Solido

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Fri Aug 08 2025

Solido Audit Report

1 Executive Summary

1.1 Project Information

Description	A liquid staking protocol built on the Supra blockchain, enabling users to stake assets across multiple validators while maintaining liquidity through tokenized shares
Type	DeFi
Auditors	MoveBit
Timeline	Wed Jul 30 2025 - Fri Aug 08 2025
Languages	Move
Platform	Supra
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Solido-Money/flow-protocol/
Commits	458fdb1f981168c53d0b3b1e3d47fcce675cc707 371e1988974453da66ff2d4f983d0182c000d9eb

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	Move.toml	6f9dc6ec83c0b3becb88e3cbdfe7b4d3dcd32920
SCO	sources/strategy_core.move	227054cf6167ccd162352ec0040b1cd1fb1cc3f6
SDE	sources/strategy_delegation.move	b262dc2cfa3ccbabc3e052ac0efedc4f2fb26f3d
SVA	sources/strategy_validator.move	286be9ac5d565097c5e7701b4a6f24c741c3cb29
VAU	sources/vault.move	aa3b52f3a7db2c2b9b06eea338863daee2a030dc
SST	sources/strategy_staking.move	1130b09c885bc31b058942fefaba118115ac8a5d

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	5	5	0
Informational	2	2	0
Minor	2	2	0
Medium	1	1	0
Major	0	0	0
Critical	0	0	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 Solido 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 5 issues of varying severity, listed below.

ID	Title	Severity	Status
SCO-1	Redundant Calculation and Unused Variable in `rebalance`	Minor	Fixed
SCO-2	`add_validator` and `remove_validator` Repeatedly Checking The Code	Minor	Fixed
SCO-3	`ERR_NOT_INITIALIZED` Not Used	Informational	Fixed
SDE-1	Use `public(friend)` instead of `public`	Informational	Fixed
VAU-1	Potential DoS via Unbounded Iteration in `claim_withdrawal`	Medium	Fixed

3 Participant Process

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

Admin

- `withdraw_unlocked` : Withdraw the funds unlocked by the policy from all registered validators
- `pause` : Suspend the vault
- `unpause` :Lift the suspension of the vault
- `set_tvl_limit` : Set the upper limit of the total locked value of the vault.
- `set_performance_fee` : Set the performance rate charged from the strategy's earnings
- `harvest` : Used to trigger the strategy for revenue harvesting and asset rebalancing
- `sync_assets` :Synchronize the total assets recorded inside the vault with the actual asset balance in the vault resource account
- `set_withdraw_delay` :Set or update the time delay between when a user initiates a withdrawal and when they can receive their assets.
- `set_fee_recipient` : Change the address for receiving all fees
- `set_withdrawal_fee` : Used to set or update the handling fee rate charged when withdrawing funds

Strategy

- `add_validator` : Add the new validator to the system's authorized validator list
- `remove_validator` : Remove a validator from the system's list of authorized validators
- `delegate` : Entrust a specified quantity of assets to a validator
- `undelegate` : Cancel the delegation from a validator

User

- `deposit` : Users deposit the specified amount of assets into the vault and receive the corresponding amount of share tokens at the current exchange rate
- `deposit_for` : User deposits the specified quantity of assets into the vault and sends the obtained share tokens to the designated address.
- `deposit_with_slippage` : User deposits assets and provides an acceptable minimum share quantity
- `withdraw` : User requests to withdraw a specified quantity of assets
- `withdraw_to` : User requests to extract a specified quantity of assets and send these assets to the designated address.
- `claim_withdrawal` :After the withdrawal delay period, users call this function to withdraw the assets they previously requested
- `redeem` : Users can redeem the corresponding amount of assets by destroying the specified number of share tokens
- `redeem_to` : User destroys the specified quantity of share tokens and sends the redeemed underlying assets to the designated address.
- `initialize_account` : Ability to allow any user to register for their own account to receive underlying assets and share tokens

4 Findings

SCO-1 Redundant Calculation and Unused Variable in `rebalance`

Severity: Minor

Status: Fixed

Code Location:

sources/strategy_core.move#196

Descriptions:

