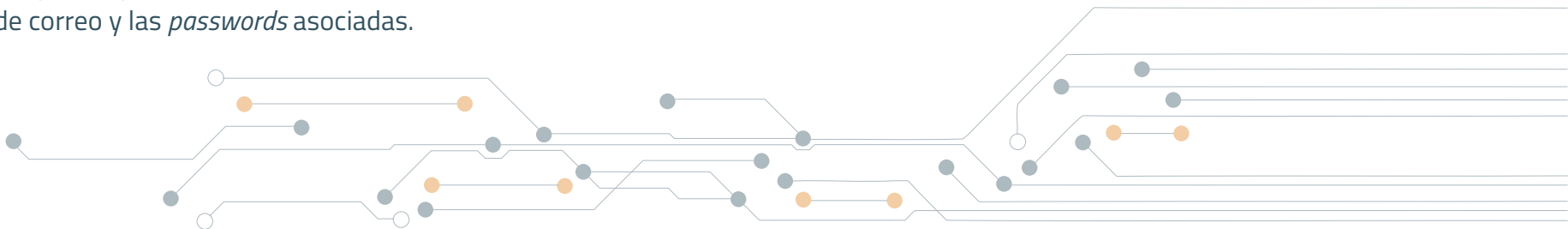
Con las técnicas de los apartados anteriores podemos encontrar diversas cuentas de correo de personal de la organización. Estas cuentas pueden estar involucradas en alguno de los múltiples DataLeaks, por lo que la *password* utilizada en estos sitios es muy posible que se haya publicado.

La mayoría de usuarios reutiliza las contraseñas en varios sitios, por lo que es muy probable que esa *password* filtrada también sea válida para otros sitios, incluyendo el propio correo o las credenciales corporativas de una organización.

Normalmente las *passwords* filtradas están ofuscadas utilizando algún algoritmo de Hasheo (en muchos casos también se filtra el Salt utilizado) bastante básico. Utilizando técnicas de fuerza bruta, diccionarios, búsquedas inversas, etc, es posible recuperar dicho dato en claro.

Hay varios sitios en internet que nos ofrecen la posibilidad de comprobar si un email ha sido publicado en algún DataLeak. En este apartado se explicará la integración y uso de dos de ellos:

haveibeenpwned.com y **hesidohackeado.com**



haveibeenpwned

Este sitio web cuenta con una gran cantidad de datos almacenados obtenidos de diversos DataLeaks y Pastes:

- 152 DataLeaks de webs completas con un total de 1,801,838,008 cuentas de correo
- 40,801 Pastes en diversas páginas con un total de 33,393,546 cuentas de correo

Aparte del buscador de la página web, este sitio ofrece un API público con la que poder interaccionar mediante scripts. En la página <https://haveibeenpwned.com/API/v2> se encuentra el detalle de todas las funcionalidades que ofrece:

- Obtener todas las brechas donde una cuenta está involucrada
- Obtener todas las brechas del sistema
- Obtener el detalle de una brecha en concreto
- Obtener todos los tipos de datos que están dados de alta en el sistema
- Obtener todos los Pastes de una cuenta concreta

Para poder interaccionar con este API de una forma más sencilla, existe una librería de Python que facilita esta labor. Es posible encontrarla en su Github oficial <https://github.com/icanhasfay/PyPwned>

Para instalarla correctamente, es necesario instalar previamente todas sus dependencias:

```
pip install requests
pip install pyOpenSSL
pip install ndg-httpsclient
pip install pyasn1
```

Y posteriormente se instala la librería con el siguiente comando:

```
pip install pypwned
```

En el siguiente ejemplo se mostrará el funcionamiento de la librería solicitando todas los Data Leaks donde está afectada una determinada cuenta de usuario:

```
>>> import pypwned
>>> result = pypwned.
getAllBreachesForAccount(email="freeman@hotmail.com")
>>> print("Numero de Data Leaks:", len(result))
Numero de Data Leaks: 14
```

Como resultado de la consulta, devuelve una lista de elementos, siendo cada elemento un diccionario con toda la información de un Data Leak diferente. Al ser una estructura muy definida, es posible recorrerla y trabajar con los datos obtenidos.

