Adevar Labs

Prepared for:
Loopscale

August 15, 2025

Audit Report

Version: after fix review

Table of Contents

1. Executive Summary

About Loopscale

The Loopscale protocol is an order book-based lending platform on Solana.

The system uses direct order book matching instead of pooled liquidity to improve capital efficiency and risk management, featuring isolated markets, fixed-rate terms, and specialized primitives like Vaults and Loops for enhanced user experience.

Audit Summary

- Number of Findings: 21 total
- Critical: 0
- High: 5
- Medium: 7
- Low: 9
- Number of Enhancement Opportunities: 10

Risk Profile

The following table summarizes the distribution of identified vulnerabilities by risk level:

Risk Level	Count	Fixed	Acknowledged
Critical	0	-	-
High	5	5	0
Medium	7	7	0
Low	9	6	3

Overall Security Posture

Before Fix Review

The Loopscale team has demonstrated a solid understanding and experience in DeFi through their ability to architect and implement complex lending mechanisms. Most of the core instructions show good implementation, laying a positive foundation.

In addition to areas where logic patterns could be improved for consistency and accuracy, our review also uncovered a few significant issues.

Finally, while the current test suite provides a good foundation, enhancing its coverage, particularly for newer features and edge cases, would be beneficial.

Three specific areas warrant additional focus to further strengthen the protocol:

- **The recently introduced loan-lock feature**: This innovative addition, combined with the borrow-caps feature, expands the protocol's capabilities. Careful attention is needed to ensure these mechanisms interact seamlessly and don't inadvertently create unintended arbitrage opportunities.
- **Switchboard oracle integration**: Validating the feed parameters and statistical calculations within the Switchboard oracle integration is important to maintain the integrity of price feeds.
- **Raydium integration component**: This component showed the highest concentration of areas for improvement and would greatly benefit from thorough testing before deployment to ensure its robustness.

After Fix Review

All High and Medium severity issues have been comprehensively resolved, demonstrating the team's commitment to security and their technical proficiency in implementing fixes.

- **loan-lock feature**: The loan-lock feature has been fixed and deployed successfully.
- **Switchboard oracle integration**: The oracle integration now implements proper statistical validation and includes the most important safety checks. The minimum number of submissions per feed is not checked internally, this responsibility will rely on the market operator when choosing a price feed.
- **Raydium integration component**: Previously the area of highest concern, the Raydium integration has undergone comprehensive fixes.

Launch Recommendations

Before Fix Review

To ensure a smooth and secure launch, we offer the following recommendations:

I. Mandatory Before Launch:

- Ensure all "High" and "Medium" issues are fixed
- Carefully review and test the implications of the loan-lock feature with newly added mechanisms
- Ensure the presence of the **bs_auth** Signer on every sensitive instruction

II. Strongly Recommended:

- Conduct comprehensive testing of the Raydium integration, including reward collection, position transfers, and tick array bitmap support.
- Consider reviewing the oracle architecture to explore opportunities for enhanced resilience against adversarial events.
- Review and compare the use of saturating arithmetic, especially in dust calculations where loss would be unacceptable
- Consider integrating additional invariant checks directly within functions

III. Post-Launch Monitoring:

- Deploy continuous fuzzing in the CI/CD pipeline
- Continuously monitor Switchboard feed health
- Track oracle manipulation attempts and unusual price movements that could indicate attacks.

After Fix Review

- "High" and "Medium" issues have been fixed.
- The **bs_auth** Signer has been implemented on missing instructions.
- Oracle architecture resilience has been significantly improved with robust validation against invalid prices and exceptional market events. We continue to recommend implementing a fallback oracle mechanism for cases where the primary market oracle becomes unresponsive.
- Arithmetic operations have been updated to use checked subtraction where precision is critical, replacing potentially unsafe saturating arithmetic in dust calculations

All other recommendations represent ongoing security best practices that should be maintained post-launch through continuous monitoring and iterative improvements.

Audit Scope

Branch feat/loan-lock

- **Repository:** https://github.com/LoopscaleLabs/loopscale-program-library
- **Files/Modules in Scope:**
- PR #1122

Branch feat/borrow-caps

- **Repository:** https://github.com/LoopscaleLabs/loopscale-program-library
- **Files/Modules in Scope:**
- PR #1087

****Branch feat/raydium-clmm**

- **Repository:** https://github.com/LoopscaleLabs/loopscale-program-library
- **Files/Modules in Scope:**
- PR #1108

Branch feat/switchboard

- **Repository:** https://github.com/LoopscaleLabs/loopscale-program-library
- **Files/Modules in Scope:**
- PR #1154

Branch feat/apl

- **Repository:** https://github.com/LoopscaleLabs/loopscale-program-library
- **Files/Modules in Scope:**
- PR #1210

2. Assumptions and Considerations

Assumptions & Considerations

Audit Assumptions

The following limitations and constraints should be considered when reviewing this security assessment report:

I. **Refinance and Liquidate do not count towards Supply and Withdraw Cap**

RefinanceLedger and **LiquidateLedger** instructions do not count towards the supply and withdraw caps.

II. **Principal Counters fully reset at the end of the cap's interval**

The **principal_*** counters reset to 0 at the end of each interval (eg. hourly), allowing potential cap exceedance. For instance, a 1-hour withdraw cap can effectively reach 2x the limit: full cap usage in the last minute of one interval, plus another full cap immediately after reset. The naming is misleading, as it doesn't reflect net principal but gross flows.

III. **Deposit and Withdraw Caps account for the overall net capital value**

In the feat/borrow-caps branch, deposit and withdraw caps (eg. per hour, per day) have been introduced to limit user actions. However, the implementation updates counters in an unexpected manner: during a deposit, the deposit counter is incremented (as expected), but the withdraw counter is decremented. Conversely, during a withdraw, the withdraw counter is incremented, but the deposit counter is decremented.

This behavior enables scenarios where caps can be effectively bypassed without altering the net asset delta. For instance, coordinated large deposits and withdrawals could result in both deposited and withdrawn amounts exceeding their respective caps, even if the overall balance remains unchanged (eg. one user deposits heavily while another withdraws equivalently). This could lead to unintended overuse of the system, potential liquidity risks, or exploitation in high-volume environments.

IV. **Borrow and Repay Cap Tracking Inflates Per-Asset Allocations**

The borrow and repay operations incorrectly track collateral asset allocations by adding/subtracting the full operation amount to each collateral asset's cap tracking, rather than splitting the amount proportionally across assets. This implementation flaw inflates the per-asset allocation counters by a factor equal to the number of collateral assets involved in the operation.

