Faculty of Computing and Information Technology

Department of Information and Communication Technology

Bachelor of Information Technology (Hons) in Information Security Year 3

Academic Year 2020/2021

BAIT2183 SOFTWARE SECURITY ASSIGNMENT

(To be filled by students)

Web Application Title	**:**	**Picture Sharing Website**
Tutorial Group	**:**	**RIS3 Group 4**
Name	**:**	**Lee Yong Hao** **19WMR09371**
Tutor	**:**	**Ms. Tan Wai Beng (Janice)**

(To be filled by supervisor)

Date/Time Received :	**[ ] On time / [ ] Late**
Remarks : *(If any)*	
Signature :	

Declaration

I confirm that I have read and shall comply with all the terms and conditions of TAR University College's plagiarism policy.

I declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work.

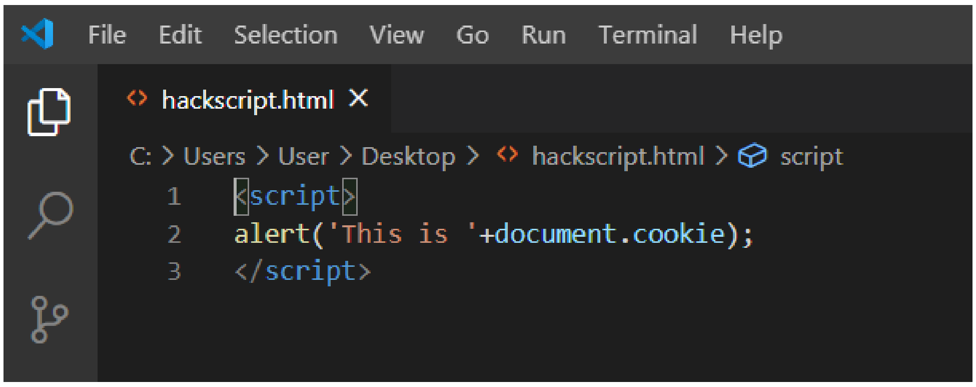
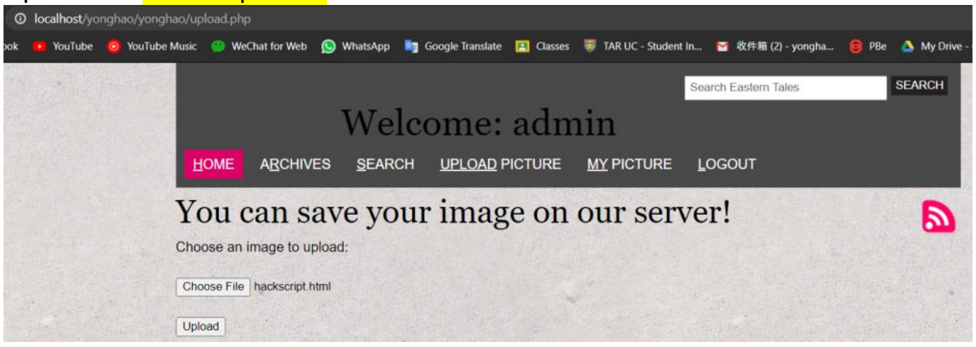
Signature	**:**	
Name	**:**	**Lee Yong Hao**
Date	**:**	**1/9/2020**

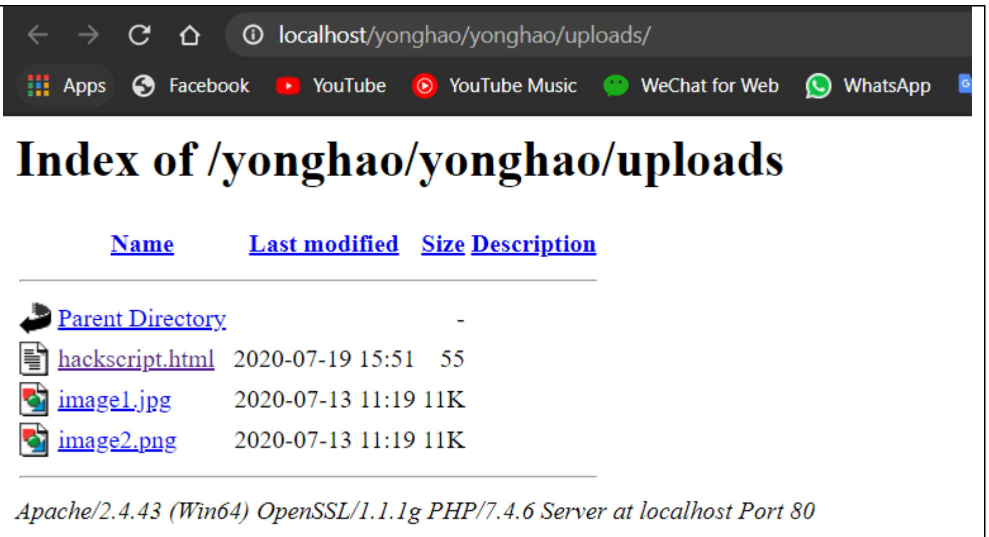
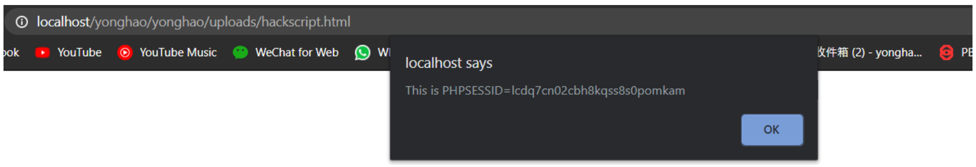
<u>Part 1</u>

Name : <u>Lee Yong Hao</u> Tutorial Group: <u>RIS3 G4</u>

<u>Vulnerability 1</u>

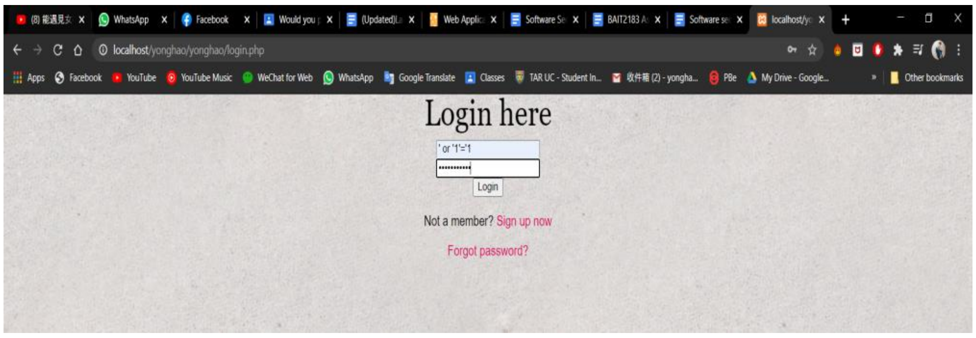
Attack	**Cross-site Scripting (XSS)**
Description	Cross-site scripting (XSS) is a vulnerability that permits an attacker to inject code (typically HTML or JavaScript) into contents of a website not under the attacker's control. When a victim views such a page, the injected code executes in the victim's browser. Thus, the attacker has bypassed the browser's same origin policy and can steal the victim's private information associated with the website in question.
Page URL	http://localhost/yonghao/yonghao/upload.php
Instance / Example	File Upload Vulnerability

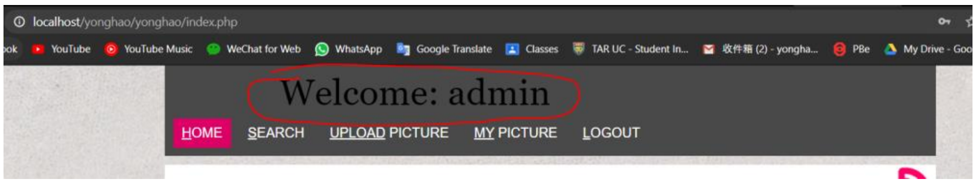
Methods	First, we have to code a malicious script and then upload it as a file into the account. The file will be uploaded to MY PICTURE. Then, open the file to execute the malicious script and get the user session ID.
	
	Example this is hackscript.html that can be run to alert the session id from the user.
	
	 File Edit Selection View Go Run Terminal Help
	hackscript.html
	C: > Users > User > Desktop > hackscript.html > script
	`<script>`
	`alert('This is '+document.cookie);`
	`</script>`
	
	Upload the hackscript.html into the database.
	
	 localhost/yonghao/yonghao/upload.php
	ok YouTube YouTube Music WeChat for Web WhatsApp Google Translate Classes TAR UC - Student In... 收件箱 (2) - yongha... PBe My Drive -
	Search Eastern Tales SEARCH
	HOME ARCHIVES SEARCH UPLOAD PICTURE MY PICTURE LOGOUT
	Choose File hackscript.html
	Upload
	Welcome: admin
	You can save your image on our server!
	Choose an image to upload:
	
	After updating the hackscript.html, the file will be stored in MY PICTURE there.

	 localhost/yonghao/yonghao/uploads/ **Index of /yonghao/yonghao/uploads** Name · Last modified · Size · Description; Parent Directory -; hackscript.html 2020-07-19 15:51 55; image1.jpg 2020-07-13 11:19 11K; image2.png 2020-07-13 11:19 11K; *Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.6 Server at localhost Port 80*	
Render	When the user runs the malicious file. It will pop out the session ID that the attackers can use this session ID to hijack the client's current session. During the session hijacking, a malicious hacker places himself in between your computer and the website's server, while you are engaged in an active session.  localhost/yonghao/yonghao/uploads/hackscript.html — localhost says: This is PHPSESSID=lcdq7cn02cbh8kqss8s0pomkam [OK]	
Impacts	When the attackers can simply upload any malicious files, they will impact the database. When the malicious files have already uploaded in the database, all the user credentials will be leaked from the database. If the attackers upload the malicious files successfully, the attackers can use the malicious file that included the script file to hijack the client's current session.	
Security Requirements	**Security Requirement**	**Justifications**
	1. Limit Execution	The system should not allow users to upload and execute script files to the system. So, we just need to set the limit only to allow gif, png, jpg, jpeg, and txt files. So, the script file like php,html cannot be uploaded to our system, then they cannot run it. If the website does not limit it, the attackers can simply upload any malicious files to the website.

