IRONNODE

ironnode.io

DiversiFi

Solana Program / Token-2022

FINAL SECURITY AUDIT REPORT

by IronNode 29 October 2026

Executive overview

1 Executive Overview

1.1 Audit Overview

DiversiFi is a Solana index program that holds constituent assets in index vaults and issues dTokens to depositors. Administrators settle deposits, withdrawals and rebalances through Jupiter routes. The program uses signed asset prices for share accounting and supports SPL and Token-2022 tokens.

IronNode audited the program's 20 Anchor instructions, account and authority checks, signed quotes, Jupiter settlement, Token-2022 fee handling, user escrows and index management. The audit identified four Low severity code findings and one Informational configuration observation. Each code finding depended on a privileged administrator, manager or quote-signer condition; the demonstrated local impacts are described in the findings. Only the rebalance fee issue was introduced by the final Token-2022 integration commit.

1.2 Remediation Summary

The client addressed all four code findings in the remediation commit. The revised valuation rejects a zero price, constituent admission rejects unsupported decimals, the rebalance fee check counts Token-2022 withheld amounts, and Jupiter settlement binds the declared mint to its vault. We replayed the original attack paths against the amended program and checked positive controls. **All four Low findings are Solved.**

The 100% initial manager-fee configuration was recorded as an Informational observation and remains available. Its final status is Acknowledged; it is not included in the four code findings.

1.3 Scope and Verification

The audit reviewed the Token-2022 branch at the original commit and the subsequent remediation commit. The remediation changed eight files: seven Rust source files and one fork test. Both revisions are linked below.

Audited commit: https://github.com/SolutioFi-io/solutiofi-program/commit/777241487e708b3d772a621c86d24021cfe3abcc

Remediation commit:
https://github.com/SolutioFi-io/solutiofi-program/commit/d262ea6b33008c211a2e5c0220e9179e313e36b6

The original assessment used source review, targeted Solana validator tests, isolated mainnet-account forks, real Jupiter routes, arithmetic models and negative controls. For remediation verification, 32 Rust library tests passed. The focused zero-price, decimal and vault-binding checks passed. An independent replay rejected the above-cap Token-2022 fee route under both current and modeled high-fee settings, while a below-cap route settled.

The broader fork suite recorded 22 passing and seven failing cases. Its fee case stopped on a public-RPC account-fetch error before the assertion; other failures involved changing Whirlpool or Jupiter fork fixtures and dependent setup. Those failures do not replace the completed focused retests.

Severity

The four code findings retain the Low labels used in the initial report. Those ratings reflect the privileged prerequisites; they do not mean the conditional financial or availability impact would be small. The initial manager-fee item is Informational.

1.4 Findings Overview

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	4	1

Four Low findings Solved. One Informational observation Acknowledged.

Item	Finding	Severity	Final status
Item 1	Signed zero price omits held collateral from share valuation	Low	SOLVED
Item 2	Unsupported mint decimals can lock redemption	Low	SOLVED
Item 3	Withheld Token-2022 fees escape the rebalance cap	Low	SOLVED
Item 4	Withdrawal mint/vault mismatch underburns dTokens	Low	SOLVED
Item 5	Initial 100% manager fee	Info	ACKNOWLEDGED

Findings

2 Findings

2.1 Signed zero price omits held collateral from share valuation

LOW SOLVED

A correctly signed quote could report zero for a constituent already held by an index. The old deposit and withdrawal loops cast that price into an unsigned value and included the holding at zero in total value. An administrator still had to settle a transaction, and the quote signer had to supply the zero; a user could not author either privileged step alone. In the original BONK/USDT fork PoC, a $10 deposit minted about 101 times the correct-price control's shares. A later redemption paid $25.013085 while most of those shares remained.

2.1.1 Code location - original audit

programs/solutiofi-program/src/contexts/deposit.rs:310-316 - audited version

```
// programs/solutiofi-program/src/contexts/deposit.rs:310-316

                let vault_value_numerator = (amount as u128)
                    .checked_mul(feeds[i].feed_value() as u128)
                    .ok_or(ProgramError::ArithmeticOverflow)?;

                let vault_usd_value = vault_value_numerator
                    .checked_div(10u128.pow(mint.decimals as u32))
                    .ok_or(ProgramError::ArithmeticOverflow)?;
```

2.1.2 Impact

If a valid zero quote and an administrator settlement coincided, existing holders could be diluted or over-redeemed. The old fork measured that result locally; it did not establish a production zero quote or deployed exploit.

2.1.3 Recommendation

Reject nonpositive prices at quote acceptance and at every share valuation. Keep the quote bound to the current mint order and validate the producer's price scale. Test the old zero-price sequence alongside a positive-price settlement.

