SECURITY ISSUES:

1. Insecure Communication on Protocol in Use
2. Degrade POST
3. Stored Cross Site Scripting
4. Host Header Injection
5. Directory Listing
6. Clickjacking
7. Httponly flag not set
8. Software Version Disclosure
9. Failure To Restrict URL Access
10. Cross Site Request Forgery
11. Accessing / Modifying Other User Information
12. Automated Form Submission
13. Malicious File Upload
14. Brute Force Attack
15. Security Headers Missing
16. Insufficient Events Logs
17. Insufficient Warnings And Errors Logs
18. **Insecure Communication on Protocol in Use :**

Issue:

The application allows users to connect to it over unencrypted connections. An attacker suitably positioned to view a legitimate user's network traffic could record and monitor their interactions with the application and obtain any information the user supplies. Furthermore, an attacker able to modify traffic could use the application as a platform for attacks against its users and third-party websites. Unencrypted connections have been exploited by ISPs and governments to track users, and to inject adverts and malicious JavaScript

Solution:

Applications should use transport-level encryption (SSL/TLS) to protect all communications passing between the client and the server. The Strict Transport-Security HTTP header should be used to ensure that clients refuse to access the server over an insecure connection

1. From server we can enable the SSL certificate
2. HTTP headers can be set for the https site inside the construct function.

header("Strict-Transport-Security: max-age=31536000");

1. **Degrade POST :**

Issue:

Downgrading a regular POST request to a GET request makes it easier for attackers to exploit other vulnerabilities that may exist in the application such as XSS, CSRF, Reflected File Download, Open Redirect, or Session Fixation.

Solution:

Validate the HTTP request method before processing the request.

public function testFunctn($value='')

{ if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {

//code here

}

}

public function testFunctn($value='')

{

if ($\_SERVER['REQUEST\_METHOD'] === GET) { //code here }

}

1. **Stored Cross Site Scripting**

Issue:

During the assessment, a file with malicious script can be uploaded and while updating the user details in the portal, the code gets executed as the file name is parsed and an alert is shown in alert box. An attacker can use this vulnerability to inject some malicious script into the vulnerable application. When a victim user encounters the script, it executes in the victim's browser. The XSS script can then perform any action that the victim is able to perform, and access all of the victim's sensitive data

Solution:

It is recommended to mitigate reflected cross site scripting attacks in the following ways

1. Input should be validated as strictly as possible on arrival, given the kind of content that it is expected to contain

2. User input should be HTML-encoded at any point where it is copied into application responses. All HTML met characters, including < > " ' and =, should be replaced with the corresponding HTML entities (&lt; &gt; etc.).

3. Integrate the strict code igniter form validation with exact conditions

$this->form\_validation->set\_rules('email', 'Email', 'valid\_email|is\_unique[student.email]',array('is\_unique' => 'This %s is already exist'));

$this->form\_validation->set\_rules('mobile', 'Mobile No.', required|is\_unique[student.phone]|numeric|exact\_length[10]',array('is\_unique' => 'This %s is already exist'));

$this->form\_validation->set\_rules('password', 'Password', 'trim|required|min\_length[5]');

$this->form\_validation->set\_rules('cnpassword', 'Password Confirmation', 'trim|required|matches[password]');

4.Add the htmlentities($this->input->post(‘name)); code while getting the values from view to controller.

5. Added the xss clean methods $this->security->xss\_clean($\_POST); , $this->security->xss\_clean($\_GET);

1. **Host Header Injection:**

ISSUES:

The host header specifies which website or web application should process an incoming HTTP request. The web server uses the value of this header to dispatch the request to the specified website or web application. Each web application hosted on the same IP address is commonly referred to as a virtual host.

Displaying version information of software information could allow an attacker to determine which vulnerabilities are present in the software, particularly if an outdated software version is in use with published vulnerabilities

Solution:

* 1. Add the below code in .htaccess

#It can allow 3rd level subdomain ({0,3}) name and in the length of 1 to 20 ({1,20}). We can change for you convenience.

