hashlock.

Security Audit

The Energy Web Foundation (Blockchain)

July 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 Foundation team partnered with Hashlock to conduct a security audit of their smart contracts. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

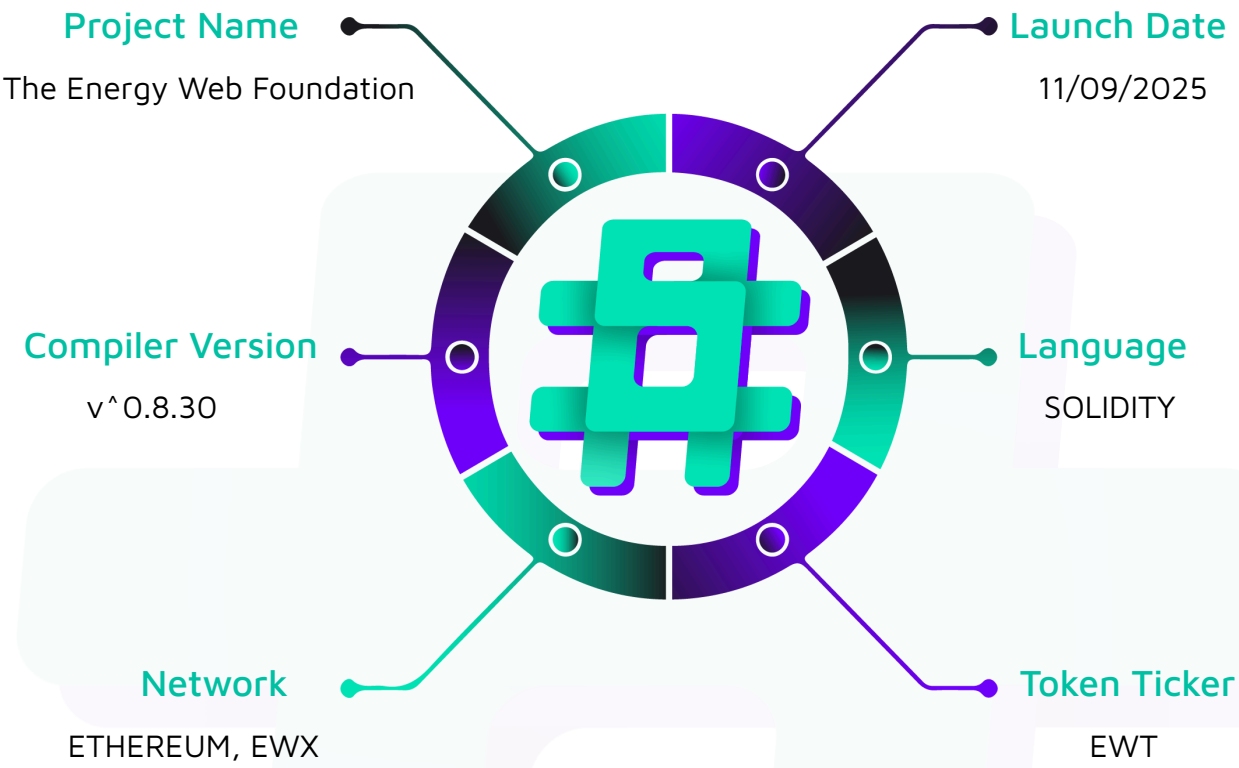
Project Context

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.

