Week 3 Homework 2

Matt Rieser

SDEV 325 – Section 6381

September 19, 2018

Professor Zachary Fair

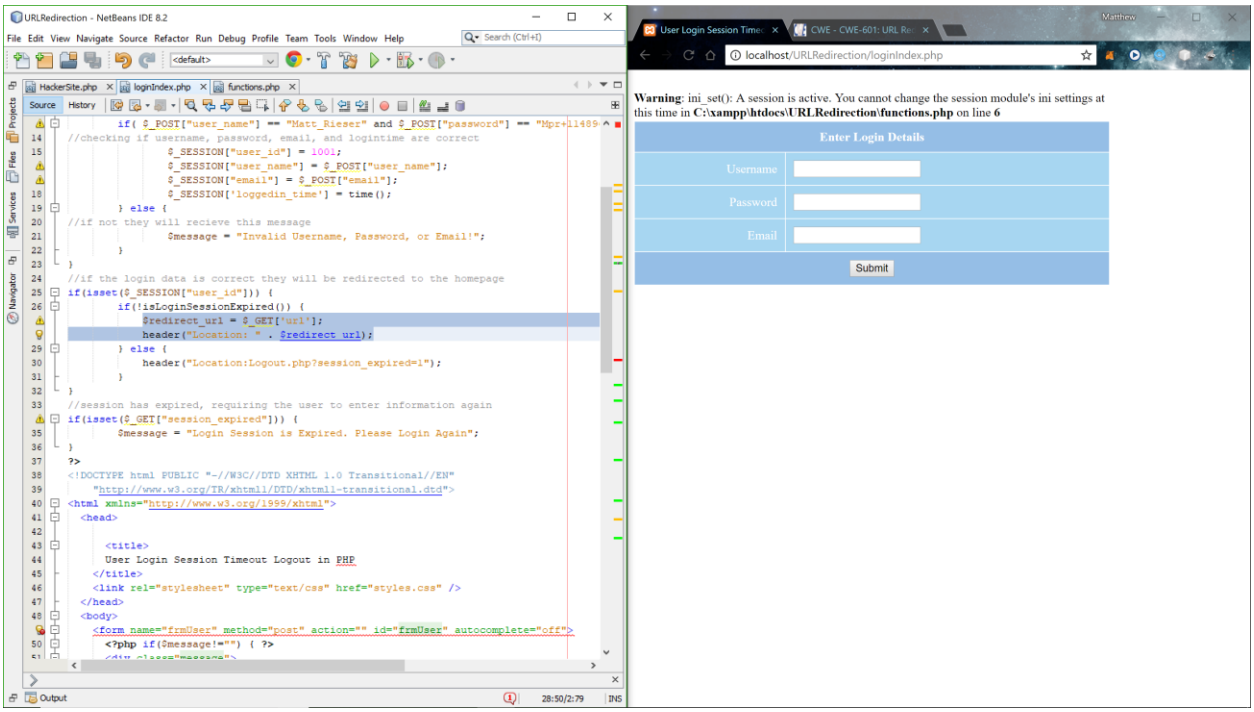
URL Redirection

Desciption - A web application accepts a user-controlled input that specifies a link to an external site, and uses that link in a Redirect. This simplifies phishing attacks.

Extended Desciption - An http parameter may contain a URL value and could cause the web application to redirect the request to the specified URL. By modifying the URL value to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts have a more trustworthy appearance.

Resource – https://cwe.mitre.org/data/definitions/601.html

Above – Highlighted is the security flaw

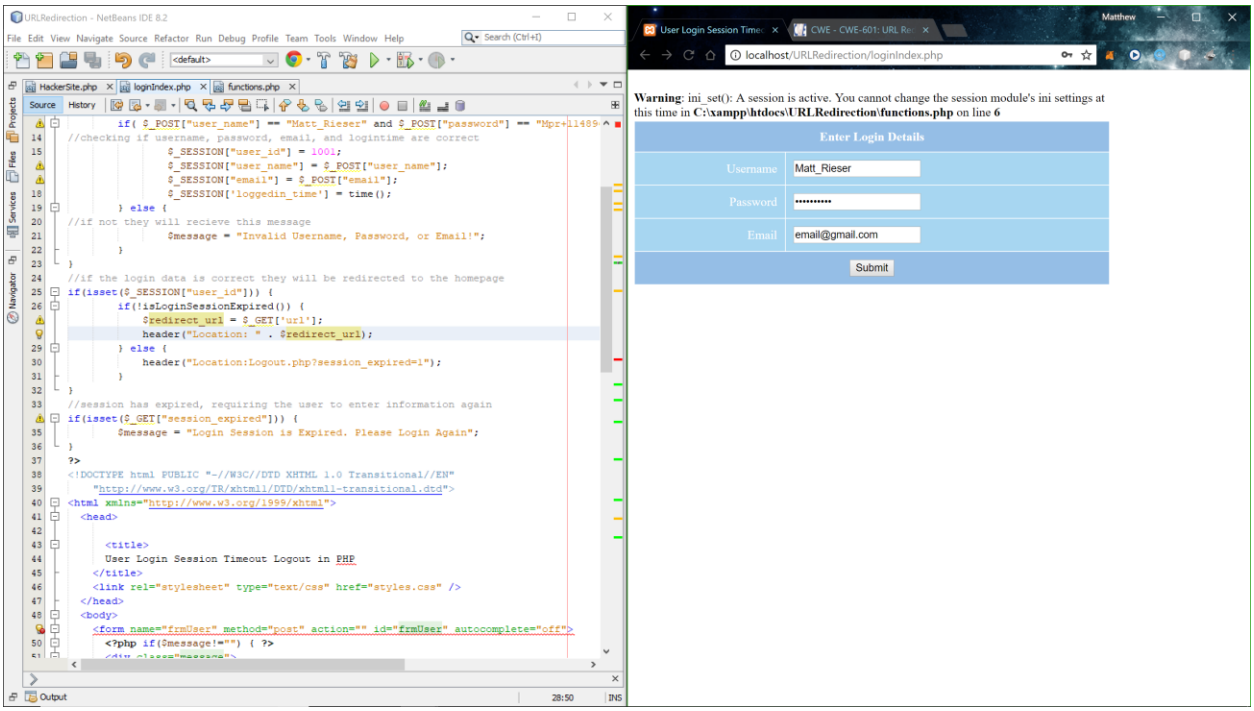


Code :

