Assignment 1: Explore CAIDA Spoofer Public Datasets (16pts)

Due on Monday, January 23rd 11:59pm

1 Introduction

The goal of this assignment is to become familiar with the API provided by CAIDA for accessing Spoofer Project data, which publicly reports the outcomes of crowd-sourced Source Address Validation (SAV) deployment tests. You will write code to access the API and conduct simple analyses.

1.1 API

The API itself is available at `https://api.spoofer.caida.org/`, and returns JSON. You can try the API out in your browser –

1. click the "GET /sessions" link that allows you to retrieve a collection of Session resources,
2. click the "Try it out" button at the top right of the page. The page will then allow you to enter values into one or more of the fields.
3. Let's find all entries after December 2022, for ASN 7377 (UCSD).
   (a) enter "2022-12-01" in the "timestamp[after]" field
   (b) enter 7377 into the ASN field.
   (c) click the "Execute" button at the bottom of the page

   when I searched for UCSD tests after December 2022, the first entry I saw in the Response body was for session 1489429. You can filter the responses in other ways, for example, by entering specific countries in the "country" field.

Examining the fields for session 1489429:

```
{
  "@id": "/sessions/1489429",
  "@type": "Session",
  "session": 1489429,
  "client4": "69.196.36.0/24",
  "client6": "2607:f720:f00::/40",
  "timestamp": "2022-12-05T21:01:30+00:00",
  "asn4": "7377",
  "asn6": "7377",
  "country": "usa",
  "nat4": true,
  "nat6": false,
  "privatespoof": "blocked",
  "routedspoof": "blocked",
  "privatespoof6": "unknown",
  "routedspoof6": "unknown"
},
```


you will see

1. the client's IPv4 address aggregated to a /24 in the "client4" field,
2. the client's IPv6 address aggregated to a /40 in the "client6" field,
3. a fine-grained timestamp that identifies when the client conducted the test in the "timestamp" field,
4. that the ASNs for both IPv4 ("asn4" field) and IPv6 ("asn6" field) was 7377,
5. that the client was geolocated to be within the U.S. ("country":"usa"),
6. that the client was behind a NAT device in IPv4 ("nat4":true), but not in IPv6 ("nat6":false),
7. that packets with private or routed addresses were blocked in IPv4 ("privatespoof": "blocked", "routedspoof": "blocked"),
8. that tests with private and routed addresses in IPv6 were inconclusive ("privatespoof6": "unknown", "routedspoof6": "unknown").

The other values that you can expect to see for the `routedspoof` and `privatespoof` fields are "received" (our server received the spoofed packet, indicating that the client's network did not filter our packet), "rewritten" (our server received the spoofed packet, but its source address had been rewritten and so we cannot determine if the network blocked the spoofed packet), and "unknown' (our server did not receive any packets, spoofed or unspoofed, so we could not make an inference). If the client tested IPv4 but did *not* test IPv6 (because the system did not have an IPv6 address) then the IPv6 fields will be null; similarly, if the client tested IPv6 but not IPv4, then the IPv4 fields will be "null".

The API is documented at `https://www.caida.org/projects/spoofer/data-api/`, including example code. (Warning: example code in Perl! you will write Python.) Importantly, the API returns at most 30 results per call, while many queries you make will receive more than 30 results. You must obtain subsequent results using the pagination functionality documented at the website, and demonstrated in the code at the API documentation.

For example, the query for ASN 7018 after January 1, 2022 returns 1055 items in total across 36 pages –


```
"hydra:totalItems": 1055,
"hydra:view": {
  "@id": "/sessions?timestamp%5Bafter%5D=2022-01-01&asn=7018&page=1",
  "@type": "hydra:PartialCollectionView",
  "hydra:first": "/sessions?timestamp%5Bafter%5D=2022-01-01&asn=7018&page=1",
  "hydra:last": "/sessions?timestamp%5Bafter%5D=2022-01-01&asn=7018&page=36",
  "hydra:next": "/sessions?timestamp%5Bafter%5D=2022-01-01&asn=7018&page=2"
},
```


Because there will be many pages of results for your queries as part of this assignment, you will need to programmatically query the API. You visit each page in sequence by querying the URL provided by the API in the "hydra:next" field.

1. some of the queries might take a long time to run, depending on your Internet connection and how busy the API server is.

2. you should save the output of the API to disk and then analyze the output in a separate program, as this will allow you to quickly re-process the raw data if you find a bug in your analysis code.

3. you can use whatever HTTP querying approach you like (e.g., "pycurl" or "requests" in python).

You are expected to write your own Python code that automatically queries the API and processes the data to answer each question, and submit that code with your answers.

1.2 Due Date

Monday, January 23rd 11:59pm

1.3 Submission Instructions

There will be 4 tasks in this assignment, and you will complete each task in a separate `python` file. In total you will submit 4 files on Gradescope.: `pa1q1.py`, `pa1q2.py`, `pa1q3.py`, `pa1q4.py`. Your scores will be immediately available upon submission and you have unlimited submission attempts until due date.

2 Questions / Tasks

2.1 Task 1: Query the API and cache the API response (1pt)

Your first task is to query the API with the following paremeters:

```
timestamp[before]: 2023-01-03
timestamp[after]: 2023-01-01
country: usa
```

This will return all results from 2023-01-01 to 2023-01-02 in the US. There will be multiple pages of results. You will then store the part of the API output into a cache file in the following steps:

(1) Look for the `hydra:member` field in the API response, which contains a `list` of `dict` objects. Each `dict` object is similar to the example in §1.1 in the response body for session 1489429.

(2) Combine all the `lists` obtained in each page of the API response into a single `list`.

