

Security Audit

Report for Burrowland, Ref-Dcl, Ref-Exchange

Date: May 22, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction	1
1.1 About Target Contracts	1
1.2 Disclaimer	2
1.3 Procedure of Auditing	2
1.3.1 Software Security	3
1.3.2 DeFi Security	3
1.3.3 NFT Security	3
1.3.4 Additional Recommendation	3
1.4 Security Model	3
Chapter 2 Findings	5
2.1 DeFi Security	5
2.1.1 Lack of sender account check when handling SwapReference message .	5
2.1.2 Lack of lock when decreasing collateral	6
2.1.3 Incorrect enumeration of tokens requiring price feeds	7
2.1.4 Unreasonable pos_id design	9
2.1.5 Lack of reasonable configuration check	12
2.1.6 Potential panic during handling message SwapReference	13
2.1.7 Unreasonable check of reserves	18
2.1.8 Potential sandwich attack in force close position token swap	22
2.1.9 Unreasonable leverage rate computation	26
2.2 Additional Recommendation	26
2.2.1 Automatically construct swap indication from the token information	26
2.2.2 Use UnorderedMap for margin_positions instead of HashMap	32
2.2.3 Incorrect error message in get_token_out()	33

Report Manifest

Item	Description
Client	Ref Finance
Target	Burrowland, Ref-Dcl, Ref-Exchange

Version History

Version	Date	Description
1.0	May 22, 2024	First release

Signature

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec 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 successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. 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 target of this audit is the code repository of Burrowland¹, Ref-Dcl², Ref-Exchange³ of Ref Finance. Note that, we did **NOT** audit all the modules in the repository. Specifically, the files covered in this audit include:

```
1 burrowland/contracts/contract/src/upgrade.rs
2 burrowland/contracts/contract/src/events.rs
3 burrowland/contracts/contract/src/price_receiver.rs
4 burrowland/contracts/contract/src/legacy.rs
5 burrowland/contracts/contract/src/config.rs
6 burrowland/contracts/contract/src/lib.rs
7 burrowland/contracts/contract/src/account_view.rs
8 burrowland/contracts/contract/src/margin_actions.rs
9 burrowland/contracts/contract/src/margin_trading.rs
10 burrowland/contracts/contract/src/margin_pyth.rs
11 burrowland/contracts/contract/src/margin_config.rs
12 burrowland/contracts/contract/src/fungible_token.rs
13 burrowland/contracts/contract/src/big_decimal.rs
14 burrowland/contracts/contract/src/margin_accounts.rs
15 burrowland/contracts/contract/src/asset_config.rs
16 burrowland/contracts/contract/src/account.rs
17 burrowland/contracts/contract/src/asset_view.rs
18 burrowland/contracts/contract/src/pyth.rs
19 burrowland/contracts/contract/src/prices.rs
20 burrowland/contracts/contract/src/storage.rs
21 burrowland/contracts/contract/src/shadow_actions.rs
22 burrowland/contracts/contract/src/margin_position.rs
23 burrowland/contracts/contract/src/utils.rs
24
25 ref-contracts/ref-exchange/src/account_deposit.rs
26 ref-contracts/ref-exchange/src/token_receiver.rs
27
28 ref-dcl-lending/contracts/dcl/src/user_asset.rs
29 ref-dcl-lending/contracts/dcl/src/dcl/utils.rs
30 ref-dcl-lending/contracts/dcl/src/api/token_receiver.rs
```

Listing 1.1: Audit Scope for this Report

¹https://github.com/burrowHQ/burrowland/tree/margin_trading

²https://github.com/ref-finance/ref-dcl/tree/margin_trading

³https://github.com/ref-finance/ref-contracts/tree/margin_trading

The auditing process is iterative. Specifically, we would 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 table. Our audit report is responsible for the code in the initial version ([Version 1](#)), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Burrowland	Version 1	74462d7e2a299acc0b9702ca278926614c4f4cc8
	Version 2	e3731c30ab8ef63e071077fad8213fb9166f5d3e
Ref Exchange	Version 1	9c3797aecf58f0f210ebe73b28b9552345e431f7
Ref Dcl	Version 1	70fbc5b70685afc52113636e5154e5dbfd414b65

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
- * Permission management
- * 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 Weak-

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

ness Enumeration ⁵. The 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	<i>High</i>	High	Medium
	<i>Low</i>	Medium	Low
		<i>High</i>	<i>Low</i>
		Likelihood	

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.

