

Securing Web Applications

Secure Programming and Scripting

DARA O'SULLIVAN
20080494
Computer Forensics and Security

Table of Contents

- 1.a HTTP POST Request.*
- 1.b HTTP GET Request.*
- 1.c Cross Site Scripting Attack on POST Request.*
- 2.a Database Query.*
- 2.b SQL Injection.*
- 3. HTML Basic Protections.*
- 4. PHP Script Validation.*
- 5. Cookie Demonstration.*
- 6. Session Variable Demonstration.*

1.a. HTTP POST Request.

Php script which creates webpage below:

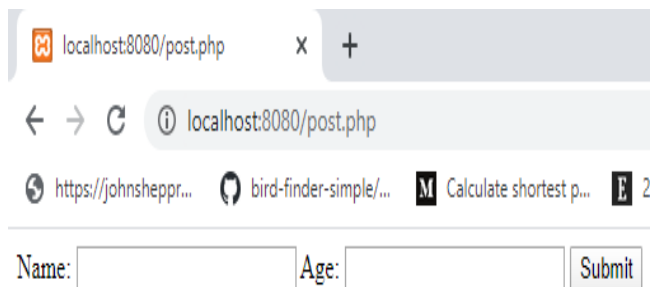
```
<?php
if( isset($_POST["name"]) || isset($_POST["age"]) ) {
    echo "Welcome ". $_POST['name']. "<br />";
    echo "You are ". $_POST['age']. " years of age.";

    exit();
}
?>
<html>
<body>
<form action = "<?php $_PHP_SELF ?>" method = "POST">

    Name: <input type = "text" name = "name"/>
    Age: <input type = "text" name = "age"/>
    <input type = "submit" />
</form>

</body>
</html>
```

Webpage:



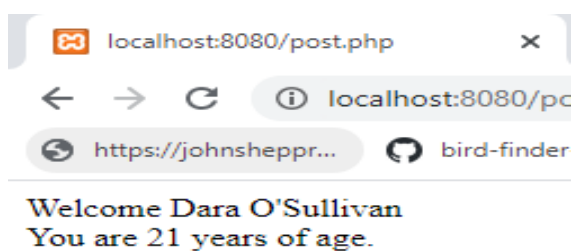
localhost:8080/post.php

localhost:8080/post.php

https://johnsheppr... bird-finder-simple/... Calculate shortest p... 2

Name: Age:

In the webpage we enter our details into the HTML script. This data is sent to the server with POST and stored in the request body of the HTTP request. This is then simply echoed out to display the data.



localhost:8080/post.php

localhost:8080/post.php

https://johnsheppr... bird-finder

Welcome Dara O'Sullivan
You are 21 years of age.

1.b. HTTP GET Request.

Php script:

```
<?php
if( $_GET["name"] || $_GET["age"] ) {
    echo "Welcome ". $_GET['name']. "<br />";
    echo "You are ". $_GET['age']. " years of age.";

    exit();
}
?>
<html>
<body>

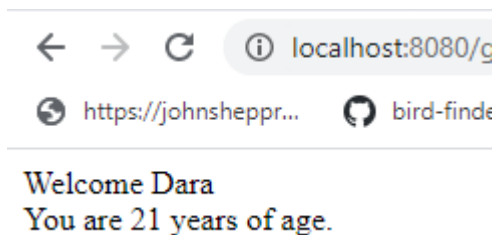
<form action = "<?php $_PHP_SELF ?>" method = "GET">
    Name: <input type = "text" name = "name" />
    Age: <input type = "text" name = "age" />
    <input type = "submit" />
</form>

</body>
</html>
```

Because the GET request holds the data in the URL I used the following to access the webpage.

<http://localhost:8080/get.php?name=Dara&age=21>

From this we get the same output as for the POST request above.



POST vs GET requests.

