Alula Finance Audit

April 2026

HIGHLAND SECURITY

Disclaimer

This report does not constitute legal or investment advice. You understand and agree that this report relates to new and emerging technologies and that there are significant risks inherent in using such technologies that cannot be completely protected against. While this report has been prepared based on data and information that has been provided by you there are likely additional unknown risks which otherwise exist. This report is also not comprehensive in scope, excluding a number of processes critical to the correct operation of this system and subsequent changes could have unintended consequences to the product's security. It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement. This report is for informational purposes only and is provided on an "as-is" basis and you acknowledge and agree that you are making use of this report and the information contained herein at your own risk. The preparers of this report make no representations or warranties of any kind, either express or implied, regarding the information in or the use of this report and shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein. Finally, the possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims.

About Highland Security

Highland Security is a leading security company providing consulting services with more than a decade of security expertise. We are experts in AI product security, web application security, security automation, and risk management. We offer our clients tailored AI security services that can provide value at any stage of maturity or product development lifecycle.

Our goal is to establish a long lasting security partnership for our clients that continues beyond the engagement. We consider the unique business needs and situations of our clients on a case-by-case basis and adapt our approach to provide the best value. Contact us at our website or hello@highlandsecurity.io to schedule a free consultation with our team today. To keep up with our latest endeavors and research follow us on X @0xHighlandSec.

Executive Summary

Engagement Summary

Alula Finance engaged Highland Security in April 2026 to perform a re-test of fixed findings that were identified and fixed from previous testing activities and to perform an additional security assessment of their Rust Soroban smart contract lending protocol. The goal of the assessment was to identify security vulnerabilities and recommend best practices to defend against likely threats. Over the course of the engagements, the Highland Security team identified several opportunities for improvement for Alula Finance management to consider fixing or acknowledge prior to launch. If the identified risks are remediated or Accepted with compensating controls planned, and no other material code changes are made to the protocol it is our opinion that the Alula Finance protocol security adequately addresses the likely security risks for a lending protocol.

The following table summarizes the results of this assessment:

Workstream	**Critical**	**High**	**Medium**	**Low**	**Info**
Soroban Security Assessment	-	1	5	4	2
Fixed	-	1	5	4	-
Accepted			-	-	2

Background and scope

In April 2026 Highland Security performed a penetration test of Alula Finance's lending protocols' Soroban smart contract. The goal of the assessment was to evaluate the Alula Finance products against security best practices, identify vulnerabilities, provide recommendations for remediation, and provide a professional opinion of the in scope product components security. This engagement was conducted from April 10th - 20th and included interviews with Alula Finance's personnel on their expected product operations and architecture, manual review of source code, dynamic testing of the product's programs and contracts Specifically we performed the following:

Name	Description	Scope
Audit Retest	A verification of the fixes performed as part of a previous audit to validate the issues identified has been fixed	• pointgroup-labs/alula • Branch: fix: halborn_audit_remeidations • Commit: 4a4153a665e93645532dc6785232f976953f1b1e
Soroban Security Assessment	A security assessment of the alula lending protocol to identify vulnerabilities and provide recommendations to address the risk	• pointgroup-labs/alula • Branch: Fix: halborn_audit_remeidations • Commit: 4a4153a665e93645532dc6785232f976953f1b1e
Patch Review	A review of the proposed fixes for the findings identified during the Soroban Security Assessment to confirm the risk from identified security issues are adequately addressed.	• pointgroup-labs/alula • Branch: fix/halborn_audit_reccomendations • Commit: ff215c2c2b507f2477114f4b9798abd8f5ab9eb5 • Branch: fix/halborn_audit_reccomendations • 945b40e7ea5a9781d82aa0d8a35677f5cf8113cd

Approach

The following was our approach to execute this engagement:

- *Kick off & planning* - During this phase the team worked with project sponsors to identify key stakeholders for the assessment. Documentation and access to source code and environments was also provisioned as part of this phase. We also performed interviews with key stakeholders to understand how the product works, key product features, and the team's approach to security.

- *Fieldwork* - Once the relevant details for the engagement was collected the fieldwork fieldwork activities started with a documentation, source code, and environment review to familiarize the team with the product functionality and create an efficient assessment approach. We then performed manual and dynamic testing of the in scope components to identify vulnerabilities and security misconfigurations.

