Bucket v2-move-contracts

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Fri Sep 12 2025

Bucket v2-move-contracts Audit Report

1 Executive Summary

1.1 Project Information

Description	This is a decentralized saving pool implementation built on Sui blockchain, and a sophisticated reward distribution system for saving pools.
Type	DeFi
Auditors	MoveBit
Timeline	Sun Aug 31 2025 - Mon Sep 01 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Bucket-Protocol/v2-move-contracts
Commits	05184abc02ce33922f2ebe8d9ef04993d340b977 de26faf9426d495bd10d551cec8fc87724f05d38 86f32fde16e7ef6214d2fb51bc2d937eafb3d44b 04f4d515eda77f49e1b70dab847f660a8bce912f d0d090de69a2a46a0fd5781e75153c3508f21ad3 e32b44a052873fd0b91b62c8bf38d65c7e5475c0

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV1	bucket_incentives/saving_incentive/Move.toml	e9d1c2aebd640001cbe86a46a0c4b00b779626c2
EVE	bucket_incentives/saving_incentive/sources/events.move	fcea5098604aecdce7e765655109d3d3f5044d91
MEM	bucket_incentives/saving_incentive/sources/memo.move	c63c33310d227f16b5a2bd29808e7900b2bf9be1
ICO	bucket_incentives/saving_incentive/sources/lib/incentive_config.move	231af7d5af087dc83a799b267c1fdcf6ddf72c28
MOV2	bucket_incentives/borrow_incentive/Move.toml	1409ec64ba056434b2384a3b5ce229d9d2266f98
BIN	bucket_incentives/borrow_incentive/sources/borrow_incentive.move	e0d20222b74a2b834347bc4f7d4b63ca97e4aa7a
BIE	bucket_incentives/borrow_incentive/sources/borrow_incentive_events.move	a5d4e59bf836860b8df9ee8042f4f202b3f73f1a
MOV5	bucket_saving/Move.toml	6c23ab4e1762aed052c05d3950e16cfb5da55d18
WIT	bucket_saving/sources/witness.move	77939217cdc5b0e1e41b247d9877095fe4d1bf72
VER	bucket_saving/sources/version.move	b935b5b9fae323237b65a97880c729e985201bc9

MOV	bucket_incentives/saving_incentive/Move.toml	573c85e488cb63582eef33bd4e18a4478368715f
EVE	bucket_incentives/saving_incentive/sources/events.move	a0bb0d479d24b1fa52a01de7ee5d402937de6e40
SIN	bucket_incentives/saving_incentive/sources/saving_incentive.move	4ccfbd6c5d7e360a0c05301ff2509018dea817d1
MOV3	bucket_saving/Move.toml	ca7b1ca4c729eb6187ec9e6bbff4a2976fa8466e
EVE1	bucket_saving/sources/events.move	8dea05b6619043d0afd5d01df80ca8161896790b
SAV	bucket_saving/sources/saving.move	7bccdca158371c5fb6a9a08b22fa1cb050e2ce14
SIN	bucket_incentives/saving_incentive/sources/saving_incentive.move	6a2a32e42de93116abd3b5d30bc7d7f2f2bb7438
EVE1	bucket_saving/sources/events.move	8b773fabd4a62dda491dd7b8fbfd32dc9116c739
SAV	bucket_saving/sources/saving.move	32f8cb413838bc829ca11bbca3eab9d890461dc6
SIN	bucket_incentives/saving_incentive/sources/saving_incentive.move	dc3be9a3f1fee60ad2231b5e172ba183839ef379
SAV	bucket_saving/sources/saving.move	4cbd351015ae92a2de099bf4b3fc11e28638567f

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	10	10	0
Informational	4	4	0
Minor	0	0	0
Medium	5	5	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 Bucket to identify any potential issues and vulnerabilities in the source code of the Bucket v2-move-contracts 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 10 issues of varying severity, listed below.