	2. Audit Log	A log can be included within the system to show when the user login and logout so that if anything happens, it will be easier to troubleshoot who is responsible for the incident

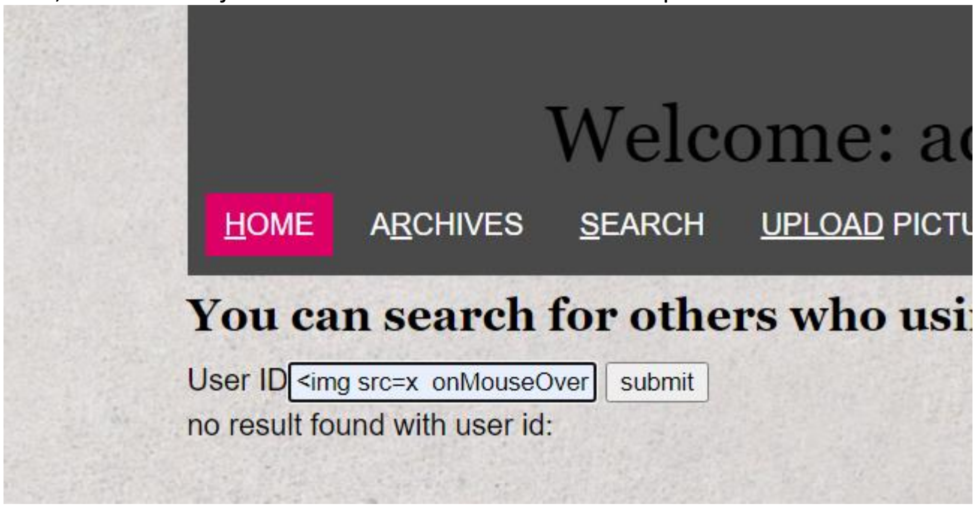
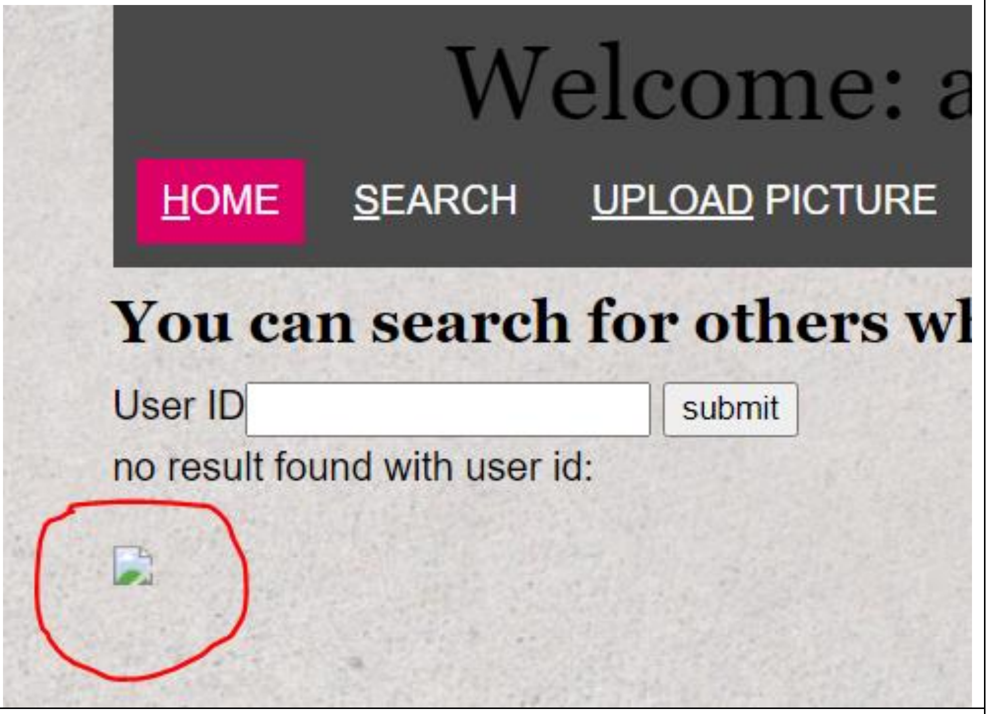
Vulnerability 2

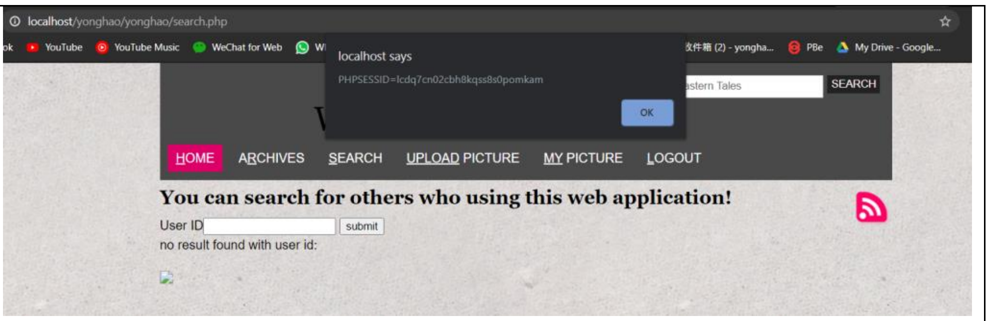
Attack	**SQL Injection**
Description	SQL Injection is a trick to inject SQL query/command as an input possibly via web pages. Many web pages take parameters from web users, and make SQL queries to the database. Take for instance when a user login, web page that user name and password and make SQL query to the database to check if a user has a valid name and password. With SQL Injection, it is possible for us to send crafted usernames and/or password fields that will change the SQL query and thus grant us something else.
Page URL	http://localhost/yonghao/yonghao/login.php
Instance / Example	Login SQL Injection
Methods	The attacker can use this command (' or '1'='1) to input for the username and password without using normal credentials. Now the application will access data for any user if their password is 1 or if 1=1. In the login page, the attacker will insert (' or '1'='1) as the username and the password in the textbox.  localhost/yonghao/yonghao/login.php Login here ' or '1'='1 Login Not a member? Sign up now Forgot password?
Render	After logging in, the attacker will automatically become the admin account. This is because it will make the system always return true. And since the condition 1=1 is always true, then it will give the attacker access to the first user within the database, which is the admin account.

	 localhost/yonghao/yonghao/index.php YouTube YouTube Music WeChat for Web WhatsApp Google Translate Classes TAR UC - Student In... 收件箱 (2) - yongha... PBe My Drive - Goo Welcome: admin HOME SEARCH UPLOAD PICTURE MY PICTURE LOGOUT	
Impacts	A successful SQL injection attack can result in confidential data being deleted, lost or stolen. The websites will also be defaced. The unauthorized access to systems or accounts and, ultimately compromise of individual machines or entire networks.	
Security Requirements	**Security Requirement**	**Justifications**
	1. Sensitive Information Protection	Ensure the system protects sensitive information such as username and password so that they won't be leaked or break easily, this can be done by encrypting or hashing.
	2. Product credential guessing	The attacker might use the information from the system to hack the system. So, the information such as the database type can be easy to exploit by the attackers. Hence, the system feedback should be hidden by only showing 'invalid' messages.

Vulnerability 3

Attack	**Cross-site Scripting (XSS)**
Description	Cross-site scripting (XSS) is a vulnerability that permits an attacker to inject code (typically HTML or JavaScript) into contents of a website not under the attacker's control. When a victim views such a page, the injected code executes in the victim's browser. Thus, the attacker has bypassed the browser's same origin policy and can steal the victim's private information associated with the website in question.
Page URL	http://localhost/yonghao/yonghao/search.php

Instance / Example	Reflected XSS Vulnerability
Methods	The attacker can use an image source tag to inject the string and perform the XSS attack. `<img%20src=x%20onMouseOver=alert(document.cookie)>` First, the attacker just needs to insert the malicious script into the textbox.  Welcome: a HOME ARCHIVES SEARCH UPLOAD PICTU You can search for others who usi User ID <img src=x onMouseOver submit no result found with user id: After that, when the malicious script has submitted, the image will be shown below the textbox.  Welcome: a HOME SEARCH UPLOAD PICTURE You can search for others wl User ID submit no result found with user id:
Render	Then, when the users move the mouse point to the image there, it will pop out the session ID that the attackers can use this session ID to hijack the client's current session. During the session hijacking, a malicious hacker places himself in between your computer and the website's server, while you are engaged in an active session.

	 localhost/yonghao/yonghao/search.php localhost says PHPSESSID=lcdq7cn02cbh8kqss8s0pomkam You can search for others who using this web application! User ID submit no result found with user id:	
Impacts	If an attacker can control a script that is executed in the victim's browser, then they can typically fully compromise that user. Beside that, the attacker also can perform any action within the application that the user can perform. The attackers also can initiate interactions with other application users, including malicious attacks, that will appear to originate from the initial victim user.	
Security Requirements	**Security Requirement**	**Justifications**
	1. Limit Execution	The system should not allow users to use html attributes to execute malicious code.For example, the system shall have the capability to remove html tags such as img <tag> before storing the input into the database.
	2. Security Data Backup/Restore	An attacker might modify the database content after a successful attack. Therefore, the system admin can restore the database by using the previous backup version.

References

Session Hijacking Takes Control of Your Accounts. Here's How. (2020). Retrieved 21 July 2020, from https://heimdalsecurity.com/blog/session-hijacking/

Cross Site Scripting (XSS) Software Attack | OWASP Foundation. (2020). Retrieved 21 July 2020, from https://owasp.org/www-community/attacks/xss/

SQL injection | OWASP Bricks Login page #1. (2020). Retrieved 21 July 2020, from https://sechow.com/bricks/docs/login-1.html

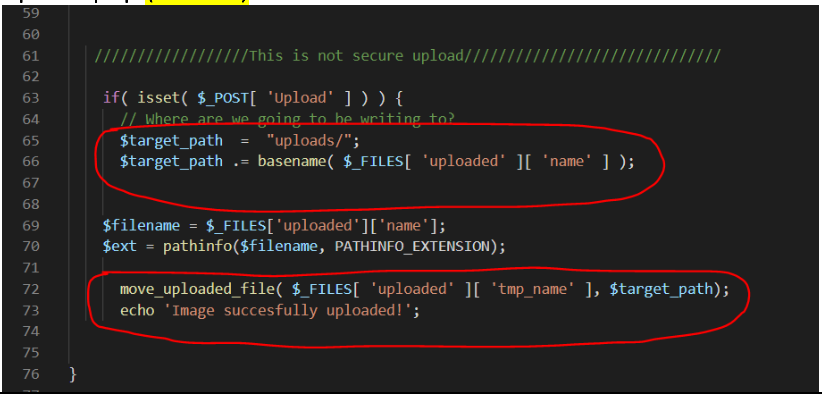
What is SQL Injection | SQLI Attack Example & Prevention Methods | Imperva. (2020). Retrieved 21 July 2020, from https://www.imperva.com/learn/application-security/sql-injection-sqli/#:~:text=The%20impact%20SQL%20injection%20can,highly%20detrimental%20to%20a%20business.