Trust Assumptions

- Market parameters can be updated by admins, but are subject to a configurable timelock
- No fallback oracles, implying that oracles are expected to always respond

3. Severity Definitions

Each issue identified in this report is assigned a severity level based on two dimensions: **Impact** and **Likelihood**. These dimensions help project our team's understanding of both the potential consequences of a vulnerability and how likely a vulnerability is to be discovered and exploited in the real world.

Impact

Impact reflects the potential consequences of the issue–particularly on **project funds**, **user funds**, and the **availability or integrity** of the protocol.

- **High Impact**: Successful exploitation could result in a complete loss of user or protocol funds, disruption of core protocol functionality, or permanent loss of control over critical components.
- **Medium Impact**: Exploitation could cause significant disruption or partial loss of funds, but not a total compromise. May impact some users or non-core functionality.
- **Low Impact**: The issue has minor or negligible consequences. It may affect edge cases, expose metadata, or degrade performance slightly without putting funds or core logic at serious risk.

Likelihood

Likelihood reflects how easy a vulnerability is to discover and exploit by an attacker, as well as how economically attractive the exploit is to an attacker.

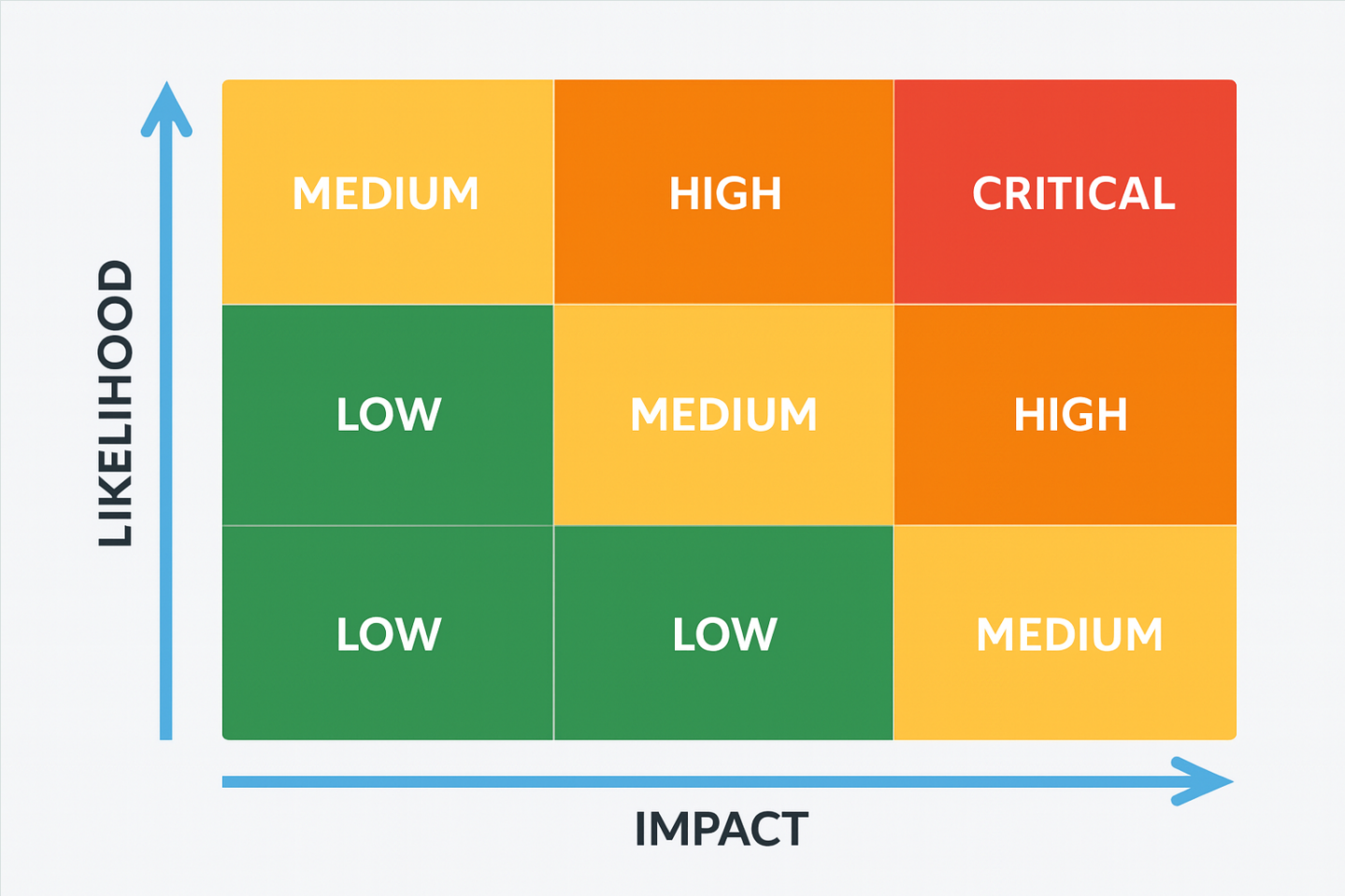
- **High Likelihood**: The vulnerability is trivially exploitable, and/or it can be exploited by a wide range of actors without privileged access rights, with minimal capital requirements, and/or low financial risks.
- **Medium Likelihood**: Can be found and exploited with moderate effort, and could require a significant capital investment with manageable financial risk.
- **Low Likelihood**: Exploitation may be impractical or theoretical due to technical constraints and could require a significant financial risk for the attacker, with a high risk of loss relative to potential gains.

Severity Classification Matrix

By combining **Impact** and **Likelihood**, we assign a severity level using the matrix below:

- **Critical**: High impact + high likelihood (e.g. a bug that could allow anyone to drain a substansial amount of protocol funds with minimal effort)
- **High**: High impact with medium likelihood, or medium impact with high likelihood
- **Medium**: Moderate impact and/or discoverability
- **Low**: Minimal impact or unlikely to be exploited

This structured approach helps teams prioritize fixes and mitigate the most dangerous threats first.



Severity Matrix

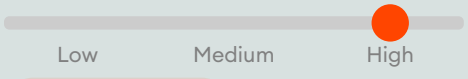
4. Findings

H01: update_single_ltv_lqt_entry Only Updates Uninitialized Entries

Status: Resolved

Impact:

Likelihood:



Severity: High

Location: Private

Description:

update_single_ltv_lqt_entry only sets LTV/LQT if orig_ltv == 0, indicating an uninitialized entry. It skips the update otherwise, which makes MarketInfoUpdate a no-op for existing entries. This issue prevents dynamic updates, leaving loans with outdated risks.

