Web scraping and social media scraping – authentication

Jacek Lewkowicz, Dorota Celińska-Kopczyńska

University of Warsaw

March 31, 2019

What will we be working on today?

- A popular way to prevent bots from gathering data from a website is requiring human-like activity
- We will discuss how to provide authentication using Python
- And talk a bit about security while sending information

Convention

- In snippets, we will highlight in violet the areas where you may put your own content
- In commands, the areas in [] are optional
- UNIX-like systems use "/" as the path separator and DOS uses "\". In this presentation the paths will be written in UNIX-like convention if not stated otherwise

Examples of user simulating areas covered today

- Forms: sending your personal data to a website
- Basic working with Cookies

A basic form

- We will start with an example of a form in which a user has to insert their data
- A website processes the information and returns a customized result
- A good example can be found here: `http://pythonscraping.com/pages/files/form.html`

A basic form – solution

- We will use `requests` library

```
import requests

# make a dict with objects that will be inserted
# in the order they have to be used

params = {'firstname': 'Example_Name', 'lastname': 'Example_Surname'}

# use a post() function to send the data to a website

r = requests.post("http://your-site-here.com", data=params)
print(r.text)
```

Submission of different formats

- The data we insert does not have to be in textual form
- With `requests::post()` function we may even submit files

```
import requests

# make a dict with files that are going to be uploaded

files = {'uploadFile': open('/path/to/file.png', 'rb')}

# use a post() function to send the data to a website

r = requests.post("http://your-site-here.com", files=files)
print(r.text)
```

Basic authentication

- Sometimes a part of the website is accessible only after successful login
- In the bot we will have to provide the username and the password
- `requests` library is helpful here

```
import requests
from requests.auth import AuthBase
from requests.auth import HTTPBasicAuth

# provide your username and password

auth = HTTPBasicAuth('username', 'password')
r = requests.post(url="https://your-site-here.com", auth=auth)
print(r.text)
```

Authentication in Scrapy

- You may need to provide different `parse()` functions for the login website and the content

```
from loginform import fill_login_form
from scrapy.http import FormRequest

class UserSpider(scrapy.Spider):
    name = 'name'
    login_user = 'user-name'
    login_password = 'pass'

    login_url = 'https://your-site.com/login/'

    def __init__(self, *args, **kwargs):
        super(UserSpider, self).__init__(*args, **kwargs)
        self.start_urls = ('https://your-site.com')

   # override the start_requests()

    def start_requests(self):
        yield scrapy.Request(self.login_url, self.parse_login)

    def parse_login(self, response):
        data, url, method = fill_login_form(response.url, response.body, self.login_user, self.login_password)
        return FormRequest(url, formdata=dict(data), method=method, callback=self.start_crawl)

    def start_crawl(self, response):
        for url in self.start_urls:
            yield scrapy.Request(url, self.parse, dont_filter=True)

    def parse(self,response):
        # your standard getting data methods here
```

Sending notifications

- Sometimes you want your bot, e.g., to notify yourself that the crawling is done
- Here is the way to send you an email with Python

```
import smtplib
from email.mime.text import MIMEText

msg = MIMEText("The body of an email goes here")
msg['Subject'] = "An Email Alert"
msg['From'] = "put your email here"
msg['To'] = "put an email here"
s = smtplib.SMTP('localhost')
s.send_message(msg)
s.quit()
```

Sending notifications – example

```
def sendMail(subject, body):
    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = email"
    msg['To'] = email"
    s = smtplib.SMTP('localhost')
    s.send_message(msg)
    s.quit()

bsObj = BeautifulSoup(urlopen("https://isitchristmas.com/"))

if (bsObj.find("a", "id":"answer").attrs['title'] == "NO"):
    print(Sorry, it is not Christmas yet ")
else:
    sendMail(Yes, it's Christmas! ")
time.sleep(3400)
```

What is a cookie?

