hashlock.

2024

Security Audit

Solauto (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

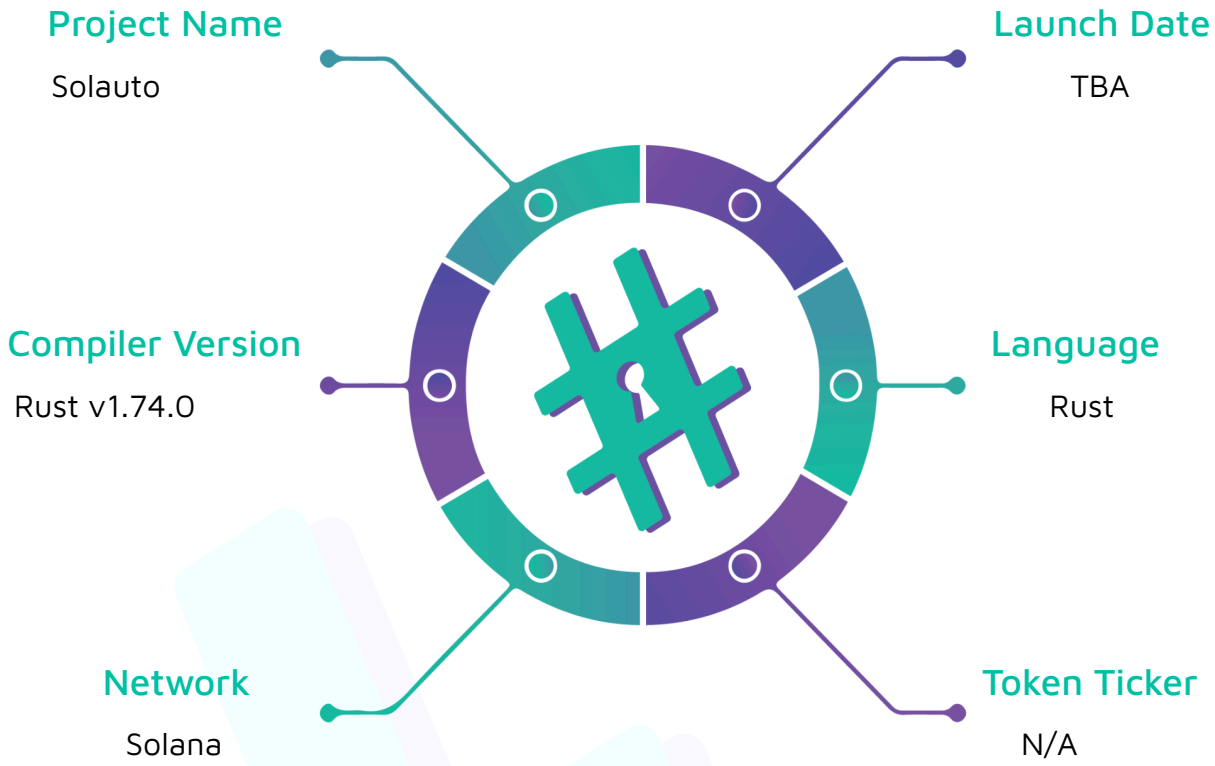
The Solauto team partnered with Hashlock to conduct a security audit of their Solauto smart contracts. Hashlock manually and proactively reviewed the code to ensure the project's team and community that the deployed contracts were secure.

Project Context

Solauto is a program built on the Solana blockchain that allows users to manage leveraged positions on auto-pilot to maximize their profit without any liquidation risks. Solauto currently integrates with MarginFi's mrgnlend system, which is a lending and borrowing protocol.

Project Name: Solauto
Compiler Version: Rust v1.74.0
Website: N/A
Logo: N/A

Visualised Context:



Audit scope

We at Hashlock audited the Rust code within the Solauto 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	**Solauto Smart Contracts**
Platform	**Solana / Rust**
Audit Date	**August 2024**
GitHub Repository	https://github.com/haven-fi/solauto
Component	programs/solauto/*.rs
GitHub Commit Hash	734f7ee99574603ef24edb332c2a89fda5d402b1

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.

