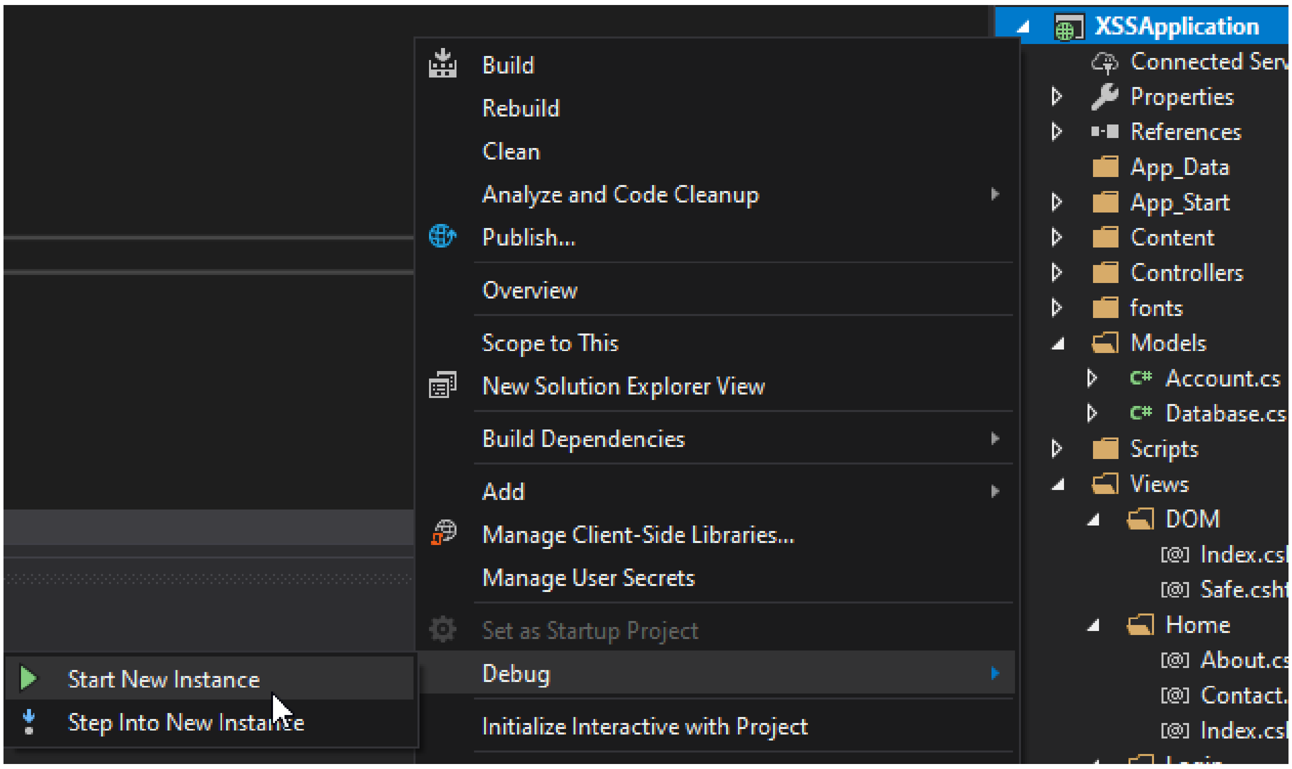
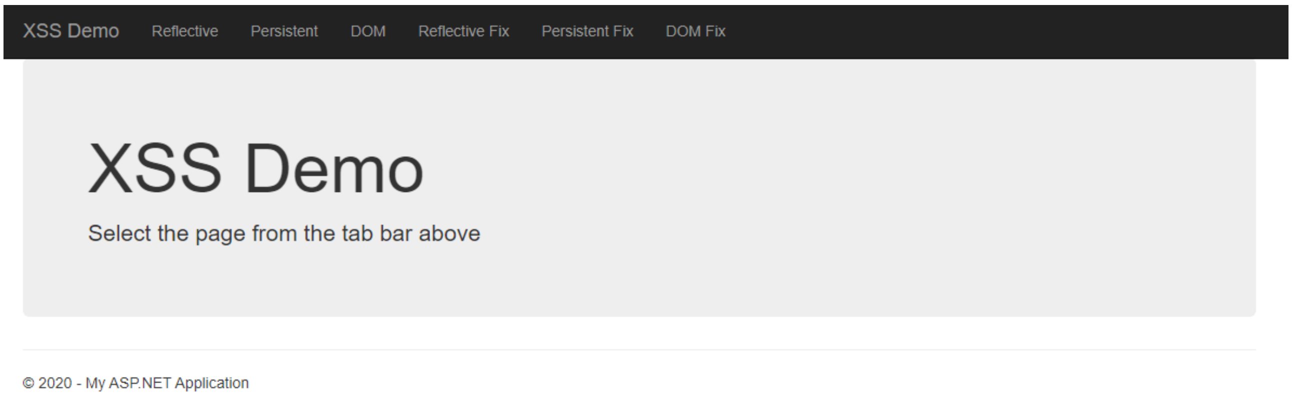
Kyle Kriskovich, 18831117