Recommendation:

Apply the updates always and check for orig_ltv == 0 only to handle initialization.

Developer Response:

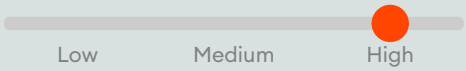
Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1218

H02: Post-filter enumeration re-indexes active items sequentially, which can corrupt matrix updates.

Status: Resolved

Impact:

Likelihood:



Severity: High

Location: Private

Description:

Post-filter enumeration re-indexes active items sequentially (e.g., skips inactive, so **ledger_index=0** might map to original index 2), which corrupts matrix updates by writing to the wrong slots.

This could zero out or incorrectly update matrix entries, leading to incorrect borrow limits or liquidations (e.g., over-borrowing if LTV overstated).

This pattern is used in several other places. Search for instances where **.enumerate()** is used after **.filter(**, and indices are used at the end. Another important place where this issue appears is in **sync_apys**.

Recommendation:

Use indexed for-loops:
[Implementation details omitted per client confidentiality requirements]

Alternatively, use **.enumerate()** before **.filter()** to preserve original indices.

Developer Response:

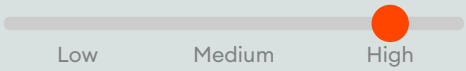
Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1234

H03: Inverted Logic When Deciding If a Loan Should Be Closed After a Collateral Withdrawal

Status: Resolved

Impact:

Likelihood:



Severity: High

Location: Private

Description:

In the **creditbook::collateral::withdraw** instruction, the logic to determine if a loan should be closed was inverted.
When the condition **params.close_if_eligible && ctx.accounts.loan.load()?.is_eligible_for_close()** was true, the loan would be updated, and when the condition was false, the loan was closed.

This would cause loans to be incorrectly closed, the user will thus lose access to their collateral:

[Implementation details omitted per client confidentiality requirements]

There is no way to recover the Loan account with the existing set of instructions, the solution would be to develop a special "recover" instruction with admin rights to re-create the deleted loan with the same values as before.

Recommendation:

Update the closing logic to match the expected behavior

Proof of Concept:

1. User has a loan with collateral in it
2. User calls **withdraw_collateral** to withdraw part of its collateral
3. **params.close_if_eligible** is set to **false** and **is_eligible_for_close** returns false
4. the **else** case is executed, which calls **ctx.accounts.loan.close(...)** which is the Anchor **close** method
5. User loses access to their collateral

Developer Response:

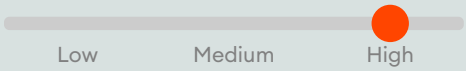
Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/commit/e1d36dc079027c53d28a5147aef0c1dce4b050e8#diff-1025214b96b8ed5fe92d8c82d52384ceea1cc63e9a14957134a26a339aaae2d6R110

H04: Users could alter the APY of a ledger by borrowing at 0% APY and instantly repaying

Status: Resolved

Impact:

Likelihood:



Severity: High

Location: Private

Description:

While in a locked loan, a user could borrow at 0% APY and reduce the ledger's blended APY even though the borrowed amount was repaid in the same transaction.

Recommendation:

Sync the APY at the end of the transaction with the correct value (the value before the locked loan borrow).

Proof of Concept:

Steps to reproduce:
1. User takes out a valid loan
2. User locks the loan, borrows at 0% APY, blended APY gets smaller
3. Returns what he borrowed, unlocks the loan

The APY remains at smaller value.

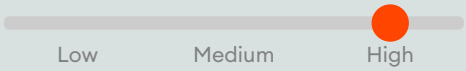
Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1192

H05: Users could borrow principal at 0% APY

Status: Resolved

Impact:



Likelihood:

Severity: High

Location: Private

Description:

The lock loan feature allows users to borrow at 0% APY as long as their loan passes the health checks when unlocking the loan. However, the implemented checks don't stop a user from borrowing at 0% APY and then adding the necessary collateral before unlocking the loan and thus allowing them to effectively borrow capital at 0% APY indefinitely.

Recommendation:

While allowing the 0% APY is intended during a locked loan, necessary checks should be made at the end in order to prevent a user from having any capital borrowed at 0% APY.

Proof of Concept:

Steps to reproduce:
1. User takes out a valid loan
2. User locks the loan, borrows at 0% APY
3. User deposits a collateral, unlocks the loan
4. Health check passes because the loan is collateralized from step3

Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1192

M01: Inconsistent Health Check Types Between Raydium and Orca Protocols

Status: Resolved

Impact:

Likelihood:

Severity: Medium

Location: Private

Description:

The Raydium liquidity management functions use inconsistent health check types compared to the Orca implementation when decreasing liquidity.

Recommendation:

Change the Raydium decrease liquidity health check to use LTV for consistency:

[Implementation details omitted per client confidentiality requirements]

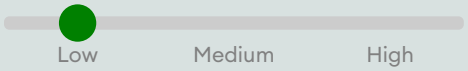
Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1222/files#diff-0bd349b7151057dd9025e6c213e73d147eade0c07fd50f63d55a38f0e03f0b70R234

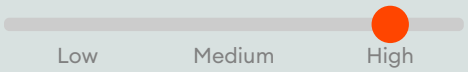
M02: Hardcoded Index 0 Breaking Orca Transfers at Other Collateral Slots

Status: Resolved

Impact:



Likelihood:



Severity: Medium

Location: Private

Description:

The **TransferOrcaPosition** and **ClaimOrcaFee** account structures had hardcoded collateral index constraints that incorrectly assumed the position collateral would always be at index 0.

Recommendation:

Replace the hardcoded index 0 with the dynamic collateral index parameter:

[Implementation details omitted per client confidentiality requirements]

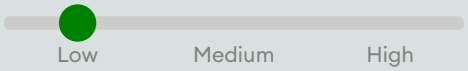
Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1222/files#diff-d6919cb683f20f4059b1412edb8b2c303ffa84130b11f158a489775a98d7abf9R43

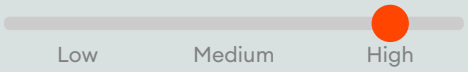
M03: Hardcoded Index 0 Breaking Raydium Transfers at Other Collateral Slots

Status: Resolved

Impact:



Likelihood:



Severity: Medium

Location: Private

Description:

The **TransferRaydiumPosition** account structure had hardcoded collateral index constraints that incorrectly assumed the position collateral would always be at index 0.

