Security Audit Report for Ref-Exchange, Burrowland, Boost-Farm

Date: Jan 25th, 2024

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2024.01.26 10:58:42 +08'00'

Contents

Report Manifest

Item	Description
Client	Ref Finance
Target	Ref-Exchange, Burrowland, Boost-Farm

Version History

Version	Date	Description
1.0	January 25th, 2024	First Version

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Rust
Approach	Semi-automatic and manual verification

The repository that has been audited includes ref-exchange [1], burrowland [2], boost-farm [3].

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following. Our audit report is responsible for the only initial version (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Burrowland	Version 1	21c3655e6b9fa26c83bda255be374f5118aae208
	Version 2	7b074fd85f4fc7eb3c129e8b7f553c9a42975090
Ref-Exchange	Version 1	3b6321d644ce96f163194f1fac6ae44003c90340
	Version 2	a9339d4674ea5722d3a6e8a8735017b5f3506cf8
Boost-Farm	Version 1	a6568e2f69e3c34541ac2fa7e48132a4c87a63b0
	Version 2	876a48b8b2fd279fa39f6331541f4cb11d870ad5

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include **burrowland/contracts/contract/src**, **ref-contracts/ref-exchange/src**, and **boost-farm/contracts/boost-farming/src** folder contract only. Specifically, the files covered in this audit include:

```
burrowland/contracts/contract/src
 |-- account_asset.rs
 |-- account_farm.rs
 |-- account.rs
 |-- account_view.rs
 |-- actions.rs
 |-- asset_config.rs
 |-- asset_farm.rs
 |-- asset.rs
 |-- asset_view.rs
 |-- big_decimal.rs
 |-- booster_staking.rs
 |-- config.rs
 |-- events.rs
 |-- fungible_token.rs
 |-- legacy.rs
```

[1] https://github.com/ref-finance/ref-contracts/tree/lp_as_collateral

[2] https://github.com/burrowHQ/burrowland/tree/lp_as_collateral

[3] https://github.com/ref-finance/boost-farm/tree/lp_as_collateral

```
|-- lib.rs
|-- pool.rs
|-- position.rs
|-- price_receiver.rs
|-- prices.rs
|-- shadow_actions.rs
|-- storage.rs
|-- storage_tracker.rs
|-- upgrade.rs
|-- utils.rs

ref-contracts/ref-exchange/src
|-- account_deposit.rs
|-- action.rs
|-- admin_fee.rs
|-- custom_keys.rs
|-- errors.rs
|-- legacy.rs
|-- lib.rs
|-- multi_fungible_token.rs
|-- owner.rs
|-- pool.rs
|-- rated_swap
|  |-- linear_rate.rs
|  |-- math.rs
|  |-- mod.rs
|  |-- nearx_rate.rs
|  |-- rate.rs
|  |-- README.md
|  |-- stnear_rate.rs
|-- shadow_actions.rs
|-- simple_pool.rs
|-- stable_swap
|  |-- curve.md
|  |-- math.rs
|  |-- mod.rs
|-- storage_impl.rs
|-- token_receiver.rs
|-- unit_lpt_cumulative_infos.rs
|-- utils.rs
|-- views.rs

boost-farm/contracts/boost-farming/src
|-- actions_of_farmer_reward.rs
|-- actions_of_farmer_seed.rs
|-- actions_of_seed.rs
|-- big_decimal.rs
|-- booster.rs
|-- errors.rs
|-- events.rs
|-- farmer.rs
|-- farmer_seed.rs
|-- legacy.rs
```

```
|-- lib.rs
|-- management.rs
|-- owner.rs
|-- seed_farm.rs
|-- seed.rs
|-- shadow_actions.rs
|-- storage_impl.rs
|-- token_receiver.rs
|-- utils.rs
|-- view.rs
```

Listing 1.1: Audit Scope for this Report

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [4] and Common Weakness Enumeration [5]. The

