Securitize Solana Vault Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Farouk

Al-Qa-Qa

May 23, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

The Securitize Vault is a Solana-based implementation of a tokenized vault system inspired by the ERC4626 standard. It enables users to deposit DS Tokens—regulated digital securities—and receive Vault Tokens in return, which are SPL tokens representing a share of the underlying assets. These Vault Tokens can later be redeemed for the original DS Tokens or, alternatively, liquidated into stablecoins via external liquidity providers.

The architecture is modular and extensible, implemented using the Anchor framework and organized into multiple programs. The primary `sc-vault` program contains the core logic for asset management, while additional interface programs define interactions with external NAV providers and redemption systems. Mock programs for both NAV and redemption are included to facilitate testing and simulation.

Key components include a persistent `VaultState` account that holds configuration data, manages authorized roles (admins, operators, liquidators), and tracks links to NAV and redemption providers. The NAV provider is responsible for supplying accurate asset valuations used in share conversions, and the redemption interface allows for flexible strategies in converting Vault Tokens back into DS Tokens.

The protocol supports role-based access control:

- **Admins** configure vault parameters and permissions.
- **Operators** manage deposits and redemptions.
- **Liquidators** convert Vault Tokens into stablecoins in under-collateralized scenarios.
- **Users** interact with view-only endpoints for queries and conversions.

The system is built with compatibility for both Token Program v1 and Token-2022, ensuring compliance with advanced token extension requirements like transfer hooks for KYC enforcement. Instructions cover the full lifecycle of the vault—from initialization and configuration to asset deposit, share redemption, and liquidation.

5 Audit Scope

Cyfrin conducted an audit of the Securitize Solana Vault based on the code present in the repository commit hash e7244a8. The following files construct the scope of the audit:

```
programs
 nav-provider-interface
    src
        lib.rs
 redemption-interface
    src
        lib.rs
 sc-vault
     src
         constants.rs
         errors.rs
         events.rs
         instructions
            admin
               add_liquidator.rs
               add_operator.rs
               add_share_metadata.rs
               change_admin.rs
               initalize.rs
               mod.rs
               pause.rs
               revoke_liquidator.rs
               revoke_operator.rs
               set_liquidation_open_to_public.rs
               unpause.rs
               update_nav_provider.rs
            liquidator
               liquidate.rs
               mod.rs
            mod.rs
            operator
               accounts.rs
               deposit.rs
               mod.rs
               redeem.rs
            user
                accounts.rs
                convert_to_assets.rs
                convert_to_shares.rs
                get_share_value.rs
                is_admin.rs
                is_liquidator.rs
                is_operator.rs
                mod.rs
         lib.rs
         states
            mod.rs
            vault_counter.rs
            vault_state.rs
         utils
             conversions.rs
             get_rate.rs
             get_vault_authority_signer.rs
             math.rs
             mod.rs
             transfer_from_vault.rs
```

6 Executive Summary

Over the course of 6 days, the Cyfrin team conducted an audit on the Securitize Solana Vault smart contracts provided by Securitize. In this period, a total of 10 issues were found.

This audit focused on the Securitize Vault, a Solana-based adaptation of the ERC4626 standard, designed to enable users to wrap DS Tokens into Vault Tokens—SPL tokens representing claims on the underlying assets. Built with the Anchor framework, the system extends the original Solidity implementation by supporting liquidations into stablecoins through liquidity providers, while also integrating Solana-specific features such as Token-2022 compliance checks and program-derived authority accounts.

The protocol incorporates several security measures, including controlled token flows, deterministic signer logic via PDAs, and reliance on external NAV providers like Redstone to ensure accurate share valuation. The design aims to maintain regulatory compliance while enabling flexible interaction with DeFi protocols.

During the audit, eleven issues were identified, ranging from precision manipulation due to unchecked mint configuration to slippage handling and inconsistent validation logic. While one issue was classified as high risk and a few as medium risk, most were of low or informational severity. These findings highlight areas for improved robustness in mint handling, slippage checks, and account validation. Overall, the protocol demonstrates a solid security posture with some critical areas requiring attention.

