Exercises: HTTP Basics

Experiential Learning Workshop

1 General Guidelines

1. Make a team of two unless stated otherwise.
2. For each exercise, use wireshark capture to verify contents
3. Ensure to use proper capture filter and don't capture unnecessary traffic
4. Where appropriate or applicable, use `wget` or `nc` to access the web server.
5. The default client for accessing web server is assumed to be browser, preferably firefox. You can use Chrome or any other browse as well.
6. The webserver in the example below is taken as 'myweb.com'. Please use your hostname or corresponding IP address instead in your exercise.
7. To kill any program in the linux terminal, please press **`Ctrl-C`** and not **`Ctrl-Z`**. The latter will suspend the program and not stop it.
8. It is assumed that your web server is running on a machine which is called `myweb.com`. For example, if web server is running 10.1.1.101, then on the client machine, create an entry for this IP Address for the hostname myweb.com in the file `/etc/hosts` (refer to appendix 7.3)

Note:

Appendix A provides instructions on installing Apache and other packages.
Appendix B provides sample programs to be used and deployed with Apache.

2 Hands-on 1: Basics of HTTP

The example programs mentioned below are listed in appendix. Refer to the same when there is a need to create such programs. You can any text editor e.g. `nano`, `gedit` or `vi` to create these text files.

2.1 Status Code 200

1. Create a simple webpage e.g. `welcome.html (8.1)`. Save this page in *DocumentRoot* directory of Apache web server. Typically, by default this value is `/var/www/html`.
2. Access the web page in the browser on a different machine i.e. `http://<serverIP>/welcome.html`
3. Verify the status code 200 and other required headers.
4. Access the same webpage using '`wget –d`' and verify the status code.
5. Access the same webpage using `nc` and verify the status code

```
nc -C myweb.com 80
GET /welcome.html HTTP/1.1
Host:  myweb.com
```

Note 1: The option `-C` (Capital) on Linux ensures that both CR and LF are sent after each line. On Apple Mac, use `-c` (small case) for the same.

Note 2: At times `nc` may result in web server responding with status code 400. This is because, web server does not get HTTP request line properly without CR/LF. Do the wireshark debugging to ensure that both CR and LF are sent after each line.

2.2 Content-Type

1. On webserver, Copy `welcome.html` file as `welcome.txt`
2. Access the url `http://myweb.com/welcome.txt`.
3. Look at the content displayed on browser.
4. Analyze the wireshark capture to study the header `Content-Type`:
5. Repeat the exercise with `wget`.
6. Study the headers.

2.3 Status code 404

1. On client, access a non-existent webpage e.g. `nonexist.html`
2. Check the status code in wireshark.
3. Verify this status code using `wget` as well.

2.4 Status code 403

7. On web server, copy `welcome.html` file as `restricted.html`
1. Remove the read permission for others i.e. issue the command
   ```
   chmod o-r retricted.html
   ```
2. Access this file using browser i.e.
   `http:/myweb.com/restricted.html`
3. Browser should display the contents of the file without formatting.
4. Analyze the response in the browser. Analyze HTTP status codes.

2.5 Status code 400

1. To experience this access code, we need to use `nc`. By default, both browser and `wget` send the proper HTTP header.
2. This exercise requires that client should send invalid header e.g. '`Host myweb.com`' instead of '`Host: myweb.com`'. Please note that HTTP header field name should be separated by its value by Colon (:) character.
3. Create a text file e.g. `req-badhdr.txt` (8.3) with bad headers.
   ```
   GET /accs/welcome.html HTTP/1.1
   Host 10.211.55.9
   User-Agent: IEEE Workshop
   ```
4. Send the bad headers using nc e.g.
   ```
   cat req-badhdr.txt | nc myweb.com 80
   ```
5. Analyze the response given by web server and verify that it corresponds to '400 Bad Request'.
6. Make another access with different header with syntax error.
7. Verify the Bad Request error

2.6 Status code 301

1. In the client machine, access google.com with wget debug option i.e
   a. `wget -d http://google.com`
2. This will display all debug headers.
3. Analyze the first response and it will correspond to HTTP Redirect with status code 301.

3 HTTP Persistent Connections

3.1 HTTP Non-Persistent Connections.

1. Configure Apache web server with `KeepAlive Off` and restart.
2. Create a web page (e.g. pictures.html) with multiple embedded images (say 10) images as in http://rprustagi.com/workshops/web/pictures.html
3. Access the web page from your local web server with `KeepAlive Off` in firefox browser and do a wireshark capture. How many TCP connections you notice. There should be as many connection as number of embedded objects plus 1.

