hashlock.

Security Audit

Energy Web 4th (DeFi)

October, 2025 info@hashlock.com.au Hashlock Pty Ltd

Table of Contents

CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE WHICH COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR USE OF THE CLIENT.

Executive Summary

The Energy Web team partnered with Hashlock to perform a security audit of the EWX parachain node, focusing on its integration with Polkadot Asset Hub. Hashlock conducted a manual review of the related Substrate runtime and pallet code to verify that the implementation and cross-chain mechanisms were secure and followed best practices.

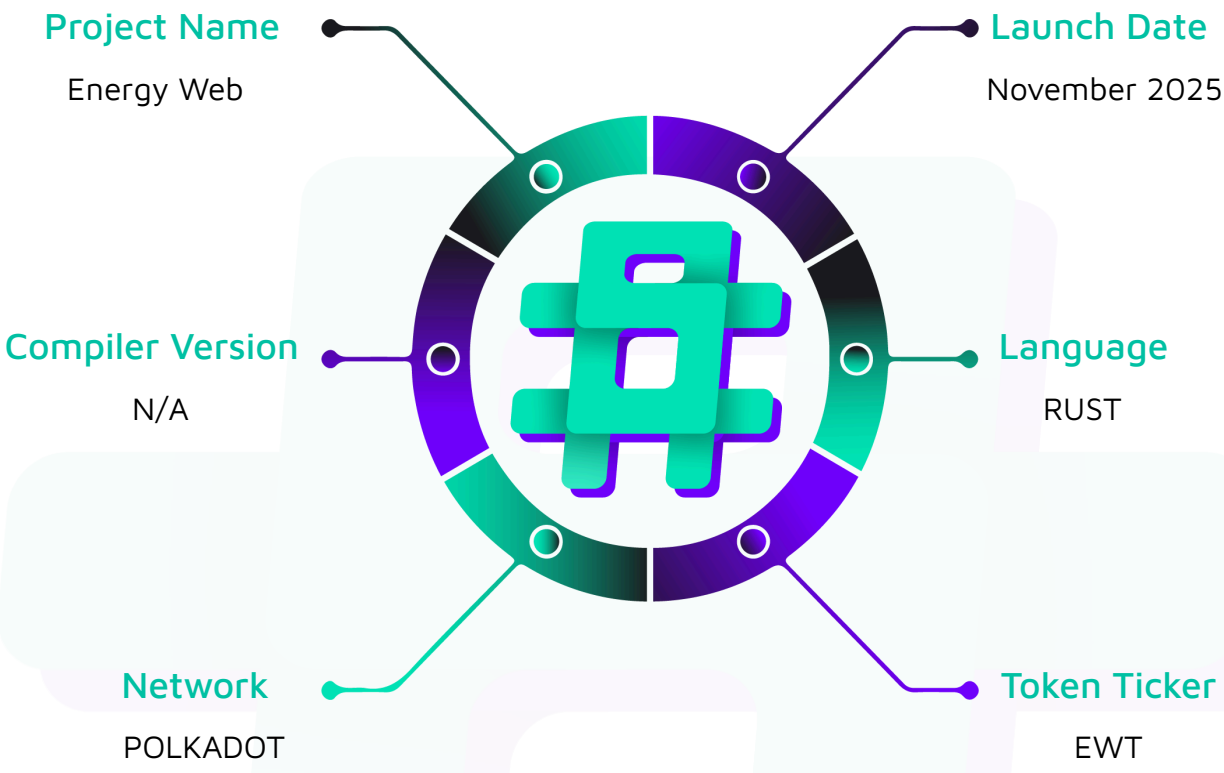
Project Context

Energy Web is a global, open-source nonprofit focused on accelerating the clean energy transition through decentralized digital infrastructure. Launched in 2017, the organization stewarded the Energy Web Chain (EWC), an enterprise-focused Proof-of-Authority blockchain. Energy Web is now transitioning to it's flagship network: Energy Web X (EWX), a Substrate-based Polkadot parachain.

