Security Audit Report

Unipay

v1.0

April 4, 2025

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Unipay to perform a security audit of Security audit of the Unipay Solana contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/UnipayFI/stablecoin
Commit	`e6300e337ce17bc428e99dd2e50761cdcff6faa5`
Scope	All contracts were in scope.
Fixes verified at commit	`f140d31124eebe463d1052337333c7332b92434d` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes, such as additional features, have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Unipay's USDu is a fully collateralized stablecoin. Users can stake USDu to earn sUSDu, a reward-bearing token. The system, built on Solana, uses programs to manage minting, staking, and unstaking (with a cooldown period) in a transparent manner.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	While generally readable, the code contains some dead code and could benefit from more in-line documentation to design choices.
Level of documentation	Medium	Documentation covers most important functionality. However, improvements are needed, particularly for the vault and guardian programs.
Test coverage	Low	Test coverage is currently low at 11.34%, covering only basic functionality. Comprehensive testing, including edge cases, is strongly recommended to ensure the codebase's robustness.

Summary of Findings

No	Description	Severity	Status
1	SUSDU unstaking allows for denial-of-service attack	Critical	Resolved
2	First depositor inflation attack in `stake_usdu_mint_susdu`	Critical	Resolved
3	Incorrect USDU supply tracking during emergency withdrawal	Major	Resolved
4	Inability to remove users from blacklist	Major	Resolved
5	Undercollateralization risk due to fee-on-transfer collateral tokens	Major	Resolved
6	Address-based blacklist can be bypassed token transfers	Major	Resolved
7	Insufficient blacklist checks allow bypass in multiple functions	Major	Resolved
8	Lack of check for total USDU in active cooldowns may lead to unexpected fund loss	Minor	Acknowledged
9	Vault drain risk due to lack of price oracle to determine the actual value of the collateral	Minor	Acknowledged
10	Lack of access controls for initialization instructions	Minor	Acknowledged
11	`max_deposit` is an overflow check instead of configurable amount	Minor	Acknowledged
12	`min_shares` check prevents final depositors from withdrawing	Minor	Acknowledged
13	Irreversible admin role in access registry	Minor	Resolved
14	Unchecked `access_registry` in vault configuration	Minor	Resolved
15	Missing validation for `cooldown_duration` allows locking the vault	Minor	Resolved
16	Admin is a single point of failure	Minor	Acknowledged
17	SUSDU unstaking resets cooldown timer and increases amount under cooldown	Minor	Acknowledged

18	Unchecked addition and subtraction operations	Informational	Resolved
19	Missing implementation for ATA freezing in USDU and SUSDU	Informational	Resolved
20	Collateral not transferred to vault account after minting USDU	Informational	Resolved
21	Redistribution process lacks partial fund redistribution capability	Informational	Acknowledged
22	`adjust_blacklist` allows blacklisting privileged accounts	Informational	Acknowledged
23	Identical metadata URIs for USDU and SUSDU tokens	Informational	Acknowledged
24	Missing zero-value checks	Informational	Resolved
25	Missing event emissions in critical functions	Informational	Resolved
26	Precision loss due to rounding down in `convert_to_shares` and `convert_to_assets`	Informational	Acknowledged
27	Missing checks for correct SUSDU and USDU token address.	Informational	Resolved
28	Missing balance checks before `transfer_checked`	Informational	Acknowledged
29	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. SUSDU unstaking allows for denial-of-service attack

Severity: Critical

In `programs/vault/src/instructions/susdu/unstake_susdu.rs:201-205`, the `unstake_susdu` instruction allows any caller holding `SUSDU` to reset another user's `cooldown` PDA account because the program does not verify the owner before updating the existing account.

An attacker can repeatedly call `unstake_susdu` with themselves as caller but specify a victim as the `receiver`, thus perpetually extending the victim's cooldown timer and preventing them from withdrawing their USDU.

