Virginia Tech ■ ECE 5480 ■ Fall 2018

Project 6

Before starting this first homework assignment, please be sure that you have completed all of the following activities.

☐ ·Review the Graduate Honor System at https://graduateschool.vt.edu/academics/expectations/graduate-honor-system.html. Review the Graduate Honor System Constitution, especially Articles I (Sections 1, 2, and 3), V, VI, VII, VIII, and IX.

Please note the following.

• Solutions must be clear and presented in the order assigned. Solutions must show work needed, as appropriate, to derive your answers. Written answers should be concise, but sufficiently complete to answer the question. Neat hand drawings, where needed, are acceptable. Your final solution for each problem must be easily identified.

• At the top of the first page, include: your name (as recorded by the university); your email address; and the assignment name (e.g., "ECE 5480, Project 6"). Do not include your Virginia Tech ID number or your social security number.

• Homework must be submitted as a PDF (.pdf) file with the file name *lastname_firstname*_PROJ6.pdf, where *lastname* is your last or family name and *firstname* is your first or given name, along with requested files.

• Submit your assignment using the Assignments area of the class website.

INTRODUCTION

Especially because there is often so much data, being able to display security data visually can help the security or network admins, managers, and users see important patterns or make important decisions quickly. Also, graphics can be created from the analysis of security data that can similarly convey useful information. While there is a lot more to visualization and the graphical display of information than we can cover in this course, in this project we are going to explore the visualization of security data with Python.

RECOMMENDED IP BLOCK LIST

The Internet Storm Center at the SANS Technology Institute (https://isc.sans.edu/) monitors malicious activity on the Internet, including suspicious domains. One of the datasets they collect is a block list of suspicious malware domains: https://isc.sans.edu/block.txt. This block list is a text file that "summarizes the top 20 attacking class C (/24) subnets over the last three days." This list is updated periodically. Thus, our Python code should pull the list from the given website rather than access an out of date file previously downloaded.

An example IP address line is:

```
77.72.82.0 77.72.82.255     24    9012  NETUP-AS , RU    aospan@netup.ru
```

For the purposes of this project, use the first IP address in each IP address line.

IP ADDRESS GEOLOCATION

Based on public repositories of information, such as the geographic address of an IP address's ISP, it is possible to estimate an approximate geographic location for that IP address. This information however is not very reliable — sometimes only the nearest city, state, or country can be approximated. As an example, *please read* this article on the limitations of IP address geolocation:

```
https://splinternews.com/how-an-internet-mapping-glitch-turned-a-random-
kansas-f-1793856052
```

However, this process is often used to get a rough idea where malicious servers are located, for example in the event of a botnet attack.

For this project we are going to use a free service at `http://www.ipdata.co` to convert ip addresses to latitude/longitude coordinates. Note that you must sign up for a free API key—the free tier allows 1500 requests a day or 45,000 requests a month--> it is up to you to make sure you do not violate these restrictions. Please run `test_ipdata.py` for a demonstration (geographic results are returned in the JSON (`https://www.json.org/`)) format.

GEOPY

Unfortunately Google now requires a credit card registration to use their Google Maps API. With such an account, for example, we could query the SANS block list to generate blocked IP addresses, use ipdata.co to convert IP addresses to LAT/LONG coordinates, then plot those coordinates on a Google Map using their Google Maps API.

But for now, we will use the Python `geopy` library to convert our LAT/LONG coordinates to physical addresses. While installation of the `geopy` library will be likely with your Python installation (see examples in the test code), you can run `test_geopy.py` for an example.

REVERSE GEOLOCATION PYTHON CODE

For this project with our Python code we are going to:

1.) Fetch the SANS list of recommended IP addresses to block (`https://isc.sans.edu/block.txt`),

2.) Convert those IP addresses to geographic coordinates using `http://www.ipdata.co`

3.) Convert those geographic coordinates to physical addresses using `geopy`.

TODO:

You should do 1, 2, and 3 with one python script. Turn in your Python script `.py`, the block.txt file you used, and a text file containing the converted physical addresses, one for each ip address in your block.txt file.

HINTS:

1.) For simplicity, I recommend using the Python library requests (like we used in Project 1) to download the `block.txt` file from SANS, then store that as a file on your computer. Then read that file (as we did in Project 2) with something like:

```
lines = [line.rstrip() for line in fp.readlines()]
```

Each line of `block.txt` is now a separate entry in the `lines list` variable.

2.) Similar to how we parsed strings in Project 4, you can extract out the first IP address in each line that starts with an IP address. How to tell if a line starts with an IP address? You could create a variable:

```
nums = {‘0','1','2','3','4','5','6','7','8','9'}
```

Then test to see if the first character of each line starts with a string number like:

```
if (line[0] in nums):

        ip_addresses.append(line.split(“\t",2)[0])
```

Here I’m adding the first IP address to a `list` variable (the columns in the `block.txt` are tabs, not spaces).

Next, the basic idea to convert each ip address to a lat/long set of coordinates is to iterate through each ip in your list of ip addresses. Create a string:

```
get_url = “https://api.ipdata.co/" + ip + “?api-key=YOUR_KEY”
```

Then call:

```
request = urllib.request.Request(get_url, headers=headers)

response_body = urllib.request.urlopen(request).read()
```

as in the example code.

Here, `response_body` is the json information in bytes. To convert this to a `dict` we can use the json library:

```
import json

foo = json.loads(response_body)
```

To get the latitude value (numeric): `foo[‘latitude’]`. To get the longitude value (numeric): `foo[‘longitude’].`

There are many ways you could store this information. One way is to create a `list` of `lists`:

```
lat_long.append(foo['latitude'])

lat_long.append(foo['longitude'])

lat_longs.append(lat_long)
```

3) To convert the lat/longs to physical addresses, you could then iterate through your `lat_longs` list:

```
for lat_long in lat_longs:

    lat_long_string = str(lat_long[0]) + ", " + str(lat_long[1])

    location = geolocator.reverse(lat_long_string)

    print(location.address)
```