veSOLID

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Tue Nov 04 2025

veSOLID Audit Report

1 Executive Summary

1.1 Project Information

Description	The veSOLID contract implements a vote-escrowed token mechanism with integrated vesting rewards. Users lock SOLID tokens for a specified period to receive both voting power and APR-based rewards. The system incentivizes long-term token holding while providing transparent, time-proportional reward accumulation
Type	DeFi
Auditors	Alex,hyer
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/veSOLID
Commits	df8db8bdb2a05d6a7fe7f9b3c4ad5b23acd26722 4366a4bb5d2bee3733cf41583308bf8ca77dec40

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	veSOLID/Move.toml	fb83bb13412f467f0829a4eaf5db1ddd715df5a3
PROTO	veSOLID/sources/PROTO.move	b4ad0d112999ffd10158e024b6c9f2f1e8fb4658
VSOLID	veSOLID/sources/veSOLID.move	53ee89d954ed3e94021b22bd8456d5c5883aa844

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	7	7	0
Critical	0	0	0
Major	2	2	0
Medium	2	2	0
Minor	2	2	0
Informational	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 Money to identify any potential issues and vulnerabilities in the source code of the veSOLID 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 7 issues of varying severity, listed below.

ID	Title	Severity	Status
PRO-1	Unauthorized Mint Function Allows Arbitrary Token Minting	Major	Fixed
PRO-2	Missing `entry` in burn_shares Function	Medium	Fixed
VSO-1	Missing Admin Withdrawal Functionality for the `RewardPool` Balance	Major	Fixed
VSO-2	Missing Validation for Zero Reward in `calculate_expected()` Function	Medium	Fixed
VSO-3	The zero-value Check for the Constant `MAX_LOCK_DURATION` is Redundant	Minor	Fixed
VSO-4	Missing Critical Event Logging	Minor	Fixed
VSO-5	Redundant `old_expected` Calculation Increases Logic Complexity	Informational	Fixed

3 Participant Process

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

Admin:

- `initialize` : Initializes the reward pool, creates the token storage object, and sets the default APR.
- `set_apr` : Updates the global APR (annual percentage rate). Affects only newly created locks and emits an APR update event.
- `fund_pool` : Adds funds to the reward pool to ensure sufficient liquidity for future withdrawals.
- `initialize_token` : Initializes the SOLID token, creates its metadata, mint/burn/transfer references, and sets the maximum supply.
- `mint_to` : Allows the admin to mint new SOLID tokens and deposit them directly into a specific address.
- `burn` : Allows the admin to burn SOLID tokens from their own account, reducing total supply.

User:

- `create_lock` : Creates a new lock position by depositing tokens into the reward pool. The user earns rewards over the locking period.
- `increase_time` : Extends the duration of an existing lock and increases the expected reward based on the current APR.
- `withdraw` : Withdraws the principal and rewards after the lock period expires. The lock position is deleted after payout.
- `transfer` : Transfers SOLID tokens from the caller's account to another address.
- `burn_shares` : Burns SOLID tokens from the caller's own account.

4 Findings

PRO-1 Unauthorized Mint Function Allows Arbitrary Token Minting

Severity: Major

Status: Fixed

Code Location:

veSOLID/sources/PROTO.move#67

Descriptions:

```
/// Mint tokens (internal, called by airdrop)
  public fun mint(amount: u64): FungibleAsset acquires TokenRefs {
    let token_refs = borrow_global<TokenRefs>(@solidove);
    fungible_asset::mint(&token_refs.mint_ref, amount)
  }
```

The mint function is public and does not have a &signer parameter or any check on the caller's address/role. It also does not use the mint_cap capability stored only in protected resources to restrict who can call fungible_asset::mint. It is also possible that the assumption of "internal call only for airdrop" was written as public.

Suggestion:

Add permission checks.

Resolution:

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

PRO-2 Missing `entry` in burn_shares Function

Severity: Medium

Status: Fixed

Code Location:

veSOLID/sources/PROTO.move#92

Descriptions:

The function `burn_shares` is declared as a `public fun` instead of a `public entry fun`. In Move, only `entry` functions can be invoked directly via transactions. Without the `entry` keyword, this function cannot be called externally by users, even though it is marked as `public`.

```
public fun burn_shares(from: &signer, amount: u64) acquires TokenRefs {
    let token_refs = borrow_global<TokenRefs>(@solidove);
    let metadata = token_refs.metadata;
    let fa = primary_fungible_store::withdraw(from, metadata, amount);
    fungible_asset::burn(&token_refs.burn_ref, fa);
}
```

Suggestion:

Change the function declaration to `public entry fun` so that it can be invoked directly by users through transactions.

Resolution:

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

VSO-1 Missing Admin Withdrawal Functionality for the `RewardPool` Balance

Severity: Major

Status: Fixed

Code Location:

veSOLID/sources/veSOLID.move#259

Descriptions:

In the current contract implementation, the `RewardPool` provides two fund flow paths:

- `fund_pool` — allows the administrator to inject tokens into the pool:

  ```
  public entry fun fund_pool(admin: &signer, solid_metadata: Object<Metadata>, amount: u64)
  ```

  This function increases the pool balance via `fungible_asset::deposit`.