En el siguiente ejemplo se mostrará el nombre de todos los Data Leaks involucrados:

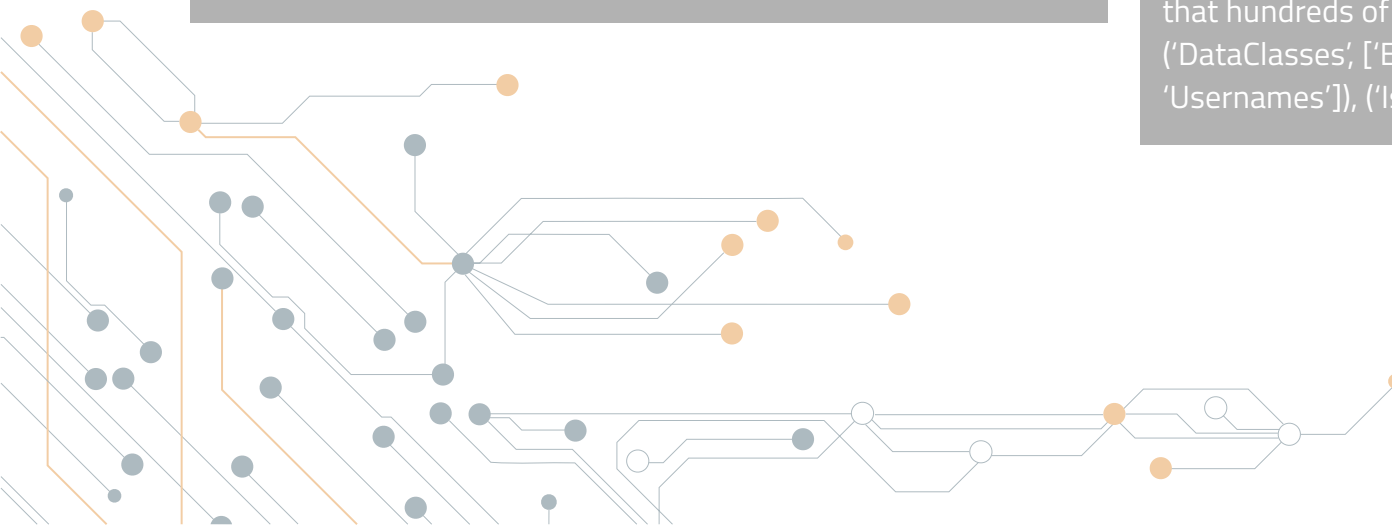
```
>>> for i in result:  
    print("Nombre del Data Leak: " + str(i["Name"]))
```

```
Nombre del Data Leak: Adobe  
Nombre del Data Leak: FlashFlashRevolution  
Nombre del Data Leak: HeroesOfNewerth  
Nombre del Data Leak: iMesh  
Nombre del Data Leak: Lastfm  
Nombre del Data Leak: Leet  
Nombre del Data Leak: Lifeboat  
Nombre del Data Leak: LinkedIn  
Nombre del Data Leak: ModernBusinessSolutions  
Nombre del Data Leak: MPGH  
Nombre del Data Leak: MySpace  
Nombre del Data Leak: Neopets  
Nombre del Data Leak: Tianya  
Nombre del Data Leak: Tumblr
```

Los diccionarios que contienen cada uno de los Data Leaks tienen siempre la misma estructura con las mismas claves. En el siguiente ejemplo se mostrará las claves y los valores para un Data Leak en concreto:

```
>>> print(result[0].items())
```

```
dict_items([('AddedDate', '2013-12-04T00:00:00Z'), ('IsVerified',  
True), ('Name', 'Adobe'), ('Title', 'Adobe'), ('Domain', 'adobe.com'),  
('BreachDate', '2013-10-04'), ('LogoType', 'svg'), ('PwnCount',  
152445165), ('IsRetired', False), ('Description', 'In October 2013,  
153 million Adobe accounts were breached with each containing  
an internal ID, username, email, <em>encrypted</em> password  
and a password hint in plain text. The password cryptography was  
poorly done and <a href="http://stricture-group.com/files/adobe-  
top100.txt" target="_blank" rel="noopener">many were quickly  
resolved back to plain text</a>. The unencrypted hints also <a  
href="http://www.troyhunt.com/2013/11/adobe-credentials-  
and-serious.html" target="_blank" rel="noopener">disclosed  
much about the passwords</a> adding further to the risk  
that hundreds of millions of Adobe customers already faced.'),  
('DataClasses', ['Email addresses', 'Password hints', 'Passwords',  
'Usernames']), ('IsActive', True), ('IsSensitive', False)])
```



