hashlock.

2024

# Security Audit

## Coinsub (DeFi)

August 2024

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 THAT 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 THE USE OF THE CLIENT.

# Executive Summary

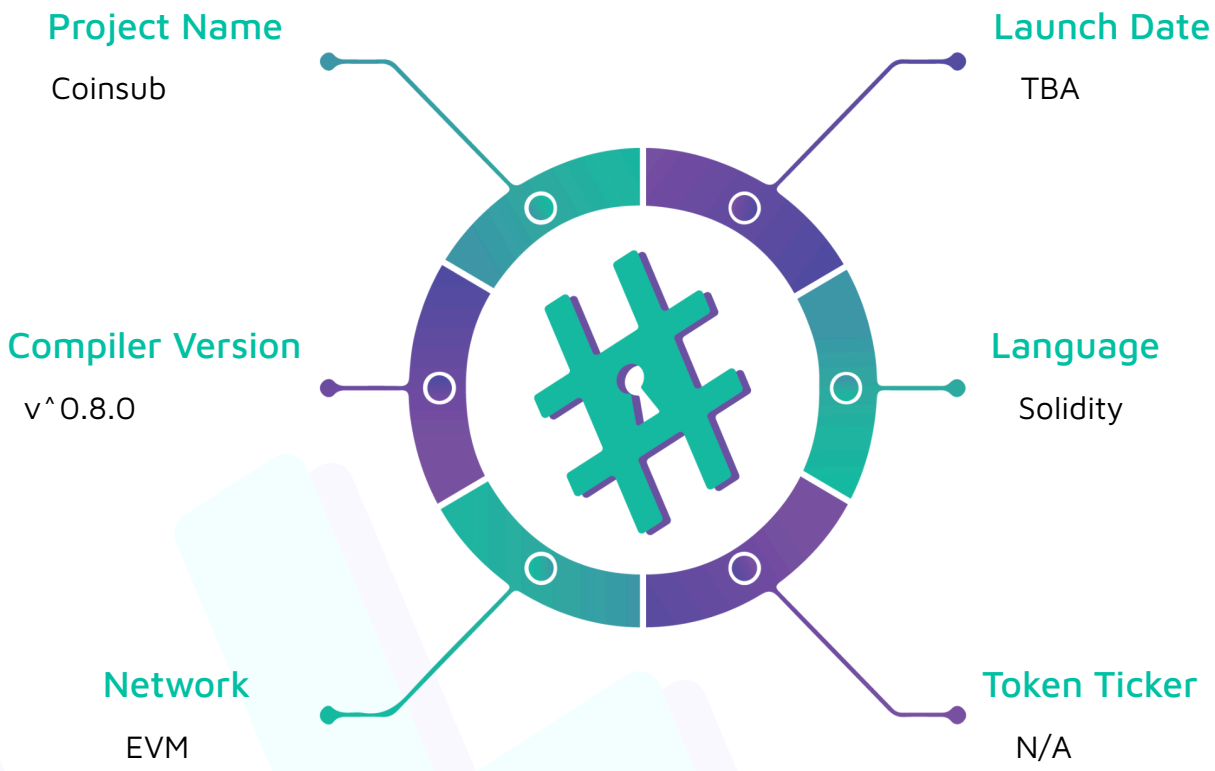
The Coinsub team partnered with Hashlock to conduct a security audit of their CoinSubPayer.sol and TokenTable.sol smart contracts. Hashlock manually and proactively reviewed the code to ensure the project's team and community that the deployed contracts were secure.

# Project Context

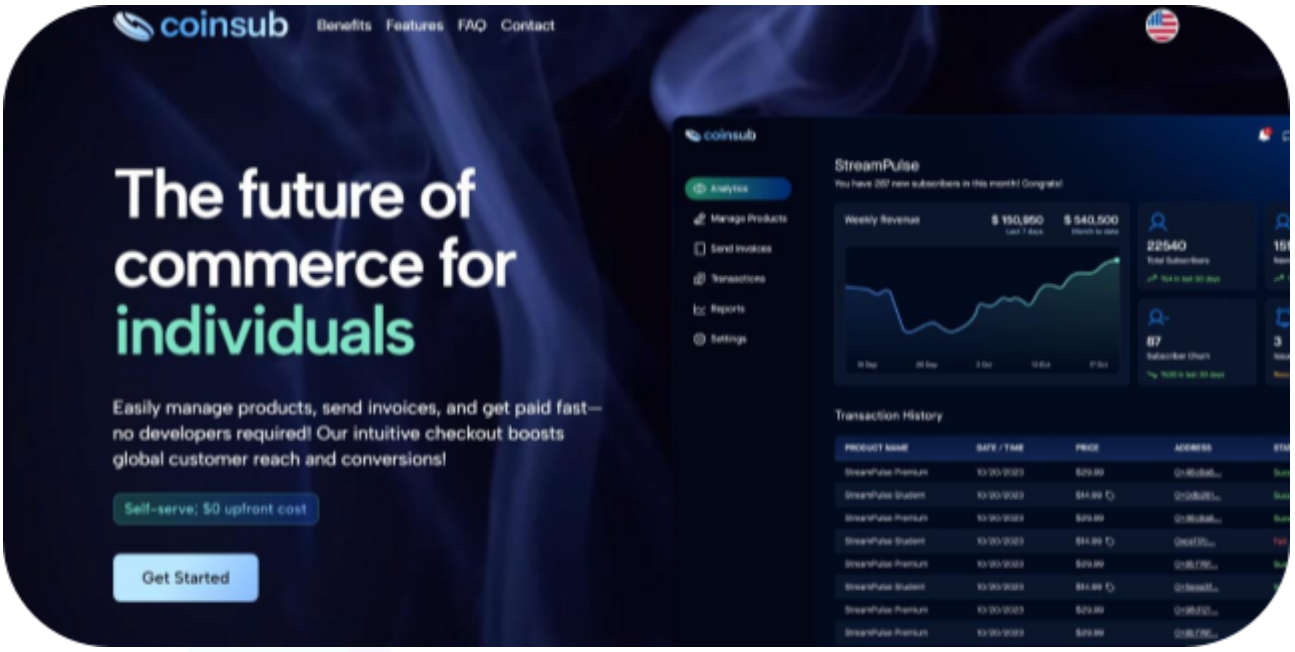
Coinsub is a forward-thinking two-sided platform that creates easy-to-use commerce solutions for both merchants and customers to take advantage of the benefits of blockchain technology.