RewriteEngine On

RewriteCond %{HTTP\_HOST} !^([a-zA-Z0-9-\_]{1,20}.){0,3}domain.com$

RewriteRule ^(.\*)$ https://domain.com/ [R=301,L]

1. **Directory Listing:**

Issue:

During security testing, it was identified that Directory Traversing is possible. It was possible to access files and directories outside the application root folder. It may be possible to access arbitrary files and directories stored on file system, including application source code, configuration and critical system files.

Malicious users may view any file on the web server (e.g. web.config or application source code). This can enable a malicious user to compromise the application or webserver or gain access to proprietary application source code.

Solution:

Develop code to ensure that only explicitly permitted files can be downloaded using the get File script. Create a white-list of files or file types that may be downloaded to the application based on business need. Deny all downloads of file types that are not explicitly permitted.

1. Add the below code in .htaccess

Options –Indexes

1. Add the below code in index.html and upload inside of all the directories

<!DOCTYPE html>

<html>

<head><title>403 Forbidden</title></head>

<body><p>Directory access is forbidden.</p>

</body>

</html>

1. **Clickjacking:**

Issue:

Clickjacking is a malicious technique of tricking a user into clicking on something different from what the user perceives, thus potentially revealing confidential information or allowing others to take control of their computer while clicking on seemingly innocuous objects, including web pages.

If a page fails to set an appropriate X-FrameOptions or ContentSecurity-Policy HTTP header, it might be possible for a page controlled by an attacker to load it within an iframe. This may enable a clickjacking attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defenses against cross-site request forgery, and may result in unauthorized actions.

Solution:

To effectively prevent framing attacks, the application should return a response header with the name XFrame-Options and the value DENY to prevent framing altogether, or the value SAMEORIGIN to allow framing only by pages on the same origin as the response itself. Note that the SAMEORIGIN header can be partially bypassed if the application itself can be made to frame untrusted websites.

1. <?php header("X-Frame-Options: DENY"); ?> add this code in all the controller inside the construct function
2. Header set X-Frame-Options DENY add the code in .htaccess file or in apache httd.conf file
3. **Httponly flag not set**

Issues:

During this assessment it was observed that cookies issued by the application does not have the httponly flag set.

httpOnly ensures that scripting languages (ie. javascript) won't be able to get the cookie value (such as through document.cookie). The only way to get it is through http request and response headers. Therefore, a missing httpOnly coupled with XSS vulnerability is a recipe for stolen session token.

Solution:

If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by clientside JavaScript. This measure makes certain client-side attacks, such as cross-site scripting, slightly harder to exploit by preventing them from trivially capturing the cookie's value via an injected script.

1. $config['cookie\_httponly'] = TRUE;

Change the 'cookie\_httponly' to true in config.php file .

1. **Software Version Disclosure:**

Issue:

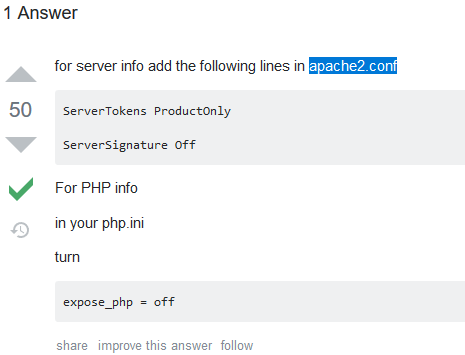
The server software versions used by the application are revealed by the web server.

Displaying version information of software information could allow an attacker to determine which vulnerabilities are present in the software, particularly if an outdated software version is in use with published vulnerabilities.

