**Language issues**

**Weak typing**

Implicit type conversions is happening here. When check whether two variables are equal, it should check the variable type also.Use *‘*===’ instead of *‘*==’

No issue found.

**Exceptions and error handling**

Use proper exception handling, error\_reporting(0); is not a proper way to hide errors.

The code from include\_file.php at line number 2

error\_reporting(1);

Instead do this

try {

//code

} catch (Exception $e) {

//code after exception caught

echo 'Caught exception: ', $e->getMessage(), "\n";

}

**Unhelpful built-ins**

No such unhelpful built-ins used in Area of Concerns module.

**Framework issues**

**URL routing**

There is no proper routing in this application.

**Input handling**

Input handling is not correctly done in this application. Input is directly is applied in the query.

The code from add-area-of-concerns.php at line number 15

if ($\_REQUEST['FrmSubmit'] || $\_REQUEST['FrmDraft']) {

//area of concern insertion

$DataArr['project\_id'] = $project\_id;

$DataArr['action\_required\_by\_id'] = $\_REQUEST['action\_required\_by'];

$DataArr['category\_id'] = $\_REQUEST['category'];

$DataArr['manager\_id'] = $projectArr[0]['proj\_crtl\_mgr\_id'];

if(isset($\_REQUEST['record\_no'])) {

$DataArr['record\_no'] = $\_REQUEST['record\_no'];

}

$DataArr['area\_of\_concern'] = $\_REQUEST['area\_of\_concern'];

$DataArr['reason\_for\_concern'] = $\_REQUEST['reason\_for\_concern'];

$DataArr['target\_date'] = $\_REQUEST['target\_date'];

$DataArr['measures\_taken'] = $\_REQUEST['measures\_taken'];

$DataArr['notes'] = $\_REQUEST['notes'];

$DataArr['is\_active'] = 1;

$DataArr['sys\_created\_by'] = $\_SESSION['USER\_ID'];

$DataArr['sys\_create\_date'] = date('Y-m-d H:i:s');

$inserted\_id = $db->insert\_tableValues('[dbo].[area\_of\_concern]', $DataArr);

if ($inserted\_id) {

//area\_of concern\_fidic insertion

$num\_fidic\_selection = count($\_REQUEST['fidic\_reference']);

for ($i = 0; $i < $num\_fidic\_selection; $i++) {

$FidArr['area\_of\_concern\_id'] = $inserted\_id;

$FidArr['fidic\_id'] = $\_REQUEST['fidic\_reference'][$i];

$FidArr['is\_active'] = 1;

$FidArr['sys\_created\_by'] = $\_SESSION['USER\_ID'];

$FidArr['sys\_create\_date'] = date('Y-m-d H:i:s');

$inserted\_fidic\_id = $db->insert\_tableValues('[dbo].[area\_of\_concern\_fidic]', $FidArr);

}

//area\_of concern\_response insertion

if ($\_REQUEST['FrmDraft']) {

$status = 1;

} else {

$status = 2;

}

$ResponseArr['area\_of\_concern\_id'] = $inserted\_id;

$ResponseArr['status\_id'] = $status;

$ResponseArr['description'] = $\_REQUEST['comment'];

$ResponseArr['is\_active'] = 1;

$ResponseArr['sys\_created\_by'] = $\_SESSION['USER\_ID'];

$ResponseArr['sys\_create\_date'] = date('Y-m-d H:i:s');

$inserted\_response\_id = $db->insert\_tableValues('[dbo].[area\_of\_concern\_response]', $ResponseArr);

//file attachments operations

$attacharr = $attachments->attach($\_FILES, $inserted\_id, $\_REQUEST['attach\_desc']);

header("Location:area-of-concerns.php");

exit();

}

}

Instead do this

Use the input data in the query after proper validation and filtering.

This problem is also in area-of-concerns.php, report-area-of-concerns.php, edit-area-of-concerns.php, response-area-of-concerns.php, area-of-concerns-pdf.php

**Template language**

HTML is not set as a template so not using template language.

Instead create a HTML template and use the template for common pages.

**Other inadequacies**

No protection mechanism is used.

Instead use a CSRF protection mechanism.

**Third party PHP code**

No third party PHP code used.

Use third party PHP code only if necessary.

Untrusted data

Any of the data input is not validated, using the correct methodology, or filtered, before considering them untainted. The data is directly used in database queries.

