Prerequisites for the Subdomain Enumeration Tool Project

By: **Inlighn Tech**

Overview

The Subdomain Enumeration Tool project tasks students with creating a Python script to identify valid subdomains of a target domain (e.g., youtube.com) by testing a list of potential subdomain names against it. The solution uses multithreading to perform concurrent HTTP requests, checking accessibility and saving results to a file. To write this script, students need foundational knowledge and skills from the curriculum, supported by practical code examples they can run locally. These prerequisites and hands-on snippets ensure students understand how enumeration works and can implement the script effectively.

Prerequisite Knowledge and Skills

1. **DNS Fundamentals – Understanding Subdomains**
    - **What to Know**:

        - The Domain Name System (DNS) translates domain names (e.g., youtube.com) into IP addresses. Subdomains (e.g., www.youtube.com, api.youtube.com) are hierarchical extensions, often pointing to distinct servers or services.
        - Subdomain enumeration tests these extensions to find active ones, a key reconnaissance technique in cybersecurity.
    - **How It Applies**:
        - The script constructs URLs (e.g., http://subdomain.youtube.com) and uses HTTP requests to verify subdomain existence via DNS resolution.
    - **Curriculum Source**:
        - "What is DNS.mp4" (Subdomain Enumeration Tool Solution, Lesson 1).
    - **Practical Prep**:
        - Run nslookup youtube.com or dig youtube.com to see DNS in action.
    - **Python Code Block**:

```
# Simple script to resolve a domain to its IP address
import socket

domain = "youtube.com"
try:
    ip = socket.gethostbyname(domain)
    print(f"IP address of {domain}: {ip}")
except socket.gaierror:
    print(f"Could not resolve {domain}")
```

        - **Run It**: Save as dns_test.py, run with python dns_test.py. Shows how DNS resolves a domain, a precursor to subdomain testing.

2. **Basic Python Syntax and Data Structures**
    - **What to Know**:
        - Variables: Store data like domain and subdomain lists.

        - Lists: Handle collections (e.g., subdomains, discivered_subdomains).
        - File I/O: Read from subdomains.txt, write to discovered_subdomains.txt.
    - **How It Applies**:
        - The script reads subdomains from a file, processes them in a list, and writes results, relying on these basics.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lessons 1-3 (Data Types, Variables), 6 (Lists), 11 (IO with Basic Files).
    - **Practical Prep**:
        - Practice list and file operations with the code below.
    - **Python Code Block**:

```
# Read and write a list to/from a file
subdomains = ["www", "mail", "api"]

# Write to a file
with open("test_subdomains.txt", "w") as f:
    for sub in subdomains:
        print(sub, file=f)

# Read from the file
with open("test_subdomains.txt") as f:
    loaded_subdomains = f.read().splitlines()
    print("Loaded subdomains:", loaded_subdomains)
```

    -

        - **Run It**: Save as file_io_test.py, run with python file_io_test.py. Creates test_subdomains.txt and reads it back, mimicking the script's file handling.

3. **String Manipulation**
    - **What to Know**:
        - Concatenation and f-strings: Build URLs from parts (e.g., subdomain and domain).
        - splitlines(): Convert file content into a list of strings.
    - **How It Applies**:

        - The script forms URLs with f'http://{subdomain}.{domain}' to test subdomains.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lessons 4-5 (Strings, Indexing/Slicing).
        - "WORKING WITH DATA STRUCTURES" Lessons 1-2 (String Properties, Print Formatting).
    - **Practical Prep**:
        - Experiment with string formatting in the code below.
    - **Python Code Block**:

```
# Build URLs with string formatting
domain = "example.com"
subdomains = ["www", "mail", "test"]

for sub in subdomains:
    url = f"http://{sub}.{domain}"
    print(f"Generated URL: {url}")
```

    -
        - **Run It**: Save as string_test.py, run with python string_test.py. Shows how to construct URLs like the script does.

4. **Functions and Arguments**
    - **What to Know**:
        - Define functions with def for reusable code (e.g., check_subdomain).
        - Pass arguments to functions to process specific data.
    - **How It Applies**:
        - check_subdomain(subdomain) encapsulates the logic to test each subdomain.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lessons 17-19 (Introduction to Functions, def Keyword, Basics of Functions).
    - **Practical Prep**:
        - Practice function creation with the example below.

- **Python Code Block**:

```
# Define and use a function with an argument
def greet(name):
    message = f"Hello, {name}!"
    print(message)

people = ["Alice", "Bob", "Charlie"]
for person in people:
    greet(person)
```

-
-

    - **Run It**: Save as function_test.py, run with python function_test.py. Demonstrates passing arguments, similar to check_subdomain.

