Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

VERSEPROP

Verseprop

Veridise Inc.
November 20, 2025

► **Prepared For:**

Verseprop
www.verseprop.com

► **Prepared By:**

Aayushman Thapa Magar
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Dec. 05, 2025 V2
Dec. 04, 2025 V1

Contents

1 Executive Summary

From Nov. 20, 2025 to Nov. 26, 2025, Verseprop engaged Veridise to conduct a security assessment of the Verseprop project. The security assessment covered the Verseprop token contract implemented on Soroban/Stellar. Veridise conducted the assessment over 10 person-days, with 2 security analysts reviewing the project over 5 days on commit `1edcfdb`. The review strategy was based on a thorough manual review of the codebase.

Project Summary. Verseprop is a commercial real estate platform that uses tokenization to let investors participate in institutional-grade credit deals using digital assets. Verseprop uses a Soroban-based token contract that issues a fixed supply of digital tokens and manages how those tokens are held and moved on-chain. When the contract is set up, an issuer and administrators are defined, the full token supply is assigned to the issuer, and a fixed price in USDC is configured. Investors can buy tokens directly from the contract by paying USDC at this fixed price, with the contract tracking both token balances and the USDC it receives.

The contract provides three main capabilities:

- **Controlled issuance to investors.** A fixed supply of tokens is created and assigned to an issuer. Investors can purchase tokens from this issuer by paying USDC at a predefined price, giving them a tokenized claim that corresponds to their share.
- **Compliance-aware ownership and transfers.** Only investors who have passed Verseprop's checks (KYC/AML and other compliance reviews) can hold or receive tokens when those controls are enabled.
- **Regulatory-style controls for administrators.** Designated administrators can manage the investors KYC/Compliance list, restrict transfers, and in exceptional cases reclaim tokens (clawback) if required for regulatory, compliance, or risk reasons. The contract also holds the USDC received from token sales and allows only administrators to withdraw those funds.

Code Assessment. The Verseprop team provided the Soroban smart contract source code and accompanying documentation for this review. The implementation follows a security-token pattern on Stellar, with an issuance and management contract layered with compliance and authorization mechanisms. The codebase appears to be primarily original work produced by the Verseprop developers, complemented by use of standard Soroban and Stellar token primitives where appropriate. It includes a dedicated README describing the contract's purpose, key features, and regulatory considerations, as well as inline documentation comments that clarify the behavior of core entrypoints and administrative functions.

The source code included a comprehensive and well-designed test suite that, as Veridise security analysts noted was very extensive, and thoroughly exercises both happy-path and failure-path behaviors across the contract.

Summary of Issues Detected. The security assessment uncovered 11 issues, 1 of which was assessed to be of high severity. Specifically, (V-VSPR-VUL-001) describes an issue where, once added, admins could not be removed, exposing the contract to risks if an admin account were ever compromised.

The Veridise analysts also identified 3 medium-severity issues, including (V-VSPR-VUL-002) and (V-VSPR-VUL-003), where the clawback feature attempted to remove an exact token amount from a specified account and would revert entirely if the account did not hold a sufficient balance, rather than withdrawing only the available amount. Moreover, providing a negative amount would incorrectly increase the account's balance instead of decreasing it.

In addition, 4 low-severity issues and 2 warnings were also identified during the assessment.

The Verseprop developers were very proactive and submitted fixes to all the identified issues in a prompt manner. Additionally, beyond addressing the initially reported issues, the developers introduced a fix in commit `ce7f698` that restricts USDC withdrawals to issuer users only.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Verseprop	`1edcfdb`	Rust	Stellar

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Nov. 20–Nov. 26, 2025	Manual & Tools	2	10 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	1	1	1
Medium-Severity Issues	3	3	3
Low-Severity Issues	4	4	4
Warning-Severity Issues	2	2	2
Informational-Severity Issues	1	1	1
TOTAL	11	11	11

Table 2.4: Category Breakdown.

Name	Number
Usability Issue	4
Logic Error	2
Data Validation	2
Maintainability,Access Control,Usabili	1
Logic Error,Usability Issue	1
Maintainability	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Methodology

The Security Assessment process consists of the following steps:

1. Gather an initial understanding of the protocol's business logic, users, and workflows by reviewing the provided documentation and consulting with the developers.
2. Identify all valuable assets in the protocol.
3. Identify the main workflows for managing these assets.
4. Identify the most significant security risks associated with these assets.
5. Systematically review the codebase for execution paths that could trigger the identified security risks, considering different assumptions.
6. Prioritize one finding over another by assigning a severity level to each.