- Cookies are used to allow HTTP servers to re-recognize clients
- HTTP itself is a stateless protocol that treats each exchange of request and response as though it was the first
- Cookies may be problematic for scraping

Keeping track of cookies

- Send the login parameters, retrieve the cookies, print the results for verification, send them to the page by setting the cookies argument
- Again we will use `requests` library here

```
import requests
params = {'username': 'name', 'password': 'password'}
r = requests.post("https://your-site-here.com", params)
print("Cookie is set to:")
print(r.cookies.get_dict())
print("-----------")
print("Going to profile page...")
r = requests.get("full_url_there-subpage", cookies=r.cookies)
print(r.text)
```

Cookies – problems

- Webpages may modify cookies without warning
- In this case and other complex issues, `session()` function may handle the problem

```
import requests
Session=requests.Session()
params = {'username': 'name', 'password': 'password'}
s = session.post("https://your-site-here.com", params)
print("Cookie is set to:")
print(s.cookies.get_dict())
print("-----------")
print("Going to profile page...")
r = session.get("full_url_there-subpage", cookies=r.cookies)
print(r.text)
```

Cookies – Scrapy

- Scrapy manages cookies automatically
- Access via `CookiesMiddleware` – enables working with sites that require cookies, such as those that use sessions
- It keeps track of cookies sent by web servers, and send them back on subsequent requests (from that spider), just like web browsers do.
- For supporting multiple cookie sessions per spider you may use `cookiejar` Requests meta key
- To see the cookies being sent and received from Scrapy enable the `COOKIES_DEBUG` setting

HTTPS protocol

- A very popular secure transfer protocol
- In HTTPS the communication protocol is encrypted by Transport Layer Security (TLS), or formerly, its predecessor, Secure Sockets Layer (SSL)
- Aimed at protection of the privacy and integrity of the exchanged data
- Prevents *man-in-the-middle* attacks

Typical associations

- While thinking about Internet Security the first thought is usually about hackers...
- Those talented, mysterious guys from Anonymous or related groups *(pop culture also does not help here)*

Source: http://pixabay.com/en/hacker-attack-mask-internet-2883632/

Sad truth

- A more probable scenario is that you will know your “hacker”
- ... because you will just handle your data on a silver platter

Source: https://pl-pl.facebook.com/Karniaki-na-Fejsie-1460797344184675//

- Jokes aside: protect your information!
- Remember the snippets about authentication?...

Secure way to provide information – tips

- **Avoid putting your logins and passwords in raw form**
- Instead you may use, e.g., **environment variables**
- Environment variable is a dynamic-names value that can affect the way running processes will behave on a computer
- Its value is accessible only in a given session

Environment variables – usage

- Setting the variable:
  - Unix-like: `export NAME='name'`
  - Windows: `set NAME='name'`
- Sending environment variables to your spider

```
import os

login_user = os.environ['NAME']
login_password = os.environ['PASS']

# you may insert those variables into your dict
params = {'login': login_user, 'password': login_password}
```

Environment variables – example

- Assume that there are two users who will use the same scraper:
  - `titu-chicken` with password `'giveMeYourPass'`
  - `marnie-cow` with password `'Nevermore!'`
- For some reason they do not want to use shared account for scraping

Environment variables – example

- Each user will independently set the variables via command line:
  - `export LOGIN='titu-chicken';export PASS='giveMeYourPass'`
  - `export LOGIN='marnie-cow';export PASS='Nevermore!'`
- But their scraper code will look the same!
- ... and if they do not save the commands in the history, after closing the session nobody will be able to access their data

```
import os

login_user = os.environ['LOGIN']
login_password = os.environ['PASS']
```

Homework

- Find out how to install Splash on your hardware
- Unix-like

  `http://splash.readthedocs.io/en/stable/install.html`

- Windows

  `https://www.quora.com/How-do-I-install-Splash-on-Windows-and-get-it-to-work-with-Scrapy`