Cross-site Scripting

To launch web site select the folder XSSApplication and launch it in visual studio, right click on project XSSApplication and select Debug and start a new Instance.



From here you should be on the Home page. Here you can select from the six pages.



XSS Demo

Select the page from the tab bar above

© 2020 - My ASP.NET Application

1) Reflective

Summary of Page:

The page is a simple login page for an account. When you login in the page displays your first name, last name and card details.

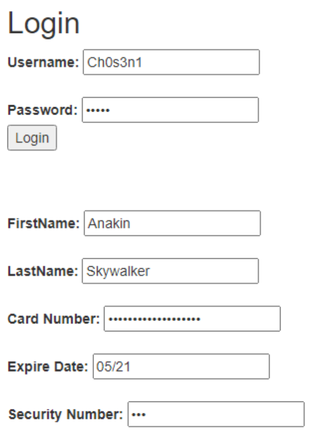

Login
Username: Ch0s3n1
Password: •••••
Login
FirstName: Anakin
LastName: Skywalker
Card Number: •••••••••••••••••••
Expire Date: 05/21
Security Number: •••

Figure 1: Correct Login with the Username: Ch0s3n1 and Password: Padme

The Insecurity:

The cross site scripting insecurity comes when you fail to login in, as the website displays the text "Sorry you entered the wrong username or password. The Username entered was: {input}". The insecurity occurs when the web site reflects the users input back to the user.

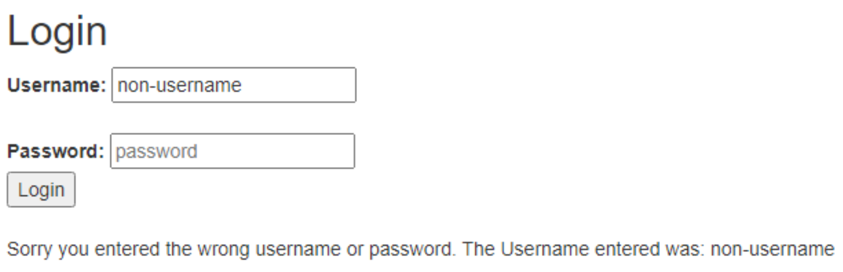

Login
Username: non-username
Password: password
Login
Sorry you entered the wrong username or password. The Username entered was: non-username