Solution :

hide software version in the responses

Add the <?php header\_remove("X-Powered-By"); ?> code in codeigniter controller construct function



1. **Failure To Restrict URL Access:**

Issue:

Without login user can access the after login functionality and can access the images after logout

Solution:

Add the below code in inside of the .htaccess file and upload into the directory

<IfModule authz\_core\_module>

Require all denied

</IfModule>

<IfModule !authz\_core\_module>

Deny from all

</IfModule>

And write the function on default or authentication controller where the session is not checked , pass the folder and file name as parameter

function show\_image($folder='',$file='') {

if ($this->session->userdata('stlid') != '' || $this->session->userdata('scinst') != '') {

$img\_path = $folder.'/'.$file;

$fp = fopen($img\_path,'rb');

header('Content-Type: image/png');

header('Content-length: ' . filesize($img\_path));

fpassthru($fp);

}else{

redirect('/','refresh');

}

}

1. **Cross Site Request Forgery :**

Issue :

Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

Solution :

Add the CSRF token functionality from code igniter library

**In controller**   
public function registerInsert()

{

$csrf = array(

'name' => $this->security->get\_csrf\_token\_name(),

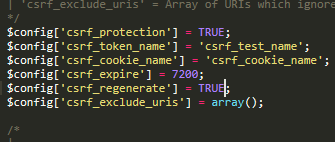
'hash' => $this->security->get\_csrf\_hash()

);

}

**In view**   
<input type="hidden" name="<?php echo $this->security->get\_csrf\_token\_name(); ?>" value="<?php echo $this->security->get\_csrf\_hash(); ?>">

Config.php



1. **Accessing / Modifying Other User Information:**

Issue :

We can access the other user information if the raw user id has been mentioned

Ex : localhost/scholarship/user/2

Solution :

Add the uniq id i.e. random\_string(‘alnum’,20); function to generate a random character and store it in db , so that we can retrieve the details using the same random characters as uniq id

1. **Automated Form Submission :**

Issues

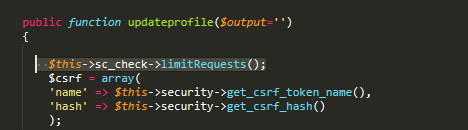
It is possible to automate the form submission process & submit multiple forms.

This vulnerability could cause loss of availability and results filling database with unwanted forms. This may leads to DOS(Denial of Service) attack to the server.

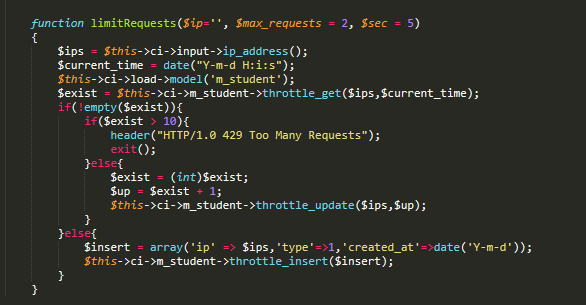
Solution:

Integrate the throttling functionality ( read codeigniter 4 throttlint for better understand)

Load the sc\_check library



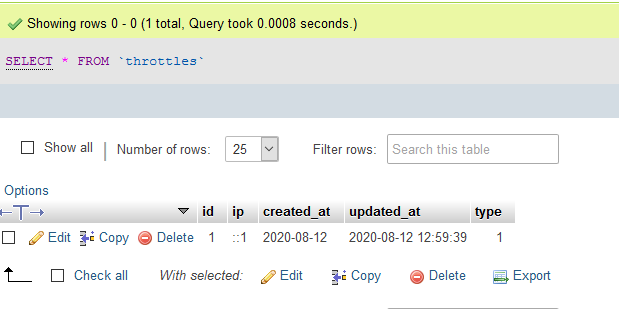
Library function



Update the limit value in db



Db table



1. **Malicious File Upload**

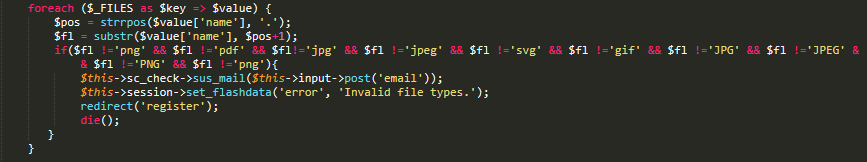
Issues:

Attacker can upload any kind of file and hack or redirect the website

Ex : script file, and sql files

Solution :

Validate the file type and sent the mail to concerned team automatically if someone try to upload the file is not mentioned in the list





1. **Brute Force Attack**

Issue:

A Brute Force Attack is the simplest method to gain access to a site or server (or anything that is password protected). It tries various combinations of usernames and passwords again and again until it gets in.

