Bucket Protocol
Smart Contract
Audit Report

MOVEBIT

contact@movebit.xyz

https://twitter.com/movebit_

06/19/2023

Bucket Protocol Smart Contract Audit Report

1 Executive Summary

1.1 Project Information

Description	CDP protocol built on Sui network
Type	DeFi
Auditors	MoveBit
Timeline	May 12, 2023 – Jun 16, 2023
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Bucket-Protocol/v1-core
Commits	30e51114deb408d2a5643d92643d66927c8ae692 c56e25d5b198a242b9b84775ddcecf27c18dc6a5

1.2 Files in Scope

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

ID	Files	SHA-1 Hash
BUC	./protocol/sources/buck.move	a288eca58fa266bc4ea07319 d567a2a5e50b1076

CON	./protocol/sources/config/const.move	2d65bbf04b54b936da1f034c96ff9d5974caa999
BUK	./protocol/sources/bucket.move	39522e08907e56e37da8effb728d348567276684
BKT	./protocol/sources/bkt.move	9cd2bba045de798ff4a30a546edca023a8b1ed8d
WEL	./protocol/sources/well.move	afac8c6d184daf9e6ecbd42b225963012f4ea6bf
BOT	./protocol/sources/bottle.move	2272461273faa8677eb3fa76e5e8b0f73e6f92ef
TAN	./protocol/sources/tank.move	1cde299ee336edfd21a59fa0cb98328582817fa4
VES	./framework/sources/vesting_lock.move	a6dbb4147c3b04121be8c30a6c177e9e33ee3457
LTB	./framework/sources/linked_table.move	a3c542c3ebfc5046e128056596a25a9830473f0b
MAT	./framework/sources/math.move	e6e1a451961f4bd8063a6d0de22403a6bb31baa9
EVE	./sources/config/events.move	65a4d3863e569c2758e667125a03f2f5cd6d3c87

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged	Partially Fixed
Total	11	9	1	1
Informational	2	2		
Minor	3	2	1	
Medium	2	2		
Major	4	3		1

Critical				

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

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 Protocol** to identify any potential issues and vulnerabilities in the source code of the **Bucket 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 **11** issues of varying severity, listed below.

ID	Title	Severity	Status
VES-01	Missing `start_time` Parameter Check	Minor	Fixed
BKT-02	`bkt_treasury_cap` Should Be Destroyed	Informational	Fixed
BKT-03	`BKT` Token Centralization Risk	Major	Partially Fixed

BUK-04	Update `minted_buck_amount` Logic Flaw	Major	Fixed
BUK-05	Infinite Loop In `handle_redeem`	Major	Fixed
BUK-06	`compute_weight` May Be 0	Minor	Fixed
BUK-07	`remaining_redemption_amount` May Not Be 0	Minor	Acknowledged
BUK-08	`bottle_table` Might Be Out of Order	Medium	Fixed
BUK-09	`token.start_p` Is Not Updated	Major	Fixed
BUK-10	Gas Optimization	Informational	Fixed
BUK-11	Collateral Maybe Insufficient When Repaying	Medium	Fixed

3 Participant Process

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

Admin

- Admin can create a `vesting_lock` to lock arbitrarily amount `BKT` through `allocate_bkt()` .

User

- User can top up collateral of user through `top_up()` .
- User can borrow `BUCK` through `borrow()` .
- User can repay `BUCK` and get collateral of user through `repay()` .
- User can redeem `BUCK` and get collateral of user through `redeem()` .
- User can withdraw `bkt_reward` and `collateral_withdrawal` from `Tank` through `withdraw()` .
- User can deposit `BUCK` into `Tank` through `deposit()` .

- User can stake `BKT` into `Well` through `stake()`.
- User can get rewards from `Well` through `claim()`.

4 Findings

VES-01 Missing `start_time` Parameter Check

Severity: Minor

Status: Fixed

Code Location: framework/sources/vesting_lock.move#L25

Descriptions: In the `vesting_lock` module, the `new` function creates `VestingLock` and lacks the check of `start_time`. It is recommended to ensure that `start_time` is greater than or equal to the current time.

Suggestion: Add assert to limit the `start_time` must be greater than the current `timestamp`.

Resolution: The client confirmed the issue and fix this issue according to the suggestion.

BKT-02 `bkt_treasury_cap` Should Be Destroyed

Severity: Informational

Status: Fixed

Code Location: protocol/sources/bkt.move#L55

Descriptions: In order to fix the total amount of `BKT` tokens, it is recommended to use the `coin::treasury_into_supply` methods of destroying `bkt_treasury_cap` instead of `transfer::public_freeze_object` and locking it into an object.

Suggestion: Use `coin::treasury_into_supply` to destroy `bkt_treasury_cap`.

Resolution: The client confirmed the issue and fixed this issue according to the suggestion.