Figure 2: Website displaying the failed login text

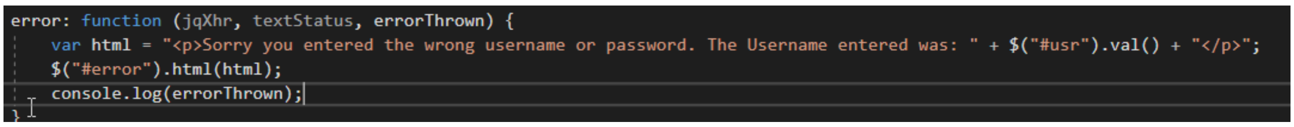

```
error: function (jqXhr, textStatus, errorThrown) {
    var html = "<p>Sorry you entered the wrong username or password. The Username entered was: " + $("#usr").val() + "</p>";
    $("#error").html(html);
    console.log(errorThrown);
}
```

Figure 3: The Insecurity in the code.

Recreate an attack:

Due to the fact the site reflects the users input back to the user on an unsuccessful login we can set the username to the following “Username = <script>alert("Reflective Insecurity")</script> ” and trying to login causing the website to output our username and successfully running our own input <script> on the website.

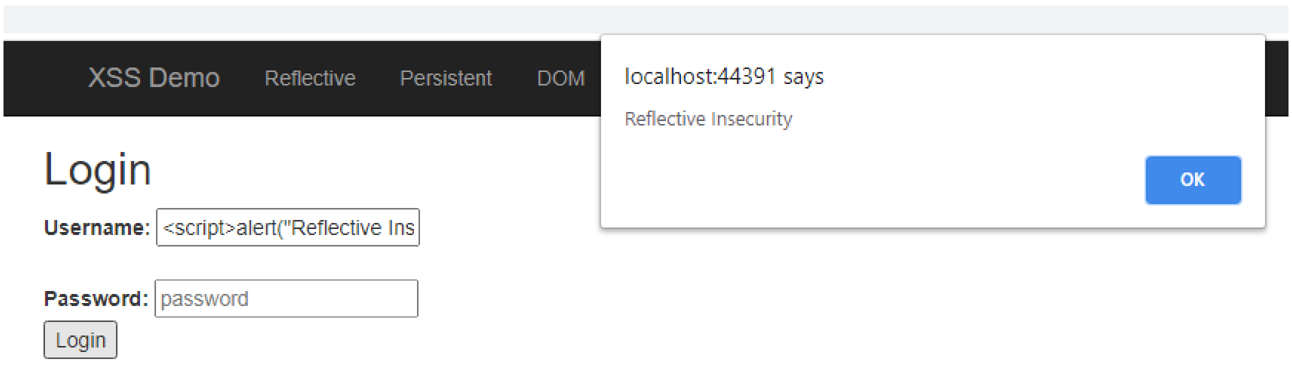
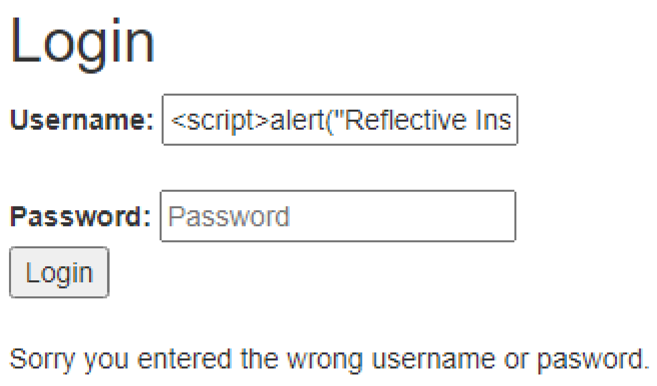


Figure 4: The input script causing an alert to occur on the page.