Summary

Project Name	Securitize Solana Vault
Repository	bc-solana-vault-sc
Commit	e7244a875189...
Audit Timeline	May 12th - May 19th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	5
Informational	3
Gas Optimizations	0
Total Issues	10

Summary of Findings

[H-1] Precision Manipulation Due to Missing Mint Check	Resolved
[M-1] Missing Slippage Check on `liquidation_amount` in Redemption-Enabled Vaults	Acknowledged

[L-1] Off-By-One Error in Liquidator and Operator Capacity Check	Resolved
[L-2] Unsafe Addition in `convert_to_shares` Causes View Function Inaccuracy	Resolved
[L-3] Strict Comparison in Slippage Check Incorrectly Blocks Valid Redemptions	Resolved
[L-4] Incorrect Splitting of `remaining_accounts` Causes Misrouting Between NAV and Redemption Accounts	Resolved
[L-5] `Liquidate` Event Emits Shares and Assets in Wrong Order	Resolved
[I-1] Invalid Zero Rate Not Rejected During Deposit	Resolved
[I-2] Redundant `vault_authority_signer` passed to `BurnChecked` in `redeem` and `liquidate`	Resolved
[I-3] Lack of `mut` on `liquidation_token_mint` restricts redemption flexibility	Resolved

7 Findings

7.1 High Risk

7.1.1 Precision Manipulation Due to Missing Mint Check

Description: `deposit_handler` chooses the conversion precision (`conv_decimals`) as:

```
let conv_decimals = ctx
    .accounts
    .liquidation_token_mint
    .as_ref()
    .map(|mint| mint.decimals)          // if Some  use its decimals
    .unwrap_or(ctx.accounts.asset_mint.decimals);
```

The account context *documents* that `liquidation_token_mint` and `liquidation_token_vault` must appear together, but there is **no runtime constraint** enforcing it. An operator can therefore:

1. Pass `liquidation_token_mint = Some` with any mint they control (e.g., 0 decimals).
2. Pass `liquidation_token_vault = None`, bypassing the only check that references the vault.

Because `conv_decimals` now equals `0`, the denominator inside `convert_to_shares` becomes `10^0 = 1` instead of (say) `10^6`. All arithmetic that scales by this factor is therefore inflated by `10^decimals`, dramatically increasing the number of shares minted for the same asset deposit.

Impact: A privileged operator can mint an arbitrarily large supply of shares at a discount (10^D × cheaper, where **D** is the asset's normal decimals). They can later redeem those shares for real assets, draining the vault and diluting all honest shareholders—effectively a direct loss of user funds.

Proof of Concept:

1. Create a fake SPL mint with `decimals = 0`.
2. Construct a `deposit` instruction where
   - `liquidation_token_mint` → the fake mint account
   - `liquidation_token_vault` → **omitted** (encode as `None`)
   - All other required accounts are valid.
3. Deposit 1 unit of the underlying asset.
4. Observe that `convert_to_shares` uses a factor of `1` instead of `10^asset_decimals`, minting roughly `10^asset_decimals` more shares than intended.

Recommended Mitigation: Enforce account coupling:

```
require!(
    liquidation_token_mint.is_some() == liquidation_token_vault.is_some(),
    ScVaultError::InvalidLiquidationAccounts
);
```

or add an Anchor constraint tying the two options together as the docstring promises.

Securitize: Fixed in b0cabd3.

Cyfrin: Verified

7.2 Medium Risk

7.2.1 Missing Slippage Check on `liquidation_amount` in Redemption-Enabled Vaults

Description: When calling `liquidate_handler()` the liquidator provide the minimum amount of assets he is willing to receive. either assets or liquidation tokens

```
/// ## Arguments
...
/// - `min_output_amount`: An optional minimum output amount to ensure sufficient assets or liquidation
↪  tokens are received.
```

The slippage check is implemented only for assets before checking the type of the Vault weather it support Redemption or not.

```
>>  if let Some(min_output_amount) = min_output_amount {
        require_gt!(
            assets,
            min_output_amount,
            ScVaultError::InsufficientOutputAmount
        );
    }

    ...

    if let Some(ref redemption_program) = ctx.accounts.redemption_program {
        ...
        // Transfer received liquidation tokens to liquidator.
        transfer_from!(
            liquidation_token_vault,                                    // from
            liquidator_liquidation_ata,                          // to
            ctx.accounts.vault_authority,                               // authority
            ctx.accounts.liquidation_token_program.as_ref().unwrap(), // token_program
            liquidation_token_mint,                                     // mint
>>          liquidation_amount,                                         // amount
            liquidation_token_mint.decimals,                            // decimals
            vault_authority_signer                                      // signer
        );
    } else {
        ...
    }
```

As we can see the amount the liquidator receives in case of Redemption is not `assets` value calculated. it is `liquidation_amount` received after redeeming.

This wil result in incorrect slippage, as the liquidator will provide the minimum amount he is willing to receive from `liquidate token`, but the check will be made for `asset` instead.