La librería también permite consultar si una dirección de correo está en un Data Leak en concreto:

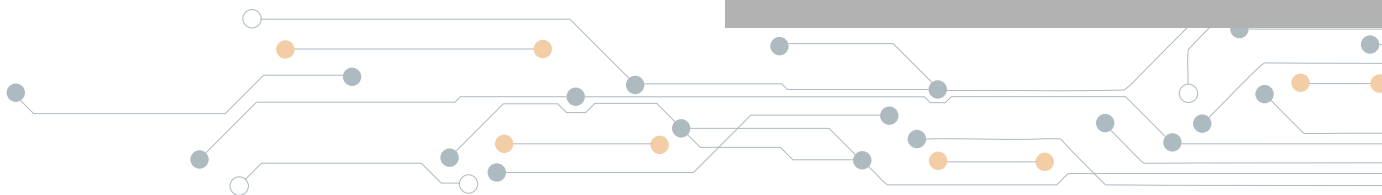
```
>>> result = pypwned.  
getAllBreachesForAccount(email="freeman@hotmail.  
com",domain="adobe.com")  
  
>>> print(result)
```

```
{'AddedDate': '2013-12-04T00:00:00Z', 'IsVerified': True, 'Name':  
'Adobe', 'Title': 'Adobe', 'Domain': 'adobe.com', 'BreachDate': '2013-  
10-04', 'LogoType': 'svg', 'PwnCount': 152445165, 'IsRetired':  
False, 'Description': 'In October 2013, 153 million Adobe accounts  
were breached with each containing an internal ID, username,  
email, <em>encrypted</em> password and a password hint  
in plain text. The password cryptography was poorly done and  
<a href="http://stricture-group.com/files/adobe-top100.txt"  
target="_blank" rel="noopener">many were quickly resolved  
back to plain text</a>. The unencrypted hints also <a href="http://  
www.troyhunt.com/2013/11/adobe-credentials-and-serious.  
html" target="_blank" rel="noopener">disclosed much about  
the passwords</a> adding further to the risk that hundreds of  
millions of Adobe customers already faced', 'DataClasses': ['Email  
addresses', 'Password hints', 'Passwords', 'Usernames'], 'IsActive':  
True, 'IsSensitive': False}]
```

Existe un método para obtener toda la información relativa a todos los Data Leaks ocurridos sobre un dominio en concreto. Esta información puede ser por ejemplo el número de cuentas filtradas, los datos concretos que se han filtrado, si la información es sensible, si el Data Leak ha sido verificado, etc:

```
>>> result = pypwned.getAllBreaches(domain="adobe.com")  
  
>>> print(result)
```

```
{'PwnCount': 152445165, 'IsSensitive': False, 'DataClasses':  
['Email addresses', 'Password hints', 'Passwords', 'Usernames'],  
'AddedDate': '2013-12-04T00:00:00Z', 'IsVerified': True, 'Name':  
'Adobe', 'LogoType': 'svg', 'Title': 'Adobe', 'BreachDate': '2013-  
10-04', 'IsActive': True, 'Domain': 'adobe.com', 'Description':  
'In October 2013, 153 million Adobe accounts were breached  
with each containing an internal ID, username, email,  
<em>encrypted</em> password and a password hint in  
plain text. The password cryptography was poorly done and  
<a href="http://stricture-group.com/files/adobe-top100.txt"  
target="_blank" rel="noopener">many were quickly resolved  
back to plain text</a>. The unencrypted hints also <a href="http://  
www.troyhunt.com/2013/11/adobe-credentials-and-serious.  
html" target="_blank" rel="noopener">disclosed much about  
the passwords</a> adding further to the risk that hundreds of  
millions of Adobe customers already faced', 'IsRetired': False}]
```