Solution:

Add the codeigniter image recapctha functionality

1. **Security Headers Missing :**

Isuue

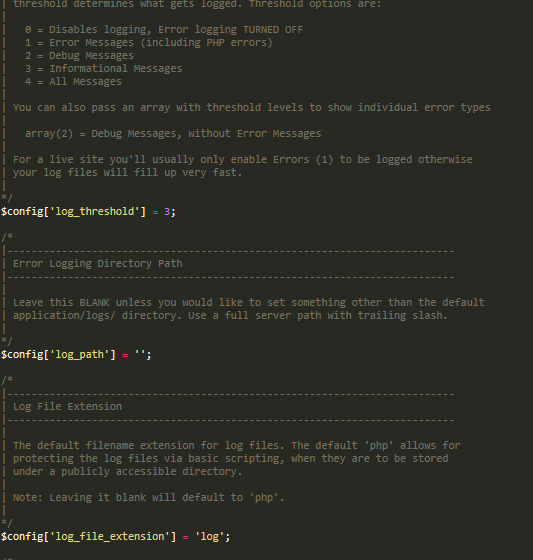
Headers are part of the HTTP specification, defining the metadata of the message in both the HTTP request and response. While the HTTP message body is often meant to be read by the user, metadata is processed exclusively by the web browser and has been included in HTTP protocol since version 1.0.

Solution :

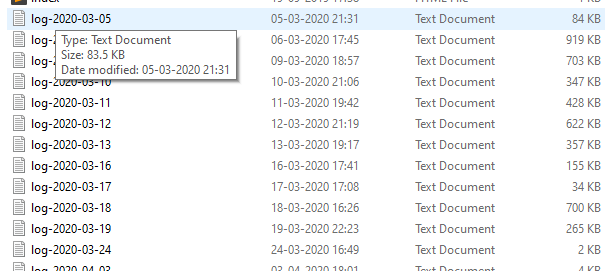
Add the below list of headers

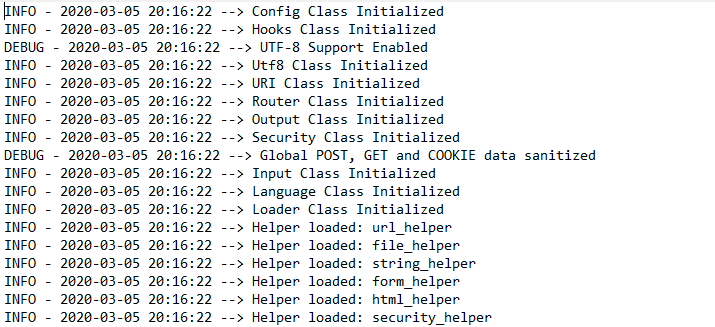
1. header("X-Frame-Options: DENY");
2. header("X-XSS-Protection: 1; mode=block");
3. header("X-Content-Type-Options: nosniff");
4. header("Strict-Transport-Security: max-age=31536000");
5. header("Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self';");
6. header("Referrer-Policy: origin-when-cross-origin");
7. header("Expect-CT: max-age=7776000, enforce");
8. **Insufficient Events Logs :**

Maintain the activaty log every user every actions by doing the below mentioned functionalities



LOG results

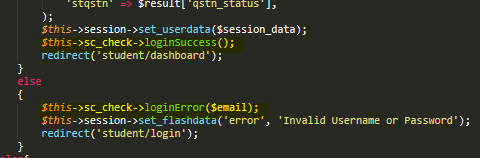




1. **Insufficient Warnings And Errors Logs :**

Store the login success and login failed data so that we can get to know that who has performed a suspisious activity by trying to login multiple times

IN controller



In Library

