Penetration Test Report

André Menrath

V 1.0
Amsterdam, February 10th, 2025
Public

Document Properties

Client	André Menrath
Title	Penetration Test Report
Target	The Event Bridge for ActivityPub plugin for WordPress.
Version	1.0
Pentester	Marcus Bointon
Authors	Marcus Bointon, Morgan Hill
Reviewed by	Morgan Hill
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	February 3rd, 2025	Marcus Bointon	Initial draft
0.2	February 6th, 2025	Morgan Hill	Review
1.0	February 10th, 2025	Marcus Bointon	1.0

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1	**Executive Summary**	**4**
1.1	Introduction	4
1.2	Scope of work	4
1.3	Project objectives	4
1.4	Timeline	4
1.5	Results In A Nutshell	4
1.6	Summary of Findings	5
1.6.1	Findings by Threat Level	5
1.6.2	Findings by Type	6
1.7	Summary of Recommendations	6
2	**Methodology**	**7**
2.1	Planning	7
2.2	Risk Classification	7
3	**Reconnaissance and Fingerprinting**	**9**
4	**Findings**	**10**
4.1	CLN-001 — Missing dependency on roave/security-advisories	10
4.2	CLN-004 — Dependency on abandoned package	11
4.3	CLN-002 — Ineffective static analysis configuration	12
5	**Non-Findings**	**14**
5.1	NF-003 — Input validation overview	14
6	**Future Work**	**17**
7	**Conclusion**	**18**
Appendix 1	**Testing team**	**20**

1 Executive Summary

1.1 Introduction

Between January 21, 2025 and January 29, 2025, Radically Open Security B.V. carried out a penetration test of Event Bridge for ActivityPub plugin for WordPress.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following target:

- The Event Bridge for ActivityPub plugin for WordPress.

The scoped services are broken down as follows:

- Code auditing.: 2 days
- **Total effort: 2 days**

1.3 Project objectives

ROS will perform a code audit of the Event Bridge for ActivityPub plugin for WordPress along with its developer, André Menrath, in order to assess its security, particularly with its interactions with other WordPress plugins and when receiving ActivityPub requests. To do so ROS will access the plugin's source code (provided in a Forgejo-hosted git repo) and guide the plugin's developers in attempting to find vulnerable pathways in the code in its interactions with its host environment and peer plugins.

1.4 Timeline

The security audit took place between January 21, 2025 and January 29, 2025.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Low and 2 N/A-severity issues.

We did not find anything critical, but did identify some broad shortcomings in input validation, which is not applied at an early stage, resulting in bad data making it too far into the processing stack before being sanitized or identified as invalid.

From a code quality perspective, the plugin uses a recent enough version of PHP to allow it to make extensive use of parameter and return typing, but this was largely missing CLN-002 (page 12).

To address those concerns we recommended applying validation earlier in the processing chain, and to make use of the phpstan static analysis tool to spot coding issues and enforce their identification in CI systems.

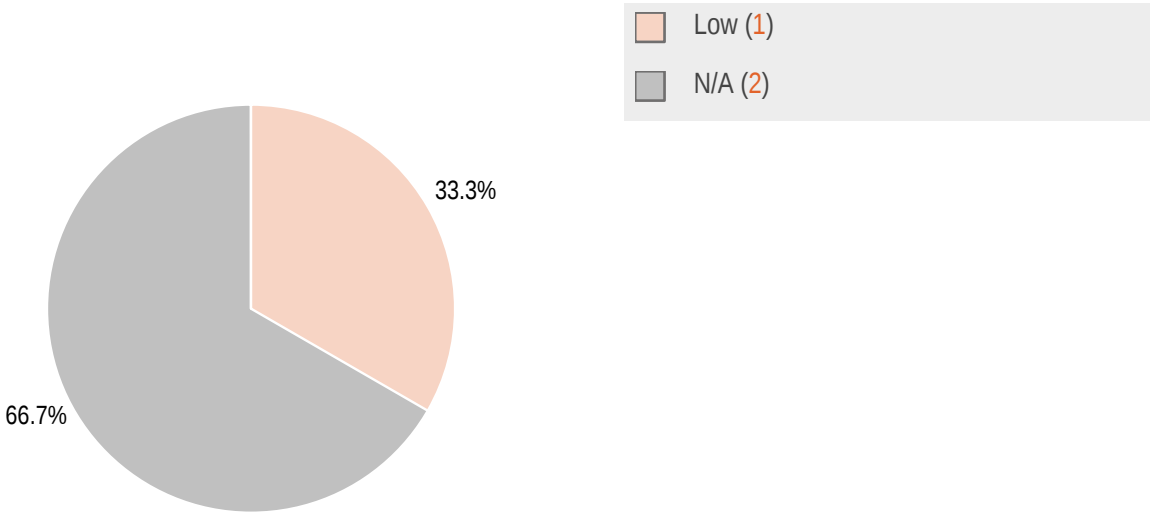
We identified an abandoned package dependency in CLN-004 (page 11), but this is only used in development contexts.