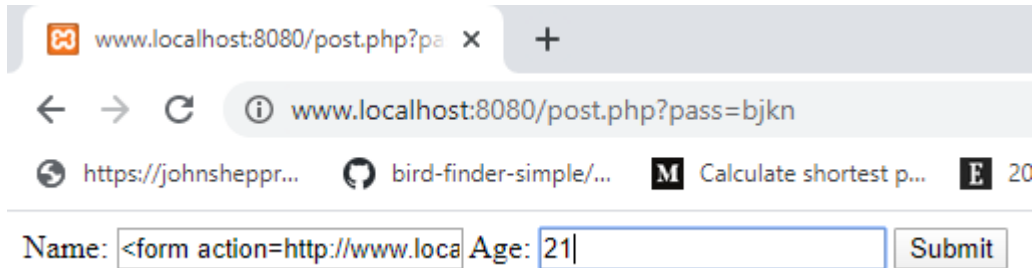
- GET requests can be cached whilst POST requests are never.
- Get requests remain in browser history. POST requests do not.
- GET requests can be bookmarked. POST requests cannot.
- POST requests can be used to modify data but GET only request data, never modify.
- POST requests have no restrictions on data length or data type, GET requests do (max length 2048 characters and only ASCII characters).
- GET is less secure compared to POST as data is sent in URL.

1.c. Cross Site Scripting Attack on POST Request.

A cross site scripting attack injects scripting code into webpages. As my php script is not secure we can do a cross site scripting attack on it. I will inject the following form which asks the user for a password which when submitted can be accessed.

```
<form action=http://www.localhost:8080/post.php>Enter  
your password  
<input type="password" name="pass_word">  
<input type="submit" value="Submit">  
</form>
```

Injecting form into the post request:



www.localhost:8080/post.php?pass=bjkn

https://johnsheppr... bird-finder-simple/... Calculate shortest p... 20

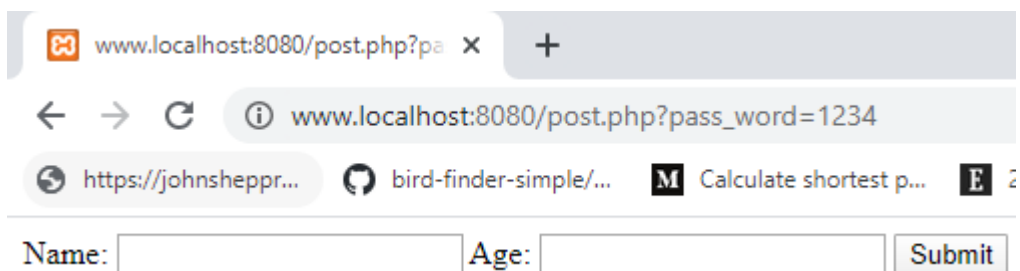
Name: <form action=http://www.localhost:8080/post.php?pass=bjkn Age: 21 Submit

Webpage now displays a password submission box.

Welcome
Enter your password Submit

You are 21 years of age.

I entered 1234 as my password and after submitting the original page reloads with the password in the URL.



www.localhost:8080/post.php?pass_word=1234

https://johnsheppr... bird-finder-simple/... Calculate shortest p... 20

Name: Age: Submit

2.a. Database Query.

I began by populating my database with the following:

```
INSERT INTO users VALUES ('user1', 'password1');
```

```
INSERT INTO users VALUES ('user2', 'password2');
```

```
INSERT INTO users VALUES ('user3', 'password3');
```

I used the following SQL query:

```
$query = "SELECT * FROM users WHERE Username = '$_POST['Username']' AND Password = '$_POST['Password']'";
```

If the username and password are a pair in the database, a table is displayed of the username and password. If they are not a match or do not exist, they are not shown in the table.

Enter Login Details

Username
Password
<input type="submit" value="Submit"/>

2.b. SQL Injection.

In my script I didn't use a Boolean statement on my query and therefore these injections don't work on my script but I will include them anyway.

A hacker could gain access to usernames and password by entering " or ""=" as the input. This would then make the sql statement look like the following:

```
SELECT * FROM Users WHERE Username = "" or ""="" AND Password = "" or ""=""
```

As OR ""="" if always true all the rows of the database could be returned.

Another possible attack would be to drop all entries in the table. This can be done with the following input:

```
user1; DROP TABLE users
```

The SQL statement would now look like the following:

```
SELECT * FROM Users WHERE Username = user1:DROP TABLE users
```

DROP TABLE will then delete all entries.

3. HTML Basic Protections.

HTML basic protections are of no real protection when it comes to halting an attack as there are ways in which these basic protections can be bypassed. These client-side validations are there for the convenience of users but do not protect or secure pages. Below I will use my php script but add a maxlength attribute set to 10 characters to see it working.

