

WAPH-Web Application Programming and Hack-ing

Instructor: Dr. Phu Phung

Student

Name: Bhargavi Murari

Email: muraribi@mail.uc.edu



Figure 1: Bhargavi's headshot

Repository Information

Repository's URL: <https://github.com/BhargaviMurari22/waph-muraribi.git>

Hackathon 1 Report:

Overview and outcomes:

The “Cross-site Scripting Attacks and Defenses” hackathon offers participants a thorough exploration of online application security, placing special emphasis on addressing the pervasive threat of cross-site scripting (XSS) attacks. Through hands-on lab exercises, attendees gain firsthand experience in understanding the risks associated with XSS vulnerabilities. These vulnerabilities pose serious threats to user data, session cookies, and the overall system integrity of web applications, allowing the injection and execution of scripts on legitimate web pages. The hackathon aims to empower participants with the knowledge and skills necessary to both exploit and defend against XSS vulnerabilities. The primary focus is on implementing robust defense strategies to significantly mitigate these risks. Using the attached GitHub link, access the lab materials: <https://github.com/BhargaviMurari22/waph-muraribi/tree/main/labs/hackathon1>

Task 1:

LEVEL 0:

URL used: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level0/echo.php>)

Script used for attacking: `<script>alert("Level0: Hacked by BHARGAVI MURARI")</script>`

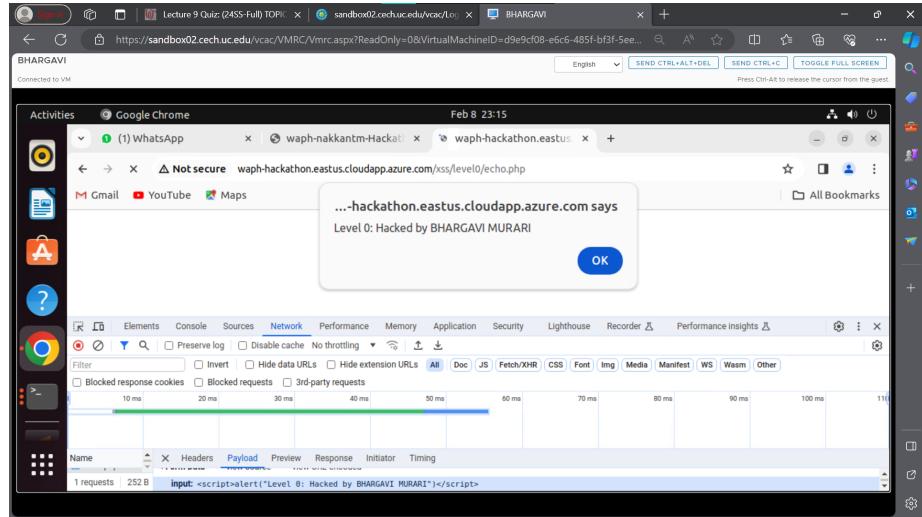


Figure 2: Hacked Level0

LEVEL 1:

URLused: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php>)

Script was written at the end of the URL as a path variable. The attacking script used was given below,

```
?input=<script>alert("Level 1: Hacked by BHARGAVI MURARI")</script>
```

LEVEL 2:

URLused: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level2/echo.php>)
This HTTP request has been converted into a basic HTML form as it does not include an input field and does not allow a path variable. The employment of a hacking script is then made easier by the attacking script being guided through this form.

```
<script>alert("Level2 Hacked by BHARGAVI MURARI")</script>
```

Source Code Guess:

```
if(!isset($_POST['input'])){
    die("{\"error\": \"Please provide 'input' field in an HTTP POST Request\"}");
} else {
    echo $_POST['input'];
}
```