ID	Title	Severity	Status
SAV-1	Inflation Attack	Critical	Fixed
SAV-2	Unvalidated `withdrawal`	Medium	Fixed
SAV-3	Lack of Slippage Protection	Medium	Fixed
SAV-4	Lack of Version Control	Medium	Fixed
SAV-5	Unordered Hotpotato May Causing Incorrect `total_stake`	Medium	Fixed
SAV-6	Missing Check for `treasury_cap.total_supply()`	Medium	Fixed
SAV-7	Missing Position `last_update_timestamp` Update	Informational	Fixed
SAV-8	Lack of Event Emit	Informational	Fixed
SAV-9	Code Optimization	Informational	Fixed
SIN-1	Duplicate Error Code	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Bucket v2-move-contracts Smart Contract :

Admin

- Admin can create a new saving pool through the `new()` function.
- Admin can update the saving interest rate through the `update_saving_rate()` function.
- Admin can update the deposit cap limit through the `update_deposit_cap()` function.
- Admin can add a witness type requirement for deposit operations through the `add_deposit_response_check()` function.
- Admin can remove a witness type requirement for deposit operations through the `remove_deposit_response_check()` function.
- Admin can add a witness type requirement for withdraw operations through the `add_withdraw_response_check()` function.
- Admin can remove a witness type requirement for withdraw operations through the `remove_withdraw_response_check()` function.
- Admin can add version through the `add_version()` function.
- Admin can remove version through the `remove_version()` function.
- Admin can add manager through the `add_manager()` function.
- Admin can remove manager through the `remove_manager()` function.
- Admin can create a new reward manager through the `new_reward_manager()` function.
- Admin can add a new reward token through the `add_reward()` function.
- Admin can withdraw tokens from reward source through the `withdraw_from_source()` function.
- Admin can burn the `USDB` coin through the `burn()` function.

Manager

- Manager can update reward flow rate through the `update_flow_rate()` function.
- Manager can update reward start timestamp through the `update_reward_timestamp()` function.

User

- User can validate deposit responses through the `check_deposit_response()` function.
- User can validate withdraw responses through the `check_withdraw_response()` function.
- User can add witness proofs to deposit responses through the `add_deposit_witness()` function.
- User can add witness proofs to withdraw responses through the `add_withdraw_witness()` function.
- User can deposit USDB into the pool through the `deposit()` function.
- User can withdraw USDB from the pool through the `withdraw()` function.
- User can supply rewards through the `supply()` function.
- User can create a deposit checker through the `new_checker_for_deposit_action()` function.
- User can update reward state during deposit through the `update_deposit_action()` function.
- User can finalize deposit actions through the `destroy_deposit_checker()` function.
- User can claim rewards through the `claim()` function.
- User can create a withdrawal checker through the `new_checker_for_withdraw_action()` function.
- User can update reward state during withdrawal through the `update_withdraw_action()` function.

- User can finalize withdrawal actions through the `destroy_withdraw_checker()` function.

4 Findings

SAV-1 Inflation Attack

Severity: Critical

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#609,560

Descriptions:

In the contract, when there is no liquidity in the pool (i.e., the first stake), the deposit_ function directly uses the staked USDB amount (deposit_val) as the number of minted LP tokens. This means the first staker receives LP tokens at a 1:1 ratio. Users can also increase the usdb_reserve_balance using `supply()` . Subsequent stakers will calculate the LP using the `1:usdb_reserve_balance` ratio, potentially resulting in zero or loss of precision, leading to losses.

Suggestion:

It is recommended to add a collateral requirement to prevent the initial collateral amount from being too low, which can lead to economic imbalance. For example, some contracts limit the USD amount to no less than 1,000.

Resolution:

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

SAV-2 Unvalidated `withdrawal`

Severity: Medium

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#626

Descriptions:

In the `withdraw_()` function, when `lp_balance * reserve < supply`, the `withdrawal` value may be 0. This may result in the user not receiving the withdrawal but the corresponding LP being destroyed.

Suggestion:

It is recommended to check the `withdrawal` value and only perform subsequent operations if it is greater than 0. Or, limit the minimum LP withdrawal amount.

Resolution:

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

SAV-3 Lack of Slippage Protection

Severity: Medium

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#460,515

Descriptions:

The `deposit()` and `withdraw()` functions in the contract do not include slippage protection. Users cannot specify a minimum expected amount of LP tokens (for deposits) or a minimum amount of `USDB` (for withdrawals) when performing these operations. This means that users may receive less than their expected output amount due to changes in the pool state (such as other users' actions or price fluctuations), resulting in slippage losses. This can be particularly significant in situations of high volatility or low liquidity.

