WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Ruthvik Suvarnakanti

 ${\bf Email: \ suvarnrk@mail.uc.edu}$



Figure 1: Ruthvik Suvarnakanti

Hackathon 1: Cross-Site Scripting Attacks and Defenses

Overview: The Hackathon centers around understanding and addressing XSS (Cross-Site Scripting) attacks, with participants engaging in two distinct tasks. In the Attack Phase (Task 1), participants are tasked with uncovering and exploiting XSS vulnerabilities within a designated website (http://waphhackathon.eastus.cloudapp.azure.com/xss/). This site is structured with six levels of vulnerabilities, designed to provide insight into the mechanics of XSS attacks. Following this, in the Defense Phase (Task 2), participants focus on implementing secure coding practices to mitigate XSS threats. Guided by OWASP guidelines, they learn techniques such as input validation and output sanitization to fortify their applications against XSS vulnerabilities effectively. Upon completing both phases, participants document their discoveries and solutions using Markdown format. Utilizing tools like Pandoc, they generate a comprehensive PDF report detailing their experiences, including the vulnerabilities identified, exploitation methods employed, and the strategies adopted to mitigate risks. Overall, the Hackathon serves as a practical learning platform, offering participants hands-on experience in identifying, exploiting, and safeguarding against XSS vulnerabilities, thus fostering a deeper understanding of web security protocols.

Link to the repository: https://github.com/suvarnrk/waph-suvarnrk/blob/ma in/labs/hackathon1/README.md

Task 1 : ATTACKS

Level 0

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

<script>alert("Level 0 : hacked by Ruthvik Suvarnakanti")</script>



Figure 2: Level 0

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php attacking script is passed as a pathvariable at the end of the URL

?input=<script>alert("Level 1: Hacked by Ruthvik Suvarnakanti")</script>



Figure 3: Level 1

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

Since the HTTP request for this URL doesn't provide an input field or accept path variables, a workaround was needed. The URL was linked to a simple HTML <form>. Through this form, the attacking script is transmitted. This method allows for a more structured and controlled injection of malicious scripts, aiding in the exploration and exploitation of XSS vulnerabilities within the web application. By embedding the attacking script directly into the form submission, participants can interact with the website and observe how their injected code affects it. This approach ensures that XSS attacks are carried out within the web application's environment, giving participants a clearer understanding of the vulnerabilities and their potential consequences for the application's security.

<script>alert("Level 2: Hacked by Ruthvik Suvarnakanti")</script>

Source code Guess of echo.php:

```
if(!isset($_POST['input'])){
```

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



Figure 4: Level 2

URL: http://waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php

In this level, if the *script>* tag is directly passed in the input variable, it is filtered out. Therefore, to carry out an attack on this URL, the malicious code had to be divided into multiple parts and then appended together to trigger an alert on the webpage. This approach bypasses the filtering mechanism and allows the attacker to execute their payload successfully.



Figure 5: Level 3

?input=<script<script>>alert("Hacked by Ruthvik Suvarnakanti")</script>t>
Source code Guess of echo.php:

```
str_replace(['<script>', '</script>'], '', $input)
```

URL: http://waph-hackathon.eastus.cloudapp.azure.com/xss/level4/echo.php

In this level, the filtering mechanism completely blocks the <script> tag, even if it's attempted by breaking the string and concatenating it. To inject the XSS script, I utilized the onerror() attribute of the tag. By leveraging this attribute, I triggered an alert to be raised on the webpage. This approach circumvents the filtering mechanism, allowing the XSS script to execute successfully.