- *Reporting & verbal briefings* - After the fieldwork was completed, the Highland Security team compiled the information gathered into a draft report document and reviewed the output with project stakeholders. This was an iterative process with feedback from the team being incorporated into a final deliverable.

Positives Practices

- **Untrusted Input Validation:** During the testing the Highland Security team observed strict checking of function input received from untrusted sources. Each parameter was validated against a range of possible values to ensure only expected inputs were handled.

Findings Overview

- **Bad Debt Handling**: We identified a time bound edge case where the bad debt lock of a market can expire without the bad debt being settled. If this edge case is realized, users can exit their liquidity positions immediately after the bad debt lock deadline, even if insurance coverage claims are still pending or unresolved. The consequence is that users can avoid their share of losses, shifting the liability entirely to the remaining liquidity providers, which breaks the protocol's expected

state exclusivity during bad-debt events. Management Accepted this issue but retained the time-based lock expiration as a feature to prevent indefinite fund lock-ups by an administrator.

- **Integration Risks**: Several findings highlighted inconsistency state enforcement and dependencies on external consent that could weaken core controls. Examples include inconsistencies in the protocol's global paused state and the ability to take market flash loans when paused and missing consent mechanisms (e.g. signatures) for specifying flash loans receivers. Management has Accepted these inconsistencies and updated the protocol to have both code paths for flash loans be enabled while a global pause is underway and provided guidance and reference template for integrations to explicitly check for authentication in flash loan call backs.
- **Fee Accounting Inconsistencies**: Finally, we identified some areas of improvement to handle the accounting of protocol fees. Specifically, the Highland Security team identified edge cases when changing beneficiaries that should trigger an accounting update prior to the fee beneficiaries being updated and an edge case if no beneficiaries are set. Management agreed with these findings and updated the account logic.

Technical Information

Overview

The below risk ratings were mutually agreed upon for use between Alula Finance and Highland Security at the start of this assessment:

Composite Risk Rating		
Rating	*Description*	*Count*
Critical	The identified exposure would lead to immediate loss of assets and have significant business impact. The issue is directly exploitable with little complexity and causes an immediate impact to the confidentiality, integrity, or availability of the system.	0
High	The identified exposure could lead to a loss of assets and have a business impact. The identified issues may be directly exploitable and would impact the confidentiality, integrity, or availability of the system for one or many users	1
Medium	The identified exposure is not directly exploitable but combined with other issues it may allow for exploitation of the system. The identified issue would impact the confidentiality, integrity, or availability of the system for a singular user.	5
Low	The identified exposure is not directly exploitable but unnecessarily increases the attack surface of the system. The identified issue does not directly impact the confidentiality, integrity, or availability of the system but could be used with additional findings to increase the likelihood of success.	4

The following table summarizes the results of this assessment:

Workstream	**Critical**	**High**	**Medium**	**Low**	**Info**
Soroban Security Assessment	0	1	5	4	2
Fixed	-	1	5	4	-

Withdrawal Operations Possible with Bad Debt
When bad debt is issued through the Recorded coverage path, the pool is locked by setting bad_debt_lock_d and incrementing bad_debt_request_count. However, withdrawal/deposit gating only checks timestamp >= bad_debt_lock_d, not whether unresolved requests still exist. As a result, at the exact deadline (and after), users can withdraw while insurance coverage remains pending and bad_debt_request_count > 0, creating a state overlap between "claim unresolved" and "pool mutable." See test case test_withdraw_unblocked_at_deadline_while_request_still_pending
Affected Location
• contracts/market/src/processors.rs:837-842 (sets bad_debt_lock_d, increments request count) • contracts/market/src/pool.rs:551-561 (unlock condition uses only timestamp >=) • contracts/market/src/processors.rs:551-566 and 263-276 (withdraw/deposit rely on require_bad_debt_unlocked) • contracts/market/src/processors.rs:949-960 (claim settlement still waiting on CoverageStatus::Ready) • Supporting lifecycle context: contracts/controlled_insurance_fund/src/storage.rs:33-36, contracts/controlled_insurance_fund/src/lib.rs:63-70, 142-170
RIsk Rating:
High
Impact
Users could exit liquidity positions before bad-deb socialization finalizes, avoiding losses and shifting to remaining suppliers. This breaks state exclusivity around bad-debt handling and could become more probable if insurance fund operations are manual.
Status: Fixed
Recommendation:
Make unlock contingent on request resolution state, not time alone: • Require bad_debt_request_count = 0 for deposit/withdrawal eligibility for LPs to exit the

pool
- Treat deadline as an optional minimum wait, not a sufficient unlock condition by itself
- Enforce strict single-state transitions for critical protocol components. For example, if a pool has unresolved bad debt all protocol accounting is immutable until claims is finalized / cancelled

