MapleCTF: HTTP Guide

This is a quick guide to discuss the wonderful world of the HTTP protocol!

Author: Vie 🦢

The Basics

HTTP stands for “Hyper-Text Transfer Protocol” and is the basis for website communication all over the web. For those taking CPSC 317, the HTTP protocol resides in the application layer of the internet stack. An extension of the HTTP protocol is the HTTPS protocol, which is the same thing but the added S represents ‘SSL/TLS’ (Secure Sockets Layer/Transport Layer Security) - a cryptographic protocol that enrypts the website communications so no one else can snoop in.

HTTP is a client-server protocol: it models how communication is supposed to work between some client and some server. When we’re browsing on the internet, we are the client whenever we access a website, which are the servers.

The Methods

As a client, in order to talk to the websites we specify certain methods of HTTP that the server will respond to. There are a few in mind, but the most common ones to know about are:

GET

The bread and butter method of HTTP! A GET request occurs when you want to simply access a resource. And this could be anything: an image, a webpage, an embedded link, etc etc.

POST

POST requests will typically hold a request body, with some user-inputted data in it and request the server do something with that data. Usually, POST requests are the backstage actors for webpage forms, file uploads, and other things which involve data transmission from a client to a server.

PUT

PUT requests seem similar to POST requests: these types of requests will ask the server to change or update some state in it, while supplying some data with which to use for that state change. The main difference between a PUT and POST is that the client can specify the location of the data transmission with a PUT request.

HEAD

HEAD requests are similar to GET requests but they only recieve the representation state of the request, but not the data, which a GET would recieve. HEAD requests typically share only the response headers of server’s response.

OPTIONS

The OPTIONS request is a unique one that a client can use to ask the server what methods are available to use. OPTIONS requests are enabled on all servers by default, so you can freely send an OPTIONS request to a server and see what methods are available.

When we browse the web, we make these sorts of requests all the time but it is mostly invisible to us. Our browsers do much of the heavy lifting for us!

This is not an exhaustive list. Plenty of HTTP methods are used! However, these will likely be the ones you may encounter most often in your CTF journey.

Server Responses

When a client sends a request and the server recieves it, it will respond in a specific format:

HTTP code: 3-letter number which is used for letting the client know if their request was succesfully processed or if it failed and what happened. There are a lot of different HTTP codes, but the ones to know about are:

- 200: Success. The request had been processed.
- 4xx: A failure code. Any HTTP code starting with 4 signifies a failure in the request processing occurred due to a client error.
    - 404 - Not found. The requested resource wasn't found on the server.
    - 401 - Unauthorized. The client requested a resource they don't have the right credentials for.
    - 403 - Forbidden. The server recieved the request but has refused to process it. Some servers may tell you why, most don't. This seems similar to the above 401 error, but the main difference is that recieving a 401 allows you to try and authenticate in order to access the resource, a 403 is a hard bar you can't pass.
- 5xx: A failure code. Any HTTP code starting with 5 signifies a failure in the request processing occurred due to a server error. Usually, something in the server screwed up, and the requested resource could not be retrieved.

Response Headers: a bunch of headers which tell the client some relevant info for the resource that was requested; how long the resource is, what data-type it is, etc etc.

The Resource: If you used a GET/POST request you can recieve the requested resource in the body of the response. The resource can be a webpage, an image, a redirect, etc etc.

REST API

The REST API is a methodology applied to HTTP and communications between a client and server. REST gives a guide to standardize all communications between the World Wide Web, and us. You can see it as a list of constraints that developers use:

- The application is a client-server infrastructure that is uniform in design - similar resources come from one place, and requests for the same resource should look the same.
- The client and server are independent from one another - decoupled, if you will. The clinet shouldn’t know anythign about the server except for the URI where it can request things from. The only connection they have are through their communication on HTTP.
- Statelessness - REST APIs tell the infrastructure that the communications between client and server are stateless. Each request is completely whole and contains all the data it needs for processing. Servers do not have server-based sessions for data storage.
- Resources should have the ability to be cached where possible - either on the server or client. If a client requests the same resource again, there doesn’t need to be too much work done to retrieve that resource, it just needs to be loaded from a cache.
- The client-server architecture is layered, so the requests and responses between the two can go through many different intermediaries before it reaches its target. The layers that a response/request can go through should be invisible to the client - it should like like the client is directly accessing the server.
- Optionally, you can offer code-on-demand services in a RESTful API. Sometimes, a response may containe executable code, but this is not a requirement.

Having a rough idea of how a RESTful API works may be useful in the future, even outside of CTFs. Most of the internet actually conforms to the RESTful standard, and it’s been a widespread API for a few number of years.