13.7: SECURITY AND API USAGE

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

It is quite common that you need some kind of "API key" to make use of a vendor's API. The general idea is that they want to know who is using their services and how much each user is using. Perhaps they have free and pay tiers of their services or have a policy that limits the number of requests that a single individual can make during a particular time period.

Sometimes once you get your API key, you simply include the key as part of POST data or perhaps as a parameter on the URL when calling the API.

Other times, the vendor wants increased assurance of the source of the requests and so they add expect you to send cryptographically signed messages using shared keys and secrets. A very common technology that is used to sign requests over the Internet is called *OAuth*. You can read more about the OAuth protocol at www.oauth.net.

As the Twitter API became increasingly valuable, Twitter went from an open and public API to an API that required the use of OAuth signatures on each API request. Thankfully there are still a number of convenient and free OAuth libraries so you can avoid writing an OAuth implementation from scratch by reading the specification. These libraries are of varying complexity and have varying degrees of richness. The OAuth web site has information about various OAuth libraries.

For this next sample program we will download the files *twurl.py*, *hidden.py*, *oauth.py*, and *twitter1.py* from www.py4e.com/code and put them all in a folder on your computer.

To make use of these programs you will need to have a Twitter account, and authorize your Python code as an application, set up a key, secret, token and token secret. You will edit the file *hidden.py* and put these four strings into the appropriate variables in the file:

```
# Keep this file separate

# https://apps.twitter.com/
# Create new App and get the four strings

def oauth():
    return {"consumer_key": "h7Lu...Ng",
            "consumer_secret": "dNKenAC3New...mmn7Q",
            "token_key": "10185562-eibxCp9n2...P4GEQQOSGI",
            "token_secret": "H0ycCFemmC4wyf1...qoIpBo"}

# Code: http://www.py4e.com/code3/hidden.py
```

The Twitter web service are accessed using a URL like this:

https://api.twitter.com/1.1/statuses/user_timeline.json

But once all of the security information has been added, the URL will look more like:

```
https://api.twitter.com/1.1/statuses/user_timeline.json?count=2
&oauth_version=1.0&oauth_token=101...SGI&screen_name=drchuck
&oauth_nonce=09239679&oauth_timestamp=1380395644
&oauth_signature=rLK...BoD&oauth_consumer_key=h7Lu...GNg
&oauth_signature_method=HMAC-SHA1
```

You can read the OAuth specification if you want to know more about the meaning of the various parameters that are added to meet the security requirements of OAuth.

For the programs we run with Twitter, we hide all the complexity in the files *oauth.py* and *twurl.py*. We simply set the secrets in *hidden.py* and then send the desired URL to the *twurl.augment()* function and the library code adds all the necessary parameters to

the URL for us.

This program retrieves the timeline for a particular Twitter user and returns it to us in JSON format in a string. We simply print the first 250 characters of the string:

```
import urllib.request, urllib.parse, urllib.error
import twurl
import ssl

# https://apps.twitter.com/
# Create App and get the four strings, put them in hidden.py

TWITTER_URL = 'https://api.twitter.com/1.1/statuses/user_timeline.json'

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    print('')
    acct = input('Enter Twitter Account:')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '2'})
    print('Retrieving', url)
    connection = urllib.request.urlopen(url, context=ctx)
    data = connection.read().decode()
    print(data[:250])
    headers = dict(connection.getheaders())
    # print headers
    print('Remaining', headers['x-rate-limit-remaining'])

# Code: http://www.py4e.com/code3/twitter1.py
```

