Lab 6: Cross-site Scripting

QUICK REVIEW

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.
Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.
An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

In this lab, we will have some general practices to see how Cross-Site Scripting attack is performed.

Important Notice:

Please carefully read the disclaimer declaration on the course webpage, before you start the lab practice, and make sure you fully understand all statements. The disclaimer is available on https://hogeschool.github.io/INFANL01-9.

LAB PRACTICES

Run the Kali Linux on your virtual machine, and login to DVWA. No need to mention that you need to start the server and database on your machine, using commands like:

```
service apache2 start
service mysql start
```

6.1. Low Security

In the first part, we try some simple attacks, by setting DVWA on Low security level. Go to DVWA Security page by clicking on the DVWA Security button. Set it on "Low", and press Submit button.

6.1.1. HTML Injection

Click on XSS (Reflected) to open the page "Vulnerability: Reflected Cross Site Scripting (XSS)".
Start with the basic HTML injection. Enter the following script in the input field "What's your name?", and press Submit button.

```
<h1>John</h1>
```

Try these:

```
<font color="green">John</font>
```

```
<a href="https://www.google.com" target="_blank">John!</a> click on your name
```

What are the results? What is the problem? Discuss it in your group.

6.1.2. Script Injection

In this practice, we would like to inject some higher-level scripts. Try the following script as your name:

```
<script> alert("Hello! It is a Script Injection test!"); </script>
```

Try it with whatever message you like.

Now, let's try another script to check the cookies. It is a good practice to manually add a cookie, before you check it with your script. Hence, add a cookie with name of "My name" and add your name as the "Value". In Firefox, you can do it using `Menu->Web Developer->Storage Inspector->cookies->+`.
Now, close the developer tools, and try the following code:

```
<script> alert(document.cookie); </script>
```

What are the cookies? Can you see the cookie you have manually added?
Discuss in your group about the vulnerability of code, in practices 6.1.1 and 6.1.2.

6.1.3. The Vulnerability

The code for Reflected Cross Site Scripting (XSS), with Security Level set on "Low" is given below:

```
<?php
header ("X-XSS-Protection: 0");
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
  // Feedback for end user
  $html .= '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
?>
```

Find the vulnerability in the code?
Figure out how this code can be modified to prevent these two attacks?
Discuss in your group.

6.2. Medium Security

Now, set the security level om "Medium".

6.2.1. Script Injection

Try the practice in 6.1.2, with this security level:

```
<script> alert("Hello! It is a Script Injection test!"); </script>
```

What is the result?
Try the second one:

```
<script> alert(document.cookie); </script>
```

What is your observation?
Discuss in your group why this time the attack it is not working.

6.2.2. The Modified Code

The modified code for Reflected Cross Site Scripting (XSS), with Security Level set on “Medium” is given below:

```
<?php
header ("X-XSS-Protection: 0");
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
  // Get input
  $name = str_replace( '<script>', '', $_GET[ 'name' ] );
  // Feedback for end user
  $html .= "<pre>Hello ${name}</pre>";
}
?>
```

What is changed in this code to prevent the script injection attack?
☐ Is there still a chance of injecting the same script? How?
Modify the code the prevent it?

6.2.3. HTML Injection

Try HTML Injection given in practice 6.1.1.
Is it still working? Discuss in your group why this attack is not prevented yet in the Medium level?
How you can modify the code in 6.2.2 to prevent HTML injection attack?

6.3. High Security

Set the security level on “High”.

6.3.1. Injection

Try the previous practices with this level, and discus your observations.

6.3.2. Code Analysis

Find the code in DVWA, and analyze the code.

6.4. Impossible Level

Set the security level on “High”, and test all previous practices on this level. Discus your observation.