Our investigations of input handling can be found in non-finding NF-003 (page 14).

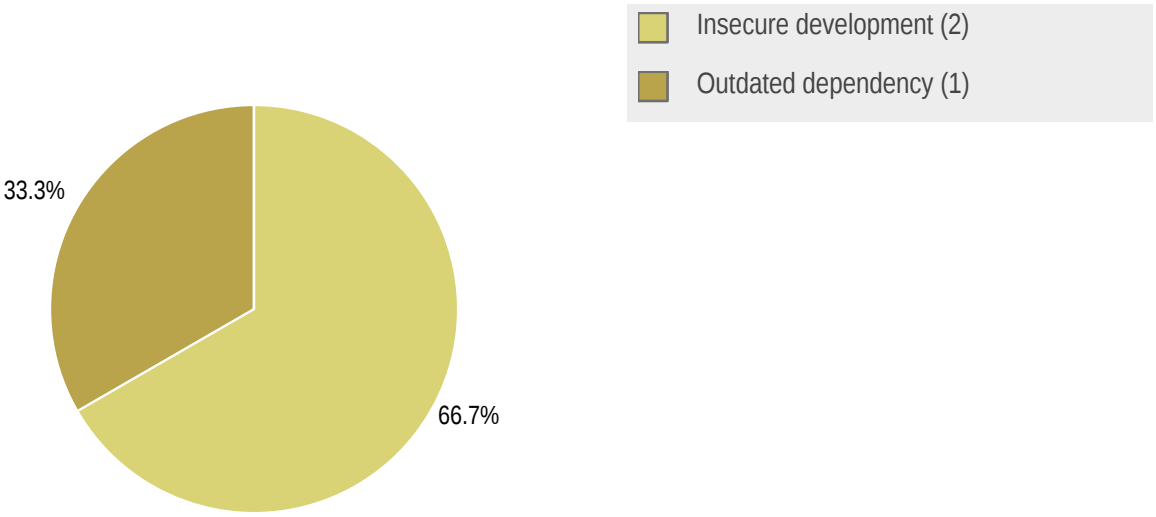
1.6 Summary of Findings

ID	Type	Description	Threat level
CLN-001	Insecure development	There is no dev dependency on roave/security-advisories to pre-emptively spot known-vulnerable dependencies.	Low
CLN-004	Outdated Dependency	composer audit reports that there is a development dependency on the phpunit/php-token-stream package, which is marked as abandoned.	N/A
CLN-002	Insecure development	There is a configuration for PHP Code Sniffer static analysis, but it fails to detect many simple issues.	N/A

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-001	Insecure development	• Add a dev dependency on the `roave/security-advisories` package.
CLN-004	Outdated Dependency	• Check whether this package is actually needed at all, or find a replacement for it.
CLN-002	Insecure development	• Configure a stricter static analysis tool and use and enforce it in CI.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2021) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- PHPStorm built-in inspections – https://www.jetbrains.com/phpstorm/
- Php Inspections (EA Ultimate) – https://plugins.jetbrains.com/plugin/16935-php-inspections-ea-ultimate-
- phpstan static analyser for PHP – https://phpstan.org

4 Findings

We have identified the following issues:

4.1 CLN-001 — Missing dependency on roave/security-advisories

Vulnerability ID: CLN-001

Vulnerability type: Insecure development

Threat level: Low

Description:

There is no dev dependency on `roave/security-advisories` to pre-emptively spot known-vulnerable dependencies.

Technical description:

The `roave/security-advisories` package extracts information about existing security issues in various composer projects from the FriendsOfPHP/security-advisories repository and the GitHub Advisory Database. The net result of that is that updating dependencies of the project will automatically create a conflict for any dependency that has a known vulnerability, providing earlier notification than would otherwise occur. This is similar to running `composer audit`, except that it happens automatically whenever dependencies are updated. Add the dependency using:

```
composer require --dev roave/security-advisories:dev-latest
```

Note that this is a dev dependency so does not affect deployed code, and is intentionally configured to depend on an unconstrained development branch so that its latest version is always used.

Impact:

Vulnerabilities in dependencies might remain in place for longer, extending their window of exposure for whatever they are vulnerable to.

Recommendation:

- Add a dev dependency on the `roave/security-advisories` package.