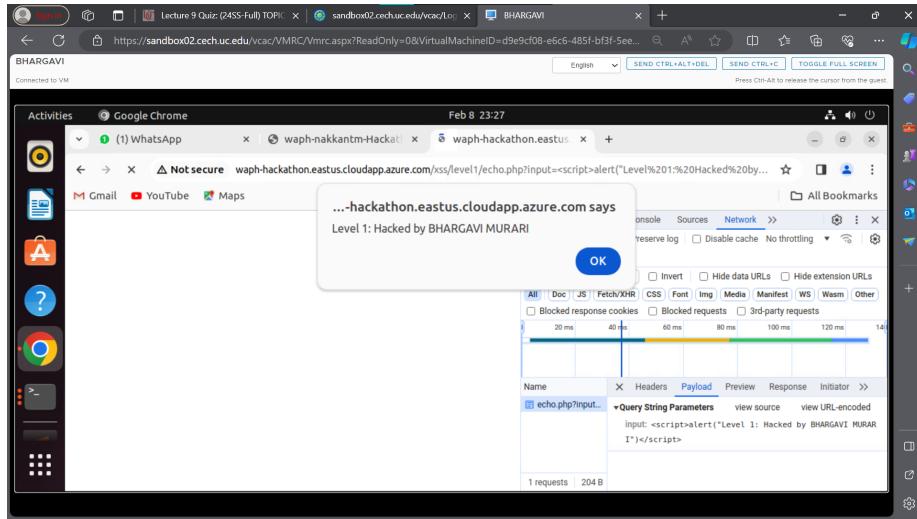


Figure 3: Hacked Level1

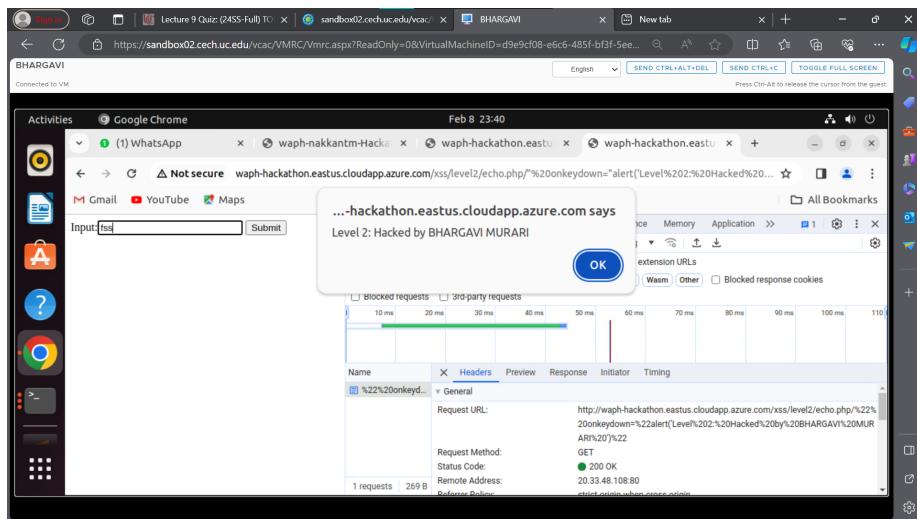


Figure 4: Hacked Level2

LEVEL 3:

URL used: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php>)
This degree of security prevents the script tag from being directly entered into the input variable. To take use of this URL, the code was broken up into several pieces and connected to cause a warning to appear on the website.

Script tag used for attacking:

```
?input=<script><script>>alert("Level 3 Hacked by BHARGAVI MURARI")</script></script>t>
```

Source code guess: Script tag may be substituted with a blank, `str_replace(['<script>', '</script>'], ' ', $input)`

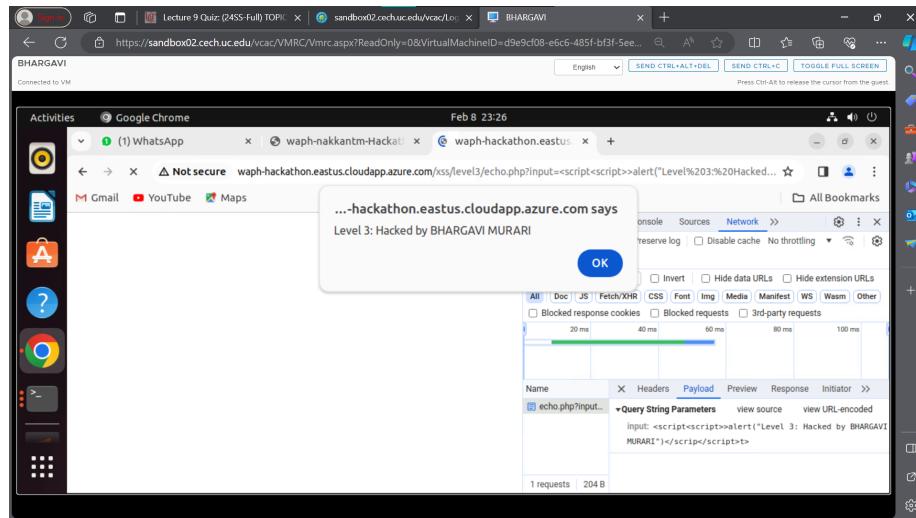


Figure 5: Hacked Level3

LEVEL 4:

URL used: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level4/echo.php>)
At this point, the script tag is fully filtered, thus it won't be able to be blocked even if the string is broken up and then joined together. Using the `onerror()` method of the tag to inject the XSS script allowed me to set off an alarm.

Script tag used:

```
?input=<img%20src="..." onerror="alert(Level 4: Hacked by BHARGAVI MURARI)">
```

Code injected:

```
?input=<button onclick="alert('Level4')"></button>
```

Source code guess:

```

$data = $_GET['input'];
if (preg_match('/<script\b[^>]*>(.*)?</script>/is', $data)) {
exit('{"error": "No \'script\' is allowed!"}');
} else {
echo $data;
}

```

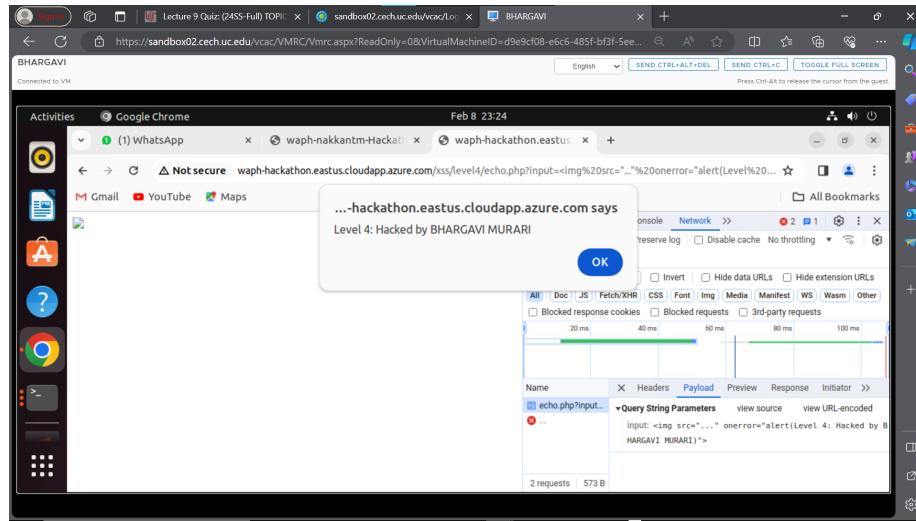


Figure 6: Hacked Level4

LEVEL 5:

URLused: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php>)
Both the alert function and the tag are filtered at this level. I combined the button tag's onerror function with unicode encoding to raise the popup alert.

Code injected:

```
?input=
```

Source Code Guess:

```

$data = $_GET['input'];
if (preg_match('/<script\b[^>]*>(.*)?</script>/is', $data) || stripos($data, 'alert') !==
exit('{"error": "No \'script\' is allowed!"}');
} else {
echo $data;
}

```

LEVEL 6:

URLused: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level6/echo.php>)

