hashlock.



Security Audit

Project 0 (DeFi)

July 2026 info@hashlock.com.au Hashlock Pty Ltd

Table of Contents

Executive Summary	4
Project Context	4
Audit Scope	7
Security Rating	8
Intended Program Functions	9
Code Quality	11
Audit Resources	11
Dependencies	11
Severity Definitions	12
Status Definitions	13
Audit Findings	14
Centralisation	22
Conclusion	23
Our Methodology	24
Disclaimers	26
About Hashlock	27

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 Project 0 team partnered with Hashlock to conduct a security audit of their code. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

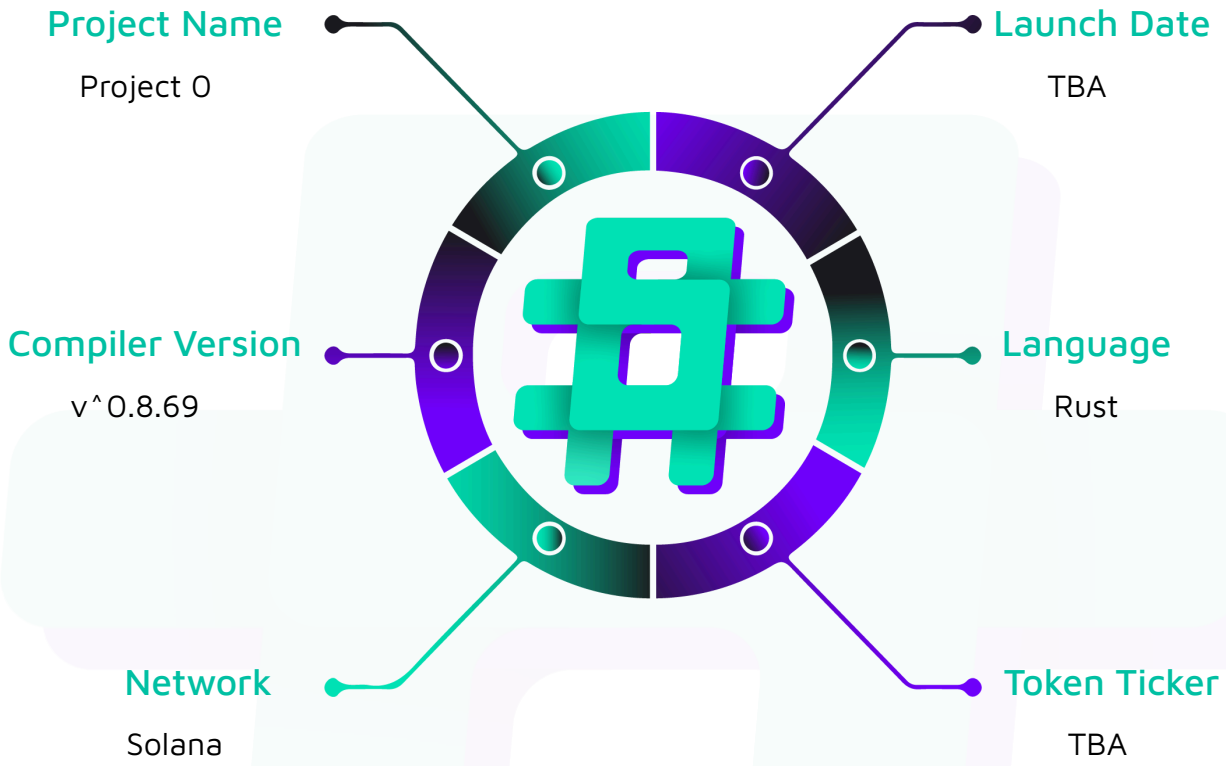
Project Context

Project 0 is a decentralized finance (DeFi) protocol built on Solana that functions as a DeFi-native prime broker, enabling users to manage borrowing, lending, leveraged trading, and portfolio margin across multiple integrated DeFi protocols through a unified risk engine. Rather than operating its own exchange or trading venue, Project 0 aggregates liquidity and positions from external protocols while providing cross-platform collateral management and capital-efficient credit, allowing users to treat their entire on-chain portfolio as a single margin account.

