Information Gathering Tools | Minor Project

Name: B. Keerthi Prasanna

Email: keerthi2004b@gmail.com

Domain: Cyber Security

A software program or script created to gather data and information about a target system, network, or website is referred to as an **information-gathering tool**, sometimes known as a reconnaissance tool. These tools are frequently used by penetration testers, ethical hackers, and security experts to evaluate a target's security posture and pinpoint any potential flaws.

Tools for acquiring information use a variety of strategies and techniques. Typical strategies include:

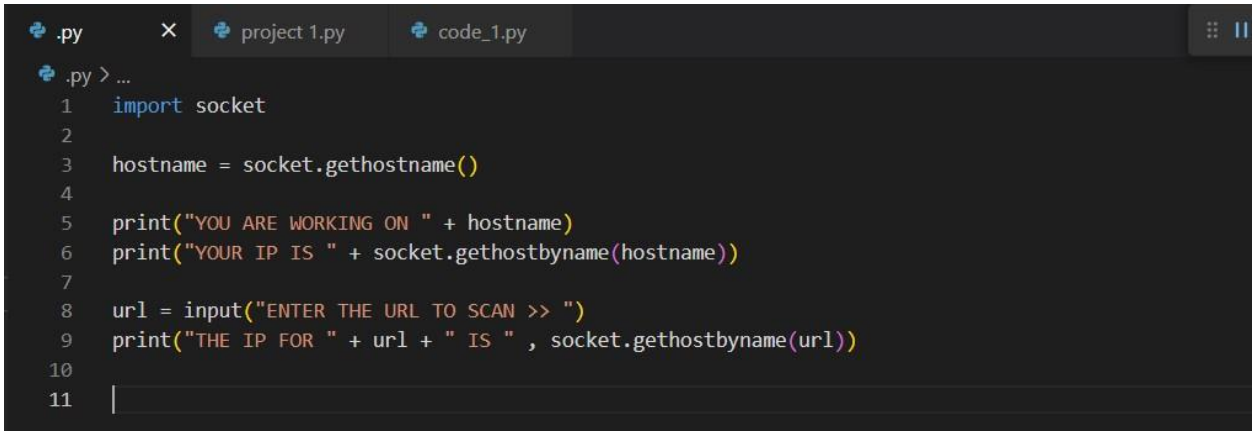
- **DNS enumeration** involves contacting DNS servers to learn the domain names, subdomains, and related IP addresses of the target.
- **Port scanning tools** search a target system for open ports and services that are using those ports. Identification of potential entry points or weaknesses may be aided by this knowledge.
- **WHOIS Lookup**: Tools for a WHOIS lookup retrieve registration and ownership facts for a domain name, including the owner's contact information.
- **OS Fingerprinting**: To identify the operating system used by the target system, OS fingerprinting programs examine network packets. Finding possible operating system-specific vulnerabilities may be possible using this knowledge.

- **Web scraping**: By analyzing the HTML code, web scraping programs gather data from websites. This may entail compiling data on the folders, files, and other elements of the website that may contain sensitive information.
- **Social media profiling** refers to software that collects data from social media sites to learn more about a person or business. Gathering names, job titles, email addresses, and other personal or professional information is one way to do this.

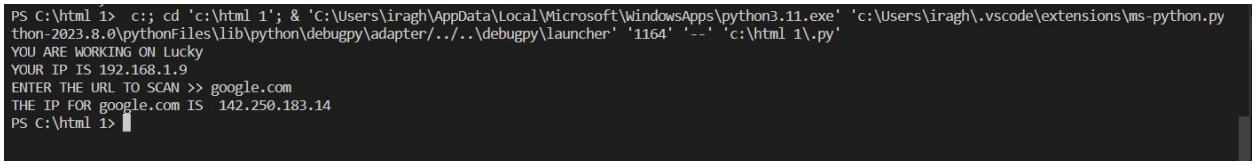
This project involves writing a Python script that retrieves the hostname and IP address of the computer that it is running on. The user is then asked for the IP address linked with the URL, which is then acquired using the **socket.gethostbyname()** function.