[4] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[5] https://cwe.mitre.org/

overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we find **six** potential issues. We also have **three** recommendations as follows:

- High Risk: 6
- Medium Risk: 0
- Low Risk: 0
- Recommendations: 3
- Notes: 0

ID	Severity	Description	Category	Status
1	High	Lack of Consistency Validation between Arguments position and asset_amount	Software Security	Fixed
2	High	Permanently Locked User	Software Security	Fixed
3	High	Lack of Consistent Precision in on_remove_shadow()	Software Security	Fixed
4	High	Potential Drain of Funds due to Early Return of Shares	Software Security	Fixed
5	High	Persistent Use of Outdated Seed State	Software Security	Fixed
6	High	Improper LP Token Collateral Price Calculation	DeFi Security	Fixed
7	-	Redundant Check of out_assets	Recommendation	Fixed
8	-	Lack of Check for withdraw token	Recommendation	Fixed
9	-	Lack of Lock Check for Liquidation Account	Recommendation	Fixed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Lack of Consistency Validation between Arguments position and asset_amount

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `PositionDecreaseCollateral` and `PositionIncreaseCollateral` actions accept two arguments: `position` and `asset_amount`. If the specified `position` is a LP token position, then the `asset_amount`'s `token_id` field should be the corresponding LP token. However, there is no check.

```
83    Action::PositionIncreaseCollateral { position, asset_amount } => {
84        need_number_check = true;
85        if position == REGULAR_POSITION {
86            assert!(!asset_amount.token_id.to_string().starts_with(SHADOW_V1_TOKEN_PREFIX));
87        }
88        let amount = self.internal_increase_collateral(&position, account, &asset_amount);
89        events::emit::increase_collateral(&account_id, amount, &asset_amount.token_id, &position);
90    }
```

Listing 2.1: burrowland/contracts/contract/src/actions.rs

```
105    Action::PositionDecreaseCollateral { position, asset_amount } => {
106        risk_check_positions.insert(position.clone());
107        let mut account_asset =
108            account.internal_get_asset_or_default(&asset_amount.token_id);
109        let amount = self.internal_decrease_collateral(
110            &position,
111            &mut account_asset,
112            account,
113            &asset_amount,
114        );
115        account.internal_set_asset(&asset_amount.token_id, account_asset);
116        events::emit::decrease_collateral(&account_id, amount, &asset_amount.token_id, &position);
117    }
```

Listing 2.2: burrowland/contracts/contract/src/actions.rs

Impact Users can pass mismatched `position` and `asset_amount` arguments, using cheaper LP tokens as collateral posing as more expensive LP tokens, or withdrawing cheaper LP tokens while redeeming back more expensive LP tokens. This results in incoherent internal accounting.

Suggestion I Add a check to ensure the `position` must match the `asset_amount`.

2.1.2 Permanently Locked User

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `callback_process_shadow_force_close_result()` function should unlock liquidated accounts after forced closure attempts. However, the callback may trigger panics when decrementing `asset.reversed` by amount. Although `internal_shadow_force_close()` verifies that `asset.reversed` must be greater than `amount`, this situation cannot be guaranteed in the callback function `callback_process_shadow_force_close_result()`.

```
409    pub fn callback_process_shadow_force_close_result (
410        &mut self,
411        liquidation_account_id: AccountId,
412        position: String,
413        collateral_sum: BigDecimal,
414        repaid_sum: BigDecimal,
415    ) {
416        let mut liquidation_account = self.internal_unwrap_account(&liquidation_account_id);
417        liquidation_account.is_locked = false;
418
419        if is_promise_success() {
420            if let Position::LPTokenPosition(position_info) = liquidation_account.positions.remove
                    (&position).unwrap(){
421                liquidation_account.add_affected_farm(FarmId::Supplied(AccountId::new_unchecked(
                        position_info.lpt_id.clone())));
422                for (token_id, shares) in position_info.borrowed {
423                    let mut asset = self.internal_unwrap_asset(&token_id);
```

```
424                 let amount = asset.borrowed.shares_to_amount(shares, true);
425
426                 asset.reserved -= amount;
427                 asset.borrowed.withdraw(shares, amount);
428
429                 self.internal_set_asset(&token_id, asset);
430
431                 liquidation_account.add_affected_farm(FarmId::Borrowed(token_id));
432             }
433             self.internal_account_apply_affected_farms(&mut liquidation_account);
434             events::emit::force_close(&liquidation_account_id, &collateral_sum, &repaid_sum, &
                    position);
435         }
436     }
437     self.internal_set_account(&liquidation_account_id, liquidation_account);
438 }
```

Listing 2.3: burrowland/contracts/contract/src/shadow_actions.rs

Impact Failing to unlock liquidated accounts will permanently freeze it from interacting with its deposited assets, borrowed assets and assets that were used as collateral.

Suggestion I Do subtraction in function `internal_shadow_force_close()` and rollback it in function `callback_process_shadow_force_close_result()` if `on_burrow_liquidation()` fails.

2.1.3 Lack of Consistent Precision in on_remove_shadow()

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `on_cast_shadow()` multiplies the parameter `amount` by `10**extra_decimals` for the purpose of scaling, but function `on_remove_shadow()` directly uses parameter `amount` with no extra decimals multiplied.

```
66  pub fn on_cast_shadow(&mut self, account_id: AccountId, shadow_id: String, amount: U128, msg:
        String) {
67      let config = self.internal_config();
68      assert!(env::predecessor_account_id() == config.ref_exchange_id);
69
70      let actions = if !msg.is_empty() {
71          match near_sdk::serde_json::from_str(&msg).expect("Can't parse ShadowReceiverMsg") {
72              ShadowReceiverMsg::Execute { actions } => actions,
73          }
74      } else {
75          vec![]
76      };
77
78      let token_id = AccountId::new_unchecked(shadow_id);
79      let asset = self.internal_unwrap_asset(&token_id);
80      let amount = amount.0 * 10u128.pow(asset.config.extra_decimals as u32);
81
```

```
82        let mut account = self.internal_unwrap_account(&account_id);
83        self.internal_deposit(&mut account, &token_id, amount);
84        events::emit::deposit(&account_id, amount, &token_id);
85        self.internal_execute(&account_id, &mut account, actions, Prices::new());
86        self.internal_set_account(&account_id, account);
87    }
88
89    pub fn on_remove_shadow(&mut self, account_id: AccountId, shadow_id: String, amount: U128, msg
          : String) {
90        let config = self.internal_config();
91        assert!(env::predecessor_account_id() == config.ref_exchange_id);
92
93        let mut account = self.internal_unwrap_account(&account_id);
94
95        if !msg.is_empty() {
96            let actions = match near_sdk::serde_json::from_str(&msg).expect("Can't parse
                  ShadowReceiverMsg") {
97                ShadowReceiverMsg::Execute { actions } => actions,
98            };
99            self.internal_execute(&account_id, &mut account, actions, Prices::new());
100       }
101
102       let token_id = AccountId::new_unchecked(shadow_id);
103       let withdraw_asset_amount = AssetAmount {
104           token_id,
105           amount: Some(amount),
106           max_amount: None,
107       };
108       let mut asset = self.internal_unwrap_asset(&withdraw_asset_amount.token_id);
109       let mut account_asset = account.internal_unwrap_asset(&withdraw_asset_amount.token_id);
110       let (shares, amount) =
111           asset_amount_to_shares(&asset.supplied, account_asset.shares, &withdraw_asset_amount,
                  false);
112
113       let available_amount = asset.available_amount();
114       assert!(
115           amount <= available_amount,
116           "Withdraw error: Exceeded available amount {} of {}",
117           available_amount,
118           &withdraw_asset_amount.token_id
119       );
120
121       account_asset.withdraw_shares(shares);
122       account.internal_set_asset(&withdraw_asset_amount.token_id, account_asset);
123
124       asset.supplied.withdraw(shares, amount);
125
126       self.internal_set_asset(&withdraw_asset_amount.token_id, asset);
127       self.internal_set_account(&account_id, account);
128       events::emit::withdraw_succeeded(&account_id, amount, &withdraw_asset_amount.token_id);
129   }
```

Listing 2.4: burrowland/contracts/contract/src/shadow_actions.rs

Impact The inconsistency between the two functions will lead to incorrect accounting.

Suggestion I Scaling the input parameter `amount` in the same extra decimals.

2.1.4 Potential Drain of Funds due to Early Return of Shares

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `ShadowActions FromFarming` and `FromBurrowland` will directly increase the `free_shares` of LP tokens, allowing users to remove liquidity in the same block. In this case, if the function `on_remove_shadow()` fails in the contract of `Boost Farming` or `Burrowland`, the LP tokens still remain in these two contracts while the liquidity can be removed.

```
61    #[payable]
62    pub fn shadow_action(&mut self, action: ShadowActions, pool_id: u64, amount: Option<U128>, msg
          : String) -> PromiseOrValue<bool> {
63        self.assert_contract_running();
64        let shadow_id = pool_id_to_shadow_id(pool_id);
65        let prev_storage = env::storage_usage();
66        let sender_id = env::predecessor_account_id();
67        let mut account = self.internal_unwrap_account(&sender_id);
68        let pool = self.pools.get(pool_id).expect(ERR85_NO_POOL);
69        let total_shares = pool.share_balances(&sender_id);
70        let (amount, max_amount) = match action {
71            ShadowActions::ToFarming => {
72                let available_amount = if let Some(record) = account.get_shadow_record(pool_id) {
73                    record.available_farming_shares(total_shares)
74                } else {
75                    total_shares
76                };
77                (amount.unwrap_or(U128(available_amount)).0, available_amount)
78            }
79            ShadowActions::ToBurrowland => {
80                let available_amount = if let Some(record) = account.get_shadow_record(pool_id) {
81                    record.available_burrowland_shares(total_shares)
82                } else {
83                    total_shares
84                };
85                (amount.unwrap_or(U128(available_amount)).0, available_amount)
86            }
87            ShadowActions::FromFarming => {
88                let in_farming_amount = if let Some(record) = account.get_shadow_record(pool_id) {
89                    record.shadow_in_farm
90                } else {
91                    0
92                };
93                (amount.unwrap_or(U128(in_farming_amount)).0, in_farming_amount)
94            }
95            ShadowActions::FromBurrowland => {
96                let in_burrowland_amount = if let Some(record) = account.get_shadow_record(pool_id)
                      {
```

```
                record.shadow_in_burrow
            } else {
                0
            };
            (amount.unwrap_or(U128(in_burrowland_amount)).0, in_burrowland_amount)
        }
    };
    assert!(amount > 0, "amount must be greater than zero");
    assert!(amount <= max_amount, "amount must be less than or equal to {}", max_amount);

    account.update_shadow_record(pool_id, &action, amount);
    self.internal_save_account(&sender_id, account);

    let contract_id = match action {
        ShadowActions::FromBurrowland | ShadowActions::ToBurrowland => {
            self.burrowland_id.clone()
        }
        ShadowActions::FromFarming | ShadowActions::ToFarming => {
            self.boost_farm_id.clone()
        }
    };

    match action {
        ShadowActions::ToFarming | ShadowActions::ToBurrowland => {
            let storage_fee = self.internal_check_storage(prev_storage);
            ext_shadow_receiver::on_cast_shadow(
                    sender_id.clone(),
                    shadow_id,
                    U128(amount),
                    msg,
                    &contract_id,
                    0,
                    GAS_FOR_ON_CAST_SHADOW
                )
                .then(ext_self::callback_on_shadow(
                        action,
                        sender_id,
                        pool_id,
                        U128(amount),
                        U128(storage_fee),
                        &env::current_account_id(),
                        0,
                        GAS_FOR_ON_CAST_SHADOW_CALLBACK
                    )
                )
                .into()
        }
        ShadowActions::FromFarming | ShadowActions::FromBurrowland => {
            let storage_fee = if prev_storage > env::storage_usage() {
                (prev_storage - env::storage_usage()) as Balance * env::storage_byte_cost()
            } else {
                0
            };
```

```
150                 ext_shadow_receiver::on_remove_shadow(
151                         sender_id.clone(),
152                         shadow_id,
153                         U128(amount),
154                         msg,
155                         &contract_id,
156                         0,
157                         GAS_FOR_ON_CAST_SHADOW
158                     )
159                     .then(ext_self::callback_on_shadow(
160                             action,
161                             sender_id,
162                             pool_id,
163                             U128(amount),
164                             U128(storage_fee),
165                             &env::current_account_id(),
166                             0,
167                             GAS_FOR_ON_CAST_SHADOW_CALLBACK
168                         )
169                     )
170                     .into()
171             }
172         }
173     }
```

Listing 2.5: ref-contract/ref-exchange/src/shadow_actions.rs

Impact The funds can be drained in `Boost Farming` and `Burrowland`.

Suggestion I Revise the corresponding logic.

2.1.5 Persistent Use of Outdated Seed State

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Functions `get_seed()` and `get_seed_unwrap()` prioritize fetching data from the outdated `Farmer.seeds`. Only if there is no data there, they will retrieve data from the new `Farmer.vseeds`. However, the function `set_seeds()` directly inserts data into `Farmer.vseeds`, regardless of the presence or absence of existing seeds in `Farmer.seeds`.

In summary, the state originally stored in `Farmer.seeds` remains unchanged, while its outdated data continues to be utilized for updates to `Farmer.vseeds`.

```
75     pub fn get_seed_unwrap(&self, seed_id: &SeedId) -> FarmerSeed {
76         if let Some(seed) = self.seeds.get(seed_id) {
77             seed.into()
78         } else {
79             self.vseeds.get(seed_id).unwrap().into()
80         }
81     }
82
```

```
83    pub fn get_seed(&self, seed_id: &SeedId) -> Option<FarmerSeed> {
84        if let Some(seed) = self.seeds.get(seed_id) {
85            Some(seed.into())
86        } else {
87            self.vseeds.get(seed_id).map(|v| v.into())
88        }
89    }
90
91    pub fn remove_seed(&mut self, seed_id: &SeedId) {
92        if self.seeds.remove(seed_id).is_none() {
93            self.vseeds.remove(seed_id);
94        }
95    }
96
97    pub fn set_seed(&mut self, seed_id: &SeedId, seed: FarmerSeed) {
98        self.vseeds.insert(seed_id, &seed.into());
99    }
```

Listing 2.6: boost-farm/contracts/boost-farming/src/farmer.rs

Impact The persistent utilization of outdated data will lead to inaccuracies in updating both `Farmer.seeds` and `Farmer.vseeds`.

Suggestion I Migrate the state from `Farmer.seeds` to `Farmer.vseeds` correctly.

2.2 DeFi Security

2.2.1 Improper LP Token Collateral Price Calculation

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `get_collateral_sum_with_volatility_ratio()` is used to calculate the total collateral value of a position. The calculation method will sum the prices of the underlying tokens that the staked LP tokens can redeem. However, the quantity of underlying tokens that the LP tokens can redeem can be easily manipulated, the corresponding price of staked LP tokens can be manipulated as well.

```
182   pub fn get_collateral_sum_with_volatility_ratio(&self, position_info: &Position, prices: &
          Prices) -> BigDecimal {
183       match position_info {
184           Position::RegularPosition(regular_position) => {
185               regular_position
186               .collateral
187               .iter()
188               .fold(BigDecimal::zero(), |sum, (token_id, shares)| {
189                   let asset = self.internal_unwrap_asset(&token_id);
190                   let balance = asset.supplied.shares_to_amount(*shares, false);
191                   sum + BigDecimal::from_balance_price(
192                       balance,
193                       prices.get_unwrap(&token_id),
194                       asset.config.extra_decimals,
```

```
195                 )
196                 .mul_ratio(asset.config.volatility_ratio)
197             })
198         }
199         Position::LPTokenPosition(lp_token_position) => {
200             let collateral_asset = self.internal_unwrap_asset(&AccountId::new_unchecked(
                    lp_token_position.lpt_id.clone()));
201             let collateral_shares = lp_token_position.collateral;
202             let collateral_balance = collateral_asset.supplied.shares_to_amount(
                    collateral_shares, false);
203             let unit_share_tokens = self.last_lp_token_infos.get(&lp_token_position.lpt_id).
                    expect("lp_token_infos not found");
204             let config = self.internal_config();
205             assert!(env::block_timestamp() - unit_share_tokens.timestamp <= to_nano(config.
                    lp_tokens_info_valid_duration_sec), "LP token info timestamp is too stale");
206             let unit_share = 10u128.pow(unit_share_tokens.decimals as u32);
207             unit_share_tokens.tokens
208                 .iter()
209                 .fold(BigDecimal::zero(), |sum, unit_share_token_value|{
210                     let token_asset = self.internal_unwrap_asset(&unit_share_token_value.
                            token_id);
211                     let token_stdd_amount = unit_share_token_value.amount.0 * 10u128.pow(
                            token_asset.config.extra_decimals as u32);
212                     let token_balance = u128_ratio(token_stdd_amount, collateral_balance, 10u128
                            .pow(collateral_asset.config.extra_decimals as u32) * unit_share);
213                     sum + BigDecimal::from_balance_price(
214                         token_balance,
215                         prices.get_unwrap(&unit_share_token_value.token_id),
216                         token_asset.config.extra_decimals,
217                     )
218                     .mul_ratio(token_asset.config.volatility_ratio)
219                 }).mul_ratio(collateral_asset.config.volatility_ratio)
220         }
221     }
222 }
```

Listing 2.7: burrowland/contracts/contract/src/position.rs

Impact Attackers can exploit manipulated collateral prices to borrow excess funds or force the liquidation of positions that are not in risk.

Suggestion I Revise the logic of LP token collateral price calculation.

2.3 Additional Recommendation

2.3.1 Redundant Check of out_assets

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The action `Liquidate` already ensures that `out_assets.len() == 1`, so the additional check of `!out_assets.is_empty()` is redundant.

```
148     Action::Liquidate {
149         account_id: liquidation_account_id,
150         in_assets,
151         out_assets,
152         position
153     } => {
154         assert_ne!(
155             account_id, &liquidation_account_id,
156             "Can't liquidate yourself"
157         );
158         let position = position.unwrap_or(REGULAR_POSITION.to_string());
159         if position == REGULAR_POSITION {
160             assert!(!in_assets.is_empty() && !out_assets.is_empty());
161             self.internal_liquidate(
162                 account_id,
163                 account,
164                 &prices,
165                 &liquidation_account_id,
166                 in_assets,
167                 out_assets,
168             );
169         } else {
170             assert!(!in_assets.is_empty() && !out_assets.is_empty()
171                 && out_assets.len() == 1 && out_assets[0].token_id.to_string() == position);
172             let mut in_asset_tokens = HashSet::new();
173             in_assets.iter().for_each(|v| assert!(in_asset_tokens.insert(&v.token_id), "Duplicate
                    assets!"));
174             let mut temp_account = account.clone();
175             self.internal_shadow_liquidate(
176                 &position,
177                 account_id,
178                 &mut temp_account,
179                 &prices,
180                 &liquidation_account_id,
181                 in_assets,
182                 out_assets,
183             );
184             let mut liquidation_account = self.internal_unwrap_account(&liquidation_account_id);
185             liquidation_account.is_locked = true;
186             account.is_locked = true;
187             self.internal_set_account(&liquidation_account_id, liquidation_account);
188         }
189     }
```

Listing 2.8: burrowland/contracts/contract/src/actions.rs

Suggestion I Remove the check of `!out_assets.is_empty()`

2.3.2 Lack of Check for withdraw token

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The action `Withdraw` will execute successfully but revert later in the callback function if `asset_amount.token_id` refers to a LP token, which is a waste of gas.

```
71    Action::Withdraw(asset_amount) => {
72        let amount = self.internal_withdraw(account, &asset_amount);
73        self.internal_ft_transfer(account_id, &asset_amount.token_id, amount);
74        events::emit::withdraw_started(&account_id, amount, &asset_amount.token_id);
75    }
```

Listing 2.9: burrowland/contracts/contract/src/actions.rs

Suggestion I Add a check to ensure the asset to be withdrawn is not a LP token in action `Withdraw` for gas saving.

2.3.3 Lack of Lock Check for Liquidation Account

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When the actions `ForceClose` and `Liquidate` are executed, they lock the targeted account and unlock it in the callback function. However, they fail to verify if the account to be closed or liquidated is already locked. Otherwise, if multiple accounts want to liquidate the same position, they will all succeed until the callback function.

```
148   Action::Liquidate {
149       account_id: liquidation_account_id,
150       in_assets,
151       out_assets,
152       position
153   } => {
154       assert_ne!(
155           account_id, &liquidation_account_id,
156           "Can't liquidate yourself"
157       );
158       let position = position.unwrap_or(REGULAR_POSITION.to_string());
159       if position == REGULAR_POSITION {
160           assert!(!in_assets.is_empty() && !out_assets.is_empty());
161           self.internal_liquidate(
162               account_id,
163               account,
164               &prices,
165               &liquidation_account_id,
166               in_assets,
167               out_assets,
168           );
169       } else {
170           assert!(!in_assets.is_empty() && !out_assets.is_empty()
171               && out_assets.len() == 1 && out_assets[0].token_id.to_string() == position);
172           let mut in_asset_tokens = HashSet::new();
173           in_assets.iter().for_each(|v| assert!(in_asset_tokens.insert(&v.token_id), "Duplicate
                assets!"));
174           let mut temp_account = account.clone();
175           self.internal_shadow_liquidate(
```

```
176                 &position,
177                 account_id,
178                 &mut temp_account,
179                 &prices,
180                 &liquidation_account_id,
181                 in_assets,
182                 out_assets,
183             );
184             let mut liquidation_account = self.internal_unwrap_account(&liquidation_account_id);
185             liquidation_account.is_locked = true;
186             account.is_locked = true;
187             self.internal_set_account(&liquidation_account_id, liquidation_account);
188         }
189     }
190     Action::ForceClose {
191         account_id: liquidation_account_id,
192         position
193     } => {
194         assert_ne!(
195             account_id, &liquidation_account_id,
196             "Can't liquidate yourself"
197         );
198         let position = position.unwrap_or(REGULAR_POSITION.to_string());
199         if position == REGULAR_POSITION {
200             self.internal_force_close(&prices, &liquidation_account_id);
201         } else {
202             self.internal_shadow_force_close(&position, &prices, &liquidation_account_id);
203             let mut liquidation_account = self.internal_unwrap_account(&liquidation_account_id);
204             liquidation_account.is_locked = true;
205             self.internal_set_account(&liquidation_account_id, liquidation_account);
206         }
207     }
```

Listing 2.10: burrowland/contracts/contract/src/actions.rs

Suggestion I Add corresponding check for `Liquidation Account` in `ForceClose` and `Liquidate` actions.