Project Name: Project 0
Project Type: Defi
Website: https://app.marginfi.com/
Logo:

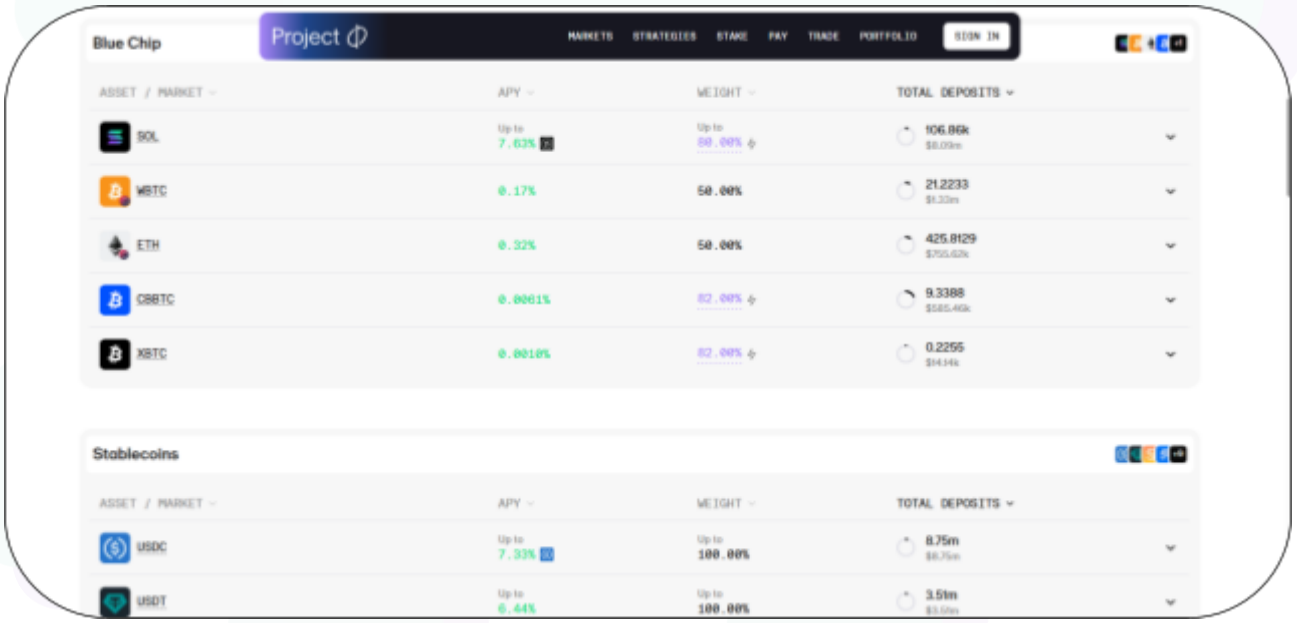


Visualised Context:



Project Visuals:





Audit Scope

We at Hashlock audited the Rust code within the Project 0 project, the scope of work included a comprehensive review of the programs and modules listed below. We tested the programs to assess their security, functionality, and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

Description	Project 0 Smart Contracts
Platform	Solana / Rust
Audit Date	July, 2026
GitHub 1	https://github.com/0dotxyz/marginfi-v2/pull/567
GitHub 2	https://github.com/0dotxyz/marginfi-v2/pull/585
Audited GitHub Commit Hash	4d57e2c234a41a868c519b722a6cce1c0431a699
Fix Review GitHub Commit Hash	1f1f7a6980954b743d98f83b1d4e77d2e0358855

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 codebase to be **"Secure"**. The code all follows simple logic, with correct and detailed ordering. They use a series of interfaces and external dependencies, including standard open-source libraries.



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 Program Functions section.

All vulnerabilities initially identified have now been resolved.