**Project Name**: Coinsub
**Compiler Version**: ^0.8.0
**Website**: www.coinsub.io
**Logo**:

**Visualised Context:**



**Project Visuals:**




# Audit scope

We at Hashlock audited the solidity code within the Coinsub 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** | **Coinsub Smart Contracts** |
|---|---|
| **Platform** | **EVM / Solidity** |
| **Audit Date** | **August, 2024** |
| **Contract 1** | CoinSubPayer.sol |
| **Contract 1 MD5 Hash** | 33e7043dd2bd28fe7818496a5619f3f4 |
| **Contract 2** | TokenTable.sol |
| **Contract 2 MD5 Hash** | 635c7e817984146a75bbedf01d022eaa |

# Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Secure"**. 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.

We initially identified some significant vulnerabilities that have since been addressed.

**Hashlock found:**

1 Low-severity vulnerabilities

2 QA

1 Gas Optimisations

**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 Behaviours

| Claimed Behaviour | Actual Behaviour |
| --- | --- |
| **CoinSubPayer.sol**<br>- Subscription Management:<br>  - Manages periodic payment subscriptions using signed messages.<br>  - Allows creation, processing, and cancellation of subscriptions.<br>  - Subscriptions can be paid with ERC-20 tokens.<br>- Fees:<br>  - Supports both flat fees (in cents) and percentage-based fees on transactions.<br>  - Allows updating of these fees, with constraints to ensure they don't exceed specified limits.<br>- Signature Verification:<br>  - Validates that the subscription creation request is genuine by checking the provided signature against the subscriber's address.<br>- Bulk Operations:<br>  - Supports bulk creation, payment processing, and cancellation of subscriptions to optimise gas usage.<br>- Authorization Management:<br>  - Allows the contract owner to authorise other addresses to manage subscriptions and perform other administrative tasks.<br>- Pausable: | **Contract achieves this functionality.** |

| | |
|---|---|
| - Includes mechanisms to pause and unpause contract operations, ensuring flexibility in managing the contract's state.<br>- Token Handling:<br>  - Utilises the TokenTable contract to resolve token names to addresses.<br>  - Ensures that sufficient funds and allowances are available before processing a payment.<br>- Withdrawal:<br>  - Allows the owner or authorised addresses to withdraw tokens or native cryptocurrency held by the contract.<br>- Events:<br>  - Emits events to log significant actions like subscription creation, cancellation, payment processing, fee updates, and authorization changes.<br>  - This contract facilitates recurring payments using cryptocurrency, providing mechanisms for secure and flexible subscription management. | |
| **TokenTable.sol**<br>- Allows owner to:<br>  - Set the token address per a token name | **Contract achieves this functionality.** |

# Code Quality

This Audit scope involves the smart contracts of the Coinsub project, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices to help avoid unnecessary complexity that increases the likelihood of exploitation, however, some refactoring was required.

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 Coinsub projects smart contract code in the form of the contract files.

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 help understand the overall architecture of the protocol.

# Dependencies

Per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry-standard open-source projects.

# Severity Definitions

| 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 |

# Audit Findings

## Low

### [L-01] CoinSubPayer#withdrawNative - Deprecated Ether transfer function

**Description**

The Istanbul update made some changes provided to the EVM, which made the `.transfer()` function deprecated for the ETH transfer.

However, the `withdrawNative()` function uses `.transfer()` to transfer the native coins.

**Recommendation**

Use `.call()` instead of `.transfer()` to transfer the native coins.

**Status**

Resolved

# QA

## [Q-01] CoinSubPayer#changeTokenTable, TokenTable#setTokenAddress - Lack of emitting events

**Description**

The `changeTokenTable()` and `setTokenAddress()` functions are missing events when key storages are updated.

**Recommendation**

Add relevant events based on the variables to be updated.

**Status**

Resolved

## [Q-02] CoinSubPayer - Unused imports

**Description**

The `ECDSA` and `console` imports are not used at all in the contract.

**Recommendation**

Remove unused imports.

**Status**

Resolved

# Gas

## [G-01] CoinSubPayer - Sub struct could be packed

### Description

The `Sub` struct uses 8 slots but it could be packed by moving the `isActive` attribute to the first or second attribute position.

```
struct Sub {
    address token;
    address to;
    uint256 amount;
    uint256 durationInDays;
    uint256 nextPaymentDate;
    uint256 numberOfPayments;
    uint256 paymentCount;
    bool isActive
}
```

### Recommendation

Move the `isActive` attribute.

### Status

Resolved

# Centralisation

The Coinsub project values security and utility over decentralisation.

The owner executable functions within the protocol increase security and functionality but depend highly on internal team responsibility.

# Conclusion

After Hashlocks analysis, the Coinsub project seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved. 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 behavior 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 still need to verify the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown not to 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 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


#Hashlock.