To create and run this program, we're utilizing Visual Studio Code.

CODE:



OUTPUT:



VERIFICATION:

```
C:\Users\iragh>ping google.com

Pinging google.com [142.250.183.14] with 32 bytes of data:
Reply from 142.250.183.14: bytes=32 time=25ms TTL=118
Reply from 142.250.183.14: bytes=32 time=25ms TTL=118
Reply from 142.250.183.14: bytes=32 time=25ms TTL=118
Reply from 142.250.183.14: bytes=32 time=27ms TTL=118

Ping statistics for 142.250.183.14:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 25ms, Maximum = 27ms, Average = 25ms
```

```
C:\Users\iragh>ping twitter.com

Pinging twitter.com [104.244.42.129] with 32 bytes of data:
Reply from 104.244.42.129: bytes=32 time=82ms TTL=53
Reply from 104.244.42.129: bytes=32 time=84ms TTL=53
Reply from 104.244.42.129: bytes=32 time=85ms TTL=53
Reply from 104.244.42.129: bytes=32 time=85ms TTL=53

Ping statistics for 104.244.42.129:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 82ms, Maximum = 85ms, Average = 84ms
```

Explanation of the code:

1. The 'socket' module is imported, giving access to numerous networking features including hostname and IP address retrieval.
2. It calls 'socket.gethostname()' to retrieve the hostname of the machine running the script and assigns it to the 'hostname' variable.
3. 'The print()' method is used to display the hostname.
4. It calls 'socket.gethostbyname(hostname)' to retrieve the IP address associated with the hostname and prints it using the 'print()' function.
5. To assign a URL to the URL variable, it asks the user to do so using the 'input()' function.
6. The IP address connected to the entered URL is obtained by calling 'socket.gethostbyname(URL)', which is then printed using the 'print()' function.

Subdomain enumeration is the process of discovering subdomains associated with a particular domain.

Subdomains are prefixes that are a part of the domain name and are separated from it by a dot, for example, "subdomain.example.com". Enumerating subdomains is frequently done to do reconnaissance and security analysis.

Enumerating all potential subdomains of a specified domain is the aim of subdomain enumeration. Discovering subdomains makes it possible to learn more about the infrastructure of the target, find potential attack entry points, and find services that are open to the public.

There are many different subdomain enumeration strategies, however the following are a few widely used ones:

- **Wordlist-based enumeration**: This method makes use of a dictionary or wordlist that includes potential subdomains. The target domain and each entry in the wordlist are combined to create a subdomain. Following that, requests are made to confirm the existence of each subdomain.
- **Brute-force subdomain enumeration**: This technique includes methodically creating subdomain names and comparing them to the target domain. Although this approach is more thorough, it can also be time- and resource-consuming.
- **Search engine scraping**: By using certain search operators, search engines can be utilized to find subdomains. The destination domain's connected indexed subdomains are frequently included in the search results.
- **DNS zone transfers**: In rare circumstances, DNS zone transfer errors could provide an attacker access to the full list of subdomains. However, because correct DNS server setups exist, this strategy is less frequently used.
- **Passive enumeration**: Passive approaches entail monitoring and examining open data sources, such as DNS records, public code repositories, certificate transparency logs, and historical DNS information. These resources may offer information on prospective subdomains.

In the following project , we are creating a Python script that uses a **wordlist of subdomains to do subdomain enumeration** for a given domain.