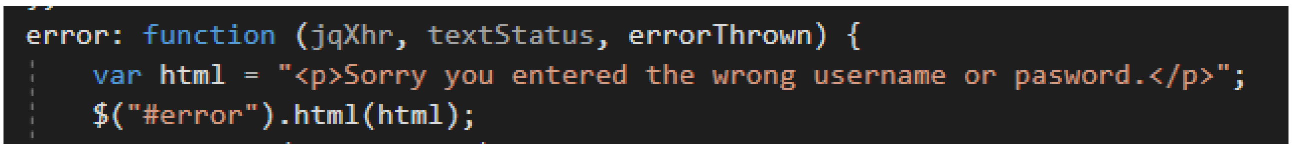
Solving the Insecurity:

The easiest fix for this security is to not reflect the users input in any way back to the user so they can’t use it to cause damage. This can be seen in the Reflective Fix page.



Sorry you entered the wrong username or pasword.

Figure 5: The webpage outputting the new error message without displaying the user input



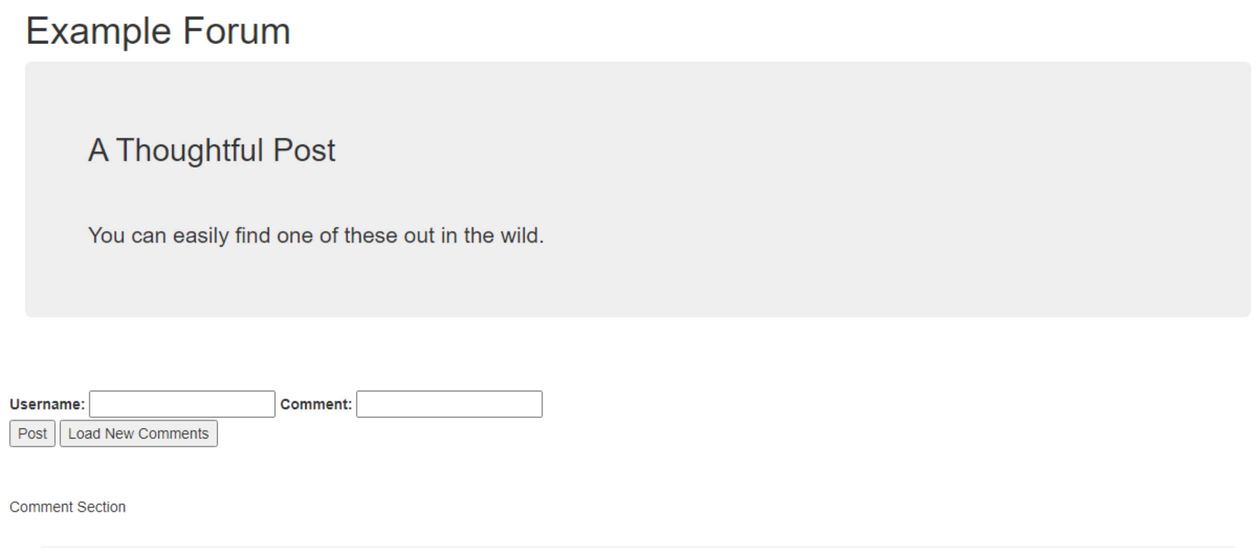
```
error: function (jqXhr, textStatus, errorThrown) {
    var html = "<p>Sorry you entered the wrong username or pasword.</p>";
    $("#error").html(html);
```

Figure 6: New error message in the code without displaying user input

2) Persistent

Summary of Page:

The persistent page is a simple forum with a post and a comments section. Allowing users to submit comments in the comment section so other users can later see them. Comments on this site are stored on a static database object located in the models folder.



Example Forum

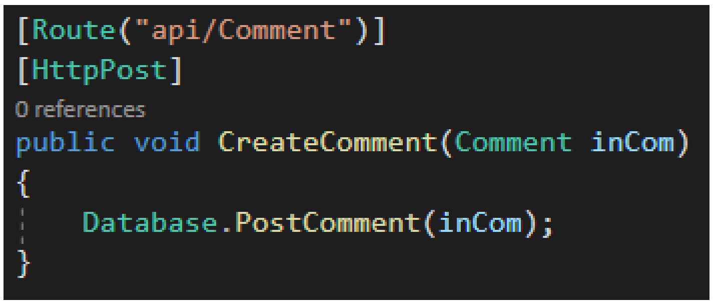
A Thoughtful Post

You can easily find one of these out in the wild.