BKT-03 `BKT` Token Centralization Risk

Severity: Major

Status: Partially Fixed

Code Location: protocol/sources/bkt.move#L69

Descriptions: The `allocate_bkt` function has too much authority and can be locked up by anyone, in any amount, at any time, which creates a risk of centralization.

Suggestion: Manage `BktAdminCap` with a multi-signature account to mitigate the risk.

Resolution: The client confirmed the issue and partially fixed this issue according to the suggestion.

BUK-04 Update `minted_buck_amount` Logic Flaw

Severity: Major

Status: Fixed

Code Location: protocol/sources/bucket.move#L125

Descriptions: In the `handle_redeem` function, `buck_input_amount` has been updated to zero after `buck_input_amount` is subtracted from `bottle_buck_amount`, so the number of `minted_buck_amount` will not decrease.

Suggestion: Use a variable to save the value of `buck_input_amount` in advance.

Resolution: The client confirmed the issue and fix this issue according to the suggestion.

BUK-05 Infinite Loop In `handle_redeem`

Severity: Major

Status: Fixed

Code Location: protocol/sources/bucket.move#L125

Descriptions: The `handle_redeem` function may have an infinite loop, for example, there is only one `bottle`, and the `buck_input_amount` value is greater than the `buck_amount` of the current bottle. Because `buck_input_amount` > `bottle_buck_amount`, linked_table will use `push_back` to add a bottle with a `buck_amount` of 0 to the `linked_table`. At this time, the length of the `linked_table` will increase by one, and the `buck_input_amount` will not decrease, because `buck_amount` always be 0 after, which case infinite loop.

Suggestion: Break out of the loop when a `Bottle` with a debt value of 0.

Resolution: The client confirmed the issue and fix this issue according to the suggestion.

BUK-06 `compute_weight` May Be 0

Severity: Minor

Status: Fixed

Code Location: protocol/sources/bucket.move#125

Descriptions: In the calculation of the `compute_weight` function, the value of `stake_amount` may be less than `MAX_LOCK_TIME/lock_time`, resulting in a return value of 0, and the user has no benefit.

Suggestion: Limit the `weight` of the user to greater than 0 after `stake`.

Resolution: The client confirmed the issue and fix this issue according to the suggestion.

BUK-07 `remaining_redemption_amount` May Not Be 0

Severity: Minor

Status: Acknowledged

Code Location: protocol/sources/bucket.move#L184

Descriptions: In the `handle_redeem` function, if the `buck_input_amount` can repay all the bottles, it may cause the `remaining_redemption_amount` to remain and not equal to 0. The restriction of the `assert` may be too strict, or add another judgment to determine whether all the bottles have been repaid.

Suggestion: It can be added that when there is no `bottle` with debt in the `bottle_table`, it can also successfully pass the judgment of the `assert` statement.

BUK-08 `bottle_table` Might Be Out of Order

Severity: Medium

Status: Fixed

Code Location: protocol/sources/bucket.move#L184

Descriptions: In the `handle_redeem` function, if the `buck_input_amount` is fully repaid for a part of the `Bottle`, the debt of this `Bottle` will be 0. If it directly pushes at the end of the

linked list, the order of the `linked_table` will be out of order.

Suggestion: Ensure that the `linked_table` is in an ordered state after each insertion.

Resolution: The client confirmed the issue and fix this issue according to the suggestion.

BUK-09 `token.start_p` Is Not Updated

Severity: Major

Status: Fixed

Code Location: protocol/sources/bucket.move

Descriptions: After calling the `claim_collateral` function in the `tank` module to collect rewards, the value of `start_p` in the token is not updated in time, which may cause logic errors.

Suggestion: `token.start_p` needs to be updated to the latest value of `tank.current_p` after `claim_collateral` called.

Resolution: The client confirmed the issue and fix this issue according to the suggestion.

BUK-10 Gas Optimization

Severity: Informational

Status: Fixed

Code Location: protocol/sources/bucket.move

Descriptions: When updating the value of `tank.current_p`, there is no need to calculate it again. The value of `new_p` has been calculated in advance and can be reused directly.

Suggestion: Use the `new_p` variable directly.

Resolution: The client confirmed the issue and fix this issue according to the suggestion.

BUK-11 Collateral Maybe Insufficient When Repaying

Severity: Medium

Status: Fixed

Code Location: protocol/sources/bucket.move

Descriptions: When calling `record_repay_capped` for the repayment, the amount of collateral calculated to be repaid may exceed the amount of `collateral_amount` in the bottle, resulting in a situation where `bottle.collateral_amount` is less than the returned amount of collateral `return_sui_amount`.

Suggestion: According to the `bottle.collateral_amount` and the calculated `return_sui_amount`, dynamically determine how much the user really needs to repay.

Resolution: The client confirmed the issue and fix this issue according to the suggestion.

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.