# Assignment 6: The HTTP Protocol

Athabasca University - COMP 470
Stephanie Petrone
January 12, 2023

(1) **GET /**

This command causes the website to return the default html file (index.html) which is expected. It did not request any specific file, so the server automatically provides the default file. This is the equivalent of typing in the host name with no additional information after the domain extension.

**Here are the results returned by the server:**

---

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
<meta charset="utf-8">
<title>Home Page for the Website</title>
</head>
<body>
<h1>Welcome to the home page for Mycelii.com!</h1>
</body>
</html>
```

Look at the resulting code (just enough to see what has been returned, no need to understand the mangled mess of HTML code that may have been returned). If you were a web browser, what would you do next? What problems might you face? Hint—pictures? Another hint—what if you wanted to get the page again? Is there any way you could avoid having to get the whole file?

The web browser will receive the contents of the web page from the server in the body HTTP response along with all the metadata needed to render it properly. The browser will render the DOM elements of the html file. Within that file, there may be embedded files, such as images, JavaScript or css files. Only the URLs to these files are sent in the response, as HTTP responses only return the contents of a single file. So, for all of the files within the web page that is being rendered, the web browser will have to make additional HTTP GET requests to the server using the URLs embedded in the html. The web browser will create a new, separate request for each of these files, whether they are images, scripts, stylesheets, etc.

To avoid retrieving the whole files subsequently (i.e. files that have already been loaded by the browser), caches and proxies can be used. Files can be stored in a web browser's cache (which is essentially making the host a proxy), or the web browser may request the file from a proxy first. Proxies can be on the client-side or server-side, and their purpose is to make it so that HTTP requests don't have to repeatedly go to the web server for the same files. An browser using a cache or on a network configured to use a proxy will first search the cache or send the request to the proxy to see if the file is already there, and it is there and recent enough, it will load it from there without having to send an HTTP request to the server.

(2) 
```
GET / HTTP/1.1
Host: localhost
Accept: */*
```
This command does an HTTP 1.1 GET request, accepting all file types to the host (localhost here). Again, it does not specify a file so the default file is expected, which for this server it index.html in the /var/www/html file (document root for the server). This is what is received.

```
Here are the results returned by the server:
________________________________________________________________________
HTTP/1.1 200 OK
Date: Sun, 27 Nov 2022 06:19:20 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Tue, 22 Nov 2022 08:06:13 GMT
ETag: "d8-5ee0aa544d2db"
Accept-Ranges: bytes
Content-Length: 216
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
<meta charset="utf-8">
<title>Home Page for the Website</title>
</head>
<body>
<h1>Welcome to the home page for Mycelii.com!</h1>
</body>
</html>
```

And press return twice (why not once, as before?).
The return twice ensures that there is a carriage return and line feed so the web server knows when the body of a request ends and the start of a new header begins. This is part of the formatting for HTTP 1.1 packets. They are needed in HTTP 1.1 since requests contain multiple lines with bodies. Since HTTP 0.9 is a one-line protocol, this isn't needed as the server only needs to know when the end of the line has been reached. In HTTP 1.1 carriage returns and line feeds separated lines and and the body components of the packet.

What differences do you notice?

The difference is that there are many more attributes in the header, which is because HTTP 1.1 was used rather than HTTP 0.9 like the previous example and it involves much more information in the header, including information about the server. Additionally, there is a body to the response packet as well.

**(3)**

```
GET /testDir HTTP/1.1
Host: localhost
Accept: */*
```

This requests a file called testDir (a directory created on the server's document root). However, testDir is not a file. The result is a 301 error saying that the file has moved permanently. It is expected when requesting a directory, that the directory's file structure would be returned as the index page in html, however this is lacking an extra "/" at the end that would define it as a directory so the server does not handle it properly, thinking that it is a file.

**Here are the results returned by the server:**

---