Recommendation:

Replace the hardcoded index 0 with the dynamic collateral index parameter:

[Implementation details omitted per client confidentiality requirements]

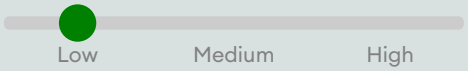
Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1222/files#diff-0bd349b7151057dd9025e6c213e73d147eade0c07fd50f63d55a38f0e03f0b70R234

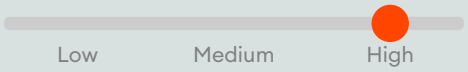
M04: Incorrect Asset Identifier Constraint Blocking Position Transfers

Status: Resolved

Impact:



Likelihood:



Severity: Medium

Location: Private

Description:

The **TransferRaydiumPosition** accounts structure in **transfer_position.rs** had an incorrect constraint for validating the position mint's asset identifier.

Correct Implementation:
[Implementation details omitted per client confidentiality requirements]

Recommendation:

Change the asset identifier constraint to validate against the pool key:

[Implementation details omitted per client confidentiality requirements]

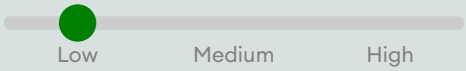
Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/commit/19785b1fd9bf3851b4dc1ae35c6ef91bb088f9aa#diff-094cb50c60e8cfbc1d426837532eaf79ac6ec32f88f84aa41939437afd4e9af5R46

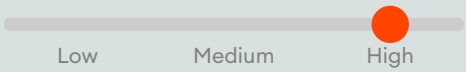
M05: Incorrect Price Ratio Calculations in Delta Amount Functions

Status: Resolved

Impact:



Likelihood:



Severity: Medium

Location: Private

Description:

The **get_token_amounts_for_raydium_position** function in **program-utils/src/clp/raydium.rs** has an incorrect parameter order when calculating delta amounts for positions where the current price was within the position's tick range.

Recommendation:

Correct the parameter order in the middle branch to match Raydium's implementation:

[Implementation details omitted per client confidentiality requirements]

Developer Response:

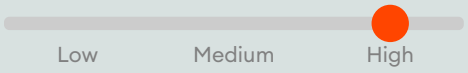
Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/commit/d3f17855101a859edec7d0c3db35fdf502087acd#diff-094cb50c60e8cfbc1d426837532eaf79ac6ec32f88f84aa41939437afd4e9af5R254

M06: Same protocol_position Used for Old and New Positions in transfer_position

Status: Resolved

Impact:

Likelihood:



Severity: Medium

Location: Private

Description:

The protocol_position account is shared between **decrease_liquidity_v2** (old position) and **open_position_with_token22_nft** (new position). However, in Raydium CLMM, **ProtocolPositionState** is a PDA derived from the pool and specific tick range (**[pool_id, tick_lower, tick_upper]**). If **params.tick_lower_index** or **params.tick_upper_index** differ from the old position's ticks, this uses the wrong PDA.

CPI fails if ticks change (invalid account data or non-existent PDA). Prevents repositioning to new ranges, limiting functionality. Could lead to incorrect liquidity addition if PDAs coincide by chance.

Recommendation:

Add a separate **new_protocol_position: AccountInfo<'info>** to the context, compute its PDA offchain based on new ticks, and pass it to the open CPI.

Developer Response:

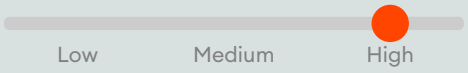
Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1222/files#diff-094cb50c60e8cfbc1d426837532eaf79ac6ec32f88f84aa41939437afd4e9af5R37

M07: Incorrect account passed in OpenPositionWithToken22Nft in transfer_position

Status: Resolved

Impact:

Likelihood:



Severity: Medium

Location: Private

Description:

In the **open_position** method, the CPI context passes **self.position.to_account_info()** (the old personal position account) as the **personal_position** account. However, this should be **self.new_position.to_account_info()** (the new position's PDA, derived from **new_position_mint**). The old position key is unrelated to the new mint, and the account was just closed in the previous step.

Recommendation:

Fix to personal_position: **self.new_position.to_account_info()**. Ensure offchain calculation of the new PDA is correct.

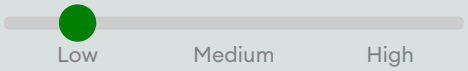
Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1222/files#diff-094cb50c60e8cfbc1d426837532eaf79ac6ec32f88f84aa41939437afd4e9af5R156

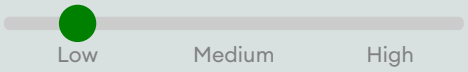
L01: Switchboard Integration Should Validate Against Negative Prices

Status: Resolved

Impact:



Likelihood:



Severity: Low

Location: Private

Description:

The **get_switchboard_price_with_quantity()** function only checks for zero prices but doesn't validate against negative values. Since Switchboard oracles return i128 values that are cast to Decimal (both signed types), negative prices can pass through undetected.
Negative prices could cause unexpected behavior in downstream calculations and potential economic exploits.

Recommendation:

The function should check against negative and zero price, and revert if this is the case.

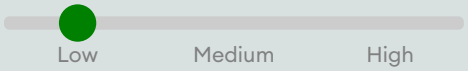
Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1237

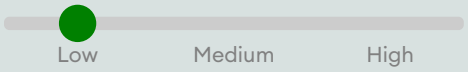
L02: Missing auth check in the manage liquidity account struct.

Status: Resolved

Impact:



Likelihood:



Severity: Low

Location: Private

Description:

The **ManageRaydiumLiquidity** and **TransferRaydiumPosition** accounts structures in the **manage_liquidity.rs** and **transfer_position.rs** files are missing the required **bs_auth** signer field that served as a temporary security measure. According to the development team's specifications, all instructions should have required the **bs_auth** signer as a security control.

Recommendation:

Add the **bs_auth** signer field to both the **ManageRaydiumLiquidity** and **TransferRaydiumPosition** accounts structures with the proper constraint validation:

[Implementation details omitted per client confidentiality requirements]

Developer Response:

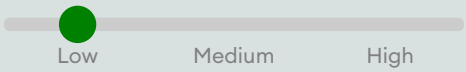
Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/commit/32c72f7af957a6f78e1c7ac39969f25781f42860#diff-0bd349b7151057dd9025e6c213e73d147eade0c07fd50f63d55a38f0e03f0b70R16

L03: ALP oracle integration should not read price from paused pools

Status: Resolved

Impact:

Likelihood:



Severity: Low

Location: Private

Description:

Adrena pools store numerous information that can be accessed by external programs.:
[Implementation details omitted per client confidentiality requirements]

While the **get_alp_price_with_quantity** function checks some of them, it does not verify the **allow_trade** and **allow_swap** states.
A pool with paused swaps and trades may indicate suspicious activity that could affect the LP token price.
Similarly, checking the **initialized** state of the pool would make the read operation more robust.

