ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

SECURITY ASSESSMENT REPORT

VERIFIED BY ASSURE DEFI®

INTEGRITY • TRUST • CREDIBILITY

NAME:	RIFTS PROTOCOL
STATUS:	FAIL
DATE:	02/01/2025

Risk Analysis

Vulnerability summary

Classification	Description
High	High-severity issues can lead to **direct loss of funds**, **unauthorized state changes**, or **permanent corruption of on-chain data**. These vulnerabilities may allow attackers to drain program-owned accounts, bypass signer or ownership checks, or arbitrarily manipulate critical program logic.
Medium	Medium-severity issues are generally **more difficult to exploit** or require specific conditions, but they can still **negatively affect program security or correctness**. Examples include insufficient account validation, missing constraints, or logic flaws that could enable unintended behavior under certain circumstances.
Low	Low-severity issues typically relate to **best-practice deviations**, **inefficient logic**, or **edge-case behavior** that does not immediately threaten funds or program integrity. These findings generally have minimal impact on execution but may reduce code robustness or maintainability.
Informational	Informational findings include **code style issues**, **unused variables or instructions**, **documentation gaps**, or **general recommendations.** These do not affect program security or execution and are provided solely to improve code clarity and long-term maintainability.

Executive Summary

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

Insecure	Poorly Secured	Secured	Well Secured

Scope

Target Code And Revision

For this audit, we performed research, investigation, and review of the Voltaic verifying both the functional logic and surface-level implementation of the program, with access to the underlying source code.

Target Code And Revision

Project	Assure
Language	Rust
Codebase	https://github.com/riftsprotocol/rifts-protocol-v2 Commit: 756ef8054bf719cd4c19b0d808f35a9034be75e9
Audit Methodology	Static, Manual

AUDIT OVERVIEW

HIGH

1. Fee distribution can be redirected to arbitrary token accounts (treasury + partner theft)

Issue:

distribute_fees_from_vault (underlying fee vault)

Token program ownership + unpack succeeds, but do not enforce that:

- treasury_account is owned by rift.treasury_wallet
- partner_account is owned by rift.partner_wallet
- or that either is the correct ATA for (wallet, mint)

Evidence (line numbers from src/lib.rs):

- Treasury account validation only checks program owner + unpack (no owner/mint/ATA binding): L3460–L3478
- Partner account validation only checks program owner + unpack (no owner/mint/ATA binding): L3480–L3501

Transfers use whatever accounts are passed: L3572–L3608

Authorization allows creator OR partner OR treasury OR program authority to call distribution (L3509–L3514).

A malicious (or compromised) creator can:

provide the real treasury_wallet pubkey (passes check at L3519–L3523),

but pass a treasury_account token account they control,

and the program will transfer the “treasury share” to the attacker-controlled account.

Recommendation:

Mandatory account binding checks before any transfer:

- Enforce treasury_account.owner == treasury_wallet and treasury_account.mint == underlying_mint
- Enforce partner_account.owner == partner_wallet and partner_account.mint == underlying_mint

Stronger: enforce ATA derivation:

- treasury_account.key() == get_associated_token_address(treasury_wallet, underlying_mint)
- partner_account.key() == get_associated_token_address(partner_wallet, underlying_mint)

Prefer Anchor constraints using associated_token::authority and associated_token::mint where possible (reduces manual parsing risk).

2. Withheld-fee distribution (Token-2022) can also be redirected to arbitrary token accounts

Issue:

distribute_withheld_vault (RIFT withheld fee vault)

Evidence:

Treasury account validation checks only program owner (L3867–L3873) and later unpacks for balance checks but never asserts owner == treasury_wallet.

Partner account similarly is not bound to partner wallet (L3875–L3884).

Transfers send to passed accounts: L3982–L4019.

Recommendation:

Same as **1. Fee distribution can be redirected to arbitrary token accounts (treasury + partner theft)** (enforce owner/mint/ATA binding).

3. Oracle average logic can deadlock rebalance and related flows ("stale entry = hard fail")

Issue:

Rift::get_average_oracle_price() → get_average_oracle_price_with_options(false) (L5632–L5635)

get_average_oracle_price_with_options iterates stored oracle samples and returns error immediately if any sample is stale (unless allow_stale_fallback=true):

Evidence:

Staleness logic: L5645–L5663

if age > MAX_ORACLE_AGE and allow_stale_fallback == false → return Err(OraclePriceStale) (L5661–L5663)

MAX_ORACLE_AGE = 3600 seconds (L5642)

Recommendation:

Change the algorithm:

Skip stale samples in normal mode too, and require:

fresh_count >= MIN_FRESH_SAMPLES (for example, 3)

otherwise fallback (or fail with a clearer error)

Alternatively: store only the most recent sample + EMA, rather than a strict staleness constraint over a ring buffer.

Ensure update_manual_oracle calls get_average_oracle_price_with_options(true) if the purpose is to recover from deadlock (your comments suggest this intent).

MEDIUM

1. Emergency withdraw can sign for PDAs derived from arbitrary pubkeys (broad "2-of-2 admin" drain surface)

Issue:

admin_emergency_withdraw_vault signs with seeds derived from closed_rift_pubkey passed in instruction data (not inherently tied to the provided rift account).

