

# 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  
Pip  
Pip installed httpplib2  
A Text Editor (e.g. Atom)  
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. Now we need to code our client. Create a file called random-client.py in the ex2 directory and type the following code (available at <http://freo.me/rand-client> )

```
import httpplib2
import json

url = "http://localhost:8080"

h = httpplib2.Http()
resp, content = h.request(url, "GET")

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

*Hint: python is **indentation-sensitive**!*

3. 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.
4. You can run this by typing  
`python random-client.py`
5. You should see something like:  

```
oxsoa@oxsoa:~/ex2$ python random-client.py
return code: 200
random number: 13
```
6. 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.
7. 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.
8. 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.
9. Start a **new terminal window** and type:  
`mitmdump --port 8000 -dd --reverse http://localhost:8080`  
  
This starts up mitmdump listening on port 8000. -dd implies that it will give detailed output. --reverse indicates that any traffic it receives should be sent on to http://localhost:8080.
10. Our Python client is not 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.*

11. Try the python client again. You should see something like:

```
oxsoa@oxsoa: ~  
oxsoa@oxsoa:~$ mitmdump --port 8000 -dd --reverse http://localhost:8080  
127.0.0.1:48642: clientconnect  
127.0.0.1 GET http://localhost:8080/  
  host: localhost:8080  
  accept-encoding: gzip, deflate  
  user-agent: Python-httpplib2/0.9.2 (gzip)  
  
<< 200 OK 13B  
  Date: Tue, 24 May 2016 11:59:05 GMT  
  Connection: keep-alive  
  content-length: 13  
  
  {"random":72}  
127.0.0.1:48642: clientdisconnect
```

12. 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 the second approach

13. 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 --port 8000 -dd`

15. Modify your Python program as follows:

Firstly, at the top, add a line:  
`import socks`

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

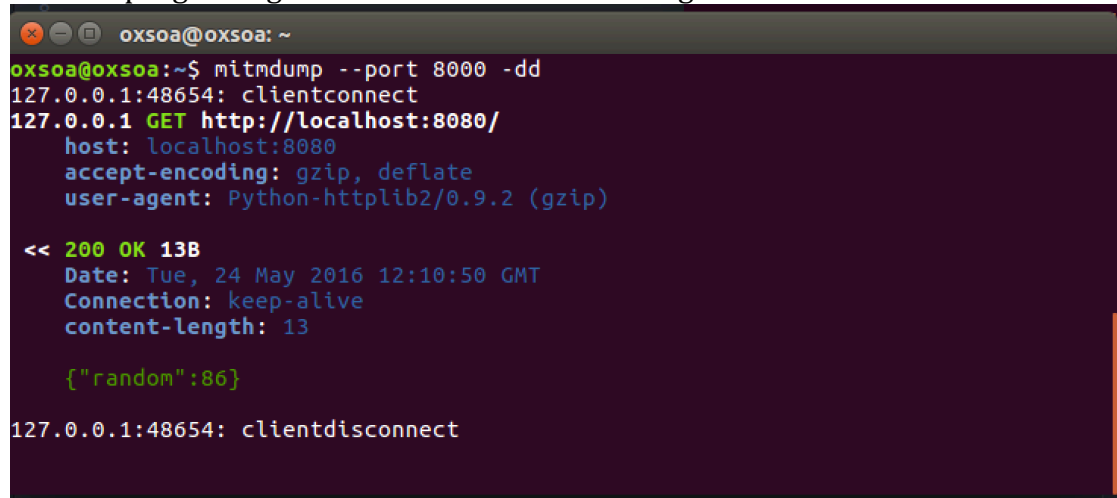
Replace the line:

`h = httpplib2.Http()`

with:

`proxy_info = httpplib2.ProxyInfo  
 (socks.PROXY_TYPE_HTTP, "localhost", 8000)  
h = httpplib2.Http(proxy_info = proxy_info)`

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



```
oxsoa@oxsoa: ~  
oxsoa@oxsoa:~$ mitmdump --port 8000 -dd  
127.0.0.1:48654: clientconnect  
127.0.0.1 GET http://localhost:8080/  
    host: localhost:8080  
    accept-encoding: gzip, deflate  
    user-agent: Python-httpplib2/0.9.2 (gzip)  
  
<< 200 OK 13B  
    Date: Tue, 24 May 2016 12:10:50 GMT  
    Connection: keep-alive  
    content-length: 13  
  
    {"random":86}  
127.0.0.1:48654: clientdisconnect
```

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

18. Congratulations. This exercise is complete.