Recommendation:

Ensure **allow_trade** and **allow_swap** states are set to the correct value.

Developer Response:

We agree that the initialized and swap checks are worthwhile due to the impact on the ability to redeem the underlying asset. However based on our understanding of how Adrena uses the allow_trade flag, there may be valid reasons for turning off trading temporarily that we would not want to freeze loans during.
This is intended to be a regulatory/UX focused change to enable temporary pauses if compelled externally which would not modify the ALP value.

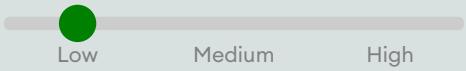
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1238

L04: Checking Cap Counter on CapOpType::Subtract Can Temporarily DoS Users Operations if Max Caps Are Updated

Status: Resolved

Impact:

Likelihood:



Severity: Low

Location: Private

Description:

Market parameters can be updated at any time by a market administrator using the Timelock instruction set.
Caps are one of those modifiable parameters. Only when the caps are increased, a timelock delay is enforced:

[Implementation details omitted per client confidentiality requirements]
Such a situation could prevent users from withdrawing their collateral until the cap is reset, which could be up to 24h depending on when it happens in the 24h window, or less depending on the **TIMELOCK_DELAY** value.

Recommendation:

Caps should be checked only for **CapOpType::Add** operations.

Proof of Concept:

Scenario:
- market has a **max_24h** deposit cap of $1000
- actual consumed cap **principal_24hr** deposit is at $950
- **max_24h** cap is decreased to $800 (with no timelock delay)
- user tries to withdraw $100
- deposit **principal_24hr > max_24hr** condition is checked: **950 - 100 > 800** check fails, the operation is rejected
- admin notices the issue and wants to increase **max_24h** back to a higher value: a timelock delay is applied, delaying the return to normal operations

Developer Response:

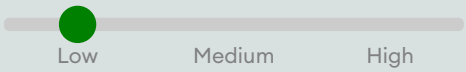
Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/commit/68fb3db5a3263bc343b720e3941f1241b5271ee2

L05: Consider Adding an Oracle Fallback Mechanism

Status: Acknowledged

Impact:

Likelihood:



Severity: Low

Location: Private

Description:

Currently, each asset in a market is associated with only one price oracle.
While Switchboard allows the aggregation of multiple sources, increasing its reliability, a back-up plan should be in place.

If, for any reason, the oracle becomes out-of-service and/or returns invalid data causing a revert in the program, the affected market will not be able to operate, as most operations require accessing the price feed in order to verify the collateralization of the loans.

Such a situation could even cause a loss of funds for lenders if the liquidation of a borrower cannot be achieved in time.
Having a fallback oracle could also allow the protocol to select the source with the best metrics on each interaction.

Recommendation:

Consider having two oracles per market, one main oracle and one fallback when applicable. If the main oracle fails, the price will be gathered from the fallback oracle.

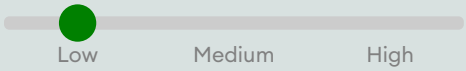
Developer Response:

Markets can have their oracle accounts updated on the fly and if there were issues with oracles, they could be replaced quickly. It is both impractical and only marginally useful to have an onchain, preconfigured redundancy when the vault manager can adjust this if needed.

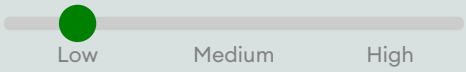
L06: Failure to Collect Rewards Before Closing Position

Status: Acknowledged

Impact:



Likelihood:



Severity: Low

Location: Private

Description:

decrease_liquidity_v2 collects fees automatically (added to withdrawn tokens), but rewards (from pool emissions) require additional recipient accounts passed as remaining accounts in the CPI. The code doesn't pass any (no **.with_remaining_accounts()**), and no separate **collect_reward** call is made. Raydium requires all liquidity removed, fees collected, and rewards collected before **close_position** succeeds. If pending rewards > 0, **close_position** fails. Hence the tx fails for positions with rewards, reverting everything.

Recommendation:

Add reward recipient accounts to the context of **decrease_liquidity**. Or add a separate instruction to collect the rewards as in **orca/collect_fees.rs**.

Developer Response:

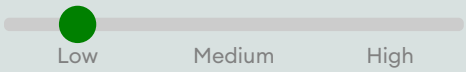
Acknowledged as we are moving away from CPIs for these instructions and will support reward claims via Loan Locking.

L07: Missing TickArrayBitmapExtension Support (Failure in Large Pools)

Status: Resolved

Impact:

Likelihood:



Severity: Low

Location: Private

Description:

For pools with many tick arrays (wide ranges or high volatility), Raydium requires a TickArrayBitmapExtension account for overflow. The code doesn't include this account or pass it via remaining accounts. Hence **decrease_liquidity_v2** fails with **MissingTickArrayBitmapExtensionAccount** if needed, breaking the instruction for certain pools.

Recommendation:

Add optional **tick_array_bitmap_extension: AccountInfo<'info>** to the context. Pass via **.with_remaining_accounts()** if required (detect offchain via pool state).

Developer Response:

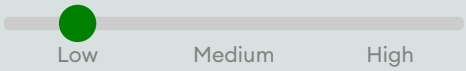
Fixed according to the provided recommendation.

L08: Switchboard Price Feeds Low Samples Count Makes Price Vulnerable to Manipulation

Status: Acknowledged

Impact:

Likelihood:



Severity: Low

Location: Private

Description:

Switchboard Solana feeds have a small number of samples (max 5, and often less than 3). With such a low number of samples, the mean and the median are easily manipulable in the event of an attacker getting access to a feed queue.
1. **mean**: this is straightforward, the mean will be impacted by the new sample value.
2. **median**: adding a sample either at the lower end, or upper end of the samples will allow the attacker to shift the median to a different sample (lesser impact)
3. **standard deviation**: even with a high number of samples this value can be highly modified, as it is sensitive to the square of the difference with the mean: **$(x - mean)^2$**

This could then allow an attacker to either:
1. DoS the oracle (std_dev > max allowed) forcing it to revert, and so preventing the associated market to operate.
2. trigger liquidations by manipulating the collateral price

Recommendation:

Make sure to use price feeds with a sufficient number of samples (see Appendix A.) to limit the risks of manipulation by checking **CurrentResult.num_samples**.
Additionally, you can also verify **PullFeedAccountData.max_variance** (which returns the maximum allowed variance of the job results (as a percentage) for an update to be accepted on-chain) and only integrate feeds with an acceptable value.