Impact:

- Liquidator receives less than he wants

Proof of Concept:

- liquidator made the `min_output_amount` as `1000`
- firing `liquidate_handler`
- assets value is `1100` after calculations
- slippage passed
- firing `redemption_program::redeem()`
- liquidation_amount is `900`

- liquidator receives `900` token, although he mentioned he only accepts `1000` or more

Recommended Mitigation:

- Move the liquidation check and transfer it to the `else` block (the Vault that is not supporting Redemption)
- Make another liquidation check aganist `liquidation_amount` for Vaults supporting redemption

Securitize: Acknowledged, it's acceptable from our side since it matches the behavior in the EVM version and we don't intend to change it. However, the slippage documentation has been clarified in 56c8f9e.

7.3 Low Risk

7.3.1 Off-By-One Error in Liquidator and Operator Capacity Check

Description: When adding new liquidators or operators, the check is performed to check that the MAX_LENGTH is greater than or equal the current length. But after this we push the new element.

- bc-solana-vault-sc/programs/sc-vault/src/instructions/admin/add_liquidator.rs
- bc-solana-vault-sc/programs/sc-vault/src/instructions/admin/add_operator.rs

```
pub fn add_liquidator_handler(
    ctx: &mut Context<AddLiquidator>,
    new_liquidator: Pubkey,
) -> Result<()> {
    let vault_state = &mut ctx.accounts.vault_state;

>>  require_gte!(
        MAX_LIQUIDATORS,
        vault_state.liquidators.len(),
        ScVaultError::MaxLiquidators
    );
    ...
>>  vault_state.liquidators.push(new_liquidator);
    ...
}
// ----------------
pub fn add_operator_handler(ctx: &mut Context<AddOperator>, new_operator: Pubkey) -> Result<()> {
    let vault_state = &mut ctx.accounts.vault_state;

>>  require_gte!(
        MAX_OPERATORS,
        vault_state.operators.len(),
        ScVaultError::MaxOperators
    );
    ...
>>  vault_state.operators.push(new_operator);
    ...
}
```

If the current length is the same as `MAX_LIQUIDATORS/OPERATORS` the check will pass, as it enforces the MAX to be greater than or equal. But this is incorrect. as if the length is the `MAX` we should prevent adding, as this will result in out of bound array access.

Impact: Incorrect behavior of the Program and getting incorrect error results.

Recommended Mitigation: Consider using `require_gt` instead of `require_gte`.

Securitize: Fixed in 179f8f3.

Cyfrin: Verified

7.3.2 Unsafe Addition in `convert_to_shares` Causes View Function Inaccuracy

Description: When providing `assets` parameter there is no check if the addition of `total_assets` with `asset` will result in a number greater than `u64::MAX` or not.

```
pub fn convert_to_shares(
    assets: u64,
    rate: u64,
    decimals: u8,
    total_assets: u64,
    total_supply: u64,
    rounding: Rounding,
```

```
) -> Result<u64> {
    let liq_token_decimals_factor = 10u64.pow(decimals.into());
    let total_shares_after_deposit = math::mul_div(
        rate,
>>      total_assets + assets,
        liq_token_decimals_factor,
        rounding,
    )?;
    ...
}
```

When calling `convert_to_assets_handler`, the asset is provided as input. it is like a `view` function to check the corresponding shares the operator will take for this amount. and this `assets` are added to `total_assets`.

So if he provided `assets` with value that makes `total_assets + assets` goes greater than `U64::MAX` this will result in overflow, leading to incorrect shares returned value for that amount.

This only affects the `view` function and `convert_to_assets_handler` as `deposit_handler` transfers the assets from the depositer (operator) before it, and since supply is `u64` it will not reach the max.

Impact: Incorrect return values when calling `convert_to_assets_handler` with large asset amount

Recommended Mitigation: Use safe additions in `convert_to_shares`, so that the function revert with overflow if the user provided large asset amount.

Securitize: Fixed in 9189e4c.

Cyfrin: Verified

7.3.3 Strict Comparison in Slippage Check Incorrectly Blocks Valid Redemptions

Description: When checking minimum outcome from the liquidate, we are implementing the check to enforce the returned assets to be more than the minimum amount we put.

bc-solana-vault-sc/programs/sc-vault/src/instructions/liquidator/liquidate.rs#liquidate_handler

```
    if let Some(min_output_amount) = min_output_amount {
>>      require_gt!(
            assets,
            min_output_amount,
            ScVaultError::InsufficientOutputAmount
        );
    }
```

This check is incorrect as if the `assets` equals `min_output_amount`. the tx will revert, but in reaility it should success as the user accepts this value is the minimum value he accepts, but it will be treated as less than desired by the user and revert the tx.