The code from area-of-concerns.php at line number 6

if (isset($\_REQUEST['aocid'])) {

$aoc\_id = $\_REQUEST['aocid'];

$UpdateArr['is\_active'] = 0;

$UpdateArr['sys\_deleted\_by'] = $\_SESSION['USER\_ID'];

$UpdateArr['sys\_delete\_date'] = date('Y-m-d H:i:s');

$aoc\_delete\_status = $db->update\_tableValue('[dbo].[area\_of\_concern]', $UpdateArr, "id = $aoc\_id");

$aoc\_fidic\_status = $db->update\_tableValue('[dbo].[area\_of\_concern\_fidic]', $UpdateArr, "area\_of\_concern\_id = $aoc\_id");

$aoc\_response\_status = $db->update\_tableValue('[dbo].[area\_of\_concern\_response]', $UpdateArr, "area\_of\_concern\_id = $aoc\_id");

$aoc\_attach\_status = $db->update\_tableValue('[dbo].[attachment]', $UpdateArr, "form\_id = $aoc\_id");

header("Location:area-of-concerns.php");

exit();

}

Instead do this

Use the input data in the query after proper validation and filtering.

This problem is also in add-area-of-concerns.php, report-area-of-concerns.php, edit-area-of-concerns.php, response-area-of-concerns.php, area-of-concerns-pdf.php

File uploads

File upload is did as per object oriented manner.

Use of $\_REQUEST

$\_REQUEST is used in this application.

The code from report-area-of-concerns.php at line number 4

$aoc\_id = $\_REQUEST['id'];

Instead do this

Specify $\_GET, $\_POST or $\_COOKIE instead of $\_REQUEST. Here the data arrived as POST so use   
$aoc\_id = $\_POST['id'];

This problem is also in area-of-concerns.php, add-area-of-concerns.php, edit-area-of-concerns.php, response-area-of-concerns.php, area-of-concerns-pdf.php

Database Cheat Sheet

Since a single SQL Injection vulnerability permits the hacking of your website, and every hacker first

Never concatenate or interpolate data in SQL

There is string of SQL that includes user data.

The code from left-nav.php at line number 44

$sql\_form = "SELECT F.id,F.name,F.form\_page\_name,F.report\_page\_name FROM [dbo].[user\_access] AS UA LEFT JOIN [dbo].[form] AS F ON UA.form\_id = F.id LEFT JOIN [dbo].[module] AS M ON F.module\_id = M.id WHERE UA.user\_id = '".$\_SESSION['USER\_ID']."' AND M.id = '". $SubmoduleArr[$k]['id']."' AND UA.access IN (1,3) AND F.form\_type IN (1,3)";

Instead do this

Use prepared statement.

$conn = sqlsrv\_connect( $serverName, $connectionInfo);

if( $conn === false) {

die( print\_r( sqlsrv\_errors(), true));

}

$sql = "UPDATE Table\_1

SET OrderQty = ?

WHERE SalesOrderID = ?";

// Initialize parameters and prepare the statement.

// Variables $qty and $id are bound to the statement, $stmt.

$qty = 0; $id = 0;

$stmt = sqlsrv\_prepare( $conn, $sql, array( &$qty, &$id));

Escaping is not safe

mysql\_real\_escape\_string is not used in this application.

Use prepared statement.

$conn = sqlsrv\_connect( $serverName, $connectionInfo);

if( $conn === false) {

die( print\_r( sqlsrv\_errors(), true));

}

$sql = "UPDATE Table\_1

SET OrderQty = ?

WHERE SalesOrderID = ?";

// Initialize parameters and prepare the statement.

// Variables $qty and $id are bound to the statement, $stmt.

$qty = 0; $id = 0;

$stmt = sqlsrv\_prepare( $conn, $sql, array( &$qty, &$id));

Use Prepared Statements

Prepared statements not used.

Use prepared statement.

$conn = sqlsrv\_connect( $serverName, $connectionInfo);

if( $conn === false) {

die( print\_r( sqlsrv\_errors(), true));

}

$sql = "UPDATE Table\_1

SET OrderQty = ?

WHERE SalesOrderID = ?";

// Initialize parameters and prepare the statement.

// Variables $qty and $id are bound to the statement, $stmt.