Management Response:

Alula Management has refactored the bad debt lifecycle to require all bad debt requests (bad_deb_request_count = 0) to be settled prior to allowing pool operations to resume. Key changes to the bad debt lifecycle include:

• Withdraw Gate Enhancement: Pools now remain locked until ALL pending bad debt requests are resolved (not just until the deadline expires)

• Automatic Request Cleanup: Added permissionless cancellation for expired pending requests to prevent permanent lock states

• Per-Request Deadlines: Each bad debt request now tracks its own expiry to prevent newer requests from extending older ones

• Missing Request Handling: Graceful handling of requests that disappear from the insurance fund storage

Batched FlashBorrow bypasses global market freeze protection

We noted that batched Request::FlashBorrow in submit_requests_batch takes a separate path from process_flash_borrow that does not check global frozen status. This creates a mismatch in capabilities where batched flash loan liquidity remains usable while normal unbatched flash loan is paused while the market is frozen.

See case test_batch_flash_borrow_bypasses_frozen_market_check

Affected Location

- contracts/market/src/contract.rs:L791 (direct flash_loan path enforces freeze check)
- contracts/market/src/processors.rs:L112 (Batch routing to Request::FlashBorrow)
- contracts/market/src/processors.rs:L527 (process_flash_borrow missing require_market_not_frozen)

RIsk Rating:

Medium

Impact

Attackers can continue executing flash-borrow-powered batch strategies during freeze windows, including swaps and liquidations, undermining emergency controls. This weakens incident containment and allows value extraction while users/operators reasonably expect protocol activity to be halted.

Status: Fixed

Recommendation:

Enforce a single freeze policy across all flash borrow paths (all market paths) when a market is frozen:

- Add `require_market_not_frozen` helper inside process_flash_borrow
- Keep behavior consistent across operations that are semantically the same (ie flash_loan)

Management Response:

Having flash borrows available even when the market is `Frozen` is something Alula would like to have enabled while the market is frozen. We plan to leverage flash borrows during liquidations via the `submit_requests_batch` endpoint and liquidations are possible in a `Frozen` market. We removed the 'frozen' check from the ERC3156 flash loan implementation to ensure consistent behavior while the market is paused.

Steal Fees from users via Flash_Loan

Market.flash_loan allows any authenticated caller to initiate a flash loan against an arbitrary receiver contract without protocol-level receiver consent.

The function authenticates only the provided caller, then process_flash_loan transfers funds to contract, invokes exec_op, and unconditionally attempts transfer_from for amount + fee from that receiver. If the receiver approves repayment in its callback and does not enforce strict initiator authorization, a third party can repeatedly trigger flash loans that force the receiver to pay fees.

Preconditions Required for Exploitation

1. flash_loan is externally callable and does not verify caller authorization relative to the receiver.
2. A victim receiver contract address can be supplied as a contract.
3. The receiver implements exec_op and lacks robust initiator authorization checks.
4. The receiver grants Market approval to pull repayment (amount + fee) during callback

(or equivalent allowance exists).

5. Receiver has sufficient token balance to cover fee(s).
6. The pool has flash loans enabled and enough available liquidity.
7. Flash-loan fees are non-zero for economic drain impact.

Affected Location

- contracts/market/src/contract.rs:L277

RIsk Rating:

Medium

Impact

Unauthorized third parties can force flash loans against vulnerable receivers. The Receiver pays flash-loan fee each successful forced invocation and repeated calls can drain receiver balances over time (griefing/value extraction). This could be compounded with other operations to cause additional damage. For example in a compromised-admin scenario, an attacker could raise flash-loan fees very high and then repeatedly trigger flash loans against vulnerable receiver contracts to maximize fee drain. There are some practical constraints to minimize the risk of this scenario such as the Admin must wait the queue delay before applying (update.