The `direct_settle_amount` variable is calculated within the `rebalance` function but is never used in any subsequent logic. This constitutes dead code, which reduces code clarity.

```
        // Handle delay transfer amount and update pending_withdrawals
        if (delay_transfer_amount > 0) {
            // Check that delay_transfer_amount >= unlocked_amount
            assert!(delay_transfer_amount >= unlocked_amount,
ERR_INSUFFICIENT_DELAY_TRANSFER);
            let direct_settle_amount = delay_transfer_amount - unlocked_amount;
            // Check that we have enough liquid funds after all operations
            let current_liquid_balance = coin::balance<AssetType>(resource_addr);
            assert!(current_liquid_balance >= delay_transfer_amount,
ERR_INSUFFICIENT_FUNDS);

            // Transfer the specified amount to delayed withdrawal account
            let delayed_withdrawal_addr = {
                let core_info = borrow_global<CoreInfo>(@vault);
                core_info.delayed_withdrawal_addr
            };
```

Suggestion:

Remove the line that declares the `direct_settle_amount` variable.

Resolution:

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

SCO-2 `add_validator` and `remove_validator` Repeatedly Checking The Code

Severity: Minor

Status: Fixed

Code Location:

sources/strategy_core.move#281-302

Descriptions:

In the 'add_validator' and 'remove_validator' functions, the same administrator permission verification logic is repeated

```
public entry fun add_validator(
    admin: &signer,
    validator: address
) {
    let admin_addr = signer::address_of(admin);
    assert!(admin_addr == @vault, ERR_NOT_ADMIN);

    // Add validator
    strategy_validator::add_validator(admin, validator);
}
```

```
public(friend) fun add_validator(
    admin: &signer,
    validator: address
) acquires ValidatorRegistry {
    let admin_addr = signer::address_of(admin);
    assert!(admin_addr == @vault, ERR_NOT_ADMIN);

    let registry = borrow_global_mut<ValidatorRegistry>(@vault);
    .....
```

Suggestion:

Delete the duplicate verification logic

Resolution:

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

SCO-3 `ERR_NOT_INITIALIZED` Not Used

Severity: Informational

Status: Fixed

Code Location:

sources/strategy_core.move#20;

sources/strategy_staking.move#18

Descriptions:

The ERR_NOT_INITIALIZED error code parameter is not used

```
const ERR_NOT_INITIALIZED: u64 = 3;
```

Suggestion:

Delete the redundant code

Resolution:

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

SDE-1 Use `public(friend)` instead of `public`

Severity: Informational

Status: Fixed

Code Location:

sources/strategy_delegation.move#96

Descriptions:

The `undelegate()` function is invoked through the `strategy_staking` contract. It is recommended to change the visibility of the `undelegate()` function to `public(friend)`. Similarly, the same recommendation applies to the `withdraw_unlocked()` function.

```
if (amount > 0) {
        // Undelegate assets from validator
        strategy_delegation::undelegate<AssetType>(strategy_signer, validator, amount);

        // Emit unstaking event
        event::emit(AssetsUnstakedEvent {
           validator,
           amount,
           timestamp: timestamp::now_seconds()
        });
    };
```

Suggestion:

It is recommended to change the visibility of the `undelegate()` function to `public(friend)`.

Resolution:

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

VAU-1 Potential DoS via Unbounded Iteration in `claim_withdrawal`

Severity: Medium

Status: Fixed

Code Location:

sources/vault.move#553-658

Descriptions:

The `claim_withdrawal` function iterates through all of a user's withdrawal requests stored in the `user_requests.request_ids` vector. While processed requests are marked as `processed = true`, they are never removed from this vector or the underlying `requests` table.

This design flaw allows the `request_ids` vector to grow indefinitely with each new withdrawal request. Consequently, the gas cost for calling `claim_withdrawal` increases linearly with the number of historical requests, which can lead to a DoS condition where transactions fail due to exceeding the gas limit, preventing the user from claiming their funds.

```
        while (i < request_ids_len) {
            let request_id = *vector::borrow(&user_requests.request_ids, i);
            if (table::contains(&user_requests.requests, request_id)) {
                let request = table::borrow_mut(&mut user_requests.requests, request_id);
                if (!request.processed && current_time >= request.request_time +
    request.delay_at_request) {
                    total_claimable = total_claimable + request.assets;
                    request.processed = true;
                };
            };
            i = i + 1;
        };
```

Suggestion:

Modify the function to remove processed requests after they are claimed.

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.