2.1.4 Remediation

The remediation uses the shared `usd_value` helper in deposit and withdrawal valuation. It rejects a nonpositive price. The zero-price fork case reverted at `InvalidOracleFeeds`, minted no shares, and settled the same leg after a valid quote. The share-dilution path is Solved. `update_oracle` can still store a validly signed zero, which pauses settlement until a replacement quote is written.

Reviewed fix

programs/solutiofi-program/src/guards.rs:65-74 - remediation commit

```
// programs/solutiofi-program/src/guards.rs:65-74

  /// USD value at 1e18 of a raw token amount, from a 1e18 price. A zero or negative price is
  /// refused: a constituent priced at nothing would undervalue the index and mint excess shares.
  pub fn usd_value(amount: u64, price: i128, decimals: u8) -> Result<u128> {
      require!(price > 0, ErrorCode::InvalidOracleFeeds);
      let scale = 10u128.checked_pow(decimals as u32).ok_or(ProgramError::ArithmeticOverflow)?;
      (amount as u128)
          .checked_mul(price as u128)
          .map(|v| v / scale)
          .ok_or_else(|| ProgramError::ArithmeticOverflow.into())
  }
```

2.2 Unsupported mint decimals can lock redemption

LOW SOLVED

A manager could add a 39-decimal SPL mint to the index. The old deposit and withdrawal loops evaluated `10^decimals` for every active mint, even a zero-balance vault; `10^39` overflows `u128` and aborted settlement. One raw atom in the new vault prevented normal removal because the removal path requires an empty vault. The original fork showed a Jupiter leg rolling back after the overflow, while a 38-decimal control settled.

2.2.1 Code location - original audit

programs/solutiofi-program/src/contexts/deposit.rs:310-316 - audited version

```
// programs/solutiofi-program/src/contexts/deposit.rs:310-316

                let vault_value_numerator = (amount as u128)
                    .checked_mul(feeds[i].feed_value() as u128)
                    .ok_or(ProgramError::ArithmeticOverflow)?;

                let vault_usd_value = vault_value_numerator
                    .checked_div(10u128.pow(mint.decimals as u32))
                    .ok_or(ProgramError::ArithmeticOverflow)?;
```

2.2.2 Impact

A manager error or malicious admission could block ordinary settlement and redemption until code or state recovery. Pending user escrows could still use their refund paths, subject to fees already paid. No affected deployed index was identified.

2.2.3 Recommendation

Enforce the supported decimal range at both admission paths and use checked scaling in valuation. Existing indexes should be checked for a funded unsupported constituent.

2.2.4 Remediation

The remediation caps admitted constituents at 18 decimals during initialization and later weight updates. Valuation uses `checked_pow` and returns an error instead of panicking. A real 39-decimal mint was rejected with `InvalidMint`, and no vault was created. The new admission path is Solved. The change does not remove a funded unsupported mint already present in an index; none was identified in this audit.

Reviewed fix

programs/solutiofi-program/src/contexts/update_weights.rs:149-157 - remediation commit

```
// programs/solutiofi-program/src/contexts/update_weights.rs:149-157

          // Process added mints: create vaults
          for added_mint in added_mints.iter() {
              let mint_account = &remaining_accounts[remaining_idx];
              let vault_account = &remaining_accounts[remaining_idx + 1];

              // Verify the mint account matches
              require_keys_eq!(mint_account.key(), added_mint.mint, ErrorCode::UnexpectedMint);
              crate::guards::require_supported_decimals(crate::guards::mint_decimals(mint_account)
  ?)?;
```

programs/solutiofi-program/src/guards.rs:76-81 - remediation commit

```
// programs/solutiofi-program/src/guards.rs:76-81

  /// A constituent must have decimals the valuation can scale, or every settlement on the index
  /// would overflow and the mint could not be traded out to remove it.
  pub fn require_supported_decimals(decimals: u8) -> Result<()> {
      require!(decimals <= MAX_CONSTITUENT_DECIMALS, ErrorCode::InvalidMint);
      Ok(())
  }
```

2.3 Withheld Token-2022 fees escape the rebalance cap

LOW SOLVED

The last audited commit capped a rebalance platform fee using only the treasury token account's spendable amount. A Token-2022 transfer fee can hold part of the same charged fee in that account's withheld extension. The old local Jupiter replay charged 150 bps gross but exposed about 45 bps to the check under a modeled future fee schedule. The route and fee schedule required an administrator-controlled rebalance and a fee-bearing output mint.