Name:	<input type="text" value="Dara O'Sul"/>	Age:	<input type="text" value="21"/>	<input type="submit" value="Submit"/>
-------	---	------	---------------------------------	---------------------------------------

So, it is working as it should. Preventing us from entering any more than 10 characters.

There are multiple ways an attacker can bypass these protections, below are a few.

- Submit an HTTP request directly to the server via the form's `action` URL

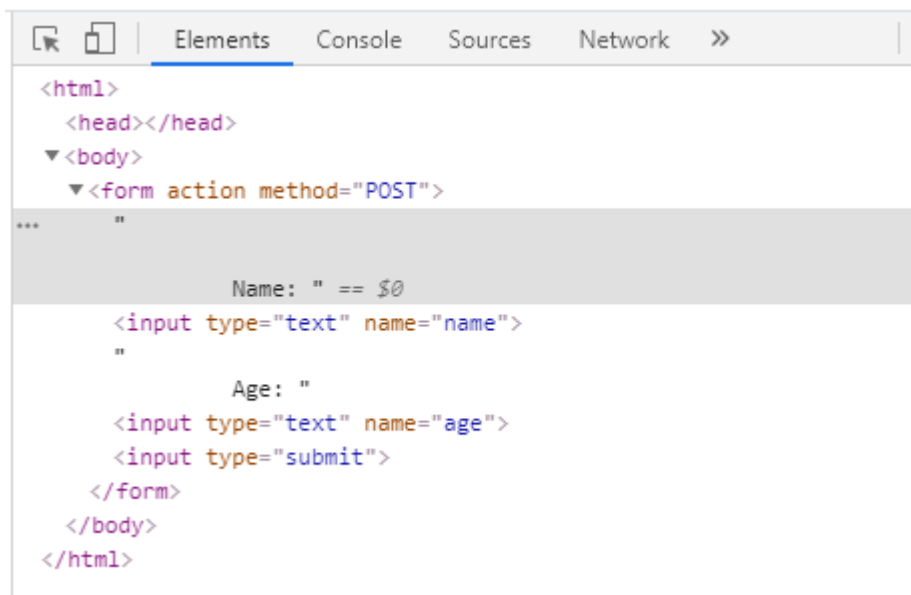
I will demonstrate this with my get request script which I used above. I will add an attack alert to the URL.

```
http://localhost:8080/get.php?name=Dara&age=21<script>alert('attacked')</script>
```

Using this URL we get the following pop up warning which tells us that a malicious attack could be made despite using the maxlength attribute.



- Because these are client-side protections they can be disabled. I will demonstrate this with max length. I will use my POST script from above to demonstrate. On the webpage if you right click on the form field and chose inspect the code is made visible to us. From here you can simply edit the code and remove the protection attributes.

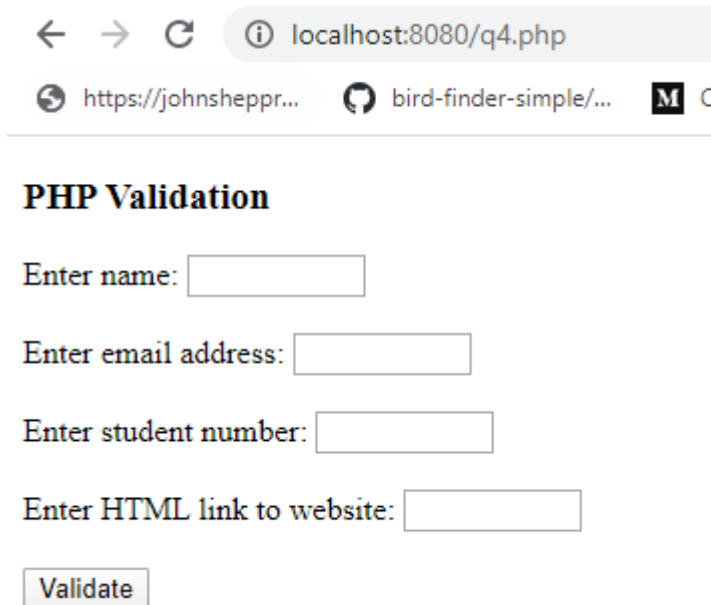


You should now be able to enter as many characters as you wish just as an attacker would be able to.

- Some web browsers don't honour maxlength which allows an easy window of opportunity for attackers to bypass this protection.