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:

2 High-severity vulnerabilities

1 Medium-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 Smart Contract Behaviours

Claimed Behaviour	Actual Behaviour

programs/solauto/*.rs
- Allows users to:
  - Update referral states
  - Convert referral fees
  - Claim referral fees
  - Open new positions
  - Update existing positions
  - Close positions
  - Cancel active DCA
  - Refresh position data
  - Interact with MarginFi through Solauto position
  - Rebalance positions
	Program achieves this functionality.

Code Quality

This audit scope involves the smart contracts for the Solauto 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 Solauto 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 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.
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

Audit Findings

High

[H-01] referral_state#process_claim_referral_fees - Missing authentication validation allows attackers to steal referral fees

Description

When claiming referral fees, no validation ensures the caller is the referral state authority, allowing attackers to steal victims' referral fees.

Vulnerability Details

The `process_claim_referral_fees` function in `programs/solauto/src/processors/referral_state.rs:172` does not validate that the transaction signer (`ctx.accounts.signer.key`) equals the referral state authority (`referral_state.data.authority`). This validation is important because it ensures only the owner of the referral state can withdraw their referral fees.

Impact

Attackers can steal referral fees from a victim, causing a loss of funds.

Recommendation

Consider validating that the signer equals the referral state's authority.

Status

Resolved

[H-02] solauto_utils#create_new_solauto_position - Existing positions will be removed when opening a new position

Description

If a user calls the `MarginfiOpenPosition` instruction with an existing position, the position will be removed.

Vulnerability Details

The `create_new_solauto_position` function in `programs/solauto/src/utils/solauto_utils.rs:85-92` sets the `SolautoPosition` to a new `PositionData` and `PositionState` if the caller specified `UpdatePositionData.setting_params` as `None`. If there is any existing position information in `solauto_position`, it will be overwritten to an empty position in `programs/solauto/src/instructions/open_position.rs:112`.

Impact

The user will lose access to their existing position, including any deposited funds.

Recommendation

Consider checking the position has no existing data with the `account_has_data` function before overwriting it.

Status

Resolved

Medium

[M-01] refresh#marginfi_refresh_accounts - Unvalidated PDA may cause the user's position to be updated incorrectly

Description

The PDA accounts are not validated to be the expected account when users refresh their position, potentially allowing users to manipulate their position.

Vulnerability Details

The `marginfi_refresh_accounts` function in `programs/solauto/src/instructions/refresh.rs:13` does not validate the `supply_price_oracle` and `debt_price_oracle` accounts to be the expected PDA. Validating the PDA to be correct is essential because the PDA values will be used to refresh and update the user's position. This may happen when the user calls the `MarginfiRefreshData` instruction.

Consequently, users can provide malicious supply and debt price oracles to inflate the position's supply, debt, and total net worth values.

Impact

Users' positions will be updated to an incorrect state. This may cause unintended consequences when interacting with MarginFi or rebalancing the position. For example, the `MarginfiProtocolInteraction` or `MarginfiRebalance` instructions may fail to work correctly, causing a denial of service.

Recommendation

Consider validating that the `supply_price_oracle` and `debt_price_oracle` accounts are the correct PDA accounts.

Status

Acknowledged

Low

[L-01] solauto_utils#cancel_dca_in_if_necessary - Duplicate validation

Description

The if statement in `programs/solauto/src/utils/solauto_utils.rs:264` is entered if the `dca_in` function and `solauto_position.data.position.dca.debt_to_add_base_unit > 0` are both evaluated as true. This statement is duplicated because the `dca_in` function performs the same validation (`self.debt_to_add_base_unit > 0`) in `programs/solauto/src/state/solauto_position.rs:205-207`.

Consequently, the same validation is performed twice, which is inefficient as it consumes unnecessary gas.

Recommendation

Consider removing the `dca_in` function condition requirement.

Status

Resolved

[L-02] Programs - Usage of magic numbers

Description

There are several magic numbers defined in the codebase:

- Instances of 50 (e.g., `programs/solauto/src/utils/validation_utils.rs:142`)
- Instances of 10000 (e.g., `programs/solauto/src/utils/validation_utils.rs:154`)
- Instances of 10000.0 (e.g., `programs/solauto/src/utils/solauto_utils.rs:64`)

This decreases the code readability and maintainability.

Recommendation

Consider replacing the magic numbers with descriptive variable names and comments.

Status

Resolved

Centralisation

The Solauto 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 Solauto 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 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.