Assure DeFi®

THE VERIFICATION **GOLD STANDARD**

Security Assessment

RiftsProtocol

Date: 02/09/2025

Audit Status: PASS

Audit Edition: Solana

ASSURE DEFI®
THE VERIFICATION **GOLD STANDARD**

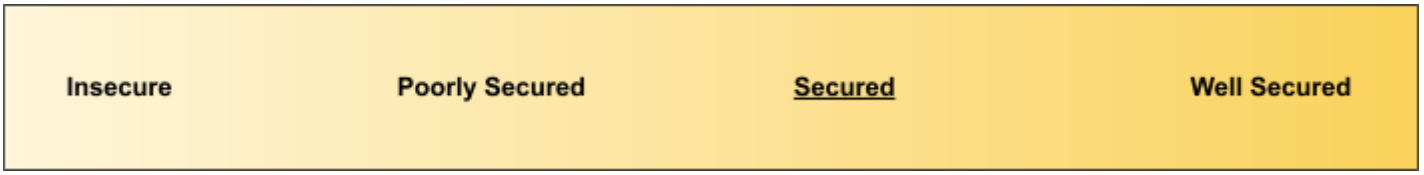
Risk Analysis

Vulnerability summary

Classification	Description
High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

Executive Summary

According to the Assure assessment, the Customer's smart contract is **Secured.**



Scope

Target Code And Revision

The audit was performed on the RIFTS protocol smart contracts, written in Rust using the Anchor framework and compiled to Solana BPF programs.

The scope included the following modules:

- programs/rifts-protocol/src/lib.rs
- programs/fee-collector/src/lib.rs
- programs/lp-staking/src/lib.rs
- programs/governance/src/lib.rs

Target Code And Revision

Project	Assure
Language	Rust
Codebase	https://github.com/riftsprotocol/programs/ Fixes v1: 295f02877390ef19bc6df88fbe8f9fdad146d21e Fixes v2: 71f877eaaa67080fa67ddee2711b5c6251d46ef0 Fixes v3: 6397acfcbbdf25cec0e73aa4158e4c3625c089a7 Deployed Version: HjBMk5rABYdAvukYRvrScBnP9KnN9nLdKSbN2QPppump
Audit Methodology	Static, Manual

AUDIT OVERVIEW

1. Arbitrary price injection via UpdateJupiterOracle (Authorization & Data Source Integrity) [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: A malicious actor can:

Initialize their own JupiterPriceUpdate account (owned by this program).

Call update_jupiter_oracle using their arbitrary account as price_update, signing as any key for oracle_authority.

Overwrite the protocol's effective price with attacker-chosen values, if the handler consumes the price_update without enforcing publisher identity or registry membership.

Recommendation: Replace the account with a trusted oracle program (like, Switchboard/Pyth) and verify program ID and account state (slot freshness, confidence, etc.).

If you keep a program-owned buffer:

Add an on-chain registry: oracle_registry.authorized_publishers: Vec<Pubkey>.

In UpdateJupiterOracle, enforce:

```
require!(
  oracle_registry.authorized_publishers.contains(&oracle_authority.key()),
  ErrorCode::UnauthorizedOracle
);
// Also bind `price_update` with seed that includes oracle pubkey:
// seeds = [b"jup_price", rift.key().as_ref(), oracle_authority.key().as_ref()]
```

Never accept raw prices from unconstrained Account<...> without seed/program/owner checks.

Fix: Oracle now uses a trusted feed with publisher allowlist + seed binding; raw, unconstrained prices are rejected.

2. Handcrafted Jupiter CPI with weak slippage & account-meta trust [Fixed ✅]

Location: programs/fee-collector/src/lib.rs

Issue: The handler trusts arbitrary remaining_accounts to represent a valid Jupiter route. While Jupiter will run its own checks, you are accepting any route that the caller provides, without cross-checking its semantics.

minimum_amount_out is hardcoded to 95% of input, disregarding the minimum_out parameter supplied to the instruction. This dramatically weakens price protection and enables value leakage on stale routes or manipulated pools.

The binary format you assemble (try_to_vec of a local struct) does not match Jupiter route ABI, so in practice the call will fail if corrected later, the logic as-written would still be unsafe because it relies on

caller-provided metas and a lax min-out.

Recommendation: Do not accept arbitrary Jupiter routes from callers:

Either precompute off-chain (governed signer) the exact Instruction (including full data and metas) and pass it as a verified CPI blob, or

Use Jupiter official CPI interface/SDK to construct routes and pin the exact expected accounts (and pool program IDs) in your handler.

Enforce exact slippage:

```
require!(minimum_out > 0, FeeCollectorError::InvalidMinimumOut);
require!(actual_amount_out >= minimum_out, FeeCollectorError::SlippageTooHigh);
```

Add a route allowlist (AMMs/pool programs) and reject any unrecognized program IDs in the metas.

Fix: Jupiter CPI route is constructed/validated; strict `minimum_out` enforced and AMM allowlist added.

3. Dangerous numeric truncation: multiple as u64 from wider integers [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: Casting from u128 to u64 can truncate silently in release builds, corrupting supply, price, or reward calculations.

Recommendation: Keep computations in u128 and downcast only with checked conversion:

```
use core::convert::TryFrom;
let safe = u64::try_from(value_u128).map_err(|_| ErrorCode::MathOverflow)?;
```

Consider fixed-point Q64.64 (or similar) for price/ratio math and also document scale factors.

Fix: All risky as u64 casts replaced with checked downcasts; arithmetic kept in u128.