$redirect_url = $_GET['url'];

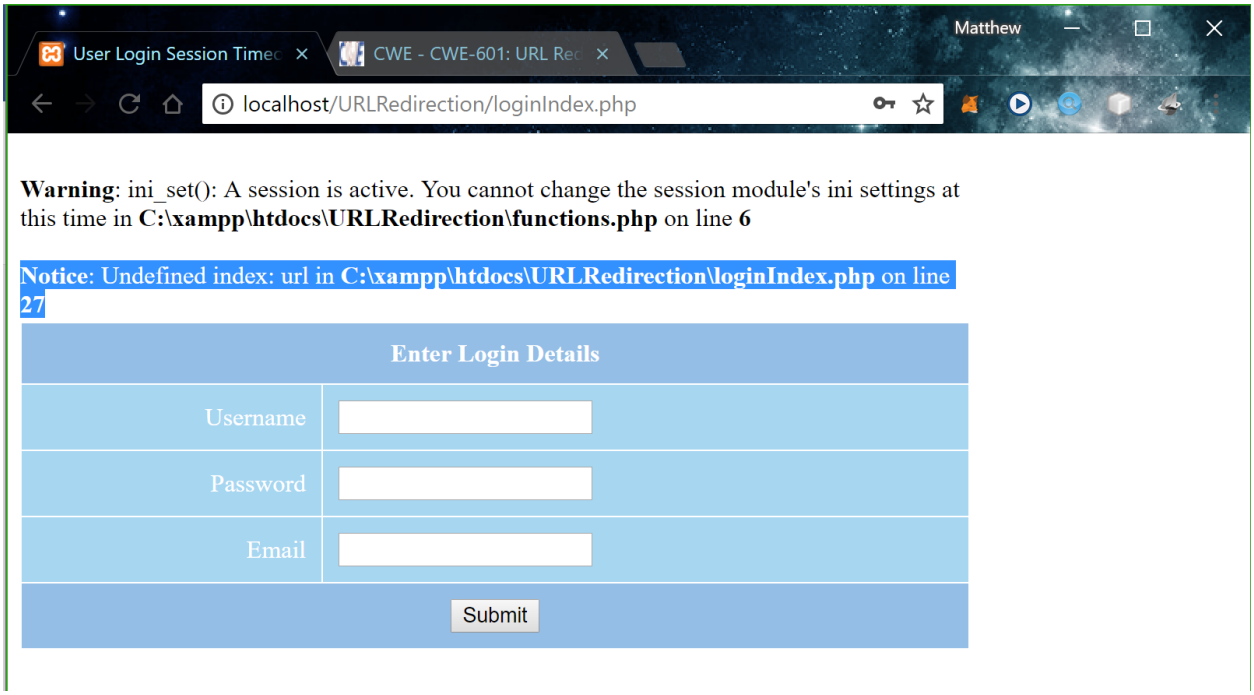
header("Location: " . $redirect_url);

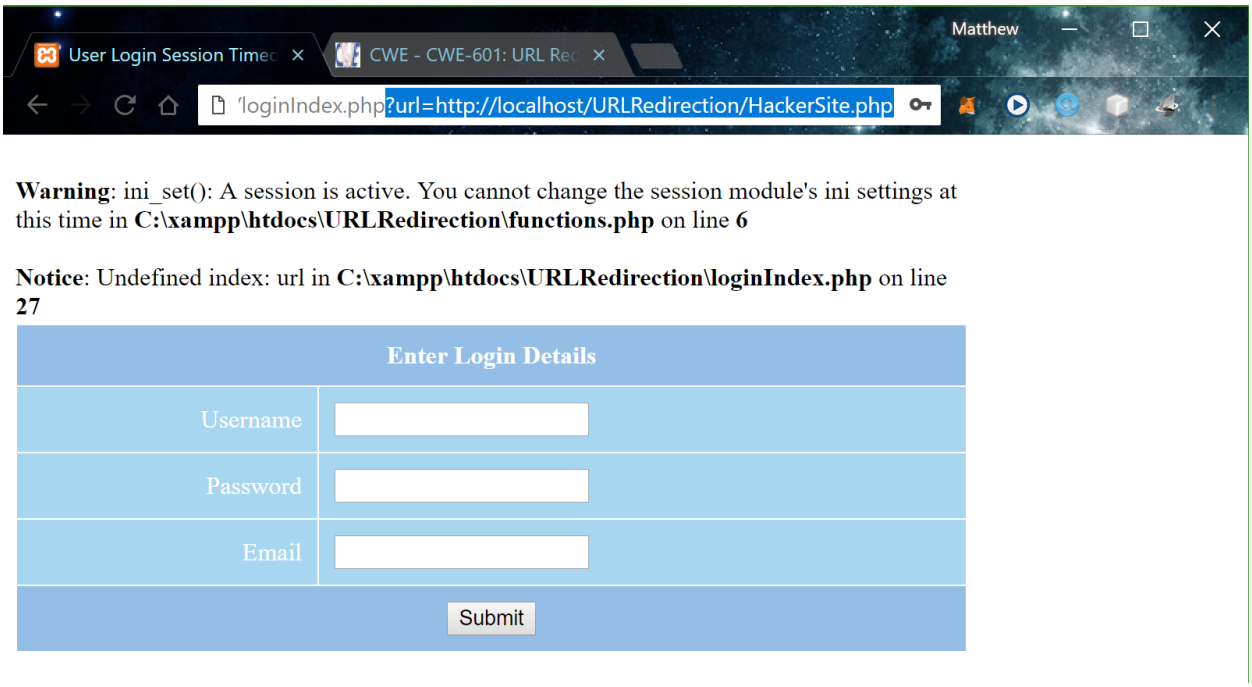
This code grabs the url from the get request. There is no specificed location to redirect the user to after they login which is where the security flaw is.



