SOLIDO

SOLID

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Tue Nov 04 2025

SOLID Audit Report

1 Executive Summary

1.1 Project Information

Description	This project implements a comprehensive token management system on the Supra blockchain, featuring a flexible vesting mechanism and an efficient Merkle tree–based airdrop. The vesting contract supports multiple use cases such as team allocations, advisor rewards, and airdrop recipients who choose the vesting option, following a three phase model with TGE, a cliff period, and linear vesting. The fungible token module provides minting, burning, and transfer capabilities, while the Merkle tree–based airdrop ensures secure and gas-efficient token distribution by verifying eligibility through cryptographic proofs without storing individual allocations on-chain
Type	DeFi
Auditors	Alex,PeiQi0
Timeline	Sun Oct 12 2025 - Tue Nov 04 2025
Languages	Move
Platform	Others
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Solido-Money/SOLID

Commits	167f857dd50a0fbd543a88f2b97acdcd02e56d66 45c33853b1a06fcbecd6e1de8bedb47005165b88

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	vesting_solid/Move.toml	8335790a071bde8ee18c0a42f4f5520130daa7dc
MOV1	merkle_solid/Move.toml	97ff610ed8b1e6c02e80ffe332b115983f46a4e6
MOV2	token_solid/Move.toml	f312bd142422702cee2ae9e41a4921bf4390513d
VES	vesting_solid/sources/vesting.move	820fb56c2710e0216bfcceb950d0a8403e957290
AIR	merkle_solid/sources/airdrop.move	4edd54bce5e61730363b3c3fc7ea26e3cb71bc31
TOK	token_solid/sources/token.move	4218c2f570f143609e6ff35d998e914c21872402

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	5	1
Critical	0	0	0
Major	1	1	0
Medium	2	2	0
Minor	2	2	0
Informational	1	0	1

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 Money to identify any potential issues and vulnerabilities in the source code of the SOLID 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 6 issues of varying severity, listed below.

ID	Title	Severity	Status
AIR-1	`emergency_withdraw` Should Be Calculated Based On The Balance	Medium	Fixed
AIR-2	`fund_airdrop` Lack Of inspection	Minor	Fixed
AIR-3	`claim_with_slashing` Redundant Parameters	Minor	Fixed
AIR-4	Missing Leaf Node Usage Tracking in `claim_with_slashing()` Function	Informational	Acknowledged
TOK-1	`mint` No Permission Control	Major	Fixed
VES-1	`create_airdrop_vesting` Modifier Error	Medium	Fixed

3 Participant Process

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

User

- `claim_airdrop_vesting` : Releases unlocked tokens from their vesting position.
- `claim_with_slashing` : Claim the airdrop 50% to user, 50% burned
- `claim_with_vesting` : Claim the airdrop tokens and directly place these tokens in the locking contract for
- `claim_for_vesolid_lock` : Lock these tokens directly into the veSOLID lock contract to obtain governance weights and additional benefits

Admin

- `fund_airdrop` : Inject funds into the contract
- `end_airdrop` : End the airdrop
- `clear_airdrop` : Clear the airdrop data

4 Findings

AIR-1 `emergency_withdraw` Should Be Calculated Based On The Balance

Severity: Medium

Status: Fixed

Code Location:

merkle_solid/sources/airdrop.move#368

Descriptions:

The function uses the difference between the state variables total_allocation and total_claimed to calculate the number of tokens to be withdrawn, rather than directly using the actual token balance held in the Treasury. In DeFi and token contracts, it is the best practice to directly use the account balance rather than the calculated value. Because the balance is the actual number of tokens held in the account, it will not be affected by the calculation error of the state variable

```
    public entry fun emergency_withdraw(admin: &signer, to: address) acquires Airdrop,
Treasury {
        assert!(signer::address_of(admin) == @PROTO, E_NOT_ADMIN);
        let airdrop = borrow_global_mut<Airdrop>(@PROTO); //

        let remaining = airdrop.total_allocation - airdrop.total_claimed; //

        if (remaining > 0) {
            let treasury = borrow_global_mut<Treasury>(@PROTO);
            let extracted = coin::extract(&mut treasury.coins, remaining);
            coin::deposit(to, extracted);
            airdrop.total_allocation = airdrop.total_claimed;
        };

        airdrop.end_time = timestamp::now_seconds();
```

```
    event::emit(EmergencyWithdrawEvent {
        to,
        amount: remaining,
    });
}
```