Status: Fixed

Recommendation:

Consider the one or more of the following mitigation approaches:

- Enforce receiver consent at protocol level before executing flash loan.
- Bind authorization to receiver and initiator, e.g. require receiver-signed auth over (initiator, pool, amount, nonce, expiry) and verify on-chain.
- Include initiator in the callback interface so the receiver can validate who initiated the loan.
- Add market-side allowlist/permissions model for approved initiators per receiver.

Management Response:

Alula management decided to require market registration on the callback side before using our ERC-3156 flash loans preventing unauthorized or unexpected flash loans for the reciever.

Inconsistent Upgrade Pattern

The Market contract requires two step upgrade + timelock however other critical components such as oracles and swap routers do not require the same upgrade pattern.

Affected Location

- contract/aggregated-oracle/src/contract.rs:L106
- contract/soroswap_swap_provider/src/lib.rs:L49

RIsk Rating:

Medium

Impact

A single point of failure exists in the protocol which could allow adversaries to upgrade critical components, such as oracles, to malicious contracts without the time lock and other upgrade features for market upgrades.

Status: Fixed

Recommendation:

Management should consider implementing contract upgrades consistently throughout the protocol. Alternatively consider making the sub components such as oracles and routers immutable once deployed.

Management Response:

Removed upgradabilities from both the aggregated oracle and swap adapters contracts.

Fee accounting with None Beneficiaries

distribute_pool_fees is permissionless and clears fee accumulators even when no beneficiaries are configured.

In that case, no token transfers are executed, but accounting is still mutated as if distribution happened.

Affected Location

- contracts/market/src/contract.rs:L810 (entrypoint)
- contracts/market/src/contract.rs:L1090 - 1122 (fees cleared even when beneficiaries are none)
- contract/market/src/pool.rs:L755 (default config with beneficiaries set to None)

RIsk Rating:

Medium

Impact

Protocol fees can be erased from accounting without payout and total_available can be reduced with no token transfer.

Status: Fixed

Recommendation:

Require beneficiaries to be set before distribution, otherwise fail/revert.

Management Response:

Enforced by mutating the 'pool.total_available' and 'pool.take_rate_fees_sum' only when take_rate_beneficiaries are present.

Fee Accounting when Switching Beneficiaries

When set_take_rate_fees_beneficiaries (or operation-fee equivalent) is called, the contract tries to distribute fees before switching beneficiaries. However, that distribution path does not accrue interest first. Since take-rate fees are only materialized during accrual, any fees earned since the last accrual remain latent during the pre-update distribution and are later materialized under the new beneficiary set. This results in value earned during the old beneficiary period that can be paid to new beneficiaries.

See test_set_take_rate_beneficiaries_can_shift_unaccrued_past_fees_to_new_beneficiary

Affected Location

- contracts/market/src/contract.rs::630-655 (pre update distribution of fees)
- contracts/market/src/processor.rs:L1090 (distribution path missing accrual)
- contracts/market/src/interest_rate.rs:L54-61 (take-rate fees are realized only during accrual)

RIsk Rating:

Medium

Impact

If the beneficiaries are economically different entities, Misattribution of protocol revenue across beneficiary epochs and earned fees can be redirected.

Status: Fixed

Recommendation:

Force accrual before any fee distribution done as part of beneficiary updates by "settling the old epoch" before applying new beneficiaries

Management Response:

Started accruing interest before distributing fees in the process of updating the beneficiaries

No LTV Gap Enforced in Code

The protocol does not enforce a positive separation between borrow and liquidation thresholds Markets can be configured with open_ltv_bps == close_ltv_bps, enabling immediate liquidation eligibility at/near borrow boundary due to rounding and oracle movement.

Trigger Preconditions:

Admin config sets open_ltv_bps == close_ltv_bps;

The user borrows near maximum healthy amount.

Affected Location

- contract/market/src/pool.rs:L1000-L1002

RIsk Rating:

Low

Impact

Instant liquidations are possible if an admin misconfigures a market or becomes compromised and acts maliciously

Status: Fixed

Recommendation:

Enforce a positive separation between open_ltv_bps and close_ltv_bps when opening or configuring new markets

Management Response:

Added LTV Gap enforcement to the non-zero closeLTV scenario

Zero BPS Liquidation Incentives Allowed

Liquidation incentive is allowed to be zero at config level. Trustless liquidation can become unprofitable, especially for smaller positions, increasing bad-debt accumulation risk.