4.2 CLN-004 — Dependency on abandoned package

Vulnerability ID: CLN-004

Vulnerability type: Outdated Dependency

Threat level: N/A

Description:

`composer audit` reports that there is a development dependency on the `phpunit/php-token-stream` package, which is marked as abandoned.

Technical description:

`composer audit` reports this abandoned package, but does not suggest a replacement:

```
composer audit
No security vulnerability advisories found.
Found 1 abandoned package:
+--------------------------+-----------------------------------------+
| Abandoned Package        | Suggested Replacement                   |
+--------------------------+-----------------------------------------+
| phpunit/php-token-stream | none                                    |
+--------------------------+-----------------------------------------+
```

Impact:

Future vulnerabilities in this package will not be patched. There is no production security impact as this package is only used in development, but using outdated or abandoned software in any context is inadvisable.

Recommendation:

- Check whether this package is actually needed at all, or find a replacement for it.

4.3 CLN-002 — Ineffective static analysis configuration

Vulnerability ID: CLN-002	**Status:** Resolved
Vulnerability type: Insecure development	
Threat level: N/A	

Description:

There is a configuration for PHP Code Sniffer static analysis, but it fails to detect many simple issues.

Technical description:

Modern PHP static analysis tools are mature and highly effective in spotting straightforward problems in code.

There is a configuration for PHP Code Sniffer (phpcs, note not to be confused with PHP CS-Fixer, an entirely different tool) that is also run in CI, however, it appears to be concerned only with PHP version compatibility and coding standards (using the WordPress standard) and is not configured to spot anything further. Running other tools like the static analyser in PHPStorm, the commercial plugin Php Inspections (EA Ultimate), or phpstan (https://phpstan.org) reveals many small problems that might lead to production bugs. Issues include missing parameter and return types, type mismatches, inconsistent returns, unused properties or variables, and many other issues. None of these present immediate security concerns, but any bug or inconsistency is an opportunity for unexpected behaviour.

Impact:

No specific security impact, but simple code problems and bugs may go unnoticed until they cause problems in production.

Recommendation:

- Configure a stricter static analysis tool and use and enforce it in CI.

Update 2025-01-31 18:09:

In line with what we recommended in future work (page 17), the author has configured phpstan and used it to find numerous (over 600) common issues, in particular incorrect or missing docblocks, parameter types, and return types. It's

also been integrated into CI, so future regressions or issues in new code can be spotted more easily at an early stage. This investment provides a great improvement in code quality.

5 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

5.1 NF-003 — Input validation overview

The job of this bridge plugin is to convert an ActivityPub event request into an event record belonging to one of several popular event management plugins for WordPress via a set of adapters ("transmogrifiers"), thus extending their reach to the fediverse. To achieve that, it is mainly concerned with accepting standard ActivityPub requests, and remapping them into the data structures that the various Event plugins expect. These plugins normally accept requests in their own right, and as such are expected to handle input sanitization and validation themselves, meaning that the responsibilities of this bridge plugin end at that boundary, but it is of course in this plugin's interest to act in accordance with Postel's law (a.k.a. the robustness principle):

> Be conservative in what you send, be liberal in what you accept.

So we should expect the plugin to be somewhat tolerant of out-of-spec inputs, but forward only strictly known-good data to the destination plugins.

The plugin accepts several different ActivityPub request types:

- `Accept`
- `Create`
- `Delete`
- `Undo`
- `Update`

The most important of these from a security perspective is `Create`, because it accepts user-supplied data of multiple types. `Update` shares the same handler code, but uses an existing Event instead of making a new one. The others have very minimal attack surface accepting little other than an identifier.

The attack surface here is very small, really just this one entry point. To ascertain the security of this request handling, we examined what happens to each field in a typical ActivityPub `Create` request body:

```
{
    "id": "https://example.social/@user/events/party#create",
    "type": "Create",
    "actor": "https://example.social/users/User",
    "object": {
        "id": "https://example.social/@user/events/party",
        "type": "Event",
        "startTime": "2025-01-30T10:00:00Z",
        "name": "Fediverse Party",
        "to": "https://www.w3.org/ns/activitystreams#Public",
        "published": "2020-01-01T00:00:00Z"
```

```
    }
}
```