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

Source code guess of echo.php:

```
$data = $_GET['input']
if (preg_match('/<script\b[^>]*>(.*?)<\/script>/is', $data)) {
     exit('{"error": "No \'script\' is allowed!"}');
      }
else
      echo($data);
• • • Ø waph-hackathon.eastus.cloud × +
                                                                                                ~
← → X ▲ Not Secure | waph-hackat
                                                                t(%27Level%204%20:%20Hacked%... 🖞 🕁 🛓 🖪 📵 Update 🔅
                                          hon.eastus.cloudapp.azu
∎1 🛞 E X
    ▼ Q □ Pr
                              ((<sup>9</sup>
                                 Î
            erve log
                           ling
```

 2
 Q
 Peeserve
 Daab cache No browting * % 1.2.
 S. 2.
 Sec.
 Sec.</td

Figure 6: Level 4

URL: http://waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php

In this level, both the <script> tag and the alert() method are filtered out. To trigger a popup alert, I employed a combination of Unicode encoding and the onerror() method of the tag. By utilizing these techniques together, I successfully raised a popup alert on the webpage. This method effectively bypasses the filtering mechanisms in place, allowing the XSS payload to execute.

```
?input=<img src="invalid"
    onerror="\u0061lert(Level 5: Hacked By Ruthvik Suvarnakanti)">
```



Figure 7: Level 5

URL: http://waph-hackathon.eastus.cloudapp.azure.com/xss/level6/echo.php

In this level, user input is accepted, but it appears that the source code utilizes the htmlentities() method to convert all relevant characters into their corresponding HTML entities. This ensures that the user input is rendered strictly as text on the webpage. To trigger an alert on the webpage in this scenario, I utilized JavaScript event listeners such as onmouseover(), onclick(), and onkeyup(). Specifically, I opted for the onkeyup() event listener, which generates the alert on the webpage whenever a key is pressed within the input field. This approach allowed me to bypass the HTML entity conversion and successfully execute the desired action.

```
/" onkeyup="alert('Level 6 : Hacked by Ruthvik Suvarnakanti')"
```

on passing the above script in the url , this will append to the code and manipulates the input form element as below.

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

🔍 🔍 📀 waph-hacks	athon.eastus.cloud × +									~
← → C ▲ Not Sec	ure waph-hackathon.ea	astus.cloudapp.azure.com/xss/	evel6/echo.php/"onl	keyup="alert('Lev	el%206%20:%20h	acked%20by%20	Ruthvik%20Suvarn	akanti')" 🖞 😭		Update :
Input[{	Submit	wapt Level	-hackathon.eastus	.cloudapp.azure Suvarnakanti	.com says ОК					
K Lo Elements Consol	e Sources Network I	Performance Memory Applicat	ion Security Lig	hthouse Recorde	r 🛆 Performance	insights 凸			1	⊛ : ×
● Ø 🔻 🤇 🗆 Preserv	ve log 🗌 Disable cache No	o throttling 🔻 🙃 土 🛓								۲
Filter	Invert 🗌 Hide data URLs 🗌	Hide extension URLs All Fetch/	HR JS CSS Img	Media Font Doc	WS Wasm Manif	est Other DBlo	cked response cookies	Blocked requests	3rd-party	requests
10 ms 20 ms	30 ms 40 ms	50 ms 60 ms	70 ms 80 ms	90 ms	100 ms 111	i ms 120 ms	130 ms	140 ms 150 m	s 160 r	ns 170 m
Name	X Headers Preview	Response Initiator Timing C	ookies							
%22onkeyup=%22alert('Leve	▼ General									
	Request URL: Request Method: Status Code: Remote Address: Referrer Policy:	http://waph-hackat GET 200 OK 20.33.48.108:80 strict-origin-when-o	hon.eastus.cloudapp.azu ross-origin	ire.com/xss/level6/ec	10.php/%22onkeyup=1	622alert("Level%206	%20:%20hacked%20by	%20Ruthvik%20Suva	makanti')%22	
	▼ Response Headers	Baw								
	Connection: Content-Encoding: Content-Length: Content-Type:	Keep-Alive gzip 165 text/html; charset=	JTF-8							
1 requests 417 B transferred	Date:	Wed, 07 Feb 2024	04:27:59 GMT							

Figure 8: Level 6

source code guess of echo.php:

echo htmlentities(\$_REQUEST('input'));