Developer Response:

The identified examples are generally true of any oracle. Price feeds need to be chosen with sources that are reputable, attackers cannot submit prices to all feeds. If price movements were to be high such that ETH did increase to 2200 as in the example, we would want to properly back off the usage of the price until it stabilized regardless.
A small number of submissions would result in less prices being accepted, meaning participants using these oracles would need to ensure they can generate stable prices (or use a higher uncertainty bound) when deciding on an oracle.
Given Switchboard is generally a fringe asset oracle and the specific up to the discretion of the lender, with the intent of being used for longer duration pricing and slower moving assets, we believe the severity of this issue should be Low as Impact is Medium/Low and Likelihood is Low as this is a decision made by the two loan participants when deciding on a feed.
To address some of the variance issues, we've reverted to using MAD which should resolve this in relevant cases.

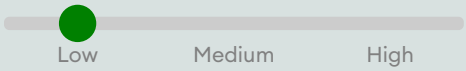
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1237

L09: Current Use of Standard Deviation for Price Boundary Estimation Is Flawed

Status: Resolved

Impact:

Likelihood:



Severity: Low

Location: Private

Description:

Switchboard feeds returns multiple statistical measures, including **median**, **mean** and **standard deviation**.

The current implementation contains two fundamental statistical inconsistencies:
1. In order to use the most favorable prices (for the protocol) for each operations, Loopscale uses when possible confidence interval (computed from the standard deviation) to estimate the lower and upper bounds of a price, such as **min_price = price - confidence_interval** and **max_price = price + confidence_interval**.
The code at the time of the audit uses the median as the base price, applying the standard deviation to it. This is mathematically incorrect because standard deviation measures the spread of data around the mean, not the median (see definition)
2. The calculation of the confidence intervals using the 3 sigmas rule which assumes the oracle price submissions follow a normal distribution.
Looking at the existing Switchboard feeds, most of them have less than 5 number of samples, which usually does not fit well with a normal distribution, and thus with the use of that measurement.

Recommendation:

1. Don't apply the standard deviation to the median as it is statistically incorrect.
2. Don't apply the standard deviation for feeds with less than ~30 samples.
3. See Appendix A. for more details on possible alternative solutions.

Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1237

5. Enhancement Opportunities

E01: Additional Security Consideration for Switchboard Feeds

Location: Private

Description:

Switchboard feeds could be updated by their admin at any time.
For that reason, some parameters should be validated in order to detect potential malicious tampering:
- **authority**: a malicious actor could take control of the feed
- **queue**: could redirect to a compromised oracle queue
- **feed_hash**: could alter the job schema.

Potential Benefit:

Increase the reliability of the feed's data.

Recommendation:

For each integrated feed, save those values and check against them in **get_switchboard_price_with_quantity**.

Developer Response:

Acknowledging the improvement but we do not intend to implement at this time as we would expect this data to be monitored off chain and updates to the MarketInformation account to take its place. In the future, we may explore some additional paths for validation depending on feedback from managers.

E02: Ambiguous Error Codes in Raydium Liquidity Management Operations

Location: Private

Description:

The Raydium liquidity management functions use the same error code (**InvalidLiquidityAmount**) for both increase and decrease liquidity operations, making it difficult to distinguish which specific operation failed during debugging and error handling.

Potential Benefit:

Enhanced Error Monitoring: System monitoring and logging can more accurately track and categorize different types of failures.

Recommendation:

[Implementation details omitted per client confidentiality requirements]

Developer Response:

This code has been removed in an above PR and we are highlighting more explicit errors here.

E03: Dead Code due to if statement guaranteed to not being executed

Location: Private

Description:

The first **if zero_out_matrix** check guarantees that **zero_out_matrix** is false within the subsequent **else** branch. Therefore, the second **if zero_out_matrix** check inside that branch is redundant, as it is equivalent to **if false**.

Potential Benefit:

Removing dead code, such as unreachable if branches, improves code clarity and reduces potential maintenance issues.

Recommendation:

Remove the dead code and ensure that the intended logic of zeroing the matrix entries is correctly implemented elsewhere.

Developer Response:

This code was removed in one of the PRs reviewed, it's possible it was not merged across all. Apologies.

E04: Over-Updating Allocations in Multi-Collateral Borrows/Repays

Location: Private

Description:

The function adds the full operation amount to each collateral asset's cap tracking during a borrow and subtracts the full amount from each during a repay, instead of splitting the amount proportionally. This inflates/deflates per-asset allocations by the number of assets, leading to inaccurate cap enforcement, either rejecting valid operations or allowing over-allocations.

1. Borrow 100$ backed by assets A and B
2. The program adds 100$ to A's allocation tracker: **A_allocation += 100**
3. The program adds 100$ to B's allocation tracker: **B_allocation += 100**
4. Total tracked allocation: 200$ (but actual borrow: 100$)

The reverse is true for repayments.

Potential Benefit:

Implementing a more sophisticated allocation would improve capital efficiency.

Recommendation:

Change params to **Vec<(Pubkey, u64)>** for per-asset amounts. Loop with split values, validate **sum == total amount**, and update accordingly to track accurately.

Developer Response:

This is intentional as we would otherwise be forced to adjust caps based on weight matrix updates generally. This is primarily a UX trade-off that could result in less collateral being available for borrowers, but lenders always have the ability to increase caps if needed.

E05: Some Tests Fail on Borrow Caps branch

Location: Private

Description:

Running tests using **cargo test --features=sandbox,test-bpf** results in 5 failing tests in **tests::ledger::sell::sell_tests**.

Potential Benefit:

Adding the below fix in the test files would make all tests execute successfully without causing issues in the CI/CD pipeline, such as false-positive failure alerts.

Recommendation:

The original problem was in the sell.rs test setup sequence:

1. Before fix: The test was calling **add_mock_strategy_apy** after creating the **AccountLoader** objects:
[Implementation details omitted per client confidentiality requirements]
2. After fix: The test now calls **add_mock_strategy_apy** before creating the **AccountLoader**:
[Implementation details omitted per client confidentiality requirements]

Developer Response:

This was due to a bad push that overrode existing CI checks and was resolved.

E06: get_alp_price_with_quantity assumes the quote_mint is USD_QUOTE_MINT

Location: Private

Description:

The function assumes **quote_mint** is **USD_QUOTE_MINT** but it doesn't check it internally. However, the check is made outside of the function, before it is called.