Hashlock found:

1 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 Program Functions

Claimed Behaviour	Actual Behaviour
bank.rs Tracks EMA reference price against live oracle observations, evaluates deviation against tiered thresholds, trips/escalates halts, auto-promotes to full pause on repeated tier-3 trips, freezes interest accrual during a halt.	Halt/escalation state machine works as intended. Interest freeze depends on whether `accrue_interest` happens to be called while the halt is active rather than tracking the frozen interval explicitly, causing inconsistent borrower/lender/fee accounting across halt windows.
bank_config.rs Holds circuit-breaker configuration: deviation tiers, halt durations, sustain-observation count, escalation window, EMA alpha.	**Contract achieves this functionality.**
borrow.rs Live price gate blocks borrow instructions when the involved bank's oracle price deviates from the circuit-breaker reference beyond the threshold.	**Contract achieves this functionality.**
liquidate.rs Direct liquidation should be gated by the live circuit breaker, so liquidation cannot use a breaching oracle price; restricted appropriately when a halted bank is involved.	Does not run the live price gate before consuming oracle prices or mutating balances, so a first liquidation can execute on a breaching price before the halt takes effect.
emode.rs Groups banks as same-asset eligible and applies improved (lower) risk weights only	Equivalence check only compares mint and primary oracle key, doesn't check oracle setup, secondary

where participating banks price the shared asset equivalently.	oracle/venue accounts, or multiplier/exchange-rate behavior, so banks with differing pricing semantics can still qualify.
configure.rs Configures same-asset eMode leverage limits and bank oracle settings; oracle changes should be restricted while same-asset eligibility is enabled to keep pricing semantics consistent.	Allows changing oracle setup post-eligibility as long as the primary oracle key stays the same, letting pricing semantics drift out of sync with the same-asset assumption.

Code Quality

This audit scope involves the smart contracts of the Project 0 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 Project 0 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] Marginfi_account_liquidate.rs -First Liquidation will Execute Before the Circuit Breaker Blocks a Breaching Price

Description

In PR #567, direct liquidation can execute using a breaching live oracle price before the circuit breaker records and enforces a halt, allowing liquidation on prices that should be blocked. The recommended mitigation is to run the live `cb_price_gate` before liquidation uses oracle prices or mutates balances.

Vulnerability Details

Direct liquidation only checks whether a bank is already in a stored circuit-breaker halt, `any_balance_bank_is_cb_halted` checks the stored state through `bank.is_cb_halted(now)` or `BankOperationalState::CircuitBroken`. It does not fetch the live oracle price and does not run `cb_price_gate`.

The liquidation then computes health and liquidation amounts using live oracle prices before the circuit breaker is updated

The cache update happens only after the liquidation accounting

At that point, `update_cache_price` may trip the circuit breaker, but it returns successfully and does not revert the liquidation.

A liquidator can execute the first liquidation after a breaching oracle update (manipulative prices) before the circuit breaker protection takes effect.

Impact

This can allow liquidations using prices that should have been blocked by the live circuit-breaker gate, potentially causing unfair collateral seizure, liquidation of accounts

that should not be liquidatable under accepted prices, or bad debt, depending on the manipulated price direction.

Recommendation

Run the live `cb_price_gate` before direct liquidation uses oracle prices or mutates account balances.

Status

Resolved

Medium

[M-01] State_bank.rs-Circuit Breaker Interest Freeze Depends on Transaction Timing

Description

In PR #567, circuit-breaker interest freezing depends on whether an instruction happens to call `Bank::accrue_interest` while the halt is active, creating inconsistent borrower, lender, and fee accounting. The recommended mitigation is to track the frozen circuit-breaker interval explicitly instead of relying on opportunistic accrual calls.

Vulnerability Details

During a circuit-breaker halt, `Bank::accrue_interest` skips interest accrual and moves `last_update` to the current timestamp:

```
let time_delta: u64 = (current_timestamp - self.last_update).try_into().unwrap();

if self.is_cb_halted(current_timestamp) {

    self.last_update = current_timestamp;

    return Ok(());

}
```

This makes the frozen-interest amount depend on whether any instruction calls `accrue_interest` while the halt is active. If no call happens during the halt, the first post-halt call accrues interest over the full elapsed period. If a call happens during the halt, that portion of time is permanently skipped.

As a result, halt windows can produce different borrower, lender, and fee, interest accounting depending only on unrelated transaction activity.