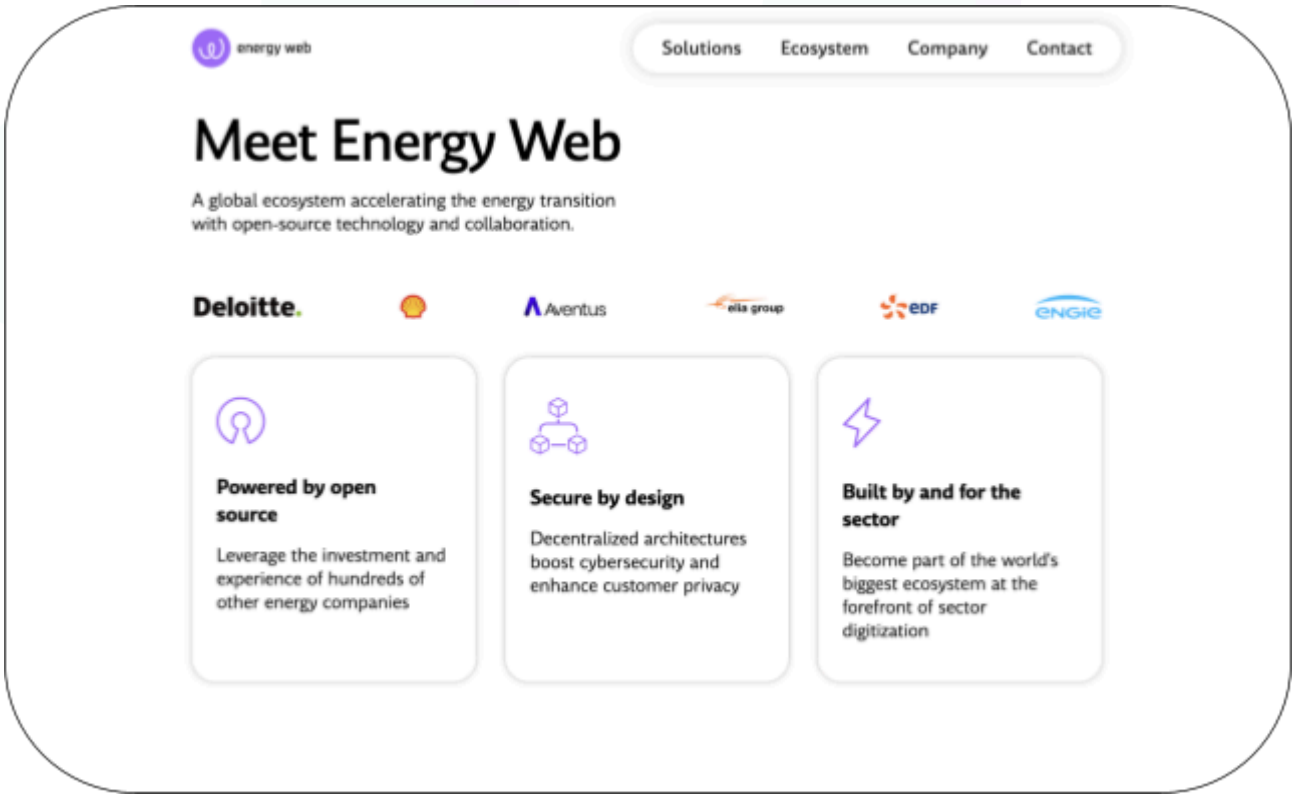
EWX introduces a permissionless Proof-of-Stake consensus model, enabling broad validator and delegator participation while unlocking staking rewards for participants and supporting a robust on-chain economy. To expand liquidity and interoperability, the Energy Web Token (EWT) is transitioning into a fully compliant ERC-20 token on Ethereum mainnet, supported by a dual bridge architecture: a bidirectional bridge between Ethereum and EWX as well as continued support for lifting from EWC to EWX.

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

Visualised Context:



Project Visuals:



Audit Scope

We at Hashlock audited the solidity code within The Energy Web Foundation project, the scope of work included a comprehensive review of the smart contracts listed below. We tested the smart contracts 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	**The Energy Web Foundation Smart Contracts**
Platform	**Ethereum / Solidity**
Audit Date	**July, 2025**
Contract 1	EnergyBridge.sol
Contract 2	EnergyWebToken.sol
Audited GitHub Commit Hash	f8acf620fd2ccae3274b556b4693cd5bb98ad700
Fix Review GitHub Commit Hash	ab920c3fc89780f179a6f4e2fd7c663493fa4cac

Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Hashlocked"**. The contracts all follow simple logic, with correct and detailed ordering. They use a series of interfaces, and the protocol uses a list of Open Zeppelin contracts.

