hashlock.

2024

Security Audit

Soarchain (Staking)

December 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 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

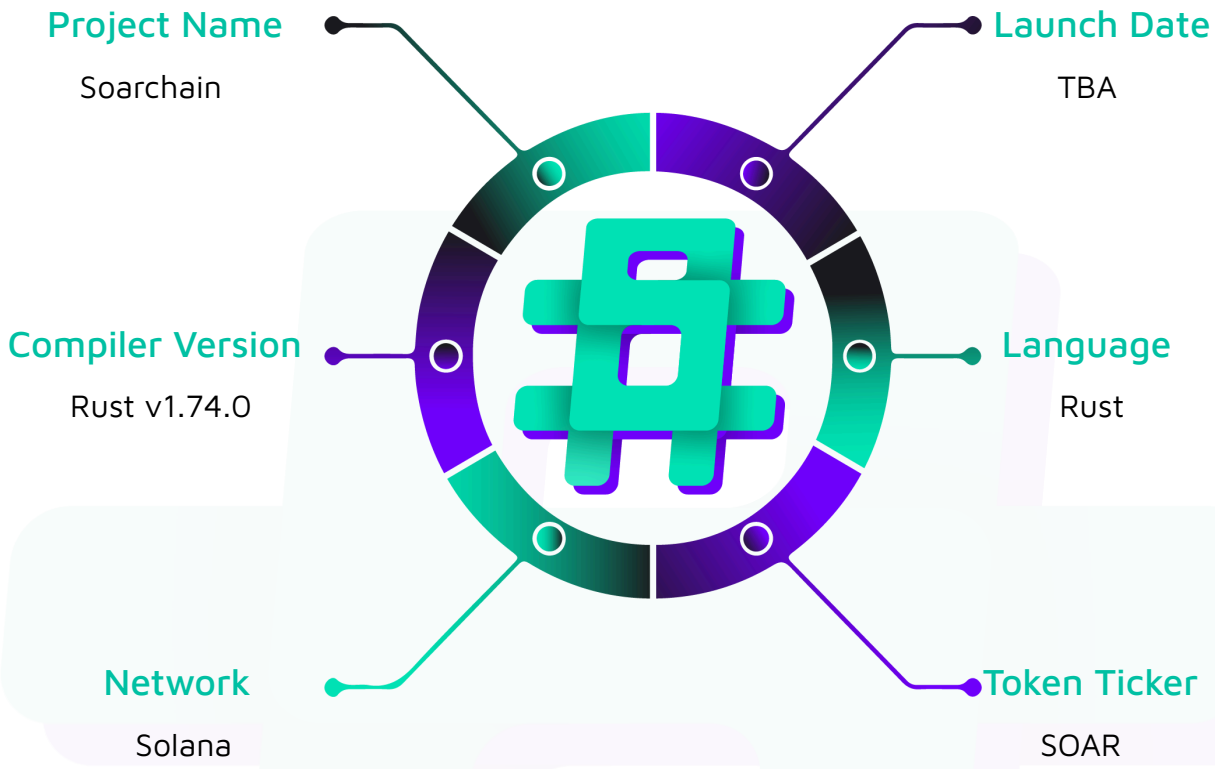
The Soarchain team partnered with Hashlock to conduct a security audit of their soarchain-rewards and soarchain-staking smart contracts. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

Project Context

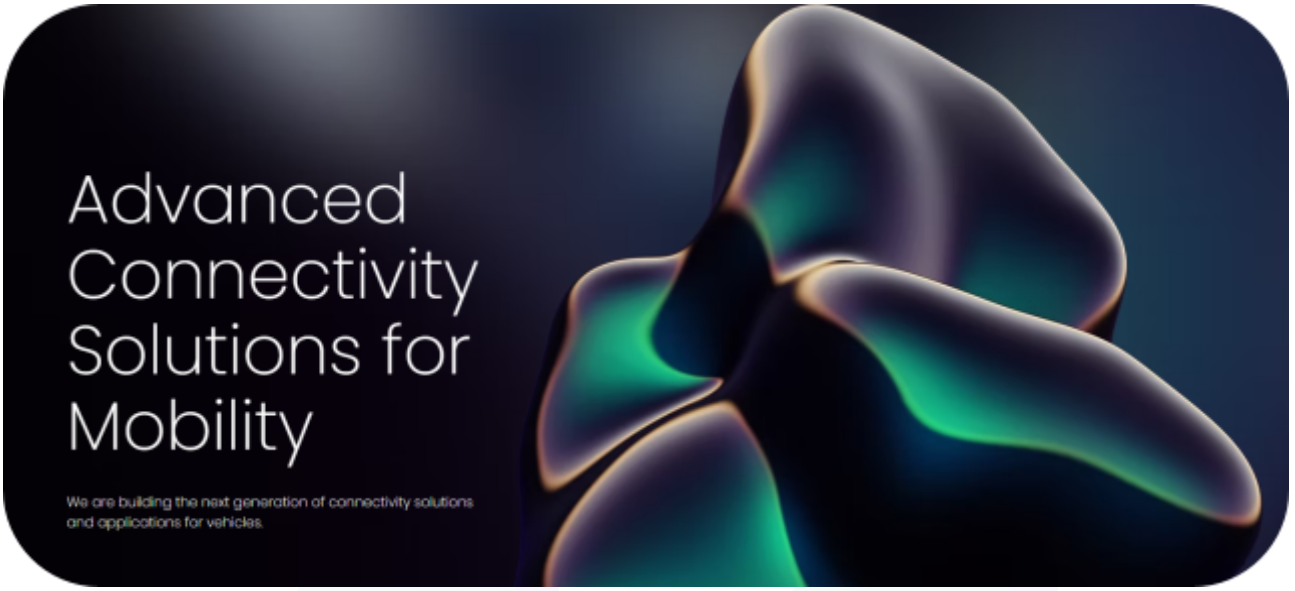
Soar Dev Labs is redefining mobility with a Web3-powered platform that integrates advanced Vehicle-to-Everything (V2X) connectivity and decentralized AI networks. Their Soarchain stack enables developers to build secure, interoperable mobility applications, leveraging blockchain for trust and simulation tools for real-world scenario modeling and staking SOAR, their token. By bridging vehicles, infrastructure, and digital ecosystems, Soar Dev Labs is shaping an intelligent, self-sustaining mobility future.