<u>**Part 2**</u>

Name : <u>**Lee Yong Hao**</u> **Tutorial Group:** <u>**RIS3 G4**</u>

<u>**Vulnerability 1**</u>

Vulnerability	Cross-site Scripting (XSS) - File Upload Vulnerability
Description of the vulnerability	This vulnerability is that the attacker is able to upload a malicious file to the web application. This is because it doesn't check for the file extension whether it is a valid file or not, so the attacker can simply upload any malicious file that the attackers want. Once the file is uploaded, the attacker can execute the file to get the sensitive information from the database such as the username and password.
Category *(Authentication, Authorization, etc)*	Confidentiality
Weakness (Refer to cwe_latest.pdf)	<u>**CWE-434: Unrestricted Upload of File with Dangerous Type**</u> This weakness occurs when an application does not validate or improperly validates files types before uploading files to the system. So, when the attacker uploads the files of dangerous types that can be automatically processed.
Source File	upload.php

Code Snippets (Line Number)	upload.php (Line 72)
	
	`59`
	`60`
	`61   ////////////////This is not secure upload/////////////////////////////`
	`62`
	`63     if( isset( $_POST[ 'Upload' ] ) ) {`
	`64       // Where are we going to be writing to?`
	`65       $target_path  =  "uploads/";`
	`66       $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );`
	`67`
	`68`
	`69     $filename = $_FILES['uploaded']['name'];`
	`70     $ext = pathinfo($filename, PATHINFO_EXTENSION);`
	`71`
	`72       move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path);`
	`73       echo 'Image succesfully uploaded!';`
	`74`
	`75`
	`76   }`

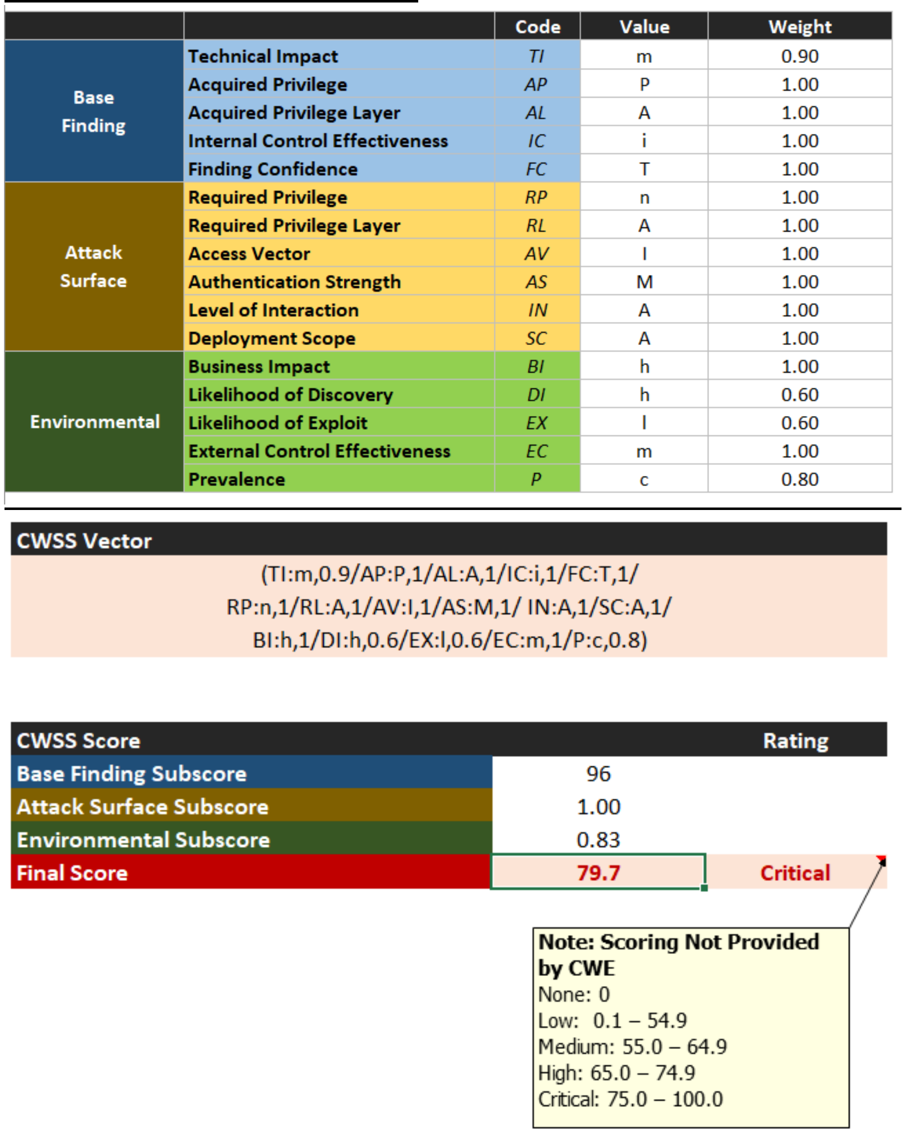
Description of the code	At the line 65, the **$target_path = "uploads/"** is where the file is going to be placed. So, I created a new directory in the directory where *upload.php* resides, called "uploads". Once the system receives a file from the user, the file will be automatically uploaded to the **"&target_path"** which is the file uploads directory.

	At line 66, which means the original filename is already added to our target path, which is **$target_path .= basename($_FILES['uploaded']['name']).** At line 72, this **move_uploaded_file** function will move the uploaded file with a particular name to the **"target_path"**, which is the uploads directory, but the line of code did not check the extension of the file whether it is a valid file or not. So, the attacker is able to upload any file extension without any restriction by the system. At the line 73, if the upload is successful, then you will see the text "Image successfully uploaded". This is because ***move_uploaded_file*** returns ***true*** if the file was uploaded. - The **$_FILES** array is where PHP stores all the information about files. There are two elements of this array which is **$_FILES['uploaded']['name']** and **$_FILES['uploaded']['tmp_name']**. - **uploaded** - *uploaded* is the reference we assigned in our HTML form. We will need this to tell the $_FILES array which file we want to execute it. - **$_FILES['uploaded']['name']** - *name* contains the original path of the user uploaded file. - the **$_FILES['uploaded']['tmp_name']** - *tmp_name* is about to contain the path to the temporary file that resides on the server. The file should exist on the server in a temporary directory with a temporary name.
Risk / Likelihood of Exploit *(Low/Medium/High)*	Medium

Risk Score Analysis

Common Weakness Scoring System (CWSS)

Result (Calculation Screenshot)



		Code	Value	Weight
Base Finding	Technical Impact	TI	m	0.90
	Acquired Privilege	AP	P	1.00
	Acquired Privilege Layer	AL	A	1.00
	Internal Control Effectiveness	IC	i	1.00
	Finding Confidence	FC	T	1.00
Attack Surface	Required Privilege	RP	n	1.00
	Required Privilege Layer	RL	A	1.00
	Access Vector	AV	I	1.00
	Authentication Strength	AS	M	1.00
	Level of Interaction	IN	A	1.00
	Deployment Scope	SC	A	1.00
Environmental	Business Impact	BI	h	1.00
	Likelihood of Discovery	DI	h	0.60
	Likelihood of Exploit	EX	l	0.60
	External Control Effectiveness	EC	m	1.00
	Prevalence	P	c	0.80

CWSS Vector

(TI:m,0.9/AP:P,1/AL:A,1/IC:i,1/FC:T,1/
RP:n,1/RL:A,1/AV:I,1/AS:M,1/ IN:A,1/SC:A,1/
BI:h,1/DI:h,0.6/EX:l,0.6/EC:m,1/P:c,0.8)

CWSS Score		Rating
Base Finding Subscore	96	
Attack Surface Subscore	1.00	
Environmental Subscore	0.83	
Final Score	79.7	Critical

Note: Scoring Not Provided by CWE
None: 0
Low: 0.1 – 54.9
Medium: 55.0 – 64.9
High: 65.0 – 74.9
Critical: 75.0 – 100.0