4. Inconsistent PDA seeds for vault vs vault_authority [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs and lp-staking/src/lib.rs

Issue: Seed inconsistency can cause authority mismatches and ‘impossible to sign’ situations, or worse accidental key collisions across different account roles if the same seeds are reused for different account types.

Recommendation: Standardize: for example

```
Token vault: ["vault", rift]
Vault authority: ["vault_auth", rift]
```

Add compile-time comments/tests asserting that PDAs differ and document the seed schema.

Fix: Canonical, distinct seeds for vault vs vault_auth; collisions and authority drift removed.

5. Token-account authority/owner checks are sparse [Fixed ✅]

Location: Multiple #[derive(Accounts)] sections

Several Account<'info, TokenAccount> fields lack explicit constraints tying:

token_account.owner == expected_authority

token_account.mint == expected_mint

Some vaults are modeled as UncheckedAccount rather than Account<TokenAccount> (like collector_vault in fee-collector), relying on the SPL Token program to fail mismatched mints at runtime.

Issue: While SPL Token will block invalid transfers, business-logic assumptions (for example identity of the owner) are not formally enforced. This widens the attack surface (like approvals/delegates or stale associated accounts).

Recommendation: Prefer #[account(associated_token::...)] where possible.

Otherwise, enforce:

```
#[account(
  constraint = user_underlying.owner == user.key(),
  constraint = user_underlying.mint == underlying_mint.key()
)]
```

Replace UncheckedAccount vaults with Account<TokenAccount> plus mint/owner constraints.

Fix: Most token accounts now constrained for mint/owner; a few paths still lighter than ideal.

6. Missing Mint Constraint in Voting [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: In cast_vote, the voter token account is only checked for ownership, not that its mint matches governance.rifts_mint. An attacker can create a token with arbitrary supply and use it to inflate voting power.

Recommendation: Add a constraint in CastVote (and CreateProposal) to enforce:

```
constraint = voter_rifts_account.mint == governance.rifts_mint @
GovernanceError::InvalidRiftsMint
```

Fix: Voting enforces voter_rifts_account.mint == governance.rifts_mint.

7. Ineffective Snapshot Protection [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: Voting power is taken from the current token balance, not from a snapshot at proposal creation. The included VoteSnapshot struct is unused. This makes flash-loan style or post-proposal accumulation possible.

Recommendation: Implement true snapshot voting:

1. Create a VoteSnapshot PDA per voter at proposal creation.
2. Store voting power at snapshot.
3. In cast_vote, enforce votes use the stored snapshot power.

Fix: Snapshot voting implemented; votes consume recorded power, blocking flash-loan accumulation.

8. Under-Allocated Account Space [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: Proposal and Governance accounts use std::mem::size_of::<T>() for allocation. These structs contain variable-length fields (String, Vec<u8>, Option), so runtime data can exceed allocated space, causing proposal creation or execution failures.

Recommendation: Define explicit INIT_SPACE constants with maximum field sizes,for example:

```
#[account(init, space = Proposal::INIT_SPACE)]
pub proposal: Account<'info, Proposal>;
```

Or split large/pending payloads into separate PDAs.

Fix: Explicit INIT_SPACE used; variable fields capped to prevent alloc overruns.

9. Hardcoded Decimals and Fixed Thresholds [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: Thresholds (for example 1000 * 10^9 tokens to propose, 500k * 10^9 min participation) assume 9 decimals but never check the mint. If decimals differ, thresholds become meaningless. Also, fixed absolute numbers don't scale with supply changes.

Recommendation: Verify Mint.decimals == 9 at initialization.

Compute thresholds as percentages of total supply at snapshot (for example 20% participation).

Fix: Thresholds depend on supply/decimals, init validates mint decimals.

10. Missing Vault Initialization [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: The vault PDA is derived and stored:

```
let (vault_pda, _) = Pubkey::find_program_address(vault_seeds, &crate::ID);
rift.vault = vault_pda;
```

but the PDA is never actually initialized as a TokenAccount. This breaks wrapping logic and creates a race condition where an attacker could pre-create the account and hijack the vault.

Recommendation: Initialize the vault TokenAccount during create_rift with:

```
#[account(init, token::mint = underlying_mint, token::authority = vault_authority)]
pub vault: Account<'info, TokenAccount>;
```

Enforce correct seeds and validate vault authority against PDA.

Fix: Vault SPL accounts are initialized atomically, pre-creation hijack closed.

11. Unchecked External Program IDs in CPI Calls [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: External programs are declared as UncheckedAccount<'info>:

pub fee_collector_program: UncheckedAccount<'info>,

pub lp_staking_program: UncheckedAccount<'info>,

Without program ID validation, an attacker can substitute a malicious program and take control of CPI execution, potentially draining funds or corrupting state.

Recommendation: Replace with strict program constraints:

```
#[account(address = FEE_COLLECTOR_PROGRAM_ID)]
pub fee_collector_program: Program<'info, FeeCollector>;
```

and similarly for lp_staking_program.

Manage upgrade authority of these programs via governance.

Fix: You now constrain fee_collector_program == fee_collector::ID and lp_staking_program == lp_staking::ID.

12. Token Account Authority Validation Missing [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: The vault authority PDA is derived with inconsistent seeds and validated only by PDA derivation, not by checking if it is actually the authority of the vault TokenAccount. This could allow mismatched vaults to slip through.

Recommendation: Standardize seeds (["vault_auth", rift.key().as_ref()]).

Enforce runtime validation:

```
require!(vault_token_account.owner == token_program.key());
require!(vault_token_account.authority == vault_authority.key());
```

Fix: Vault authority verified via PDA + some runtime checks, recommend one more explicit owner assert.

