

Plugin code review (Security)

For

User Field plugin = local_navigatr

by

Catalyst IT Europe LTD

October 2025

Confidential




Document purpose	2
Overall Result	4
Security Item Results	4
In-Depth Analysis	5
SQL Parameterisation	5
Permissions/Authentication	6
Cross-Site Scripting (XSS)	7
Use of Globals for User-Supplied Data	8
Session Hijacking (CSRF)	9
Other	10

Document purpose

To document analysis and ratings of plugins in relation to security issues.

Depending on the size of the plugin, not all code may need to be reviewed. In the case where the plugin is too large to feasibly review it all, code samples will be chosen at random for review and assumptions driven from those samples. For example, if 10 SQL queries are reviewed and 2 found to be insecure, it will be the assumption that 20% of SQL queries in the plugin are insecure.

Each security item in this document will be assigned a colour-coded grade (green, amber, red) based on the review. The plugin itself will then be assigned a similar grade as an overall assessment of the security of the plugin:

-  **Green** - This means that the plugin is authorised for deployment into your system.
-  **Amber** - This means that the plugin requires approval from the client before deployment into the system, and may be removed without warning if in the future it is deemed to be a risk.
-  **Red** - This means that the plugin is not authorised for deployment into your system, and the security issues must be resolved before it can be authorised.

Overall Result

Plugin Version	Git Hash	Result	Reviewed by
2025010701	fa74760		Simon Thornett (16-10-2025)

Security Item Results

Security Item	Result	Notes
2025010701 (16-10-2025)		
SQL Parameterisation	✓	
Permissions/Authentication	●	
Cross-Site Scripting (XSS)	✓	
Use of \$GLOBALS	●	
Session Hijacking (CSRF)	✓	
Other	✗	
xxxxxxxxxx (DD-MM-YYYY)		
SQL Parameterisation	✓●✗	
Permissions/Authentication	✓●✗	
Cross-Site Scripting (XSS)	✓●✗	
Use of \$GLOBALS	✓●✗	
Session Hijacking (CSRF)	✓●✗	
Other	✓●✗	
xxxxxxxxxx (DD-MM-YYYY)		
SQL Parameterisation	✓●✗	
Permissions/Authentication	✓●✗	
Cross-Site Scripting (XSS)	✓●✗	
Use of \$GLOBALS	✓●✗	
Session Hijacking (CSRF)	✓●✗	
Other	✓●✗	

In-Depth Analysis

In this section we go into greater detail on each security item – what we reviewed, what issues we found, and how to resolve them.

Add a new row for each result/issue as required.

SQL Parameterisation

In this step we check the SQL queries built in the plugin to ensure that no variables are passed directly into the query string, and all user-supplied data of any kind is passed as a parameter to the query. This includes variables holding config values, which can be manipulated through the config UI.

Results		
Plugin Version (Date)	Sampled	Issues Found
2025010701 (16-10-2025)	All queries	0

Issues			
Issue No.	Plugin Version (Date)	Issue	File name:line no

Permissions/Authentication

In this step we check that relevant capability checks are done on any page where the user could alter system data, or see data from other users. It is expected that the plugin will define capabilities to be used in these scenarios, or use in-built capabilities. We also check that, if a page does not define any capabilities to restrict access, it at the very least does an authentication check to ensure the user is logged in (unless there is a specific reason for allowing guest access).

Results		
Plugin Version (Date)	Sampled	Issues Found
2025010701 (16-10-2025)	All files	1

Issues			
Issue No.	Plugin Version (Date)	Issue	File name:line no
1	2025010701 (16-10-2025)	Capability mismatch between settings.php and settings_page.php checking either custom capability or site:config	settings.php:32 settings_page.php:29

Cross-Site Scripting (XSS)

In this step we check that there are no XSS vulnerabilities in the code, whereby user-submitted data is output directly into the HTML without any sanitisation or filtering. It is expected that the plugin will use Mustache templates for rendering HTML, with the data passed into it via a renderer and filtered using the relevant output tags within the template. Some plugins may build their HTML directly using PHP, which can make it much harder to accurately review. In cases where a lot of HTML is built in this way, the review will be done on code samples, and assumptions based on those samples.

