

CSC 361 Tutorial 1: P1 Spec Go- through, Design Hints, Python

P1 Goal

Build a tool at web client to collect information regarding a web server. The purpose of this project is twofold:

- Experience with socket programming in Python
- Help students understand the application-layer protocols HTTP/HTTPS

Background - HTTP

- HTTP request: a header and a body
- HTTP response: a header and a body

*A single-line
header of
HTTP request*

```
zhiming@DESKTOP-M225BQ0:~$ netcat www.csc.uvic.ca 80
GET /index.html HTTP/1.0

HTTP/1.1 302 Moved Temporarily
Server: nginx/1.14.0 (Ubuntu)
Date: Wed, 20 Jan 2021 04:36:32 GMT
Content-Type: text/html
Content-Length: 170
Connection: close
Location: https://itsupport.cs.uvic.ca/accounts/

<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx/1.14.0 (Ubuntu)</center>
</body>
</html>
```

Background - HTTP

- HTTP request: a header and a body
- HTTP response: a header and a body

The method field

```
zhiming@DESKTOP-M225BQ0: ~$ netcat www.csc.uvic.ca 80  
GET /index.html HTTP/1.0
```

The URI field

```
HTTP/1.1 302 Moved Temporarily  
Server: nginx/1.14.0 (Ubuntu)  
Date: Wed, 20 Jan 2021 04:36:32 GMT  
Content-Type: text/html  
Content-Length: 170  
Connection: close  
Location: https://itsupport.cs.uvic.ca/accounts/
```

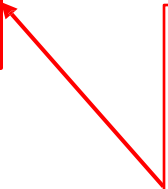
The HTTP version field

```
<html>  
<head><title>302 Found</title></head>  
<body bgcolor="white">  
<center><h1>302 Found</h1></center>  
<hr><center>nginx/1.14.0 (Ubuntu)</center>  
</body>  
</html>
```

Background - HTTP

- HTTP request: a header and a body
- HTTP response: a header and a body

*The header of
HTTP
response*



```
zhiming@DESKTOP-M225BQ0: ~$ netcat www.csc.uvic.ca 80  
GET /index.html HTTP/1.0
```

```
HTTP/1.1 302 Moved Temporarily  
Server: nginx/1.14.0 (Ubuntu)  
Date: Wed, 20 Jan 2021 04:36:32 GMT  
Content-Type: text/html  
Content-Length: 170  
Connection: close  
Location: https://itsupport.cs.uvic.ca/accounts/
```

```
<html>  
<head><title>302 Found</title></head>  
<body bgcolor="white">  
<center><h1>302 Found</h1></center>  
<hr><center>nginx/1.14.0 (Ubuntu)</center>  
</body>  
</html>
```

Background - HTTP

- HTTP request: a header and a body
- HTTP response: a header and a body

*The http
version field*

```
zhiming@DESKTOP-M225BQ0: ~$ netcat www.csc.uvic.ca 80
GET /index.html HTTP/1.0
```

```
HTTP/1.1 302 Moved Temporarily
```

```
Server: nginx/1.14.0 (Ubuntu)
```

```
Date: Wed, 20 Jan 2021 04:36:32 GMT
```

```
Content-Type: text/html
```

```
Content-Length: 170
```

```
Connection: close
```

```
Location: https://itsupport.cs.uvic.ca/accounts/
```

```
<html>
```

```
<head><title>302 Found</title></head>
```

```
<body bgcolor="white">
```

```
<center><h1>302 Found</h1></center>
```

```
<hr><center>nginx/1.14.0 (Ubuntu)</center>
```

```
</body>
```

```
</html>
```

*The status
code field*

*The phrase
field*

- Two main status codes include 200 and 404.
 - 200: the request succeeded, and the information is returned in the response
 - Example: "HTTP/1.0 200 OK\r\n\r\n data data data ..."
 - 404: the requested document does not exist on this server
 - Example: "HTTP/1.0 404 Not Found\r\n\r\n"
- Other useful codes in this assignment:
 - 505: "HTTP Version Not Supported",
 - 302: "302 found" for URL redirection
 - 401: when a password-protected web page is accessed without a correct credential.