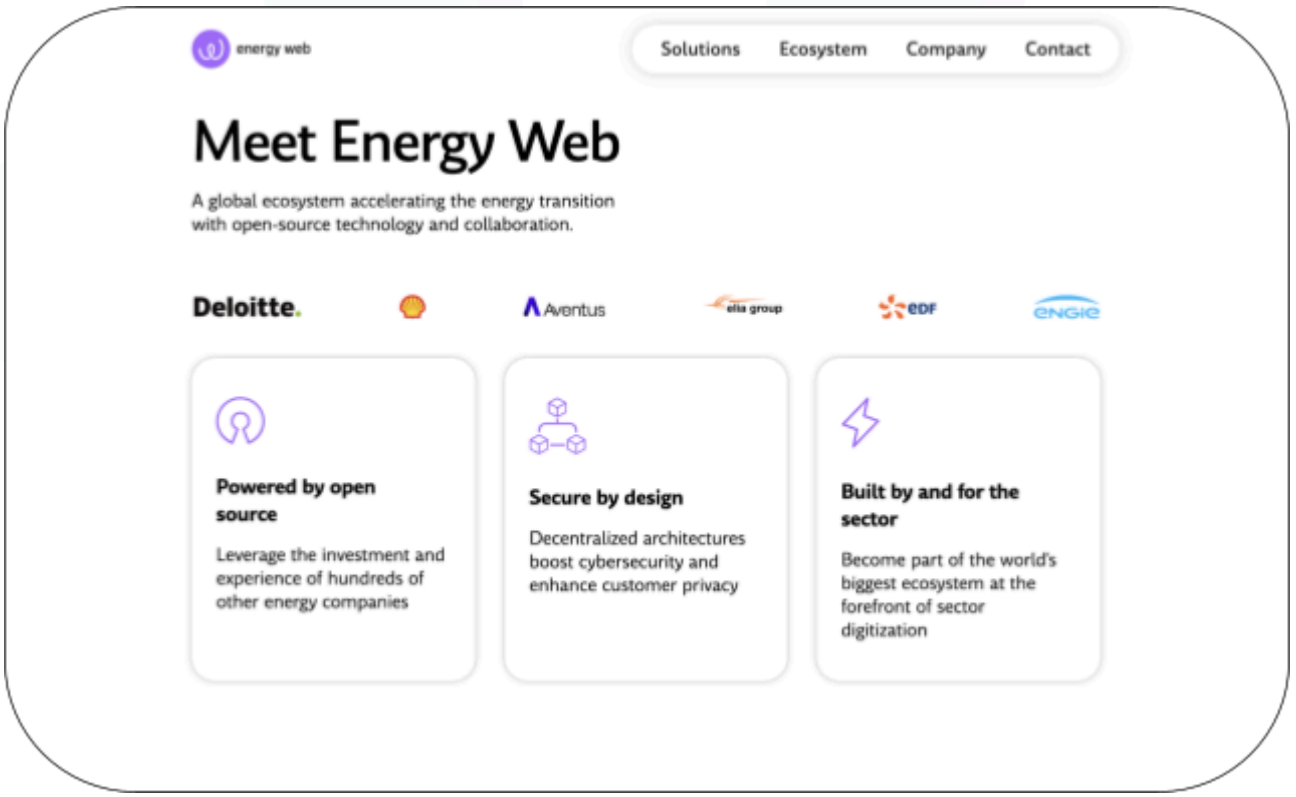
EWX introduces a permissionless Proof-of-Stake consensus model, enabling broad validator and delegator participation while unlocking staking rewards for participants and supporting a robust on-chain economy. To expand liquidity and cross-chain functionality, EWX integrates natively with Polkadot Asset Hub, enabling seamless registration, transfer, and utilization of foreign assets within the Energy Web ecosystem while maintaining compatibility with Ethereum and the legacy EWC network.

Project Name: Energy Web
Project Type: DeFi, Token, Bridge
Website: https://www.energyweb.org/
Logo:

Visualised Context:



Project Visuals:



Audit Scope

We at Hashlock audited the Rust code within the Energy Web project, the scope of work included a comprehensive review of the Substrate runtime and associated pallets related to the integration with Polkadot Asset Hub. We tested the EWX Integration with Asset Hub to check for their security and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

Description	**Energy Web EWX Integration with Asset Hub**
Platform	**Polkadot / Rust**
Audit Date	**October, 2025**
Contracts	lib.rs
Contracts 2	mod.rs
Contracts 3	pallet_xcm_benchmarks_fungible.rs
Contracts 4	xcm_config.rs
Audited GitHub Commit Hash	ea8065ea3853da7c003ef1fb6ce219c7afc04700
Fix Review GitHub Commit Hash	477ebd4b2f1d47a69fe72d5a288d86e3f4059ca3

Note: *Hashlock initially audited the code associated with the "Audited GitHub Commit Hash" and the findings presented in this report pertain specifically to that commit. For the "Fix Review GitHub Commit Hash", Hashlock solely verified the status of the identified findings and did not conduct a comprehensive review to assess any additional code implementations.*

Security Rating

After Hashlock's Audit, we found the pallet to be **"Hashlocked"**. The pallet code all follow simple logic, with correct and detailed ordering.