Above - Login with Username – Matt_Rieser, Password- Mpr+114896, and email- email@gmail.com

Below - Once logged if using an xampp server you will see the notice: Undefined index: url in C: (wherever your file is)



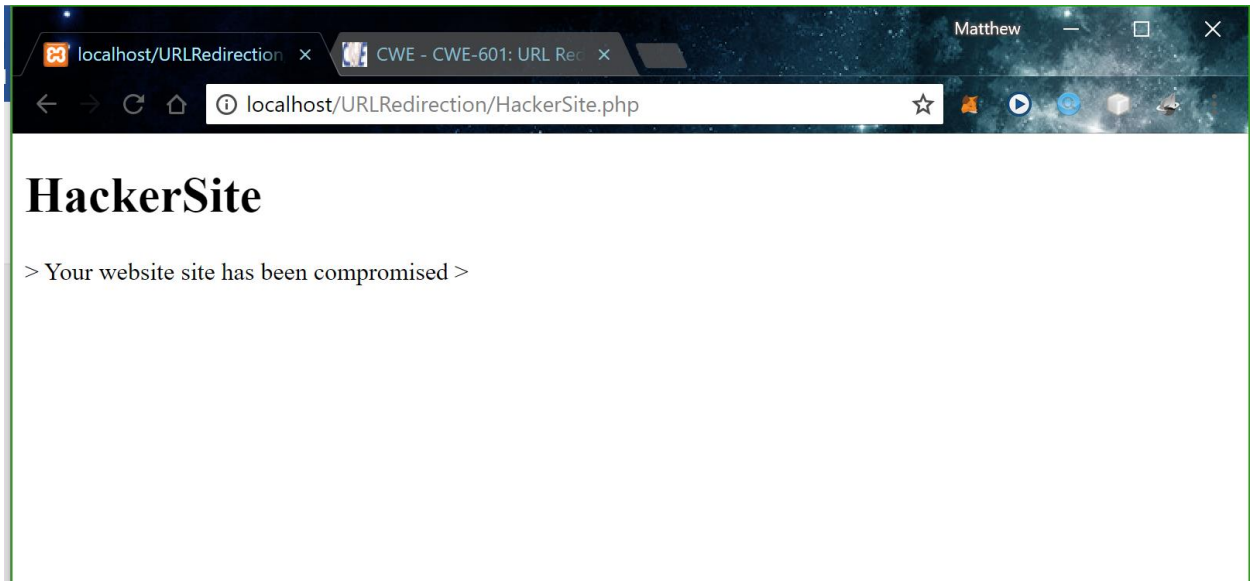


Above – Now to exploit, in the url type the following after the normal url

Type:

?url=http://localhost/URLRedirection/HackerSite.php

Then chick enter

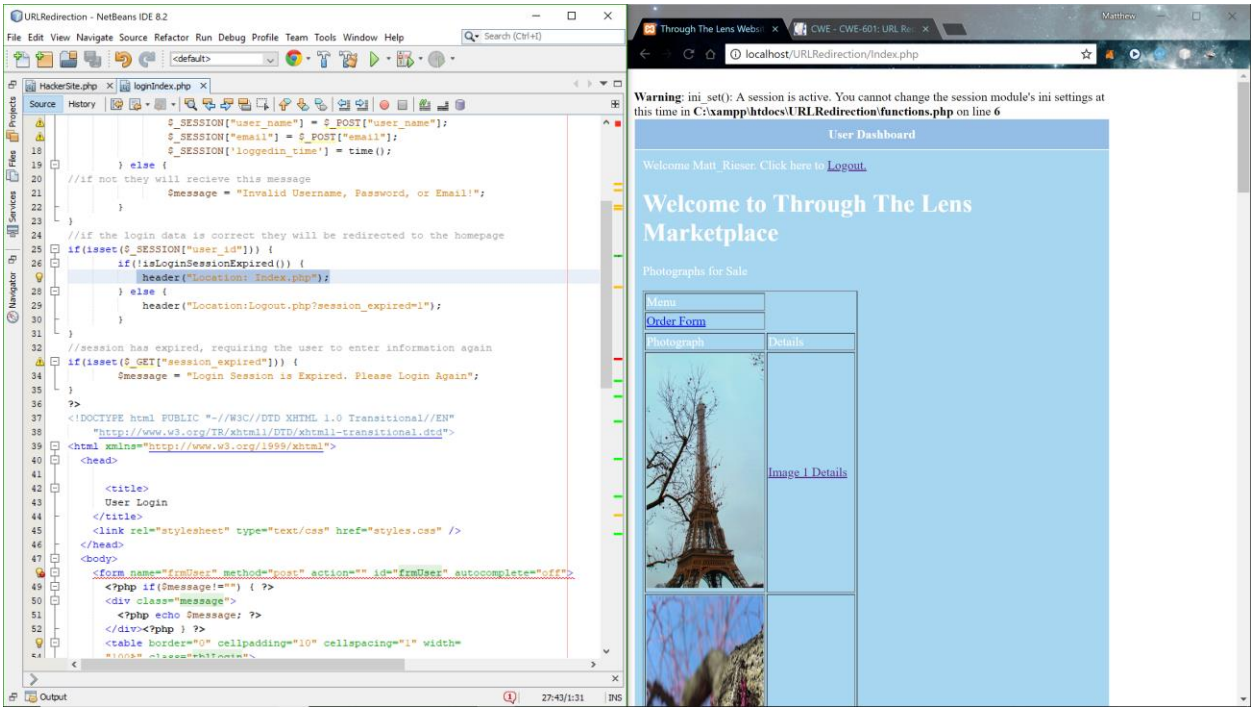


Above - The logged in user's session is now redirected to the HackerSite.php, where an attacker now has the ability to steal user creditentals through phishing.

Fix

Specify a header location

I replaced



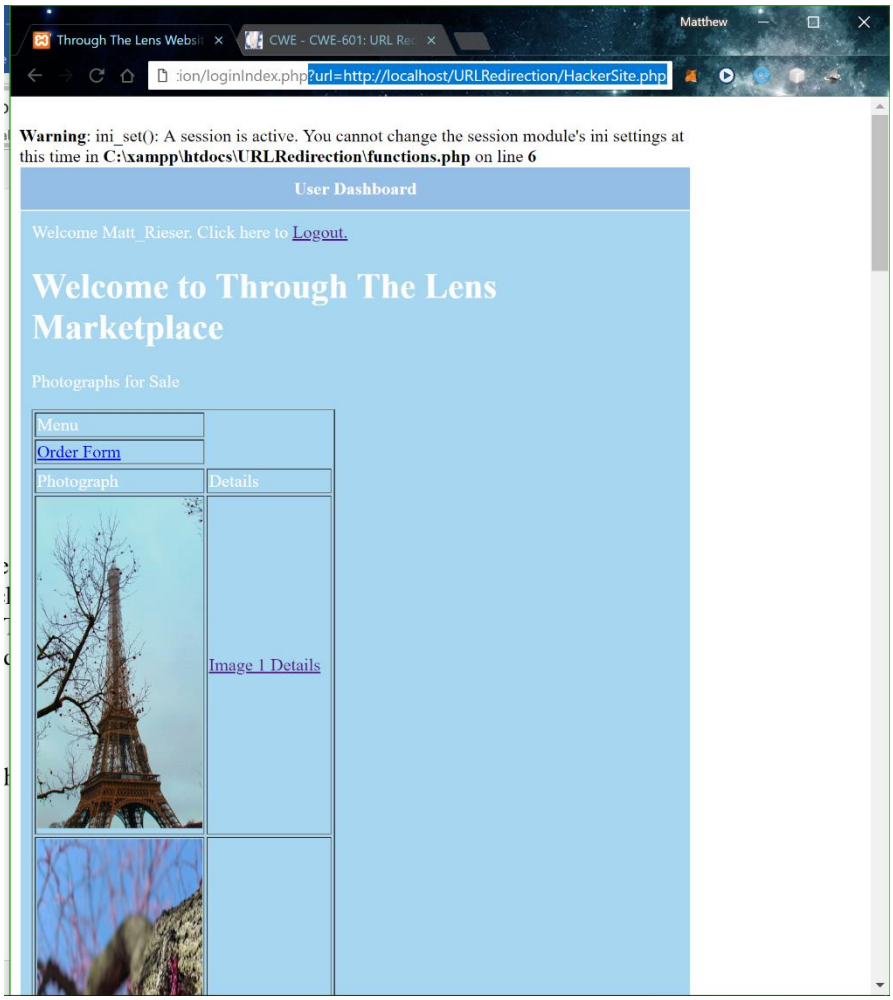
code :

$redirect_url = $_GET['url'];