Recommendation

We recommend implementing either a check that the caller owns the existing `cooldown` account or changing the cooldown account derivation to include both caller and receiver.

Status: Resolved

2. First depositor inflation attack in `stake_usdu_mint_susdu`

Severity: Critical

In the `stake_usdu_mint_susdu` instruction, during the calculation of SUSDU to mint for a given USDU deposit the `convert_to_shares` function is used by `preview_deposit` to determine the SUSDU amount: `shares = (assets * (total_shares + 1)) / (total_assets + 1)`.

Where:

- `assets` is the amount of USDU being deposited.
- `total_shares` is the total supply of SUSDU before the current deposit.
- `total_assets` is the total amount of USDU in the `vault_stake_pool_usdu_token_account` before the current deposit.

When the pool is empty (first deposit), both `total_shares` and `total_assets` are 0. The formula simplifies to `shares = (assets * 1) / 1 = assets`.

This allows for the following attack:

1. The attacker deposits a tiny amount of USDU (e.g., 1 wei). They receive an equal amount of SUSDU.

2. The attacker directly transfers a large amount of USDU to the `vault_stake_pool_usdu_token_account` without invoking `stake_usdu_mint_susdu`. This increases `total_assets` but not `total_shares`.
3. Since the attacker holds nearly all of the SUSDU, and `total_assets` is now significantly larger, they can withdraw a disproportionately large amount of USDU, effectively stealing the donated funds.

The current `check_min_shares` function only checks if the total SUSDU supply (`total_shares`) is greater than or equal to `MIN_SHARES`. It does not enforce that the first `MIN_SHARES` are minted to a burn address or otherwise locked. The attacker's small initial deposit, followed by the donation, will still result in `total_shares` being less than `MIN_SHARES` until after they have already received their initial SUSDU. The check occurs too late to prevent the attack.

Additionally, the current implementation of `total_shares` is not correct. The `susdu_config` account is not properly reloaded to reflect the updates from the `mint_susdu` CPI.

Recommendation

We recommend implementing the standard mitigation of minting the first `MIN_SHARES` (e.g., 1000, or 10^6, depending on the token's decimals) of SUSDU to the zero address (or a dedicated burn address) upon the first deposit.

Status: Resolved

3. Incorrect USDU supply tracking during emergency withdrawal

Severity: Major

In `programs/vault/src/instructions/admin/emergency.rs`, the `process_emergency_withdraw_vault_stake_pool_usdu` instruction withdraws USDU tokens from the stake pool but does not update the `total_usdu_supply` field in `vault_config`. This field is critical for multiple calculations, including `VaultConfig::total_assets` and the `convert_to_shares` and `convert_to_assets` functions. Since partial withdrawals are allowed, failing to decrement `total_usdu_supply` causes ongoing miscalculations in user share allocations and can result in liquidity imbalances.

Recommendation

We recommend updating `total_usdu_supply` immediately before tokens are withdrawn, subtracting the amount removed from the stake pool to ensure accurate asset calculations.

Status: Resolved

4. Inability to remove users from blacklist

Severity: Major

In `programs/vault/src/instructions/admin/adjust_blacklist.rs`, the `process_adjust_blacklist` function can only add users to the blacklist. There is no corresponding function to remove a user from the blacklist. Once a user is blacklisted, they are permanently blacklisted, unless the program itself is upgraded. This is a significant operational and potentially legal issue.

The check `if ctx.accounts.blacklist_state.is_initialized { return ... }` combined with `init_if_needed`, is the core of the problem. If the `blacklist_state` account already exists (meaning the user has been blacklisted before), this condition is true, and the function returns an error (`BlacklistStateAlreadyInitialized`). The code never reaches the lines that would update `is_frozen_susdu` or `is_frozen_usdu`.

Recommendation

We recommend creating a new, separate instruction, `remove_from_blacklist`. This instruction should not use `init_if_needed` on the `blacklist_state` account. It would load the account, verify the caller has the `GrandMaster` role, and then either close the account or set flags (`is_initialized`, `is_frozen_susdu`, `is_frozen_usdu`) to `false`.