Results		
Plugin Version (Date)	Sampled	Issues Found
2025010701 (16-10-2025)	All files	0

Issues			
Issue No.	Plugin Version (Date)	Issue	File name:line no

Use of Globals for User-Supplied Data

In this step we check to make sure that any user-submitted data is being filtered through Moodle’s in-built methods optional_param and required_param, and not being accessed directly from the global variables. This is important because those in-built methods can filter the data based on expected type, whereas direct access via the globals relies on the plugin developer to remember to filter all of the data manually.

Results		
Plugin Version (Date)	Sampled	Issues Found
2025010701 (16-10-2025)	All files	2

Issues			
Issue No.	Plugin Version (Date)	Issue	File name:line no
1	2025010701 (16-10-2025)	Accessing globals through data_submitted instead of form->get_data	settings_page.php:37
		Optional param usage instead of required param	course_settings.php:30 badge_selection.php:29

Session Hijacking (CSRF)

In this step we ensure that any form or page which creates, edits, or deletes data from the system, checks that the form was intentionally submitted by the logged in user. This is done in Moodle by doing a check with the function `require_sesskey()`. This checks that a parameter has been passed to the page, either via a form submission, or through the URL, which contains the user's session ID, and that it matches the session ID of the logged in user. This means the form will fail to submit if they submit a fraudulent form/click a fraudulent URL, as it will not contain their valid session ID.

Results		
Plugin Version (Date)	Sampled	Issues Found
2025010701 (16-10-2025)	All files	0

Issues			
Issue No.	Plugin Version (Date)	Issue	File name:line no

Other

In this step we check for any other smaller security issues which we feel may be present, based on the review of the code we have done so far. These can include:

- Missing MOODLE_INTERNAL checks to ensure pages cannot be loaded via URL if they should not be
- Data Leakage/Exposure – Any areas where it appears data may be viewable by users who should not be able to see it
- Transmission to 3rd parties – Any areas where data is transferred out of the system to a 3rd party
- Correct Parameter Types – Any parameters created using optional/required_param, should set a sensible type. E.g. PARAM_INT, PARAM_TEXT, PARAM_LOCALURL, etc...
- Any form fields containing passwords should use a specific “password” type, not plain text
- No external javascript. Or if it is unavoidable, only from trusted sources.
- SSRF – Making sure any inclusion of files based on user-input is properly handled to avoid inclusion of unexpected files
- Shell Scripts – Making sure there are no shell scripts executed in the plugin unless for very good reason
- Logging – Any action which alters data should be logged for audit purposes

Results		
Plugin Version (Date)	Sampled	Issues Found
2025010701 (16-10-2025)	All files	44

Issue	File
Accessing of globals - Using \$form->get_data() instead of data_submitted() is strongly advised. In this form the usage of data_submitted bypasses validation and additionally still triggers the saving of the form on cancel (test by changing the username and clicking cancel, the row is still updated in the database).	local/navigatr/settings_page.php

admin_externalpage_setup('local_navigator_settings'); should be used instead of the login checks and capability checks. Currently the menu item is visible to users with the capability 'local/navigator:managecredentials' but the settings page can only be viewed by site config users making this redundant. AJAX handler: AJAX requests should be done through internal webservices, however this code appears to be redundant as I can't see anywhere that is calling it

local/navigator/settings_page.php

As mentioned above the capability checks do not match. Would advise using either '\$ADMIN->fulltree' or has_capability check

local/navigator/settings.php

Calls to debugging and error_log without errors. IF needed then events should be triggered as these are logged to the database. Debugging can output to the screen breaking the users workflow, and error_logs are not reliable for reporting etc

local/navigator/classes/observer.php

Cancel redirect to the wrong page resulting in an error message.

local/navigator/settings_page.php

Checking for course mapping - this is redundant as was already done in the observer so will never be hit.

local/navigator/classes/task/issue_badge_task.php

Codechecker results: errors 417

All

Codechecker results: warnings 577

All

Double saving of data - The settings form is saving the data on lines 73 - 89 without form validation and then again on 135 - 140 with form validation. This also results in the notification being output twice.

local/navigator/settings_page.php

File is in the wrong place and as such is not used by Moodle stating "This plugin does not implement the Moodle privacy API"

local/navigator/privacy/provider.php

get_access_token - the tokens are only valid for 60 seconds. This should be stored in a cache with a 60 second TTL not in the config table.

local/navigator/classes/local/token_manager.php

get_course_score - if there is no grade available it attempts to get the course progress with

local/navigator/classes/task/issue_badge_task.php

"get_course_progress_percentage", however this will fail as it's passing an int to something expecting an object.