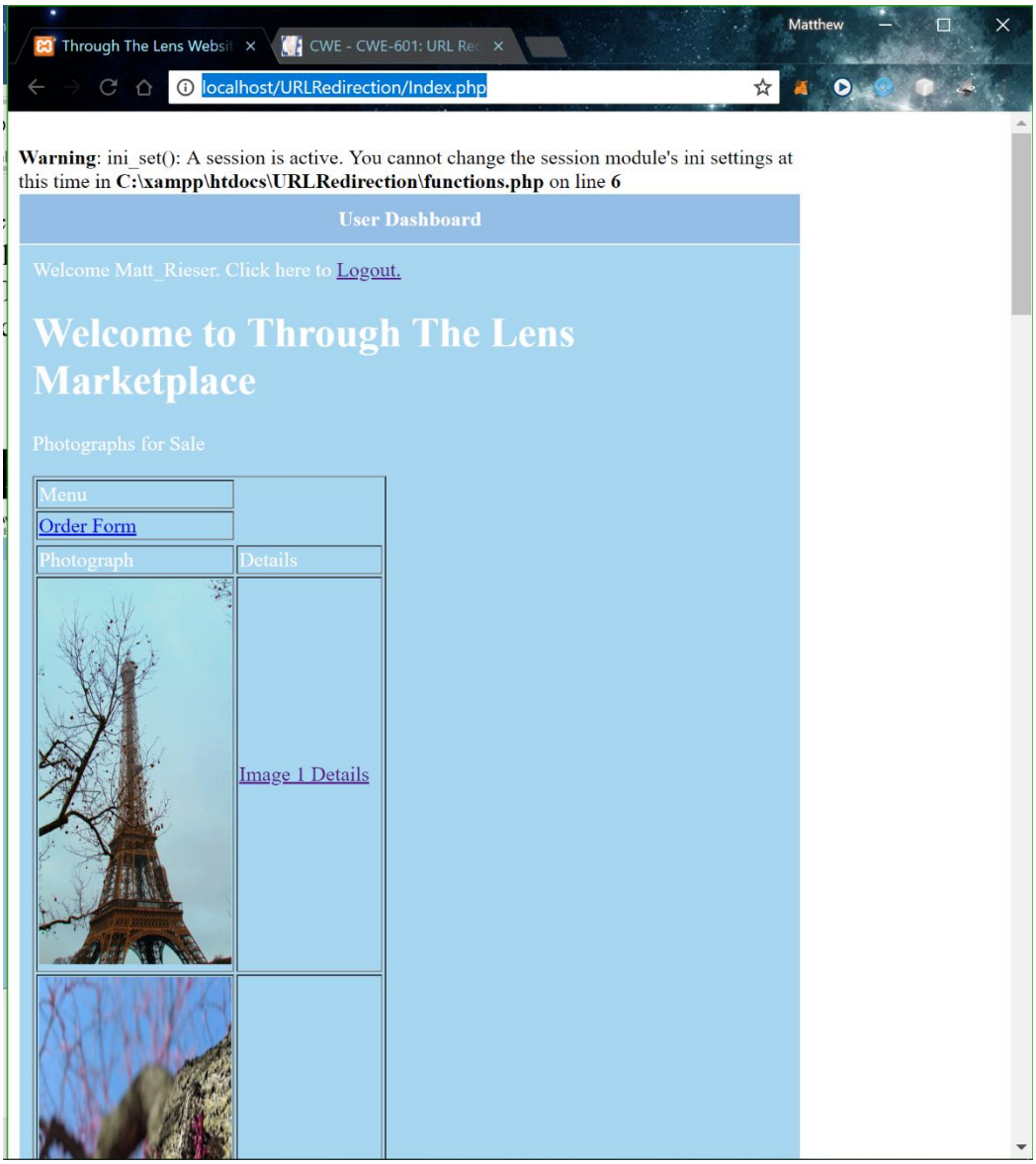
header("Location: " . $redirect_url);

with :

header("Location: Index.php");



The result was that I wasn't able to use the ?url=http://localhost/URLRedirection/HackerSite.php attack on any of the php pages like loginIndex.php and Index.php. The reason being that as long as a user's session was valid, they would be redirected to the Index.php page (homepage of my ecommerce website)



Cross-Site Scripting

Desciption - The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

Extended Desciption - Cross-site scripting (XSS) vulnerabilities occur when:

1. Untrusted data enters a web application, typically from a web request.

2. The web application dynamically generates a web page that contains this untrusted data.

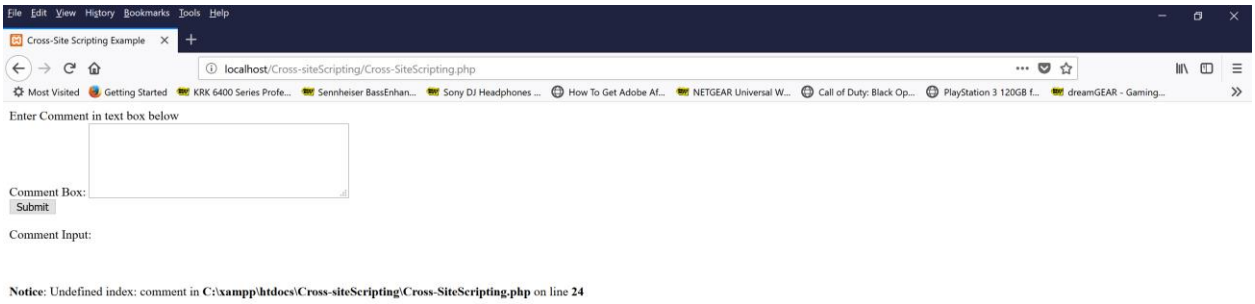
3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc.

4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.

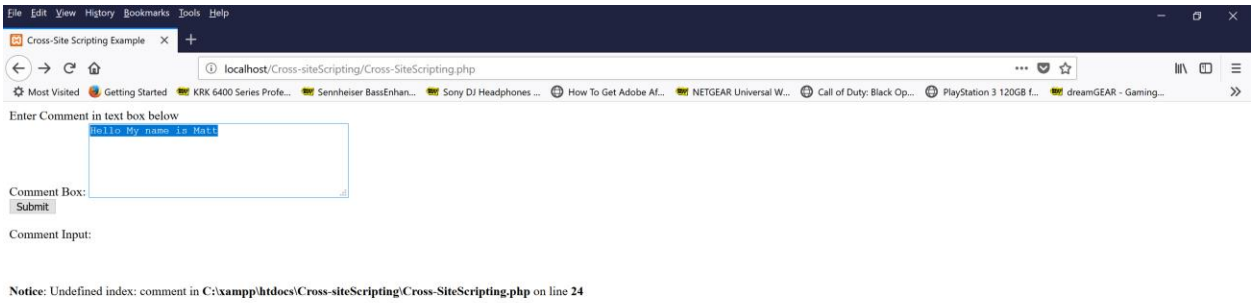
5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain.

6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain.