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 Smart Contract Functions section.

All vulnerabilities initially identified have now been resolved and acknowledged.

Hashlock found:

1 High severity vulnerability

1 Medium severity vulnerability

4 Low severity vulnerabilities

1 Gas Optimisations

1 QA

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 Smart Contract Functions

Claimed Behaviour	Actual Behaviour
EnergyBridge.sol - bridging between EWX and Ethereum - author consensus (add/remove) - trigger periodic inflation of EWT - lift and lower ERC20 tokens - UUPS proxy upgradeable via EIP-1822	**Contract achieves this functionality.**
EnergyWebToken.sol - upgradeable ERC20 token with ERC-2612 permit - The owner can set the bridge and create an initial supply - bridge-only mint and burn	**Contract achieves this functionality.**

Code Quality

This audit scope involves the smart contracts of The Energy Web Foundation 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 was 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 Foundation project smart contract 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 smart contracts infrastructure are based on well-known industry standard open source projects.
Apart from libraries, its functions are used in external smart contract 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] EnergyBridge#_requiredConfirmations - Insufficient consensus threshold allows minority control over critical operations

Description

The `_requiredConfirmations` function implements a threshold calculation that results in an incorrect consensus requirement for small author sets, falling short of Byzantine fault tolerance standards, which require greater than 66.7% agreement.

Vulnerability Details

The function calculates required confirmations as `N - floor(2N/3)`, which produces inconsistent and insufficient thresholds:

- 4 authors: requires 2 signatures (50%)
- 5 authors: requires 2 signatures (40%)
- 6 authors: requires 2 signatures (33.3%)
- 7 authors: requires 3 signatures (42.8%)

This allows a minority of authors to control critical operations, including adding/removing authors, publishing roots, and triggering token inflation.

```
function _requiredConfirmations() private view returns (uint256 required) {

    required = numActiveAuthors;

    unchecked {

      required -= (required * 2) / 3;

    }

  }
```

Impact

A malicious minority can compromise the bridge's security model by approving unauthorized state transitions, adding malicious authors, or triggering unintended token minting.

Recommendation

Implement proper super-majority calculation that ensures at least `floor(2N/3) + 1` signatures are required.

Status

Resolved

Medium

[M-01] EnergyBridge#setGrowthRate - Unbounded loops when iterating over assets from the asset manager could cause out of gas error

Description

The `setGrowthRate` function permits rates up to exactly 10,000 basis points (100%), allowing the entire token supply to be doubled in a single `triggerGrowth` call.

Vulnerability Details

The validation uses `>` instead of `>=`:

```
function setGrowthRate(uint16 newRate) public onlyOwner {
    if (newRate > BASIS_POINTS) revert RateOutsideRange();
    growthRate = newRate;
    emit LogGrowthRateUpdated(newRate);
  }
```

This allows `growthRate = 10,000`, which in `triggerGrowth` calculates to:

```
amount = totalSupply * 10,000 / 10,000 = totalSupply
```

Impact

Admin can instantly double the token supply, potentially destabilizing the token economy and diluting holder value.

Recommendation

We recommend changing the validation to prevent 100% inflation.

Status

Resolved

Low

[L-01] EnergyBridge#removeAuthor - Dynamic threshold reduction enables author removal with insufficient signatures

Description

The `removeAuthor` function decrements `numActiveAuthors` before verifying the removal proof, allowing the operation to complete with fewer signatures than initially required. Flow:

1. Sets `isAuthor[id] = false`
2. Decrements `numActiveAuthors` ifthe author was active
3. Calls `_verifyConfirmations`, which uses the already reduced count

For example, with 7 authors where 3 signatures are required, the threshold drops to 2 signatures mid-operation after decreasing to 6 authors.