Background - HTTP

- HTTP request: a header and a body
- HTTP response: a header and a body

*The http
response body*



```
zhiming@DESKTOP-M225BQ0: ~$ netcat www.csc.uvic.ca 80  
GET /index.html HTTP/1.0
```

```
HTTP/1.1 302 Moved Temporarily  
Server: nginx/1.14.0 (Ubuntu)  
Date: Wed, 20 Jan 2021 04:36:32 GMT  
Content-Type: text/html  
Content-Length: 170  
Connection: close  
Location: https://itsupport.cs.uvic.ca/accounts/
```

```
<html>  
<head><title>302 Found</title></head>  
<body bgcolor="white">  
<center><h1>302 Found</h1></center>  
<hr><center>nginx/1.14.0 (Ubuntu)</center>  
</body>  
</html>
```


Background – URI (Uniform Resource Identifiers)

- Known as the combination of Uniform Resource Locators (URL) and Uniform Resource Names (URN)
- A formatted string which identifies a network resource, i.e.,
protocol://host[:port]/filepath

When a port is not specified, the default HTTP port number is 80, and the default HTTPS port number is 443.

Background - Cookies

- A small piece of data sent by the server.
- The browser may store it and send it back with later requests to the same server.
- Typically, it's used to tell if two requests came from the same browser — keeping a user logged-in, for example.

The Set-Cookie and Cookie headers

- The Set-Cookie HTTP response header sends cookies from the server to the user agent, e.g.,

Set-Cookie: <cookie-name>=<cookie-value>

```
zhiming@DESKTOP-M225BQ0: $ netcat www.google.ca 80
GET /index.html HTTP/1.1

HTTP/1.1 200 OK
Date: Wed, 20 Jan 2021 05:03:21 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2021-01-20-05; expires=Fri, 19-Feb-2021 05:03:21 GMT; path=/; domain=.google.com; Secure
Set-Cookie: NID=207=AYfcxvmpYH5o4fdMjYWoik_oVXWVIYDgYb05q505DU6h1CfFvaotJKdh1U3R1QUEeFnznNf0c4Ub1LnGqw30s6xBnGGLpn01nfIP
qKVYVLfPCWV3cy7vH5L3YaJJJo5fBTCRMGiA545ajtdMtZLU1kNMXLqhxOnmXj6FOoabRUTs; expires=Thu, 22-Jul-2021 05:03:21 GMT; path=/;
domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

Project Description

- Given the URL of a web server, your WebTester needs to find out the following information regarding the web server:
 1. Whether or not the web server supports **HTTP2**,
 2. The **cookie name**, the **expire time** (if any), and the **domain name** (in any) of cookies that the web server will use.
 3. Whether or not the requested web page is **password-protected**.

Output Example

- Run code with