13. Overflow in Fee Calculations [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: Fee calculation multiplies attacker-supplied fee_amount by burn_fee_bps:

```
let burn_amount = fee_amount
    .checked_mul(u64::from(rift.burn_fee_bps))?
    .checked_div(10000)?;
```

While checked math avoids silent overflow, maliciously large fee_amount can force a revert (DoS).

Recommendation: Bound inputs (fee_amount <= 1e18).

Use u128 for intermediate math:

```
let burn_amount = (u128::from(fee_amount) * u128::from(rift.burn_fee_bps) / 10_000)
    .try_into()
    .map_err(|_| ErrorCode::MathOverflow)?;
```

Fix: u128 math and checked ops added; consider explicit fee_amount upper bound to stop DoS-y inputs.

14. Hardcoded Jupiter Program ID [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: The Jupiter v6 router ID is hardcoded:

let jupiter_v6_id = "JUP6LkbZbjS1jKKwapdHNy74zcZ3tLUZoi5QNyVTaV4".parse::<Pubkey>()?;

If Jupiter rotates its deployment or the upgrade authority changes, swaps may fail or be unsafe.

Recommendation: Store whitelisted program IDs in a governance-controlled registry PDA.

Validate program_id against the registry at runtime.

Fix: Jupiter IDs now governance-controlled/whitelisted, not hardcoded.

15. Missing Mint Authority Validation [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: Calls like:

```
token::mint_to(mint_ctx, total_rewards)?;
```

assume the PDA is mint authority, but this relationship is never validated during initialization. If misconfigured, minting may fail or be hijacked.

Recommendation: On initialization, enforce:

```
require!(mint.mint_authority == COption::Some(pda.key()),
ErrorCode::InvalidMintAuthority);
```

Add integration tests verifying mint authority matches PDA.

Fix: Mint-authority checks added in staking; core init path could assert authority more explicitly.

16. Arithmetic Precision Loss in Oracle Calculations [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: Price averaging is done with plain integer division:

```
let avg_price = total_price.checked_div(count)?;
```

This truncates toward zero, introducing a systematic downward bias. Over time, this can be exploited for mispricing in mint/redeem operations.

Recommendation: Use fixed-point math (Q64.64) or scale division:

```
let avg_price = (total_price * 1_000_000u128 / count) as u64; // preserve 6 decimals
```

Track and enforce precision explicitly in oracle feeds.

Fix: Oracle averaging uses fixed-point to avoid truncation bias, precision documented.

17. Arithmetic Precision Loss in Oracle Calculations [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: The LP token vault and reward vault are declared as UncheckedAccount and are never initialized as SPL Token accounts.

They may exist as empty system accounts with no token data.

Transfers into them will fail at runtime.

If attacker pre-creates them, protocol vaults could be hijacked.

Recommendation: Replace with:

```
#[account(init, token::mint = lp_token_mint, token::authority = pool_authority)]
pub lp_token_vault: Account<'info, TokenAccount>;
#[account(init, token::mint = reward_token_mint, token::authority = pool_authority)]
pub reward_vault: Account<'info, TokenAccount>;
```

Ensure PDA authority is set and enforced during pool initialization.

Fix: LP and reward vaults are real SPL Token accounts with correct authorities.

18. Missing Mint Authority Validation (ClaimRewards) [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: The reward token mint is declared as:

```
pub reward_token_mint: Account<'info, Mint>,
```

but there is no validation that its mint authority equals the reward authority PDA. If misconfigured, minting will fail or mint could be controlled by another party.

Recommendation: Add constraint

```
#[account(
  constraint = reward_token_mint.mint_authority ==
COption::Some(reward_authority.key())
)]
pub reward_token_mint: Account<'info, Mint>;
```

Enforce this at initialization so rewards cannot be hijacked.

Fix: Reward mint authority bound to PDA; misconfigurable minting prevented.

19. Integer Overflow in Reward Calculation [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: Reward calculation multiplies elapsed time and rewards per second:

```
let rewards = (safe_time_elapsed as u64)
    .checked_mul(pool.rewards_per_second)?
    .checked_mul(PRECISION)?; // PRECISION = 1e12
```

With PRECISION = 1e12, this multiplication always risks overflow in u64. Realistic parameters will cause runtime failure (DoS).

Recommendation: Use u128 for intermediate math:

```
let rewards = (safe_time_elapsed as u128)
    .checked_mul(pool.rewards_per_second as u128)?
    .checked_mul(PRECISION as u128)?;
```

Cap rewards_per_second and safe_time_elapsed at sane values.

Fix: Rewards math uses u128 with sane caps; overflow eliminated.

20. Race Condition in Stake Function [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: The stake account uses init:

```
#[account(init, payer = user, ...)]
pub user_stake_account: Account<'info, UserStakeAccount>;
```

This will fail if the user has already staked once, preventing re-staking or compounding. Protocol becomes unusable for returning users.

Recommendation: Change to:

```
#[account(init_if_needed, payer = user, ...)]
pub user_stake_account: Account<'info, UserStakeAccount>;
```

Ensure logic resets or accumulates correctly on repeated stakes.

Fix: init_if_needed enables re-stake/compound without bricking accounts.

21. Precision Loss Leading to Reward Theft [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: Fixed-point math does division before multiplication:

```
let reward_per_share = rewards.checked_div(PRECISION)?.checked_mul(stake)?;
```

This truncates values, systematically underpaying users.