Status: Resolved

5. Undercollateralization risk due to fee-on-transfer collateral tokens

Severity: Major

The `deposit_collateral_mint_usdu` instruction in the `vault` program can lead to undercollateralization if a fee-on-transfer token is used as collateral.

The `transfer_checked` instruction (used to transfer the collateral) only verifies that the transfer is correctly formatted according to the token's decimals. It does not check if the destination account receives the expected amount after any potential transfer fees are deducted by the token itself.

This means the vault could receive less collateral than expected, leading to a situation where the minted USDU is not fully backed. While the code checks for a delegate and its amount, this is insufficient to prevent this issue.

The same issue exists in the `redeem_usdu_withdraw_collateral` instruction, where the benefactor might receive less collateral than expected due to fees.

This could lead to vault undercollateralization, potentially resulting in losses for USDU holders if collateral value falls below minted USDU value and users cannot redeem their USDU for the full expected collateral amount.

Although the documentation states only approved collateral types (SOL, ETH, BTC) are accepted, and a whitelisting feature partially exists in `programs/vault/src/utils/token.rs`, it is not utilized.

Recommendation

We recommend implementing a strict whitelist of allowed collateral tokens, restricting it to approved tokens without fee-on-transfer mechanisms or other potentially dangerous extensions (like rebasing).

Status: Resolved

6. Address-based blacklist can be bypassed token transfers

Severity: Major

In the `process_adjust_blacklist` instruction, users are blocked based on their account address by creating a `blacklist_state` storing their account address along with other relevant parameters.

As the restriction is not applied at the SPL Token2022 level, this design flaw does not prevent the users from transferring the tokens to another address for which `blacklist_state` is not initialized. This allows continued participation of blacklisted users in disallowed operations, thus nullifying the intent of blacklisting.

Recommendation

We recommend implementing the SPL Token2022 transfer hook extension and attaching it to USDU and SUSDU to ensure that all token transfers invoke a blacklist check for both sender and receiver, preventing blocked users from moving funds to new non-blacklisted accounts or receiving funds from non-blacklisted accounts.

Status: Resolved

7. Insufficient blacklist checks allow bypass in multiple functions

Severity: Major

Several vault functions have inadequate blacklist checks, allowing users to bypass restrictions. The issue arises because the caller is not checked against the `blacklist_state.owner`. Consequently, any `BlacklistState` account can be specified. We classify this as a major issue because the functions mentioned below do not require the caller to have privileged roles.

Affected locations:

- `programs/vault/src/instructions/susdu/withdraw_usdu.rs:64`
- `programs/vault/src/instructions/susdu/unstake_susdu.rs:130`
- `programs/vault/src/instructions/susdu/stake_usdu_mint_susdu.rs:116`

Additionally, this flaw is present in the `redistribute_locked` instruction, although it is less severe due to requiring the caller to have a `GrandMaster` role.

Recommendation

We recommend loading the specific `BlacklistState` account to the designated receiver.

Status: Resolved

8. Lack of check for total USDU in active cooldowns may lead to unexpected fund loss

Severity: Minor

The protocol does not verify the total amount of USDU in active cooldowns before processing withdrawals. While this does not impact global accounting (e.g., `total_usdu_supply`), it might result in users with active cooldowns losing their cooldown-protected funds without an update to their cooldown accounts, leading to unexpected behavior.

Recommendation

We recommend implementing a logic for the distribution of `cooldown.underlying_token_amount` to designated receivers, followed by transferring an additional amount to the protocol's intended account.

Status: Acknowledged

9. Vault drain risk due to lack of price oracle to determine the actual value of the collateral

Severity: Minor

In `programs/vault/src/instructions/usdu/deposit_collateral_mint_usdu.rs`, the protocol assumes a 1:1 relationship between USDU and collateral. It does not use a price oracle to determine the actual value of the collateral being withdrawn. If the collateral's value drops significantly, the user could redeem USDU and withdraw a disproportionately large amount of collateral, draining the vault.