Evidence (handler excerpt earlier in file + account struct):

Uses closed_rift_pubkey to derive vault_auth_seeds and signs CPI transfers.

Accounts struct AdminEmergencyWithdrawVault does not bind closed_rift_pubkey to rift.key() (see struct region around vault signer derivation).

Recommendation:

Bind emergency withdraw strictly to rift.key():

enforce closed_rift_pubkey == rift.key() (or remove parameter entirely and use rift.key()).

Add invariant checks that the vault being withdrawn is exactly rift.vault / rift.fees_vault (as applicable).

Use multisig / governance program for admin keys.

2. Underlying mint authority/freeze risks (SPL Token path)

Issue:

In create_rift, SPL Token underlying mints are allowed even if:

mint authority exists,

freeze authority exists,

Token-2022 has explicit blacklisting of dangerous extensions (PermanentDelegate, TransferHook, NonTransferable, etc.), but classic SPL Token does not have equivalent protections.

Recommendation:

If the protocol assumes "safe underlyings only," enforce:

- mint_authority == None
- freeze_authority == None

Or maintain an allowlist of acceptable underlying mints.

3. "95% threshold" leakage acceptance can hide meaningful losses for some tokens

Issue:

Both distribution functions accept actual_sent >= amount*95% style thresholds. This tolerates up to 5% leakage. If a token's effective fee behavior deviates, the protocol may silently lose value (though you cap some underlying fee configs earlier).

Recommendation:

either enforce exact deltas, or enforce token-specific fee maximums based on mint configuration.

4. distribute_fees_from_vault does not decrement rift.total_fees_collected

Issue:

distribute_fees_from_vault transfers underlying out of fees_vault to treasury/partner but does not decrement rift.total_fees_collected. As a result, on-chain accounting diverges from actual fees_vault balances, and close_rift can be permanently blocked because it requires total_fees_collected == 0.

Recommendation:

After successful transfers, decrement total_fees_collected by the actual amount debited (use pre/post balance diffs on fees_vault, or compute actual_sent). Update state only after all CPIs succeed, and use checked math.

5. Internal fee routing can leak value on transfer-fee underlying mints (Token-2022)

Issue:

When the underlying mint is Token-2022 with TransferFeeConfig, protocol fee routing transfers (vault to fees_vault) inside wrap_tokens and unwrap_from_vault can themselves be charged transfer fees. This can (a) reduce backing in vault more than expected, (b) reduce the credited amount in fees_vault, and (c) break strict “backing == minted minus burned” accounting assumptions unless you measure actual deltas.

Recommendation:

Either:

Measure actual credited/debited amounts for internal fee transfers via pre/post balance diffs and account accordingly, or avoid internal transfers for transfer-fee underlyings (keep fees in vault and distribute using measured deltas), or disallow transfer-fee underlying mints if strict invariants are required.

6. Incomplete Token-2022 extension validation for underlying mints (DoS risk)

Issue:

Underlying mint validation blocks several extensions (TransferHook/MemoTransfer/PermanentDelegate/NonTransferable/MintCloseAuthority) but effectively allows other extensions by default. Certain extensions (notably DefaultAccountState = Frozen, and other restrictive/confidential transfer extensions) can render vault token accounts unusable, permanently breaking wrap/unwrap even though rift creation succeeds.

Recommendation:

Switch to a deny-by-default policy for underlying Token-2022 mints:

- Explicitly allow only the minimal set you have audited and can support.
- Explicitly reject DefaultAccountState (Frozen), confidential transfer / transfer restriction extensions, and any hook/delegate-style extensions.
- Add a controlled allowlist (admin/multisig) only if you need broader mint support.

LOW

1. Vanity seed slicing in account constraints can panic pre-handler

Issue:

CreateRiftWithVanityPDA derives PDA seeds using &vanity_seed[..seed_len as usize]. If seed_len is invalid, the slice can panic during account validation before instruction logic runs, causing a hard failure (DoS for that transaction and sharp edge for clients).

Recommendation:

Avoid slicing in account attributes. Enforce bounds using a fixed-length seed approach:

Use full [u8; 32] seed (no dynamic slice), or

Accept Vec<u8> and only use it after explicit length checks by deriving PDA inside the handler (or by providing a safe pre-validated seed field).

INFORMATIONAL

1. Comment/code mismatch around oracle recovery mode

Issue:

There are comments indicating recovery mode should skip staleness, but some call sites still use with_options(false). This is a future footgun and makes incidents harder to resolve.

2. Oracle change execution is permissionless after timelock (operational risk)

Issue:

Once an oracle change is proposed and the delay has elapsed, any caller can execute the oracle switch. This is a common timelock pattern, but it may surprise operators expecting only the creator/admin to "pull the trigger," and it can shift execution timing control.

Recommendation:

If intended: document clearly as "anyone can execute after delay."

If not intended: require a signer (for example creator: Signer) and enforce rift.creator == creator.key() in the execute context/handler.

Technical Findings Summary

Findings

Vulnerability Level		Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
	HIGH	3					
	MEDIUM	6					
	LOW	1					
	INFORMATIONAL	2					

Assessment Results

Score Results

Review	Score
Global Score	**65/100**
Assure KYC	Not Completed
Audit Score	65/100

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 FAIL

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.

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.