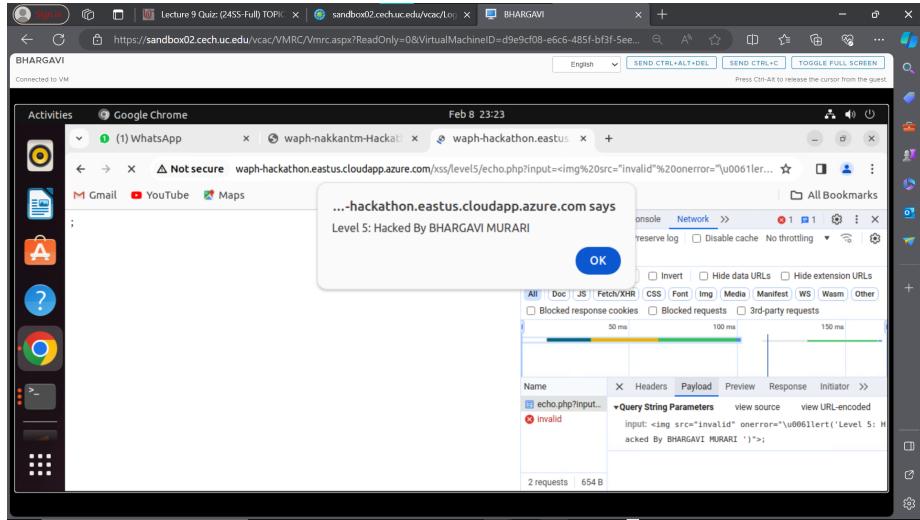


Figure 7: Hacked Level5

This level still takes input even though I think the original code uses the `htmlentities` method to translate necessary characters into their correct HTML entities. As a result, the webpage shows user input as plain text. In certain situations, JavaScript eventListeners like `onmouseover`, `onclick`, `onkeyup`, and `onmouseenter` can be used to initiate an alert. In this case, every time a key is hit in the input field, the `onmouseenter` eventListener that I used triggers an alarm on the website. The input form element is altered when the script is inserted through the URL, as shown in the illustration below. It appends to the code.

```
<form action="/xss/level6/echo.php/" onkeyup="alert('Level 6 : Hacked by BHARGAVI MURARI')">
Input:<input type="text" name="input" />
<input type="submit" name="Submit"/></form>
```

Source Code Guess:

```
echo htmlentities($_REQUEST['input']);
```

Task 2.

A. By adding code for input validation and editing the `echo.php` file, Lab 1's defensive measures against XSS were put into place. After a preliminary check to make sure the input is empty, PHP is stopped from running. When the input is verified as genuine, the `htmlentities` technique is used to clean it up. It then gets converted into the appropriate HTML characters so that it shows up on the page as text only.

```
echo.php defence <?php if(!isset($_REQUEST["data"])){ die("{"\\"error\"":
```

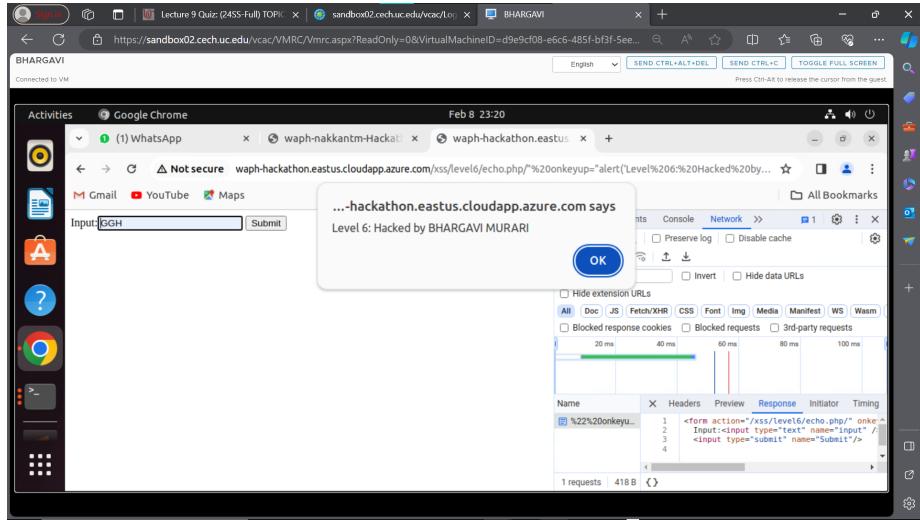


Figure 8: Hacked Level6

```
\\"Please provide 'data' field\\"); } echo htmlentities($_REQUEST['data']);
?>
```

B. Once external input points were identified, the waph-muraribi.html code underwent a significant change. After each of these inputs was verified, the result texts were edited. i) There is now validation applied to the input data for the HTTP GET and POST request forms. The request cannot be handled until the user input has been validated thanks to the introduction of a new function called validateInput.