3.2 HTTP Persistent Connections.

1. Configure Apache web server with following configurations.
   a. `KeepAlive On.`
   b. `MaxKeepAliveRequests 10`
   c. `KeepAliveTimeout 5`
2. Restart Apache.
3. Access the web page again with firefox browser. Analyze the number of TCP Connections. By default, firefox makes 6 concurrent TCP connections. You should see similar number and, on few connections, you should see two or more HTTP requests (e.g. images).
4. Configure Firefox to setup only 3 persistent connection.
   a. Type `about:config` in firefox browser.
   b. Search the field *`max-persistent-connections-per-server`*
   c. Set the value to 3.
5. Access the page again and analyze the number of TCP Connections.
6. Refresh the page after 5 seconds. Analyze setup of new TCP connections.
7. Refresh the web page within 4 seconds multiple times e.g. 10 times. Analyze the wireshark capture on when does a new TCP connection is made.
8. Tweak (or reconfigure) the value `MaxKeepAliveRequests` to your other values.
9. Continue to refresh the page multiple times less than configured timeout value (e.g. 5s). Analyze and understand when does a browser make a new TCP Connection.

4 HTTP Caching

4.1 Using Caching.

1. Using wget –d http://myweb.com/welcome.html, note down value of response header "`Last-Modified`:", as well as that of "`Etags`:"
2. Using wget header option, pass on the value corresponding to response header "Last-Modified" in previous step, e.g.
   `wget --header="If-Modified-Since: Fri, 26 Jan 2018 13:21:02 GMT"`
3. Analyze the response and verify that status is 304 and not 200.
4. On the web server modify the date/time of the file using touch command e.g.
   `touch welcome.html`
5. Make the request again using `wget` as in step 2.
6. You should get the full content with status code 200.

7. Modify the date/time of `welcome.html` file
8. Make another request using the header i.e. "`If-Non-Match:` " with the value corresponding to "`Etags:`" in the response as in step. e.g.
   ```
   wget -d  --header='If-None-Match: "d1-5689f483546d1"' http://myweb.com/accs/welcome.html
   ```
9. Analyze the response and HTTP Status code.\

5 HTTP Authentication and Dynamic Web Access

5.1 HTTP Authentication.

1. Modify apache config `/etc/apache2/sites-enabled/000-default.conf` file to specify a directory that requires authentication e.g
   ```
   <Directory /var/www/html/private>
        AuthType  Basic
        AuthName  "For Experiential Learning"
        AuthBasicProvider   file
        AuthUserFile   /etc/apache2/passwdfile
        Require valid-user
   </Directory>
   ```

2. Using the tool `htpasswd`, create a username and password for web authentication e.g.
   ```
   sudo htpasswd /etc/apache2/passwdfile ieee
   ```

3. Create the directory `/var/www/html/private` as specified in directive "`<Directory` " in the step 1.
4. Create a file e.g. restricted.html in this authentication directory
   ```
   sudo touch /var/www/html/private/restricted.html
   ```
5. Access the URL http://myweb.com/private/restricted.html.
6. Enter the username (e.g. ieee) but incorrect password. Understand the response.
7. Next time enter the correct username and password.
8. Analyze the wireshark capture. Compute your own value of base64 for username and password and compare it with the value in wireshark capture.

5.2 Dynamic web - success

1. Enable apache to use *cgi-bin* and restart e.g.
   ```
   sudo a2enmod cgi
   sudo service apache2 restart
   ```

2. Create executable program `goodcgi.sh` (8.7) in cgi-bin directory for apache web server e.g. `/usr/lib/cgi-bin.`
3. Access this URL e.g. http://myweb.com/cgi-bin/goodcgi.sh
4. Analyze the response and wireshark capture.

5.3 Dynamic web – Internal server.

1. Copy the file goodcgi.sh to badcgi.sh
   ```
   sudo cp goodcgi.sh badcgi.sh
   ```

2. Edit `badcgi.sh` (8.8) to comment the line which outputs empty line after the header. As a result, when this script is executed, there will not be separator empty line between HTTP header and web page content.
3. Access the URL http://myweb.com/cgi-bin/badcgi.sh corresponding to bad cgi program.
4. You should get the response as `500 Internal Server Error`.
5. Modify the badcgi.sh to uncomment the empty line output but insert a bad header format e.g. "Server myweb.com". Please note that do not put colon(:) between header name and value. This makes it as a bad header.
6. Re-access the URL and analyze the response.

6 HTTP Other features

6.1 Using Accept-Language.

1. Change the preferred language setting in firefox browser.
2. Access google.com.
3. The browser should display the content of web page.

6.2 Status code 301 (or 302)

1. Access `google.com` using `wget -d`. Identify the first response.
2. Make following changes in Apache config file `/etc/apache2/sites-enabled/000-default.conf`.

```
Redirect /oldwelcome.html /welcome.html
```

3. Restart Apache webserver (`sudo service apache2 restart`)
4. Access the url http://myweb.com/oldwelcome.html
5. Verify that content is served from the file `welcome.html`
6. Verify that status code 301 being returned, and second access to new url in the wireshark
7. Repeat the exercise using '`wget`' and verify HTTP redirect.
8. Analyze all HTTP headers and status code in wireshark

6.3 Using chunk based delivery

1. Create a php file like `chunk-xfer.php` (8.4) to generated HTTP Response in terms of chunks.
2. Copy your favourite image file (e.g. your own photo) and save it in `DocumentRoot`.
3. Ensure that your image filename is correctly described in the program file `chunk-xfer.php`
4. From the browser, access the URL http://myweb.com/chunik-xfer.php
5. Analyze the HTTP response in wireshark and identify chunks.
6. Edit the program file and change chunk sizes or mimic the internet delay i.e. sleep time.
7. Re-access the URL and analyze the response.