```
HTTP/1.1 301 Moved Permanently
Date: Sun, 27 Nov 2022 07:05:22 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Location: http://localhost/testDir/
Content-Length: 233
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://localhost/testDir/">here</a>.</p>
</body></html>
```

This is the equivalent of typing http://localhost/dirname into your browser's location box - you might want to try this to see what happens, so that you can compare the results.

What happens in the web browser is that the index page listing files in the directory is provided in the web browser. It seems that the browser automatically appends the trailing slash, because the directory index page is retrieved rather than a 301 error and the slash gets added in the search bar. Potentially, though, this is happening on the server side, when no file called testDir existed, it searched for a directory and returned the index page of the directory instead.

What is the server telling you now? What should you have typed? Try it. Remember this lesson when accessing websites in future.
The server is telling you that this is not the correct location to search for the directory. What should have been typed was `GET /testDir/ HTTP/1.1` (notice the extra /).

**(4)**

```
GET / HTTP/1.1
Host: employee.mycelii.com
Accept: */*
```

I actually made the document root for a virtual host protected which is employee.mycelii.com. So the request has no specific file request but the host is specified to be the virtual host at employee.mycelii.com. As expected, the HTTP request returned 401 Unauthorized because the page requires credentials to access and none are given (and with this method of sending a GET request without a browser, there is no browser to prompt the user and send another GET request with the credentials).

**Here are the results returned by the server:**

---

```
HTTP/1.1 401 Unauthorized
Date: Sun, 27 Nov 2022 07:13:44 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
WWW-Authenticate: Basic realm="Restricted Content"
Content-Length: 381
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Unauthorized</title>
</head><body>
<h1>Unauthorized</h1>
<p>This server could not verify that you
are authorized to access the document
requested. Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.</p>
</body></html>
```

Next, the same request is done, but with encoded authorization credentials for basic HTTP authorization:

```
GET / HTTP/1.1
Host: employee.mycelii.com
Accept: */*
Authorization: Basic dXNlcjE6cGFzc3dvcmQ=
```

As expected, this time, the page was accessed with an HTTP code 200 (OK) and the default file index.html was returned for the virtual host.

**Here are the results returned by the server:**

---

```
HTTP/1.1 200 OK
Date: Sun, 27 Nov 2022 08:13:30 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Tue, 22 Nov 2022 08:50:57 GMT
ETag: "e0-5ee0b4546dd02"
Accept-Ranges: bytes
Content-Length: 224
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
<meta charset="utf-8">
<title>Welcome to the Employee Virtual Server</title>
</head>
<body>
<h1>Success! Virtual host for employees!</h1>
</body>
</html>
```

***Why do you think that HTTP requires the username and password to be encoded?*** **Hint: it has almost nothing to do with security. If that is puzzling you, try to think about the things that would make a well-chosen password secure.**
A well-chosen password would have letters, numbers and special characters. The base64 encoding is done to ensure that special characters end up being encoded in ASCII.

**(5)**

```
GET /nonexistantfile.html HTTP/1.1
Host: localhost
Accept: */*
```

This command, as expected, returns an HTTP code 404, which means that the file was not found. I think this is a code that pretty much anyone who uses the internet has seen many times before when you've clicked on a broken link.

```
HTTP/1.1 404 Not Found
Date: Sun, 27 Nov 2022 08:24:26 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Content-Length: 196
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>
```

**(6)**

```
GET /image1.jpg HTTP/1.1
Host: localhost
Accept: */*
```

As expected, this command returns a code 200 for OK because the file exists there (I just put it there) and it returns the metadata for the file. In the browser, it opens the image.

**Here are the results returned by the server:**

---

```
HTTP/1.1 200 OK
Date: Sun, 27 Nov 2022 23:05:58 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Sun, 27 Nov 2022 23:05:19 GMT
ETag: "113a8-5ee7bc9ead5e1"
Accept-Ranges: bytes
Content-Length: 70568
Content-Type: image/jpeg
```

*Changing the Accept field to text/*::*

```
GET /image1.jpg HTTP/1.1
Host: localhost
Accept: text/*
```

**Here are the results returned by the server:**

---