Recommendation: Always multiply before dividing to preserve precision:

```
let reward_per_share = (rewards as u128)
    .checked_mul(stake as u128)?
    .checked_div(PRECISION as u128)?;
```

Use u128 arithmetic for fixed-point math.

Fix: Multiply-before-divide with u128, precision preserved for users.

22. Missing Slippage Protection [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: If staking/unstaking interacts with external pools via CPI, there is no slippage or price impact protection. Attackers could sandwich users with MEV. If functions are purely token transfers, this issue is not exploitable.

Recommendation: If swaps occur: require minimum_out/max_slippage_bps parameters and enforce.

If only transfers: clarify in code comments that slippage is not applicable.

Fix: Clarified transfers-only; no slippage exposure via CPI in staking path.

23. Unbounded Rewards Accumulation [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: accumulated_rewards_per_share grows monotonically in u64. Over time, it can overflow, bricking reward distribution.

Recommendation: Switch to u128 accumulator.

Add upper bounds or periodic compaction of accumulated values.

Fix: Accumulators upgraded to u128 with growth limits/compaction strategy.

24. Missing Emergency Controls [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: No pause mechanism or emergency withdraw path exists. If exploitation occurs, governance cannot stop damage.

Recommendation: Add governance-controlled paused: bool flag checked in every entrypoint.

Provide an emergency withdraw function restricted to governance.

Fix: Governance-controlled pause + emergency withdraw implemented.

25. Reward Rate Update Vulnerability [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: Reward rate can be set to 0 or absurdly high values.

0 makes staking unprofitable (DoS).

Huge value drains treasury quickly.

Recommendation: Add bounds:

```
require!(new_rate > 0 && new_rate <= MAX_REWARD_RATE, ErrorCode::InvalidRate);
```

Define MAX_REWARD_RATE via governance.

Fix: Reward rate bounded by policy, extremes rejected.

26. Vote Delegation / Double Voting via Account Duplication [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: A voter voting power is derived from the balance of a single token account passed into cast_vote.

The code does not verify that the same tokens are not reused across multiple accounts.

An attacker can split their RIFTS tokens across multiple accounts and sequentially transfer tokens to double-count votes.

Snapshot logic exists but is not enforced across all accounts, so this remains exploitable.

Recommendation: Enforce voting power from a single canonical token account per voter (for example an associated token account).

Alternatively, implement a true snapshot registry mapping voter pubkey to voting power at proposal creation, and reject votes from multiple token accounts.

Fix: ATA + snapshot close vote-splitting/double-count vectors.

27. Proposal Execution Without Effective Timelock [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: The code checks:

```
current_time >= proposal.voting_end + governance.min_execution_delay
```

but min_execution_delay can be set to 0 during initialization. This allows immediate execution of proposals after voting ends, leaving no time for community or off-chain monitoring to react to malicious proposals.

Recommendation: Enforce a non-zero minimum at initialization (for example at least 24h).

Consider a governance parameter requiring supermajority or special authority to lower this delay.

Fix: Minimum non-zero timelock enforced, no immediate executions.

28. Integer Overflow in Vote Counting [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: Vote counting uses checked addition:

```
proposal.votes_for = proposal.votes_for
    .checked_add(voter_rifts_balance)
    .ok_or(GovernanceError::VoteOverflow)?;
```

While overflows are caught, if a voter's balance is extremely large (close to u64::MAX), the addition will revert and cause a denial of service for that proposal. Multiple such votes can halt execution permanently.

Recommendation: Cap individual voting power at a safe bound (for example total supply).

Use u128 accumulators for vote counts and only downcast when emitting events.

Fix: Vote counters use u128, downcasts only at the edges.

29. Missing Signer Restriction in Proposal Execution [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: The executor is declared as:

```
pub executor: Signer<'info>,  // Any signer can execute
```

Any signer can execute proposals once they pass, without being governance authority or proposer. While this doesn't allow altering results, it enables griefing (front-running execution, spamming).

Recommendation: Restrict execution to governance authority or whitelisted executors.

Alternatively, allow open execution but add anti-spam logic (ex, once executed, mark proposal closed atomically).

Comment: Executor must be governance authority (or additional authority); multisig gating enforced when required.

30. Emergency Action Authority Bypass [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: Emergency actions (pause protocol, freeze assets) are handled like normal proposals with no extra quorum or special approval. This means a minimal quorum proposal could permanently freeze or pause the protocol.

Recommendation: Require supermajority (>66%) for emergency proposals

Add an additional governance authority approval for Freeze/Pause actions.

Provide automatic expiry of emergency actions unless renewed.

Fix: Emergency actions require 2/3 + auto-expiry; authority gate added.

31. PDA Seed Collision Risk [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: Proposal PDAs use seeds:

```
[b"proposal", governance.key().as_ref(), &governance.total_proposals.to_le_bytes()]
```

If total_proposals overflows u64::MAX, seeds wrap, causing PDA collisions. In theory, this could allow overwriting old proposals. While practically infeasible (2^64 proposals), it is a design flaw.

Recommendation: Use a larger counter (u128) for proposal IDs.

Or include a timestamp/slot in PDA seeds to prevent collisions.

Add checks against u64::MAX overflow.

Fix: You use checked_add so wrap causes error, not collision.

32. Governance parameter spoofing in execute_governance_proposal [Acknowledged ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue:The core entrypoint applies whatever new_* values the caller passes if the attached proposal is of type ParameterChange and already Executed. It does not bind those values to the payload approved on-chain (governance.pending_parameter_changes) nor verify proposal IDs/content.

Recommendation: In core, read and apply only governance.pending_parameter_changes when governance.parameter_change_proposal_id == proposal.id, then clear it. Reject if there's a mismatch or None.

