Lab3: Peeking under the Web

COM-208: Computer Networks

The main goal of this lab is to "peek under the web", i.e., get a sense of how web browsers and web servers communicate: we will sniff and look inside **HTTP messages**, then play around with **caching** and **cookies**. We will close with a glance at two other applications that you use all the time: **email** and **streaming**.

Inside HTTP

HTTP (Hypertext Transfer Protocol) is the communication protocol used by the web application: it specifies the messages that may be exchanged between web clients (also called web browsers) and web servers.

The most common exchange is one where a web browser requests a resource, and a web server sends it; you will now initiate such an exchange and look inside the resulting messages. For this, you will use the **telnet** utility, which allows you to type in messages and send them to a remote process, i.e., a process running on a remote computer. Remember: there is only one process, behind each network interface, that is associated with a given port number; and all processes in the world that are associated with port number 80 are web-server processes.

Open a terminal and type:

```
telnet example.com 80
```

By typing this, you indicated that you want to communicate with the process that has name "example.com, 80" (network interface example.com, port number 80). As a result, the TCP code running on your computer generated a TCP connection setup request and sent it to the TCP code running on example.com; if that code accepted your request and sent a response, you got back a prompt, where you can type in messages that will be sent to the web-server process running on example.com.

Type:

```
GET / HTTP/1.1
host: example.com
[press <return> twice]
```

You just manually created an HTTP request. You specified that:

- the type of your message is GET,
- the resource you want is "/" (the home page of example.com),
- the version of the HTTP protocol you are using is 1.1,
- and the origin web server who owns the resource you want is example.com.

By pressing twice, you told the `telnet` utility that your message was ready to be sent.

What you just did manually is what your web browser does (among other things) under the covers when you type in a URL.

If all went as it should, you received an HTTP response from the web server process, which contains the base file of the target resource. If the HTTP request had been sent by your web browser (not by you via `telnet`), it is your web browser that would have received the response; it would have processed it, retrieved the URLs of all the referenced resources, and sent a new HTTP GET request for each resource.

- What is the content type and size of the HTTP response?
- Use the same approach to get the same file, but make your request of type HEAD (instead of GET). How does the HTTP response differ?
- Send a GET request like the first one, but for resource `index.html` (instead of `/`).

Caching at the web browser

Web browsers **cache** resources, so that they don't need to download them again if the user requests them again. You will now experience the difference this browser behavior can make.

Use a Firefox web browser, if you can. It comes with a nice tool, the web-developer network console (≡/Web Developer/Network), which visualizes each HTTP request that the browser makes, as well as the correspoding HTTP response that the browser receives. If you click on an HTTP request from the list on the left, you will see all the relevant information in the panel on the right.

Get ready to capture web traffic:

- Open your web browser and clear the cache. To do so in Firefox:
  ≡`/Preferences/Privacy & Security/Cookies and Site Data/Clear Data...`

- In Firefox, open the web-developer network console.
- Open Wireshark and start a new traffic capture.

Answer the following questions, using the web-developer network console, or Wireshark, or (ideally) both. Using Wireshark is a bit harder this time, but we will guide you:

- Visit Welcome to Rio. Where is this resource downloaded from?
- How long did it take to download it?

The web server where this resource is downloaded from uses a secure version of the HTTP protocol called HTTPs, which is, essentially, HTTP on top of SSL (the Secure Sockets Layer that we mentioned in class). This makes using Wireshark a bit harder: HTTP messages are encrypted within SSL packets, so Wireshark cannot simply display them. You need to:

- Apply the `ssl` filter to see all the SSL packets sent or received by your computer.
- Identify one of the packets sent by you to the web server or vice versa.
- Click on it, then go to Analyze/Follow/TCP Stream. Ignore/close the window that pops up. Now you should see only the packets that belong to the same TCP connection as the packet you chose.
- Look for the last SSL packet carrying encrypted `Application Data`. This is the packet that carried the resource from the web server to your computer.
- Look for an SSL packet of type `Client Hello`. This is the first packet that your computer sent to the web server to initiate their communication.
- Restart your web browser. Visit Welcome to Rio again. How long did it take to load it this time? What explains the difference?
- Open a second tab in your web browser and visit Welcome to Rio II. Where is this resource downloaded from? How long did it take to download it?
- When you visited `Welcome to Rio II`, your web browser had already cached `Welcome to Rio`, which is essentially the same image. Do you think your browser served `Welcome to Rio II` from the cache, or it downloaded it from its origin web server? Why do you think your browser behaved this way?

Caching at a proxy web server

It is not only web browsers that cache resources; **proxy web servers** are web servers that act as **intermediaries**: they cache resources that are originally stored in other web servers (called **origin web servers**) and serve them to nearby web clients.

Before you start, clear your browser cache and find the proxy settings of your web browser. In Firefox, navigate to `about:preferences`, then `Network Settings/Settings`. Setup a proxy web server using the following settings: `HTTPS Proxy: 51.75.147.33,` ↪ `Port: 3128`. (you could use any proxy web server from https://free-proxy-list.net/ that does HTTPS caching).