4. PHP Script Validation.

I used `empty()`, `preg_match()`, `filter_input()` and `htmlspecialchars()` as my validation functions. I created a form which obtained a name, email, student number and website.



PHP Validation

Enter name:

Enter email address:

Enter student number:

Enter HTML link to website:

I validated my name input with the `empty()` function. If nothing is inputted the name variable will be invalid.

```
if (empty($name))
```

To validate the email input, I used the `preg_match` regular expression function. This regular expression requires three groups. Group one is then followed by '@' symbol, then group two followed by a full stop, followed by group three.

Group one and group two accept a string of characters a-z (capital or lowercase) or a string of numbers or both. Group three accepts a minimum of 2 characters and a maximum of 5 characters, lowercase or capital a-z.

```
if(preg_match("/^([a-zA-Z0-9_-\.]+)@([a-zA-Z0-9_-\.]+\.[a-zA-Z]{2,5})$/", $email))
```

To validate student number input, I used `filter_input` as my validation function. I wanted to ensure the output was only numbers, to do this I used the `FILTER_SANITIZE_NUMBER_INT` option. This filter will ignore anything other than an int number and only return the integers.

```
if($student_no=filter_input(INPUT_POST, 'studentno', FILTER_SANITIZE_NUMBER_INT))
```

To validate the website input I used `htmlspecialchars()`. This converts all the applicable characters to HTML entities.

```
echo htmlspecialchars($website)." is your website as an entity!<br>";
```


I will demonstrate this now first using valid inputs and secondly using invalid ones. Using valid input which adheres to the functions used, the script simply echoed that they are valid.

```
Dara O'Sullivan is valid
daraosullivan7@gmail.com is valid
20080494 is valid
google.com is your website as an entity!
```

Next, I will use invalid inputs.

```
Variable name is invalid.
daraosullivan7@gmail.c is invalid!
20080494 is valid
google.com is your website as an entity!
```

I left the name variable empty therefore it was invalid. The email entered only has one letter in the third entry which is too little for the requirement therefore invalid. I entered the student number as W20080494 but the filter changed this to just integers as we would hope.

5. Cookie Demonstration.

A cookie is a small text file used to identify a user. The server stores the cookie on the user's computer. Every time a user requests the webpage the cookies are sent too. With PHP, we can create and retrieve cookie values.

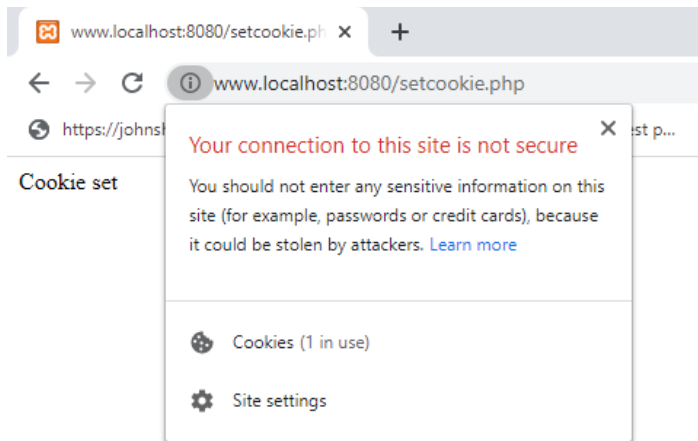
The `setcookie()` function allows us to set cookies using the following syntax.

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

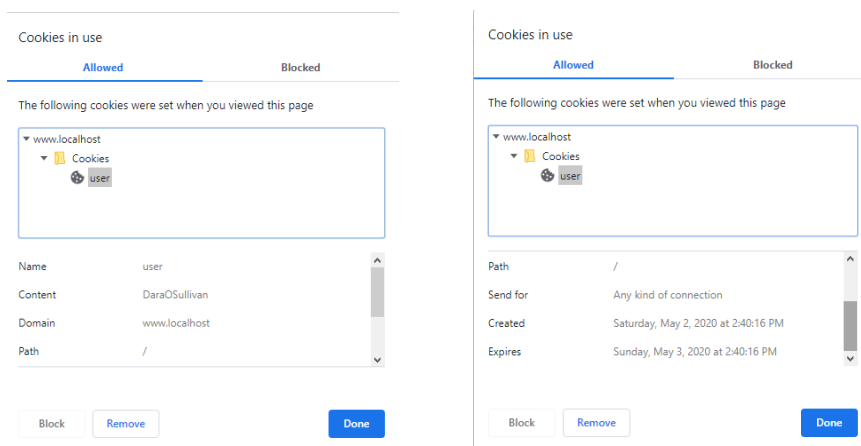
Below is a script to set cookie named "user" which expires after 24 hours and is assigned the value "DaraOSullivan":

```
<?php
$cookie_name = "user";
$cookie_value = "DaraOSullivan";
setcookie($cookie_name, $cookie_value, time() + (3600 * 24));
echo "Cookie set"
?>
```