5. **Exception Handling**
    - **What to Know**:
        - Use try/except to catch errors (e.g., network failures).
        - Handle specific exceptions like requests.ConnectionError.
    - **How It Applies**:
        - The script uses try/except to ignore invalid subdomains without crashing.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lesson 21 (Errors and Exception Handling).
    - **Practical Prep**:
        - Test error handling with the code below.
    - **Python Code Block**:

```
# Handle errors with try/except
numbers = [1, 0, 2]
for n in numbers:
    try:
        result = 10 / n
        print(f"10 / {n} = {result}")
    except ZeroDivisionError:
        print(f"Error: Division by {n} failed")
```

-

    - **Run It**: Save as error_test.py, run with python error_test.py. Shows how to catch errors, akin to handling ConnectionError.

6. **Networking Basics – HTTP and Requests**
    - **What to Know**:
        - HTTP is the web communication protocol; a successful GET request (e.g., status 200) indicates a subdomain exists.
        - The requests library simplifies HTTP requests in Python.
    - **How It Applies**:
        - requests.get(url) tests subdomain accessibility in the script.
    - **Curriculum Source**:
        - "NETWORKING, SOCKETS, AND CYBERSECURITY" PDFs (TCP/IP basics, implied HTTP).
        - Note: requests isn't explicitly taught; students need pip install requests.
    - **Practical Prep**:
        - Experiment with requests in the code below.
    - **Python Code Block**:

```
# Test HTTP requests with requests library
import requests

url = "http://example.com"
try:
    response = requests.get(url)
    print(f"Status code for {url}: {response.status_code}")
except requests.ConnectionError:
    print(f"Failed to connect to {url}")
```

    -

        - **Run It**: Install requests (pip install requests), save as http_test.py, run with python http_test.py. Shows HTTP request success/failure.

7. **Multithreading**
    - **What to Know**:

        - Threads enable parallel task execution (e.g., checking multiple subdomains).
        - threading.Thread creates threads; start() launches them; join() waits for completion.
        - threading.Lock ensures thread-safe access to shared data (e.g., discivered_subdomains).
    - **How It Applies**:
        - The script uses threads for speed and a lock to safely update the results list.
    - **Curriculum Source**:
        - "Network Scanner" Lesson 4 ("Working with Threads for our program.mp4").
        - "Port Scanner" Lesson 3 ("Working with Threads.mp4").
    - **Practical Prep**:
        - Try threading with the example below.
    - **Python Code Block**:

```
# Basic multithreading example
import threading
import time

def task(name):
    print(f"Thread {name} starting")
    time.sleep(1)  # Simulate work
    print(f"Thread {name} finished")

threads = []
for i in range(3):
    t = threading.Thread(target=task, args=(i,))
    t.start()
    threads.append(t)

for t in threads:
    t.join()
print("All threads done")
```

    -

- **Run It**: Save as thread_test.py, run with python thread_test.py. Shows concurrent execution, preparing for the script's threading.

How Subdomain Enumeration Works

- **Concept**:
  - Subdomain enumeration identifies active subdomains by testing a list (e.g., from subdomains.txt) against a domain. A successful HTTP request confirms existence; a failure (e.g., connection error) indicates it doesn't.
  - This is a reconnaissance technique to map a target's infrastructure.
- **Script Workflow**:
  - Load subdomain list from a file.
  - Spawn threads to test each subdomain's URL concurrently.
  - Catch connection errors to skip invalid subdomains.
  - Use a lock to safely collect discovered subdomains.
  - Save results to a file after all threads complete.
- **Why Multithreading?**:
  - Sequential checks are slow for large lists; threads parallelize the process, enhancing efficiency.

How to Write the Enumeration Script

Using the prerequisites, students can build the script step-by-step:

1. **Setup**:
   - Import requests and threading.
   - Set domain = 'youtube.com', create discivered_subdomains = [], and initialize lock = threading.Lock().
2. **Read Subdomains**:
   - Use with open('subdomains.txt') as file: and splitlines() to load subdomains. Create a subdomains.txt with entries like www, mail, api.
3. **Define Check Function**:

- Write def check_subdomain(subdomain):, form url = f'http://{subdomain}.{domain}', and use try/except with requests.get(url) to test it.
- If successful, print and append to discivered_subdomains with lock.

4. **Launch Threads**:
   - Iterate over subdomains, create threading.Thread with target=check_subdomain, start each, and store in threads.
5. **Wait and Save**:
   - Use thread.join() in a loop to wait for completion, then write discivered_subdomains to discovered_subdomains.txt.

Notes for Students

- **Setup**: Ensure Python 3 is installed and run pip install requests before using the requests snippets.
- **Engagement**: Each code block is simple, standalone, and builds toward the project. Run them to see concepts in action (e.g., threading speed vs. sequential).
- **Next Steps**: After mastering these, combine them into the full script, starting with a small subdomains.txt (e.g., 5-10 entries) to test locally.

Jump to the Subdomain Enumeration Project Description if you are ready now