HTML - html_writter usage should be replaced with moustache templates.	local/navigatr/settings_page.php
HTML and CSS - html_writter usage should be replaced with moustache templates and in-line css with a stylesheet.	local/navigatr/course_settings.php
misinformation - The "Navigatr Password" is not stored encrypted in the database as both the notification and readme state. It is stored in plain text in the config_plugins table.	local/navigatr/settings_page.php
Missing \$PAGE->set_context call	local/navigatr/badge_selection.php
Mostly redundant tests just checking if methods exist.	local/navigatr/tests/observer_test.php
On 401 the task will attempt to call "reauth" however this is not possible as it is a private method	local/navigatr/classes/task/issue_badge_task.php
Optional param \$courseid should be `required_param('courseid', PARAM_INT)` and then the subsequent exception would not be needed	local/navigatr/course_settings.php
Optional param \$courseid should be `required_param('courseid', PARAM_INT)` and then the subsequent exception would not be needed	local/navigatr/badge_selection.php
Password is required to be readdd each time due to \$form->set_data not being used	local/navigatr/settings_page.php
refresh_token - it appears that this is valid for 1 day, I would again advise using an application cache, not config	local/navigatr/classes/local/token_manager.php
See codechecker for issues	local/navigatr/lang/en/local_navigatr.php
See issue_badge_task_test	local/navigatr/tests/classes/local/token_manager_test.php
See issue_badge_task_test	local/navigatr/tests/classes/local/cache_test.php
See issue_badge_task_test	local/navigatr/tests/classes/local/api_client_test.php
See issue_badge_task_test	local/navigatr/tests/classes/form/admin_settings_form_test.php
See local/navigatr/settings.php regarding unneeded local/navigatr:managecredentials capability	local/navigatr/db/access.php
Set defaults - defined values should be set via set_data method, not setDefaults in the form.	local/navigatr/classes/form/provider_selection_form.php
Set defaults - defined values should be set via set_data method, not setDefaults in the form.	local/navigatr/classes/form/admin_settings_form.php

test_connection - \$debuglevel will log to the error log (assuming site level settings are configured), however a level of "error" does nothing.	local/navigatr/classes/local/api_client.php
Tests do not run	local/navigatr/tests/behat/features/navigatr_badge_issuance.feature
Tests reference log entries for strings that don't exist (i.e "Duplicate badge issuance prevented")	local/navigatr/tests/behat/features/navigatr_badge_issuance.feature
Tests reference steps that do not exist	local/navigatr/tests/behat/features/navigatr_badge_issuance.feature
Tests that only look at if a function exists of does the equivalent check of "a" == "a" (see test_execute_with_valid_data) no actual calling of the task is done.	local/navigatr/tests/classes/task/issue_badge_task_test.php
The "dedupekey" - without checking the API with multiple providers myself the question I have is "Is it possible that 2 providers would have teh same Badge ID? i.e A moodle admin changes their credentials to a different provider, would this dupe check falsly flag a duplicate?"	local/navigatr/classes/task/issue_badge_task.php
The class does not include the required methods to function.	local/navigatr/privacy/provider.php
Unused file	local/navigatr/tests/fixtures/navigatr_test_data.xml
Unused file	local/navigatr/tests/behat/behat_navigatr.php
unused function - get_badges	local/navigatr/classes/form/provider_selection_form.php
unused functions - get_providers, set_providers, clear_all, clear_providers, clear_badges, clear_user_detail.	local/navigatr/classes/local/cache.php
write_audit - Whilst good to have a record within the plugin database, there are no means to view this table and no core event triggers. If there are errors it will require direct database access to review and diagnose. I would advise either adding event triggers as mentioned, or including a report source for easier viewing.	local/navigatr/classes/task/issue_badge_task.php