(3) Write the list of `dict` objects into a json file `output.json` in the **same directory as your python file**. The ordering of `dict` objects does not matter. You can simply use the `json.dump()` function from the `json` package for this step.

You will complete this task in `pa1q1.py`. Your code should run with the following command:

`python3 pa1q1.py`

and generate `output.json` in the same directory as your `python` file.

2.2 Task 2: Spoofable IPv4 blocks by country (5pts)

Complete the subsequent tasks using the provided API cache file `spoofer_cache.json`. The cache contains API results between 2022-01-01 and 2023-01-01 for 5 countries: USA(usa), Brazil (bra), New Zealand (nzl), Australia (aus), and India (ind).

In this task, you will output, for a given country, a list of routable IPv4 networds (IPv4 prefixes) that allow spoofing for that country. The cache file will provide you spoofer test results collected over a year. You will first remove all tests whose "routedspoof" result is "unknown". Then you will only count the most recent test for each unique IPv4 /24 block reported in your results, and look for tests whose "routedspoof" result is "received".

You will complete the task in the following steps:

(1) Load the spoofer cache file `spoofer_cache.json`. The autograder will place the cache file in the same directory it places your `python` file. You can download the cache file here: `https://cseweb.ucsd.edu/classes/wi23/cse291-e/pa1/spoofer_cache.json`

(2) Read an input file `input.txt`, which contains a **single** country code (ISO-3166). You will be tested on 5 countries: USA(usa), Brazil (bra), New Zealand (nzl), Australia (aus), and India (ind). Example `input.txt`:

```
usa
```

(3) Process the spoofer test results according to the instructions above. Find all the IPv4 blocks for the given country whose "routedspoof" result is "received".

(4) Print the IPv4 prefixes, each separated by a newline, into `output.txt`. The ordering of prefixes does not matter. Example `output.txt`:

```
1.0.0.0/24
44.2.0.0/24
```

You will complete this task in `pa1q2.py`. Your code should run with the following command:

`python3 pa1q2.py`

and generate `output.json` in the same directory as your `python` file.

2.3 Task 3: Spoofable IPv4 blocks of MANRS networks (5pts)

Complete the subsequent tasks using the provided API cache file `spoofer_cache.json` from Task 2.

In this task, you will output, for a given country, a list of routable IPv4 networks (prefixes) that allow spoofing and are originated by ASes that claim they are performing SAV. For this, we have provided you a Comma Separated Values (CSV) file `manrs.csv` that, for each line in the file, identifies an organization in the first column, their ASes separated by a semi-colon in the third column, and whether or not they commit to "Action 2: Anti-Spoofing" (fifth column), i.e., source address validation (SAV).

You will complete the task in the following steps:

(1) Load the spoofer cache file `spoofer_cache.json`. The autograder will place the cache file in the same directory it places your `python` file. You can download the cache file here: `https://cseweb.ucsd.edu/classes/wi23/cse291-e/pa1/spoofer_cache.json`

(2) Read the MANRS participants file `manrs.csv`, which the autograder will place in the same directory it places your `python` file. You can download the cache file here: `https://cseweb.ucsd.edu/classes/wi23/cse291-e/pa1/manrs.csv` Find the AS numbers of networks who have "Yes" in the "Action 2: Anti-Spoofing" column (column 5), i.e., claim they are implementing SAV.

(3) Read an input file `input.txt`, which contains a **single** country code (ISO-3166). You will be tested on 5 countries: USA(usa), Brazil (bra), New Zealand (nzl), Australia (aus), and India (ind). This step is the same as step (2) in Task 2.

(3) Process the spoofer test results according to the instructions provided in Task 2 (you can re-use the code for Task 2). Then keep only the test results whose `asn4` field matches the AS numbers from step (2). Find all the IPv4 blocks for the given country whose "routedspoof" result is "received".

(4) Print the IPv4 prefixes, each separated by a newline, into `output.txt`. The ordering of prefixes does not matter.

You will complete this task in `pa1q3.py`. Your code should run with the following command:

`python3 pa1q3.py`

and generate `output.json` in the same directory as your `python` file.

2.4 Task 4: Spoofable IPv6 blocks by country (5pts)

Complete the subsequent tasks using the provided API cache file `spoofer_cache.json`.

This task is similar to task 2, the only difference is you will analyze the IPv6 blocks instead of IPv4 blocks.

In this task, you will output, for a given country, a list of routable IPv6 networks (prefixes) that allow spoofing. The cache file will provide you spoofer test results collected over a year. You will first remove all tests whose "routedspoof6" result is "unknown". Then you will only count the most recent test for each unique IPv6 /40 block reported in your results, and look for tests whose "routedspoof6" result is "received".

You will complete the task in the following steps:

(1) Load the spoofer cache file `spoofer_cache.json`. The autograder will place the cache file in the same directory it places your `python` file. You can download the cache file here: `https://cseweb.ucsd.edu/classes/wi23/cse291-e/pa1/spoofer_cache.json`

(2) Read an input file `input.txt`, which contains a **single** country code (ISO-3166). You will be tested on 5 countries: USA(usa), Brazil (bra), New Zealand (nzl), Australia (aus), and India (ind).

(3) Process the spoofer test results according to the instructions above. Find all the IPv6 blocks for the given country whose "routedspoof6" result is "received".

(4) Print the IPv6 prefixes, each separated by a newline, into `output.txt`. The ordering of prefixes does not matter. Example `output.txt`:

```
2601:152:4000::/40
2601:41:4100::/40
```

You will complete this task in `pa1q4.py`. Your code should run with the following command:

`python3 pa1q4.py`

and generate `output.json` in the same directory as your `python` file.