Before writing the code, **activate a virtual environment**, often referred to as "venv," which allows you to install and manage Python packages separately from your system-wide Python installation. By following the below command:

On Windows: myenv\Scripts\activate.bat

On Unix/Linux: source myenv/bin/activate

[replace myenv with the desired name of your virtual environment]

The requests library is a popular Python library used for making HTTP requests. It provides a convenient and straightforward way to send HTTP/1.1 requests and handle responses, making it easy to interact with web services and APIs. To install request we use the following command :

pip install requests

```
C:\Users\iragh\PycharmProjects\pythonProject1\venv>Scripts\activate

(venv) C:\Users\iragh\PycharmProjects\pythonProject1\venv>pip install requests
Requirement already satisfied: requests in c:\users\iragh\pycharmprojects\pythonproject1\venv\lib\site-packages (2.31.0)Requirement already satisfied: chars
et-normalizer<4,>=2 in c:\users\iragh\pycharmprojects\pythonproject1\venv\lib\site-packages (from requests) (3.1.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\iragh\pycharmprojects\pythonproject1\venv\lib\site-packages (from requests) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\iragh\pycharmprojects\pythonproject1\venv\lib\site-packages (from requests) (2.0.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\iragh\pycharmprojects\pythonproject1\venv\lib\site-packages (from requests) (2023.5.7)

(venv) C:\Users\iragh\PycharmProjects\pythonProject1\venv>
```

SOURCE CODE 1 :

.py | project 1.py | code_1.py

code_1.py > ...

```python
import requests

# the domain to scan for subdomains
domain = "google.com"

# read all subdomains
file = open("subdomains.txt")
# read all content
content = file.read()
# split by new lines
subdomains = content.splitlines()

for subdomain in subdomains:
    # construct the url
    url = f"http://{subdomain}.{domain}"
    try:
        # if this raise an ERROR, that means the subdomain does not exist
        requests.get(url)
    except requests.ConnectionError:
        # if the subdomain does not exist, just pass, print nothing
        pass
    else:
        print("[+] discovered subdomain:", url)
```

OUTPUT 1:

```
Region: Maharashtra
[+] discovered subdomain: http://blog.google.com
[+] discovered subdomain: http://m.google.com
[+] discovered subdomain: http://mobile.google.com
[+] discovered subdomain: http://search.google.com
[+] discovered subdomain: http://api.google.com
[+] discovered subdomain: http://admin.google.com
[+] discovered subdomain: http://news.google.com
[+] discovered subdomain: http://sms.google.com
[+] discovered subdomain: http://video.google.com
[+] discovered subdomain: http://ads.google.com
[+] discovered subdomain: http://wap.google.com
[+] discovered subdomain: http://download.google.com
[+] discovered subdomain: http://chat.google.com
[+] discovered subdomain: http://image.google.com
[+] discovered subdomain: http://tv.google.com
[+] discovered subdomain: http://services.google.com
[+] discovered subdomain: http://music.google.com
[+] discovered subdomain: http://images.google.com
[+] discovered subdomain: http://pay.google.com
PS C:\html 1>
```

SOURCE CODE 2 :

.py | project 1.py | code_1.py

code_1.py > ...

```python
import requests

# the domain to scan for subdomains
domain = "twitter.com"

# read all subdomains
file = open("subdomains.txt")
# read all content
content = file.read()
# split by new lines
subdomains = content.splitlines()

for subdomain in subdomains:
    # construct the url
    url = f"http://{subdomain}.{domain}"
    try:
        # if this raise an ERROR, that means the subdomain does not exist
        requests.get(url)
    except requests.ConnectionError:
        # if the subdomain does not exist, just pass, print nothing
        pass
    else:
        print("[+] discovered subdomain:", url)
```

OUTPUT 2 :

```
[+] discovered subdomain: http://blog.google.com
[+] discovered subdomain: http://m.google.com
[+] discovered subdomain: http://mobile.twitter.com
[+] discovered subdomain: http://search.twitter.com
[+] discovered subdomain: http://api.twitter.com
[+] discovered subdomain: http://dev.twitter.com
[+] discovered subdomain: http://sms.twitter.com
[+] discovered subdomain: http://marketing.twitter.com
[+] discovered subdomain: http://video.twitter.com
[+] discovered subdomain: http://media.twitter.com
[+] discovered subdomain: http://static.twitter.com
[+] discovered subdomain: http://ads.twitter.com
[+] discovered subdomain: http://download.twitter.com
PS C:\html 1>
```