Impact: Reverting liquidations that are supposed to pass.

Recommended Mitigation: Consider adjusting the check to use `require_gt`.

Securitize: Fixed in de507ba.

Cyfrin: Verified

7.3.4 Incorrect Splitting of `remaining_accounts` Causes Misrouting Between NAV and Redemption Accounts

Description: When spliting accounts in `liquidate_handler`, we are assuming that `nav_provider_program` will take the MAX value. where we take the minimum value from the `MAX_NAV_PROVIDER_ACCOUNTS (5)` and the remaining accounts.

bc-solana-vault-sc/programs/sc-vault/src/instructions/liquidator/liquidate.rs#liquidate_handler

```
    let nav_provider_accounts_count = MAX_NAV_PROVIDER_ACCOUNTS.min(ctx.remaining_accounts.len());
    let (nav_provider_accounts, redemption_accounts) =
        ctx.remaining_accounts.split_at(nav_provider_accounts_count);
```

The problem is that in case of `redemption` is activated, it takes at least `4` accounts.

bc-solana-vault-sc/programs/sc-vault/src/constants.rs

```
pub const MIN_REDEMPTION_ACCOUNTS: usize = 4;
```

So if the liquidator is firing in a vault state where it activate the `redemption` and the nav provider only accepts `1` account as rate. this will result in incorrect splitting, and redemption accounts will goes to nav_provider instead.

Impact:

- Reverting liquidate function, resulting in inapility to do the liquidation process

Proof of Concept:

- Vault state activate `Redemption`, with minimum accounts required (4)
- Vault state has `NAV Provider program` accepting only one account (minimum).
- The liquidator fired liquidate putting remaining accounts as following: first one is `nav_provider_state`, and the other `4` are for redemption. i.e total 5.
- `nav_provider_accounts_count` will be 5.min(5), i.e 5
- All `5` accounts will goes to `nav_provider_accounts` and no account will goes to `redemption_accounts`
- This will lead to revert the tx when checking `redemption_accounts` aganist minimum as they should be at least of 4 length

Recommended Mitigation: Provide the split index as input, so that for `nav_providers` that don't need all `5` accounts, you can make them take the accounts they need. and make redemption accounts with correct values

Securitize: Fixed in ab400a8.

Cyfrin: Verified

7.3.5 `Liquidate` Event Emits Shares and Assets in Wrong Order

Description: The `Liquidate` event is fired making `shares` as the second parameter and `assets` as the third parameter

bc-solana-vault-sc/programs/sc-vault/src/instructions/liquidator/liquidate.rs#liquidate_handler

```
    emit!(crate::events::Liquidate {
        liquidator: ctx.accounts.liquidator.key(),
2:      shares,
3:      assets,
    });
```

But the event construction is not like this, as `assets` are the second parameter not third. and shares is the third parameter not second.

bc-solana-vault-sc/programs/sc-vault/src/events.rs

```
#[event]
pub struct Liquidate {
    pub liquidator: Pubkey,
2:  pub assets: u64,
3:  pub shares: u64,
}
```

There is also another thing to point out here, which is `assets` themselves. As the `assets` will be transferred to the liquidator if the vault is not activating `redemption`, but in case of supporting redemption the actual amount transferred to the liquidator is `liquidation_amount`. This may cause confusion at case weather assets are the actual received balance, or what.

Impact: Incorrect event emission leads to incorrect tracking, analysis of the liquidation process.

Recommended Mitigation:

- Swap assets position with shares position

```
    emit!(crate::events::Liquidate {
        liquidator: ctx.accounts.liquidator.key(),
-       shares,
        assets,
+       shares,
    });
```

- And for `liquidation_amount` this can be mitigated by adding another parameter for `liquidation_amount` (default is zero if no Redemption is not supported)

Securitize: Fixed in c766076.