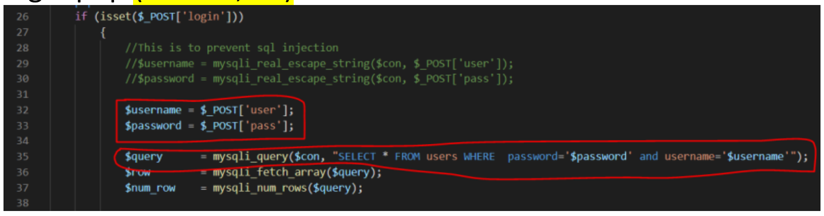
Overall Explanation

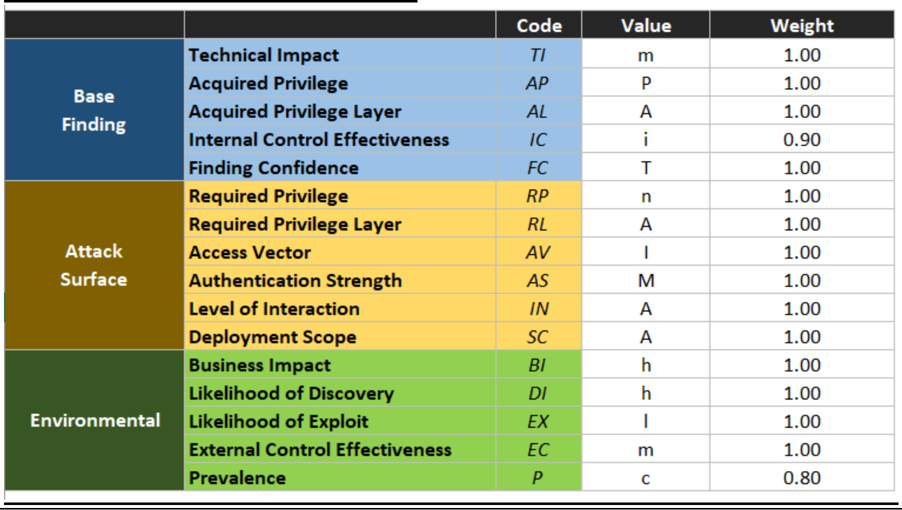
- The final score for the cwss is 79.7, it is rated as critical. This is because the attacker is able to upload the malicious script and then all the user credentials will be leaked from the database then taken by the attackers.
- Base Finding Subscore is 96 out of 100, which means it is very dangerous to the system. This is because the technical impact of this vulnerability is very serious, so anyone is able to upload any malicious file and run it. Since there are no restrictions that are in place to mitigate the attack, which means the attack can be carried out regardless of the system used by the attacker.
- Attack Surface Subscore is 1.00, which is a full score. That means the attack is easy to perform because there are no privileges required to perform it, so any user can simply upload any malicious file. The attacker also must have access to the Internet to reach the weakness. The weakness also does not require any authentication

	before the file uploads are being done. The last thing is there is no other person that is required to interact with in order to perform the attack. - Environmental Subscore is 0.83 out of 1.00. This is because if the attacker can upload any malicious file, the attacker stole all the user credentials from the database, this may cause any user to lose their account. The likelihood of exploitation is high because the attacker will successfully target this weakness using a reliable vulnerability that is easy to develop.

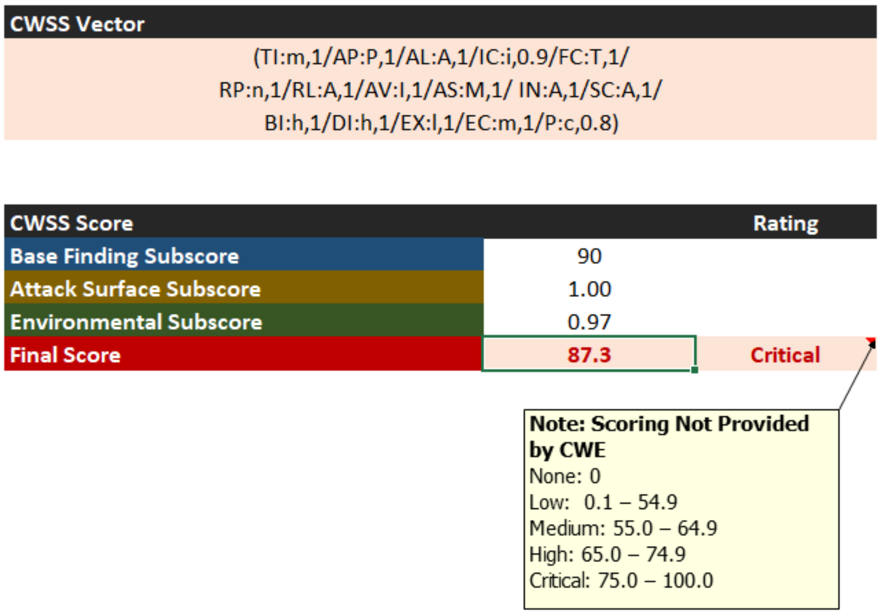
Vulnerability 2

Vulnerability	SQL Injection - Login SQL Injection
Description of the vulnerability	This Sql injection is an attack where the attacker is able to use a sql command to enter in the username and password field. When the attacker logs in, the attacker can log into the web application as an admin without any correct credentials. This is because the command will make the system always return true. And since the condition 1=1 is always true, then it will give the attacker access to the first user within the database, which is the admin account.
Category *(Authentication, Authorization, etc)*	Authentication
Weakness (Refer to cwe_latest.pdf)	**CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')** This is a weakness that are commonly appear in a lot of web application, especially in login pages that require user to input username and password to proceed to the next page, if the web application doesn't have sufficient removal or quoting of SQL syntax in user-controllable inputs, the attacker can input sql command as the input and the command will be executed and this can be use to bypass any security check.
Source File	login.php

Code Snippets (Line Number)	login.php (Line 32, 33)
	
	`26     if (isset($_POST['login']))`
	`27         {`
	`28             //This is to prevent sql injection`
	`29             //$username = mysqli_real_escape_string($con, $_POST['user']);`
	`30             //$password = mysqli_real_escape_string($con, $_POST['pass']);`
	`31`
	`32             $username = $_POST['user'];`
	`33             $password = $_POST['pass'];`
	`34`
	`35             $query      = mysqli_query($con, "SELECT * FROM users WHERE  password='$password' and username='$username'");`
	`36             $row        = mysqli_fetch_array($query);`
	`37             $num_row    = mysqli_num_rows($query);`
	`38`

Description of the code	Based on the code above line 32 and 33, there is no sanitization implemented in the system to sanitize out any possible input such as sql injection. The code didn't check for any input sanitization when a user input in the login form. So, the attacker can use (' or 1='1) sql command and insert into the username and the password section to login into the web application. After that, the system will store the sql command in the $username and $password variable, which then the system will run the sql command in line 35. Beside that, a normal user would be required to enter the correct combination of username and password, "SELECT * FROM users WHERE password='$password' and username='$username'"
Risk / Likelihood of Exploit *(Low/Medium/High)*	High
Risk Score Analysis **Common Weakness Scoring System (CWSS)**	**<u>Result (Calculation Screenshot)</u>** 

		Code	Value	Weight
Base Finding	Technical Impact	*TI*	m	1.00
	Acquired Privilege	*AP*	P	1.00
	Acquired Privilege Layer	*AL*	A	1.00
	Internal Control Effectiveness	*IC*	i	0.90
	Finding Confidence	*FC*	T	1.00
Attack Surface	Required Privilege	*RP*	n	1.00
	Required Privilege Layer	*RL*	A	1.00
	Access Vector	*AV*	l	1.00
	Authentication Strength	*AS*	M	1.00
	Level of Interaction	*IN*	A	1.00
	Deployment Scope	*SC*	A	1.00
Environmental	Business Impact	*BI*	h	1.00
	Likelihood of Discovery	*DI*	h	1.00
	Likelihood of Exploit	*EX*	l	1.00
	External Control Effectiveness	*EC*	m	1.00
	Prevalence	*P*	c	0.80



CWSS Vector
(TI:m,1/AP:P,1/AL:A,1/IC:i,0.9/FC:T,1/ RP:n,1/RL:A,1/AV:I,1/AS:M,1/ IN:A,1/SC:A,1/ BI:h,1/DI:h,1/EX:l,1/EC:m,1/P:c,0.8)

CWSS Score		Rating
Base Finding Subscore	90	
Attack Surface Subscore	1.00	
Environmental Subscore	0.97	
Final Score	87.3	Critical

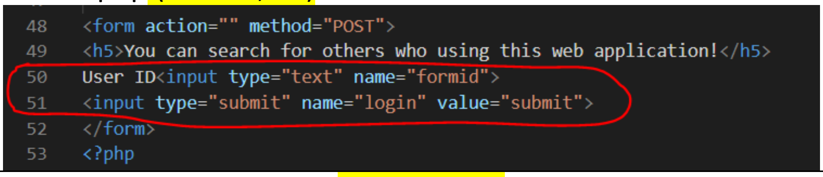
Note: Scoring Not Provided by CWE
None: 0
Low: 0.1 – 54.9
Medium: 55.0 – 64.9
High: 65.0 – 74.9
Critical: 75.0 – 100.0

Overall Explanation

- The final score for the SQL injection vulnerability is 87.3, it is rated as critical. This is because the attacker can log into the web application as an admin without any correct credentials.
- Base Finding Subscore is 90 out of 100. The technical impact is very serious because the attacker is able to easily gain access to an admin account. After the attacker gains access into the website, the attacker can do anything inside the website such as upload malicious files to steal the sensitive information.
- Attack Surface Subscore is a full score of 1.00 out of 1.00. This is because after the attacker gains access to the website as administrator, the attacker will take control over the system. This is because the attack is very easy to be conducted, with only one simple line of sql command. The attacker must have access to the Internet to reach the weakness. There are no special privileges required as anyone can perform the attack easily, and there is also no interaction with another user to perform this attack, which is why the attack surface scores a full score.
- Environmental Subscore is 0.97 out of 1.00. This is because this vulnerability will cause a very big impact to the web application, the attacker can use the admin account to login into the website and steal all the user information from the database.