Visit the same two resources that you visited before.

- Where were the resources downloaded from?
- How long did it take to download each resource this time? Why did the download time change?
- What will happen to your web browser if the proxy web server that you specified fails? Will your browser be able to load any web pages? Can you think of a way to verify your answer?

IMPORTANT: Restore your original proxy settings.

Cookies

Cookies enable a web server to **link subsequent HTTP requests** to the same web browser: if you send 10 HTTP GET requests, for 10 different resources, to the same web server, the web server can use cookies to figure out that these 10 requests came from the same web browser, even if you did not explicitly provide any identification information (e.g., you did not login).

Before you start, figure out how to control cookie settings in your browser. In Firefox:

- To allow or disallow your browser to exchange cookies with web servers: ≡`/Preferences/Privacy & Security/Enhanced Tracking Protection/Custom`, and then uncheck or check blocking `Cross-site and social media trackers`.
- To view or delete the cookies that have been stored on your computer: ≡`/Preferences/Privacy & Security/Cookies and Site Data/`, and then `Manage` ↪ `Data` or `Clear Data...`

- You can also view the cookies that your computer sends along with an HTTP request, or receives along with the corresponding response, through the web developer network console: select an HTTP request from the list of requests on the left, then select the `Cookies` menu from the panel on the right.

First, see cookies in action:

- Allow your browser to exchange cookies. Delete existing cookies. Visit MeteoSuisse. Did the MeteoSuisse web server send you any cookies?
- By default, MeteoSuisse shows you the weather for Geneva. Choose another location for which you want to see the weather. Restart your web browser and re-visit MeteoSuisse. Do you get the weather for Geneva as before? Explain your browser's behavior.
- Delete existing cookies. Restart your web browser and re-visit MeteoSuisse. Do you get the weather for Geneva or for your chosen location? Explain your browser's behavior.

Now think about **cookies as state**, as information about the user that can be exchanged between third parties:

- Visit Google. Once the resource has finished loading, view the cookies that have been stored on your computer. How many are they? Notice that each set of cookies is associated with a "site" or "domain", e.g., `google.ch`, or `youtube.com`. Which web server sent each of these cookies to your web browser?
- Visit YouTube. Did your web browser send along any cookie when it contacted the YouTube web server? Which one?
- View again the cookies that have been stored on your computer. How do you think your web browser decides which cookie(s) to send along with each HTTP request?

Think about your web browser's communications. Did the Google and YouTube web servers just exchange information about you without talking to each other directly?

IMPORTANT: Restore your original cookie settings.

Inside an email server (or: where spam comes from)

Web clients and web servers communicate through the HTTP protocol; email clients and email servers communicate through another application-layer protocol, called SMTP (Simple Message Transfer Protocol).

At the beginning of this lab, you used `telnet` to "talk directly" to a web-server process, as if you were a web client (web browser); now you will use `telnet` to "talk directly" to

an email-server process, as if you were an email client. So, instead of manually creating HTTP requests, you will manually create email messages (not just the subject and body of the email, but also the headers).

To create email messages, you need to learn some of the language that an SMTP email server understands. You can find an example of SMTP use on Wikipedia and another one on the Microsoft Exchange mail server documentation. If you are curious, the SMTP protocol is specified in RFC 2821 (you don't need to read the whole thing in order to finish he lab).

If you are doing the lab on your own computer, and you are not connected to the EPFL network, make sure you are connected to the EPFL VPN server.

- To connect to an email server process, you need to know the port number that is associated with email-server processes. Which one is it?
- Use `telnet` to connect to the email-server process running on `mail.epfl.ch`. Send a message from `bill.gates@microsoft.com` to your own email address.
- If all went as expected, you should have received your email (check your spam folder, just in case). Who appears to be the sender of the email? Figure out how your email client displays email headers and view the headers of this email. Is there anything in the headers that should make you suspicious about the sender of the message?

  As someone famous said: "With great power comes great responsibility."

Don't use your newfound knowledge to spam others, even if it's just for a joke.

- Last year, some of your colleagues sent around email messages that appeared to be coming from the EPFL president. As a result, they had their EPFL accounts suspended (and Katerina had some explaining to do). How do you think the EPFL sysadmins figured out who sent the fake messages? Couldn't we do something like that to stop all spammers?
- What does the email server do if you give commands in the wrong order, e.g., "rcpt to" before "mail from"? What does it do if you give a command that is not part of the SMTP protocol, e.g. "bonjour" instead of "helo"?
- In the old days, an email client could use the VRFY command to verify that an email address was valid. Ask the EPFL email server to verify `katerina.argyraki@epfl.ch`. What does the email server say? Can you guess why?

Back to layers, headers, encapsulation...

Open this webpage in Firefox and start playing the music (it might play automatically): https://soundcloud.com/relaxdaily/instrumental-music-to-relax

Open the web-developer network console and find the audio stream (you can filter "Media" streams from the button bar). Start a Wireshark capture.

- Which application-layer protocol carries the audio messages?
- What is the content type and content size?
- Which transport-layer protocol encapsulates the audio messages?