⁵<https://cwe.mitre.org/>

Chapter 2 Findings

In total, we found **nine** potential security issues. Besides, we have **three** recommendations.

- High Risk: 4
- Medium Risk: 1
- Low Risk: 4
- Recommendation: 3

ID	Severity	Description	Category	Status
1	High	Lack of sender account check when handling SwapReference message	DeFi Security	Fixed
2	High	Lack of lock when decreasing collateral	DeFi Security	Fixed
3	High	Incorrect enumeration of tokens requiring price feeds	DeFi Security	Fixed
4	Low	Unreasonable pos_id design	DeFi Security	Fixed
5	Low	Lack of reasonable configuration check	DeFi Security	Fixed
6	High	Potential panic during handling message SwapReference	DeFi Security	Fixed
7	Low	Unreasonable check of reserves	DeFi Security	Fixed
8	Low	Potential sandwich attack in force close position token swap	DeFi Security	Confirmed
9	Medium	Unreasonable leverage rate computation	DeFi Security	Fixed
10	-	Automatically construct swap indication from the token information	Recommendation	Confirmed
11	-	Use UnorderedMap for margin_positions instead of HashMap	Recommendation	Fixed
12	-	Incorrect error message in get_token_out()	Recommendation	Fixed

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of sender account check when handling SwapReference message

Severity High

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description The function `ft_on_transfer()` does not check if the `sender` is a registered [Ref V1](#) or [Ref V2](#) contract when handling the `SwapReference` message.

```
88 TokenReceiverMsg::SwapReference { swap_ref } => {
89     let mut account = self.internal_unwrap_margin_account(&swap_ref.account_id);
90     if swap_ref.op == "open" {
91         self.on_open_trade_return(&mut account, amount, &swap_ref);
92     } else if swap_ref.op == "decrease"
93         || swap_ref.op == "close"
```



```

94         || swap_ref.op == "liquidate"
95         || swap_ref.op == "forceclose"
96     {
97         let event = self.on_decrease_trade_return(&mut account, amount, &swap_ref);
98         events::emit::margin_decrease_succeeded(&swap_ref.op, event);
99     }
100     self.internal_set_margin_account(&swap_ref.account_id, account);
101     return PromiseOrValue::Value(U128(0));
102 }

```

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

Impact An attacker can impersonate a [REF V1](#) or [REF V2](#) contract by simply sending the custom [SwapReference](#) message to the contract. The contract would then correspondingly reduce the user's debt and add assets, while the attacker does not incur any cost.

Suggestion I Add a check to ensure that the [sender](#) must be [REF v1](#) or [REF V2](#) contract.

2.1.2 Lack of lock when decreasing collateral

Severity High

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description When a user opens a margin position, the [token_c_shares](#) value is set appropriately, while the [token_d_shares](#) and [token_p_amount](#) values are set to zero. These [token_d_shares](#) and [token_p_amount](#) values are intended to be reset upon receiving the [SwapReference](#) message from the [Ref V1](#) or [Ref V2](#) contract. In between these two processes, this margin position is locked.

However, the function [internal_margin_decrease_collateral\(\)](#) does not check if the margin position is locked. This allows the user to remove almost all of the collateral, putting the margin position in a state where it can be liquidated or force closed.

```

224 pub(crate) fn internal_margin_decrease_collateral(
225     &mut self,
226     account: &mut MarginAccount,
227     pos_id: &PosId,
228     amount: Balance,
229     prices: &Prices,
230 ) -> AccountId {
231     let margin_config = self.internal_margin_config();
232     let mut mt = account
233         .margin_positions
234         .get(pos_id)
235         .expect("Position not exist")
236         .clone();
237     let token_id = mt.token_c_id.clone();
238     let asset = self.internal_unwrap_asset(&mt.token_c_id);
239     let shares = asset.supplied.amount_to_shares(amount, true);
240
241     // collateral can NOT decrease to 0