- `id` is a URL used as an identifier for the event request. It is sanitized and validated by the built-in WordPress function `esc_url`. If it lacks a URL scheme, `http://` is prepended. If it contains a scheme that's not on an allow list (such as `javascript`), the entire URL is set to an empty string. Surprisingly, if that happens, the Event is still created. After that, the URL doesn't appear to be used at all.
- `type` is the operation type of the request. This is validated to be one of `Create`, `Update`, `Delete`, `Announce`; any other value causes an error.
- `actor` is an arbitrary identifier of the entity that triggered the request; For this to be accepted, it must match a pre-configured entry in the Event Sources table exactly (i.e. an allowlist).
- `object` contains data about the event itself:
    - `id` is an identifier, expected to be a URL, and will be saved in the `guid` field in the database. If something other than a valid URL is passed in, it will fail validation, and a new URL is generated based on the host WordPress URL and a slug based on the `name` element (see below) instead.
    - `type` is the type of the request; anything other than `Event` is rejected.
    - `startTime` is a string containing an ISO-8601 string representing the start time. This is validated by attempting to parse it as a time string, and if that fails, or the parsed value is in the past, the request will be rejected. While no explicit validation of the length or format is applied (which might be a sensible addition), it will not end up as anything but a valid time string or a rejection.
    - `name` is a free-form text description of the event. The plugin does not parse, sanitise, or filter this value itself, but it is eventually passed through WordPress' standard `_sanitize_text_fields` function, which strips all HTML tags and URL-encoded chars. This appears effective, but is not in scope here.
    - `to` is an ActivityPub value expected to be exactly `https://www.w3.org/ns/activitystreams#Public`. This is a special value as defined in https://www.w3.org/TR/activitypub/#public-addressing that indicates that an event is not targeted at any specific person, but is fully public. If it's not this exact value, the request is ignored (though it still returns a `202 Accepted` response code).
    - `published` is meant to be an ISO time string. It is extracted and used to build an Event object without any apparent validation or sanitisation, however, it's never actually used or written to the database. It would seem sensible to explicitly discard this field if it's not being used.

Of these, the only field that is permitted to be a free-form value is `name`; everything else is required to conform to a strict set of values. Since this was the most interesting target, we attempted various kinds of injection into this field, including XSS and SQL, but the validation prevented any of them from succeeding.

Overall, everything in here is safe, however, much of the sanitisation and validation is either left late or passed on to WordPress when it could be handled earlier on in the process.

6 Future Work

- **Add support for phpstan in CI**
  Phpstan can help developers enforce coding standards and conventions beyond simple code style – enforcing types and helping avoid hard-to-maintain structures. This works even better if it's made part of automated CI processes to enforce the chosen rules.
- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Let us emphasize that security is a process, and this penetration test is just a single snapshot. Security must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security.

7 Conclusion

We discovered 1 Low and 2 N/A-severity issues during this penetration test.

This project had a very tight focus – examining the proposed changes to a WordPress plugin that acts as a broker between other WordPress plugins in order to add support for ActivityPub events. The attack surface was correspondingly small, really coming down to the handling of a single JSON endpoint. Consequently, almost all the effort was expended in tracing exactly what happened to each and every field within this structure, checking how it was sanitized and validated, and how it was handled after being passed on to its destination plugins, and ultimately into the WordPress database.

This processes discovered some shortcomings or omissions in validation processes, but no critical issues. The primary issue was that bad data was allowed to progress too far into the stack, when validation should have occurred much earlier in the processing chain. After reporting these concerns, the author immediately set about addressing them, and this work was almost complete by the end of the audit.

We also raised some small concerns about code quality configuration, in particular a lack of parameter and method return types (which can be used as the minimum PHP version is 7.4, which supports them). Static analysis tools, especially phpstan, are very effective in spotting and fixing these issues, so we recommended using them to deal with this concern. Progress was also made in configuring support for making use of phpstan in CI flows, aiding long-term stability and code quality and consistency.

We spotted one dependency on an abandoned PHP package that should be removed or swapped out for a supported implementation. We also recommended adding support for the `roave/security-advisories` package, which helps developers avoid introducing new dependencies on known-vulnerable packages.

At the time of finishing this audit, not all issues had been resolved, but significant progress has been made, particularly in configuring phpstan, and integrating it into CI. It's notable that we did not find any critical issues even prior to the changes. A primary driver for conducting this audit was to assuage upstream security concerns about this integration, and what has been done should act to assure developers that this has been well taken care of.

The author, André Menrath, was helpful, responsive, and enthusiastic throughout the audit, a pleasure to work with! We hope that this audit will help upstream developers accept the proposed changes, and contribute to expanding the reach of the fediverse.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Marcus Bointon	Marcus has a BSc in Computer Science and Digital Electronics from King's College London, and an MSc in interactive computer system design from Loughborough University of Technology. He maintains PHPMailer (a top-10 PHP project on GitHub), is a regular contributor to innumerable open-source projects, and he wrote the HTML5 specification for email addresses. He's the chief architect of the smartmessages.net email marketing system, and has worked with ROS since 2016, mainly as a technical writer & editor. You'll often find Marcus speaking at developer conferences on web development, email, devops, privacy, data protection, and security.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.