$qty = 0; $id = 0;

$stmt = sqlsrv\_prepare( $conn, $sql, array( &$qty, &$id));

ORM

ORMs (Object Relational Mappers) not used in this application.

Use any ORM. Eg: Propel.

Encoding Issues

**Use UTF-8 unless necessary**

UTF-8 encoding not specified in the code.

Use

$DB->set\_charset('utf8');

Other Injection Cheat Sheet

Shell Injection

There is no method used in this application for preventing shell injection.

Use prepared statement.

$conn = sqlsrv\_connect( $serverName, $connectionInfo);

if( $conn === false) {

die( print\_r( sqlsrv\_errors(), true));

}

$sql = "UPDATE Table\_1

SET OrderQty = ?

WHERE SalesOrderID = ?";

// Initialize parameters and prepare the statement.

// Variables $qty and $id are bound to the statement, $stmt.

$qty = 0; $id = 0;

$stmt = sqlsrv\_prepare( $conn, $sql, array( &$qty, &$id));

Code Injection

eval() and preg\_replace() is not using in this application.

Use prepared statement.

$conn = sqlsrv\_connect( $serverName, $connectionInfo);

if( $conn === false) {

die( print\_r( sqlsrv\_errors(), true));

}

$sql = "UPDATE Table\_1

SET OrderQty = ?

WHERE SalesOrderID = ?";

// Initialize parameters and prepare the statement.

// Variables $qty and $id are bound to the statement, $stmt.

$qty = 0; $id = 0;

$stmt = sqlsrv\_prepare( $conn, $sql, array( &$qty, &$id));

Other Injections

No methods used to prevent any injection.

Use prepared statement.

$conn = sqlsrv\_connect( $serverName, $connectionInfo);

if( $conn === false) {

die( print\_r( sqlsrv\_errors(), true));

}

$sql = "UPDATE Table\_1

SET OrderQty = ?

WHERE SalesOrderID = ?";

// Initialize parameters and prepare the statement.

// Variables $qty and $id are bound to the statement, $stmt.

$qty = 0; $id = 0;

$stmt = sqlsrv\_prepare( $conn, $sql, array( &$qty, &$id));

XSS Cheat Sheet

No Tags

In this application, they using standard PHP templating and the echo statement.There is no filtering before showing html to the browser.

You should use a template engine that applies HTML escaping **by default** - see below. All HTML should be passed out through the template engine.

If you cannot switch to a secure template engine, you can use the function below on all untrusted data.

**Keep in mind that this scenario won't mitigate XSS when you use user input in dangerous elements (style, script, image's src, a, etc.)**, but mostly you don't. Also keep in mind that every output that is not intended to contain HTML tags should be sent to the browser filtered with the following function.

//xss mitigation functions

function xssafe($data,$encoding='UTF-8')

{

return htmlspecialchars($data,ENT\_QUOTES | ENT\_HTML401,$encoding);

}

function xecho($data)

{

echo xssafe($data);

}

//usage example

<input type='text' name='test' value='<?php

xecho ("' onclick='alert(1)");

?>' />

Untrusted Tags

No Secure Encoding library is used.

you have to use a **Secure Encoding** library. This is usually hard and slow, and that's why most applications have XSS vulnerabilities in them. OWASP ESAPI has a bunch of codecs for encoding different sections of data. There's also OWASP AntiSammy and HTMLPurifier for PHP.

Templating engines

They are not using any template engines.

You should use any templating engines. There exist several of these engines. A good example is twig. Other popular template engines are Smarty, Haanga and Rain TPL.

Others

No hashing is required in this module.

CSRF Cheat Sheet

CSRF mitigation is not implemented.

CSRF mitigation is easy in theory, but hard to implement correctly. First, a few tips about CSRF:

* Every request that does something noteworthy, should be CSRF mitigated. Noteworthy things are changes to the system, and reads that take a long time.
* CSRF mostly happens on GET, but is easy to happen on POST. Don't ever think that post is secure.

The [OWASP PHP CSRFGuard](https://www.owasp.org/index.php/PHP_CSRF_Guard) is a code snippet that shows how to mitigate CSRF. Only copy pasting it is not enough. In the near future, a copy-pasteable version would be available (hopefully). For now, mix that with the following tips:

* Use re-authentication for critical operations (change password, recovery email, etc.)
* If you're not sure whether your operation is CSRF proof, consider adding CAPTCHAs (however CAPTCHAs are inconvenience for users)
* If you're performing operations based on other parts of a request (neither GET nor POST) e.g Cookies or HTTP Headers, you might need to add CSRF tokens there as well.
* AJAX powered forms need to re-create their CSRF tokens. Use the function provided above (in code snippet) for that and never rely on Javascript.
* CSRF on GET or Cookies will lead to inconvenience, consider your design and architecture for best practices.