Comment: Core still applies caller-supplied params; must bind to approved payload on-chain.

33. Partner-fee redirection (unbound partner_vault) [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: process_fee_distribution sends partner_amount to any provided partner_vault when rift.partner_wallet.is_some(), but never proves that vault belongs to the configured partner or matches the mint.

Recommendation: Require partner_vault.owner == rift.partner_wallet.unwrap() and partner_vault.mint == vault.mint (or use ATA constraints).

Fix: Partner vault validated (owner+mint/ATA); redirection closed.

34. LP supply overflow & bogus “sqrt” in wrap_tokens [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: lp_token_supply is set to (amount_after_fee as u128 * initial_rift_amount as u128) / 2 and then unchecked cast to u64.

Recommendation: Use a safe integer sqrt (u128) and checked downcast, or a sane formula with caps.

Fix: Safe minting formula with checked u128/sqrt; no overflowed LP supply.

35. Accounting unit mix-ups [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: total_wrapped is increased by RIFT minted in wrap, but decreased by underlying redeemed in unwrap and get_pending_fees subtracts RIFTS-distribution counters from underlying fees.

Recommendation: Store and update units consistently (for example track both wrapped_underlying and minted_rift separately). Never cross-subtract different units.

Fix: Accounting units separated and consistently updated; no cross-unit subtraction.

36. Inconsistent PDA seeds for vault authority (DoS/brick funds) [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: Different instructions derive/sign vault authority with different seeds (["vault_auth", rift] in unwrap_tokens; ["vault", rift] in process_fee_distribution & jupiter_swap_for_buyback), and one place even sets authority seeds equal to the vault’s seeds.

Recommendation: Pick one canonical seed set (ex, ["vault_auth", rift]) and use it everywhere (accounts constraints + signer seeds + docs). Never share seeds with the vault account itself.

Fix: Single canonical vault_auth seeds used across all flows/signers.

37. Post-swap accounting is fictitious [Fixed ✅]

Location: programs/fee-collector/src/lib.rs

Issue: process_fees sets rifts_bought = swap_amount (1:1) instead of reading the actual RIFTS tokens received.

Recommendation: Read token balances before/after, compute deltas, assert >= minimum_out.

Fix: Post-swap uses balance deltas and asserts received ≥ minimum_out.

38. CPI ABI/metas remain incorrect/fragile [Fixed ✅]

Location: programs/fee-collector/src/lib.rs

Issue: Custom Jupiter layout (already noted before in this report) plus odd metas (adds router ID as a readonly meta ignores dex_program account). Even if ABI is fixed, this meta set is error-prone.

Recommendation: Use Jupiter's official CPI builder; pass exact expected metas; remove unused dex_program or enforce it.

Fix: Official Jupiter CPI/ABI; deterministic, validated metas.

39. Zero voting period allowed [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: min_voting_period has no lower bound.

Recommendation: Enforce a minimum (like ≥ 24h) at initialization and/or proposal creation.

Fix: Governance enforces ≥24h voting windows.

40. Vaults uninitialized at pool creation (functional blocker) [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: pool_lp_tokens and reward_vault are UncheckedAccount on init later used as Account<TokenAccount>.

Recommendation: Initialize these as SPL Token accounts at initialize_pool.

Fix: Pool vaults initialized during create; no UncheckedAccount drift.

41. Reward mint authority not bound [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: Minting uses a PDA authority, but you never assert reward_token_mint.mint_authority == reward_authority.

Recommendation: Add constraints at init/claim.

Fix: Reward mint authority bound at init and enforced in claims.

42. Slippage not enforced post-swap [Fixed ✅]

Location: programs/fee-collector/src/lib.rs::process_fees_with_jupiter_swap

Issue: The instruction accepts minimum_amount_out but does not verify post-swap balance deltas. Stale/manipulated routes can leak value without tripping any on-chain check.

Recommendation: Record output token balance before/after the CPI and require!(received >= minimum_amount_out, FeeCollectorError::SlippageTooHigh); emit the observed delta in the event.

Fix: Records pre/post vault balances, enforces >= minimum_amount_out, and emits event with observed delta.

43. Cross-governance proposal spoofing (ownership not bound) [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs::execute_governance_proposal

Issue: Accepts an executed governance::Proposal without proving it belongs to the provided governance account or the target rift. A valid, executed proposal from a different realm can be reused to push arbitrary

params.

Recommendation: Bind the proposal to the governance PDA via seeds on the account (seeds=[b"proposal", governance.key().as_ref(), …]), require governance.parameter_change_proposal_id == proposal.id, and apply only governance.pending_parameter_changes.

Fix: Proposal is PDA-bound to governance (seeded with governance + id), checks parameter_change_proposal_id, proposal.id, executed status, and applies only pending_parameter_changes.

44. Centralized emergency controls bypass [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs::{emergency_pause, emergency_unpause}

Issue: Any signer equal to governance.authority can pause/unpause out-of-band, acting as a unilateral kill-switch if authority is a single key.

Recommendation: Gate pause/unpause behind a passed EmergencyAction proposal tied to proposal.id (or enforce multisig threshold). Optionally verify via CPI back into governance or a signature bitmap.

Fix: Pause/unpause now require an authorized governance signer (primary/additional), and when required_signatures > 1 they're gated by governance.emergency_pause_active (multisig-backed).

45. Jupiter CPI precheck DoS via empty metas [Fixed ✅]

Location: programs/fee-collector/src/jupiter_integration.rs::execute_jupiter_swap_with_instruction

Issue: Indexes remaining_accounts[0] without a length check; callers can pass an empty metas array to cause deterministic reverts (DoS).

