A Quick Guide to Web Systems

Eike Ritter

February 19, 2020

This document gives a very brief introduction to the technologies you need to know for the web security part of the " Computer Security" module.

HTML & JavaScript

Basic webpages are usually written in HTML and JavaScript, I hope that you are familiar with these already, if not then please work through the online tutorials here:

`http://www.w3schools.com/html/default.asp` `http://www.w3schools.com/js/default.asp`

An important point to remember about JavaScript is that it is executed by the client's browser.

URL: Uniform Resource Locator

A basic URL takes the form: `Protocol://host/FilePath`, where `Protocol` tells you what protocols to use to access the resource, `host` is the name or IP address of the computer the resource is on, and `FilePath` is the path to the resource on the host. E.g. the URL `http://www.cs.bham.ac.uk/index.php` refers to the file `index.html` that is on the computer `www.cs.bham.ac.uk` (147.188.192.42), which should be requested using the HTTP protocol.

Additional data can be included in a URL as a query string:

```
Protocol://host/FilePath?field1=value1&field2=value2
```

HTTP: Hypertext Transfer Protocol

HTTP is the protocol for requesting webpages. The two most common HTTP commands are `GET` and `POST`. These commands can usually be used interchangeably, however it is good practice to use `GET` when requesting a webpage and `POST` when sending data to a server.

HTTP servers listen on port 80. Client should send ACSII text to the server, the first line of which should be either a `GET` or `POST` followed by the filePath of the URL, and the version number of HTTP used (currently 1.1). As well as the command, the client should also send a number of HTTP headers that provide additional information about the request. The only compulsory header is the `Host` header that specifies the name of the host the client is trying to connect to. E.g. the school's homepage could be requested using HTTP by opening a socket connection to port 80 on `www.cs.bham.ac.uk`/147.188.192.42, and sending the text:

```
GET /index.php HTTP/1.1
Host:www.cs.bham.ac.uk
```

Modern browsers will also include many additional header fields which give useful information to the server e.g.:

```
GET /index.php HTTP/1.1
Host: www.cs.bham.ac.uk
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:25.0) Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
```

For more information about HTTP header fields see `http://en.wikipedia.org/wiki/HTTP_header`.

The server replies with a HTTP response that starts with a status code, e.g. 200 for "OK" or 404 for "file not found"., then there are a number of response-headers which give information to the browser, after which comes the requested resource. E.g. the schools web server replies to the above request with:

```
HTTP/1.1 200 OK
Date: Tue, 17 Dec 2013 15:55:42 GMT
Server: Apache
X-Powered-By: PHP/5.3.3
Vary: Accept-Encoding
Content-Type: text/html; charset=UTF-8
Content-Length: 40408

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
...
```

For more on HTTP see `http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol`.

Proxying your web traffic

To observe and alter the HTTP going between your web browser and a server, you can use a web proxy such as the Burp Proxy: `http://portswigger.net/burp/proxy.html`. If you download and run Burp it will start a http proxy on port 8080 of your machine. To send traffic via Burp you need to configure your web browser to use this proxy. E.g. in Firefox, click on Preference > Advanced > Network > Settings... > Manual proxy configuration, then enter 127.0.0.1 8080 as the HTTP Proxy. To see incoming traffic in Burp go to Proxy > Options > Intercept Server Responses and click "Intercept responses based on the following rules:".

N.B. you will need to use Burp for the next exercise.

HTTP Forms

HTTP forms are an easy way for websites to pass user data to a server. E.g. the HTML form:

```
 <form action="resultPass.php" onsubmit="return validate_form(this);"
                                                        method="get">
   <p>Number 1: <input type="text" name="noOne" /></p>
   <p>Number 2: <input type="text" name="noTwo" /></p>
   <p>Username: <input type="text" name="user" /></p>
   <p>Password: <input type="password" name="pass" /></p>
   <p><input type="submit" /></p>
</form>
```

will display as a form with 4 fields and a submit button.

When the user enters data and clicks on submit a `GET` request is sent to the server with the data as the URL query string. E.g.:

```
GET /resultPass.php?noOne=4&noTwo=5&user=tpc&pass=Password1 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:25.0) Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://127.0.0.1/test.html
Cookie: nobody
Connection: keep-alive
```

The `onsubmit` attribute in the `form` element specifies some JavaScript that should be run went the user clicks submit. The form is only sent if this JavaScript function returns true. This can be used for input validation, for instance, to check that the text entered into the numbers fields were numbers.

If the `method` attribute is changed to POST then the data is sent in the body of the message. E.g.:

```
POST /resultPass.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:25.0) Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://127.0.0.1/test.html
Cookie: nobody
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 39

noOne=4&noTwo=5&user=tpc&pass=Password1
```

This means that the data is not displayed on the URL bar or in web logs (useful for passwords), and is also not automatically re-sent if the user clicks back (useful for actions that should not be accidentally repeated e.g. credit card transactions).

PHP

PHP is the most popular web language[1]. PHP can be added to a website by putting it between `<?php`, `?>` tags. The website file must end with `.php` for a server to execute the PHP code. E.g. hello world in PHP is:

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
    <?php echo "Hello World"; ?>
  </body>
</html>
```

[1] `http://w3techs.com/technologies/overview/programming_language/all`

A really useful feature of PHP is the ability to get the `$_GET` and `$_POST` parameters and compute with them. E.g. the following PHP code would add up and print the two numbers submitted by the form above.

```
<?php
  $noOne = $_GET["noOne"];
  $noTwo = $_GET["noTwo"];
  echo "Number 1 + Number 2 = ".(intval($noOne)+intval($noTwo));
?>
```

If you want to accept both posts and gets you can use `$_REQUEST`.

You can set cookies on the client's browser using `setcookie` and read back the value from a cookie in a clients using `$_COOKIE[$cookie_name]`. PHP can also automatically manage cookies for you use sessions (see e.g. `http://www.w3schools.com/php/php_sessions.asp`).

For more about PHP see the online tutorial at: `http://www.w3schools.com/php/default.asp`. The key difference between JavaScript and PHP is that JavaScript executes on the client, where as PHP executes on the server. Other common server side web languages include, ASP, JSP (Java) and CGI(Perl), which are similar to PHP but not as widely used.

SQL

Some typical SQL commands include:

```
SELECT * FROM users WHERE username='tpc';
```

which finds all database records in the table `users` where the username equals `tpc`.

```
UPDATE users SET uses=5 WHERE id=3
```

which changes the number of `uses` for the user with `id`to equal 3.

```
DROP TABLE users
```

which deletes the database table `users`.

Typically, a SQL server will run listening on a port, and other processes will connect to it and send it queries. Using PHP we can connect to the mySQL database using the following command:

```
$con=mysqli_connect(<host name>,<username>,<password>,<database name>);
```

We can execute a SQL query and get the results using the PHP code:

```
$result = mysqli_query($con,"SELECT * FROM users WHERE username='".$user."'");
$row = mysqli_fetch_array($result);
```

`row` then contains a array with the results of the query.

For these examples I'm using a mySQL database (`http://www.mysql.com`). For a PostgreSQL database (`http://www.postgresql.org`) just change the start of each command from `mysqli` to `pg`.