Figure 7: Basic forum with the ability to post comments

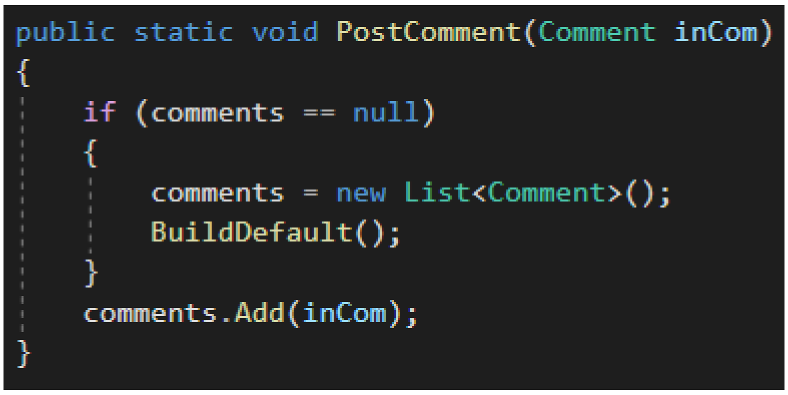
The Insecurity:

The insecurity of this page occurs with the posting of comments as the database will safe whatever was input in the comment section without making sure it is not malicious. Allowing the user to exploit this by inputting a malicious comment that can be seen by whoever is viewing this page.



```
[Route("api/Comment")]
[HttpPost]
0 references
public void CreateComment(Comment inCom)
{
    Database.PostComment(inCom);
}
```

Figure 8: Api sending the comment directly to the database without checking for malicious intent.

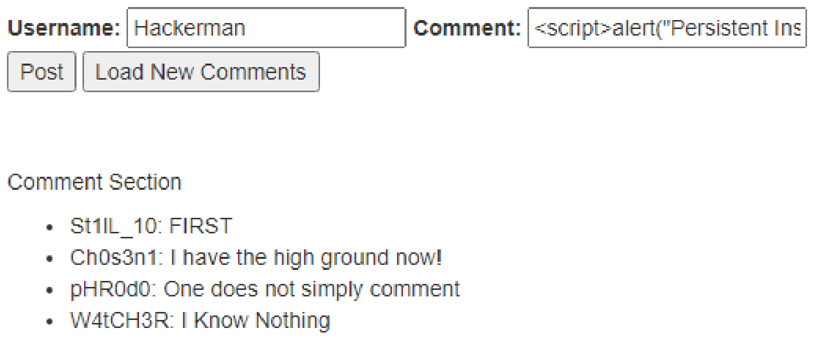


```
public static void PostComment(Comment inCom)
{
    if (comments == null)
    {
        comments = new List<Comment>();
        BuildDefault();
    }
    comments.Add(inCom);
}
```

Figure 9: Database code adding the incoming comment.

Recreate an attack:

To cause an attack on this website open two tabs of the same page. In one set the comment to the string “<script>alert("Persistent Insecurity")</script>” and then click the Post button.