```

```
242     assert!(  
243         mt.token_c_shares.0 > shares.0,  
244         "Not enough collateral to decrease"  
245     );  
246     mt.token_c_shares.0 -= shares.0;  
247  
248     assert!(  
249         !self.is_mt_liquidatable(&mt, prices, margin_config.min_safety_buffer),  
250         "Margin position would be below liquidation line"  
251     );  
252     assert!(  
253         !self.is_mt_forcecloseable(&mt, prices),  
254         "Margin position would be below forceclose line"  
255     );  
256  
257     assert!(  
258         self.get_mtp_lr(&mt, prices).unwrap()  
259         <= BigDecimal::from(margin_config.max_leverage_rate as u32),  
260         "Leverage rate is too high"  
261     );  
262  
263     account.deposit_supply_shares(&mt.token_c_id, &shares);  
264     account.margin_positions.insert(pos_id.clone(), mt);  
265  
266     token_id  
267 }
```

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

Impact After opening the position, an attacker can immediately withdraw almost all of the collateral and trigger the swap action of the contract. Despite having slippage protection, there is still some room for a sandwich attack during this swap.

When [Ref V1](#) or [Ref V2](#) returns the swap result, the margin position may be in force-closable state and need to be closed using the contract's [reserves](#), resulting in a loss of the contract's assets.

Suggestion I Add a check to ensure that the margin position is not locked before operating in the function [internal_margin_decrease_collateral](#).

2.1.3 Incorrect enumeration of tokens requiring price feeds

Severity High

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description The function [margin_involved_tokens\(\)](#) is intended to enumerate the tokens that require price feeds for a given [MarginActions](#). It adds the [token_c_id](#) and [token_d_id](#) of the relevant margin positions. However, according to the design, the [token_c_id](#) may be the same as the [token_d_id](#). In this case, the [token_p_id](#), whose price is needed for calculating slippage protection, is not included in this enumeration.

```
107 pub fn margin_involved_tokens(&self, account: &MarginAccount, actions: &Vec<MarginAction>) ->
    Vec<TokenId> {
108     let mut tokens = HashSet::new();
109     actions.iter().for_each(|action|{
110         let pos_id = match action {
111             MarginAction::DecreaseCollateral { pos_id, amount: _ } => {
112                 Some(pos_id)
113             }
114             MarginAction::OpenPosition { token_c_id, token_c_amount: _, token_d_id,
115                 token_d_amount: _, token_p_id: _, min_token_p_amount: _, swap_indication: _ }
116                 => {
117                 tokens.insert(token_c_id.clone());
118                 tokens.insert(token_d_id.clone());
119                 None
120             },
121             MarginAction::DecreaseMTPosition { pos_id, token_p_amount: _, min_token_d_amount: _
122                 , swap_indication: _ }=> {
123                 Some(pos_id)
124             }
125             MarginAction::CloseMTPosition { pos_id, token_p_amount: _, min_token_d_amount: _,
126                 swap_indication: _ } => {
127                 Some(pos_id)
128             }
129             MarginAction::LiquidateMTPosition { pos_owner_id, pos_id, token_p_amount: _,
130                 min_token_d_amount: _, swap_indication: _ } => {
131                 let pos_owner_account = self.internal_get_margin_account(pos_owner_id).expect("
132                     Margin account not exist");
133                 let mt = pos_owner_account.margin_positions.get(pos_id).expect("Position not
134                     exist");
135                 tokens.insert(mt.token_c_id.clone());
136                 tokens.insert(mt.token_d_id.clone());
137                 None
138             }
139             MarginAction::ForceCloseMTPosition { pos_owner_id, pos_id, token_p_amount: _,
140                 min_token_d_amount: _, swap_indication: _ } => {
141                 let pos_owner_account = self.internal_get_margin_account(pos_owner_id).expect("
142                     Margin account not exist");
143                 let mt = pos_owner_account.margin_positions.get(pos_id).expect("Position not
144                     exist");
145                 tokens.insert(mt.token_c_id.clone());
146                 tokens.insert(mt.token_d_id.clone());
147                 None
148             }
149             _ => None
150         };
151         if let Some(pos_id) = pos_id {
152             let mt = account.margin_positions.get(pos_id).expect("Position not exist");
153             tokens.insert(mt.token_c_id.clone());
154             tokens.insert(mt.token_d_id.clone());
155         }
156     });
157     tokens.into_iter().collect()
```

138 }

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

Impact The callback function `callback_margin_execute_with_pyth()` that is invoked after `internal_margin_execute_with_pyth()` may fail due to missing price feeds for certain tokens required by the margin action.

