5 strategies to write unblock-able web scrapers in Python

Adnan Siddiqi Follow
May 5, 2018 · 6 min read

People who read my posts in **scraping series** often contacted me to know how could they write scrapers that don't get blocked. It is very difficult to write a scraper that NEVER gets blocked but yes, you can increase the life of your web scraper by implementing a few strategies. Today I am going to discuss them.

User-Agent

The very first thing you need to take care of is setting the *user-agent*. User Agent is a tool that works on behalf of the user and tells the server about which web browser the user is using for visiting the website. Many websites do not let you view the content if the *user-agent* is not set. If you are using the `requests` library you can then do something like below:

```
headers = {
        'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X
10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87
Safari/537.36',
    }
r = requests.get('example.com',headers=headers)
```

You can get a User-Agent string by writing *what is my user agent* in Google search bar and it will return you your own User-Agent.

Alright, you already have set a User-Agent but how about making it more real? Well, the best way to do is to pick a random User-Agent from a text file, a Python or `list` from the database. **Udger** shares a massive list of UAs w.r.t browsers. For instance, for Chrome it looks like **this** and for Firefox like **this** one. Now let's make a function that will return a random UA that you could use in `requests`

```
import numpy as np

def get_random_ua():
    random_ua = ''
    ua_file = 'ua_file.txt'
    try:
        with open(ua_file) as f:
            lines = f.readlines()
        if len(lines) > 0:
            prng = np.random.RandomState()
            index = prng.permutation(len(lines) - 1)
            idx = np.asarray(index, dtype=np.integer)[0]
            random_proxy = lines[int(idx)]
    except Exception as ex:
        print('Exception in random_ua')
        print(str(ex))
    finally:
        return random_ua
```

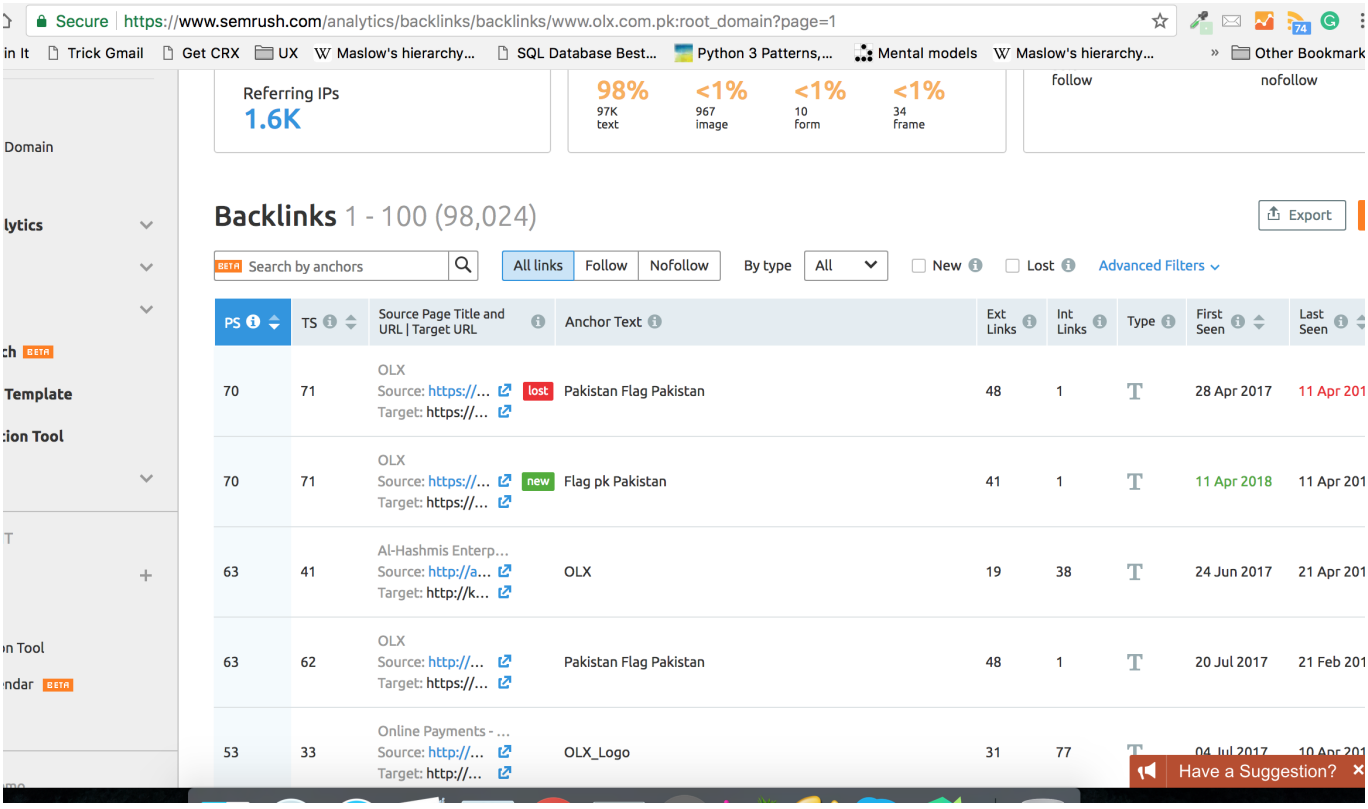
The `ua_file.txt` contains one UA per line from the website I shared above. The function `get_random_ua` will always return a unique UA from the text file. You can now call the function like:

```
user_agent = get_random_ua()
headers = {
        'user-agent': user_agent,
    }
r = requests.get('example.com',headers=headers)
```