Malicious authors can exploit this vulnerability to remove legitimate authors with fewer approvals than the protocol intended, gaining unauthorized control.

Recommendation

We recommend verifying confirmations before modifying the state.

Status

Resolved

[L-02] EnergyBridge#claimLower - Zero address recipient validation missing enables permanent token loss

Description

The `claimLower` function does not validate that the recipient address extracted from the proof is non-zero before executing the token transfer. The function calls `_extractLowerData` to parse the recipient from raw bytes at offset 52-72, then directly transfers tokens via `safeTransfer(recipient, amount)` without any address validation, allowing permanent token loss if authors accidentally sign proofs with zero addresses.

Recommendation

We recommend adding recipient validation before the token transfer.

Status

Resolved

[L-03] EnergyBridge#_domainSeparator - Static EIP-712 version allows signature replay across upgrades

Description

The domain separator uses hardcoded `VERSION_HASH = keccak256('1')` while the contract's `version` state variable increments on each upgrade via `_authorizeUpgrade`.

This inconsistency means EIP-712 signatures remain valid across all contract versions, even when proof structures or validation logic changes, as the domain separator hash remains constant despite the contract implementation evolving.

Recommendation

We recommend dynamically incorporating the current contract version into the domain separator calculation.

Status

Resolved

[L-04] EnergyBridge#checkLower - Perpetual validity of lower proofs prevents time-boxed withdrawals

Description

Lower proofs lack any expiry parameter or timestamp validation, remaining valid indefinitely once signed by authors. Unlike other bridge operations (`addAuthor`, `removeAuthor`, `publishRoot`, `triggerGrowth`) that include an `expiry` parameter validated by the `withinCallWindow` modifier, lower proofs can be claimed at any time in the future, preventing implementation of temporary withdrawal windows or proof expiration policies.

Recommendation

We recommend adding an expiry field to the `LOWER_DATA_TYPEHASH` and validating it in `claimLower`.

Status

Acknowledged

[L-05] EnergyBridge#addAuthor - Missing T2 public key validation allows invalid registrations

Description

The contract accepts any `bytes32` value as a T2 public key without validating format, non-zero status, or cryptographic validity. While T1 public keys are validated to be exactly 64 bytes and properly derive to the expected address, T2 keys undergo no validation beyond checking for duplicate registration, allowing registration of malformed keys that cannot function on the T2 network.

Recommendation

We recommend adding basic T2 key validation.

Status

Resolved

Gas

[G-01] EnergyBridge#_domainSeparator - Repeated computation wastes gas

Description

The `_domainSeparator` function recalculates the complete EIP-712 domain separator on every proof verification by encoding and hashing constant values (`DOMAIN_TYPEHASH`, contract name, version, chain ID, and address).

Since these values remain constant between upgrades (except potentially chain ID in rare fork scenarios), this repeated computation wastes approximately 3,000-5,000 gas per verification across all proof-based operations.

Recommendation

We recommend caching the domain separator as a state variable during initialization.

Status

Acknowledged

QA

[Q-01] EnergyBridge#triggerGrowth - Unused signed parameters create

Description

The `triggerGrowth` function requires authors to sign a proof containing `rewards`, `avgStaked`, and `period` parameters via the `TRIGGER_GROWTH_TYPEHASH`, but only uses `period` in the actual implementation. The inflation calculation completely ignores the signed `rewards` and `avgStaked` values, instead computing `amount = (IERC20(EWT).totalSupply() * growthRate) / BASIS_POINTS`, creating a confusing mismatch between the secured proof data and execution logic that could lead to integration errors.

Recommendation

We recommend removing unused parameters from the function signature and proof structure as indicated by the TODO comment.

Status

Acknowledged

Centralisation

The Energy Web Foundation 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 Foundation 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 smart contract 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 smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the smart contract 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 smart contract.

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

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

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