Exercise 2

Create a simple JSON HTTP client

Prior Knowledge
Unix Command Line Shell
Some simple Python

Learning Objectives
Understand the basics of a programmatic web client
Send data between two languages/frameworks
Parse JSON data into variables

Software Requirements
Python
The server from Exercise 1.

Creating a simple python client

Python is a powerful dynamic language that is widely used in scripting and web applications. It is a common target for creating and consuming services.

Some people find one aspect of Python a little frustrating: it is sensitive to indentation. I recommend using a Python-aware editor like Atom or PyCharms.

1. Firstly, create a new directory for Exercise 2
   ```
   mkdir ~/ex2
   cd ~/ex2
   ```

2. Let's make sure python3's pip is installed and at the latest version.

```
sudo apt install python3-pip
sudo pip3 install pip --upgrade
```

3. Now we need to code our client. Create and edit a file called random-client.py
in the ex2 directory and type the following code (available at https://freo.me/rand-client3)
*Hint: python is **indentation-sensitive!***

```
import httplib2
import json

url = "http://localhost:8080"

h = httplib2.Http()
resp, content = h.request(url, "GET")

print ("return code: " + resp['status'])
result = json.loads(content)
print ("random number: " + str(result['random']))
```

4. The code is pretty simple. It first imports two required libraries (one for HTTP and the other for JSON). After instantiating an HTTP object, it calls it against the server's URL. It then prints out the return code, parses the response, and then prints the parsed random number as a String.

5. You can run this by typing
`python random-client.py`

6. You should see something like:

```
oxsoa@oxsoa:~/ex2$ python random-client.py
return code: 200
random number: 13
```

If your localhost server isn't running, look back at Exercise 1 to get it running again.

7. One useful aspect of having a text-based protocol is when it comes to debugging. We are going to insert a simple proxy between the client and the server and use this to show the flow of messages between the two. This utility is very useful especially in debugging difficult problems with embedded software or libraries that are perhaps producing unexpected results.

8. There are a number of proxy tools that can do this, or advanced Linux users can use tools like tcpdump or wireshark. The one we will use for this module is called *mitmdump* (man-in-the-middle dump) and it is a part of a more advanced tool called *mitmproxy*. Its written in Python and

should run on any Python capable system. It is already installed on the Ubuntu systems you are using.

9. mitmdump can be used in two different ways. One is as a genuine HTTP Proxy/SOCKS Proxy. The second approach is where it acts as a reverse proxy. Let's try the reverse proxy approach first.

10. Start a **new terminal window** and type (all on one line):
```
mitmdump --listen-port 8000 --set flow_detail=3
 --mode reverse:http://localhost:8080
```

    This starts up mitmdump listening on port 8000. The flow_detail=3 implies that it will give very detailed output. "--mode reverse" indicates that any traffic it receives should be sent on to http://localhost:8080.

11. Our Python client is not yet going to use this however, because we are still sending requests to port 8080. We need to modify the Python client to send requests to port 8000 instead. It is pretty obvious how to do this!

    Modify your python client to send requests to http://localhost:8000 instead.

12. Try the python client again. You should see something like this in the MITMDUMP window:

```
127.0.0.1:50574: clientconnect
127.0.0.1:50574: GET http://localhost:8080/
    Host: localhost
    accept-encoding: gzip, deflate
    user-agent: Python-httplib2/0.9.2 (gzip)

<< 200 OK 13b
    X-Powered-By: Express
    Content-Type: application/json; charset=utf-8
    Content-Length: 13
    ETag: W/"d-tJ2OSdUhmGDIQZeBN3gOhj6hw0s"
    Date: Sun, 14 Mar 2021 08:42:04 GMT
    Connection: keep-alive

    {
        "random": 63
    }

127.0.0.1:50574: clientdisconnect
```

13. While Reverse Proxy mode is very simple, there are cases where it doesn't work. For example, sometimes the server responds with a fully qualified URL instead of a relative URL, and the client then uses this URL to make a further request. This will ignore the proxy. Hence there is a second approach which is more reliable under all circumstances.

    HTTP includes support for proxies and there is a well-defined specification of how this works. Many systems have a way of configuring a proxy server and port in settings files outside of code, which means that using this model can be used with third-party software, libraries and off the shelf systems.

14. Stop mitmdump (Ctrl-C) and restart it in normal proxy mode:
    `mitmdump --listen-port 8000 --set flow_detail=3`

15. Modify your Python program as follows:

    Firstly, at the top, add a line:

    ```
    import httplib2.socks as socks
    ```

    Now change back the URL to point to http://localhost:8080

    Replace the line:
    ```
    h = httplib2.Http()
    ```
    with:
    ```
    proxy_info = httplib2.ProxyInfo(socks.PROXY_TYPE_HTTP, "localhost", 8000)
    h = httplib2.Http(proxy_info = proxy_info)
    ```

16. Run the program again. You should see something similar to:

17. Can you spot the difference in the two MITMDUMP traces?
    How does this relate to the different styles of usage of the client?

18. Before you finish, please close down the node server and mitmdump server.

19. Congratulations. This exercise is complete.