Grow Protocol

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Jun 09 2025

Grow Protocol Audit Report

1 Executive Summary

1.1 Project Information

Description	The Grow protocol is a yield-generating derivative of Solido's stablecoin, $CASH. It serves as the first line of defense during liquidations and utilizes the pooled $CASH to liquidate underperforming troves, thereby earning collateral penalties and a liquidation reserve. The rewards are compounded back into $CASH, and the $bCASH (the receipt token) accumulates value over time with respect to $CASH
Type	DeFi
Auditors	MoveBit
Timeline	Mon May 26 2025 - Mon Jun 09 2025
Languages	Move
Platform	Supra
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Solido-Money/grow-protocol
Commits	78c9a37c7cc0ca980d36e85cc344576d626e68d2 c76d57f029b0b4a3bf45d7e18a7263d3d28cae2d e17052bf4a257631d320ddbe5cc52a794ec4b0e0

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MOV	Move.toml	be967a5d74d38c90b6e1d5914043e5f13df3f68b
STR	sources/strategy.move	713547608d3e3367bdfae8855dfbe87c62706fb0
VAU	sources/vault.move	bbee35d9a2417438202212387c563f2406f67b21

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	9	8	1
Informational	3	3	0
Minor	1	0	1
Medium	4	4	0
Major	0	0	0
Critical	1	1	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Solido to identify any potential issues and vulnerabilities in the source code of the Grow Protocol smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 9 issues of varying severity, listed below.

ID	Title	Severity	Status
STR-1	Unused Constants	Informational	Fixed
VAU-1	Share Manipulation Attack	Critical	Fixed
VAU-2	Shares Transferred to the Wrong Address	Medium	Fixed
VAU-3	Fee Evasion via Small Withdrawals	Medium	Fixed
VAU-4	The Last Few Users may be unable to Redeem	Medium	Fixed
VAU-5	The Case where `new_balance < original_balance` is not Handled	Medium	Fixed
VAU-6	Front-Running `harvest` for Arbitrage	Minor	Acknowledged
VAU-7	Unused `shares` Field	Informational	Fixed
VAU-8	Ineffective `earn` Function	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Grow Protocol Smart Contract :

Admin

- `pause` : Pause vault operations
- `unpause` : Resume vault operations
- `set_tvl_limit` : Set vault deposit cap
- `set_performance_fee` : Set performance fee percentage
- `harvest` : Harvest strategy yield and collect fees
- `set_withdraw_delay` : Set withdrawal unlock delay time
- `set_fee_recipient` : Set address to receive fees
- `set_withdrawal_fee` : Set withdrawal fee percentage
- `update_slippage` : Update maximum allowed slippage
- `set_min_withdrawal_amount` : Update the minimum withdrawal amount
- `sync_assets` : Update total_assets to actual vault balance

User

- `deposit` : Deposit assets into vault for self
- `deposit_for` : Deposit assets into the vault for another address
- `withdraw` : Request asset withdrawal to own address
- `withdraw_to` : Request asset withdrawal to another address
- `redeem` : Redeem shares for assets to own address
- `redeem_to` : Redeem shares for assets to another address
- `claim_withdrawal` : Claim delayed withdrawals after unlock time

4 Findings

STR-1 Unused Constants

Severity: Informational

Status: Fixed

Code Location:

sources/strategy.move#3;

sources/vault.move#162

Descriptions:

There are unused constants in the contract.

```
const ERR_INSUFFICIENT_ASSETS: u64 = 3;
```

```
const ERR_STRATEGY_FAILED: u64 = 6;
```

Suggestion:

It is recommended to remove unused constants if there's no further design.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

VAU-1 Share Manipulation Attack

Severity: Critical

Status: Fixed

Code Location:

sources/vault.move#261-490

Descriptions:

When a user makes their first deposit, the `convert_to_shares` function grants shares at a 1:1 ratio. However, this share calculation can be manipulated, especially during the initial stages when the vault is empty.

The share price is determined using `total_assets`, which is implemented as:

```
public fun total_assets<AssetType>(): u64 {
    strategy_core::balance_of<AssetType>()
}
```

Attack Scenario:

1. Attacker A deposits 1 unit of `CASH` via the `deposit` function and receives 1 `bCASH` (share token).
2. Attacker then directly transfers `10**8` `CASH` tokens to either the `strategy_addr` or `vault_resource_addr`, inflating the vault's balance.
3. Victim B deposits `10**8` `CASH`, but due to precision loss, receives:
   `10**8 * 1 / (10**8 + 1) = 0 bCASH`
   —effectively getting no shares.
4. Attacker A then withdraws their 1 `bCASH`, extracting the entire `10**8 + 1` `CASH`, profiting unfairly.

This results in a severe imbalance and exploits honest users who deposit afterward.