Igualmente es posible obtener la misma información para un Data Leak en concreto identificando el nombre del mismo:

```
>>> result = pypwned.getSingleBreachedSite(name="adobe")

>>> print(result)

{'PwnCount': 152445165, 'IsSensitive': False, 'DataClasses':
['Email addresses', 'Password hints', 'Passwords', 'Usernames'],
'AddedDate': '2013-12-04T00:00:00Z', 'IsVerified': True, 'Name':
'Adobe', 'LogoType': 'svg', 'Title': 'Adobe', 'BreachDate': '2013-
10-04', 'IsActive': True, 'Domain': 'adobe.com', 'Description':
'In October 2013, 153 million Adobe accounts were breached
with each containing an internal ID, username, email,
<em>encrypted</em> password and a password hint in
plain text. The password cryptography was poorly done and
<a href="http://stricture-group.com/files/adobe-top100.txt"
target="_blank" rel="noopener">many were quickly resolved
back to plain text</a>. The unencrypted hints also <a href="http://
www.troyhunt.com/2013/11/adobe-credentials-and-serious.
html" target="_blank" rel="noopener">disclosed much about
the passwords</a> adding further to the risk that hundreds of
millions of Adobe customers already faced.', 'IsRetired': False}
```

Del mismo modo que con los Data Leaks, esta librería nos permite consultar contra el API web sobre todos los Pastes (Pastebin, etc) donde aparece una cuenta de correo en concreto. Su funcionamiento es de la misma forma que para los Data Leaks:

```
>>> result = pypwned.
getAllPastesForAccount(account="freeman@hotmail.com")
>>> print("Numero de Pastes:", len(result))
Numero de Pastes: 396

>>> for i in range(5):
    print(result[i])

{'Source': 'Pastebin', 'Id': 'xg6qErM5', 'Date':
'2016-10-26T14:25:57Z', 'EmailCount': 2193, 'Title': None}

{'Source': 'Pastebin', 'Id': 'DyLxrXwQ', 'Date':
'2016-10-20T17:36:00Z', 'EmailCount': 2244, 'Title': None}

{'Source': 'Pastebin', 'Id': '2GncNYs2', 'Date':
'2016-10-03T11:46:05Z', 'EmailCount': 10611, 'Title': None}

{'Source': 'Pastebin', 'Id': 'aUrSRXzy', 'Date':
'2016-09-30T20:44:01Z', 'EmailCount': 5066, 'Title': '5s7f5940'}

{'Source': 'Pastebin', 'Id': 'vT5maSq2', 'Date':
'2016-09-28T21:40:30Z', 'EmailCount': 5066, 'Title':
'xsayr54o05v'}
```

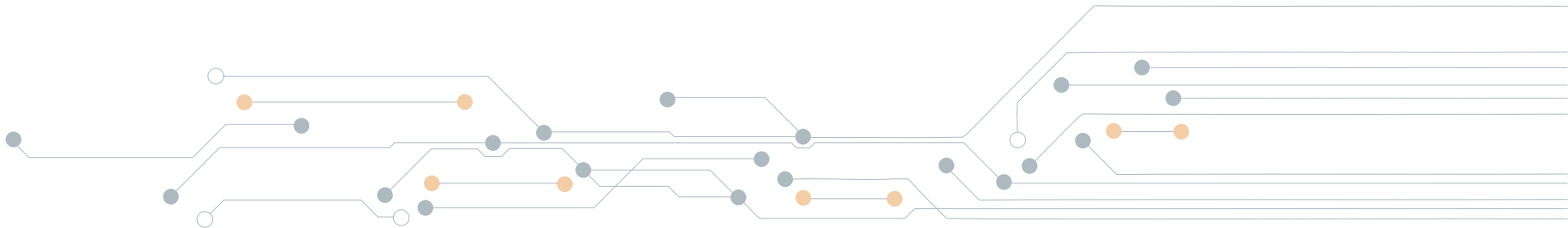
hesidohackeado

Este sitio Web tiene un funcionamiento similar a “haveibeenpwned”. Escanea constantemente todos los Sites donde se publican Pastes y Data Leaks y almacena e indexa toda la información para poder realizar búsquedas sobre ella. En el interfaz de usuario tiene la posibilidad de comprobar si una cuenta de correo se encuentra en alguna de estas fugas de información.