Figure 9: Level 6 after injecting XSS code

TASK 2 : DEFENSE

A . echo.php

The echo.php file in Lab 1 has been updated to include input validation and XSS defense measures. Initially, it checks whether the input is empty. If it is, the PHP execution stops. If the input is valid, the htmlentities() function is utilized to sanitize the input data. This function converts special characters into their HTML entities, ensuring that the text is displayed as plain text on the webpage. This action serves to protect against XSS attacks by neutralizing any potentially harmful scripts embedded within the input.

```
if(empty($_REQUEST["data"])){
    exit("please enter the input field 'data'");
    }
    $input=htmlentities($_REQUEST["data"]);
    echo ("The input from the request is <strong>" .$input. "</strong>.<br>");
```

→ C (<u> </u>	github.com/suvarnrk/waph-suvarnrk/commit/ad68351af8d2e2c022506a3cbc835657be534ee5						•	1 1		9 🤇	Update
Commit												
Update e 안 main	echo.	php								Bro	wse fi	iles
这 suvarn	hrk co	emmitted 2 minutes ago (Verified)							t c80249	9 0 com	it ad	68351
howing 1 ch	hange	ed file with 11 additions and 2 deletions.			w	hitespa	ice		whitespa	ce Sj	olit (L	Unified
✓ 13 ■		labs/lab1/echo.php (-										
4	2 + 3 + 4 + 5 + 6 + 7 + 8 - 9 - 10 - 11 12 13	<pre>-/php Edua = 5_RCOUST["input"]; extu Stat; inputDat =RCOUST["dat"]; rchu The equested input is strongs". SimputData . "// firemptis.Recurrent/Stat"]); // interptis.Recurrent/Stat"]; // interptis.Recurrent/Stat"]; // input = NationtifiedS.RECURST["dats"]; // inputStateStateStateStateStateStateStateSta</pre>										
commen	nts or	n commit ad68351								₿ Loc	k conv	ersati
•	Write											
- I												

Figure 10: Defense echo.php

B . Lab 2 front-end part

The waph-suvarnrk.html file underwent a thorough review and update, focusing on identifying and securing areas where external input is accepted. Validation procedures were implemented to ensure the integrity of input data, while measures were taken to sanitize output text to enhance security. i) In particular, when dealing with HTTP GET and POST request forms, input data is meticulously validated. A new function named validateInput() was introduced for this purpose. This function mandates that users provide input text before executing their request. This precautionary measure significantly reduces the risk of receiving invalid or potentially malicious input, thereby fortifying the overall security posture of the application.

←	→ c		github.com/suvarnrk/waph-suvarnrk/commit/ffb00f04757cb93da6abc86fb961ba4db05dec00		୯ ∆ ☆ C	J 🚯 (Update 🚦
ſ	dofon	oo 1					
	ueren	ce-i				Browse	tiles
	P main						
(📀 suv	arnrk	ommitted now Verified		1 parent ad68351		ffbeefe
sł	nowing	1 chan	sed file with 2 additions and 2 deletions.	Whitespace	Ignore whitespace		Unified
		4	💵 labs/lab2/waph-suvarnrk.html 🖵				
	143	143	<bs></bs> 				
			<i>> Form with an HTTP GET request</i>				
			- <form action="/echo.php" method="GET"></form>				
		146	<pre>+ <form action="/echo.php" method="GET" onsubmit="return validateInput('data-get')"></form></pre>				
			Your Input: <input name="input"/>				
			<input type="submit" value="Submit"/>				
	149	149					
			@@ -152,7 +152,7 @@ <h3> Student: Ruthvik Suvarnakanti</h3>				
			<1> FORM WITH all HITP FOST request(<)1> of an entire and index in the index index in the index index in the index index in the index i				
	155	134	Your action- years, pip methods der name etnigent / ankeynesser ansale lan('You pressed a key')'s				
			 Your Input: "Input mane linput" tones" est" of "data-ent" operates "onsole.lon('You pressed a key 	v')">			
	156	156	<pre><inout type="submit" value="Submit"></inout></pre>				
0	comm	nents o	n commit ffb@f0		e) Lock co	nversation
		Write					