When the program runs it produces the following output:

```
Enter Twitter Account:drchuck
Retrieving https://api.twitter.com/1.1/ ...
[{"created_at":"Sat Sep 28 17:30:25 +0000 2013","
id":384007200990982144,"id_str":"384007200990982144",
"text":"RT @fixpert: See how the Dutch handle traffic
intersections: http:\/\/t.co\/tIiVWtEhj4\n#brilliant",
"source":"web","truncated":false,"in_rep
Remaining 178

Enter Twitter Account:fixpert
Retrieving https://api.twitter.com/1.1/ ...
[{"created_at":"Sat Sep 28 18:03:56 +0000 2013",
"id":384015634108919808,"id_str":"384015634108919808",
"text":"3 months after my freak bocce ball accident,
my wedding ring fits again! :)\n\nhttps:\/\/t.co\/2XmHPx7kgX",
"source":"web","truncated":false,
Remaining 177

Enter Twitter Account:
```

Along with the returned timeline data, Twitter also returns metadata about the request in the HTTP response headers. One header in particular, *x-rate-limit-remaining*, informs us how many more requests we can make before we will be shut off for a short time period. You can see that our remaining retrievals drop by one each time we make a request to the API.

In the following example, we retrieve a user's Twitter friends, parse the returned JSON, and extract some of the information about the friends. We also dump the JSON after parsing and "pretty-print" it with an indent of four characters to allow us to pore through the data when we want to extract more fields.

```
import urllib.request, urllib.parse, urllib.error
import twurl
import json
import ssl

# https://apps.twitter.com/
# Create App and get the four strings, put them in hidden.py

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    print('')
    acct = input('Enter Twitter Account:')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '5'})
    print('Retrieving', url)
    connection = urllib.request.urlopen(url, context=ctx)
    data = connection.read().decode()

    js = json.loads(data)
    print(json.dumps(js, indent=2))

    headers = dict(connection.getheaders())
    print('Remaining', headers['x-rate-limit-remaining'])

    for u in js['users']:
        print(u['screen_name'])
        if 'status' not in u:
            print('   * No status found')
            continue
        s = u['status']['text']
        print('  ', s[:50])

# Code: http://www.py4e.com/code3/twitter2.py
```

Since the JSON becomes a set of nested Python lists and dictionaries, we can use a combination of the index operation and `for` loops to wander through the returned data structures with very little Python code.

The output of the program looks as follows (some of the data items are shortened to fit on the page):

```
Enter Twitter Account:drchuck
Retrieving https://api.twitter.com/1.1/friends ...
Remaining 14
```

```
{
    "next_cursor": 1444171224491980205,
    "users": [
        {
            "id": 662433,
            "followers_count": 28725,
            "status": {
                "text": "@jazzychad I just bought one .__.",
                "created_at": "Fri Sep 20 08:36:34 +0000 2013",
                "retweeted": false,
            },
            "location": "San Francisco, California",
            "screen_name": "leahculver",
            "name": "Leah Culver",
        },
        {
            "id": 40426722,
            "followers_count": 2635,
            "status": {
                "text": "RT @WSJ: Big employers like Google ...",
                "created_at": "Sat Sep 28 19:36:37 +0000 2013",
            },
            "location": "Victoria Canada",
            "screen_name": "_valeriei",
            "name": "Valerie Irvine",
    ],
    "next_cursor_str": "1444171224491980205"
}
```


```
leahculver
   @jazzychad I just bought one .__.
_valeriei
   RT @WSJ: Big employers like Google, AT&amp;amp;T are h
ericbollens
   RT @lukew: sneak peek: my LONG take on the good &a
halherzog
   Learning Objects is 10. We had a cake with the LO,
scweeker
   @DeviceLabDC love it! Now where so I get that "etc

Enter Twitter Account:
```

The last bit of the output is where we see the for loop reading the five most recent "friends" of the *drchuck* Twitter account and printing the most recent status for each friend. There is a great deal more data available in the returned JSON. If you look in the output of the program, you can also see that the "find the friends" of a particular account has a different rate limitation than the number of timeline queries we are allowed to run per time period.

These secure API keys allow Twitter to have solid confidence that they know who is using their API and data and at what level. The rate-limiting approach allows us to do simple, personal data retrievals but does not allow us to build a product that pulls data from their API millions of times per day.