Project Name: Soarchain
Compiler Version: Rust v1.74.0
Website: https://www.soardevlab.com/
Logo:

Visualised Context:



Project Visuals:



Services

We offer services to streamline the creation of Vehicle-to-Everything (V2X) applications and use case development on the Soarchain stack.

Development

We help you implement your custom applications for proprietary use cases.

Simulation

We use hyperrealistic visual and physical simulation environments to safely test your use cases by replicating real-life scenarios in virtual worlds.

Integration

We assist you with integrating applications, use cases, and connectivity solutions into your vehicle designs.

Audit scope

We at Hashlock audited the Rust code within the Soarchain 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	Soarchain Smart Contracts
Platform	Solana / Rust
Audit Date	December, 2024
GitHub Repository	https://github.com/Soar-Development/soarchain-program-library
Component 1	programs/soarchain-rewards/*
Component 2	programs/soarchain-staking/*
GitHub Commit Hash	0f6986f8373e439b52b1b1f185fa7726e60c86c5

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

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 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 vulnerabilities

1 Low severity vulnerabilities

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
programs/soarchain-rewards/* - Allows users to: - Create reward accounts - Add funds into the vault account - Claim rewards - Sync rewards with the latest reflection account parameters - Close reward accounts	**Contract achieves this functionality.**
programs/soarchain-staking/* - Allows users to: - Create stake and vault accounts - Initiate unbonding - Cancel their unbonding request - Increase stake amount - Extend staking duration - Close stake accounts - Withdraw funds from vault accounts - Allows guardians to: - Slash user accounts - Update settings	**Contract achieves this functionality.**

Code Quality

This audit scope involves the smart contracts of the Soarchain 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 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 Soarchain 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

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 purely informational and don't impact functionality. These notes help clients improve the clarity, maintainability, or overall structure of the code, ensuring a cleaner and more efficient project. They should be addressed for optimization but are not critical to the system's performance or security.

Audit Findings

High

[H-01] claim.rs, enter.rs, sync.rs - The stake account can be supplied with fake PDAs to steal rewards

Description

The `StakeAccount` account supplied is not validated to be the intended PDA, allowing attackers to create and provide a malicious stake account with arbitrary parameters.

Vulnerability Details

The `Claim`, `Enter`, and `Sync` instructions do not validate that the provided `StakeAccount` is the intended PDA account by enforcing the `seeds` constraint. This validation is important because attackers may pass PDAs with arbitrary parameters to trigger unintended behaviors in the program.

Impact

In this case, the `StakeAccount.xsoar` parameter can be set by an attacker to a high value in order to steal rewards from the vault account. This causes a loss of funds when other legitimate users want to claim their rewards.

Recommendation

Consider enforcing the `seeds` constraint in `StakeAccount` to prevent attackers from passing incorrect PDAs. This can be achieved by applying the `soarchain_staking` function validation from `common/src/pda.rs` to the `Claim`, `Enter`, and `Sync` instructions.

Status

Resolved

Low

[L-01] cancel_unbond.rs - Incorrect error message

Description

When validating the `StakeAccount`, a `constraint` with `stake.time_unbond != 0` condition is enforced. If this validation fails, a `SoarchainStakingError::AlreadyStaked` will be returned.

This is incorrect because the error message should be `SoarchainStakingError::NotUnbonded`, similar to `Close` and `Withdraw` instructions.

Recommendation

Consider updating the error to use `SoarchainStakingError::NotUnbonded`.

Status

Resolved

QA

[Q-01] state.rs, close.rs - Unused account and field

Description

The `user` account must be specified in the `Close` instruction, but the account is never used throughout the execution. The `RewardAccount.bump` field is initialized in the `Enter` instruction, but its value has never been used throughout the program.

Recommendation

Consider removing the unused account and field.

Status

Acknowledged

Centralisation

The Soarchain 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 Hashlock's analysis, the Soarchain 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