Username: Hackerman Comment: <script>alert("Persistent Ins

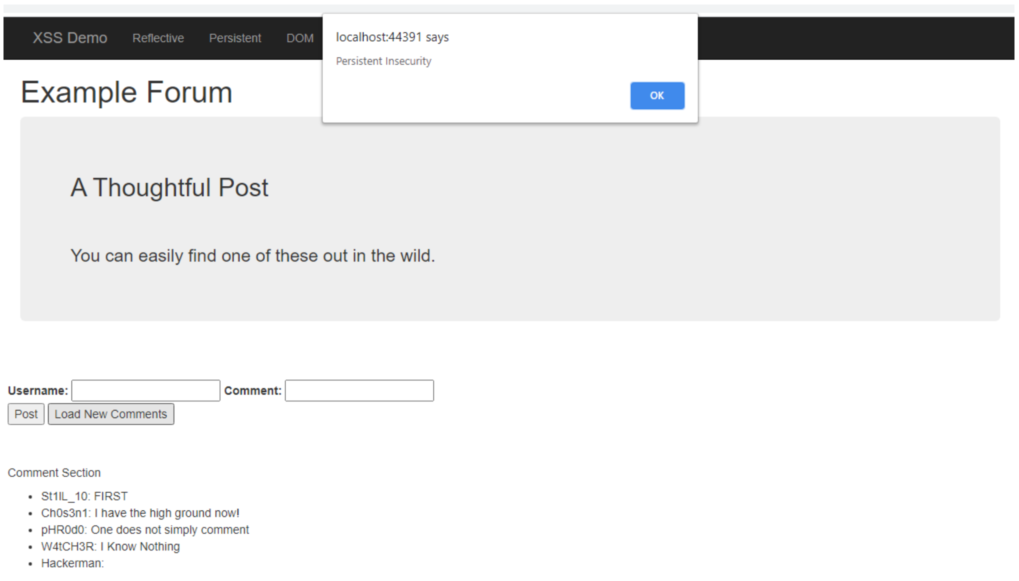
Post | Load New Comments

Comment Section

- St1IL_10: FIRST
- Ch0s3n1: I have the high ground now!
- pHR0d0: One does not simply comment
- W4tCH3R: I Know Nothing

Figure 10: Comment containing malicious script

Now go to the second tab of the website and select the load comments button and the script input from the first tab will be loaded and run on the page in the second tab.



localhost:44391 says

Persistent Insecurity

Example Forum

A Thoughtful Post

You can easily find one of these out in the wild.

Username: Comment:

Post | Load New Comments

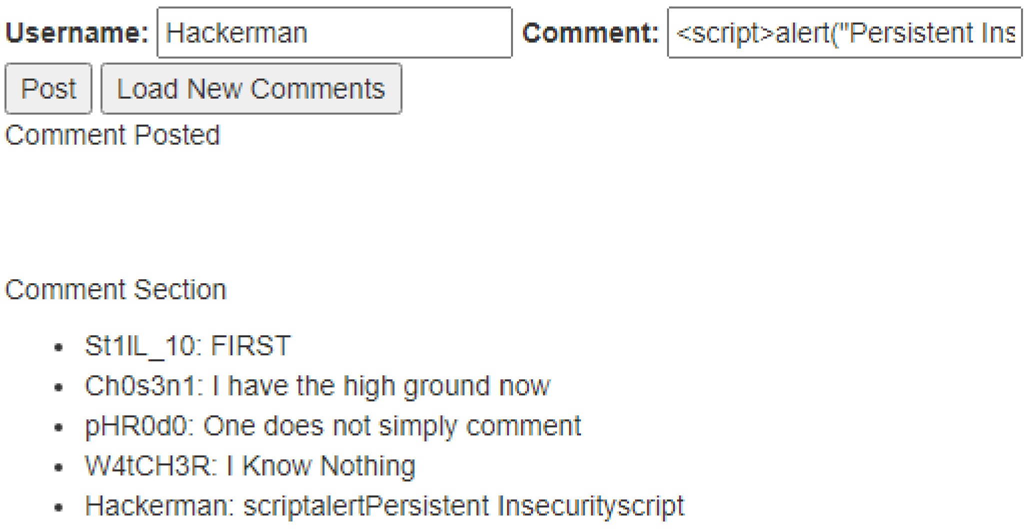
Comment Section

- St1IL_10: FIRST
- Ch0s3n1: I have the high ground now!
- pHR0d0: One does not simply comment
- W4tCH3R: I Know Nothing
- Hackerman:

Figure 11: Script that was saved to the database is loaded and run on every page that loads comments

Solving the Insecurity:

The insecurity occurs when the database just saves whatever string has been sent by any user. The fix that was implemented in the Persistent fix now checks that anything inputted into the database is cleaned of any special character and only contains letters, numbers and select special characters like space or full stop.