Affected Location

- market/src/pool.rs:L1012-L1013

RIsk Rating:

Low

Impact

Liquidations for positions will become unprofitable potentially resulting in bad debt for the protocol

Status: Fixed

Recommendation:

Enforce a minimum threshold of BPS incentive provided to liquidators to incentivize liquidations and prevent bad debt from accruing from small positions

Management Response:

Started enforcing 0.5% minimum 'max_liquidation_incentive_bps' value

Oracle Price Truncation Downward

In extreme scenarios Borrowers can obtain more debt than intended collateral constraints when oracle prices are truncated downward before market health/bound calculations. In the Oracle implementation checked_div truncates prices downward. Market borrow bound uses oracle price in the denominator; smaller price increases computed max borrow amount.

Preconditions:

Market oracle is configured to one of the affected adapters/aggregators (aggregated oracle, redstone, or soroswap);

price normalization path uses division with remainder;

user borrows assets whose effective debt price is downward-truncated.

See truncated_oracle_price_increases_market_max_borrow_capacity test case

Affected Location

- redstone_sep_40_adapter/src/contract.rs:L90-L95
- soroswap_sep_40_adapter/src/swap.rs:L27-L30
- aggregated-oracle/src/computations.rs:L173-L181

RIsk Rating:

Low

Impact

Borrows can obtain more debt than intended with collateral constraints in extreme scenarios

Status: Fixed

Recommendation:

We should explicitly round in favor of the protocol and make it context aware instead of generic truncations:

- For prices used to value debt / borrow limits / health checks, protocol should round up
- For prices used to value collateral, round down

Management Response:

The current solution rounds down the amount of borrowed/withdrawn assets a user can take from the protocol, which is a better approach.
Note: `compute_max_healthy_collateral_removed_amount` truncates the max healthy amount that can be borrowed/withdrawn/removed(as plain collateral), which is the intended behavior. The user receives less, and the protocol does not lose due to precision issues. We've preserved the current behavior but made a few mathematical rearrangements so that the borrower/withdrawer/plain collateral remover loses less (still, the market doesn't lose at all).

Aggregated Oracle overwrites freshness timestamp

AggregatedOracleContract::lastprice() returns a median price but overwrites the timestamp with current ledger time, instead of propagating source oracle observation time. This creates "fresh-looking" prices even when underlying data is older (but still within aggregator max_age), which can defeat stricter downstream stale checks.

See aggregated_oracle test_lastprice_timestamp_laundering_can_bypass_strict_downstream_stale_check

Affected Location

- contracts/aggregated-oracle/src/contract.rs:l217-219

RIsk Rating:

Low

Impact

Downstream consumers may accept stale prices as fresh since the aggregated oracle overwrites the source oracle timestamp which could have unintended consequences during periods of volatility

Status: Fixed

Recommendation:

- Do not stamp aggregated output with `now` as the sole freshness signal
- Return provenance-preserving timestamp metadata, for example use the oldest contributing source timestamp or induce both the aggregated_at and source_observed_at timestamps

Management Response:

Fixed. Started using the oldest source timestamp for a median price timestamp

State/Accounting is finalized before token transfer settlement
The market updates obligation and pool state using the requested transfer amount before token settlement is validated. If a token is fee-on-transfer, rebasing, or hook-modified, the contract may receive less than expected while still crediting full internal value.
Affected Location
• contracts/market/src/request.rs:L193 (core transfer execution, no balance check) • contract/market/src/processors.rs:L293 (state updated before transfer settlement) • contract/market/src/processor.rs:L765 (same logic in liquidation)
RIsk Rating:
Info
Impact
If non standard transfer tokens (fee on transfer, rebasing etc) protocol accounting can become corrupted.
Status: Accepted
Recommendation:
Ensure non standard tokens are prohibited by the protocol (ie never deployed by admin) or change the contract logic to use balance delta changes instead of the requested amount in accounting operations like the insurance fund logic
Management Response:
For the near future, we have no plans to work with such a type of assets, but we'll consider something if we are there

Missing Deadline Protection in DEX Operations

The soroswap swap provider hardcodes the deadline parameter to u64:MAX effectively making the deadline infinite. This could have unintended consequences if a swap request is not immediately filled or canceled

Affected Location

- contracts/market/src/request.rs:L193 (core transfer execution, no balance check)
- contract/market/src/processors.rs:L293 (state updated before transfer settlement)
- contract/market/src/processor.rs:L765 (same logic in liquidation)

RIsk Rating:

Info

Impact

User swaps can execute much later than intended under changed market conditions, increasing adverse execution risk.