<u>**Vulnerability 3**</u>

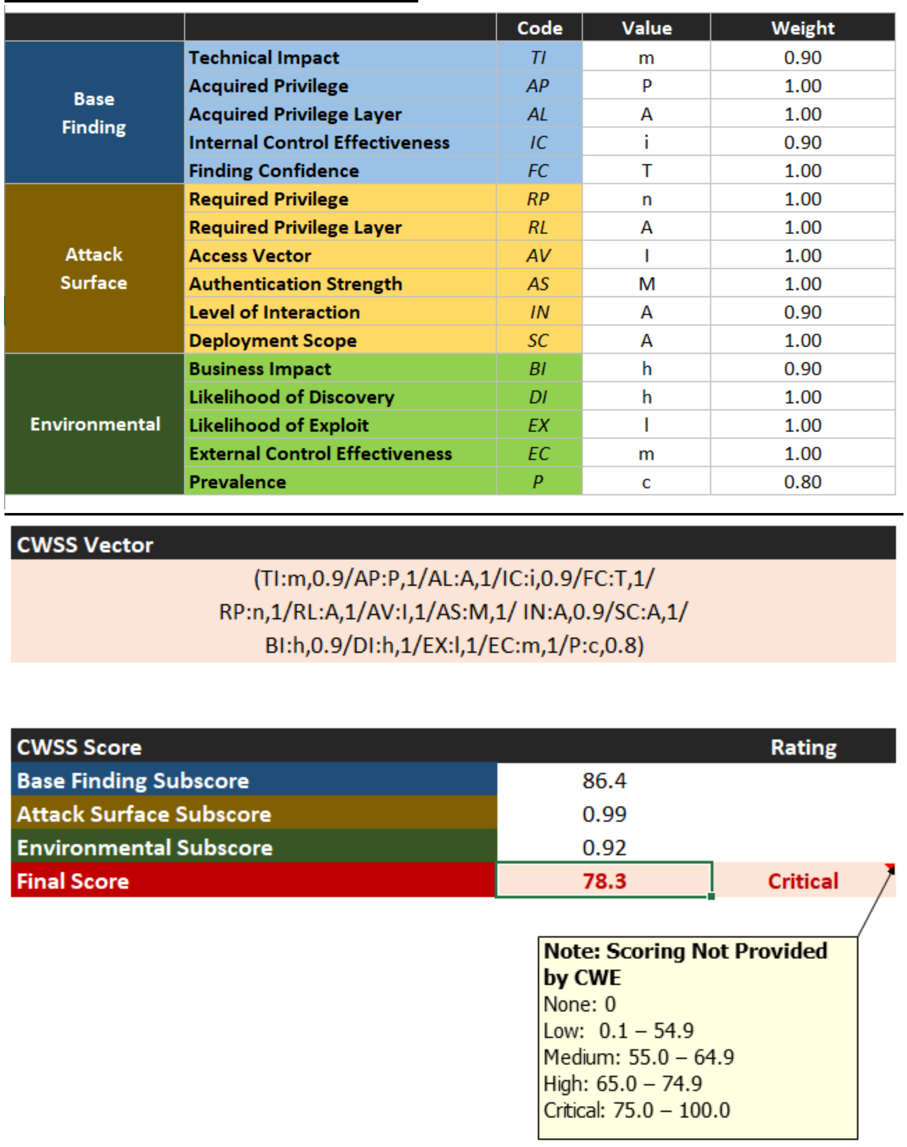
Vulnerability	Cross-site Scripting (XSS) - Reflected XSS Vulnerability
Description of the vulnerability	Cross site scripting is an attack where the attacker are able to input malicious script within inputs field in the web application, in the web application, the attacker are able to run scripts within the input fields that require user to input data, such as changing on how the web application looks with css command, with this vulnerability, the attacker can modify the url of the web application then send it to another user to lure them into clicking the malicious functionalities that are added by the attacker.
Category *(Authentication, Authorization, etc)*	Confidentiality
Weakness (Refer to cwe_latest.pdf)	<u>**CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')**</u> This weakness are commonly happen in page where it contains input fields for the user to enter data, if data input by the user are not sanitized properly, the attacker can exploit this by inputting scripts such as css command, causing some changes within the web application on how it looks, or even adding some extra functionality that may not intended for the web application.
Source File	search.php

Code Snippets (Line Number)	search.php (Line 50, 51)
	
	`48 <form action="" method="POST">`
	`49 <h5>You can search for others who using this web application!</h5>`
	`50 User ID<input type="text" name="formid">`
	`51 <input type="submit" name="login" value="submit">`
	`52 </form>`
	`53 <?php`

Description of the code	Based on the code above at line 50 and 51, the system does not sanitize the data from the user input before executing it, which means that the attacker can upload script commands into the input field and the command will be executed. In the web application, the attacker can use the command <img%20src=x%20onMouseOver=alert(document.cookie)> in order to create a small icon within the webpage, then send the url to another user, and once the user hovers his mouse icon to the image, the attacker will be able to get the victim session ID which then the attacker can use it to spoof as the victim.

Risk / Likelihood of Exploit *(Low/Medium/High)*	High
Risk Score Analysis **Common Weakness Scoring System (CWSS)**	**Result (Calculation Screenshot)**

Result (Calculation Screenshot)



		Code	Value	Weight
Base Finding	Technical Impact	*TI*	m	0.90
	Acquired Privilege	*AP*	P	1.00
	Acquired Privilege Layer	*AL*	A	1.00
	Internal Control Effectiveness	*IC*	i	0.90
	Finding Confidence	*FC*	T	1.00
Attack Surface	Required Privilege	*RP*	n	1.00
	Required Privilege Layer	*RL*	A	1.00
	Access Vector	*AV*	I	1.00
	Authentication Strength	*AS*	M	1.00
	Level of Interaction	*IN*	A	0.90
	Deployment Scope	*SC*	A	1.00
Environmental	Business Impact	*BI*	h	0.90
	Likelihood of Discovery	*DI*	h	1.00
	Likelihood of Exploit	*EX*	l	1.00
	External Control Effectiveness	*EC*	m	1.00
	Prevalence	*P*	c	0.80

CWSS Vector

(TI:m,0.9/AP:P,1/AL:A,1/IC:i,0.9/FC:T,1/
RP:n,1/RL:A,1/AV:I,1/AS:M,1/ IN:A,0.9/SC:A,1/
BI:h,0.9/DI:h,1/EX:l,1/EC:m,1/P:c,0.8)

CWSS Score		Rating
Base Finding Subscore	86.4	
Attack Surface Subscore	0.99	
Environmental Subscore	0.92	
Final Score	78.3	Critical

Note: Scoring Not Provided by CWE
None: 0
Low: 0.1 – 54.9
Medium: 55.0 – 64.9
High: 65.0 – 74.9
Critical: 75.0 – 100.0

Overall Explanation

- The final score for the cwss is 79.5, it is rated as critical. This is because the attacker is able to input malicious script within the input field in the web application and let the user click for it. After that, all the user credentials will be leaked from the database then taken by the attackers.
- Base Finding Subscore is 86.4 out of 100, which means it is very dangerous to the system. This is because the technical impact of this vulnerability is very serious, so anyone is able to upload any malicious script within the input field and let the user run it. Since there are no restrictions that are in place to mitigate the attack,

	which means the attack can be carried out regardless of the system used by the attacker. - Attack Surface Subscore is 0.99.. That means the attack is easy to perform because there are no privileges required to perform it, so any user can simply upload any malicious script. The attacker also must have access to the Internet to reach the weakness. The weakness also does not require any authentication before the file uploads are being done. The last thing is there is some interaction with the user which lets the user click the image that was created by the attacker. - Environmental Subscore is 0.92 out of 1.00. This is because if the attacker can upload any malicious script within the input field, the attacker stole all the user credentials from the database, this may cause any user to lose their account. The likelihood of exploitation is high because the attacker will successfully target this weakness using a reliable vulnerability that is easy to develop.

References

Unrestricted File Upload - AppSec io. (2020). Retrieved 9 August 2020, from https://applicationsecurity.io/appsec-findings-database/unrestricted-file-upload/#:~:text=Unrestricted%20file%20upload%20is%20a,system%20through%20a%20web%20shell.

WordPress Security - File Upload Vulnerabilities. (2020). Retrieved 9 August 2020, from https://www.wordfence.com/learn/how-to-prevent-file-upload-vulnerabilities/#:~:text=A%20local%20file%20upload%20vulnerability,Internet%20and%20store%20it%20locally.

PHP Tutorial - File Upload. (2020). Retrieved 9 August 2020, from http://www.tizag.com/phpT/fileupload.php

Unrestricted Upload of File with Dangerous Type Vulnerability | CWE-434 Weakness | Exploitation and Remediation. (2020). Retrieved 9 August 2020, from https://www.immuniweb.com/vulnerability/unrestricted-upload-of-file-with-dangerous-type.html#:~:text=%5BCWE%2D434%5D-,Unrestricted%20Upload%20of%20File%20with%20Dangerous%20Type%20%5BCWE%2D434%5D,file%20types%20when%20uploading%20files.

CWE-434 - Unrestricted Upload of File with Dangerous Type. (2020). Retrieved 9 August 2020, from https://www.cybersecurity-help.cz/vdb/cwe/434/

Part 3

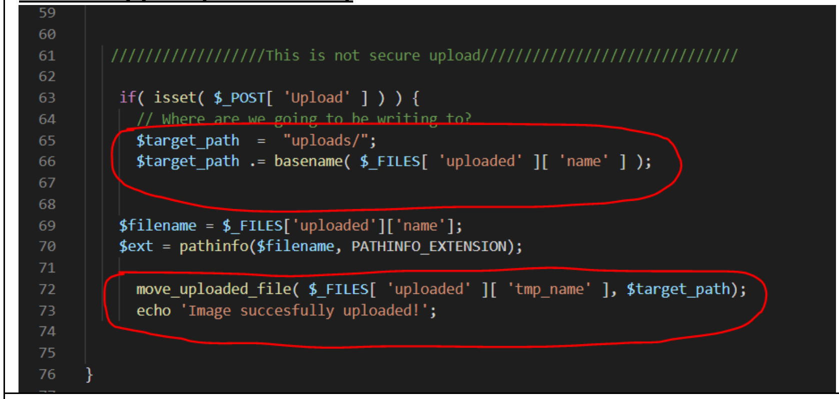
Name : Lee Yong Hao Tutorial Group: RIS3 G4

VULNERABILITY REMEDIATION REPORT

Vulnerability 1- Cross-site Scripting (XSS) - File Upload Vulnerability

BEFORE (Vulnerable Version)

Code Snippets (Screenshot)

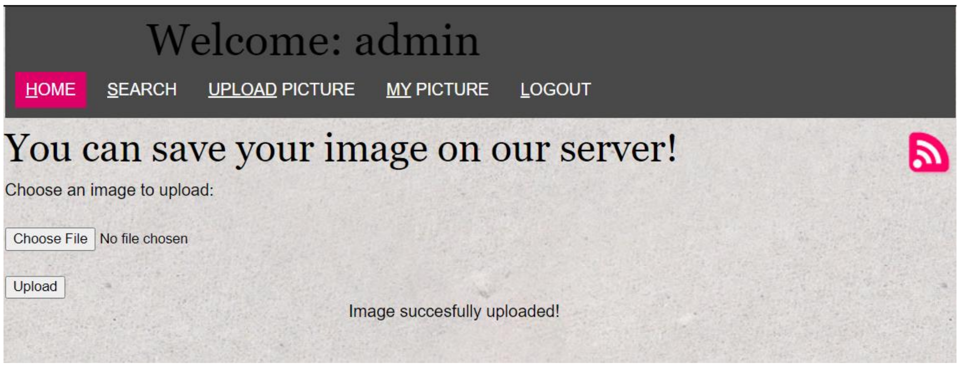


```
59
60
61    /////////////////This is not secure upload////////////////////////////
62
63    if( isset( $_POST[ 'Upload' ] ) ) {
64      // Where are we going to be writing to?
65      $target_path  =  "uploads/";
66      $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );
67
68
69    $filename = $_FILES['uploaded']['name'];
70    $ext = pathinfo($filename, PATHINFO_EXTENSION);
71
72      move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path);
73      echo 'Image succesfully uploaded!';
74
75
76  }
```

Description of code

In the vulnerable version, the code didn't limit the user from uploading any other kind of files. The code which is line 72 did not check the extension of the file whether it is a valid file or not. So, the attacker is able to upload any file extension without any restriction by the system.