```
%python WebTester.py www.uvic.ca
```

- Output the received response from the server (optional), e.g.,

---Request begin---

GET http://www.uvic.ca/index.html HTTP/1.1

Host: www.uvic.ca

Connection: Keep-Alive

---Request end---

HTTP request sent, awaiting response...

---Response header ---

HTTP/1.1 200 OK

Date: Tue, 02 Jan 2018 22:42:27 GMT

Expires: Thu, 19 Nov 1981 08:52:00 GMT

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0

Pragma: no-cache

Set-Cookie: SESSION_UV_128004=VD3v0JhqL3YUbmazSTJre1; path=/; domain=www.uvic.ca

Set-Cookie: uvic_bar=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/; dom

Keep-Alive: timeout=5, max=100

Connection: close

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

Set-Cookie: www_def=2548525198.20480.0000; path=/

Set-Cookie: TS01a564a5=0183e07534a2511a2dcd274bee873845d67a2c07b7074587c948f80a42c427b1f7ea

Set-Cookie: TS01c8da3c=0183e075346a73ab4544c7b9ba9d7fa022c07af441fc6214c4960d6a9d0db2896; p

Set-Cookie: TS014bf86f=0183e075347c174a4754aeb42d669781e0fafb1f43d3eb2783b1354159a9ad8d81f7

--- Response body ---

Body Body (the actual content)

- Output final results (mandatory)

website: www.uvic.ca

1. Supports http2: no

2. List of Cookies:

cookie name: SESSID_UV_128004, domain name: www.uvic.ca

cookie name: uvic_bar, expires time: Thu, 04-Jan-2018 00:00:01 GMT; domain name: .uvic.ca

cookie name: www_def,

cookie name: TS01a564a5

cookie name: TS01c8da3c, domain name: www.uvic.ca

cookie name: TS014bf86f, domain name: .uvic.ca

3. Password-protected: no

Note that the above output were outdated and does not reflect the ground truth of the current configuration of the web server.

Marking Scheme

you are required to submit your source code to brightSpace in a single zip file (with a readme file)

Components	Weight
Error handling	10
Correct output for “support of http2 ”	20
handling http redirect 302/301	20
List of Cookies	30
Correct output for “password-protected”	15
Readme.txt	5
Total Weight	100

It is possible that you will not get a unique, static answer due to the dynamic changes in some cookies created dynamically based on users’ interactive input.

Some online tool, such as <http://www.cookie-checker.com/>, can find cookies that are triggered by javascript or php code. Nevertheless, finding those cookies is optional for this Assignment.

Notes

- Use python3
- Test/run code on *linux.csc.uvic.ca*
- Use packages supported by *linux.csc.uvic.ca*.
- Do not use packages `httplib`, `TCPClient`, `hyper`, `requests` and other similar third-party packages
- Printout:
 - Anything not specified in Assignment 1 is optional
 - You can show intermediate results to help TA marking when you code cannot work
- Readme file:
 - State clearly how to run your code
- More information about HTTP, HTML, URI: <http://www.w3.org>

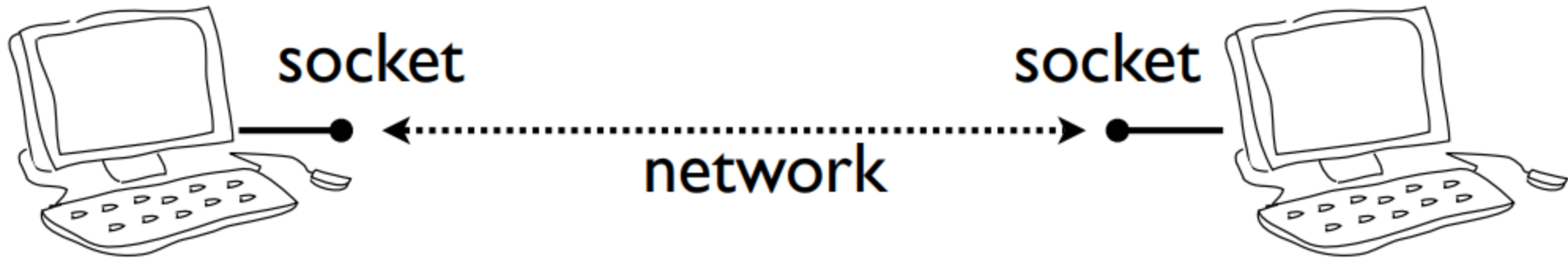
Design Hints

1. Python socket programming
2. http2 detection
3. password-protected detection

Python Socket Programming
(brief, more details next week)

Sockets

- Programming abstraction for network code
- Socket: A communication endpoint



TCP Client

How to make an outgoing connection