Status: Accepted

Recommendation:

Consider updating the hardcoded value to be user supplied or some reasonable alternative such as 1 hour.

Management Response:

This is the only way to call `swap` endpoints from another contract that doesn't expect a 'deadline' parameter. Using something like `e.ledger().timestamp() + 10` as a deadline parameter causes the invocation to be rejected by the validators during execution, since the timestamp differs between simulation and execution. The only option for integrating contracts that expect a `deadline` parameter is to use a constant value, and `u64::MAX` is a descriptive choice. We'll add comments regarding this in the upcoming remediation commit

Appendix

Appendix A: Halborn Audit Retest Results

Halborn ID	Name	Risk	Report Status	Highland Status
7.1	Share inflation via donation enables repeated value capture in new pools	Critical	Solved	Addressed
7.2	Distributed fees do not reduce available liquidity accounting	Critical	Solved	Addressed
7.3	Inconsistent asset value scaling across core protocol operations	Critical	Solved	Addressed
7.4	Unrestricted external control of market operational state	Critical	Solved	Addressed
7.5	Referrer fees	Critical	Solved	Addressed

	charged but never distributed			
7.6	Liquidation repayments desynchronize pool liquidity accounting	Critical	Solved	Addressed
7.7	Unauthorized fee draining from flash loan receivers	High	Partially Solved	See finding above
7.8	Zero oracle price is accepted as valid and propagated to valuation logic	High	Solved	Addressed
7.9	Spot reserve price can be manipulated and used as an oracle output	High	Accepted	Open / Accepted
7.10	Pool supply limit can be bypassed via donation	Medium	Solved	Addressed
7.11	Global protocol state can be irreversibly erased by an administrative action	Medium	Solved	Addressed
7.12	Liquidity	Medium	Solved	Addressed

	fragmentation through multiple pools for the same asset			
7.13	Multiply pair configuration allows identical underlying assets on both sides	Medium	Solved	Addressed
7.14	Reserved fee liquidity can be treated as withdrawable or lendable liquidity	Medium	Solved	Addressed
7.15	Swap execution does not allow user defined minimum output protection	Medium	Solved	Addressed
7.16	Leveraged deposit can misaccount borrow and referrer fees	Medium	Solved	Addressed
7.17	Repay fee can prevent full debt closure in leveraged withdrawal flow	Medium	Solved	Addressed
7.18	Leveraged withdrawal can	Medium	Solved	Addressed

	consume tokens already accounted as withdraw fees			
7.19	Small collateral deposits can be forcibly removed through bad debt coverage	Medium	Solved	Addressed
7.20	Missing validation of transfer amounts before execution	Low	Accepted	See finding above
7.21	Centralized market upgrades blocked by misaligned authorization	Low	Solved	Addressed
7.22	Unbounded bootstrap configuration can degrade interest accrual and block pool operations	Low	Solved	Addressed
7.23	Overlapping bootstrap periods alter liquidity release behavior	Low	Solved	Addressed
7.24	Hardcoded external router address	Low	Solved	Addressed

	reduces integration safety and upgrade flexibility			
7.25	Liquidations can revert due to negative accrued interest accounting	Low	Solved	Addressed
7.26	Swap allows identical input and output tokens	Low	Solved	Addressed
7.27	Rounding effects can produce zero token side effects despite positive input amounts	Low	Solved	Addressed
7.28	Zero amount operations trigger unnecessary state changes	Informational	Solved	Addressed
7.29	Unbounded scarcity fee configuration can block withdrawals	Informational	Solved	Addressed
7.30	Simulation path may diverge from actual withdrawal behavior	Informational	Solved	Addressed

7.31	State changing operations execute without emitting events	Informational	Solved	Addressed
7.32	Unused functions	Informational	Solved	Addressed