Suggestion:

Modify it to the actual number of tokens held in the account

Resolution:

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

AIR-2 `fund_airdrop` Lack Of inspection

Severity: Minor

Status: Fixed

Code Location:

merkle_solid/sources/airdrop.move#145

Descriptions:

When the contract sets total_claimed to be equal to total_allocation, all tokens represent that they have been claimed. At this point, attempting to inject funds may result in these funds being locked in the contract. The function needs to add a check that no further injection of funds is allowed after the airdrop ends

Suggestion:

Add a verification that does not allow further injection of funds after the airdrop ends

Resolution:

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

AIR-3 `claim_with_slashing` Redundant Parameters

Severity: Minor

Status: Fixed

Code Location:

merkle_solid/sources/airdrop.move#163

Descriptions:

In the function, solid_metadata is passed as a parameter, but this object is not referenced or used anywhere within the function body. Removing or properly handling this unused parameter will enhance the security, clarity and maintainability of the code, while reducing potential usage confusion.

```
public entry fun claim_with_slashing(
    account: &signer,
    amount: u64,
    index: u64,
    proof: vector<vector<u8>>,
    solid_metadata: Object<Metadata>
)
```

Suggestion:

Delete unused parameters

Resolution:

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

AIR-4 Missing Leaf Node Usage Tracking in `claim_with_slashing()` Function

Severity: Informational

Status: Acknowledged

Code Location:

merkle_solid/sources/airdrop.move#291-297

Descriptions:

In the `claim_with_slashing()` function, the protocol calculates the leaf node based on the `user`, `amount`, and `index`, and then verifies whether the computed Merkle root matches the one specified by the official source.

```
// Verify merkle proof
        let leaf_data = vector::empty<u8>();
        vector::append(&mut leaf_data, bcs::to_bytes(&user));
        vector::append(&mut leaf_data, bcs::to_bytes(&amount));
        vector::append(&mut leaf_data, bcs::to_bytes(&index));
        let leaf = hash::sha3_256(leaf_data);

        assert!(verify_merkle_proof(&leaf, &proof, &airdrop.merkle_root, index),
E_INVALID_PROOF);

        // Mark as claimed and record claim type
        *vector::borrow_mut(&mut airdrop.claims, index) = true;
        *vector::borrow_mut(&mut airdrop.claim_types, index) = CLAIM_TYPE_VESOLID;
        airdrop.total_claimed = airdrop.total_claimed + amount;
```

Suggestion:

It is recommended that once a leaf node has been used, it should be marked as `true` (claimed) to prevent signature reuse or repeated claims.

TOK-1 `mint` No Permission Control

Severity: Major

Status: Fixed

Code Location:

token_solid/sources/token.move#65

Descriptions:

Although the function comments clearly indicate that this is an internal function, the function is actually defined as public and has no permission verification mechanism at all. Unlike other administrative functions (such as mint_to, burn), this function does not check whether the caller address is the @PROTO administrator address. This means that any module can directly call this function to mint tokens without any limit.

```
public fun mint(amount: u64): FungibleAsset acquires TokenRefs {
    let token_refs = borrow_global<TokenRefs>(@PROTO);
    fungible_asset::mint(&token_refs.mint_ref, amount)
}
```

Suggestion:

Add permission checks

Resolution:

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

VES-1 `create_airdrop_vesting` Modifier Error

Severity: Medium

Status: Fixed

Code Location:

vesting_solid/sources/vesting.move#117

Descriptions:

The create_airdrop_vesting comment describes it as "Create a vesting position for airdrop (called by airdrop contract)", but the function is actually an external call and should be modified to an internal call

```
public entry fun create_airdrop_vesting(
    user: &signer,
    amount: u64,
    solid_metadata: Object<Metadata>
) acquires VestingConfig {
```

Suggestion:

Modify it to an internal 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.