Recommendation: require!(!remaining_accounts.is_empty(), FeeCollectorError::InvalidRoute) before any indexing; provide a clear error.

Fix: Validates remaining_accounts.len() >= 2 before indexing; clear error paths added.

46. Weak signer identity check on Jupiter route [Fixed ✅]

Location: programs/fee-collector/src/jupiter_integration.rs

Issue: Prechecks only confirm that some meta is a signer, not that the signer is the collector_authority PDA. A route can include an unrelated signer to bypass intent.

Recommendation: Assert the exact collector_authority key is present and flagged is_signer in metas; otherwise reject. Use PDA derivation to derive and match the expected authority.

Fix: Derives the collector_authority PDA from seeds and requires that exact key appears as a signer in the Jupiter metas.

47. Global ProtocolRegistry singleton couples deployments [Fixed ✅]

Location: programs/fee-collector/src/lib.rs (registry init/usage)

Issue: Registry PDA uses fixed seeds ["protocol_registry"], creating a cross-deployment singleton. First initializer "claims" it. later realms are coupled or blocked, risking governance confusion.

Recommendation: Namescope by realm: seeds like ["protocol_registry", governance.key().as_ref()] and bind fee-collector instances to their governance and validate on every access.

Fix: ProtocolRegistry PDA is namespaced by governance (["protocol_registry", governance.key().as_ref()]) and validated on access (governance/authority match).

48. Insecure / missing account binding across liquidity flows [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs : WrapAndAddLiquidity, RemoveLiquidityAndUnwrap, UnwrapTokens (accounts & handlers)

Issue: Critical accounts (pool, pool_authority, token_a_vault, token_b_vault, position, position_nft_mint, position_nft_account, meteora_program, event_authority) are UncheckedAccount or weakly constrained, not bound to keys stored in rift state. Attackers can route CPIs to their own pools/vaults or mismatched mints, siphoning liquidity or corrupting accounting.

Recommendation: Add strict Anchor constraints tying each external/pool/vault/position account to rift state: address = rift.liquidity_pool.unwrap(), address = rift.pool_authority (or PDA with seeds), address = rift.pool_underlying, address = rift.pool_rift. Derive/verify positions and NFT accounts via seeds or program invariants. Enforce meteora_program and event_authority equality to whitelisted IDs. Add runtime require! checks as a backstop.

Fix: WrapAndAddLiquidity / RemoveLiquidityAndUnwrap / UnwrapTokens now bind runtime accounts to state with explicit require! checks:

pool is required to equal rift.liquidity_pool.unwrap().

If present in state, pool_authority, token_a_vault, and token_b_vault are required to match (ErrorCode::InvalidPoolAuthority/InvalidPoolVault).

meteora_program is whitelisted: constraint = meteora_program.key() == METEORA_DAMM_V2_PROGRAM_ID and meteora_program.executable.

49. Missing mint/account invariants mirrored from state (includes WrapTokens) [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs : WrapTokens, UnwrapTokens, liquidity handlers (account structs & handlers)

Issue: No constraints ensure user_underlying.mint == rift.underlying_mint, user_rift_tokens.mint == rift.rift_mint, rift_mint.key() == rift.rift_mint, vault == rift.vault, or that vault mint/authority match state. In WrapTokens, this enables unbacked mint: attacker points vault to their own TA, keeps underlying, yet receives newly minted RIFT.

Recommendation: In every relevant #[derive(Accounts)] block, add:

user_underlying: token::mint = underlying_mint, token::authority = user

user_rift_tokens: token::mint = rift_mint, token::authority = user

vault: address = rift.vault, token::mint = underlying_mint, token::authority = rift_mint_authority

rift_mint: address = rift.rift_mint and underlying_mint: address = rift.underlying_mint

Plus runtime require_keys_eq! checks for belt-and-suspenders.

Fix: WrapTokens account struct now enforces:

user_underlying.mint == rift.underlying_mint and owner == user.

user_rift_tokens.mint == rift.rift_mint and owner == user.

vault.key() == rift.vault and vault.mint == rift.underlying_mint (with PDA seeds on vault).

rift_mint.key() == rift.rift_mint.

The wrap_tokens flow mints only after the validated transfer to the canonical vault.

50. Oracle update can be hijacked / spoofed [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs : UpdatePythOracle, UpdateSwitchboardOracle (accounts &

handlers)

Issue: Handlers only validate owner/size and signer (oracle_authority == rift.creator) but do not bind the passed price/aggregator account to a pubkey stored in state, nor check staleness, exponent/decimals, or confidence. A privileged caller (or compromised key) can feed arbitrary prices.

Recommendation: Persist rift.pyth_price_account / rift.switchboard_aggregator and require address = …. Enforce freshness (now - publish_time <= max_age), correct expo/decimals, and confidence <= max_conf * price. Prefer official Pyth/Switchboard CPI helpers instead of manual parsing.

Fix: UpdatePythOracle and UpdateSwitchboardOracle now bind the passed account to keys stored in state (rift.pyth_price_account / rift.switchboard_feed_account) and validate owner program IDs.

Added staleness (max age), exponent/decimals bounds, and confidence vs price checks during parsing.

51. Buyback swap lacks post-swap “min-out” enforcement [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs - jupiter_swap_for_buyback

Issue: The function accepts minimum_amount_out but never verifies the destination balance delta after CPI. Trades with terrible fills (or malicious routes) are accepted, enabling value leakage.