```
HTTP/1.1 200 OK
Date: Sun, 27 Nov 2022 23:16:09 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Sun, 27 Nov 2022 23:05:19 GMT
ETag: "113a8-5ee7bc9ead5e1"
Accept-Ranges: bytes
Content-Length: 70568
Content-Type: image/jpeg
```

This returns the same 200 OK code which is not expected. I would expect an HTTP 406 error code which communicates that the MIME type is mismatched / unacceptable for the request. So, I looked through the configuration files. I did not find any issues in the httpd.config or the mime.types file so I cannot figure out where the issue is. I wonder if it is in the magic file. Even when specifying text/html, it still returns the image with the status code 200.

**(7)**

```
HEAD /page1.html HTTP/1.1
Host: localhost
```

**Here are the results returned by the server:**

---

```
HTTP/1.1 200 OK
Date: Mon, 28 Nov 2022 00:11:06 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Mon, 28 Nov 2022 00:08:38 GMT
ETag: "d6-5ee7cac5c4040"
Accept-Ranges: bytes
Content-Length: 214
Content-Type: text/html; charset=UTF-8
```

This command returns the page metadata for the test page I created (page1.html) but it does not return the file contents, in this case html, like the GET command does. The ETag is “d6-53337cac5c4040”. Upon another request with the same command, it is the exact same ETag - all the same information is returned except for the time stamp for the Date field showing that it was cached.

I then went and made a slight change to the file, adding another <p> tag to it. This was the result:

```
HTTP/1.1 200 OK
Date: Mon, 28 Nov 2022 00:14:07 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Mon, 28 Nov 2022 00:14:03 GMT
ETag: "fa-5ee7cbfb6a2ea"
Accept-Ranges: bytes
Content-Length: 250
Content-Type: text/html; charset=UTF-8
```

The ETag has changed as well as the content length (since I added to the file) along with the last-modified date.

The HEAD request shows just the header information without the body of the GET request, making for more lightweight requests but still with all the information about the web server. There are web crawlers that send these requests to devices connected to the internet and make the information accessible/searchable in search engines (ZoomEye, Shodan, Censys).

**(8)**

I actually already played with the host field during this assignment because for question 4 I needed to access a virtual host because it was the document root for the virtual host employee.mycelii.com that I had protected with http authentication. For this exercise, I will try out those commands with employee.mycelii.com and monitor.mycelii.com, my two virtual hosts.

For employee.mycelii.com, I got the same response I did for question 4 (when authorization credentials were not supplied):

**Here are the results returned by the server:**

---

```
HTTP/1.1 401 Unauthorized
Date: Mon, 28 Nov 2022 00:20:22 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
WWW-Authenticate: Basic realm="Restricted Content"
Content-Length: 381
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Unauthorized</title>
</head><body>
<h1>Unauthorized</h1>
<p>This server could not verify that you
are authorized to access the document
requested. Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.</p>
</body></html>
```

And this is for monitor.mycelii.com:

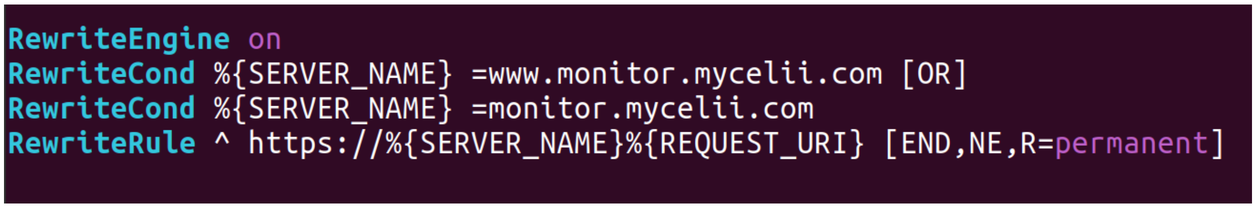
**Here are the results returned by the server:**

---

```
HTTP/1.1 301 Moved Permanently
Date: Mon, 28 Nov 2022 00:21:20 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Location: https://monitor.mycelii.com/
Content-Length: 236
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="https://monitor.mycelii.com/">here</a>.</p>
</body></html>
```

This shows an error 301 code. This is because in the <VirtualHost> directive for this virtual host I have this:



```
RewriteEngine on
RewriteCond %{SERVER_NAME} =www.monitor.mycelii.com [OR]
RewriteCond %{SERVER_NAME} =monitor.mycelii.com
RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
```

This is to force the use of TLS by redirecting to HTTPS and it was a recommendation in the guide I followed for setting up SSL/TLS to force the use of HTTPS rather than HTTP. At the beginning of this assignment I commented this out for the <VirtualHost> directive for mycelii.com because I was getting this error code which was obviously not going to work for this assignment. I do not have employee.mycelii.com setup with an SSL certificate and requiring HTTPS (in production it would though!) so its directive doesn't have these which is why it worked initially. So now, with those lines commented out for monitor.mycelii.com and the server restarted this is what is returned:

**Here are the results returned by the server:**

---

```
HTTP/1.1 200 OK
Date: Mon, 28 Nov 2022 00:27:22 GMT
Server: Apache/2.4.54 () OpenSSL/1.0.2k-fips
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Tue, 22 Nov 2022 07:17:19 GMT
ETag: "db-5ee09f6670793"
Accept-Ranges: bytes
Content-Length: 219
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
<meta charset="utf-8">
<title>Welcome to the Monitor Page</title>
</head>
<body>
<h1>Success! Virtual host for Monitoring Farm!</h1>
</body>
</html>
```

**(9)**

```
GET /phpMyAdmin/ HTTP/1.1
Host: mycelii.com
Accept: */*
```

Excerpt from what the server returned:

```
Vary: Accept-Encoding
Set-Cookie: pma_lang=en; expires=Wed, 28-Dec-2022 00:49:22 GMT; Max-Age=2592000;
path=/phpMyAdmin/; SameSite=Strict; HttpOnly
Set-Cookie: phpMyAdmin=gpeh4661ca6fefl9icrmpsjq6b; path=/phpMyAdmin/;
SameSite=Strict; HttpOnly
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Mon, 28 Nov 2022 00:49:22 GMT
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
```

I used the phpMyAdmin php application to see what cookies are being sent in the header. This one sends two cookies: pma_lang=en and phpMyAdmin=gpeh4661ca6fefl9icrmpsjq6b

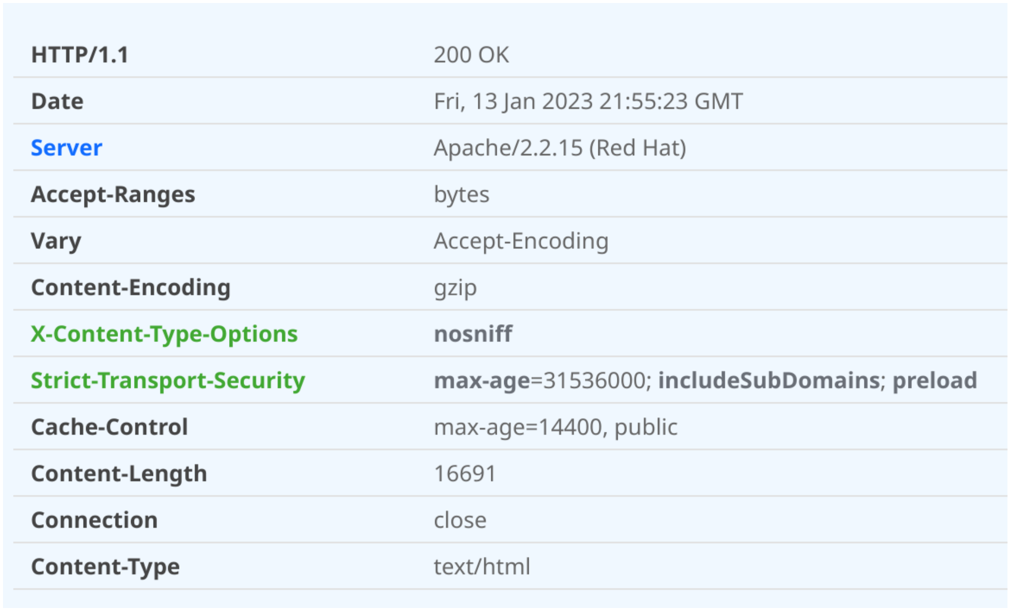
```
GET /phpMyAdmin/ HTTP/1.1
Host: mycelii.com
Accept: */*
Cookie: pma_lang=fr;
```

The command above has the added line: Cookie: pma_lang=fr; The response from the server no longer includes the Set-Cookie field attribute for that cookie. The contents of the page are then also in French, so evidently sending the cookie value for the language as french (fr) causes the phpMyAdmin login page to be in French and so the server used the value of the cookie.

## Other Applications:

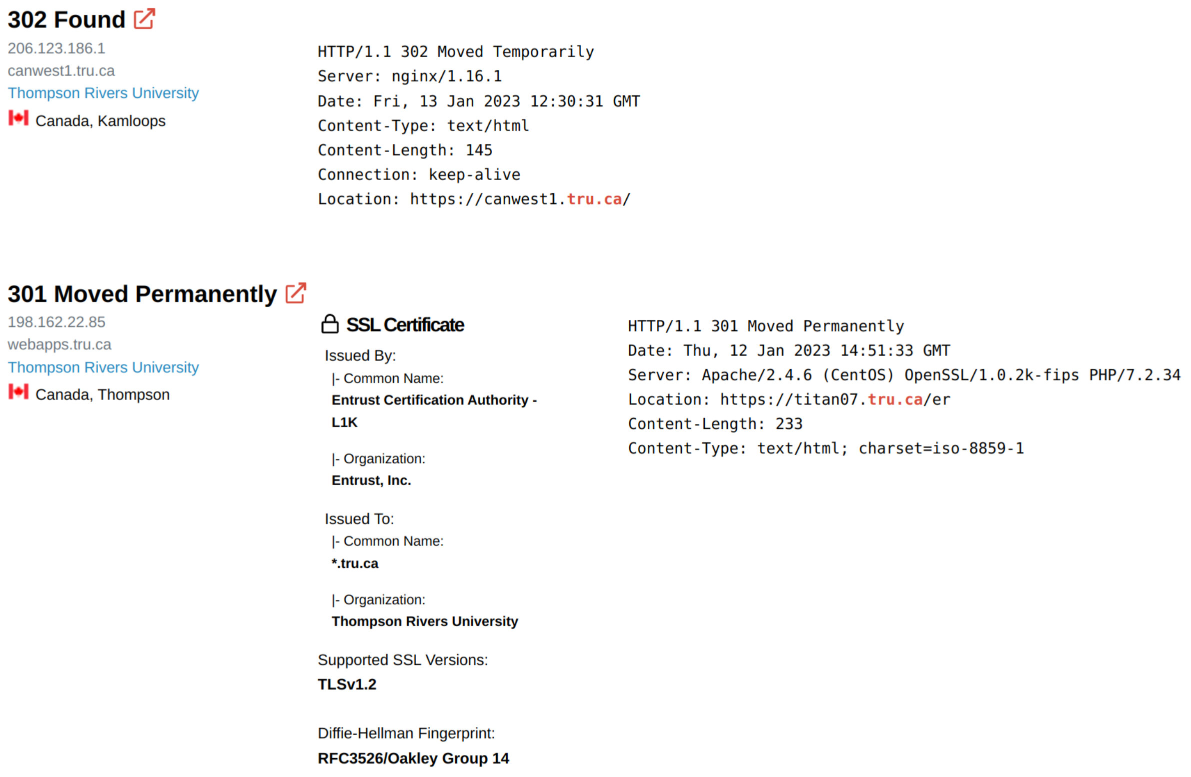
There are other applications showing http headers. I did not want to install a browser extension that reads headers as I do not trust those applications. I instead used some web applications that check headers, mostly related to security checks. They all show similar information.

This is a response from **securityheaders.com** for my home university's website tru.ca:



| | |
|---|---|
| **HTTP/1.1** | 200 OK |
| **Date** | Fri, 13 Jan 2023 21:55:23 GMT |
| **Server** | Apache/2.2.15 (Red Hat) |
| **Accept-Ranges** | bytes |
| **Vary** | Accept-Encoding |
| **Content-Encoding** | gzip |
| **X-Content-Type-Options** | **nosniff** |
| **Strict-Transport-Security** | **max-age**=31536000; **includeSubDomains**; **preload** |
| **Cache-Control** | max-age=14400, public |
| **Content-Length** | 16691 |
| **Connection** | close |
| **Content-Type** | text/html |

Here is what Shodan retrieves for tru.ca. It shows header and certificate information:



**302 Found**

206.123.186.1
canwest1.tru.ca
Thompson Rivers University
Canada, Kamloops