Suggestion I Modify the function `margin_involved_tokens()` to replace `token_c_id` with `token_p_id`, so that it adds the position token (`token_p_id`) and debt token (`token_d_id`) to the list of tokens requiring price feeds.

2.1.4 Unreasonable pos_id design

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description The `pos_id` is generated by concatenating the `account_id` with the `env::block_timestamp()`. This means that within the same block, all generated `pos_ids` with the same `margin` account will be identical.

```

174 pub(crate) fn internal_margin_open_position(
175     &mut self,
176     ts: Timestamp,
177     account: &mut MarginAccount,
178     token_c_id: &AccountId,
179     token_c_amount: Balance,
180     token_d_id: &AccountId,
181     token_d_amount: Balance,
182     token_p_id: &AccountId,
183     min_token_p_amount: Balance,
184     swap_indication: &SwapIndication,
185     prices: &Prices,
186 ) -> EventDataMarginOpen {
187     let pos_id = format!("{}", account.account_id.clone(), ts);
188     assert!(
189         !account.margin_positions.contains_key(&pos_id),
190         "Margin position already exist"
191     );
192
193     let asset_c = self.internal_unwrap_asset(token_c_id);
194     let asset_p = self.internal_unwrap_asset(token_p_id);
195     let mut asset_d = self.internal_unwrap_asset(token_d_id);
196     let margin_config = self.internal_margin_config();
197
198     // check legitimacy: assets legal; swap_indication matches;
199     margin_config.check_pair(&token_d_id, &token_p_id, &token_c_id);
200     let mut swap_detail = self.parse_swap_indication(swap_indication);
201     let ft_d_amount = token_d_amount / 10u128.pow(asset_d.config.extra_decimals as u32);
202     assert!(
203         swap_detail.verify_token_in(token_d_id, ft_d_amount),

```

```
204         "token_in check failed"
205     );
206     let ft_p_amount =
207         min_token_p_amount / 10u128.pow(asset_p.config.extra_decimals as u32);
208     assert!(
209         swap_detail.verify_token_out(token_p_id, ft_p_amount),
210         "token_out check failed"
211     );
212
213     // check safty:
214     // min_position_amount reasonable
215     assert!(
216         is_min_amount_out_reasonable(
217             token_d_amount,
218             &asset_d,
219             prices.get_unwrap(&token_d_id),
220             &asset_p,
221             prices.get_unwrap(&token_p_id),
222             min_token_p_amount,
223             margin_config.max_slippage_rate,
224         ),
225         "min_position_amount is too low"
226     );
227     // margin_hf more than 1 + safty_buffer_rate(10%)
228     let mut mt = MarginTradingPosition::new(
229         ts,
230         token_c_id.clone(),
231         asset_c.supplied.amount_to_shares(token_c_amount, false),
232         token_d_id.clone(),
233         token_p_id.clone(),
234     );
235     mt.token_d_shares = asset_d.margin_debt.amount_to_shares(token_d_amount, true);
236     mt.token_p_amount = min_token_p_amount;
237     assert!(
238         !self.is_mt_liquidatable(&mt, prices, margin_config.min_safty_buffer),
239         "Debt is too much"
240     );
241     assert!(
242         !self.is_mt_forcecloseable(&mt, prices),
243         "Debt is too much"
244     );
245     // leverage rate less than max leverage rate
246     assert!(
247         self.get_mtp_lr(&mt, prices).unwrap()
248             <= BigDecimal::from(margin_config.max_leverage_rate as u32),
249         "Leverage rate is too high"
250     );
251
252     // passes all check, start to open
253     let event = EventDataMarginOpen {
254         account_id: account.account_id.clone(),
255         pos_id: pos_id.clone(),
256         token_c_id: token_c_id.clone(),
```

```

257         token_c_amount,
258         token_c_shares: mt.token_c_shares,
259         token_d_id: token_d_id.clone(),
260         token_d_amount,
261         token_p_id: token_p_id.clone(),
262         token_p_amount: min_token_p_amount,
263     };
264     account.withdraw_supply_shares(token_c_id, &mt.token_c_shares);
265     mt.token_d_shares.0 = 0;
266     mt.token_p_amount = 0;
267     asset_d.increase_margin_pending_debt(token_d_amount, margin_config.pending_debt_scale);
268     self.internal_set_asset(token_d_id, asset_d);
269     // TODO: may need to change to store in an unorderedmap in user Account
270     account.margin_positions.insert(pos_id.clone(), mt);
271
272     // step 4: call dex to trade and wait for callback
273     // organize swap action
274     let swap_ref = SwapReference {
275         account_id: account.account_id.clone(),
276         pos_id: pos_id.clone(),
277         amount_in: token_d_amount.into(),
278         op: format!("open"),
279         liquidator_id: None,
280     };
281     swap_detail.set_client_echo(&swap_ref.to_msg_string());
282     let swap_msg = swap_detail.to_msg_string();
283     ext_fungible_token::ext(token_d_id.clone())
284         .with_attached_deposit(1)
285         .with_static_gas(GAS_FOR_FT_TRANSFER_CALL)
286         .ft_transfer_call(
287             swap_indication.dex_id.clone(),
288             U128(ft_d_amount),
289             None,
290             swap_msg,
291         )
292         .then(
293             Self::ext(env::current_account_id())
294                 .with_static_gas(GAS_FOR_FT_TRANSFER_CALL_CALLBACK)
295                 .callback_dex_trade(
296                     account.account_id.clone(),
297                     pos_id.clone(),
298                     token_d_amount.into(),
299                     U128(0),
300                     format!("open"),
301                 ),
302         );
303     event
304 }