Username: Hackerman **Comment:** <script>alert("Persistent Ins

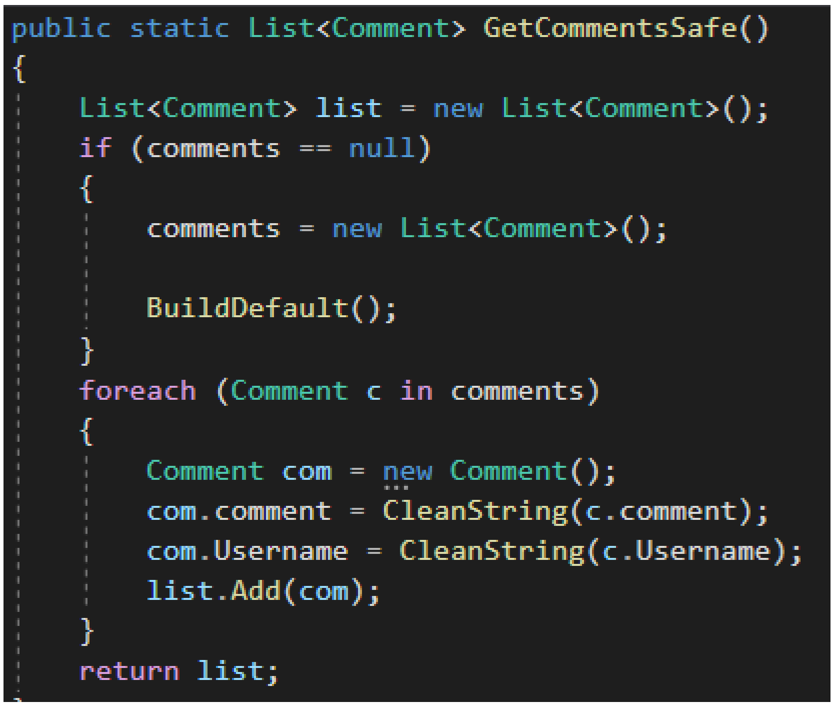
Post | Load New Comments

Comment Posted

Comment Section

- St1IL_10: FIRST
- Ch0s3n1: I have the high ground now
- pHR0d0: One does not simply comment
- W4tCH3R: I Know Nothing
- Hackerman: scriptalertPersistent Insecurityscript

Figure 12: Website removes all special characters from the comment.



```
public static List<Comment> GetCommentsSafe()
{
    List<Comment> list = new List<Comment>();
    if (comments == null)
    {
        comments = new List<Comment>();

        BuildDefault();
    }
    foreach (Comment c in comments)
    {
        Comment com = new Comment();
        com.comment = CleanString(c.comment);
        com.Username = CleanString(c.Username);
        list.Add(com);
    }
    return list;
```

Figure 13: Code of the get comments that returns a list of comments that has been cleaned of any special characters

3) DOM

Summary of Page:

The page is a simple search page that would theoretically search a website for an input string.