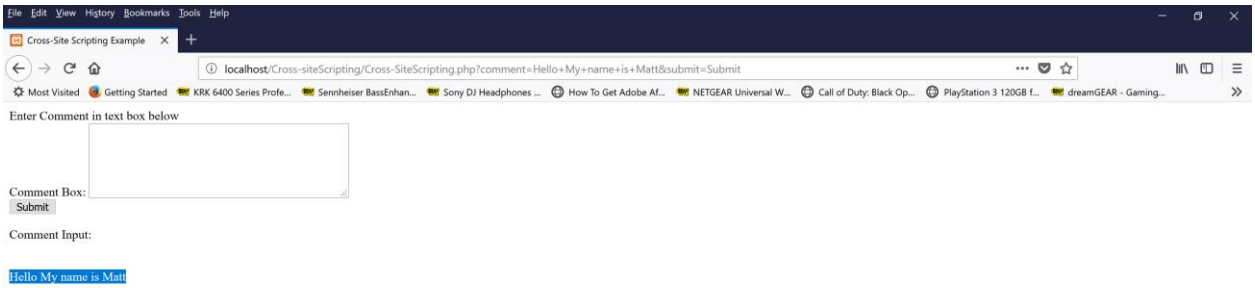
Resource - https://cwe.mitre.org/data/definitions/79.html



Above – This is meant to represent a comment box of a site that users interact with. Examples could be Facebook, Twitter, Instagram, banking site, blog, etc. This site is susceptible to the script running within the browser and being displayed to the user. If this site had a database implemented, the database could be compromised with scripting code. And all the users that visit the site could be affected by it.

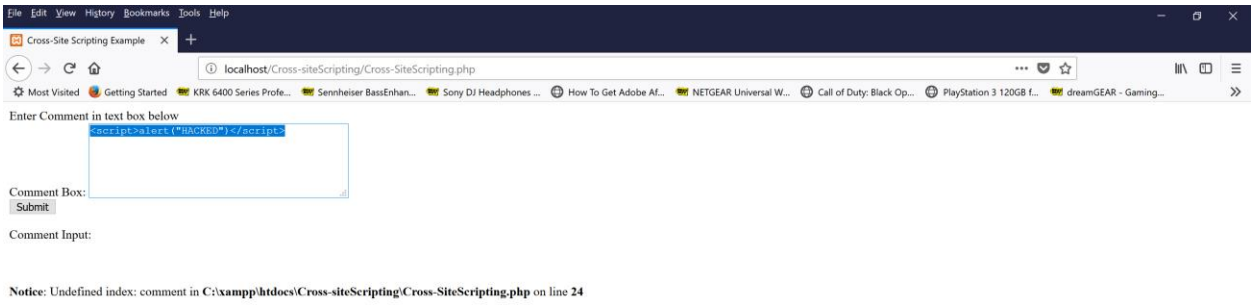


Above – Highlighted is a normal user input into the comment box - “Hello My name is Matt”