```

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

Impact Users are unable to open multiple margin positions within the same block, as all the `pos_ids` generated in that block will be identical.

Suggestion I Use a more reasonable method for generating the `pos_id`, such as an auto-incrementing variable.

2.1.5 Lack of reasonable configuration check

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description The newly introduced `MarginConfig` contains many global parameters for managing margin trading. These global parameters are used to compare against the user-provided parameters, ensuring that the user does not input values that are too large or too small. However, some of these parameters lack the necessary range checks. As shown in the table below:

Parameter	Reasonable Range
<code>max_leverage_rate</code>	bigger than 1
<code>pending_debt_scale</code>	(0, MAX_RATIO)
<code>max_slippage_rate</code>	(0, MAX_RATIO)
<code>min_safty_buffer</code>	(0, MAX_RATIO)
<code>margin_debt_discount_rate</code>	(0, MAX_RATIO)
<code>open_position_fee_rate</code>	(0, MAX_RATIO)

In addition, the variable “safty” is a typo of “safety”.

```

69  #[payable]
70  pub fn update_max_leverage_rate(&mut self, max_leverage_rate: u8) {
71      assert_one_yocto();
72      self.assert_owner();
73      let mut mc = self.internal_margin_config();
74      mc.max_leverage_rate = max_leverage_rate;
75      self.margin_config.set(&mc);
76  }
77
78  #[payable]
79  pub fn update_pending_debt_scale(&mut self, pending_debt_scale: u32) {
80      assert_one_yocto();
81      self.assert_owner();
82      let mut mc = self.internal_margin_config();
83      mc.pending_debt_scale = pending_debt_scale;
84      self.margin_config.set(&mc);
85  }
86
87  #[payable]
88  pub fn update_max_slippage_rate(&mut self, max_slippage_rate: u32) {
89      assert_one_yocto();
90      self.assert_owner();
91      let mut mc = self.internal_margin_config();
92      mc.max_slippage_rate = max_slippage_rate;
93      self.margin_config.set(&mc);
94  }
95

```

```
96  #[payable]
97  pub fn update_min_safety_buffer(&mut self, min_safety_buffer: u32) {
98      assert_one_yocto();
99      self.assert_owner();
100     let mut mc = self.internal_margin_config();
101     mc.min_safety_buffer = min_safety_buffer;
102     self.margin_config.set(&mc);
103 }
104
105  #[payable]
106  pub fn update_margin_debt_discount_rate(&mut self, margin_debt_discount_rate: u32) {
107      assert_one_yocto();
108      self.assert_owner();
109      let mut mc = self.internal_margin_config();
110      mc.margin_debt_discount_rate = margin_debt_discount_rate;
111      self.margin_config.set(&mc);
112 }
113
114  #[payable]
115  pub fn update_open_position_fee_rate(&mut self, open_position_fee_rate: u32) {
116      assert_one_yocto();
117      self.assert_owner();
118      let mut mc = self.internal_margin_config();
119      mc.open_position_fee_rate = open_position_fee_rate;
120      self.margin_config.set(&mc);
121 }
```

Listing 2.5: burrowland/contracts/contract/src/margin_config.rs

Impact Unreasonable configuration values due to missing range checks can cause the contract to not work as intended.