Explanation of the code:

1. Importing the requests module to make HTTP requests.
2. Assigning the domain to scan for subdomains.
3. Opening a file called "subdomains.txt" that contains a list of subdomains.
4. Reading the content of the file and storing it in the content variable.
5. Splitting the content by new lines to obtain a list of subdomains, stored in the subdomains variable.
6. Iterating over each subdomain in the subdomains list.
7. Constructing the URL by combining the current subdomain with the main domain using string formatting. For example, if the current subdomain is "api", the URL will be "http://api.twitter.com".
8. Using a try-except block to handle any connection errors that may occur when making the HTTP request. If a connection error is raised, it means the subdomain does not exist, so the script moves to the next subdomain without printing anything.
9. If no connection error occurs, it means the subdomain exists, so the script prints a message indicating the discovered subdomain.

Note that this script assumes that the "subdomains.txt" file exists in the same directory as the script and contains a list of subdomains, each on a separate line.

To use this script, you need to provide a "subdomains.txt" file with the list of subdomains you want to scan and run the script using Python. The script will output the discovered subdomains, if any, for the specified domain.

In the following project, we are writing a Python script that **retrieves information about a given URL**. It uses the requests library to send **HTTP requests and retrieve data from the URL**. Additionally, it utilizes the **'socket'** module to retrieve the IP address of the URL and the json module to parse the **'JSON'** response.

Code:

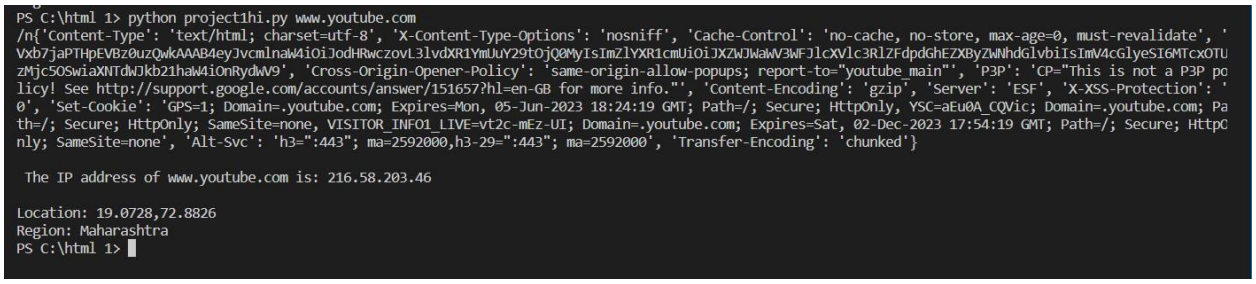
.py | project 1.py | code_1.py

project 1.py > ...

```python
import sys
import requests
import socket
import json

if len(sys.argv) < 2:
    print("Usage: " + sys.argv[0] + " <url>")
    sys.exit(1)
url = sys.argv[1]
try:
    req = requests.get("https://"+ url)
    print("/n"+ str(req.headers))

    ip_address = socket.gethostbyname(url)
    print("\n The IP address of "+ url + " is: " + ip_address + "\n")

    req_two = requests.get("https://ipinfo.io/"+ ip_address + "/json")
    resp_ = json.loads(req_two.text)
    print("Location: " + resp_["loc"])
    print("Region: " + resp_["region"])
except requests.RequestException:
    print("Error occurred while making the request")
except socket.gaierror:
    print("Invalid URL or unable to retrieve IP address.")
except KeyError:
    print("Error occurred while retrieving location or region information. ")
```