```
HTTP/1.1 302 Moved Temporarily
Server: nginx/1.16.1
Date: Fri, 13 Jan 2023 12:30:31 GMT
Content-Type: text/html
Content-Length: 145
Connection: keep-alive
Location: https://canwest1.tru.ca/
```

**301 Moved Permanently**

198.162.22.85
webapps.tru.ca
Thompson Rivers University
Canada, Thompson

**SSL Certificate**

Issued By:
|- Common Name:
**Entrust Certification Authority - L1K**

|- Organization:
**Entrust, Inc.**

Issued To:
|- Common Name:
***.tru.ca**

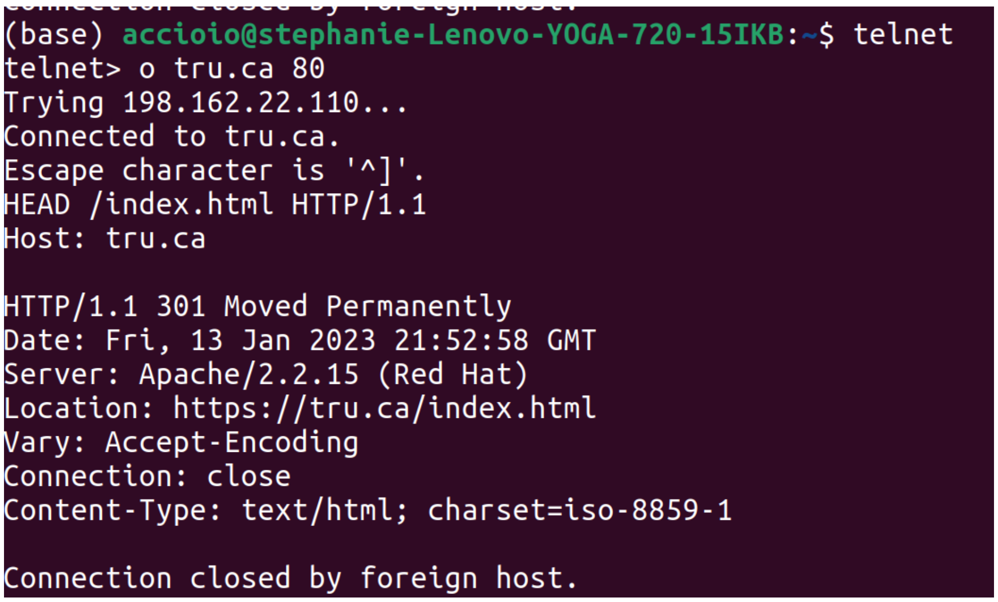
|- Organization:
**Thompson Rivers University**

Supported SSL Versions:
**TLSv1.2**

Diffie-Hellman Fingerprint:
**RFC3526/Oakley Group 14**

```
HTTP/1.1 301 Moved Permanently
Date: Thu, 12 Jan 2023 14:51:33 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.2.34
Location: https://titan07.tru.ca/er
Content-Length: 233
Content-Type: text/html; charset=iso-8859-1
```

Using telnet to get header information for tru.ca shows the following:



```
(base) accioio@stephanie-Lenovo-YOGA-720-15IKB:~$ telnet
telnet> o tru.ca 80
Trying 198.162.22.110...
Connected to tru.ca.
Escape character is '^]'.
HEAD /index.html HTTP/1.1
Host: tru.ca

HTTP/1.1 301 Moved Permanently
Date: Fri, 13 Jan 2023 21:52:58 GMT
Server: Apache/2.2.15 (Red Hat)
Location: https://tru.ca/index.html
Vary: Accept-Encoding
Connection: close
Content-Type: text/html; charset=iso-8859-1

Connection closed by foreign host.
```