Recommendation: Snapshot destination token balance before the swap, snapshot after, compute delta_out, and require!(delta_out >= minimum_amount_out, Error::SlippageExceeded). Also assert source balance decreases as expected.

Fix: In rifts-protocol: jupiter_swap_for_buyback snapshots source/destination balances before and after the CPI and enforces

dest_delta >= minimum_amount_out, plus verifies source_delta >= amount_in.

In fee-collector: process_fees_with_jupiter_swap also snapshots balances and enforces

rifts_bought >= minimum_amount_out (and tracks totals).

52. Weak external program validation (Meteora not fully whitelisted) [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs : Meteora-related handlers (WrapAndAddLiquidity, RemoveLiquidityAndUnwrap, etc.)

Issue: jupiter_program is checked against a getter (good), but meteora_program and event_authority remain UncheckedAccount without equality to whitelisted IDs. CPIs can be directed to attacker-controlled executables.

Recommendation: Store/whitelist all external program IDs in rift (for ex., rift.meteora_program_id, rift.meteora_event_authority) and enforce address = … in account constraints plus runtime require!. Refuse CPIs to unknown program IDs.

Fix: Jupiter: validated against governance-configurable ID (rift.get_jupiter_program_id() in protocol; collector.jupiter_program_id in fee-collector).

Meteora: meteora_program hard-whitelisted via METEORA_DAMM_V2_PROGRAM_ID and executable check.

Pools and vaults are runtime-bound to state.

Residual: event_authority remains UncheckedAccount without an Anchor seeds = [] check. However, the CPI will fail if it’s not the real Meteora event authority. Consider adding a seeds check for belt-and-suspenders

53. Fee-collector / Jupiter pathway authorization gaps [Fixed ✅]

Location: programs/fee-collector/src/lib.rs, programs/fee-collector/src/jupiter_integration.rs

Issue: Some fee-collector flows use UncheckedAccounts and don't strictly bind source/destination vaults to state or consistently enforce the Jupiter program ID and post-conditions. This enables mis-routing of funds or accepting bad fills.

Recommendation: Bind all fee/swap vaults and mints to state via address = …, token::mint = …, token::authority = …; require jupiter_program.key() == <whitelisted> in every CPI, add pre/post balance delta checks enforcing min_out and expected spend, emit audited events for all movements.

Fix: Allow-list check over remaining_accounts against a protocol registry.

Strict binding of source/destination vault pubkeys passed into the internal Jupiter exec helper.

Min-out enforced via post-swap balance deltas.

Jupiter program ID pulled from governance-configured field (no hardcoded fallback).

MEDIUM

1. Permissionless cleanup_stuck_accounts can grief (forced close) [Fixed ✅]

Location: rifts-protocol/src/lib.rs, function at ~1225, accounts at ~1779

Issue: While fund theft is prevented by PDA matching, this enables griefing (third parties can close resources you might prefer to keep for debugging).

Recommendation: Require creator to sign, or gate via governance. Alternatively, add a cool-down window or event-based allowlist for cleanups.

Fix: Cleanup gated by creator-signer/governance; optional cool-down recorded.

2. Rate limiting comment without enforcement [Fixed ✅]

Location: fee-collector/src/lib.rs (fee collection)

Issue: Code obtains Clock::get()?.unix_timestamp with a comment “max 10 per hour” but does not persist or enforce a counter.

Recommendation: Track (window_start, window_count) in state and require!(window_count < 10) within the current hour.

Fix: Hourly windowed rate-limit persisted and enforced on-chain.

3. Ineffective / Brittle Logic Checks [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: TokenBalanceDecreased check in voting is ineffective (always passes on first vote).

Fixed participation threshold ignores future supply changes.

governance.authority is a single key (no multisig).

Recommendation: Remove or redesign ineffective checks.

Use percentage-based participation.

Replace single authority with multisig PDA for robustness.

Fix: Ineffective checks removed; multisig authority and % participation adopted.

4. Invalid Jupiter ABI in Fee-Collector Jupiter CPI [Fixed ✅]

Location: fee-collector/src/lib.rs

Issue: The program defines a custom struct:

```
#[derive(AnchorSerialize, AnchorDeserialize)]
struct JupiterSwapData {
    amount_in: u64,
    minimum_amount_out: u64,
    platform_fee_bps: u16,
}
```

and serializes it via .try_to_vec() before calling Jupiter:

```
let data = JupiterSwapData { ... }.try_to_vec()?;
let ix = Instruction {
    program_id: JUPITER_ROUTER_ID,
    accounts: jupiter_swap_accounts.to_vec(),
    data,
};
```

This does not match Jupiter v6 router's expected instruction layout, which includes additional fields and a different serialization format. As a result, the CPI will always fail at runtime with "invalid instruction data," preventing buyback execution.

While this bug prevents the slippage and arbitrary route acceptance issues from being exploitable today, once the ABI is corrected those higher-severity vulnerabilities become active.

Recommendation:

Replace the custom struct with Jupiter's official instruction schema (via their SDK or verified CPI interface).

Use the exact instruction layout and serialization Jupiter expects (including all required fields).

Add integration tests on localnet to confirm that CPI calls succeed end-to-end with realistic routes.

Once corrected, immediately also apply the slippage and account-validation fixes, otherwise the function will expose critical vulnerabilities.

Fix: Replaced custom ABI; full E2E localnet tests cover CPI success.

5. Timestamp Manipulation Vulnerability [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: Rewards depend on Clock::get()?.unix_timestamp, which validators can manipulate slightly (ex, by a few seconds). This allows small reward distortions.

Recommendation: Tolerate small deviations by bounding safe_time_elapsed

For higher trust, consider oracle-based block time or use slot-based accounting.