Ouput:

```
PS C:\html 1>  c:; cd 'c:\html 1'; & 'C:\Users\iragh\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\iragh\.vscode\extensions\ms-python.python-2023.8.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '1364' '--' 'c:\html 1\project 1.py'
Usage: c:\html 1\project 1.py <url>
PS C:\html 1> python project1hi.py www.google.com
/n{'Date': 'Mon, 05 Jun 2023 17:54:06 GMT', 'Expires': '-1', 'Cache-Control': 'private, max-age=0', 'Content-Type': 'text/html; charset=ISO-8859-1', 'Content-Security-Policy-Report-Only': "object-src 'none';base-uri 'self';script-src 'nonce-VKXJJIFWRNeJJtcFJ9vFDQ' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp", 'P3P': 'CP="This is not a P3P policy! See g.co/p3phelp for more info."', 'Content-Encoding': 'gzip', 'Server': 'gws', 'X-XSS-Protection': '0', 'X-Frame-Options': 'SAMEORIGIN', 'Set-Cookie': '1P_JAR=2023-06-05-17; expires=Wed, 05-Jul-2023 17:54:06 GMT; path=/; domain=.google.com; Secure, AEC=AUEFqZei3f1EvZ8V9zlUP38ygd7G-fgYrH4chqVTpYKY4gChL9rGpgkq4o8; expires=Sat, 02-Dec-2023 17:54:06 GMT; path=/; domain=.google.com; Secure; HttpOnly; SameSite=lax, NID=511=ev_PigzzZwB3NNgyFcycyrBizJMCGYI4laTh5xKqdjR83WkfFEhu4jqiRFKqkG8Bjnr_nBYhb7bd7U0qmofI7PqQmGldyqU-WD9fS-ovIwqoO_nopSw3DW4WHmE5_GimxS45zSm97KOG16HxkEWy7Yb4zLo62fKOWikPY9HKpc4; expires=Tue, 05-Dec-2023 17:54:06 GMT; path=/; domain=.google.com; HttpOnly', 'Alt-Svc': 'h3=":443"; ma=2592000,h3-29=":443"; ma=2592000', 'Transfer-Encoding': 'chunked'}

 The IP address of www.google.com is: 142.250.192.68

Location: 19.0728,72.8826
Region: Maharashtra
```



Explanation of the code:

1. Importing the necessary modules: sys, requests, socket, and json.
2. Checking if the script is provided with the URL as a command-line argument. If not, it displays a usage message and exits the script.
3. Retrieving the URL from the command-line argument (sys.argv[1]).
4. Using the requests.get() function to send an HTTP GET request to the URL provided.
5. Printing the headers of the response obtained from the URL.
6. Using the socket.gethostbyname() function to retrieve the IP address associated with the URL.
7. Printing the IP address.
8. Sending another HTTP GET request to "https://ipinfo.io//json" to retrieve information about the IP address.
9. Loading the JSON response using json.loads().
10. Printing the location and region information obtained from the JSON response.

CONCLUSION:

The combined project consists of three codes that perform various tasks related to URLs, IP addresses, subdomain enumeration, and HTTP requests.

The project encompasses three codes that collectively offer functionalities related to working with URLs and retrieving relevant information.

Firstly, the **' socket.gethostbyname()'** function is utilized to obtain the IP address associated with a given URL. This functionality allows for easy retrieval of the IP

address and can be beneficial for network troubleshooting, system administration, and networking applications.

Secondly, the project involves subdomain enumeration using a wordlist of subdomains for a specified domain. The code iterates through the subdomain list, constructs URLs by combining subdomains with the domain, and checks their existence by sending HTTP requests. This approach aids in identifying potential subdomains associated with the given domain, which can be useful in security assessments and penetration testing.

Lastly, the code retrieves information about a specified URL by employing the **'requests'** library to send HTTP requests and retrieve data. Additionally, the **'socket'** module is used to retrieve the IP address of the URL, and the **'JSON'** module assists in parsing the JSON response received from the URL. This comprehensive approach enables the retrieval of various information, such as headers, IP address, location, and region details, which can be valuable for tasks like web scraping, data analysis, or network monitoring.

In **conclusion**, this project brings together functionality for obtaining IP addresses, performing subdomain enumeration, and retrieving information from URLs. However, it is essential to use these capabilities responsibly, respecting legal and ethical considerations, and ensuring proper authorization and permissions when interacting with third-party domains or networks.