Impact

Borrowers may pay less interest when any transaction triggers accrual during the halt. Lenders, insurance fees, group fees, and protocol fees receive inconsistent revenue for the same market conditions.

This creates non-deterministic economic behavior around circuit-breaker events and makes the intended "interest freeze" policy dependent on incidental instruction timing.

Recommendation

Track the circuit-breaker frozen interval explicitly instead of relying on opportunistic `accrue_interest` calls.

Status

Resolved

Low

[L-01] State_emode.rs -Same-Asset E-Mode Uses Incomplete Pricing Equivalence Checks

Description

In PR #585, same-asset e-mode can apply improved weights to banks that share only partial pricing identifiers, even though their effective pricing semantics may differ. The recommended mitigation is to enforce full pricing equivalence for same-asset participants and block pricing-semantic oracle changes while same-asset eligibility is enabled.

Vulnerability Details

Same-asset e-mode assigns higher weights when an account's liabilities and collateral are treated as the same asset. This relies on a strict assumption: the involved banks must price the asset in the same way.

PR #585 checks only a partial invariant. During `health` calculation and liability reconciliation, banks are grouped by `mint` and `oracle_keys[0]`, with additional checks for eligibility, non-fixed pricing, and collateral tier. The logic does not require the full pricing configuration to match.

In particular, it does not compare `oracle_setup`, secondary oracle, or venue accounts, or multiplier/exchange-rate behavior. This leaves room for two banks to be treated as price-equivalent even if their effective pricing paths `differ`.

The oracle update path has the same gap. While same-asset eligibility is enabled, `config_bank_oracle` prevents changes to `oracle_keys[0]` but still allows changes to `oracle_setup` if the primary oracle key remains the same. This can make the bank's pricing semantics stale relative to the same-asset eligibility assumptions.

Impact

If two eligible banks share the same mint and primary oracle key but use different pricing semantics, same-asset e-mode may apply higher weights even though price risk does not fully cancel out.

This is mainly a configuration and future-integration risk. It may not be exploitable with current standard bank setups, but a future bank type, integration asset, or admin misconfiguration could silently weaken the leverage assumptions behind same-asset e-mode.

Recommendation

Enforce full pricing equivalence for same-asset e-mode participants.

At a minimum, require eligible banks in the same group to use the same `oracle_setup`, the same relevant `oracle_keys`, and equivalent multiplier or exchange-rate behavior.

Block oracle configuration changes that affect pricing semantics while same-asset eligibility is enabled. Admins should first disable same-asset eligibility, update the oracle configuration, then re-enable eligibility after validation.

Apply these checks both when enabling eligibility and when computing same-asset weights, so stale or misconfigured banks cannot receive same-asset treatment.

Status

Resolved

[L-02] Marginfi_account_borrow.rs -Circuit Breaker Price Gate Can Block User Actions During Real Volatility

Description

In PR #567, the circuit-breaker price gate can block legitimate user actions during genuine market volatility, not only during oracle manipulation. The recommended mitigation is to confirm this product and risk intent, and consider narrowly allowing clearly risk-reducing actions to bypass the live price gate where safe.

Vulnerability Details

`cb_price_gate` is a read-only guard that rejects an instruction when the live oracle price deviates from `cb_reference_price` by at least the first configured circuit-breaker threshold.

PR #567 applies this gate to borrows, risk-carrying withdrawals, and order execution.

This is a fail-safe design, but it also means legitimate users can be blocked during genuine market volatility, not only during oracle manipulation.

Impact

Users may be unable to borrow, execute orders, or withdraw while carrying liabilities when prices move beyond the circuit-breaker threshold.

This can prevent some users from adjusting or hedging positions during volatile periods. The behavior may be acceptable if the protocol prioritizes fail-safe protection over user action availability, but it should be an explicit product and risk decision.

Recommendation

Confirm and document that blocking these actions during real volatility is intended. Consider allowing clearly risk-reducing actions to bypass the live price gate, where safe. Examples may include liability repayment, collateral deposits, or withdrawals from accounts with no liabilities.

If bypasses are added, define them narrowly and test that they cannot increase protocol risk during a circuit-breaker event.

Status

Resolved

Centralisation

The Project 0 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 Project 0 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 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