Now the cookie should exist on the browser and we can check that it does by viewing the site information of the webpage. You can see in the picture below the cookie was set.



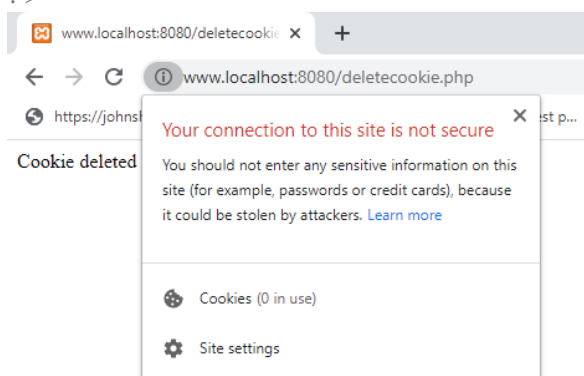
Next, we can view the cookie information by clicking into it.



You can delete a cookie from the user interface above or it can also be done by re-setting the cookie expiration date to a negative time therefore making the cookie expired.

```
<?php
$cookie_name = "user";
$cookie_value = "DaraOSullivan";
setcookie($cookie_name, $cookie_value, time() - 3600));
echo "Cookie set"

?>
```

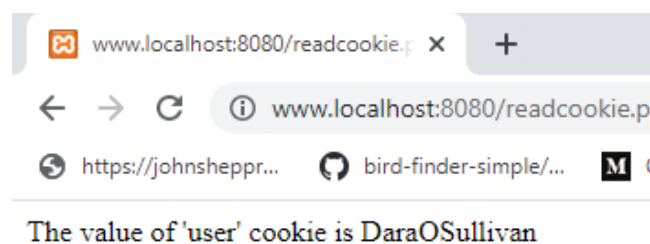


To read data from a cookie the `isset()` function is used which tells us if a variable exists and uses the following syntax.

```
isset($_COOKIE['nameOfCookie']);
```

We can also check the cookie info using a php script.

```
<?php
if (isset($_COOKIE['user'])) {
    $usercookie = $_COOKIE['user'];
    echo "The value of 'user' cookie is ".$usercookie;
} else {
    echo "No cookie";
}
?>
```



Security Options.

- Setting the path to a specific directory stops the cookie from being available within the entire domain and only available within the directory specified and its sub-directories.
- Leaving the **Domain** empty prevents subdomains from using the cookie.
- Using the **Secure** flag by also setting it to true ensures that the cookie will only be set if a secure connection exists.
- Using the **HttpOnly** flag by setting it to true will make the cookie accessible only through the HTTP protocol. This means it can't be accessed by scripting languages, e.g. JavaScript.

After these changes our php script to set a cookie would look like this:

```
<?php
$cookie_name = "user";
$cookie_value = "DaraOSullivan";
setcookie($cookie_name, $cookie_value, time() + (3600 * 24), '/example', null, true, true);
echo "Cookie set"
?>
```

Other than these method parameters it is also a good idea to never store sensitive data in cookies unless its completely necessary. You should also use session cookies whenever possible and when not possible ensure you use strict expiry dates.

6. Session Variables Demonstration.

The script below creates a session.

```
<?php
session_start();
?>
```

We can add to this by creating session variables on the server side. In the script below we have created a session variable which will hold the session count. The script reads the session variable and echoes it to the user.

```
<?php
session_start();
if(isset($_SESSION['counter']))
    $_SESSION['counter'] += 1;
else
    $_SESSION['counter'] = 1;
echo "Counter = ". $_SESSION['counter'];
?>
```

To destroy a session, the function `session_destroy` is used.

```
<?php
session_destroy();
?>
```