2.3.1 Code location - original audit

programs/solutiofi-program/src/contexts/rebalance.rs:96-105 - audited version

```
// programs/solutiofi-program/src/contexts/rebalance.rs:96-105

          let mut fee_before = 0u64;
          if let Some(fee) = &self.platform_fee_account {
              let expected_fee_account = get_associated_token_address_with_program_id(
                  &TREASURY,
                  &self.output_mint.key(),
                  &self.output_mint.to_account_info().owner,
              );
              require_keys_eq!(fee.key(), expected_fee_account, ErrorCode::UnexpectedFeeAccount);
              fee_before = fee.amount;
          }
```

programs/solutiofi-program/src/contexts/rebalance.rs:251-259 - audited version

```
// programs/solutiofi-program/src/contexts/rebalance.rs:251-259

          if let Some(fee) = &mut self.platform_fee_account {
              fee.reload()?;
              let fee_taken = fee.amount.checked_sub(fee_before).ok_or(ProgramError::ArithmeticOve
  rflow)? as u128;
              let gross = (received as u128).checked_add(fee_taken).ok_or(ProgramError::Arithmetic
  Overflow)?;
              require!(
                  fee_taken.checked_mul(10_000).ok_or(ProgramError::ArithmeticOverflow)?
                      <= gross.checked_mul(MAX_REBALANCE_FEE_BPS as
  u128).ok_or(ProgramError::ArithmeticOverflow)?,
                  ErrorCode::FeeRateTooHigh
              );
```

2.3.2 Impact

The old source could exceed its intended 100-bps gross fee policy while passing a separate 5% value floor. The passing old case used a local fee-schedule fixture, not the then-current mint schedule or a deployed target binary.

2.3.3 Recommendation

Measure the treasury's gross balance increase, including withheld Token-2022 fees, and keep the 100-bps comparison tied to the actual output-vault receipt. Test one above-cap route under both fee schedules and a below-cap route that must still settle.

2.3.4 Remediation

The new helper adds spendable and withheld token amounts before and after the Jupiter CPI. An independent real-route replay rejected 150 bps under both the current and locally modeled high-fee schedules; all affected balances rolled back. A 20-bps route succeeded and measured 19 bps gross after integer rounding. This fee-cap finding is Solved. The broader suite's fee case stopped on a public-RPC fetch error, so the independent replay is the executed fee evidence.

Reviewed fix

programs/solutiofi-program/src/guards.rs:94-107 - remediation commit

```
// programs/solutiofi-program/src/guards.rs:94-107

  /// What a token account holds for its owner plus any transfer fee Token-2022 withheld in it. A
  /// transfer into a fee-bearing mint credits the net amount and withholds the rest in the
  account.
  pub fn amount_with_withheld(account: &AccountInfo) -> Result<u128> {
      let data = account.try_borrow_data()?;
      if account.owner == &spl_token_2022::ID {
          let state = StateWithExtensions::<spl_token_2022::state::Account>::unpack(&data)?;
          let withheld = state.get_extension::<TransferFeeAmount>().map(|f|
  u64::from(f.withheld_amount)).unwrap_or(0);
          Ok(state.base.amount as u128 + withheld as u128)
      } else {
          let state = anchor_spl::token_interface::TokenAccount::try_deserialize(&mut &data[..])?;
          Ok(state.amount as u128)
      }
  }
```

programs/solutiofi-program/src/contexts/rebalance.rs:253-262 - remediation commit

```
// programs/solutiofi-program/src/contexts/rebalance.rs:253-262

          if let Some(fee) = &self.platform_fee_account {
              let fee_taken = crate::guards::amount_with_withheld(&fee.to_account_info())?
                  .checked_sub(fee_before)
                  .ok_or(ProgramError::ArithmeticOverflow)?;
              let gross = (received as u128).checked_add(fee_taken).ok_or(ProgramError::Arithmetic
  Overflow)?;
              require!(
                  fee_taken.checked_mul(10_000).ok_or(ProgramError::ArithmeticOverflow)?
                      <= gross.checked_mul(MAX_REBALANCE_FEE_BPS as
  u128).ok_or(ProgramError::ArithmeticOverflow)?,
                  ErrorCode::FeeRateTooHigh
              );
```

2.4 Withdrawal mint/vault mismatch underburns dTokens

LOW SOLVED

The old withdrawal path accepted an active declared input mint but did not bind it to the token vault Jupiter actually spent. In the prior JUP-to-USDC fork, declaring cheap BONK while supplying the JUP vault sold 10 JUP and paid USDC but burned 307 shares. Pricing the same debit as JUP required 2,753,369 shares at that pre-state. This required an administrator-signed mismatched route; user-only control was not shown.