Potential Benefit:

Adding the check inside the function could be useful in future cases for example where a developer forgets to make the check outside of the function, and the result will not be as expected.

Recommendation:

Add a check against **quote_mint** for acceptable values in the function itself to prevent cases where the function is misused with a different **quote_mint** than **USD_QUOTE_MINT**.

Developer Response:

Agree with this, will minimize opportunity for a future change to cause issues.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/commit/3278766220137965d5d49c1bdb642e5b1cc40649

E07: Max collateral allocation cap check is skipped when NAV is zero

Location: Private

Description:

When **NAV** is zero, the max collateral allocation cap check is skipped and **new_amount** is simply added to **current_allocation_amount**. When **NAV==0** the execution should never reach this code.

Potential Benefit:

Adding a check for **NAV==0** and returning an error / exiting early would be a good invariant to make sure the value is never zero.

Recommendation:

Exit with an error if NAV is zero. At the moment the code silently allows it (even if outside logic makes sure the NAV cannot be zero).

Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1239

E08: Dust Transfer Uses Saturating Sub

Location: Private

Description:

The dust calculation uses **.saturating_sub(initial_loan_mint_0_collateral)**. At the end of the transaction the amount of token A and token B in the loan account should always be bigger than the one initially (the **amount_max_a**, **amount_max_b** are used). Otherwise that would mean the collateral decreased (which would mean a loss for the ones that deposited it) and this should never happen. Note that there are other places where the dust calculation is done using **.saturating_sub()** (eg. **orca/manage_liquidity.rs**)

Potential Benefit:

Using **.checked_sub** could be used as an invariant in this case. You make sure the existing collateral never decreases when someone increases his position.

Recommendation:

Use the more restrictive **.checked_sub()** instead of **.saturating_sub()** and error if negative (as a loss for the user isn't acceptable).

Developer Response:

Suggestion implemented in previous PR.

E09: Sanity checking input before forwarding to Raydium external programs

Location: Private

Description:

The code makes CPIs to **increase_liquidity_v2** and **decrease_liquidity_v2** in Raydium AMM V3. Recent issues (as of May 2025) in Raydium's liquidity addition ("deposit" in cp-swap, allowing unbalanced adds with max_amount=0) could translate into exploits via this code if unpatched: https://github.com/raydium-io/raydium-cp-swap/commit/183ddbb11550cea212710a98351779a41873258b

Although this program is not used, it is still a Raydium program.

Potential Benefit:

It is a good rule of thumb to sanity check the input especially if it is a basic **!= 0** and not rely on external programs to do the checks because they might have forgotten / they may have bugs or vulnerabilities.

Recommendation:

Add param validations (revert if max_amount=0 or liquidity_amount=0). Monitor/pin to patched Raydium versions. Add post-CPI position checks.

Developer Response:

We will ultimately be removing these instructions from the program and opt for the loan lock approach but the liquidity amount check is worthwhile in the meantime. The token balances could be zero reasonably given single sided positions.

E10: Inconsistent handling of initial balances between handlers

Location: Private

Description:

The read logic for the initial balances differs between the two handlers (increase vs decrease) liquidity

The **increase** handler reads the recorded values
let loan_ta_a_amount = ctx.accounts.loan.load()?.get_collateral_amount

While the **decrease** handler reads the directly from the token account via deserialize
let loan_ta_a_amount = TokenAccount::try_deserialize

This inconsistency means decrease preserves actual balances, while increase attempts to preserve recorded values, potentially "correcting" desyncs but behaving differently under fees or external interference.

If balances desync (due to token2022 transfer fees, or external transfers to loan TAs), behavior diverges: decrease preserves the desynced actual, while increase may force toward recorded (but fails under fees).

If desync exists (actual > recorded), borrowers can extract excess tokens in increase (harmless for their own loan, but unintended).

If actual < recorded, health is already overestimated (unsafe), and decrease preserves it without correction.

Potential Benefit:

Standardizing how actual balances are read in both handlers would make the code less error-prone and would likely make desyncs easier to catch in tests.

Recommendation:

Standardize on reading actual balances in both handlers. And if the "correcting" of possible desyncs is intended, always sync with recorded values on each operation.

Developer Response:

Fixed according to the provided recommendation.
Change: https://github.com/LoopscaleLabs/loopscale-program-library/pull/1242

About Us

About Adevar Labs

Adevar Labs is a boutique blockchain security firm specializing in web3 audits.

Built by a mix of experienced professionals in traditional enterprise and crypto natives who have contributed to some of the most critical projects in blockchain infrastructure.

Our team's background spans companies like Bitdefender, Asymmetric Research, Quantstamp, Chainproof, and Juicebox, and includes experience securing smart contracts, bridges, and L1 and L2 protocols across ecosystems like Solana, Ethereum, Polkadot, Cosmos, and MultiversX.

With over 100 audits completed and a portfolio that includes custom fuzzers, exploit modeling, and runtime testing frameworks, Adevar Labs brings both depth and precision to every engagement.

Our auditors have discovered critical vulnerabilities, built high-impact tooling, and placed in top positions in premier audit competitions including Code4rena and Sherlock.

Team members hold distinctions such as PhDs in software protection, and key roles at Fortune 500 companies, and leadership of flagship conferences like ETH Bucharest.

With team members having publications with over 1,100 academic citations and trusted by projects securing over $500M in on-chain value, Adevar Labs blends elite technical rigor with real-world security impact.

We also collaborate with some of the best independent security researchers in the web3 space.

Projects may optionally request specific contributors to be part of their audit.

Audit Methodology

Our audit methodology is specialized to provide thorough security assessments of Solana programs. We focus explicitly on rigorous manual analysis, detailed threat modeling, and careful validation of implemented fixes to ensure your programs operate securely and as intended.

1. Program Context and Architecture Analysis

Our auditors begin by deeply examining your Solana program's documentation, intended functionality, and account design. We meticulously map out program interactions, instruction processing flows, and state management logic. Special attention is given to understanding how your program interfaces with critical Solana system programs, such as the SPL Token Program, Stake Program, and System Program. Last but not least we check external integrations with other Solana projects and verify if the inputs and return values are handled properly.