Este sitio Web también cuenta con un API con el cual interaccionar con toda su operativa, pero su uso es mucho más sencillo y solo cuenta con un solo método de invocación en el cual se especifica la cuenta de correo que se quiere comprobar. Como resultado se recibirá toda la información relativa a los Data Leaks y Pastes en la que se encuentra involucrado.

Entre todos esos datos, en muchas ocasiones se encuentran los enlaces de donde obtener el Dump completo de estos Data Leaks y Pastes. Con lo cual se convierte en una fuente muy importante de información.

A diferencia el anterior Site, este no cuenta con una librería Python que nos abstraiga de estas comunicaciones. Por este motivo será necesario crearnos nuestras peticiones manualmente y tratar el JSON devuelto en el resultado:



```
>>> import requests
>>> import json
>>> r = requests.get("https://hesidohackeado.com/api?q=freeman@hotmail.com")
>>> r.json()

{'data': [{'source_lines': 43571259, 'source_network': 'darknet', 'author': 'anon', 'title': 'Last.fm', 'date_leaked':
'2016-09-22T00:00:00+00:00', 'source_url': '#', 'date_created': '2016-09-23T00:00:00+00:00', 'is_vrf': True,
'details': 'https://hesidohackeado.com/leak/anon-lastfm20120322sql', 'emails_count': 37192134, 'source_
provider': 'anon', 'source_size': 7919551421}, {'source_lines': 3231257, 'source_network': 'darknet', 'author':
'anon', 'title': 'dlh.net (main)', 'date_leaked': '2016-09-04T00:00:00+00:00', 'source_url': '#', 'date_created': '2016-
09-12T00:00:00+00:00', 'is_vrf': True, 'details': 'https://hesidohackeado.com/leak/anon-dlhnetmainsha1md5a',
'emails_count': 3227486, 'source_provider': 'anon', 'source_size': 330948209}, {'source_lines': 3110831,
'source_network': 'darknet', 'author': 'anon', 'title': 'dlh.net (forum)', 'date_leaked': '2016-09-04T00:00:00+00:00',
'source_url': '#', 'date_created': '2016-09-12T00:00:00+00:00', 'is_vrf': True, 'details': 'https://hesidohackeado.
com/leak/anon-dlhnetforumvbaugus', 'emails_count': 3108871, 'source_provider': 'anon', 'source_size':
310406861}, {'source_lines': 418128998, 'source_network': 'darknet', 'author': 'anon', 'title': 'Linkedin', 'date_
leaked': '2016-05-16T00:00:00+00:00', 'source_url': '#', 'date_created': '2016-06-18T00:00:00+00:00', 'is_vrf':
True, 'details': 'https://hesidohackeado.com/leak/anon-linkedin2012v2tx', 'emails_count': 159752107, 'source_
provider': 'anon', 'source_size': 22679041728}, {'source_lines': 68680743, 'source_network': 'darknet', 'author':
'anon', 'title': 'Dropbox', 'date_leaked': '2012-07-15T00:00:00+00:00', 'source_url': '#', 'date_created': '2016-
09-05T00:00:00+00:00', 'is_vrf': True, 'details': 'https://hesidohackeado.com/leak/anon-dropbox68m2012txt',
'emails_count': 68435086, 'source_provider': 'anon', 'source_size': 5077855241}

...

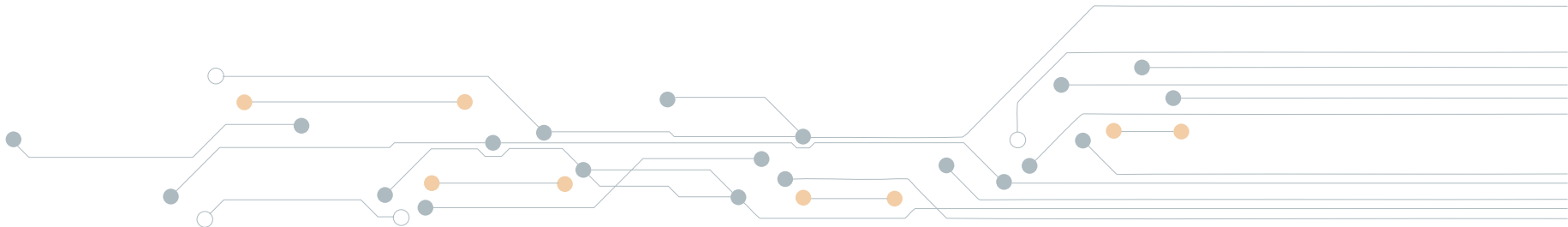
], 'results': 254, 'status': 'found', 'query': 'freeman@hotmail.com'}
```