2.4.1 Code location - original audit

programs/solutiofi-program/src/contexts/withdraw.rs:73-78 - audited version

```
// programs/solutiofi-program/src/contexts/withdraw.rs:73-78

      // Jupiter-related accounts
      pub input_mint: Box<InterfaceAccount<'info, MintInterface>>,
      #[account(mut)]
      /// CHECK: Validated manually in redeem_usdc_to_user using
  get_associated_token_address_with_program_id
      /// with the correct token program (supports both Token and Token2022)
      pub input_vault: Box<InterfaceAccount<'info, TokenAccountInterface>>,
```

programs/solutiofi-program/src/contexts/withdraw.rs:342-354 - audited version

```
// programs/solutiofi-program/src/contexts/withdraw.rs:342-354

          // Find the index of the input_mint in active_mints to get the corresponding feed value
          let mint_index = active_mints.iter()
              .position(|active_mint| active_mint.mint == self.input_mint.key())
              .ok_or(ErrorCode::InvalidOutputMint)?;

          // Calculate USD amount using the feed value at the same index
          let usd_amount_numerator = token_amount
              .checked_mul(feeds[mint_index].feed_value() as u128)
              .ok_or(ProgramError::ArithmeticOverflow)?;

          let usd_amount = usd_amount_numerator
              .checked_div(10u128.pow(self.input_mint.decimals as u32))
              .ok_or(ProgramError::ArithmeticOverflow)?;
```

2.4.2 Impact

A malformed privileged route could leave too many dTokens outstanding against less collateral, diluting other holders. The original one-holder fork established a severe one-leg underburn, not an executed multi-holder drain or production keeper defect.

2.4.3 Recommendation

Before CPI, require the input vault to be the canonical index ATA for the declared input mint and token program, and verify the account's stored mint. Keep the same real route as a wrong-pair negative case and a correctly paired positive control.

2.4.4 Remediation

The withdrawal code now binds the canonical ATA and token-account mint before Jupiter. The JUP/BONK replay reverted with `UnexpectedVault`; JUP, USDC, dToken supply and the user's ledger remained unchanged. With JUP correctly declared, the route spent 10 JUP, paid 2.690860 USDC and burned 2,697,611 shares. The finding is Solved. Deposit also received a matching output-vault check.

Reviewed fix

programs/solutiofi-program/src/contexts/withdraw.rs:115-123 - remediation commit

```
// programs/solutiofi-program/src/contexts/withdraw.rs:115-123

          // The swap must spend from the input mint's own vault. Any other constituent vault
  would be
          // measured as the input mint and priced at its price, burning too few shares.
          let expected_input_vault = get_associated_token_address_with_program_id(
              &self.index.key(),
              &self.input_mint.key(),
              self.input_mint.to_account_info().owner,
          );
          require_keys_eq!(self.input_vault.key(), expected_input_vault,
  ErrorCode::UnexpectedVault);
          require_keys_eq!(self.input_vault.mint, self.input_mint.key(),
  ErrorCode::UnexpectedVault);
```

2.5 Initial 100% manager fee configuration

INFORMATIONAL ACKNOWLEDGED

Index initialization still accepts a manager fee of 10,000 bps, while a later fee update is capped at 1,000 bps. The original local SBF fork showed a user funding a visibly configured 100% fee index: the manager received the deposit, while the user received no index shares or refundable credit. The comment explicitly permits 100%, and no client product rule establishes that creation should be capped at 10%.

2.5.1 Code location - original audit

programs/solutiofi-program/src/contexts/initialize.rs:87-89 - audited version

```
// programs/solutiofi-program/src/contexts/initialize.rs:87-89

      // Check if the fee is not greater than 100%
      let fee_value = fee.unwrap_or(0);
      require_gte!(10_000, fee_value, ErrorCode::InvalidFee);
```

2.5.2 Impact

A user funding this configuration can pay the full deposit as a manager fee. Whether that is an intended product choice or a defect depends on the client's stated policy and the fee shown to the user.

2.5.3 Recommendation

Decide the maximum initial manager fee. If the intended cap is 10%, align index creation with updates. If 100% is intentional, display it before the user authorizes a deposit and consider a user-set maximum acceptable fee.

2.5.4 Remediation

The fee setting was unchanged. An audit-only unit test still accepted 10,000 bps, calculated zero credited deposit, and rejected 10,001 bps. This design observation is Informational and Acknowledged; it is not counted among the four code findings.

Reviewed code

programs/solutiofi-program/src/contexts/initialize.rs:87-89 - remediation commit

```
// programs/solutiofi-program/src/contexts/initialize.rs:87-89

      // Check if the fee is not greater than 100%
      let fee_value = fee.unwrap_or(0);
      require_gte!(10_000, fee_value, ErrorCode::InvalidFee);
```

Thank you for choosing us!