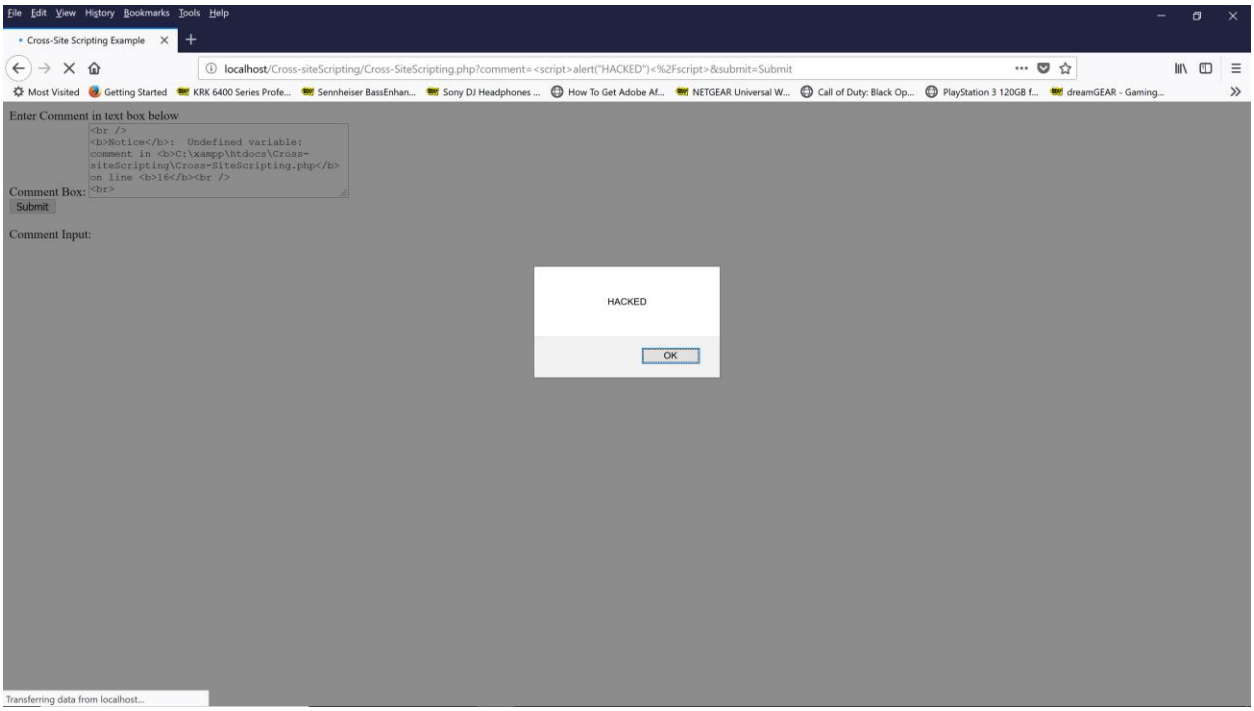


Above – Highlighted is a normal input of comment

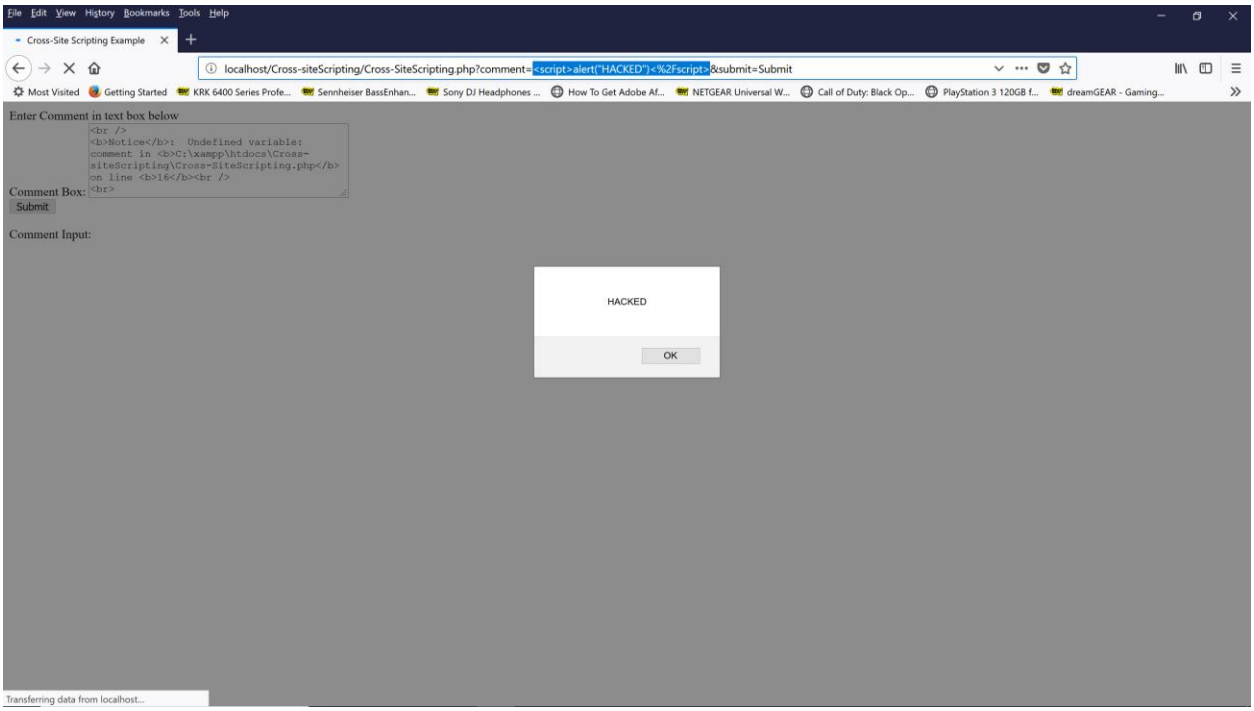
Above – Notice the page is getting the input from the url



Above – This is an example of what a malicious attack could be. The script that is being run is - <script>alert(“HACKED”)</script>. Once executed by pressing the submit button. The script will display a box saying - HACKED