3.2 Identified Security Risks

After the initial phase of the security assessment was completed, a list of potential security risks was generated. Security analysts used this list during the code review as a starting point to identify potential attack vectors. A few of these risks, when expressed as questions, include the following:

- Could an attacker exploit limitations in the asset-delivery flow to force tokens to be sent to unintended recipients?
- Is it possible for adversaries or unexpected system states to prevent partial or full recovery of tokens during administrative recovery workflows?
- Can token-recovery operations be abused or misconfigured in a way that unintentionally alters total supply or diverts recovered assets away from authorized custody?
- Could incorrect trust or authorization settings block critical protocol operations or create conditions where required actors cannot participate in the token lifecycle?
- Does the governance model allow privileged roles to persist indefinitely, enabling attackers or compromised accounts to retain long-term control without a way to revoke access?
- Is the contract's initialization sequence vulnerable to front-running or unauthorized invocation, allowing attackers to set or manipulate core system parameters before legitimate operators?
- Could critical on-chain state expire or become inaccessible over time, resulting in partial system failure, denial-of-service, or operational disruption?
- Are there pathways through which balance-changing operations could be inverted, bypassed, or abused by providing unsafe or malformed input values?
- Does the system include features that appear optional but cannot actually be disabled, potentially locking the protocol into a permanently restricted operational mode?

3.3 Scope

The scope of this security assessment is limited to a single file, as agreed upon with the Verseprop developers.

- `/contracts/token/src/lib.rs`

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Security Review Assumptions

4.1 Operational Assumptions

Veridise analysts assumed the following properties held when modeling security for Verseprop.

- The issuer is governed through a multisignature wallet that performs operations in alignment with the long-term security and stability of the protocol.
- Administrator accounts are provisioned with defined lifecycles and are promptly removed when their credentials are compromised or no longer required.
- Token contract owner regularly extends the Time-to-Live (TTL) of relevant storage entries to ensure continued availability and proper functioning of the token.
- The Verseprop token price is initialized once and remains static throughout the lifetime of the protocol.
- When applicable, both users and Verseprop administrators undergo required KYC and compliance checks prior to interacting with restricted protocol operations.

Operational Recommendations. Highly-privileged operations should be operated by a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. Highly-privileged operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-VSPR-VUL-001	Lack of admin revocation mechanism . . .	High	Fixed
V-VSPR-VUL-002	Clawback logic does not handle . . .	Medium	Fixed
V-VSPR-VUL-003	Clawback mechanism burns tokens . . .	Medium	Fixed
V-VSPR-VUL-004	Negative clawback amount allows . . .	Medium	Fixed
V-VSPR-VUL-005	Purchase fails due to missing Issuer . . .	Low	Fixed
V-VSPR-VUL-006	Potential front-running of Token's . . .	Low	Fixed
V-VSPR-VUL-007	Storage items TTLs not updated in . . .	Low	Fixed
V-VSPR-VUL-008	Clawback mode cannot be disabled	Low	Fixed
V-VSPR-VUL-009	Missing beneficiary parameter in purchase . . .	Warning	Fixed
V-VSPR-VUL-010	Name, Symbol, and Home Domain can be . . .	Warning	Fixed
V-VSPR-VUL-011	Maintainability Issues	Info	Fixed

5.1 Detailed Description of Issues

5.1.1 V-VSPR-VUL-001: Lack of admin revocation mechanism creates governance risk

Severity	High	**Commit**	1edcfdb
Type	Access Control	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:296`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/7`, daad576		

Description The current access-control design allows new admin accounts to be added but provides no mechanism to revoke or demote an existing admin. As a result, once an address is granted admin privileges, the privilege is permanent for the lifetime of the contract.

This creates a critical security concern. If an admin's private key becomes compromised-or if an admin behaves maliciously-the issuer has no way to remove that account's privileges. Even if the issuer remains fully trusted and compliant, the compromise of *any* admin key becomes unrecoverable at the protocol level.

