Jericho Tang

**Q1.** Two main differences between the preview I see and what I see in a normal web browser is that my HTML file does not have any text or website formatting, and it also lacks any images whatsoever. (The links also do not work.)

This can be explained by the fact that although I have the HTML of watchout4snakes’ home page saved, I do not have any of its auxiliary files (i.e. CSS files containing formatting instructions, images, scripts, etc.) saved in my directory, so when I open my saved HTML file in the browser it has no formatting instructions or images to render.

**Q2.** the command I used in terminal was:

curl -X GET **<http://localhost:8081/wo4s_test.html>**

where wo4s\_test.html was the file where I saved the watchout4snakes webpage’s HTML. The “http://” is there as the standard opener for a URL, “localhost” to indicate the file is on my computer, “:8081” to indicate which port I am connecting to, and “/wo4s\_test.html” to indicate what file I want to retrieve.

When running this command, on the server’s side I see:

GET /wo4s\_test.html HTTP/1.1

Host: localhost:8081

User-Agent: curl/7.55.1

Accept: \*/\*

**Q3.** I believe you don’t see “**<https://localhost:8081>**" anywhere in the first line of the request because you have already connected with the server (localhost:8081) so at this point keeping the information is redundant, and you only need to look at the commands that come after that.

**Q4.** After running the command “curl -i '**<http://watchout4snakes.com>**' ”:

HTTP/1.1 200 OK

Cache-Control: private

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

Server: Microsoft-IIS/7.0

X-AspNetMvc-Version: 3.0

X-AspNet-Version: 4.0.30319

X-Powered-By: ASP.NET

Date: Thu, 19 Sep 2019 07:27:02 GMT

Content-Length: 3571

It looks like the server used HTTP 1.1 to respond to the request, and the response code was 200.

**Q5.** The header section “content-type” specifies what is being returned; in this case it’s returning a file that can be read as text or rendered as HTML, and it is encoded in utf-8.

**Q6.** The heading when asking for a page that doesn’t exist is as follows:

HTTP/1.1 404 Not Found

Content-Type: text/html

Server: Microsoft-IIS/7.0

X-Powered-By: ASP.NET

Date: Thu, 19 Sep 2019 08:17:51 GMT

Content-Length: 1549

As stated in the first line, you would get error 404.

From Wikipedia: 1xx status codes are informational and mean the request was received by the server and are continuing to be processed; 2xx indicate successfully received and processed requests; 3xx are redirects; 4xx indicate an error in the client’s request; and 5xx indicate an error on the server’s side, being unable to fulfill an otherwise valid request.

**Q7.** As stated above, 4xx status codes indicate a “bad request” from the client’s side, such as asking for a resource that doesn’t exist, or syntax errors in the request, or other such cases. 5xx status codes, however, indicate errors on the server’s side; the server is unable to fulfill the request even though it is otherwise valid.

**Q8.** Replaced the fake url with **<http://localhost:8081/test_form.html>** where 8081 is the port and test\_form.html is the file where I saved the template.

**Q9.** The information presented to the server was as follows:

POST /test\_form.html HTTP/1.1

Host: localhost:8081

Connection: keep-alive

Content-Length: 88

Cache-Control: max-age=0

Origin: null

Upgrade-Insecure-Requests: 1

DNT: 1

Content-Type: application/x-www-form-urlencoded

…

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3

…

email=testemail%40doma.in&password=fakepassword&secret\_info=secret\_value&login=Log+In%21

The information entered into the form was sent at the very bottom in the format [variable name]=[variable value], and the & symbol separating different inputted values. A SaaS framework should parse the string and assign the variable values to the correct variable names in whatever language is being used for the framework.

**Q10.** I can add additional values by specifying more input types in the HTML file. For example, adding “<input type="hidden" name="anotherparam" value=“stuff”>" will add “&anotherparam=stuff” to the end of the string of parameters the server receives.

**Q11.** Changing the “name” parameter in the HTML input tags will change the name of the variable when sending parameters to the server. For example, changing “<input type="hidden" name="secret\_info" value=“secret\_value”>" to “<input type="hidden" name="i\_was\_secret\_info" value=“secret\_value”>" will make it so that the parameters received by the server change from “secret\_info=secret\_value” to “i\_was\_secret\_info=secret\_value”.

**Q12.** Adding a second submit button: suppose you have one submit button with the name “login” and value “Log In!” and a second button with the name “submit2” and value “Second submit:)”. If you click the first submit button, the parameters string will have “&login=Log In!” included in the string, but not “submit2=Second submit:)”; if you click the second submit button, you will have the latter but not the former. Basically you can tell which submit button was clicked by seeing which one of the submit buttons’ names is present in the parameters.

**Q13.** When you submit the form using “GET” instead of “POST,” the parameters are passed to the server in the first line of the header (i.e. GET /test\_form.html?email=get%40form.domain&password=useGET&secret\_info=secret\_value…) instead of the parameters being indicated in a string at the END of the header.

**Q14.** When I tried using POST, PATCH, and DELETE, they were converted into GET.

**Q15.** The response header for “GET /” and “GET /login” are the same, except that near the end of the header for “GET /login” there is a field called “Set-Cookie” with information to be stored in the cookie (i.e. logged\_in=true). This field is absent in “GET /”.

**Q16.** None of the information supposedly saved in the cookie has been saved and sent to the server, so the request to “GET /” after “GET /login” looks exactly the same as the “GET /” request that came before.

**Q17.** Servers don’t really have a good way of telling if cookies are “authentic” or not. What can be done is: limiting session time (expiration dates of cookies); limiting path/domain as much as possible; smuggling information in the cookie name; HTTPS authentication method; have the cookie store a key that points to further information stored in the server’s database, instead of having sensitive information stored in the cookie; and so forth.

Further reading:

<https://www.owasp.org/images/a/a0/OWASPLondon20171130_Cookie_Security_Myths_Misconceptions_David_Johansson.pdf>

<https://resources.infosecinstitute.com/risk-associated-cookies/>

<https://techblog.topdesk.com/security/cookie-security/>