Authentication and Session Management Cheat Sheet

Session Management

They are using PHP session.There is no other custom session mechanism used in this application.

Instead of this use a framework and use its custom session mechanism.

**Session Hijacking Prevention**

No storing any client IP for the prevention of Session Hijacking.

To implement Session Hijacking Prevention, simply store the client IP in the session first time it is created, and enforce it to be the same afterwards. The code snippet below returns client IP address:

$IP = getenv ( "REMOTE\_ADDR" );

**Invalidate Session ID**

There are not invalidate session when Session Hijacking. There is no logging of events.

You should invalidate (unset cookie, unset session storage, remove traces) of a session whenever a violation occurs (e.g 2 IP addresses are observed). A log event would prove useful. Many applications also notify the logged in user (e.g GMail).

**Rolling of Session ID**

Rolling of session ID is not implemented.

You should roll session ID whenever elevation occurs, e.g when a user logs in, the session ID of the session should be changed, since it's importance is changed.

**Exposed Session ID**

No exposing session ID in this application. Which is good. Transfer session ID over TLS whenever session holds confidential information, otherwise a passive attacker would be able to perform session hijacking.

**Session Fixation**

[session\_regenerate\_id()](http://www.php.net/session_regenerate_id) is not implemented.

Invalidate the Session id after user login (or even after each request) with [session\_regenerate\_id()](http://www.php.net/session_regenerate_id).

**Session Expiration**

There is no expiring of session. Session always available until browser is closed or logout.

A session should expire after a certain amount of inactivity, and after a certain time of activity as well. The expiration process means invalidating and removing a session, and creating a new one when another request is met.

Also keep the log out button close, and unset all traces of the session on log out.

* Inactivity Timeout
  + Expire a session if current request is X seconds later than the last request. For this you should update session data with time of the request each time a request is made. The common practice time is 30 minutes, but highly depends on application criteria.
  + This expiration helps when a user is logged in on a publicly accessible machine, but forgets to log out. It also helps with session hijacking.
* General Timeout
  + Expire a session if current session has been active for a certain amount of time, even if active. This helps keeping track of things. The amount differs but something between a day and a week is usually good. To implement this you need to store start time of a session.

**Cookies**

Cookies not used in this application.

Use cookie if necessary. Use this for cookie operations.

Setting new cookie

=============================

<?php

setcookie("name","value",time()+$int);

/\*name is your cookie's name

value is cookie's value

$int is time of cookie expires\*/

?>

Getting Cookie

=============================

<?php

echo $\_COOKIE["your cookie name"];

?>

Updating Cookie

=============================

<?php

setcookie("color","red");

echo $\_COOKIE["color"];

/\*color is red\*/

/\* your codes and functions\*/

setcookie("color","blue");

echo $\_COOKIE["color"];

/\*new color is blue\*/

?>

Deleting Cookie

==============================

<?php

unset($\_COOKIE["yourcookie"]);

/\*Or\*/

setcookie("yourcookie","yourvalue",time()-1);

/\*it expired so it's deleted\*/

?>

Authentication

**Remember Me**

Remember me functionality is not implemented.

If you add remember me functionality it will be better. You should use this strategy.

1. When the user successfully logs in with Remember Me checked, a login cookie is issued in addition to the standard session management cookie.
2. The login cookie contains a series identifier and a token. The series and token are unguessable random numbers from a suitably large space. Both are stored together in a database table, the token is hashed (sha256 is fine).
3. When a non-logged-in user visits the site and presents a login cookie, the series identifier is looked up in the database.
   1. If the series identifier is present and the hash of the token matches the hash for that series identifier, the user is considered authenticated. A new token is generated, a new hash for the token is stored over the old record, and a new login cookie is issued to the user (it's okay to re-use the series identifier).
   2. If the series is present but the token does not match, a theft is assumed. The user receives a strongly worded warning and all of the user's remembered sessions are deleted.
   3. If the username and series are not present, the login cookie is ignored.