Referers

The next thing you would like to set is the referer. The general rule of thumb is that if it's a listing page or home page then you can set Google's main page URL of that country. For instance, if I am scraping `olx.com.pk` then I'd set `https://google.com.pk` instead of `https://google.ca`

If you are going to scrape individual product pages then you can either set relevant category URL in referrer or can find the **backlinks** of the domain you are going to crawl. I usually use SEMRush for the purpose. For the link `https://www.olx.com.pk/furniture-home-decor/categories/` SEMRush returns something like below:



Finding Backlinks via SEMRush

If you click to view the larger version of the image you could see some links that are pointing to my required category. Once you collect all such genuine backlinks you can then use them as referrer by copying the logic inside `get_random_ua()` return random referrer. The `headers` will now become something like given below:

```
headers = {
        'user-agent': user_agent,
        'referer':referer
    }
```

Proxy IPs

I can't emphasize enough for it. Look, if you are into serious business then you have to use multiple proxy IPs to avoid blocking. Majority of websites block crawlers based on the static IP of your server or hosting provider. The sites use intelligent tools to figure out the pattern of a certain IP or a pool of IP and simply block them. This is why it's recommended to buy several IPs, 5o-100 at least to avoid blocking. There are various services available but I am happy with Shaders, now called **OxyLabs**. They are expensive but the quality of service is also good. Make sure when you order multiple IPs, ask for random IPs or at least they don't follow a certain pattern like `1.2.3.4` to `1.2.3.100.` The site admin will simply put a rule to block all IPs belongs to. `1.2.3.*.` It's as simple as that.

If you are using `requests` you can use it like below:

```
r = requests.get('example.com',headers=headers,proxies={'https': proxy_url})
```

And if you are using Selenium and wants to use proxy IPs with Selenium then it's a bit tricky.

```
r = requests.get('example.com',headers=headers,proxies={'https':
proxy_url})

proxy = get_random_proxy().replace('\n', '')
        service_args = [
            '--proxy={0}'.format(proxy),
            '--proxy-type=http',
            '--proxy-auth=user:password'
```

```
        ]
        print('Processing..' + url)
        driver = webdriver.PhantomJS(service_args=service_args)
```

Needless to say that `get_random_proxy()` is the method that returns a unique random proxy like you are getting unique and random UAs and Referer above.

You also can come up with a system where you can set the frequency of an IP to visit the website per day or per hour and if it exceeds it then it put into a cage till the next day. The company I work I devised a system where not only I have set the frequency of the IP but also record which IP is blocked. At the end, I just request the proxy service provider to replace those proxies only. Since this is beyond the scope of this post so I'd not get into details of it.

Request Headers

Things you have implemented so far are good to go but there are still some cunning website that asks you to work more, they look for certain request header entries when you access the page and if the certain header is not found they would either block the content or will spoof the content. It's very easy to mimic the entire request you are going to make on a website. For instance, you are going to access a certain Craigslist URL and wants to know which headers are being requested. Go to Chrome/Firefox and inspect the page being accessed, you should be able to see something like below:

If you click image and view, you will find various entries besides the referee and user-agent. You can either implement all or implement and test one by one. I always set these

entries regardless of whichever website I am going to access. Make sure you don't just copy/paste across sites as these entries often vary one from site to other:

```
headers = {
        'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X
10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87
Safari/537.36',
        'referrer': 'https://google.com',
        'Accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
age/apng,*/*;q=0.8',
        'Accept-Encoding': 'gzip, deflate, br',
        'Accept-Language': 'en-US,en;q=0.9',
        'Pragma': 'no-cache',
    }
```

Delays

It's always good to put some delay between requests. I use `numpy.random.choice()` for that purpose where I pass a list of random numbers I would like to delay the service:

```
delays = [7, 4, 6, 2, 10, 19]
delay = np.random.choice(delays)
time.sleep(delay)
```

You can also use `random.choice` for the same purpose if you are not already using the `numpy` library.

If you are really in hurry then you can execute URLs in parallel which I have explained **here**.

Conclusion

The uncertainty of web scrapers getting block will never go zero but you can always take some steps to avoid it. I discussed a few strategies which you should implement in your web crawler one way or another.

If you know other strategies or tricks then do let me know by sharing in comments. As always, your feedback is most welcome.

Writing scrapers is an interesting journey but you can hit the wall if the site blocks your IP. As an individual, you can't afford expensive proxies either. Scraper API provides you an affordable and easy to use API that will let you scrape websites without any hassle. You do not need to worry about getting blocked because Scraper API by default uses proxies to access websites. On top of it, you do not need to worry about Selenium either since Scraper API provides the facility of a headless browser too. I also have written a post about how to use it.

Click here to signup with my referral link or enter promo code ***SCRAPE156980****, you will get a 10% discount on it. In case you do not get the discount then just let me know via email on my site and I'd sure help you out.*

This article originally published here.

Click here to subscribe to my newsletter for future posts.

Python Web Scraping Data Scraping Data Science