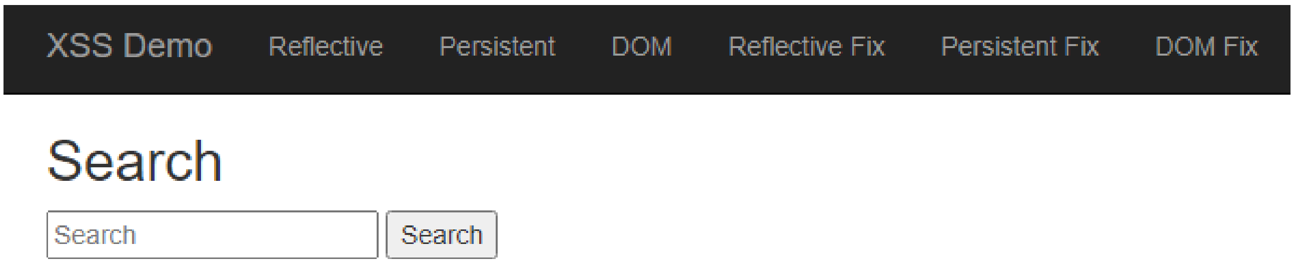


Figure 14: Search box for a search of the website

The Insecurity:

The insecurity occurs when the search happens as it submits the page and goes to url: “https://localhost:44391/DOM?search=searchterm” and displays the document.baseuri which means potentially

Recreate an attack:

To recreate the attack search for the term “<script>alert("Dom Insecurity")</script>”
url: “https://localhost:44391/DOM?search=<script>alert("Dom Insecurity")</script>”

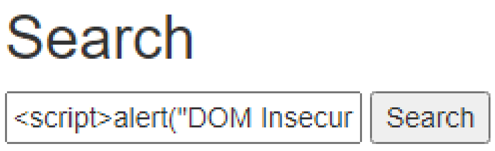
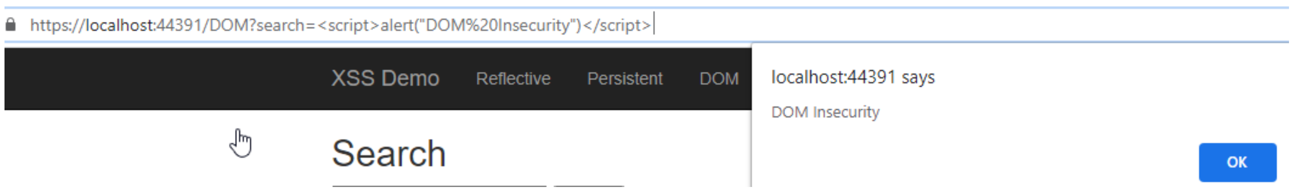


Figure 15: Searching of a malicious script


https://localhost:44391/DOM?search=<script>alert("DOM%20Insecurity")</script>

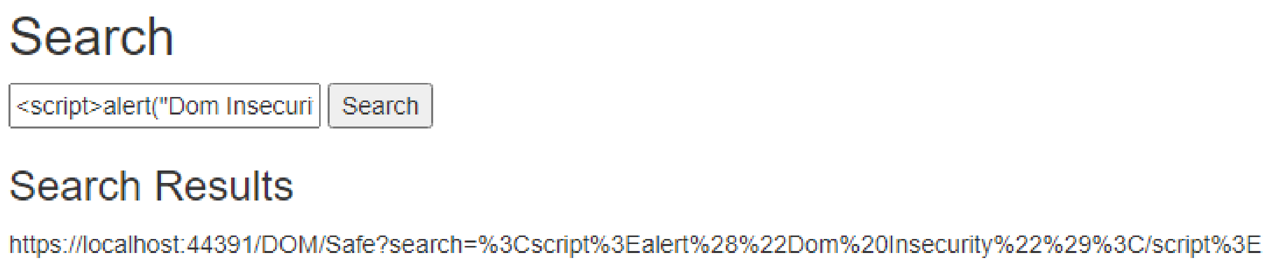
localhost:44391 says

DOM Insecurity

Figure 16: Result of the attack

Solving the Insecurity:

To solve the insecurity made sure the search results would escape before writing the document.baseuri which sanitises the input from potentially malicious strings.


Search

<script>alert("Dom Insecuri

Search

Search Results

https://localhost:44391/DOM/Safe?search=%3Cscript%3Ealert%28%22Dom%20Insecurity%22%29%3C/script%3E

Figure 17: Search results sanitise the input