Impact This limitation introduces a high-severity, systemic risk:

- A single compromised admin key grants the attacker indefinite high-level control.
- The issuer has no ability to respond to compromised keys, operational mistakes, or offboarding of personnel.
- Long-term protocol security depends on the perpetual security of every admin's private keys, which is unrealistic in production environments.
- The design may violate operational security and custody requirements for regulated deployments.

The inability to revoke admin rights significantly undermines the protocol's resilience and its ability to maintain secure governance over time.

Recommendation

Introduce a mechanism to revoke admin permissions.

Developers Response

Developers have fixed the issue at commit daad576.

5.1.2 V-VSPR-VUL-002: Clawback logic does not handle insufficient user balance gracefully

Severity	Medium	**Commit**	1edcfdb
Type	Logic Error	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:274-275`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/4`, 9463a53		

Description

The current clawback implementation attempts to withdraw **exactly** the amount specified in the `amount` argument. If the target account does not hold the full amount, the operation fails, preventing the issuer from recovering any portion of the remaining tokens.

A more robust approach would be to claw back **up to** the requested amount-i.e., withdraw the lesser of the user's current balance and the requested `amount`. This behavior more accurately reflects the intent of a clawback mechanism: recovering as much as possible without exceeding the authorized limit.

Impact

Because the function requires an exact amount match, clawback attempts may fail in scenarios where the user's balance is lower than expected (e.g., transfers, partial redemptions, rounding effects). As a result:

- The issuer may be unable to recover *any* tokens, even though a portion is still present.
- Operational workflows relying on clawback may break or require additional off-chain checks.
- Recovery procedures become brittle, especially in integrations where balances can change between authorization and execution.

Recommendation

Modify the clawback logic to withdraw: `min(requested_amount, user_balance)`

This ensures the issuer can always recover the remaining tokens up to the requested amount while still enforcing the upper bound defined by the `amount` argument. Include appropriate event emission to reflect partial clawbacks if implemented.

Developers Response

Developers have fixed the issue at commit 9463a53.

5.1.3 V-VSPR-VUL-003: Clawback mechanism burns tokens instead of returning them to the Issuer

Severity	Medium	**Commit**	1edcfdb
Type	Logic Error	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:100`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/4`, 602de76		

Description

The current implementation of the clawback workflow **burns** the clawed-back tokens rather than returning them to the issuer's address. Burning tokens during a clawback effectively reduces total supply and may contradict expected compliance workflows, including auditability, reconciliation of circulating supply, and issuer-side reporting.

Impact If this behavior is unintentional or misaligned with regulatory requirements, it may lead to:

- Incorrect total supply accounting.
- Inability for the issuer to reassign or reissue the clawed tokens.
- Non-compliance with regulatory frameworks that require clawed assets to remain under issuer control.
- Potential discrepancies during financial or legal audits.

Recommendation

Instead of burning clawed tokens, redirect them to the issuer's designated account (e.g., `issuer`, `admin`, or a regulated custody address).

Developers Response

Developers have fixed the issue at commit 602de76.

5.1.4 V-VSPR-VUL-004: Negative clawback amount allows increasing user balance

Severity	Medium	**Commit**	1edcfdb
Type	Data Validation	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:279-280`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/4`, dbb7d2f		

Description

The `clawback` function accepts a signed `i128 amount` and does not enforce that this `amount` is strictly positive.

Because `amount` is signed and never validated to be positive, an admin can call `clawback` with a **negative** value. In such a case:

- The balance check `current_balance < amount` will typically pass, because a balance will never be less than a negative `amount`.
- The line `current_balance.checked_sub(amount)` effectively becomes `current_balance - (negative amount)`, which **increases** the user's balance instead of reducing it.

This turns the clawback mechanism - intended to *remove* tokens - into a way to **mint** tokens to the `from` account.

Impact

A malicious or mistaken admin can use the `clawback` function to arbitrarily increase user balances by passing negative values as `amount`. This has several serious consequences:

- Violation of the intended semantics of clawback, which should only *reduce* balances.
- Potential arbitrary inflation of token supply (depending on how total supply is accounted for), undermining economic assumptions.
- Possibility of severe financial loss or protocol insolvency if inflated balances are redeemed or traded.