- ii) InnerText was substituted for innerHTML in cases when the display was plain text and HTML rendering was not required.
- iii) To improve security by cleaning replies, a newly added function named encodeInput has been developed. To prevent cross-site scripting attacks, this entails transforming special characters into the proper HTML entities before adding them to the HTML content. As a result, the material is not executable and is handled as text only. InnerText is the material that is injected into a newly generated div element by the code. It is then given back as HTML content.

```
function encodeInput(input){
const encodedData = document.createElement('div');
encodedData.innerText=input;
return encodedData.innerHTML;
}
```

- iv) Updates have been made to the joke retrieval API to guarantee the accuracy

The screenshot shows a GitHub commit page for a file named echo.php. The commit title is "echo.php revised; added defense mechanism". The commit message indicates it was made by BhargaviMurari22 one minute ago and is verified. The commit hash is 5cac2298f9ddacd130e67262dd53b0fb5164bf9, and it has 1 parent commit b07b031. The code diff shows a single addition to the file:

```

diff --git a/labs/lab1/echo.php b/labs/lab1/echo.php
--- a/labs/lab1/echo.php
+++ b/labs/lab1/echo.php
@@ -1,3 +1,6 @@
<?php
echo $_REQUEST["data"];
+if(!isset($_REQUEST['data'])){
+die("Error\nPlease provide 'data' field");
+
+echo htmlentities($_REQUEST['data']);
}

```

Below the code, there are tabs for "Whitespace", "Ignore whitespace", "Split", and "Unified". A note at the bottom says "0 comments on commit 5cac229".

Figure 9: Revised echo.php with added defense mechanism

The screenshot shows a GitHub commit page for a file named waph-muraribi.html. The commit message is "Interaction with forms". The code diff shows numerous changes, with many additions and deletions. The changes are primarily related to form handling and validation. Some of the visible changes include:

```

diff --git a/waph-muraribi.html b/waph-muraribi.html
--- a/waph-muraribi.html
+++ b/waph-muraribi.html
@@ -54,51 +54,31 @@
@@ -54,31 +51,31 @@
<h3> Student: Bhargavi Murari</h3>
<div>
    <i>Interaction with forms</i>
    <div>
        <form with an HTTP GET Request/>
        <form action="echo.php" method="GET">
-            Your input: <input name="data">
+            <input type="text" name="data" id="data-get" onkeyup="console.log('Just clicked a key')">
-            Your input: <input name="data"> ->
+            Your input: <input name="data"> ->
            <input type="submit" value="Submit">
        </form>
    </div>
    <div>
        <i>Form with an HTTP post Request</i>
        <form action="echo.php" method="POST">
-            Your input: <input name="data">
+            <input type="text" name="data" id="data-post" onkeyup="console.log('Just clicked a key')">
-            <input type="submit" value="Submit">
+            <input type="submit" value="Submit">
    </div>