6.4 Status 206: Partial Content Delivery

1. Create a simple text `file req-partial.txt` (8.5) to send the HTTP headers requesting specific range of content.

2. Make a request to web server for partial content delivery i.e.
3. Analyze the HTTP response headers as well wireshark capture.

6.5 Using compression

1. To ensure that HTTP Request contains header corresponding to compression, create a simple text file e.g. `req-gzip.txt` (8.6) containing header `Accept-Encoding: gzip`.
2. Make a request using this header e.g.

```
cat req-gzip.txt | nc myweb.com 80 >abc.html.gz
```

3. Analyze the response. Uncompress (`gunzip abc.html.gz`) it to get the original contents.
4. Make a request using wget for HTTP response with compression. Use the `wget` option `--header` to make such a request.

```
wget --header="Accept-Encoding: gzip"
http://myweb.com/welcome.html -O welcome.html.gz
```

5. Analyze the response as well as wireshark capture.

Appendix A

7 A: Installing Apache and other programs

7.1 A1: Installing Apache

```
$ sudo apt-get install apache2
```

7.2 A2: Installing Wireshark

```
$ sudo apt-get install wireshark
```

Running wireshark

```
$ sudo wireshark
```

7.3 A3: Overcoming DNS resolution.

To ensure that your webserver name myweb.com is resolved to a proper IP address in the client machine, make an entry in /etc/hosts file of client machine like below.

```
10.1.1.101      myweb.com
```

7.4 A4: Enabling php in Apache

```
$sudo apt-get install php5 libapache2-mod-php5
$ sudo service apache2 restart
```

8 B: Sample Programs

8.1 B1: welcome.html

```
<html>
    <head>
        <title>Welcome Page</title>
    </head>
    <body>
        <h1>Welcome to HTTP Learning</h1>
            Welcome to experiential learning of
HTTP protocol.
    </body>
</html>
```

Note: It can be downloaded from
`http://rprustagi.com/workshops/web/welcome.html`

8.2 B2: req-normal.txt

Note 1: There should be an empty line at the end, which is indicated by ↵
Note 2: Unix by default uses \n (newline) character to separate two lines whereas HTTP protocol requires that each line be separate by CRLF i.e. \r\n. Thus, on linux (as well as Macbook) append Ctrl-M character at the end to be used as input to nc. Use the utility `unix2dos` to append Ctrl-M character at the end of each line.

```
GET /welcome.html HTTP/1.1
Host: myweb.com
```

```
User-Agent: IEEE Workshop
Accept: */*
Accept-Language: en-us,en;q=0.5
↵
```

Note: It can be downloaded from
http://rprustagi.com/workshops/programs

8.3 B3: req-badhdr.txt

```
GET /accs/welcome.html HTTP/1.1
Host 10.211.55.9
User-Agent: IEEE Workshop
Accept: */*
Accept-Language: en-us,en;q=0.5
↵
```

Note: It can be downloaded from
http://rprustagi.com/workshops/programs

8.4 B4: chunk-xfer.php

```
$file = '../img/img-07.jpg';
if (is_file($file)) {
    header('Content-Type: image/jpeg');
    header('Transfer-Encoding: chunked');

    $chunkSize = 1000;
    $handle = fopen($file, 'rb');
    while (!feof($handle)) {
        $buffer = fread($handle, $chunkSize);
        # send chunk size in hex, chunk content,
empty new line
        echo sprintf("%x\r\n", $chunkSize);
        echo $buffer; echo "\r\n";
        ob_flush(); flush();
        usleep(500000); # emulate network latency
    }
    fclose($handle);
    exit;
} else {
    header('Content-Type: text/html');
    echo "\r\nNo picture available for $file\r\n";
}
?>
```

Note: It can be downloaded from
http://rprustagi.com/workshops/web

8.5 B5: req-partial.txt

```
GET welcome.html HTTP/1.1
Host: myweb.com
User-Agent: IEEE Workshop
Accept: */*
```

```
Accept-Language: en-us,en;q=0.5
Range: bytes=50-100
↵
```

Note: It can be downloaded from
http://rprustagi.com/workshops/programs

8.6 B6: req-gzip.txt

```
GET welcome.html HTTP/1.1
Host: myweb.com
User-Agent: IEEE Workshop
Accept: */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip
↵
```

Note: It can be downloaded from
http://rprustagi.com/workshops/programs

8.7 B7: goodcgi.sh

```
#!/bin/bash

# program logic
echo "Content-type: text/plain";
echo "";
echo "<h1>hello! welcome</h1>";
```

Note: It can be downloaded from
http://rprustagi.com/workshops/cgi-bin

8.8 B8: badcgi.sh

```
#!/bin/bash

# program logic
echo "Content-type: text/plain";
#echo "";
echo "<h1>hello! welcome</h1>";
```

Note: It can be downloaded from
http://rprustagi.com/workshops/cgi-bin

← end of exercises handout →