Even if admins are "trusted," a simple input mistake (e.g., a negative value due to a bug in off-chain tooling) could have catastrophic effects.

Recommendation

Enforce that `amount` is strictly positive before performing any balance updates.

Developers Response

Developers have fixed the issue at commit dbb7d2f.

5.1.5 V-VSPR-VUL-005: Purchase fails due to missing Issuer authorization

Severity	Low	**Commit**	1edcfdb
Type	Usability Issue	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:378-379`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/5/`, c6fd4ed		

Description

When the token's `config.auth_required` flag is enabled, all accounts must pass the issuer-defined KYC/compliance checks before they are allowed to interact with restricted token operations. However, the token issuer's own address is **not** automatically marked as an authorized / KYC-passed account.

As a result, when users attempt to purchase tokens, the purchase flow relies on the issuer (as the token sender or intermediary) to pass the compliance check. Since the issuer has never been recorded as "approved," the compliance verification fails, causing all purchase attempts to revert.

Impact

With `auth_required` enabled, the protocol becomes unusable for token purchases unless the issuer is explicitly marked as having passed the required KYC/compliance checks.

Recommendation

Explicitly whitelist the issuer during contract initialization.

Developers Response

Developers have fixed the issue at commit c6fd4ed.

5.1.6 V-VSPR-VUL-006: Potential front-running of Token's initialization

Severity	Low	**Commit**	1edcfdb
Type	Usability Issue	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:76-154`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/8`, 80811e5		

Description

It has been identified that the `SecurityTokenContract` uses a dedicated `initialize()` function to set initial token parameters-such as name, symbol, and decimals-after deployment. This two-step approach introduces a potential initialization front-running risk, where an attacker could invoke the initialization function with their own parameters immediately after deployment but before the legitimate initializer does. It should be noted that, starting from Soroban protocol version 22, contracts can be initialized directly via the `__constructor()` function, allowing initialization to occur in the same transaction as deployment and thereby eliminating this front-running risk.

Impact

Not using constructors can inadvertently expose contracts to front-running risks, where malicious actors may exploit knowledge of pending transactions to inject or modify initialization parameters.

Recommendation

It is recommended to either use the `__constructor()` function to initialize the contract or restrict the initialization function so that it can only be invoked by a designated admin user.

Developers Response

Developers have fixed the issue at commit 80811e5.

5.1.7 V-VSPR-VUL-007: Storage items TTLs not updated in SecurityTokenContract contract

Severity	Low	**Commit**	1edcfdb
Type	Usability Issue	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/5`, 7fb26aa		

Description

In the `SecurityTokenContract` contract implementation, the Time-To-Live (TTL) of individual storage entries is not explicitly updated during contract operations. On Soroban, all storage items have a finite TTL that decreases with ledger progression unless explicitly extended. Once the initial default TTL expires, the associated entries will be archived by the Stellar network.

As a result, the `SecurityTokenContract` contract will not be able to process requests related to these entries once they are archived. Although archived entries can be recovered by users or developers, the recovery process requires additional transactions and coordination. This creates operational overhead and increases the risk of denial-of-service coming from token malfunctioning.

Impact

- Loss of Availability: Operations referencing the storage entries may become unusable after TTL expiration
- Operational Overhead: Recovery of archived data requires additional effort by users or developers, which may involve increased fees and complexity.
- User Experience Risk: End users may perceive contracts as broken or unavailable, reducing trust in the system.

Recommendation

For instance-level items (protocol-level data) such as the token contract, its metadata, configuration, etc., it is recommended to either introduce a dedicated function that calls `extend_ttl()` for all these entries. Alternatively, implement an off-chain service that periodically extends their TTLs.

For persistent items, the TTL can be extended during user operations such as transfers or purchases. For example, on a purchase, the TTL for all entries related to the buyer can be extended. Whereas, on transfers, TTLs for entries related to both the sender and receiver can be extended.

Developers Response

Developers have fixed the issue at commit 7fb26aa.

5.1.8 V-VSPR-VUL-008: Clawback mode cannot be disabled

Severity	Low	**Commit**	1edcfdb
Type	Logic Error,Usability Issue	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:262-263`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/4/`, 15eb9de		

Description

The token configuration includes a `clawback_mode` (or similarly named) field that implies the clawback feature can be **enabled or disabled** depending on issuer or admin preference. However, the current implementation provides **no mechanism** to turn clawback **off** once it is enabled.