```

Figure 10: waph-muraribi.html defense code

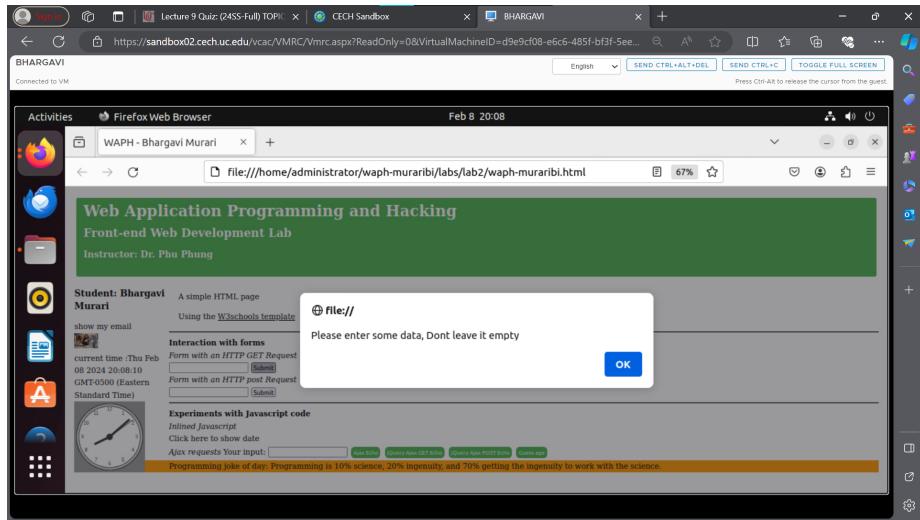


Figure 11: Validating inputs

The screenshot shows a code editor displaying a diff of a file. The changes are as follows:

```

Showing 1 changed file with 38 additions and 19 deletions.
@@ -152,6 +158,19 @@ <h3> Student: Bhargavi Murari</h3>
152     ctx.translate(radius, radius);
153     radius+=radius * 0.05;
154     setInterval(drawClock, 1000);
155 +
156     function validateInput(inputId) {
157         var input = document.getElementById(inputId).value;
158         if(input.length == 0){
159             alert("Please enter some data, Dont leave it empty");
160             return false;
161         }
162         return true;
163     }
164     function encodeInput(input){
165         const encodedData = document.createElement('div');
166         encodedData.innerText=input;
167         return encodedData.innerHTML;
168     }
169     function drawClock(){
170         const encodedData = document.createElement('div');
171         encodedData.innerText= input;
172         return encodedData.innerHTML;
173     }
174     drawClock();
175     drawFace(ctx, radius);
176     drawNumbers(ctx, radius);

```

Figure 12: Modified html to text



Figure 13: Validated Output

```

Showing 1 changed file with 38 additions and 19 deletions.
      Whitespace Ignore whitespace Split Unified
  55 +           <input type="text" name="data" id="data-get" onkeyup="console.log('Just clicked a key')">
  56 +         <!-->
  57             your input: <input name="data" -->
  58                 </form>
  59             </div>
  60             <div>
  61                 <>Form with an HTTP post Request</>
  62 -             <form action="echo.php" method="POST">
  63 -                 Your input: <input name="data">
  64 +             <form action="echo.php" method="POST" onsubmit="return validateInput(data-post)">
  65 -                 <input type="text" name="data" id="data-post" onkeyup="console.log('Just clicked a key')">
  66 +                 <input type="submit" value="Submit">
  67             </form>
  68             </div>
  69             <br>
  70             <>Experiments with Javascript code</><br>
  71             <>Inlined Javascript</>
  72 -             <div id="date" onclick="document.getElementById('date').innerHTML=date()">Click here to show date</div>
  73 +             <div id="date" onclick="document.getElementById('date').innerHTML=date()">Click here to show date</div>
  74             </div>
  75             <>Ajax requests</>
  76             <div>
  77                 <input name="data" onkeyup="console.log('you have pressed a key') id='data'">

```

Figure 14: Encode input function

of the results that are received. New validations have been added, which verify that jokes in JSON are not empty. An error notice appears when the result is null.

```

Showing 1 changed file with 38 additions and 19 deletions.
      Whitespace Ignore whitespace Split Unified
  88  85     async function guessAge(name){
  89  86         const response = await fetch(`https://api.agify.io/?name=${name}`);
  90  87         const result = await response.json();
  91  88         // `${'#response'}).html(`Hi ${name}, your age should be ${result.age}`);
  92 -         `${'#response'}).html(`Hi ${name}, your age should be ${result.age}`);
  93 +         if(result.age == null || result.age==0)
  94 +             return `${'#response').text("An error occurred at this moment")`;
  95 +         `${'#response'}).html(`Hi ${name}, your age should be ${result.age}`);
  96         )
  97         function queryAjax() {
  98             var input = $('#data').val();
  99             if (input.length == 0) return;
  100            $.get(`echo.php?data=${input}`,
  101            function(result) {
  102                `${'#response'}).html(`Response from server: ${result}`);
  103                `${'#response'}).html(`Response from server: ${encodeInput(result)}');
  104            }
  105        );
  106        $('#data').val("");
  107    }

```

Figure 15: Defense to displaying joke

- v) It is confirmed that the asynchronous method `guessAge`'s received result is either empty or non-zero. In addition, the data entered by the user is checked to make sure it is not null or empty. An error notification appears on each of these occasions.

```

if(result.age==null || result.age==0)
return ${"#response"}
.text("An error occurred at this moment, So age can't be displayed");
${"#response").text("Hello "+name+" ,your age should be "+result.age);

```



Figure 16: Validated Guess Age