Result (User Interface - Screenshot)



You can save your image on our server!

Choose an image to upload:

No file chosen

Image succesfully uploaded!

Description of result

After the attacker uploads the malicious file which is hackscript.html to the file directory, then the system will show a text which is "Image successfully uploaded". After the attacker runs the malicious file, it will pop out the session ID that the attackers can use this session ID to hijack the client's current session.

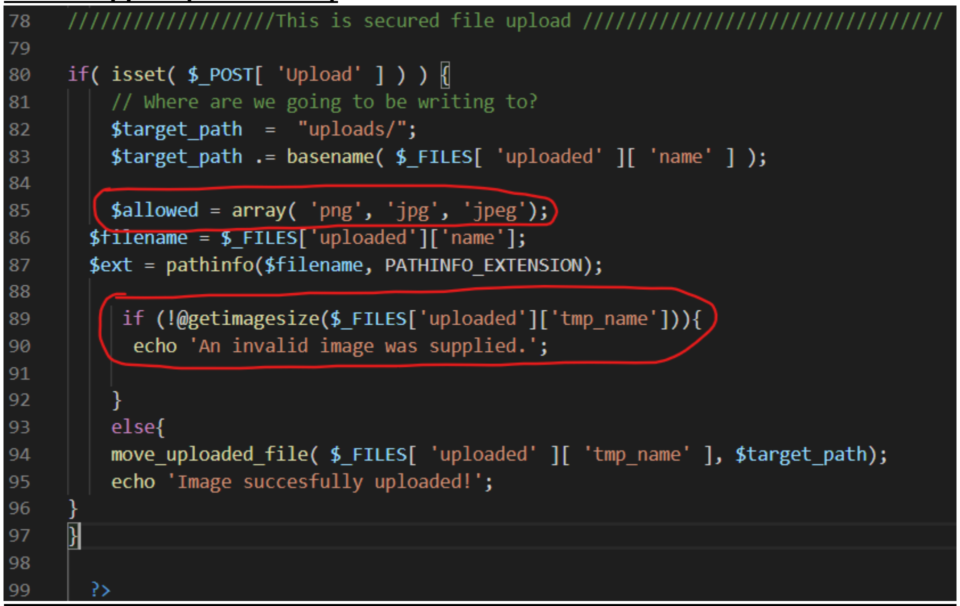
AFTER (Mitigated Version)

Implementation of Solution

Prevention Strategies used

In order to prevent this vulnerability being exploited by the attacker, there is a prevention strategy that can be used as an uploaded file checking function. Before the attacker uploads the image file, it will automatically check the extension of the file whether it is a valid file or not. Then it also checks whether the content of the inside file got malicious code or not. After checking if the extension of the file is valid, the file only can be uploaded to the file directory. For example, if the code only allows the user to upload the image files, but the attacker tries to upload an image file within a malicious file, then the upload system will deny the attacker to upload the script file because it is not a valid file that should be uploaded to the system.

Code Snippets (Screenshot)



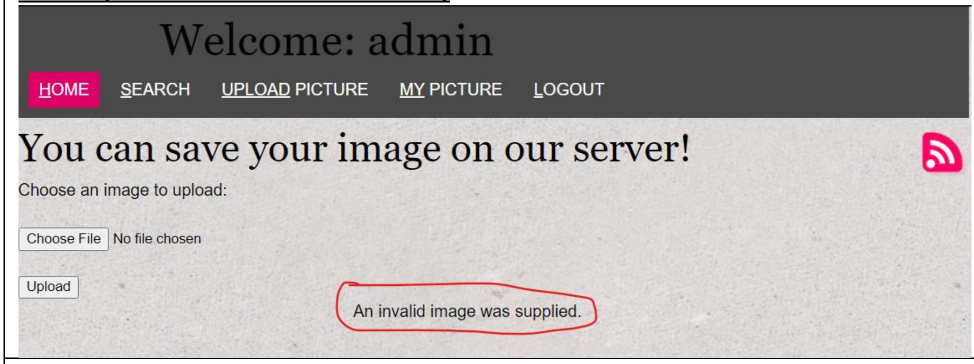
```
78  //////////////////This is secured file upload ////////////////////////////////
79
80  if( isset( $_POST[ 'Upload' ] ) ) {
81      // Where are we going to be writing to?
82      $target_path  =  "uploads/";
83      $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );
84
85      $allowed = array( 'png', 'jpg', 'jpeg');
86    $filename = $_FILES['uploaded']['name'];
87    $ext = pathinfo($filename, PATHINFO_EXTENSION);
88
89       if (!@getimagesize($_FILES['uploaded']['tmp_name'])){
90        echo 'An invalid image was supplied.';
91
92      }
93      else{
94      move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path);
95      echo 'Image succesfully uploaded!';
96  }
97  }
98
99    ?>
```

Description of code

In the mitigated version, in line 85, there was an allowed list of file extensions, which is ‘png’, ‘jpg’, and ‘jpeg’, which is used to only allow the user to upload the image file only. If any other extension types of file are uploaded, it will deny the user to upload it.

In line 89, the code getimagesize() is used to read the header information of the image and will fail on an invalid image. This is another method to verify the content you’re expecting from the user. After checking if the extension of the file is valid and the content of the file is not malicious, the file only can be uploaded to the file directory. For example, if the code only allows the user to upload the image files, but the attacker tries to upload an image within malicious code, then the upload system will automatically deny the attacker to upload the script file and the upload will fail because the image has the malicious code inside.

Result (User Interface - Screenshot)



You can save your image on our server!

Choose an image to upload:

An invalid image was supplied.

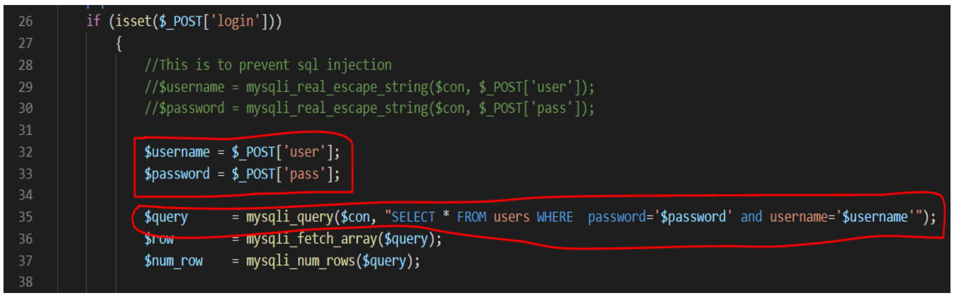
Description of result

After implementing the mitigated code, the attacker is not able to upload hackscript.html to the system anymore, because it is not a valid file that should be uploaded to the system. Then, the file will be denied by the code, and the system will show a text which is "An invalid image was supplied.". This is because the image has the malicious code inside.

Vulnerability 2 - SQL Injection - Login SQL Injection

BEFORE (Vulnerable Version)

Code Snippets (Screenshot)

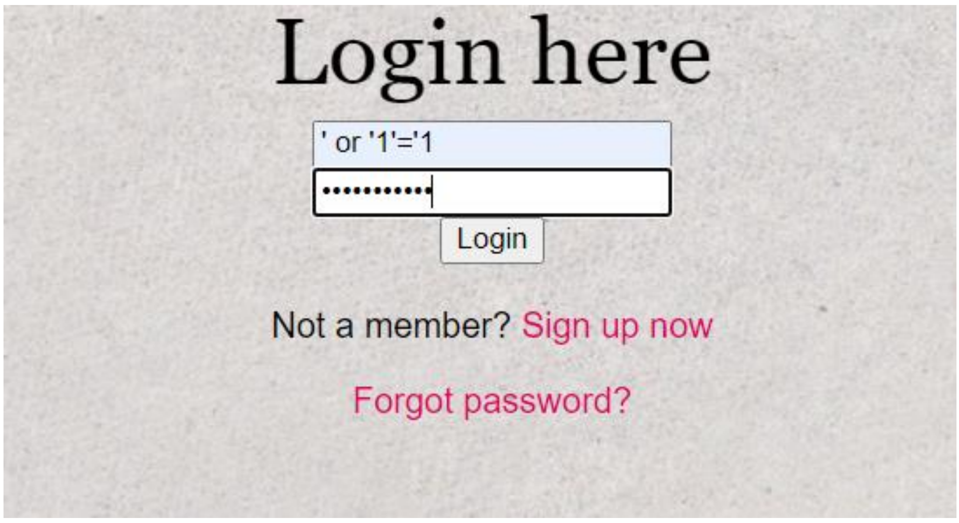
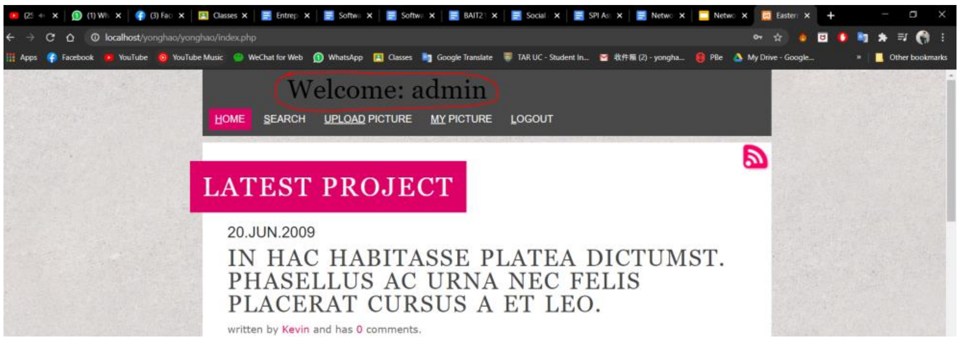


```
26  if (isset($_POST['login']))
27      {
28          //This is to prevent sql injection
29          //$username = mysqli_real_escape_string($con, $_POST['user']);
30          //$password = mysqli_real_escape_string($con, $_POST['pass']);
31
32          $username = $_POST['user'];
33          $password = $_POST['pass'];
34
35          $query      = mysqli_query($con, "SELECT * FROM users WHERE  password='$password' and username='$username'");
36          $row        = mysqli_fetch_array($query);
37          $num_row    = mysqli_num_rows($query);
38
```