Suggestion:

1. Revert if the amount received is not within a slippage tolerance (Add slip protection if possible)
2. The deployer should initialize the vault by depositing a small amount of `CASH` during deployment. This helps establish a fair initial share ratio and prevents early-stage manipulation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

VAU-2 Shares Transferred to the Wrong Address

Severity: Medium

Status: Fixed

Code Location:

sources/vault.move#261-326

Descriptions:

In the `deposit_internal` function, the calculated share tokens are incorrectly transferred to the `user_addr` (the transaction sender), rather than to the intended `receiver`.

```
// Incorrect: shares are minted and deposited to user_addr
let share_tokens = coin::mint(shares, &share_cap.mint_cap);
coin::deposit<VaultShare>(user_addr, share_tokens);
```

However, the function accepts a `receiver: address` parameter, and this address is also used in the emitted `DepositEvent`, indicating that the shares should go to the receiver.

Suggestion:

Replace `user_addr` with `receiver` when depositing the share tokens to ensure correct recipient behavior:

```
// Correct: shares are deposited to receiver
coin::deposit<VaultShare>(receiver, share_tokens);
```

This ensures the `deposit_for` function behaves as expected, allowing users to deposit on behalf of others.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

VAU-3 Fee Evasion via Small Withdrawals

Severity: Medium

Status: Fixed

Code Location:

sources/vault.move#624-632;

sources/vault.move#491-594

Descriptions:

When a user withdraws assets, a withdrawal fee is applied—such as in the `process_withdrawal` function:

```
if (!is_delayed_withdrawal) {
    let fee_amount = if (withdrawal_fee > 0) {
        let fee_u128 = (assets as u128) * (withdrawal_fee as u128) / (FEE_PRECISION as u128);
        (fee_u128 as u64) // Truncates toward zero
    } else {
        0
    };

    let withdrawal_amount = assets - fee_amount;
}
```

However, users can evade the fee by repeatedly withdrawing very small amounts. Due to integer division and rounding down, the `fee_amount` becomes zero for small `assets` values, effectively allowing free withdrawals.

Suggestion:

To prevent fee evasion:

- Introduce a **minimum withdrawal amount**.
- Or enforce that `fee_amount > 0` when `withdrawal_fee > 0`, and reject the transaction otherwise.

This will ensure that the protocol collects meaningful fees and discourages abuse through micro-withdrawals.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

VAU-4 The Last Few Users may be unable to Redeem

Severity: Medium

Status: Fixed

Code Location:

sources/vault.move#772

Descriptions:

The `redeem()` function allows users to redeem shares for underlying assets. The amount of assets to be redeemed is calculated using the formula:

`assets = shares * total_assets / total_supply`

```
public fun convert_to_assets<AssetType>(shares: u64): u64 acquires VaultInfo {
        let vault = borrow_global<VaultInfo>(@vault);
        let supply = vault.total_shares;
        let total = total_assets<AssetType>();

        if (supply == 0) {
            return shares
        };

        // Use u128 for intermediate calculations to prevent overflow
        let shares_with_precision = (shares as u128) * (PRECISION as u128);
        let assets_precise = (shares_with_precision * (total as u128)) / (supply as u128);

        // Divide by PRECISION to get back to normal scale
        let assets_precise = assets_precise / (PRECISION as u128);

        // Convert back to u64 with safety check
        assert!(assets_precise <= (MAX_U64 as u128), ERR_MATH_OVERFLOW);
        (assets_precise as u64)
    }
```

Here, `total_assets` is derived from the combined balances of `strategy_addr` and `vault_resource_addr`.

```
#[view]
public fun balance_of<AssetType>(): u64 acquires StrategyInfo, StrategyCapability {
    // Get strategy's balance
    let strategy_signer = get_strategy_signer();
    let strategy_addr = signer::address_of(&strategy_signer);
    let strategy_balance = coin::balance<AssetType>(strategy_addr);

    // Get vault's balance using stored address
    let strategy = borrow_global<StrategyInfo>(@vault);
    let vault_balance = coin::balance<AssetType>(strategy.vault_resource_addr);

    // Return total balance across both accounts
    strategy_balance + vault_balance
}
```

After calculating the redeemable amount, the protocol burns the user's shares and then updates the vault's accounting.

```
    // Burn shares immediately in all cases
    let share_cap = borrow_global<ShareTokenCapability>(@vault);
    let share_tokens = coin::withdraw<VaultShare>(user, shares);
    coin::burn(share_tokens, &share_cap.burn_cap);

    // Update vault accounting
    let vault = borrow_global_mut<VaultInfo>(@vault);
    vault.total_shares = vault.total_shares - shares;
    vault.total_assets = vault.total_assets - assets;
```

However, there is an issue: since `total_assets` is computed based on the balances of these addresses, anyone can send tokens directly to them. These unsolicited transfers inflate `total_assets`, causing the protocol to overestimate the value of each share.