The `process_deposit_collateral_mint_usdu` instruction mints USDU based solely on the quantity of collateral deposited, without considering its value. This must be addressed by incorporating a price oracle and enforcing a collateralization ratio.

We classify this issue as minor since this is a trusted instruction and can only be called by `Role::CollateralDepositor`.

Recommendation

We recommend implementing proper collateral ratio requirements or clarifying the functionality.

Status: Acknowledged

10. Lack of access controls for initialization instructions

Severity: Minor

The current implementation lacks role validations for initialization instructions, potentially allowing any address to initialize programs and assume administrative control. If initialization is not executed in a single transaction by the Unipay team, an attacker could front-run legitimate initialization transactions and seize control. We classify this issue as minor since if this were to occur it would be apparent and the protocol could be redeployed and initialized again.

Affected locations:

- `process_init_vault_config` in `programs/vault/src/instructions/admin/init_vault.rs:184`
- `process_init_vault_state` in `programs/vault/src/instructions/admin/init_vault.rs:201`
- `process_init_access_registry` in `programs/guardian/src/instructions/admin/init_access_registry.rs`
- `process_init_config` in `programs/susdu/src/instructions/admin/init_config.rs:32`

Recommendation

We recommend restricting initialization functions to the program deployer or a designated administrator account.

Status: Acknowledged

11. `max_deposit` is an overflow check instead of configurable amount

Severity: Minor

The `max_deposit` function in `programs/vault/src/instructions/susdu/stake_usdu_mint_susdu.rs:181` defines the maximum deposit as `u64::MAX` to prevent overflow errors, but it does not consider the sum of the total deposit and the new deposit. This effectively functions as an overflow check rather than a configurable maximum deposit limit. Since overflow checks are already enabled in `Cargo.toml`, the `max_deposit` check in `stake_usdu_mint_susdu` is likely intended to be a maximum deposit limit, not a redundant overflow check.

Recommendation

We recommend implementing a configurable `max_deposit` value within `VaultConfig` and utilizing it for the check. Additionally, verify that `total deposit + new deposit <= u64::MAX` to ensure proper handling of deposit limits.

Status: Acknowledged

12. `min_shares` check prevents final depositors from withdrawing

Severity: Minor

The `min_shares` implementation in `programs/vault/src/instructions/susdu/unstake_susdu.rs:264`, intended to prevent share manipulation during the initial deposit, inadvertently blocks later depositors from withdrawing. While improbable, this scenario could occur during a bank run, preventing some users from accessing their funds.

Recommendation

We recommend removing the `min_shares` check from the `unstake` functionality within `programs/vault/src/instructions/susdu/unstake_susdu.rs:264`.

Status: Acknowledged

13. Irreversible admin role in access registry

Severity: Minor

In `programs/guardian/src/instructions/admin/init_access_registry.rs` the `access_registry` admin, once initialized, cannot be changed or transferred. If the admin's private key is compromised, the entire protocol is at risk, as `access_registry.admin` has full control over all roles.

Recommendation

We recommend implementing a secure admin transfer mechanism that allows the current admin to transfer ownership to a new address. This will help mitigate the risk of a single point of failure and enhance the security of the access registry.

Status: Resolved

14. Unchecked `access_registry` in vault configuration

Severity: Minor

The `InitVaultConfig` struct in `programs/vault/src/instructions/admin/init_vault.rs` does not properly derive the `access_registry` account, unlike other parts of the codebase. This can lead to misconfigurations if an incorrect account is provided. Please note that the impact of this is limited if the instruction is properly permissioned, as mentioned in Lack of access controls for initialization instructions.