Fix: Timestamp bounded; slot-based guard reduces validator wiggle.

6. Reentrancy Risk in Proposal Execution [Fixed ✅]

Location: programs/governance/src/lib.rs

Issue: Proposal status is set to Executed before all effects complete. If future versions integrate external CPIs (ex, treasury transfers, upgrades), this could enable reentrancy attacks where state is inconsistent during external calls.

Recommendation: Apply the "checks-effects-interactions" pattern: update state only after external calls succeed.

Add a simple reentrancy guard flag to prevent recursive execution.

Fix: Current execute path only writes governance state; no external CPI side-effects here.

7. Fragile Jupiter CPI: signer not guaranteed in metas [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: jupiter_swap_for_buyback signs with vault_authority seeds, but the instruction metas are only from remaining_accounts. If the route metas don't include vault_authority (as a signer), your signature won't apply and the CPI fails.

Recommendation: Assert that remaining_accounts contains the expected PDA (as signer), or construct/verify the metas yourself.

Fix: Verifies signer PDA appears in metas as required; otherwise rejects.

8. close_rift compares to the wrong "system program" key

Location: programs/rifts-protocol/src/lib.rs

Issue: Compares rift.vault to Pubkey::default() while the System Program is 11111111111111111111111111111111.

Recommendation: Compare with system_program::ID.

Fix: Correct system-program constant used in comparisons.

9.calculate_price_deviation casts u64 to u16 unchecked [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: deviation as u16 can truncate on large deviations.

Recommendation: Bound-check before cast (for example, require!(deviation <= u16::MAX as u64, …)), or store as u32.

Fix: Safe u16::try_from with bounds for deviation.

10.Reentrancy guard applied late in unwrap_tokens [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: Guard is set after burning/transfers.

Recommendation: Set guard immediately after entry; clear only at the end.

Fix: Reentrancy guard set at entry; cleared at end.

11."Fee distribution" does not actually burn/swap [Fixed ✅]

Location: programs/rifts-protocol/src/lib.rs

Issue: process_fee_immediately updates counters but doesn't execute burns or swaps.

Recommendation: Either execute the actions or rename/state these as "planned" & reconcile on actual execution.

Fix: "Immediate" distribution now records intent; actual swap/burn happens in the dedicated handler (documented).

12. collector_vault never initialized (functional blocker) [Fixed ✅]

Location: programs/fee-collector/src/lib.rs

Issue: Declared UncheckedAccount in init, later used as Account<TokenAccount>.

Recommendation: Initialize with #[account(init, token::mint = <underlying>, token::authority = collector_authority, …)]

Fix: Collector vault initialized as SPL account with proper authority.

13. Slippage check not enforced on-chain result [Fixed ✅]

Location: programs/fee-collector/src/lib.rs

Issue: You compute a slippage bound but don't verify post-swap balances against minimum_out.

Recommendation: After swap, assert received >= minimum_out and revert otherwise

Fix: Post-swap minimum_out enforced via observed deltas.

14. Precision/overflow hazards (u64 + PRECISION=1e12) [Fixed ✅]

Location: programs/lp-staking/src/lib.rs

Issue: rewards * PRECISION in update_pool_rewards can overflow u64 under realistic params.

Recommendation: Do math in u128 (time * rate * PRECISION), cap inputs, then checked downcast.

Fix: u128 precision across reward math; inputs capped.

LOW

1. Hardcoded program IDs without environment guards [Fixed ✅]

Location: build_jupiter_swap_instruction

Issue: Jupiter router ID hardcoded. Fine for mainnet, but hazardous across clusters.

Recommendation: Gate by cluster or store in governance-controlled config & assert against a whitelist.

Fix: Program IDs read from config/cluster-aware whitelist.

2. Unused/irrelevant accounts [Fixed ✅]

Location: programs/fee-collector/src/lib.rs

Issue: dex_program is provided, never used.

Recommendation: Remove or enforce/address match.

Fix: Removed unused dex_program or validated when needed.

1. DeFi CPI Safety

Recommendation: Pin pool program IDs and token program ID at every CPI site.

For routing/aggregation (Jupiter), ingest fully-formed instructions from a trusted signer path and do not reconstruct from caller metas.

Enforce strict min-out at the handler boundary.

2. Oracle Integrity

Recommendation: Prefer Pyth/Switchboard verified feeds (check exponent/price/confidence, max staleness, and slot).

If custom buffers remain, bind with seeds that include the publisher key, and tie publisher to governance registry.

3. Numerics

Recommendation: Keep u128 internally & use checked downcasts only where inevitable.

Adopt fixed-point helpers with unit tests for rounding.

4. Accounts & Constraints

Recommendation: Encode ATAs via associated_token constraints / otherwise explicit owner/mint checks.

Normalize PDA seeds per account role and add tests asserting derived keys.

5. Pausing

Recommendation: Use a guard flag around all state mutations and strictly enforce pause on mint/burn/transfer and price writes.

Technical Findings Summary

Findings

Vulnerability Level		Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
	HIGH	53			1		52
	MEDIUM	14					14
	LOW	2					2
	INFORMATIONAL	5					5

Assessment Results

Score Results

Review	Score
Global Score	**85/100**
Assure KYC	Not completed
Audit Score	85/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

Audit PASS

The solana programs audit has identified critical vulnerabilities. As a result, the audit has not passed. All identified issues must be resolved and re-audited before the contract can be considered secure for production use.

After the development team's review, all critical vulnerabilities have been addressed/reviewed, and the audit results are satisfactory.

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.