2. Threat Modeling

We conduct targeted threat modeling tailored specifically for Solana's execution environment. We carefully define attacker capabilities and identify potential vulnerabilities that may arise from Solana-specific issues, including but not limited to:

- Unauthorized account data manipulation
- Improper ownership or signer verification

- Misuse of Program-Derived Addresses (PDAs)
- Incorrect use of Cross-Program Invocations (CPI)
- Failure to adequately handle account privileges or account states
- Risks stemming from rent-exemption and account initialization logic

3. In-depth Manual Security Review

Our experienced auditors perform an extensive manual security review of your Rust-based Solana programs. This involves a comprehensive line-by-line inspection of source code, focusing on common Solana vulnerabilities including, but not limited to:

- Missing or insufficient ownership checks
- Inadequate signer checks
- Incorrect handling of CPI calls and invocation privileges
- Arithmetic and integer overflow or underflow errors
- Unsafe deserialization and serialization of account data structures
- Improper token transfers and SPL-token logic issues
- Logic flaws in financial operations or state transitions
- Edge-case handling in instruction input validation
- Potential denial-of-service vectors related to transaction execution and account handling

During this stage, we clearly document any discovered vulnerabilities, including detailed descriptions, precise severity ratings, and recommendations for secure implementation. In situations where it is not clear how the vulnerability might be exploited we may also include a detailed proof-of-concept exploit including code snippets and instructions on how the exploit could be performed.

4. Detailed Fix Review and Validation

After the initial audit and your team's subsequent remediation efforts, we perform a comprehensive fix review to ensure the vulnerabilities identified have been effectively resolved.

We verify each fix individually, confirming that:

- Corrections effectively eliminate the security risks
- Changes do not inadvertently introduce new vulnerabilities or regressions
- The fixes align closely with Solana best practices and secure coding guidelines

Our detailed validation ensures that security improvements are robust, complete, and aligned with best practices specific to the Solana development ecosystem.

Confidentiality Notice

This report, including its content, data, and underlying methodologies, is subject to the confidentiality and feedback provisions in your agreement with Adevar Labs. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Adevar Labs.

Legal Disclaimer

The review and this report are provided by Adevar Labs on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Adevar Labs disclaims all warranties, expressed or implied, in connection with this report, its content, and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

You agree that access to and/or use of the report and other results of the review, including any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided.

You acknowledge that blockchain technology remains under development and is subject to unknown risks and flaws. Adevar Labs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open-source or third-party software, code, libraries, materials, or information accessible through the report. As with the purchase or use of a product or service in any environment, you should use your best judgment and exercise caution where appropriate.

Appendix A: Switchboard Integration Statistics

Core Concern

Our analysis identified several issues in the integration of Switchboard oracle feeds. Primarily, we observed an incorrect application of standard deviation to the median for confidence interval calculations. This issue is critically linked to the number of price samples available at each timestamp, suggesting that the price feeds could be highly susceptible to price manipulation.

Mathematical Inconsistency: Median + Standard Deviation

The current implementation utilizes the **median** as the base price but calculates confidence intervals using the **standard deviation**. This approach is mathematically inconsistent for the following reasons:

- **Standard deviation** measures the spread around the **mean**, not the median.
- These statistics belong to different mathematical frameworks: the median is a robust statistic resistant to outliers, whereas standard deviation assumes a mean-centered distribution.
- Consequently, the resulting confidence intervals lack a sound mathematical basis and may provide a misleading sense of security regarding price bounds.

Robustness of Median vs. Mean

Let's examine a realistic oracle scenario with five ETH price submissions (in USD):

Scenario 1: Normal Oracle Submissions

Submissions: [2000, 2005, 2015, 2020, 2025]

- **Mean = (2000 + 2005 + 2015 + 2020 + 2025) / 5 = 2,013**
- **Median = 2015** (middle value)
- **Standard Deviation = 9.27**

In this scenario, the median and mean are very close, both accurately representing the true price. While applying standard deviation to the median is mathematically incorrect, the error introduced might be negligible if both values are sufficiently close.

Scenario 2: One Outlier

Submissions: [2000, 2005, 2015, 2020, 2200] (Attacker submits an extreme value: 2025 is replaced by 2200)

- **Mean = (2000 + 2005 + 2015 + 2020 + 2200) / 5 = 2048** (Shifted by approximately 2%)
- **Median = 2015** (Still the middle value, demonstrating robustness)
- **Standard Deviation = 84.3** (Massively inflated)

While the median successfully identifies the true market price of 2015 despite the manipulation attempt, the standard deviation calculation is corrupted by the same outlier that the median

disregarded. This creates a fundamental inconsistency: the price estimate is accurate, but the uncertainty bounds are rendered meaningless.

Scenario 3: Additional Submission

Let's consider a specific scenario where the median could be more susceptible to manipulation:

Submissions: [1995, 2000, 2005, 2015, 2020, 2025] (Attacker submits an additional value)

- **Mean = (1995 + 2000 + 2005 + 2015 + 2020 + 2025) / 6 = 2010** (Shifted by approximately 0.1%)
- **Median = 2007.5** (Shifted by approximately 0.5%)
- **Standard Deviation = 10.80**

This scenario reveals that the median is not universally more robust than the mean. When an attacker can add new submissions rather than replace existing ones, the median's vulnerability depends on the sample size and distribution. With small, even-numbered samples, strategic placement of outliers can shift the median more significantly than the mean.

Invalid Distributional Assumptions

The confidence interval calculation uses a **1.96 multiplier** (for 95% confidence), which implicitly assumes:

- Oracle price submissions follow a **normal distribution**.
- Sample sizes are sufficient for the **Central Limit Theorem** to apply.
- Submissions are **independent and identically distributed**.

The typical threshold for normal distribution assumptions to be valid is n 30. For feeds with a sample size below this value, the use of confidence intervals loses statistical meaning.

Alternative Approaches

Min and Max Price

The Switchboard feeds return a **submissions** field, which contains all individual price submissions. An alternative approach for defining price bounds would be to use the submissions at positions **n-1** and **n+1**

from the median value. This approach is more conservative than relying on the **min_value** and **max_value** fields, which might represent values too far from the true price.

Uncertainty Threshold

Given that standard deviation is not suitable for small sample sizes, alternative approaches can help detect suspicious activity on the price feed:

- The Median Absolute Deviation (MAD) is a measure of statistical dispersion that is resilient to outliers.
- Calculate the relative spread around the median using the **n-1** and **n+1** submissions. For example: **Relative Spread = (upper_bound - lower_bound) / median * 100**.

These values can then be compared against a threshold, which should be determined based on the volatility of the analyzed asset:

1. **Stablecoins:** Characterized by very stable prices.
2. **BTC, ETH, Altcoins:** Generally less volatile than memecoins during periods of high market movement.
3. **Others (e.g., Memecoins):** Exhibit very high volatility.