- `withdraw` — allows users to claim their principal and rewards after their lock period ends:

  ```
  public entry fun withdraw(user: &signer, lock_object: Object<LockPosition>)
  ```

  This function is user-initiated and uses `SOLID::withdraw_from_pool` to withdraw both rewards and principal.

However, the contract does **not** implement any administrator-level withdrawal function for the pool balance. As a result, once the administrator deposits funds into the reward pool via `fund_pool`, these assets **cannot be retrieved** by the administrator unless users withdraw rewards or the contract is upgraded or destroyed.

Suggestion:

It is recommended to add a restricted administrator withdrawal function.

Resolution:

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

VSO-2 Missing Validation for Zero Reward in `calculate_expected()` Function

Severity: Medium

Status: Fixed

Code Location:

veSOLID/sources/veSOLID.move#67-79

Descriptions:

The `calculate_expected()` function is used to calculate the expected payout. However, the protocol does not check whether the calculated reward is greater than zero. If the principal amount is too small, the resulting reward could be zero, causing users to lock their funds without receiving any compensation.

```
/// Calculate expected payout with full u128 precision (no early flooring)
fun calculate_expected(principal: u64, duration: u64, apr_bps: u64): u64 {
    if (MAX_LOCK_DURATION == 0u64) { return principal };

    // Full precision: reward = principal * (apr * duration) / (MAX * 10000)
    let numerator = (principal as u128) * (apr_bps as u128) * (duration as u128);
    let denominator = (MAX_LOCK_DURATION as u128) * 10000u128;
    let reward = numerator / denominator;

    let total = (principal as u128) + reward;
    assert!(total <= 18446744073709551615u128, E_OVERFLOW);

    (total as u64)
}
```

Suggestion:

It is recommended to add a check to ensure that the calculated reward is greater than zero before allowing users to lock their funds.

Resolution:

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

VSO-3 The zero-value Check for the Constant `MAX_LOCK_DURATION` is Redundant

Severity: Minor

Status: Fixed

Code Location:

veSOLID/sources/veSOLID.move#68,84,94

Descriptions:

MAX_LOCK_DURATION is declared as a compile-time constant (`const`) within the module and assigned a non-zero value. Since constants are inlined at compile time, their value can never be zero at runtime; therefore, the check for `MAX_LOCK_DURATION == 0u64` is a branch that will never be executed.

```
const MAX_LOCK_DURATION: u64 = 126230400u64; // 4 years in seconds

fun calculate_expected(principal: u64, duration: u64, apr_bps: u64): u64 {
    if (MAX_LOCK_DURATION == 0u64) { return principal };
    // ...
}

fun calculate_voting_power(original_amount: u64, lock_end: u64, now: u64): u64 {
    let remaining = if (now >= lock_end) { 0u64 } else { lock_end - now };
    if (MAX_LOCK_DURATION == 0u64) { 0u64 } else {
        // ...
    }
}

fun calculate_delta_reward(principal: u64, additional_duration: u64, apr_bps: u64): u64 {
    if (MAX_LOCK_DURATION == 0u64 || additional_duration == 0u64) { 0u64 } else {
        // ...
    }
}
```

Suggestion:

Remove the redundant `MAX_LOCK_DURATION == 0u64` check to simplify the function logic.

Resolution:

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

VSO-4 Missing Critical Event Logging

Severity: Minor

Status: Fixed

Code Location:

veSOLID/sources/veSOLID.move#225 254 269

Descriptions:

In the `fund_pool`, `increase_time`, and `withdraw` functions, the contract performs critical fund movements and state changes but does not emit any event logs.

Detailed issues are as follows:

- The `fund_pool` function involves the administrator funding the reward pool, but it does not emit a `Funded` or similar event. This omission prevents on-chain tracking of the pool's funding sources and timestamps.
- The `increase_time` function updates the user's `lock_end`, `expected_total`, and the global `total_obligations`, but it does not emit a `LockExtended` or `DurationIncreased` event, making it impossible to audit lock extension history.
- The `withdraw` function handles user fund withdrawals and pool balance reductions, yet it does not emit a `Withdrawn` or similar event, hindering front-end UIs, analytics tools, and security monitoring systems from tracking fund flows accurately.

Suggestion:

Add event emissions for all critical operations.

Resolution:

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

VSO-5 Redundant `old_expected` Calculation Increases Logic Complexity

Severity: Informational

Status: Fixed

Code Location:

veSOLID/sources/veSOLID.move#205

Descriptions:

In the `increase_time` function, the developer defines a local variable:

```
let old_expected = position.expected_total;
```

and adjusts the pool's total obligations using the following logic:

```
pool.total_obligations = pool.total_obligations - old_expected;
// ...
position.expected_total = old_expected + delta_reward;
pool.total_obligations = pool.total_obligations + position.expected_total;
```

The intended logic is:

- Remove the previous expected reward value from the pool's obligations;
- Recalculate the new reward based on the extended lock duration;
- Update the total obligations accordingly.

However, this "subtract-then-add" pattern is unnecessary within this function because:

- `position.expected_total` is only updated locally (not referenced across functions) and can be directly reassigned to the new value;
- The subtraction and addition operations on `pool.total_obligations` cancel out the effect of the old value, adding unnecessary complexity to the logic.

Suggestion:

Directly update the lock position data and pool state after calculating `delta_reward`, without performing intermediate `old_expected` subtraction and addition operations.

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.