Figure 11: Defense waph-suvarnrk.html

Google	e Chron	ne File Edit View History Bookmarks Profiles Tab Window Help	🍪 📭 🗢 Q 😫 Wed Feb 7 1
•	🚮 N	fail-Si x 😌 College x 🔄 Login x 😌 Develop x 🚫 Search x 💓 Files x 🕠 Update x 🔯 ChatGP x	O Update x 🔕 sandbo: x O waph-s: x +
> c		github.com/suvarnrk/waph-suvarnrk/commit/1a7601c4bc613635d696d06380d1edaf9d0bc586	😉 🎂 🖈 🔲 🚯 🕠
owing	1 chan	ged file with 38 additions and 10 deletions.	Whitespace Ignore whitespace Split Un
		<i>> Form with an HTTP GET request</i>	
		<form action="/echo.php" method="GET" onsubmit="return validateInput('data-get')"></form>	
147		Your Input: <input name="input"/>	
		Your Input: <input id="data-get" name="input" onkeypress="console.log('You pressed a ke</p></td><td>2y')"/>	
		<input type="submit" value="Submit"/>	
		<div></div>	
		<i>> Form with an HTTP POST request</i>	
		- <form action="/echo.php" method="GET" name="echo_post"></form>	
155		 Your Input: <input id="data_get" name="input" onkeypress="console.log('You p </td><td>pressed a key')" type="text"/>	
		+ <form action="/echo.php" method="POST" name="echo_post" onsubmit="return validInput('data-post')</p></td><td>"></form>	
	174	Your Input: <input id="data-post" name="input" onkeyup="console.log('You pr</p></td><td>'essed a key')" type="text"/>	
		<input type="submit" value="Submit"/>	
158	177		
		<pre>xhttp.onreadystatechange=function(){</pre>	
		if{this.readyState==4 && this.status==280}{	
		<pre>console.log("Received data="+xhttp.responseText);</pre>	
188		 document.getElementById("response").innerTexte 	
189		 "Response from server:"+xhttp.responseText; 	
		+ document.getElementById("response").innerHtML= encodeInput(
	208	<pre>+ xhttp.responseText);</pre>	
192	211		
		\$("#data").val("");	
		function printResult(result){	

Figure 12: Validating HTTP requests input

ii) In the code, we made a change from using .innerHTML to .innerText in situations where HTML rendering wasn't necessary, and only plain text needed to be shown. This adjustment ensures that any text content is treated strictly as text, without any interpretation of HTML tags or rendering of elements. By using .innerText, we reduce the risk of unintended HTML injection or XSS vulnerabilities, making the display of text content safer within the application.



Figure 13: modifying innerHTML to innerText

iii) A new function called encodeInput() was introduced to sanitize the response. This function converts special characters into their respective HTML entities before inserting them into the HTML document, thereby preventing cross-site scripting attacks. This ensures that the content is displayed purely as text and cannot be executed. Moreover, the code creates a new <div> element and inserts the content as innerText into this newly created element. Subsequently, this content is returned as the HTML content. This method guarantees that any potentially harmful content is properly encoded and displayed securely within the HTML document, thus reducing the risk of XSS vulnerabilities. Utilizing innerText ensures that the content is treated strictly as text, preventing any unintended HTML rendering or script execution.

```
function encodeInput(input){
```

}

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

iv) Additional validations have been introduced for the API https://v2.jokeapi .dev/joke/Programming?type=single, used to fetch jokes. These validations now verify if the received result and the result.joke in the JSON response are not empty. If either of these values turns out to be null or empty, an error message is generated. This enhancement ensures that only valid and non-empty joke data is handled and presented, thereby enhancing the application's reliability and user experience.

```
if (result && result.joke) {
    var encodedJoke = encodeInput(result.joke);
    $("#response").text("Programming joke of the day: " +encodedJoke);
    }
```

```
else{
```

```
$("#response").text("Could not retrieve a joke at this time.");
```



Figure 14: handling Joke API and Guess age API

v) In the asynchronous function guessAge(), new validations have been added. These validations ensure that both the received result and the user-entered input are not empty, null, or equal to 0. If either the result or the input is found to be in such a state, an error message is thrown. These measures are put in place to uphold data integrity and guarantee the proper functioning of the function across different scenarios, thereby improving its reliability and usability.



Figure 15: Guess age function in case error is thrown