7. Análisis de Metadatos

La mayoría de ficheros tienen una parte reservada para los datos del propio documento, pero tiene otra parte adicional que contiene información que describe al propio fichero y el contenido del mismo, esta parte son los Metadatos. Estos datos contienen información adicional al fichero como puede ser la fecha de creación y de modificación, autor, herramienta utilizada, geolocalización, etc.

Esta información es habitualmente utilizada por un atacante para obtener información interna de la compañía que no podría obtener por otros medios. Por ejemplo, es posible obtener identificadores internos de usuario, herramienta y versión utilizados que pueden desencadenar en un ataque dirigido, etc.

En Python disponemos de varias librerías que nos permiten automatizar esta tarea.



Extracción de metadatos de documentos PDF

Una de las librerías más versátiles para la extracción de metadatos en documentos PDF es PyPDF2. Esta librería está diseñada para poder realizar múltiples operaciones sobre documentos PDF. Por ejemplo, permite dividirlos en varias partes, juntar varias partes en un solo documento, cifrar y descifrar, etc.

Además de poder realizar transformaciones sobre un documento PDF, también permite obtener los metadatos asociados al mismo.

Esta librería se puede encontrar en el repositorio Github:

<https://github.com/mstamy2/PyPDF2>

Para instalarla, hay que bajar la última versión estable e instalarla de la siguiente forma:

```
# python setup.py install
```

Para mostrar las capacidades de esta librería se va a crear un ejemplo sencillo donde se mostrará por pantalla toda la información que se ha podido recuperar:

```
from PyPDF2 import PdfFileReader

pdf = PdfFileReader(open('file_path.pdf','rb'))
metadata = pdf.getDocumentInfo()
for item in metadata:
    print("--" + item + "=" + metadata[item])
```

Se puede observar como este script devuelve toda la información relativa a los metadatos de ese documento:

```
--/Creator=Introduccion-sin herramientas - Microsoft Word
--/Title=Introduccion-sin herramientas.doc
--/CreationDate=D:20081117130704Z
--/Author=mdhernan
--/ModDate=D:20130320193733+01'00'
--/Producer=Acrobat PDFWriter 5.0 para Windows NT
```

Además de PyPDF2, existen más librerías que permiten recuperar los metadatos de un PDF. Una de ellas es PDFMiner la cual está enteramente enfocada a la extracción de metadatos.



Esta librería se puede encontrar en el Github del proyecto original <https://github.com/euske/pdfminer/>

Pero como no da soporte para Python 3.x se va a instalar un fork que han solventado toda la migración. Este se puede encontrar en: <https://github.com/goulou/pdfminer>

Para su instalación se tendrá que ejecutar el siguiente comando:

```
# python setup.py install
```