As a result, users may redeem more assets than they are entitled to. Over time, this discrepancy accumulates, and when the last few users attempt to redeem, the vault's

internal accounting (`vault.total_assets = vault.total_assets - assets`) may underflow, making it impossible to withdraw funds.

Suggestion:

It is recommended to maintain an accurate internal record of `vault.total_assets` .

Resolution:

This issue has been fixed. The client has adopted our suggestions.

VAU-5 The Case where `new_balance < original_balance` is not Handled

Severity: Medium

Status: Fixed

Code Location:

sources/vault.move#1009-1058

Descriptions:

In the `harvest()` function, the protocol liquidates user collateral and converts the collateral into the underlying asset, which is then transferred to `vault_resource_addr`. After this, if `new_balance > original_balance`, the protocol processes fees and updates the vault's accounting accordingly.

```
        if (new_balance > original_balance) {
            yield_amount = new_balance - original_balance;

            // Get the vault info with performance fee percentage and fee recipient
            let vault = borrow_global_mut<VaultInfo>(@vault);
            let fee_percentage = vault.performance_fee;
            let fee_recipient = vault.fee_recipient;

            // Calculate fee amount - use u128 for precision
            // With new FEE_PRECISION, 1000 = 1%
            let fee_amount = if (fee_percentage > 0) {
                // Calculate fee amount with proper precision
                let fee_u128 = (yield_amount as u128) * (fee_percentage as u128) /
(FEE_PRECISION as u128);
                (fee_u128 as u64) // Convert back to u64
            } else {
                0
            };
```

However, if `new_balance < original_balance` —for example, due to slippage, price impact, or poor liquidation execution—the protocol does not handle this case. As a result, the internal accounting may become inaccurate, potentially misrepresenting the true value of assets in the vault.

Suggestion:

It is recommended to account for this edge case by handling scenarios where the liquidation results in a loss.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

VAU-6 Front-Running `harvest` for Arbitrage

Severity: Minor

Status: Acknowledged

Code Location:

sources/vault.move#935-1013

Descriptions:

The `harvest` function is called by the admin to distribute liquidation rewards to stakers. If `harvest_delay` is set to `0`, users can immediately withdraw after harvesting. Under low-fee conditions, this opens the door for front-running and arbitrage attacks.

Attack Scenario:

1. The attacker deposits a large amount of funds via the `deposit` function.
2. The admin calls the `harvest` function, distributing rewards.
3. The attacker quickly withdraws their funds via the `withdraw` function.

By doing this, the attacker captures a disproportionate share of the rewards without meaningful participation in the staking period, which is unfair to long-term stakers.

Suggestion:

To mitigate this, consider increasing the `harvest_delay` value when arbitrage opportunities are likely, preventing immediate withdrawals and reducing the incentive for front-running the `harvest` function.

VAU-7 Unused `shares` Field

Severity: Informational

Status: Fixed

Code Location:

sources/vault.move#41

Descriptions:

The `shares` field in the `VaultInfo` struct is currently unused and not updated anywhere in the protocol.

```
/// Extended vault configuration
struct VaultInfo has key {
    name: String,
    symbol: String,
    decimals: u8,
    total_assets: u64,
    total_shares: u64,
    shares: Table<address, u64>, // Unused field
    paused: bool,
    withdraw_requests: Table<address, UserWithdrawRequests>,
    tvl_limit: u64,
    performance_fee: u64,
    harvest_delay: u64,
    asset_type: TypeInfo,
    fee_recipient: address,
    withdrawal_fee: u64,
}
```

Suggestion:

If this field is intended to track individual user shares, it should be properly integrated and updated throughout the protocol. Otherwise, consider removing it to reduce storage overhead and avoid confusion.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

VAU-8 Ineffective `earn` Function

Severity: Informational

Status: Fixed

Code Location:

sources/vault.move#329-353

Descriptions:

In the `deposit_internal` function, the `earn<AssetType>()` function is called with the comment "Deploy to strategy". However, the `earn` function currently has no operational logic—the core functionality is commented out, and it performs no state changes or external calls.

```
// Ineffective earn function
fun earn<AssetType>() acquires VaultCapability {
    let vault_signer = get_vault_signer();
    let vault_addr = signer::address_of(&vault_signer);

    let strategy_signer = strategy_core::get_strategy_signer();
    let strategy_addr = signer::address_of(&strategy_signer);

    let available_assets = coin::balance<AssetType>(vault_addr);

    // The intended logic is commented out
    // if (available_assets > 0) {
    //     coin::transfer<AssetType>(&vault_signer, strategy_addr, available_assets);
    //     strategy_core::deposit<AssetType>();
    // };
}
```

Suggestion:

If this function is not intended to be used, consider removing the `earn` function.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.