MOVEBIT
Securing the Move Ecosystem

PatronusFi Contract **Audit Report**

https://twitter.com/movebit_

contact@movebit.xyz

PatronusFi Contract Audit

1 Executive Summary

1.1 Project Information

Type	Lending
Auditors	MoveBit
Timeline	2023–03–08 to 2023–04–06
Languages	Move
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	Repository: git@github.com:patronusfi/contracts.git Received Commit: 371a893c1986865505b04fe2ef78cae52ecb4c4c Last Reviewed Commit: ad54c2a6406e87ca8ba8b4e3facdb63febc6b319

1.2 Issue Statistic

Item	Count	Fixed	Pending	Confirmed
Total	7	6		1
Minor	3	2		1
Medium	2	2		
Major	2	2		

Critical				

1.3 Issue Level

- **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.

1.4 Issue Status

- **Fixed:** The issue has been resolved.
- **Pending:** The issue has been acknowledged by the code owner, but has not yet been resolved. The code owner may take action to fix it in the future.
- **Confirmed:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

2 Summary of Findings

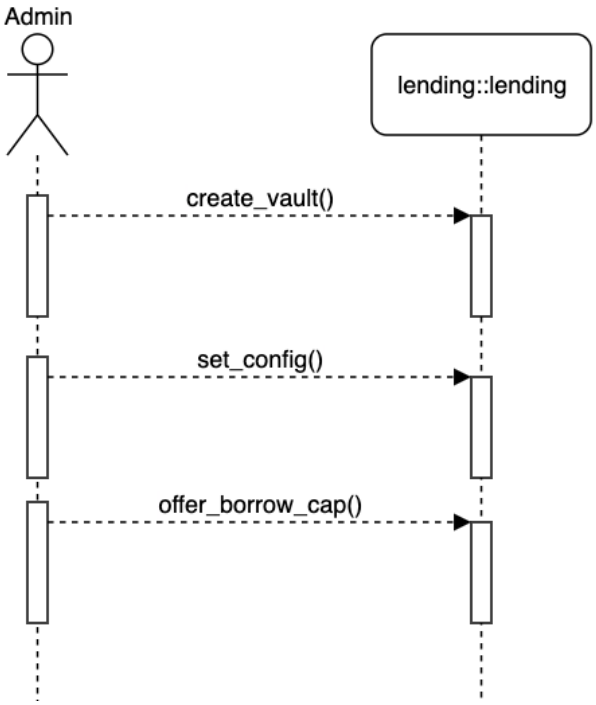
Patronus is a DeFi Liquidity Protocol which aims to provide a suite of DeFi products and services for a safer and more efficient capital flow. Our team mainly focused on reviewing the code security and normative. During the testing process, our team also maintains close communication with the project team to ensure that we have a correct understanding of business requirements. As a result, our team found a total of 7 issues. The audit and project teams discussed these issues together, and the project solved and confirmed all the issues. We also wrote some MSL specs to ensure the core modules are working properly, and have sent them to the project team. We also raised some suggestions for the project team to write move specs for other modules, and the project team accept all of them to make the contract safer.

3 Participant Process

Here are the relevant actors with their respective abilities within the `patronus.fi` Smart Contract:

Admin

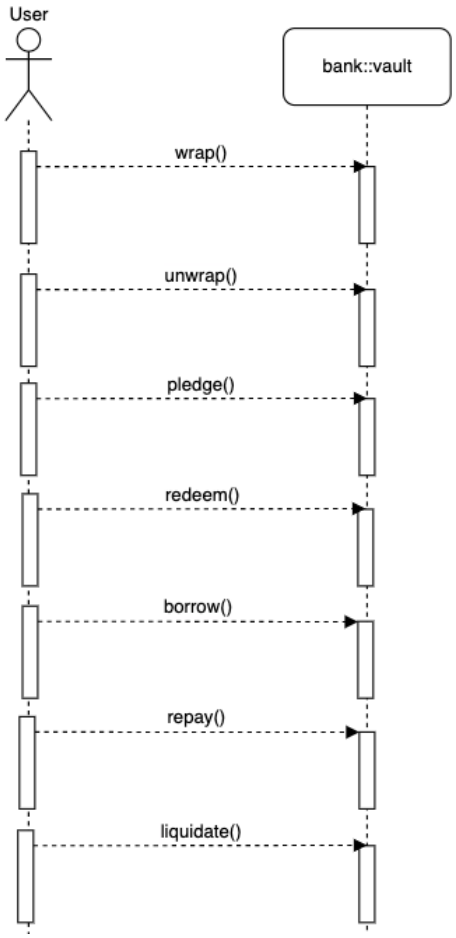
- Creators can `create_vault`.
- Creators can `set_config`.
- Creators can `offer_borrow_cap`.



lending::lending

User

- User can `wrap` coin.
- User can `unwrap` coin.
- User can `pledge`.
- User can `redeem`.
- User can `borrow`.
- User can `repay`.
- User can `liquidate`.



bank::vault

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

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

Code scope sees **Appendix 1**.

(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.

6 Findings

6.1 Configuration update error

Severity: Major

Status: Fixed

Descriptions: The parameter judgment of the update configuration is wrong, and should judge the `new_cfg`, If the value of the `new_cfg` exceeds the limitation in those `assert`, it would be set at the first time, and since then, no new configuration can be set later.

Code Location: bank/sources/config.move:52, lending/sources/config.move:60

config.move

```
public(friend) fun update(cfg: &mut Config, new_cfg: Config) {
    assert!(cfg.asset_factor <= C1E9, error::out_of_range(E_BAD_CONFIG));
    assert!(cfg.debt_factor >= C1E9, error::out_of_range(E_BAD_CONFIG));
    assert!(cfg.liquidation_threshold > cfg.asset_factor, error::out_of_range(E_BAD_CONFIG));
    assert!(cfg.liquidation_factor > C1E9, error::out_of_range(E_BAD_CONFIG));
    assert!(math::mult(cfg.liquidation_factor, cfg.liquidation_threshold) < C1E9, error::out_of_range(E_BAD_CONFIG));

    *cfg = new_cfg;
}
```

Suggestion: Modify the code as below.

config.move

```
assert!(new_cfg.asset_factor <= C1E9, error::out_of_range(E_BAD_CONFIG));
assert!(new_cfg.debt_factor >= C1E9, error::out_of_range(E_BAD_CONFIG));
assert!(new_cfg.liquidation_threshold > new_cfg.asset_factor, error::out_of_range(E_BAD_CONFIG));
assert!(new_cfg.liquidation_factor > C1E9, error::out_of_range(E_BAD_CONFIG));
assert!(math::mult(new_cfg.liquidation_factor, new_cfg.liquidation_threshold) < C1E9, error::out_of_range(E_BAD_CONFIG));
```

6.2 Logical Error

Severity: Major

Status: Fixed

Descriptions: When the temporary value of `borrow_cap` is true, the status of the `vault.paused` in line 512 will be verified as true and set to false. When running to the next if statement, the state of the `vault.paused` is already false, which will cause the contract to always panic.

Code Location: bank/sources/vault.move:477.

vault.move

```
if (borrow_cap.temporary) {
    assert!(vault.paused, error::invalid_state(E_VAULT_PAUSED));
    vault.paused = false;
} else {
    assert!(!vault.paused, error::invalid_state(E_VAULT_PAUSED));
};
...
// must repay all if temporary
if (borrow_cap.temporary) {
    assert!(vault.paused, error::invalid_state(E_VAULT_PAUSED));
    let amount_to_repay = unscale_borrowed(state.interest_index, borrow_cap.scaled_borrowed);
    assert!(amount_to_repay == amount, error::invalid_state(E_TOO_FEW_TO_REPAY));
...
```

Suggestion: Delete line 526, and modify the code as below.

lending.move

```
if (borrow_cap.temporary) {
    let amount_to_repay = unscale_borrowed(state.interest_index, borrow_cap.scaled_borrowed);
    assert!(amount_to_repay == amount, error::invalid_state(E_TOO_FEW_TO_REPAY));
    vault.paused = true;

```

6.3 Lack check of parameter

Severity: Minor

Status: Fixed

Descriptions: There is no limit to the amount greater than 0, and there is no judgment that the balance of `wcoin` is greater than the parameter `amount` passed in.

Code Location: lending/sources/lending.move:484.

lending.move

```
public fun pledge<CoinType>(user: &signer, wcoins: Coin<W<CoinType>>)
        acquires Balance, Pool, ConfigStore
{
...
let amount = coin::value(&wcoins);
let config_store = borrow_global<ConfigStore>(@lending);
let cfg = table::borrow(&config_store.configs, coin_type);
let asset = get_asset(&mut balance.assets, coin_type);
...
```

lending.move

```
public fun redeem<CoinType>(user: &signer, amount: u64): Coin<W<CoinType>>
        acquires Balance, Pool, ConfigStore
{
let coin_type = type_of<CoinType>();
vault::accrue_interest(coin_type);

let user_addr = address_of(user);
let balance = borrow_global_mut<Balance>(user_addr);
let asset = get_asset(&mut balance.assets, coin_type);
asset.wcoin_amount = asset.wcoin_amount - amount;
...
```

Suggestion: It is recommended to add assert to ensure that `asset.wcoin_amount` is greater than `amount`.

6.4 Centralization Risks

Severity: Minor

Status: Confirmed

Descriptions: The bank can offer a borrow capability to any address, same time a borrow capability can extract coins from the vault indefinitely.

Code Location: bank/sources/vault.move:168.

vault.move

```
public entry fun offer_borrow_cap<CoinType>(bank: &signer, to: address, borrow_fee: u64, borrow_limit: u64, temporary: bool)
        acquires Vault
{
    assert_bank(bank);
    assert!(borrow_fee <= 5000000, error::out_of_range(E_BAD_CONFIG));


    let vault = borrow_global_mut<Vault<CoinType>>(@bank);


    if (table::contains(&vault.pending_borrow_caps, to)) {
        let pending_borrow_cap = table::borrow_mut(&mut vault.pending_borrow_caps, to);
        pending_borrow_cap.borrow_fee = borrow_fee;
        pending_borrow_cap.borrow_limit = borrow_limit;
    } else {
        let pending_borrow_cap = PendingBorrowCapability {
            inited: false,
            temporary,
            borrow_fee,
            borrow_limit,
        };
        table::add(&mut vault.pending_borrow_caps, to, pending_borrow_cap);
    };
}
```

Suggestion: Change the bank address to be a `resource account`, so as to attribute the control right to the contract, or add an upper limit on the number of borrows in addition to the `BorrowCapability` offered to `lending`.

Confirmed: The `offer_borrow_cap` is called only by the administrator. The project team is planning to rotate the control right to a multi-sign account, so it would help to control the creation of `PendingBorrowCapability`, and make sure no abuse of `PendingBorrowCapability`.

6.5 Vault may already exist

Severity: Minor

Status: Fixed

Descriptions: When creating a vault, `create_vault` does not judge whether the vault under the bank address exists, or judges whether the token has been registered in `wcoin::create`.

Code Location: bank/sources/vault.move:104.

vault.move

```
public fun create_vault<CoinType>(bank: &signer, vault_config: Config)
        acquires StateStore
{
    assert_bank(bank);

    let coin_type = type_of<CoinType>();
    assert!(type_info::account_address(&coin_type) != @wcoin, error::inval
id_argument(E_BAD_CONFIG));
    let state_store = borrow_global_mut<StateStore>(@bank);
    let (burn_cap, freeze_cap, mint_cap) = wcoin::create<CoinType>(&state_
store.wcoin_signer_cap);
    ...
    move_to(bank, Vault {
        mint_cap,
        burn_cap,
        freeze_cap,
        convert_events: new_event_handle(bank),
        coins: coin::zero<CoinType>(),
        paused: false,
        pending_borrow_caps: table::new(),
    });
```

Suggestion: Add code: `assert!(!coin::is_coin_initialized<CoinType>(), 0);`.

6.6 Compile Error

Severity: Medium

Status: Fixed

Descriptions: The entry function has a return value causing the project to fail to compile.

Code Location: bank/sources/vault.move:449

vault.move

```
public entry fun withdraw_reserved_coins<CoinType>(bank: &signer): Coin<CoinType>
    acquires Vault, StateStore
{
    assert_bank(bank);

    let vault = borrow_global_mut<Vault<CoinType>>(@bank);
    let state_store = borrow_global_mut<StateStore>(@bank);
    let coin_type = type_of<CoinType>();
    let state = table::borrow_mut(&mut state_store.states, coin_type);
    accrue_interest_internal(state);
    ...
```

Suggestion: Add `checked_deposit` function to deposit token at the end of `withdraw_reserved_coins`, and remove the return value `Coin<CoinType>`.

```
public entry fun withdraw_reserved_coins<CoinType>(bank: &signer)
    acquires Vault, StateStore
{
    assert_bank(bank);

    let vault = borrow_global_mut<Vault<CoinType>>(@bank);
    let state_store = borrow_global_mut<StateStore>(@bank);
    let coin_type = type_of<CoinType>();
    let state = table::borrow_mut(&mut state_store.states, coin_type);
    accrue_interest_internal(state);
    ...

    checked_deposit(user, coins);
}
```

6.7 Unit Tests Failed

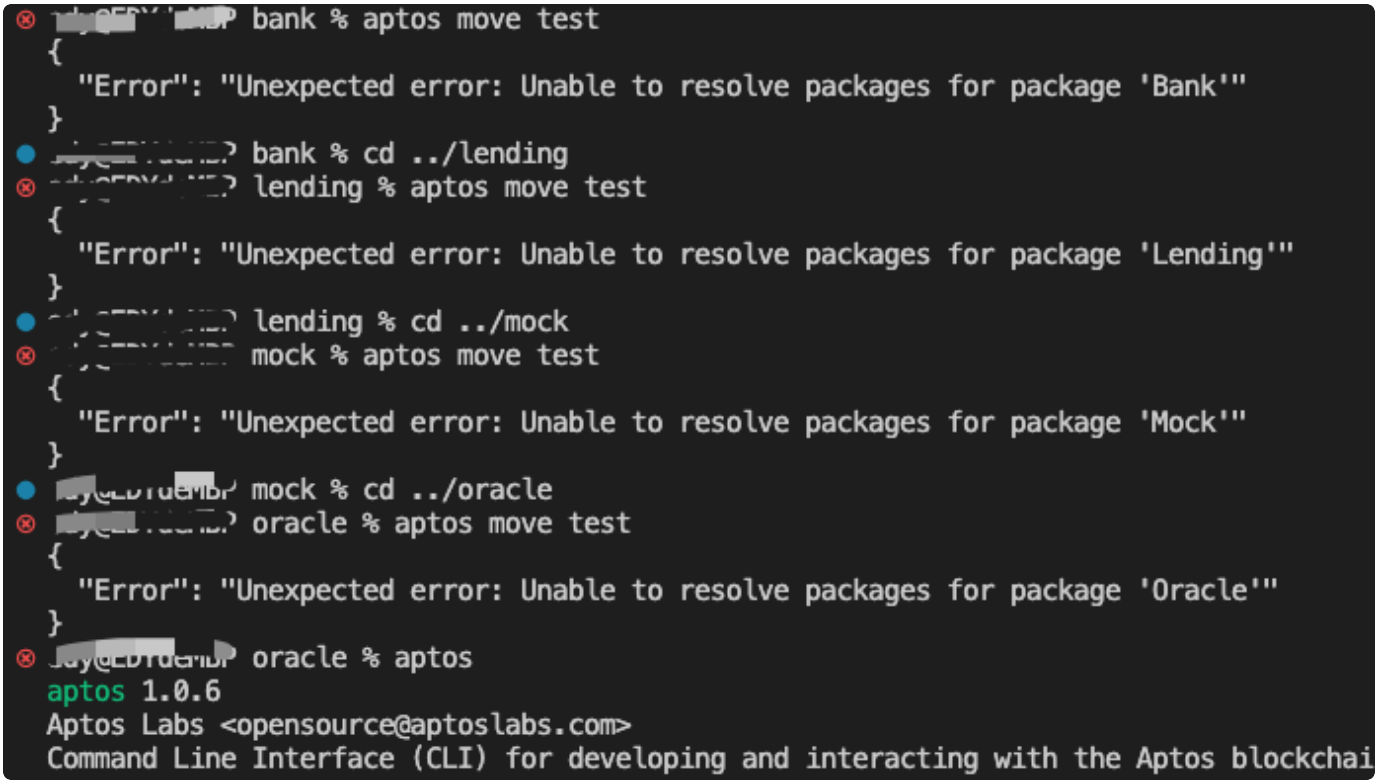
Severity: Medium

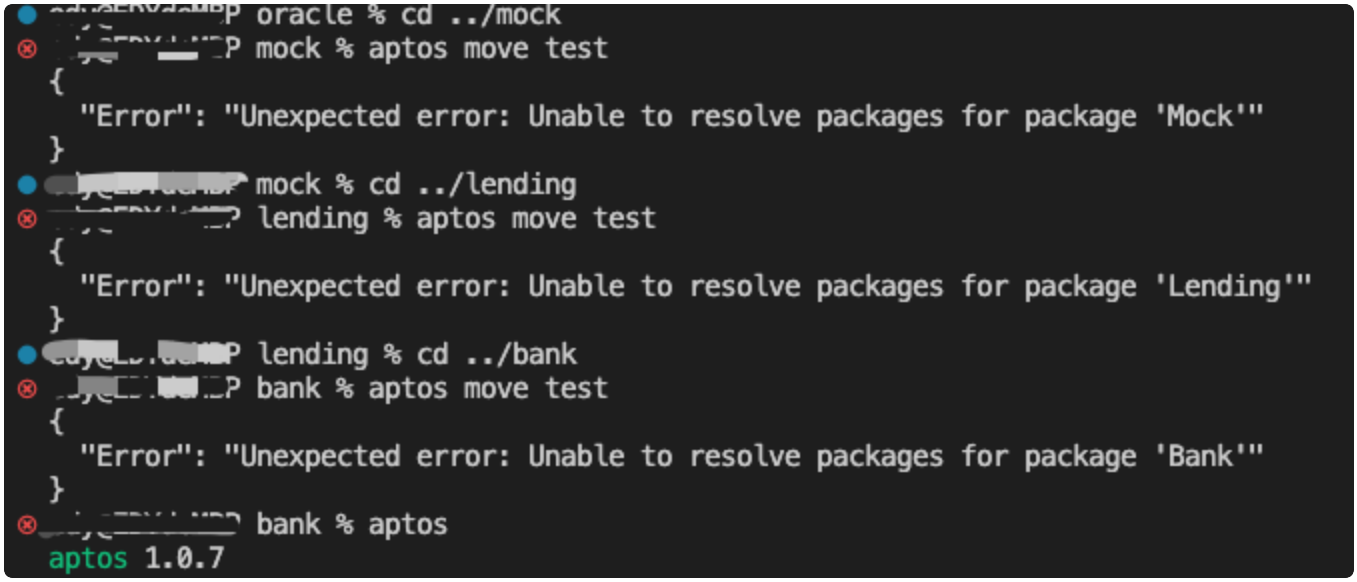
Status: Fixed

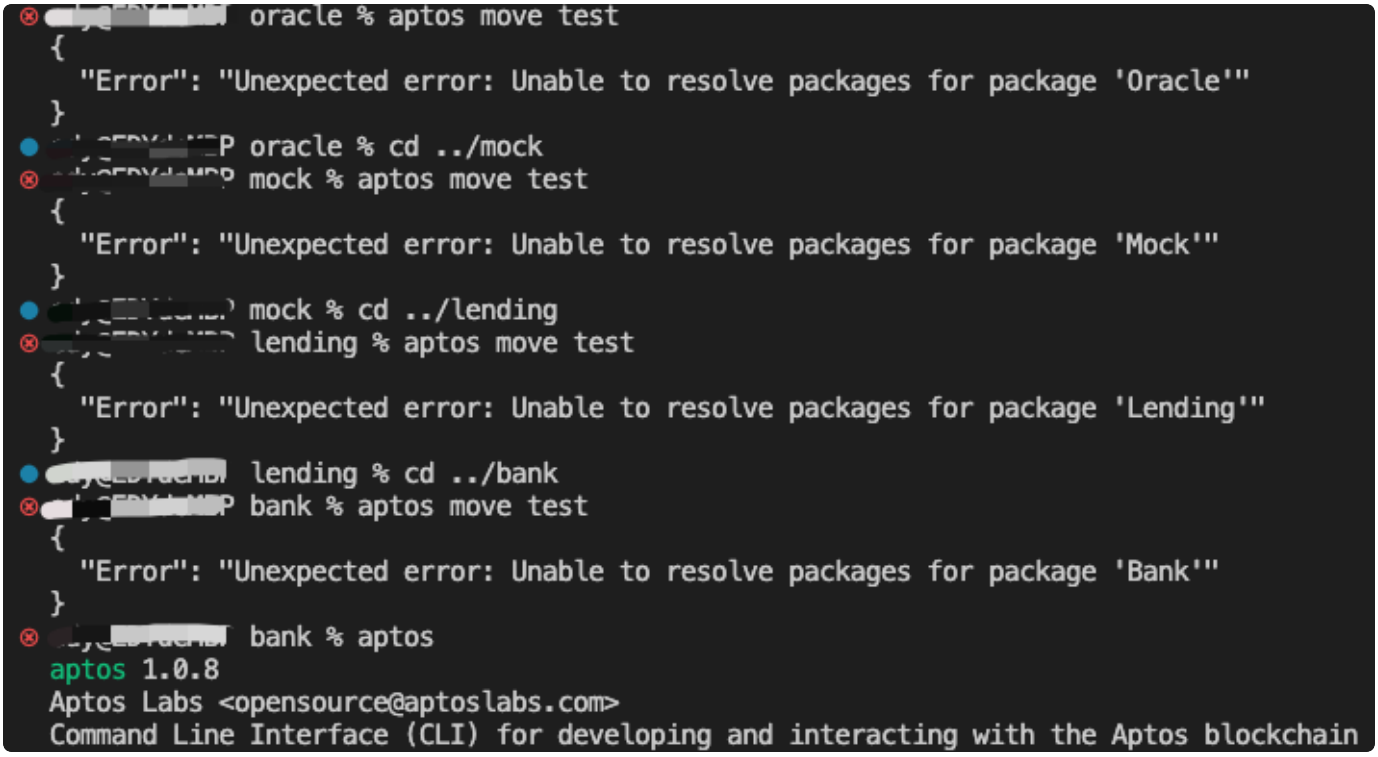
Descriptions: The unit test command cannot be executed, and the package cannot be found due to an error. We try to solve this problem by modifying the package address, but the unit test

command still cannot be executed; The aptos-cli versions `1.06`, `1.07,` and `1.08` cannot be executed either.

Code Location: bank/tests、lending/tests、mock/tests、oracle/tests.







7 Prover Formal Verification

Our team added formal specifications to some important features using Move Prover, Move Prover is a tool that uses Formal Verification to verify smart contracts written in the Move language. Formal Verification is a technique that uses rigorous mathematical methods to describe behavior and reason about the correctness of computer systems. Now it has been adopted in operating systems, compilers, and other fields where correctness is prioritized.

The formal verification report of some files and modules is as follows.

7.1 bank::vault

`Vault` deposits and generates interest by pledging the user's original tokens, and exchanges `wcoin` certificate tokens. `Vault` records the overall utilization rate, amount, etc. to provide correct interest and other information.

7.1.1 bank::vault::wrap

- When minting `wcoin` by `coin`, if the amount of `wcoin` is zero, then the final minted `w coin` is equal to the input `coin` amount.
- And the change of coin in `vault` should be as expected.

vault.spec.move

```
spec wrap<CoinType>(coins: Coin<CoinType>): Coin<W<CoinType>> {
    let coin_type = type_of<CoinType>();
    let state_store = global<StateStore>(@bank);
    let state = table::spec_get(state_store.states, coin_type);
    let coin_amount = coin::value(coins);
    let vault = global<Vault<CoinType>>(@bank);
    let post vault_post = global<Vault<CoinType>>(@bank);

    ensures state.total_wcoins == 0 ==> coin::value(result) == coin_amount;
    ensures coin::value(vault_post.coins) == coin::value(vault.coins) + coin_a
mount;
}
```

7.1.2 bank::vault::borrow

- After the user borrows, the number of `coins` in the `vault` should decrease by the amount borrowed.
- And the first return value should be the borrowed coin.

vault.spec.move

```
spec borrow<CoinType>(borrow_cap: &mut BorrowCapability<CoinType>, amount: u64
): (Coin<CoinType>, u64) {
    let vault = global<Vault<CoinType>>(@bank);
    let post vault_post = global<Vault<CoinType>>(@bank);

    ensures coin::value(vault_post.coins) + amount == coin::value(vault.coins)
;
    ensures coin::value(result_1) == coin::value(vault.coins) - coin::value(va
ult_post.coins);
}
```

7.1.3 bank::vault::repay

- After the user repays the loan, the `coins` in the `vault` increases accordingly.

vault.spec.move

```
spec repay<CoinType>(borrow_cap: &mut BorrowCapability<CoinType>, coins: Coin<
CoinType>): u64 {
    let vault = global<Vault<CoinType>>(@bank);
    let post vault_post = global<Vault<CoinType>>(@bank);
    ensures coin::value(vault_post.coins) == coin::value(vault.coins) + coin::
value(coins);
}
```

7.1.4 bank::vault::withdraw_reserved_coins

- After the user withdraws, the `coins` in the `vault` decrease accordingly.

vault.spec.move

```
spec withdraw_reserved_coins<CoinType>(bank: &signer): Coin<CoinType> {
    let coin_type = type_of<CoinType>();
    let state_store = global<StateStore>(@bank);
    let state = table::spec_get(state_store.states, coin_type);
    let vault = global<Vault<CoinType>>(@bank);
    let post vault_post = global<Vault<CoinType>>(@bank);
    let amount = state.reserved_coins;
    ensures coin::value(vault_post.coins) == coin::value(vault.coins) - amount
;
}
```

7.2 Suggestion

In addition to the `bank::vault` module, other modules can also be verified by the Prover tool, such as the `lending::lending` module, which implements the core logic of user deposit, withdrawal, borrowing, and repayment. For example, we can try to verify the following properties:

- Whether the user increases/decreases `wcoin` as expected after depositing/withdrawing.
- Is the change in debt after the user borrows and repays normal?
- Some asset changes after liquidation.

We recommend that the development team check the core code by writing specs to avoid vulnerabilities that cannot be detected by manual audits.

Appendix 1 – Files in Scope

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

Files	SHA-1 Hash
./wcoin/sources/wcoin.move	b0feb999399f99c111874618a3bb5a9f7bcbb8c6
./bank/tests/vault_test.move	05b9d343ecac322ab3093bd48d1e335b91202edb
./bank/tests/test_common.move	fc75bed0144d7c5e311d972ded08f1b6a24f0726
./bank/sources/periphery.move	3357e64d37a75bad4380bf33c3f5651348642db3
./bank/sources/fixed64x64.move	17e9c7580f0056ed52fdc3ba359d55f3ea621d81
./bank/sources/math.move	21f1fb9940bf6128dff5a4c13d002f1e986e020f
./bank/sources/vault.move	87e5c0ce35f2aaccab6cf01d9457db9f0aa8b1af
./bank/sources/config.move	e31de5d35616059a6fa44f05b24c6c9ce60d7603
./mock/sources/test_coin.move	209dbc6b0334ba6b2a92c164698bf79853c8f291
./oracle/tests/oracle_test.move	0e95380a58dea4097cae9c6f3550fb525288f4b2
./oracle/sources/oracle.move	35d382a46eebad86107fc5964a10cb528a806fba
./lending/tests/lending_test.move	b1ed42968792834cc753af206f0a1b5461218765
./lending/sources/periphery.move	a997187acc7c51dff1b55994873047439bec8d7e
./lending/sources/lending.move	3c32965d6feed8910cf7d42244777c8b85215413
./lending/sources/config.move	496e521433cb4279b0971d526bb2c0b6826ad139
./wcoin/Move.toml	32c442b98bb21e28576edcba1876d664e5d7af6b
./bank/Move.toml	ea938bd4acdfffbdc0e5d633d5434fb978e06d7d
./mock/Move.toml	5c4d61a2fe5a8aa2f11dc1a0cee4952ebe365376
./oracle/Move.toml	e7a2f00a48268a3c3f8cfaee8a3fc31ee71f1938
./lending/Move.toml	4ed5ccdb10b889e404514799e3795ed1e9412b1c

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.

MOVEBIT

Securing the Move Ecosystem

https://twitter.com/movebit_

contact@movebit.xyz