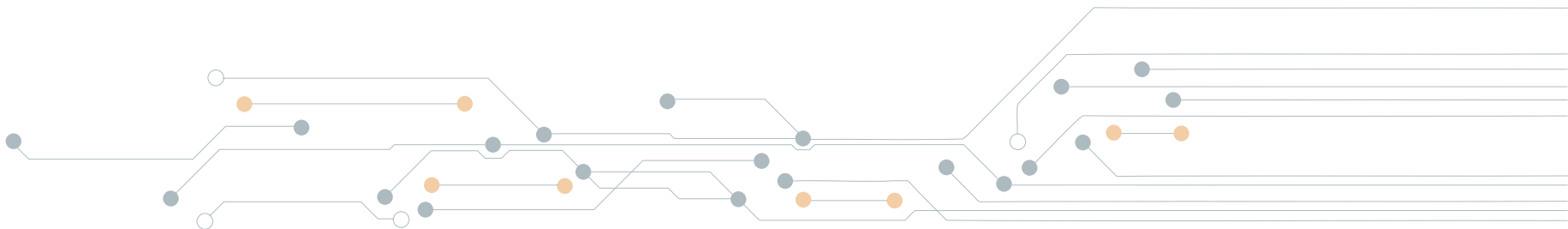
En el siguiente ejemplo se va a extraer los metadatos del mismo documento PDF utilizado en el ejemplo anterior:

```
from pdfminer import *
from pdfminer.pdfdocument import PDFDocument
fp = open('file_path.pdf', 'rb')
parser = PDFParser(fp)
doc = PDFDocument(parser)
print (doc.info)
```

Del mismo modo que el ejercicio anterior, este script devuelve toda la información relativa a los metadatos de ese documento:

```
{['CreationDate': b'D:20081117130704Z', 'Author': b'mdhernan',
'Creator': b'Introduccion-sin herramientas - Microsoft Word',
'Title': b'Introduccion-sin herramientas.doc', 'ModDate':
b'D:20130320193733+01'00'', 'Producer': b'Acrobat PDFWriter
5.0 para Windows NT']}
```

Si se comparan los dos resultados, se puede comprobar cómo ambas librerías devuelven resultados similares.



Extracción de metadatos de Imágenes

Los metadatos en todos los ficheros de imagen y de audio siguen un formato determinado denominado EXIF (Exchange Image File Format). Este estándar es muy importante ya que muchas cámaras de fotos y teléfonos móviles lo cumplen plenamente, e introducen en los ficheros una gran cantidad de información.

Uno de los metadatos más importantes que podemos obtener de las Imágenes es la localización GPS del lugar donde fue tomada. Este dato nos permite realizar un seguimiento de la víctima pudiendo determinar su domicilio o sus caminos habituales con su consiguiente riesgo que esto supone.

Históricamente esta información ha sido indispensable para localizar a asesinos, traficantes, etc, por lo que no es algo que tengamos que pasar por alto.

Para la extracción de esta información se va a utilizar la librería PIL. Esta librería permite la manipulación de las imágenes, pero además incorpora una funcionalidad que nos permite obtener los metadatos asociados a una imagen en concreto.

Esta librería se puede descargar de su página oficial:

<http://www.pythonware.com/products/pil/>

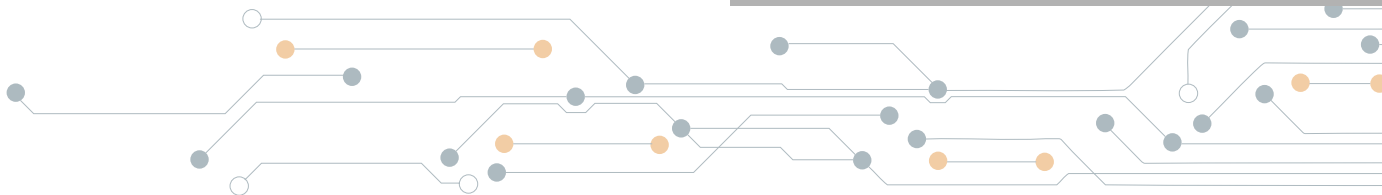
Como este proyecto no está dando soporte para la versión 3.X, se va a utilizar un Fork del mismo denominado Pillow. El cual se puede descargar utilizando PIP:

```
# pip install pillow
```

En el siguiente ejemplo se va a mostrar la forma de visualizar los metadatos de una imagen concreta:

```
from PIL import Image
from PIL.ExifTags import TAGS

exifData = {}
img = Image.open('img_path.jpg')
info = img._getexif()
print(info)
if info:
    for(tag, value) in info.items():
        n = TAGS.get(tag, tag)
        exifData[n] = value
        gps = exifData['GPSInfo']
        if gps:
            print(gps)
```



Telefonica EDUCACIÓN DIGITAL