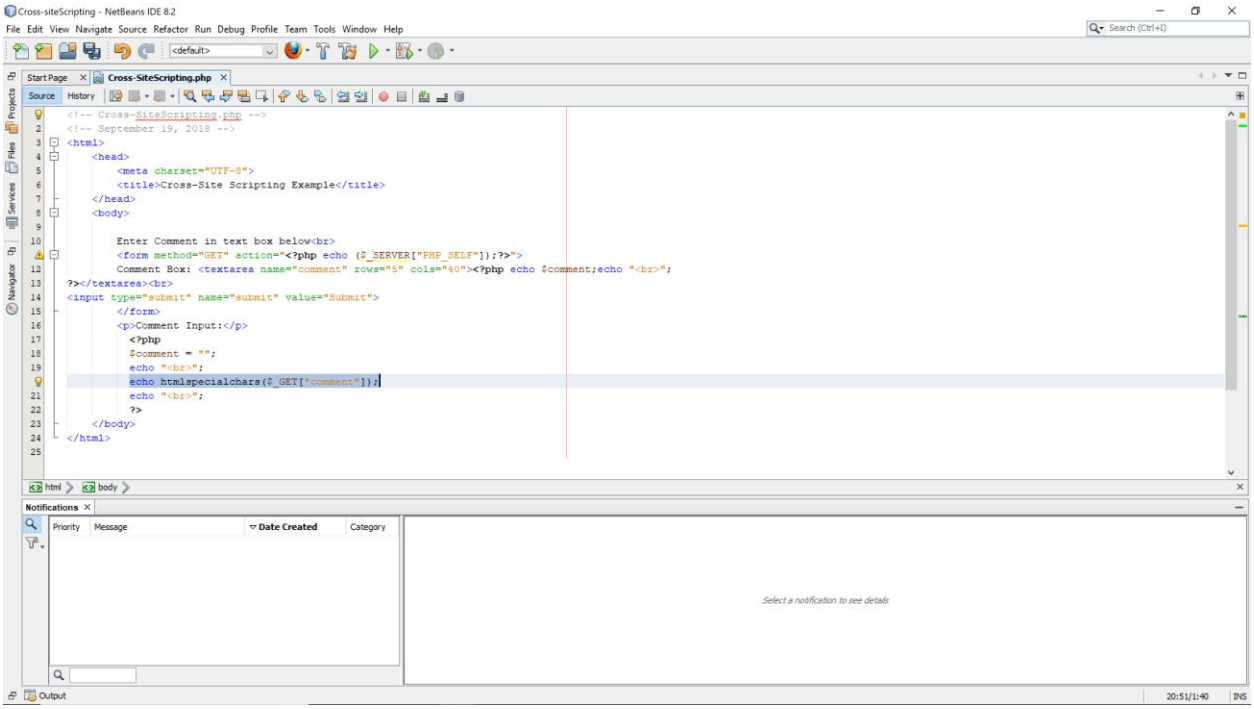


Above – The result of cross-site scripting. From this point, it could turn into a phishing attack. As the user notices the message, they want to get rid of it as quickly as they can, so they do so by clicking the ok button. This is all an attacker needs to redirect that user to their site. This site would be disguised to resemble the site the user was previously on (as to not draw suspicion). To the user, it would seem like nothing is out of the ordinary. Now the attacker could do a multiple of things to gain the users like session hijacking the user (once a user clicks on infected element they are redirected and their cookie information is stolen. Now an attacker can log in as that user) or have the user be redirected to a fake site, where they reenter their credentials that an attacker can record.



Above – Notice the file is getting the input script and running it directly, without being sanitized.

Fix



Replaced

Code :

echo $_GET["comment"];

With :

echo htmlspecialchars($_GET["comment"]);

The htmlspecialchars() function converts some predefined characters to HTML entities.

The predefined characters are:

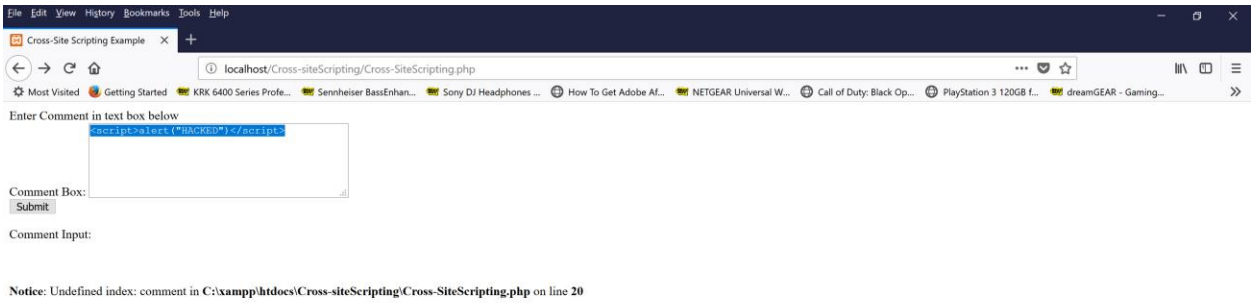
& (ampersand) becomes &

" (double quote) becomes "

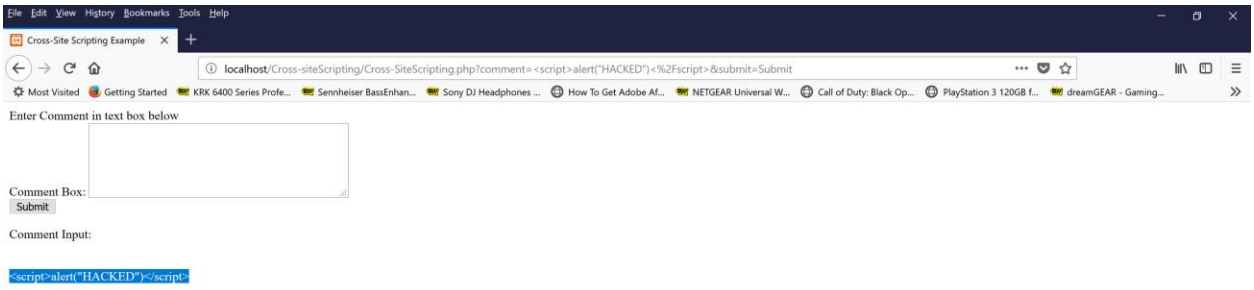
' (single quote) becomes '

< (less than) becomes <

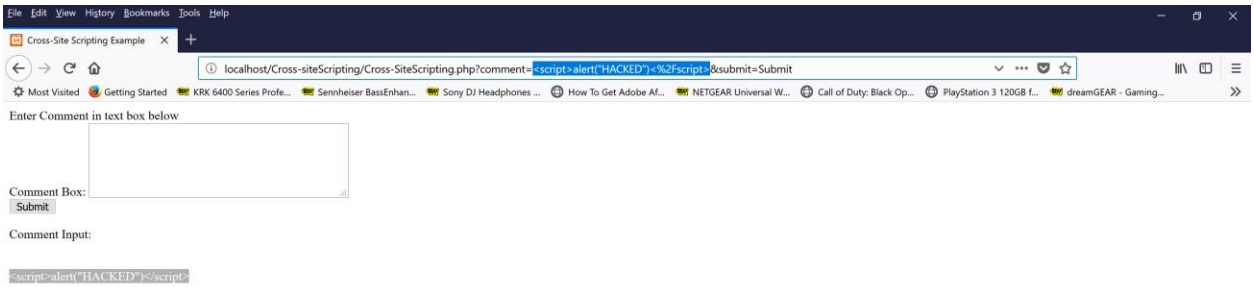
> (greater than) becomes >



Above – testing same scripting attack as before.



Above – Notice the htmlspecialchars() output the script as a string. It didn't run the script code within the browser and display the HACKED alert.



Above – Notice the URL user input is the same test as when the application was vulnerable