Recommendation

We recommend deriving the account as follows:

```
#[account(
        seeds = [ACCESS_REGISTRY_SEED],
        seeds::program = guardian::id(),
        bump = access_registry.bump,
    )]
    pub access_registry: Box<Account<'info, AccessRegistry>>,
```

Status: Resolved

15. Missing validation for `cooldown_duration` allows locking the vault

Severity: Minor

In `programs/vault/src/instructions/admin/init_vault.rs:184`, and `programs/vault/src/instructions/admin/adjust_cooldown.rs:32`, the `cooldown_duration` is taken as an input parameter without any validation. Despite documentation specifying a 7-day cooldown, there is no check to ensure it falls within a reasonable range. Setting `cooldown_duration` to an extremely large value could effectively lock the vault, while setting it to zero would disable the mechanism.

Recommendation

We recommend implementing input validation for a `cooldown_duration` in the `process_init_vault_config` and `process_adjust_cooldown` instructions, including a check for maximum duration.

Status: Resolved

16. Admin is a single point of failure

Severity: Minor

The `AccessRegistry` is initialized with a single admin account. Compromise of this account's private key grants full control over the access registry, posing a significant security risk. This effectively creates a master key to the entire system; its compromise can lead to a complete loss of control and potential asset theft. The `has_role` function (`programs/guardian/src/utils.rs`), used throughout the codebase to verify these roles, makes the `AccessRegistry` and its single admin the central security point.

Additionally, the `GrandMaster` role acts as a single point of failure. Implementing multi-signature control or decentralized governance for emergency actions should be considered.

Recommendation

We recommend implementing a multi-signature mechanism for the admin role to mitigate the single point of failure in both the `GrandMaster` role and the `AccessRegistry`.

Status: Acknowledged

17. SUSDU unstaking resets cooldown timer and increases amount under cooldown

Severity: Minor

In `programs/vault/src/instructions/susdu/unstake_susdu.rs:202-204`, when a user unstakes SUSDU, if a cooldown is already in progress and the user unstakes additional SUSDU, the cooldown timer is reset, and the total amount under cooldown is

incremented. This means the cooldown restarts with every `unstake_susdu` attempt, even when a cooldown is nearing completion.

Recommendation

We recommend modifying the `unstake_susdu` instruction to prevent resetting the cooldown timer when additional SUSDU is unstaked while a cooldown is active. Instead, additional unstaked amounts should be added to the existing cooldown queue without restarting the timer.

Status: Acknowledged

18. Unchecked addition and subtraction operations

Severity: Informational

The codebase contains several instances of unchecked addition and subtraction. Implementing explicit overflow handling is a best practice that enhances error management and prevents unexpected panics.

Identified instances:

- `programs/usdu/src/instructions/mint_usdu.rs:71`
- `programs/susdu/src/instructions/mint_susdu.rs:74`
- `programs/susdu/src/instructions/admin/redistribute_susdu.rs:82`

Recommendation

We recommend replacing the unchecked addition and subtraction operations in the specified locations with checked equivalents to handle potential overflows.

Status: Resolved

19. Missing implementation for ATA freezing in USDU and SUSDU

Severity: Informational

In `programs/usdu/src/instructions/create_usdu.rs:53` and `programs/susdu/src/instructions/create_susdu.rs:68`, the `freeze_authority` is assigned as the `usdu_token` and `susdu_token` respectively, but there is no implementation of an instruction to to utilize it.

Recommendation

We recommend either implementing the necessary logic to enforce token freezes if this functionality is desired or removing the unused `freeze_authority` variables to avoid confusion and maintain code clarity if freezing is not needed.

Status: Resolved

20. Collateral not transferred to vault account after minting USDU

Severity: Informational

In `programs/vault/src/instructions/usdu/deposit_collateral_mint_usdu.rs:136-149`, the `deposit_collateral_mint_usdu` instruction transfers the collateral from the user's (`benefactor`) account to the `fund_collateral_token_account`.

Although the `vault_collateral_token_account` is initialized (if needed), the collateral remains in the `fund` account. There is no subsequent transfer within the provided codebase to move the collateral to the `vault_collateral_token_account`.