Suggestion:

Add parameters to the deposit and withdrawal functions to allow users to specify a minimum output amount. If the output amount falls below the threshold, the transaction should be rolled back.

Resolution:

The client has added an additional method to check for slippage. Invoking this method during deposits and withdrawals can help avoid losses due to slippage.

SAV-4 Lack of Version Control

Severity: Medium

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#586

Descriptions:

The `withdraw()` function lack of version control. If this is missing, users might call the deprecated function.

Suggestion:

It is suggested to add the version control logic in the `withdraw()` function.

Resolution:

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

SAV-5 Unordered Hotpotato May Causing Incorrect `total_stake`

Severity: Medium

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#526,586;

bucket_incentives/saving_incentive/sources/saving_incentive.move#417,518

Descriptions:

Users can deposit to multiple accounts simultaneously, generating multiple `DepositResponse` objects. Problems can arise if the order of these operations becomes disordered. For example, a user might generate `DepositResponse1` and `DepositResponse2` using two addresses in sequence, but first call `update_deposit_action()` on `DepositResponse2` to update `rewarder.total_stake`. Then, if `update_deposit_action()` is called on `DepositResponse1`, the updated `rewarder.total_stake` will be incorrect. Alternatively, a user might first call `withdraw()` to generate a `WithdrawResponse`, but not call `update_withdraw_action()` to update `rewarder.total_stake`. Then, they might call `deposit()` for another account, and then call `update_deposit_action()` to update `rewarder.total_stake`.

Suggestion:

It is recommended that the other party conduct more relevant inspections to ensure that there are no potential risks.

Resolution:

The client added stricter validation in the `position_locker` to mitigate the risk. Our team has attempted various methods to attack this design, and no effective exploitable behavior for profit has been identified so far. Therefore, we have updated the status to `Fixed`.

SAV-6 Missing Check for `treasury_cap.total_supply()`

Severity: Medium

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#301

Descriptions:

The `new()` function does not check whether `treasury_cap.total_supply()` is 0. When making the first deposit, if `supply` is not zero, the amount of LP minted is based on the formula `(deposit_val * supply) / usdb_reserve`, which may fail if usdb_reserve is 0. Even if division by zero is avoided, if the reserve is not zero but the ratio is not matched, new depositors will receive LP tokens at an unreasonable ratio, resulting in dilution or unfair distribution of value to existing LP holders.

Suggestion:

It is recommended that in the `new()` function, a check should be added to ensure that `treasury_cap.total_supply()` is 0, otherwise abort.

Resolution:

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

SAV-7 Missing Position `last_update_timestamp` Update

Severity: Informational

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#483,490,531

Descriptions:

The `last_update_timestamp` field of a position is updated only when a position is created, not when a position is added, which violates the semantics of this field.

Suggestion:

It is recommended that the `last_update_timestamp` field of a position be updated when modifying it.

Resolution:

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

SAV-8 Lack of Event Emit

Severity: Informational

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#340

Descriptions:

The `update_deposit_cap()` function in the contract lack event logging, which is essential for blockchain transparency, off-chain data tracking, and frontend integration. Event logs allow external systems to monitor contract activities without querying the blockchain state directly.

Suggestion:

It is recommended to add event emission for these operations.

Resolution:

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

SAV-9 Code Optimization

Severity: Informational

Status: Fixed

Code Location:

bucket_saving/sources/saving.move#586

Descriptions:

In the `distribute_interest()` function, if the `saving_rate()` value is 0, continuing to execute subsequent code after updating the timestamp is meaningless and wastes resources.

Suggestion:

It is recommended that in the `distribute_interest()` function, when the `saving_rate()` value is 0, only update the timestamp.

Resolution:

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

SIN-1 Duplicate Error Code

Severity: Informational

Status: Fixed

Code Location:

bucket_incentives/saving_incentive/sources/saving_incentive.move#37,42;

bucket_saving/sources/saving.move#28,32

Descriptions:

Using the same value for two different error semantics makes it difficult for callers or loggers to distinguish the specific cause. This can also cause errors in external monitoring logic, hindering the location of security incidents.

Suggestion:

We recommend assigning a unique number and a unified comment to each error; and updating the `abort` constant for the corresponding `err_*` functions.

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