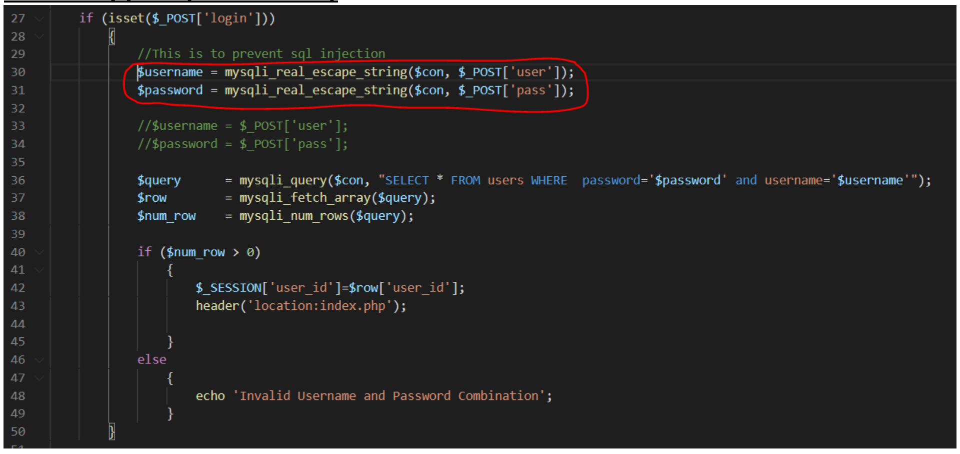
Description of code

In the vulnerable version, based on the code above line 32 and 33, there is no sanitization implemented in the system to sanitize out any possible input such as sql injection. The code didn't check for any input sanitization when a user input in the login form. So, the attacker can use (' or 1='1) sql command and insert into the username and the password section to login into the web application.

After that, the system will store the sql command in the $username and $password variable, which then the system will run the sql command in line 35. Beside that, a normal user would be required to enter the correct combination of username and password.

	Result (User Interface - Screenshot) Before Login using sql injection  Login here ' or '1'='1 Login Not a member? Sign up now Forgot password? After Login using sql injection  localhost/yonghao/yonghao/index.php Welcome: admin HOME SEARCH UPLOAD PICTURE MY PICTURE LOGOUT LATEST PROJECT 20.JUN.2009 IN HAC HABITASSE PLATEA DICTUMST. PHASELLUS AC URNA NEC FELIS PLACERAT CURSUS A ET LEO. written by Kevin and has 0 comments.
	Description of result The attacker is able to login as an admin even without the valid password, this is because the attacker used (' or 1='1) sql command to bypass the login function and insert into the username and the password section to login into the web application. Since the condition 1=1 is always true, then it will give the attacker access to the first user within the database, which is the admin account.
AFTER (Mitigated Version) Implementation of Solution	**Prevention Strategies used** The prevention strategy that is used is mysql_real_escape_string() to ensure that any dangerous characters such as ' are not passed to a SQL query in data. It also sanitizes everything by filtering user data by context. For example, username should be filtered to allow only the characters allowed in an username, password should be filtered to allow only the digits allowed in a password, and so on.

Code Snippets (Screenshot)

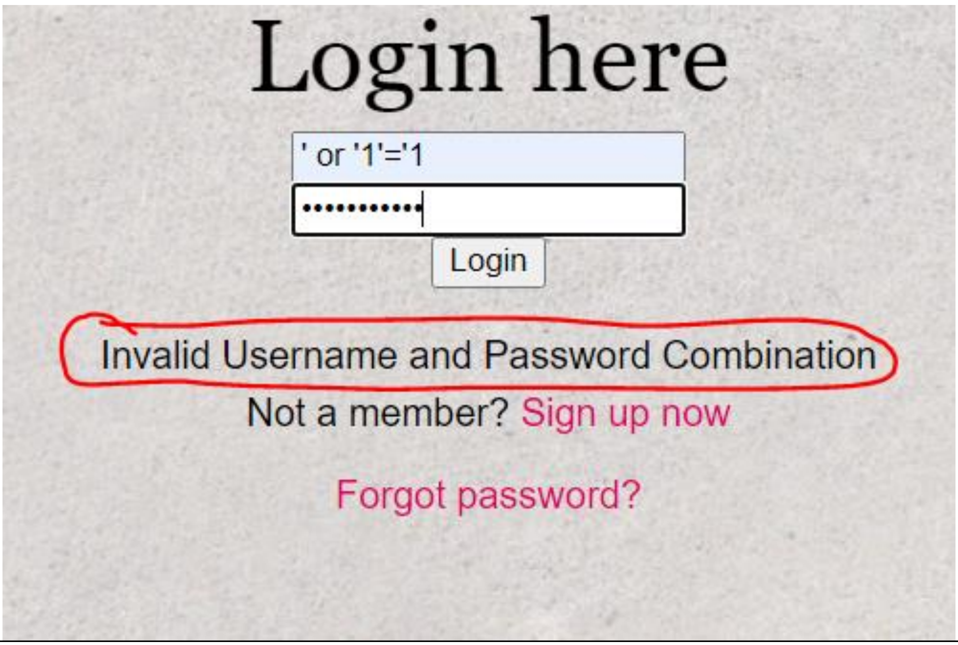


```
27      if (isset($_POST['login']))
28          {
29              //This is to prevent sql injection
30              $username = mysqli_real_escape_string($con, $_POST['user']);
31              $password = mysqli_real_escape_string($con, $_POST['pass']);
32
33              //$username = $_POST['user'];
34              //$password = $_POST['pass'];
35
36              $query      = mysqli_query($con, "SELECT * FROM users WHERE  password='$password' and username='$username'");
37              $row        = mysqli_fetch_array($query);
38              $num_row    = mysqli_num_rows($query);
39
40              if ($num_row > 0)
41                  {
42                      $_SESSION['user_id']=$row['user_id'];
43                      header('location:index.php');
44
45                  }
46              else
47                  {
48                      echo 'Invalid Username and Password Combination';
49                  }
50          }
```

Description of code

In the mitigated version, in line 30 and 31, these two lines of code will sanitize the user input before storing it in the $username and $password variable. This is because mysql_real_escape_string() will sanitize everything by filtering user data by context to ensure that any dangerous characters such as ' are not passed to a SQL query in data.

Result (User Interface - Screenshot)



Login here

' or '1'='1

•••••••••••

Login

Invalid Username and Password Combination

Not a member? Sign up now

Forgot password?

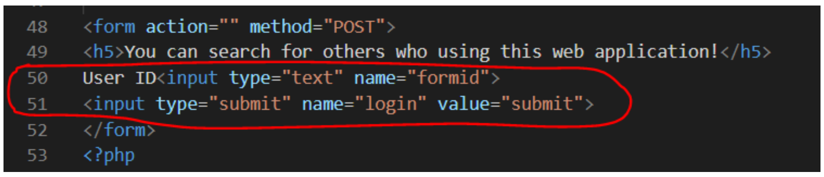
Description of result

After implementing the mitigated code, the attacker is not able to login using SQL injection. This is because there is a data sanitization function working before storing the user input into a variable then executing it. So, when the attacker uses the (' or 1='1) sql command to login, the system will show a text which is "Invalid Username and Password Combination".

Vulnerability 3 - Cross-site Scripting (XSS) - Reflected XSS Vulnerability

BEFORE (Vulnerable Version)

Code Snippets (Screenshot)

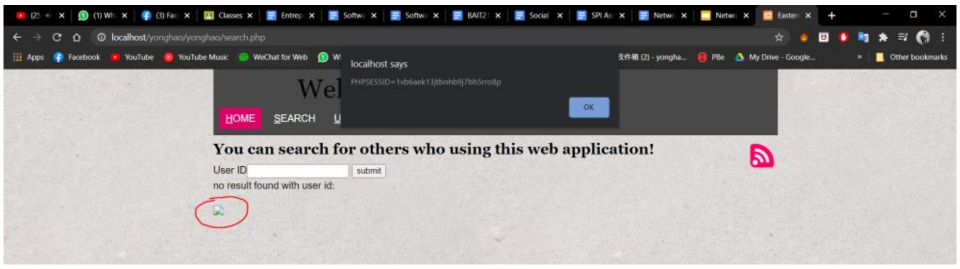


```
48    <form action="" method="POST">
49    <h5>You can search for others who using this web application!</h5>
50    User ID<input type="text" name="formid">
51    <input type="submit" name="login" value="submit">
52    </form>
53    <?php
```

Description of code

In the vulnerable version, in line 50 and 51, the system does not sanitize the data from the user input before executing it, which means that the attacker can upload script commands into the input field and the command will be executed.

Result (User Interface - Screenshot)



localhost/yonghao/yonghao/search.php

localhost says

PHPSESSID=1vb6aek13jtbnhb9j7bh5rro8p

OK

You can search for others who using this web application!

User ID submit

no result found with user id:

Description of result

In the web application, the attacker can use the command <img%20src=x%20onMouseOver=alert(document.cookie)> in order to create a small icon within the webpage, then send the url to another user, and once the user hovers his mouse icon to the image, the attacker will be able to get the victim session ID which then the attacker can use it to spoof as the victim.