The `redeem_usdu_withdraw_collateral` instruction confirms that the `fund` account holds the collateral, as it transfers from this account during redemption. This means the USDU is minted before the collateral is secured in the vault's designated collateral account.

While the `fund` account might be under the control of a trusted entity (e.g., a custody wallet), this is not explicitly confirmed in the code.

Recommendation

We recommend modifying the `deposit_collateral_mint_usdu` instruction to transfer the collateral directly to the `vault_collateral_token_account`, removing the `fund_collateral_token_account` from the deposit process. Alternatively, remove unused `vault_collateral_token_account`.

Status: Resolved

21. Redistribution process lacks partial fund redistribution capability

Severity: Informational

In `programs/vault/src/instructions/admin/redistribute_locked.rs`, the `process_redistribute_locked` instruction transfers the entire balance of the `locked_susdu_token_account`. There is no mechanism to redistribute only a portion of the locked funds.

Recommendation

We recommend modifying the `process_redistribute_locked` instruction to allow for the redistribution of a user-specified amount of locked funds, rather than always transferring the entire balance.

Status: Acknowledged

22. `adjust_blacklist` allows blacklisting privileged accounts

Severity: Informational

The `adjust_blacklist` functionality permits the blacklisting of accounts holding privileged roles. This capability could disrupt vault operations.

Recommendation

We recommend implementing checks within `process_adjust_blacklist` to prevent blacklisting accounts that hold specific roles.

Status: Acknowledged

23. Identical metadata URIs for USDU and SUSDU tokens

Severity: Informational

Both the USDU and SUSDU tokens currently utilize the same URI for their metadata:

https://bafybeib5rbwqc5hj52hhc6k6g4c5qfhlq2jkkeujypc3okvm7dqoypgcku.ipfs.w3s.link/usdu.png

While not a security concern, this can cause user confusion, as the tokens might be perceived as identical despite serving distinct purposes.

Recommendation

We recommend assigning unique URIs for each token's metadata to clearly differentiate them.

Status: Acknowledged

24. Missing zero-value checks

Severity: Informational

Several instructions lack explicit checks for zero-value inputs, potentially leading to unintended behavior.

- `process_redistribute_susdu` in `programs/vault/src/instructions/admin/redistribute_susdu.rs` does not validate the `amount` parameter.
- `redeem_usdu_withdraw_collateral in programs/vault/src/instructions/usdu/redeem_usdu_withdraw_collateral.rs` should ensure both `collateral_amount` and `usdu_amount` are greater than zero.
- `deposit_collateral_mint_usdu` in `programs/vault/src/instructions/usdu/deposit_collateral_mint_usdu.rs` lacks input validation for `collateral_amount` and `usdu_amount`

Recommendation

We recommend incorporating explicit checks for zero values in the `amount`, `collateral_amount`, and `usdu_amount` parameters within the respective instructions.

Status: Resolved

25. Missing event emissions in critical functions

Severity: Informational

Several critical functions lack event emissions, hindering transaction tracking and debugging. The following functions should emit events to log important actions:

- `process_redistribute_susdu` in `programs/vault/src/instructions/admin/redistribute_susdu.rs`
- `init_vault` instructions in `programs/vault/src/instructions/admin/init_vault.rs`
- `distribute_usdu_reward` in `programs/vault/src/instructions/admin/distribute_usdu_reward.rs` (after successful USDU reward distribution)
- Emergency instructions (e.g., emergency withdrawals) in relevant files
- `redistribute_locked` instruction in `programs/vault/src/instructions/admin/redistribute_locked.rs`
- `withdraw_usdu` in `programs/vault/src/instructions/susdu/withdraw_usdu.rs`

Recommendation

We recommend implementing event emissions in the identified functions.