This results in a mismatch between the configuration model and the actual contract behavior: the configuration suggests a two-state toggle (`on / off`), but the contract only supports a single permanent state (`on`).

Impact

Administrators cannot disable clawback even if required, e.g., due to changes in legal, regulatory, or business rules.

Recommendation

Introduce a mechanism that allows authorized entities (e.g., issuer or admins) to disable clawback functionality after initialization.

Developers Response

Developers have fixed the issue at commit 15eb9de.

5.1.9 V-VSPR-VUL-009: Missing beneficiary parameter in purchase flow

Severity	Warning	**Commit**	1edcfdb
Type	Usability Issue	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:363-364`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/1`, 115da8b		

Description

The protocol currently uses the buyer's address as the purchase beneficiary, implicitly assuming that the address initiating the purchase is also the address intended to receive the purchased tokens. The purchase function does not expose a separate `beneficiary` argument that would allow the caller to buy tokens on behalf of another address.

In most token-sale or deposit-based protocols, a dedicated `beneficiary` parameter is provided to support flexible integrations. This enables scenarios where tokens are purchased from one wallet (e.g., an exchange, aggregator, or automated system) while being delivered directly to another wallet.

Impact

The lack of a beneficiary parameter limits composability and may complicate integration with external systems such as exchanges, aggregators, and smart-contract wallets. These systems often require the ability to separate the paying address from the receiving address. Without this flexibility, third-party integrations may need additional wrapping logic or may not be feasible at all.

Recommendation

Introduce an explicit `beneficiary` parameter in the purchase function. This parameter should specify the recipient of the purchased tokens independently of the caller. Validation rules (e.g., non-zero address, optional restrictions) should be applied as appropriate.

Developers Response

Developers have fixed the issue at commit 115da8b.

5.1.10 V-VSPR-VUL-010: Name, Symbol, and Home Domain can be empty strings

Severity	Warning	**Commit**	1edcfdb
Type	Data Validation	**Status**	Fixed
Location(s)	`contracts/token/src/lib.rs:None`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/2`, 2cf58b3		

Description

It was identified that it is possible to provide an empty string for the `name`, `symbol`, and `home_domain` metadata fields when initializing the contract.

Impact

Having invalid or empty metadata can create regulatory and compliance issues, as well as cause confusion for users.

Recommendation

It is recommended to implement string validations such that empty strings are not accepted.

Developers Response

Developers have fixed the issue at commit 2cf58b3.

5.1.11 V-VSPR-VUL-011: Maintainability Issues

Severity	Info	**Commit**	1edcfdb
Type	Maintainability	**Status**	Fixed
Location(s)	`contracts/token/src/` ▶ `lib.rs` ▶ `test.rs`		
Confirmed Fix At	`https://github.com/The-Brookes-Project/soroban-sc/pull/3/`, `https://github.com/The-Brookes-Project/soroban-sc/pull/3/`, ec1cdd4, e06732a		

Description

During the review, analysts identified several aspects of the code that may hinder future maintainability and increase the likelihood of bugs:

- The `add_admin()` function re-uses a block of code same as the `is_admin()` function, to check wether the `new_admin` is already an admin. It is recommended to use the `is_admin()` function instead.
- The use of numeric literals as error codes, rather than employing named constants, negatively impacts the maintainability of the code. It is recommended to replace these literals with appropriately defined named constants to enhance code readability and facilitate future modifications.
- `ContractConfig.authoriazation_revocable` is not being used in the codebase.
- The comments in `test_initialize` indicate that a price of `100_000` is intended to represent 0.1 USDC, implying Stellar USDC is being treated as a 6-decimal asset similar to Ethereum USDC, whereas the on-chain contract reports 7 decimals (making `100_000` equal to 0.01 USDC); although current price calculations are unaffected and function correctly, it is recommended to align the comments, price value, and `decimals` configuration with the actual 7-decimal behavior to ensure consistency and avoid confusion.

Recommendation

Implement the proposed recommendations to improve the code maintainability.

Developers Response

Developers have fixed the issue at commit ec1cdd4 and e06732a.