```
from socket import *  
s = socket(AF_INET,SOCK_STREAM)  
s.connect(("www.google.ca",80)) # Connect  
s.send("GET /index.html HTTP/1.0\n\n") # Send request  
data = s.recv(10000) # Get response s.close()
```

s.connect(addr) makes a connection

Once connected, use send(),recv() to transmit and receive data

close() shuts down the connection

P1 Design Idea

1. Your program accepts URI from stdin and parses it
2. Connect to the server of the URI (`socket.connect()`)
3. Send an HTTP request (`socket.send()`)
4. Receive an HTTP response (`socket.recv()`)
5. You should also implement a routine that prints out the response from the server, **marking the header and the body**.
6. Analyze the HTTP response to find out required information
7. Your code might need to send **multiple requests** in order to find out the required information.

In particular, if you get an HTTP response with code 302 or 301, you need to send further HTTP requests to the new URI provided by the Location header.

When you finish the client, you can try to connect to any HTTP server. For instance, type ["www.uvic.ca"](http://www.uvic.ca) as the input to the client program and see what response you get.

HTTP2 Detection

1. For HTTP Servers (Port 80):

- HTTP/2 can also be supported over HTTP (without TLS), but this is rare and less common.
- A common way to check for HTTP/2 support over HTTP is to issue an OPTIONS request and inspect the HTTP2-Settings header, which is returned by the server if it supports HTTP/2.

2. For HTTPS Servers (Port 443):

- HTTP/2 is often used over TLS, so it requires an SSL/TLS handshake.
- The **Application-Layer Protocol Negotiation (ALPN)** extension in TLS helps determine the protocol that will be used (HTTP/2 or HTTP/1.1).
- You can check the negotiated protocol after completing the TLS handshake.

"Although the standard itself does not require usage of encryption, most client implementations (Firefox, Chrome, Safari, Opera, IE, Edge) have stated that they will only support HTTP/2 over TLS"

Design Idea

1. Start with simple way, i.e., check http servers with option request

- Craft the OPTIONS Request as follows:

```
request = f"OPTIONS / HTTP/1.1\r\n"
```

```
request += f"Host: {host}\r\n"
```

```
request += f"Connection: keep-alive\r\n"
```

```
request += f"Upgrade: h2c\r\n" # Upgrade to HTTP/2 (cleartext, h2c)
```

```
request += f"Accept: */*\r\n"
```

```
request += f"\r\n" # End of headers
```

- You will typically get a response as follows if the server supports http2

HTTP/1.1 200 OK

Date: Mon, 14 Jan 2025 10:00:00 GMT

Server: Apache/2.4.41 (Unix)

Allow: OPTIONS, GET, POST, HEAD

HTTP2-Settings: AAMAAEAAwAAAB3N0aW5nCgAAIYOx

Connection: keep-alive

Upgrade: h2c

Content-Length: 0

Checking TLS-ALPN

Application-Layer Protocol Negotiation (ALPN) is a Transport Layer Security (TLS) extension that allows the application layer to negotiate which protocol should be performed over a secure connection.

Therefore, at the time when you connect to a HTTPS server, you can check TLS-ALPN to know whether HTTP/2 is used in the secure connection.

2. If the first approach doesn't find http2, then you need to try the second approach: using **TLS Handshake**:

1) create an SSL-wrapped socket (more details about coding next week!)

```
context = ssl.create_default_context()
context.set_alpn_protocols(['h2', 'http/1.1'])
conn = context.wrap_socket(socket.socket(socket.AF_INET),
server_hostname="www.google.ca")
conn.connect(("www.google.ca", 443))
```

2) Get the negotiated protocol from the SSL/TLS handshake

```
negotiated_protocol = conn.selected_alpn_protocol()
```

3) Check if `h2` in the supported protocols

Test your code

- Connect to linux.csc.uvic.ca

For MACOS, LINUX: ssh netlinkid@linux.csc.uvic.ca

For WINDOWS: putty

Warning: Note that we only test your code on [linux.csc.uvic.ca](mailto:netlinkid@linux.csc.uvic.ca). Make sure your code works on that machine before submission