Status: Resolved

26. Precision loss due to rounding down in `convert_to_shares` and `convert_to_assets`

Severity: Informational

In `programs/vault/src/state/config.rs`, the `convert_to_shares` function calculates the amount of SUSDU (shares) to mint based on the deposited USDU (assets) and the current total supply of SUSDU and USDU.

The use of `Rounding::Floor` means that the result of the division (`numerator / denominator`) is always rounded down to the nearest whole number. This guarantees that the vault will never mint more SUSDU than it should, preventing inflation.

However, it also means there will almost always be a small remainder – a fractional amount of USDU that is not converted to SUSDU.

The `convert_to_assets` function in `programs/vault/src/state/config.rs` uses the same rounding logic as `convert_to_shares`, and therefore, the same potential for precision loss applies.

Recommendation

We recommend either acknowledging the precision loss and clearly document it for users or accumulating dust. When the accumulated dust reaches a certain threshold, mint the corresponding SUSDU and distribute it to SUSDU holders proportionally or burn it.

Status: Acknowledged

27. Missing checks for correct SUSDU and USDU token address.

Severity: Informational

Several critical instructions do not verify that the provided token addresses match the addresses stored in their respective configurations. This could lead to unintended actions with incorrect tokens.

Affected instructions:

- `process_mint_susdu` in `programs/susdu/src/instruction/mint_susdu.rs`
- `process_mint_usdu` in `programs/usdu/src/instruction/mint_susdu.rs`
- `process_redeem_susdu` in `programs/susdu/src/instruction/redeem_susdu.rs`
- `process_redeem_usdu` in `programs/usdu/src/instruction/redeem_susdu.rs`
- `process_redistribute_susdu` in

programs/vault/src/instructions/admin/redistribute_susdu.rs

- process_distribute_usdu_reward in programs/vault/src/instructions/admin/distribute_usdu_reward.rs
- process_init_vault in programs/vault/src/instructions/admin/init_vault.rs
- process_redistribute_locked in programs/vault/src/instructions/admin/redistribute_locked.rs
- process_unstake_susdu in programs/vault/src/instructions/susdu/unstake_susdu.rs

Recommendation

We recommend implementing the following checks in the affected instructions:

- susdu_token.key == susdu_config.susdu_token
- usdu_token.key == usdu_config.usdu_token

Status: Resolved

28. Missing balance checks before `transfer_checked`

Severity: Informational

In several instructions, check for the sufficient balance in the sender's ATA is missing before invoking `transfer_checked`. Affected instructions:

- process_stake_usdu_mint_susdu in programs/vault/src/instructions/susdu/stake_usdu_mint_susdu.rs
- process_redistribute_susdu in programs/susdu/src/instructions/admin/redistribute_susdu.rs
- process_distribute_usdu_reward in programs/vault/src/instructions/admin/distribute_usdu_reward.rs
- process_unstake_susdu in programs/vault/src/instructions/susdu/unstake_susdu.rs
- process_withdraw_usdu in programs/vault/src/instructions/susdu/withdraw_usdu.rs
- process_deposit_collateral_mint_usdu in programs/vault/src/instructions/usdu/deposit_collateral_mint_usdu.rs
- All instructions in programs/vault/src/instructions/admin/emergency.rs

Recommendation

We recommend adding balance checks in the affected instructions to ensure that the sender's account has sufficient funds before attempting a token transfer. This will improve the reliability of these operations and prevent unexpected failures.

Status: Resolved

29. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

The following are some recommendations to improve the overall code quality and readability:

- We recommend removing the unused, commented-out code block in `programs/vault/src/state/config.rs:165-175` to prevent potential unintended functionality in future updates.
- In `programs/susdu/src/instructions/admin/redistribute_susdu.rs`, the core logic involves either burning or transferring SUSDU tokens from a "locked" account based on the presence of a receiver. This design choice should be clearly documented. The current naming ("redistribute") might be misleading as it encompasses burning. We recommend renaming the `redistribute_susdu` function or providing comprehensive documentation to clarify its burning functionality.

Status: Resolved