The 'Hashlocked' rating is reserved for projects that ensure ongoing security via bug bounty programs or on chain monitoring technology.

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the Audit Findings section. The list of audited assets is presented in the Audit Scope section and the project's contract functionality is presented in the Intended EWX Integration with Asset Hub Functions section.

All vulnerabilities initially identified have now been resolved and acknowledged.

Hashlock found:

1 High severity vulnerabilities

2 Low severity vulnerabilities

Caution: *Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.*

Intended EWX Integration with Asset Hub Functions

Claimed Behaviour	Actual Behaviour
runtime/stable/src/lib.rs - Main runtime configuration file that assembles all pallets and defines core blockchain parameters.	**Contract achieves this functionality.**
runtime/stable/src/weights/xcm/mod.rs - Calculates computational costs for XCM operations based on the number of assets being processed.	**Contract achieves this functionality.**
runtime/stable/src/weights/xcm/pallet_xcm_benchmarks_fungible.rs - Benchmark data containing measured execution times for XCM fungible asset operations.	**Contract achieves this functionality.**
runtime/stable/src/xcm_config.rs - Configures cross-chain messaging (XCM) functionality, such as asset transactors, origin converters, security barriers, and fee handling.	**Contract achieves this functionality.**

Code Quality

This audit scope involves the EWX Integration with Asset Hubs of the Energy Web project, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices and to help avoid unnecessary complexity that increases the likelihood of exploitation, however, some refactoring were recommended to optimize security measures.

The code is very well commented on and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

Audit Resources

We were given the Energy Web project EWX Integration with Asset Hub code in the form of Github access.

As mentioned above, code parts are well commented. The logic is straightforward, and therefore it is easy to quickly comprehend the programming flow as well as the complex code logic. The comments are helpful in providing an understanding of the protocol's overall architecture.

Dependencies

As per our observation, the libraries used in this EWX Integration with Asset Hubs infrastructure are based on well-known industry standard open source projects.
Apart from libraries, its functions are used in external EWX Integration with Asset Hub calls.

Severity Definitions

The severity levels assigned to findings represent a comprehensive evaluation of both their potential impact and the likelihood of occurrence within the system. These categorizations are established based on Hashlock's professional standards and expertise, incorporating both industry best practices and our discretion as security auditors. This ensures a tailored assessment that reflects the specific context and risk profile of each finding.

Significance	Description
High	High-severity vulnerabilities can result in loss of funds, asset loss, access denial, and other critical issues that will result in the direct loss of funds and control by the owners and community.
Medium	Medium-level difficulties should be solved before deployment, but won't result in loss of funds.
Low	Low-level vulnerabilities are areas that lack best practices that may cause small complications in the future.
Gas	Gas Optimisations, issues, and inefficiencies.
QA	Quality Assurance (QA) findings are informational and don't impact functionality. Supports clients improve the clarity, maintainability, or overall structure of the code.

Status Definitions

Each identified security finding is assigned a status that reflects its current stage of remediation or acknowledgment. The status provides clarity on the handling of the issue and ensures transparency in the auditing process. The statuses are as follows:

Significance	Description
Resolved	The identified vulnerability has been fully mitigated either through the implementation of the recommended solution proposed by Hashlock or through an alternative client-provided solution that demonstrably addresses the issue.
Acknowledged	The client has formally recognized the vulnerability but has chosen not to address it due to the high cost or complexity of remediation. This status is acceptable for medium and low-severity findings after internal review and agreement. However, all high-severity findings must be resolved without exception.
Unresolved	The finding remains neither remediated nor formally acknowledged by the client, leaving the vulnerability unaddressed.