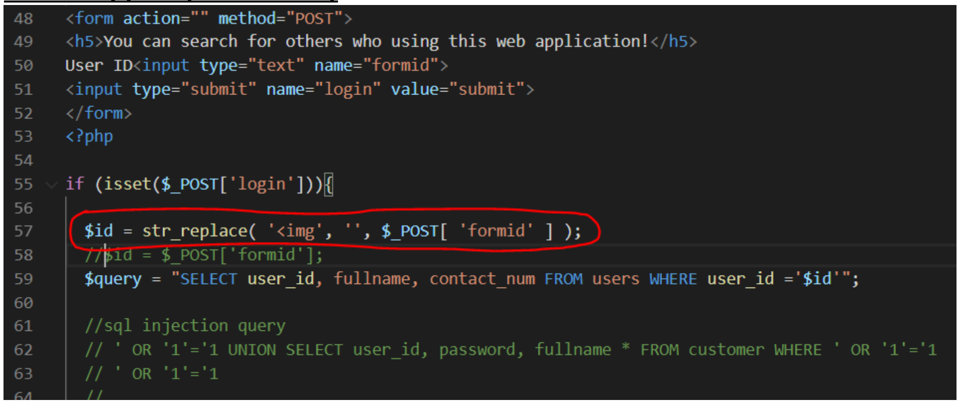
AFTER (Mitigated Version)

Implementation of Solution

Prevention Strategies used

The prevention strategy that is used is to sanitize the user input before storing the input in a variable and execute it. So, any command that is inputted by the attacker will be removed and it will be treated as a normal input.

Code Snippets (Screenshot)

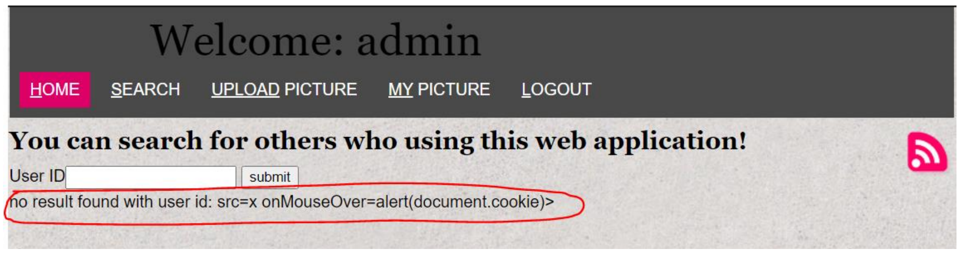


```
48    <form action="" method="POST">
49    <h5>You can search for others who using this web application!</h5>
50    User ID<input type="text" name="formid">
51    <input type="submit" name="login" value="submit">
52    </form>
53    <?php
54
55    if (isset($_POST['login'])){
56
57      $id = str_replace( '<img', '', $_POST[ 'formid' ] );
58      //$id = $_POST['formid'];
59      $query = "SELECT user_id, fullname, contact_num FROM users WHERE user_id ='$id'";
60
61      //sql injection query
62      // ' OR '1'='1 UNION SELECT user_id, password, fullname * FROM customer WHERE ' OR '1'='1
63      // ' OR '1'='1
64      //
```

Description of code

In the mitigated version, in line 57, the variable $id is contains a data sanitization function, where if the attacker tries to enter <img> command into the input field, the command will be sanitize and replaced with an empty space, so that the image command will be invalid and unable to be used to modify the interface of the web application.

Result (User Interface - Screenshot)



You can search for others who using this web application!

User ID

no result found with user id: src=x onMouseOver=alert(document.cookie)>

Description of result

After implementing the mitigated code, the attacker is not able to use the command <img%20src=x%20onMouseOver=alert(document.cookie)> into the input field. So, when the attacker use the command <img%20src=x%20onMouseOver=alert(document.cookie)>, the system will sanitize it as an invalid input and then show a text which is "no result found with user id : src=x on MouseOver=alert(document.cookie)".

References

DVWA 1.9+: XSS Reflected. Medium. (2020). Retrieved 26 August 2020, from https://medium.com/hacker-toolbelt/dvwa-1-9-xss-reflected-58047a2d0ac1.

(2020). Retrieved 26 August 2020, from https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications.

Unrestricted File Upload | OWASP. Owasp.org. (2020). Retrieved 26 August 2020, from https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload.

Attacks, H., & Rubens, P. (2020). *How to Prevent SQL Injection Attacks*. Esecurityplanet.com. Retrieved 27 August 2020, from https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks.html.

Assignment Rubric :

STUDENT NAME	TUTORIAL GROUP
Lee Yong Hao	RIS3 G4

NO	CRITERIA	TOTAL MARK	POOR	AVERAGE	GOOD	EXCELLENCE	MARK
ASSIGNMENT (PART 1): Vulnerability Assessment Report (20%) – CLO1(C4, PLO1)							
1.	**Description of Vulnerabilities**	**5**	**0 – 1 mark** Unclear and incomplete description of vulnerabilities.	**2 - 3 marks** Reasonably clear and concise description of vulnerabilities.	**4 marks** Clear and concise description of vulnerabilities.	**5 marks** Very clear, concise, correct and comprehensive description of vulnerabilities.	
2.	**Description of methods, renders, instances, impacts**	**5**	**0 – 1 mark** Unclear, incorrect and incomplete description of methods, renders, instances, impacts.	**2 – 3 marks** Reasonably clear and correct description of methods, renders, instances, impacts.	**4 marks** Clear, concise and correct description of methods, renders, instances, impacts.	**5 marks** Very clear, concise, correct and comprehensive description of methods, renders, instances, impacts.	
3.	**Appropriate use of Security Requirements**	**5**	**0 – 1 mark** Incorrect use of security requirements and principles.	**2 – 3 marks** Partially correct use of security requirements and principles.	**4 marks** Almost comprehensive and mostly correct use of security requirements and principles.	**5 marks** Perfectly comprehensive and correct use of security requirements and principles.	
4.	**Presentation** - Understanding - Justification	**5**	**0 – 1 mark** Poor oral and written communication skills.	**2 - 3 marks** Average oral and written communication skills.	**4 marks** Effective oral and written communication skills.	**5 marks** Highly effective oral and written communication skills.	

NO	CRITERIA	TOTAL MARK	POOR	AVERAGE	GOOD	EXCELLENCE	MARK
			Do not understand the content of the assignment at all. No justifications were provided.	Average understanding of the assignment contents. Some justifications were provided.	Good understanding of the assignment contents. Good justifications were provided.	Very good understanding of the assignment contents. Strong and convincing justification were provided.	
ASSIGNMENT (PART 2) : Source Code Analysis Report (40%) – CLO3(A3, PLO5)							
6.	**Description of Vulnerable codes**	**10**	**0 – 2 marks** Unclear, incorrect and incomplete description of vulnerable codes.	**3 – 5 marks** Reasonably clear and correct description of vulnerable codes.	**6 - 8 marks** Clear, concise and correct description of vulnerable codes.	**9 - 10 marks** Very clear, concise, correct and comprehensive description of vulnerable codes.	
7.	**Description of Weakness**	**10**	**0 – 2 marks** Unclear, incorrect and incomplete description of weakness.	**3 – 5 marks** Reasonably clear and correct description of weakness.	**6 - 8 marks** Clear, concise and correct description of weakness.	**9 - 10 marks** Very clear, concise, correct and comprehensive description of weakness.	
8.	**Description of Risk Score Analysis** **Digital Skills** -Able to use Common Weakness Scoring System (CWSS) to prioritize software weaknesses in a consistent manner.	**10**	**0 – 2 marks** Unclear, incorrect and incomplete description of Risk Score Analysis.	**3 – 5 marks** Reasonably clear and correct description of Risk Score Analysis.	**6 - 8 marks** Clear, concise and correct description of Risk Score Analysis.	**9 - 10 marks** Very clear, concise, correct and comprehensive description of Risk Score Analysis.	

NO	CRITERIA	TOTAL MARK	POOR	AVERAGE	GOOD	EXCELLENCE	MARK
	Numeracy Skills - able to understand on quantitative measurement of the unfixed weaknesses that are present within a software application.						
9.	**Presentation** **-** Understanding - Justification	**10**	**0 – 2 marks** Poor oral and written communication skills. Do not understand the content of the assignment at all. No justifications were provided.	**3 - 5 marks** Average oral and written communication skills. Average understanding of the assignment contents. Some justifications were provided.	**6 - 8 marks** Effective oral and written communication skills. Good understanding of the assignment contents. Good justifications were provided.	**9 - 10 marks** Highly effective oral and written communication skills. Very good understanding of the assignment contents. Strong and convincing justification were provided.	
ASSIGNMENT (PART 3) : Vulnerability Remediation Report (40%) – CLO2(C4, PLO2)							
10.	**Description of the Prevention Strategies developed**	**10**	**0 – 2 marks** Unclear, incorrect and incomplete description of the prevention strategies developed.	**3 – 5 marks** Reasonably clear and correct description of the prevention strategies developed.	**6 - 8 marks** Clear, concise and correct description of the prevention strategies developed.	**9 - 10 marks** Very clear, concise, correct and comprehensive description of the prevention strategies developed.	

NO	CRITERIA	TOTAL MARK	POOR	AVERAGE	GOOD	EXCELLENCE	MARK
11.	**Innovation and Initiative demonstrated in solution implementation**	**10**	**0 – 2 marks** Does not demonstrate any innovation and initiative in implementation of solution.	**3 – 5 marks** Partially demonstrate innovation and initiative in implementation of solution.	**6 - 8 marks** Demonstrates high level of innovation and initiative in implementation of solution.	**9 - 10 marks** Demonstrates very high level of innovation and initiative in implementation of solution.	
12.	**Design and Implementation of Solution**	**10**	**0 – 2 marks** Poor design and implementation, demonstrating lack of programming discipline.	**3 – 5 marks** Average design and implementation with use of constructs and logic.	**6 - 8 marks** Good design and implementation with correct use of constructs and logic.	**9 - 10 marks** Excellent design and implementation with elegant use of constructs and logic.	
13.	**Presentation** - Understanding - Justification	**10**	**0 – 2 marks** Poor oral and written communication skills. Do not understand the content of the assignment at all. No justifications were provided.	**3 - 5 marks** Average oral and written communication skills. Average understanding of the assignment contents. Some justifications were provided.	**6 - 8 marks** Effective oral and written communication skills. Good understanding of the assignment contents. Good justifications were provided.	**9 - 10 marks** Highly effective oral and written communication skills. Very good understanding of the assignment contents. Strong and convincing justification were provided.	
		100				**Total :**	

Overall Comments: