Project Introduction
We here at Malwarebytes deal with suspicious activity on a daily basis. One of the hottest new ideas was to build a suspicious login detection system to help us thwart the bad guys, and defend our customers on another level.. This is why we set out to build SUSS, suspicious user session system, our next generation early warning detection system. Imagine if user credentials were used to login from New York at 09:00 and then again at 09:15, but this time from London. When a user attempts to login we send telemetry to the backend, then by looking up the IP address in a GeoIP database, we can determine if this login pair is suspicious...how did they travel from New York to London in 15 minutes!? This can be a strong indicator of credentials being compromised.

Key Requirements:

- SUSS will have an API that accepts JSON payload with the following fields:
    - UUID identifying login attempt
    - username
    - UNIX epoch timestamp in seconds
    - IPv4 Address
- JSON response body (see below for details)
- data structures or existing libraries for fast mapping from IP address to latitude and longitude geolocation lookups provided by a local copy of the MaxMind GeoIP database
    - Download and use the MaxMind City database for IP geolocation: https://dev.maxmind.com/geoip/geoip2/geolite2/
- SUSS threshold is 500 miles per hour, in other words, any subsequent login attempts that would require a human to exceed 500 MPH results in a suspicious login attempt
    - Assume the Earth is a perfect sphere where distances should be calculated via Haversine formula
- README (how to compile/test/run your project)
- Use SQLite or H2 as your database
- Use any external libraries or dependencies, but please document and cite (what & why)

Key Considerations:

- Runtime performance for IP Geolocation lookups and the travel distance computations
- Handling **out-of-order arrival time** of login attempts
- Maxmind GeoIP Database includes a radius (in kilometers) around each lat/long pair to express uncertainty

- How would you handle IPs flagged as VPN or Tor Exit nodes? ****Note** this data is not included in the free GeoIP database, thus not a required part of the solution, but would be interesting to consider how it might affect suspicious result

Nice to have:

- Docker build & runtime image

Expected Request:
Your API should accept a POST request over HTTP that provides information about the time and IP that a user is connecting from. A sample cURL request might look like:

```
curl -X POST \
  http://localhost:5000/v1/event \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "bob",
    "unix_timestamp": 1514764800,
    "event_uuid": "85ad929a-db03-4bf4-9541-8f728fa12e42",
    "ip_address": "206.81.252.6"
}'
```

Expected Response:
In response to the above POST request, your API should return JSON payload informing the client about the geo information for the current IP access event as well as the nearest previous and subsequent events (if they exist). For the preceding/subsequent events, it should also include a field **suspiciousTravel** indicating whether travel to/from that geo is suspicious or not, as well as the **speed**.

```
{
        "currentGeo": {
                "lat": 39.1702,
                "lon": -76.8538,
                "radius": 20
        },
        "travelToCurrentGeoSuspicious": true,
        "travelFromCurrentGeoSuspicious": false,
        "precedingIpAccess": {
                "lat": 30.3764,
                "lon": -97.7078,
                "radius": 5,
                "speed": 55,
                "ip": "24.242.71.20",
                "timestamp": 1514764800
        },
        "subsequentIpAccess": {
                "lat": 34.0494,
                "lon": -118.2641,
                "radius": 200,
                "speed": 27600,
                "ip": "91.207.175.104",
                "timestamp": 1514851200
        }
}
```