Audit Findings

High

[H-01] runtime/stable/src/xcm_config.rs#to_asset_balance - Logic error causes asset conversions to fail

Description

The `to_asset_balance` function contains a critical logic error in its validation check that causes all foreign asset balance conversions to fail:

```
// Since chain ED is 0 we set a reference value for the native ED.

const REF_NATIVE_ED: Balance = 10_000_000_000_000_000;

// Ensure we don't divide by zero

ensure!(!REF_NATIVE_ED != 0, DispatchError::Other("Mock ED is zero"));
```

The condition `!REF_NATIVE_ED != 0` uses incorrect double-negative logic. Given that `REF_NATIVE_ED` is hardcoded as `10_000_000_000_000_000`, this evaluates as:

- `REF_NATIVE_ED != 0` → true (since `10^16 != 0`)
- `!true` → `false`
- `ensure!(false, DispatchError)` → always fails

Impact

This means every call to `to_asset_balance` will fail with "`Mock ED` is zero" error, completely breaking foreign asset functionality in `XCM` operations.

Recommendation

Fix the logic error by removing the negation operator:

```
// Ensure we don't divide by zero

ensure!(REF_NATIVE_ED != 0, DispatchError::Other("Reference ED cannot be zero"));
```

Alternatively, since `REF_NATIVE_ED` is a hardcoded non-zero constant, this validation could be removed entirely.

Status

Resolved

Low

[L-01] runtime/stable/src/xcm_config.rs#to_asset_balance - Potential precision loss due to division before multiplication

Description

The asset balance conversion function performs division before multiplication, which can cause precision loss if `asset_min_balance < REF_NATIVE_ED`:

```
Ok(FixedU128::saturating_from_rational(asset_min_balance, REF_NATIVE_ED)
    .saturating_mul_int(balance))
```

The current implementation uses the following equation: `(asset_min_balance / REF_NATIVE_ED) * balance`. If `asset_min_balance` is smaller than `REF_NATIVE_ED` (`10^16`), the division creates a fractional ratio that loses precision.

Impact

The asset balance may become zero if a rounding error occurred, potentially resulting in incorrect fee calculations.

Recommendation

Modify the implementation to perform multiplication before division.

Status

Acknowledged

[L-02] runtime/stable/src/weights/xcm/pallet_xcm_benchmarks_fungible.rs - Outdated benchmark weights

Description

The `XCM` benchmark weights are significantly outdated, having been generated on February 18, 2025, but are still being used in the current runtime deployment:

```
//! THIS FILE WAS AUTO-GENERATED USING THE SUBSTRATE BENCHMARK CLI VERSION 4.0.0-dev
//! DATE: 2025-02-18, STEPS: `50`, REPEAT: `20`, LOW RANGE: `[]`, HIGH RANGE: `[]`
//! WORST CASE MAP SIZE: `1000000`
//! HOSTNAME: `ip-172-31-46-227`, CPU: `AMD EPYC 7R32`
//! WASM-EXECUTION: Compiled, CHAIN: Some("dev"), DB CACHE: 1024
```

Impact

This is problematic because the benchmarks from February 2025 may not reflect current runtime performance.

Recommendation

Re-run `XCM` benchmarks before any production deployment.

Status

Resolved

Centralisation

The Energy Web X Liquid Staking project is moving toward full decentralization by having many independent validators make all key decisions instead of a single team. A temporary admin role is only in place during upgrades, after which governance will be fully community-driven.

Conclusion

After Hashlock's analysis, the Energy Web project seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved and acknowledged. Overall, most of the code is correctly ordered and follows industry best practices. The code is well commented on as well. To the best of our ability, Hashlock is not able to identify any further vulnerabilities.

Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits is to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

Manual Code Review:

In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behaviour when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our methodologies include manual code analysis, user interface interaction, and white box penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high-level understanding of what functionality the EWX Integration with Asset Hub under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When a potential issue is discovered, we immediately create an issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contract details are made public.

Disclaimers

Hashlock's Disclaimer

Hashlock's team has analysed these EWX Integration with Asset Hubs in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the EWX Integration with Asset Hub source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this EWX Integration with Asset Hub.

Hashlock is not responsible for the safety of any funds and is not in any way liable for the security of the project.

Technical Disclaimer

EWX Integration with Asset Hubs are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the EWX Integration with Asset Hub can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee the explicit security of the audited EWX Integration with Asset Hubs.

About Hashlock

Hashlock is an Australian-based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3-oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

Website: hashlock.com.au
Contact: info@hashlock.com.au