Foot Printing a Web Server and a Web Application

The concept of foot printing a web server The concept of penetration testing cannot be explained or performed in a single step; therefore, it has been divided into several steps. Foot printing is the first step in pentesting, where an attacker tries to gather information about a target. In today's world, e-commerce is growing rapidly. Due to this, web servers have become a prime target for hackers. In order to attack a web server, we must first know what a web server is. We also need to know about the web-server hosting software, hosting operating system, and what applications are running on the web server. After getting this information, we can build our exploits. Obtaining this information is known as foot printing a web server

This is the code:

```
import re
import random
import urllib.request
from urllib.error import HTTPError

url2 = input("Enter the URL: ")
a_tag = "<address>"  # Adjust the regular expression pattern as needed
file_text = open("result.txt", 'a')


while True:  # Use a while loop with break statements for better control flow
    try:
        if not re.match(r'^https?://', url2):
            raise ValueError("Invalid URL format. Please include the scheme (e.g.,
http://)")

        http_r = urllib.request.urlopen(url2)
        content = http_r.read().decode('utf-8')  # Decode content here

        # Print request attributes

        print("---------------------------------")
        print("------------REQUEST------------:")
        print("---------------------------------")
        print(f"\nMethod: GET")  # Or replace GET with the actual HTTP method you used
        print(f"\nURL: {url2}")

        # Print response headers
        print("\nHEADERS:")
        print("---------------------------------")
        for header, value in http_r.headers.items():
            print(f"\n{header}: {value}")
        print("\n---------------------------------\n")

        if http_r.getcode() == 404:
```

```
            file_text.write(f"-------------\n{url1}-------------\n")
            file_text.write(content)
            matches = re.finditer(a_tag, content)
            for match in matches:
                print("Coding is not good")
                print(match.group())  # Print matched content
            break  # Break out of the loop since 404 is found

        elif http_r.getcode() == 200:
            print("\n---------------------------------\n")
            print("Web page is using a custom Error page")
            print("\n---------------------------------\n")
            server_header = http_r.headers.get('Server')
            if server_header:
               print("\n---------------------------------\n")
               print(f"The server is: {server_header}")
               print("\n---------------------------------\n")
            else:
                print("\n---------------------------------\n")
                print("Server information not available")
                print("\n---------------------------------\n")
            break  # Break out of the loop since 200 is found
    except HTTPError as e:
        print(f"HTTP Error {e.code}: {e.reason}")
        break  # Break out of the loop on error

file_text.close()  # Close the file after writing to it
```

the output :

```
PS C:\Users\Mohamed> python -u "d:\cyper security\GRADUATION PROJECT\python network\tempCodeRunnerFile.py"
Enter the URL: http://testphp.vulnweb.com/
---------------------------------
-------------REQUEST-------------:
---------------------------------

Method: GET

URL: http://testphp.vulnweb.com/

HEADERS:
---------------------------------

Server: nginx/1.19.0

Date: Mon, 12 Feb 2024 14:35:56 GMT

Content-Type: text/html; charset=UTF-8

Transfer-Encoding: chunked

Connection: close

X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1

---------------------------------

Web page is using a custom Error page

---------------------------------

The server is: nginx/1.19.0

---------------------------------
```

```
PS C:\Users\Mohamed> python -u "d:\cyper security\GRADUATION PROJECT\python network\tempCodeRunnerFile.py"
Enter the URL: https://www.remotasks.com/en/tasklog
HTTP Error 403: Forbidden
PS C:\Users\Mohamed>
```

```
Server: ESF

X-XSS-Protection: 0

Set-Cookie: GPS=1; Domain=.youtube.com; Expires=Mon, 12-Feb-2024 15:19:05 GMT; Path=/; Secure; HttpOnly

Set-Cookie: YSC=nFIjX1gFH9c; Domain=.youtube.com; Path=/; Secure; HttpOnly; SameSite=none

Set-Cookie: VISITOR_INFO1_LIVE=eM28ViXxvB8; Domain=.youtube.com; Expires=Sat, 10-Aug-2024 14:49:05 GMT; Path=/; Secure; HttpOnly; SameSite=none

Set-Cookie: VISITOR_PRIVACY_METADATA=CgJFRxIEGgAgHw%3D%3D; Domain=.youtube.com; Expires=Sat, 10-Aug-2024 14:49:05 GMT; Path=/; Secure; HttpOnly; SameSite=lax

Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

Accept-Ranges: none

Vary: Accept-Encoding

Connection: close

Transfer-Encoding: chunked

---------------------------------


---------------------------------

Web page is using a custom Error page

---------------------------------


---------------------------------

The server is: ESF

---------------------------------

PS C:\Users\Mohamed>
```

Let's break down the provided Python code step by step:

1. **Importing Libraries**:

- `import re`: Imports the regular expression (regex) module, which is used for pattern matching.

- `import random`: Imports the random module, which is used to generate random characters.

- `import urllib.request`: Imports the urllib library for making HTTP requests.

- `from urllib.error import HTTPError`: Specifically imports the HTTPError class from the urllib.error module, which is used to handle HTTP errors.

2. **Input URL**:

- `url2 = input("Enter the URL: ")`: Prompts the user to enter a URL, which is then stored in the `url2` variable.

3. **Opening File**:

- `file_text = open("result.txt", 'a')`: Opens a file named "result.txt" in append mode. This file will be used to store the results of the script.

4. **Main Loop**:

- `while True:`: Starts an infinite loop, which will continue until explicitly broken.

5. **URL Validation**:

- `if not re.match(r'^https?://', url2):`: Checks if the URL starts with "http://" or "https://". If not, raises a ValueError indicating that the URL format is invalid.

6. **HTTP Request**:

- `http_r = urllib.request.urlopen(url2)`: Sends an HTTP request to the URL specified by `url2` and stores the response in the `http_r` variable.

- `content = http_r.read().decode('utf-8')`: Reads the content of the HTTP response and decodes it as UTF-8.

7. **Print Request Attributes**:

- Prints the HTTP method used (always GET in this case) and the URL.

8. **Print Response Headers**:

- Iterates over the headers in the HTTP response and prints each header and its corresponding value.

9. **Handling HTTP Error**:

- Catches any HTTP errors that occur during the request and prints the error code and reason.

10. **Closing File**:

- `file_text.close()`: Closes the file after writing to it.

This script essentially makes an HTTP request to the specified URL, prints the request attributes and response headers, and handles any errors that may occur during the process. It provides detailed information about the HTTP request and response, including headers and status codes.