API Exploration Process

1. **Initial Observations:**

    - I started with the v1 version of the API. Initially, I noticed that it returned names starting with the query string and was limited to only 10 results per request.

    - Since the URL contained "v1," I assumed there might be multiple versions. Upon further investigation, I found v2 and v3.

    - I carried out an Nmap scan to pinpoint the open ports. I discovered that both port 22 (SSH) and port 8000 (API) were open. On **port 8000, I** used `nmap -A` to discover the **backend server** was running **Uvicorn** (`nmap -sV -A -p 8000 {url}).` On port 22, it required a private key, so I stopped.
    - Used **SQLMap** to check for **SQL injection vulnerabilities**, assuming SQL might be used in the backend, but found **no vulnerabilities**.
    - I hypothesized that a backend function was dynamically generating results based on queries.

    - We attempted to reconstruct the backend function using reverse engineering techniques, but found no conclusive patterns. If a function was involved, it was either too complex or not easily reversible.

2. **Exploring Endpoints:**

    - The documentation suggested trying different endpoints, so I used **dirb** to enumerate them. This led me to discover the **/hint**, **/help,** and **/solution** endpoints.

3. **Requesting a Hint:**

    - Sending a **GET** request to the hint endpoint returned a message indicating that I needed a "good way to ask for a hint."

    - I experimented with different types of requests and found that a **POST** request successfully provided a hint.

4. **Finding the Query Parameter for Increasing Results:**

- The hint suggested looking for a query parameter that would increase the number of names returned per request.

- I initially tried generic parameters like `count` and `cnt`, but they did not work.

- After seeking suggestions from ChatGPT, I shifted my focus to parameters related to requests. Eventually, I discovered that **`max_requests`** was the correct parameter—this was a tough guess.

5. **Extracting Names from Different API Versions:**

    - I carefully analyzed the responses and identified the characters involved in the names.

      ```
      v1_char_set: a to z

      v2_char_set: 0 to 9 and a to z

      v3_char_set: space, +, -, .,  0 to 9and a to z
      ```

    - Querying with a single character did not return the full list due to the **50-result limit**.

    - To bypass this, I generated all **two-character combinations** and queried the API to retrieve names.

6. **Handling the Result Limit Recursively:**

    - Some two-character queries returned more than 50 results. In such cases, I recursively queried using **three-character combinations**.

    - If three-character combinations still exceeded the limit, I extended to **four-character combinations** where the first three characters remained fixed.

7. **Adapting to API Version Differences:**

    - v1 and v2 returned names ranging from **2 to 10 characters**, whereas v3 included names **from 1 to 10 characters**.

    - For v3, I started with **single-character queries** and followed the same recursive approach to extract all names.

8. **Rate Limiting Challenges:**
    - Each API version imposed rate limits:
        - **v1:** 100 requests
        - **v2:** 50 requests
        - **v3:** 80 requests
    - To handle this, I **implemented request throttling** using Python's `time` module to ensure I stayed within the allowed limits.
    - I explored IP rotation techniques (ProxyChains) to test API rate limits, but streamlined the process for efficiency.

9. **Verification and comparison with the solution endpoint:**
    - After extracting the data, I verified my results using the **solution endpoint**, which returned an encoded response. I encoded it using the Base64 method, yielding the following result:

```
v1_size: 18632

v2_size: 13730

v3_size: 12517

v1_char_set: a-z
v2_char_set: a-z0-9
v3_char_set: a-z0-9+-.

v1_query_params: query=ado&max_results=10
max_results_range=1-50
v2_query_params: query=ado&max_results=12
max_results_range=1-75
v3_query_params: query=ado&max_results=15
max_results_range=1-100
```

    - My extracted results were:

        - **v1_size:** 18,609
        - **v2_size:** 13,701
        - **v3_size:** 11,275
    - There was a slight difference between my extracted values and the official values, likely due to missing edge cases or unseen constraints.

10. **Final Thoughts:**

- The process involved **exploring unknown endpoints**, **deciphering undocumented parameters**, and **handling rate limits effectively**.
- It was a challenging yet insightful experience in **API enumeration, request handling, and adaptive querying strategies**.
- The final verification step helped in identifying discrepancies and areas for further optimization.

Findings of the API

The API is a **RESTful autocomplete service** that takes a few characters as a query and returns a list of names that start with the given input.

Available Endpoints

1. **Help Endpoint**
    - `POST http://35.200.185.69:8000/help`
    - It provides guidance on how to proceed with the API exploration.
    - Suggests trying the **`/hint`** endpoint.
    - Mentions that **rate limiting** is enforced on the IP address.
2. **Hint Endpoint**

- `POST http://35.200.185.69:8000/hint`
- Provides a hint when accessed with a **POST** request.

3. **Solution Endpoint**
   - `GET http://35.200.185.69:8000/solution`
   - Returns encoded statistics about the solution.
4. **Version 1 (v1) Autocomplete Endpoint**
   - `GET http://35.200.185.69:8000/v1/autocomplete?query=a`
   - By default, it returns **10 results**, but this can be adjusted using the **`max_results`** parameter within the range of **1 to 50**.
5. **Version 2 (v2) Autocomplete Endpoint**
   - `GET http://35.200.185.69:8000/v2/autocomplete?query=a`
   - By default, it returns **12 results**, but the **`max_results`** parameter allows modification within the range of **1 to 75**.
6. **Version 3 (v3) Autocomplete Endpoint**
   - `GET http://35.200.185.69:8000/v3/autocomplete?query=a`
   - By default, it returns **15 results**, but the **`max_results`** parameter can be used to modify the count within the range of **1 to 100**.