Cyfrin: Verified

7.4 Informational

7.4.1 Invalid Zero Rate Not Rejected During Deposit

Description: When despoiting assets, and convert the shares to be minted to the despositer we are not checking the validity of rate value (weather it is greater zero or not)

bc-solana-vault-sc/programs/sc-vault/src/utils/conversions.rs#convert_to_shares

```
pub fn convert_to_shares( ... ) -> Result<u64> {
    let liq_token_decimals_factor = 10u64.pow(decimals.into());
    let total_shares_after_deposit = math::mul_div(
>>      rate,
        total_assets + assets,
        liq_token_decimals_factor,
        rounding,
    )?;

    if total_shares_after_deposit >= total_supply {
        Ok(total_shares_after_deposit - total_supply)
    } else {
        Ok(0)
    }
}
```

This is not the case when redeeming where we check that the rate value is greater than zero

bc-solana-vault-sc/programs/sc-vault/src/utils/conversions.rs#convert_to_assets

```
pub fn convert_to_assets( ... ) -> Result<u64> {
    if total_supply == 0 {
        return Ok(0);
    }
>>  require_gt!(rate, 0, ScVaultError::InvalidRate);
    let liq_token_decimals_factor = 10u64.pow(decimals.into());
    Ok(u64::min(
        math::mul_div(shares, liq_token_decimals_factor, rate, rounding)?,
        math::mul_div(shares, total_assets, total_supply, rounding)?,
    ))
}
```

This will make `total_shares_after_deposit` ends being `0`, results in minting `0` shares to the depositer (operator).

Impact:

- In case of incorrect return value from RedStone, the operator will take `0` shares

Recommended Mitigation: Check that `rate` is greater than `0`

```
pub fn deposit_handler<'info>( ... ) -> Result<()> {
    ...
    // Get rate from nav provider.
    let rate = get_rate!(ctx.remaining_accounts, ctx.accounts.nav_provider_program);
+   require_gt!(rate, 0, ScVaultError::InvalidRate);
    ...
    let shares = conversions::convert_to_shares( ... )?;

    ...
}
```

Securitize: Fixed in 2764f90.

Cyfrin: Verified

7.4.2 Redundant `vault_authority_signer` passed to `BurnChecked` in `redeem` and `liquidate`

Description: Both `redeem_handler` and `liquidate_handler` build their burn CPI like this:

```
let burn_ctx = CpiContext::new_with_signer(
    ctx.accounts.share_token_program.to_account_info(),
    BurnChecked {
        mint: ctx.accounts.share_mint.to_account_info(),
        from: ctx.accounts.<share_ata>.to_account_info(),
        authority: ctx.accounts.<operator_or_liquidator>.to_account_info(),
    },
    vault_authority_signer,              // ← unnecessary
);
burn_checked(burn_ctx, shares, ctx.accounts.share_mint.decimals)?;
```

The SPL-Token program requires **only the** `authority` **account to sign**. Here the authority is the operator/liquidator, who is already a transaction-signer. Adding `vault_authority_signer`:

- Produces an extra PDA signature that the token program ignores.
- Consumes compute units each time the instruction runs.

Recommended Mitigation: * Build the burn context without extra signers:

```
let burn_ctx = CpiContext::new(
    ctx.accounts.share_token_program.to_account_info(),
    BurnChecked {
        mint: ctx.accounts.share_mint.to_account_info(),
        from: ctx.accounts.<share_ata>.to_account_info(),
        authority: ctx.accounts.<operator_or_liquidator>.to_account_info(),
    },
);
```

- Keep `vault_authority_signer` only for calls that truly need the PDA to sign (e.g., vault asset transfers). This removes superfluous signatures, lowers compute costs, and avoids accidental brittleness.

Securitize: Fixed in 3635c15.

Cyfrin: Verified

7.4.3 Lack of `mut` on `liquidation_token_mint` restricts redemption flexibility

Description: When firing `liquidate_handler` and the Vault is redepmtion vault. we provide the necessary liquidate account (mint, vault, token_program, ...). `liquidation_token_mint` is not market with `mut` flag.

bc-solana-vault-sc/programs/sc-vault/src/instructions/liquidator/liquidate.rs#Liquidate

```
pub struct Liquidate<'info> {
    ...
    /// The mint account for the liquidation token.
    ///
    /// This account is only required when redemption program is enabled.
    /// It defines the liquidation tokens that will be received after redemption.
    #[account(
        mint::token_program = liquidation_token_program,
    )]
    pub liquidation_token_mint: Option<Box<InterfaceAccount<'info, Mint>>>,
    ...
}
```

This will prevent Redemption program to change the `mint` program. This will prevent the redemption program to mint `liquidation_amount` for example (if the program is the authority of liquidation token), as it will need to write to the account.

If the `redemption program` has the authority of `liquidation_token_mint` and it will work by taking `assets` and mint necessary `liquidation_tokens` it can't be done using the current interface. this affects the flexibility of the redemption programs to be introduced.

Impact:

- Preventing redemption program to supporting minting liquidation_token

Recommended Mitigation:

- Mark the account with `mut` in both `Liquidate` and `redemption_interface::Redeem`

Securitize: Fixed in 61d4f8c.

Cyfrin: Verified