Suggestion I Add corresponding checks according to the table.

2.1.6 Potential panic during handling message SwapReference

Severity High

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description When handling the [SwapReference](#) message, the transaction should not fail. Otherwise, the user would lose the assets they obtained from the [Ref V1](#) or [Ref V2](#) contract, causing an inconsistency in the contract state. The corresponding margin position would also remain locked and unable to be unlocked. However, during this process, the function [internal_set_margin_account\(\)](#) needs to be invoked to write back the account states of the [liquidator](#) and [account](#), which includes storage checks that may cause a panic by insufficient storage fees.

```
88  TokenReceiverMsg::SwapReference { swap_ref } => {
89      let mut account = self.internal_unwrap_margin_account(&swap_ref.account_id);
90      if swap_ref.op == "open" {
91          self.on_open_trade_return(&mut account, amount, &swap_ref);
```



```

92     } else if swap_ref.op == "decrease"
93     || swap_ref.op == "close"
94     || swap_ref.op == "liquidate"
95     || swap_ref.op == "forceclose"
96     {
97         let event = self.on_decrease_trade_return(&mut account, amount, &swap_ref);
98         events::emit::margin_decrease_succeeded(&swap_ref.op, event);
99     }
100     self.internal_set_margin_account(&swap_ref.account_id, account);
101     return PromiseOrValue::Value(U128(0));
102 }

```

Listing 2.6: burrowland/contracts/contract/src/fungible_token.rs

```

366 pub(crate) fn on_decrease_trade_return(
367     &mut self,
368     account: &mut MarginAccount,
369     amount: Balance,
370     sr: &SwapReference,
371 ) -> EventDataMarginDecreaseResult {
372     let mut mt = account.margin_positions.get(&sr.pos_id).unwrap().clone();
373     let mut asset_debt = self.internal_unwrap_asset(&mt.token_d_id);
374     let mut asset_position = self.internal_unwrap_asset(&mt.token_p_id);
375     let (mut benefit_m_shares, mut benefit_d_shares, mut benefit_p_shares) =
376         (0_u128, 0_u128, 0_u128);
377
378     // figure out actual repay amount and shares
379     // figure out how many debt_cap been repaid, and charge corresponding holding-position fee
380     // from repayment.
381     let debt_amount = asset_debt
382         .margin_debt
383         .shares_to_amount(mt.token_d_shares, true);
384     let hp_fee = u128_ratio(
385         mt.debt_cap,
386         asset_debt.unit_acc_hp_interest - mt.uahpi_at_open,
387         UNIT,
388     );
389     let repay_cap = u128_ratio(mt.debt_cap, amount, debt_amount + hp_fee);
390
391     let (repay_amount, repay_shares, left_amount, repay_hp_fee) = if repay_cap >= mt.debt_cap {
392         (debt_amount, mt.token_d_shares, amount - debt_amount - hp_fee, hp_fee)
393     } else {
394         let repay_hp_fee = u128_ratio(hp_fee, repay_cap, mt.debt_cap);
395         (
396             amount - repay_hp_fee,
397             asset_debt
398                 .margin_debt
399                 .amount_to_shares(amount - repay_hp_fee, false),
400             0,
401             repay_hp_fee,
402         )
403     };
404     asset_debt.margin_debt.withdraw(repay_shares, repay_amount);

```

```
404     mt.token_d_shares.0 -= repay_shares.0;
405     mt.debt_cap = if repay_cap >= mt.debt_cap {
406         0
407     } else {
408         mt.debt_cap - repay_cap
409     };
410     // distribute hp_fee
411     asset_debt.prot_fee += repay_hp_fee;
412
413     // handle possible leftover debt asset, put them into user's supply
414     if left_amount > 0 {
415         let supply_shares = asset_debt.supplied.amount_to_shares(left_amount, false);
416         if supply_shares.0 > 0 {
417             asset_debt.supplied.deposit(supply_shares, left_amount);
418             benefit_d_shares = supply_shares.0;
419         }
420     }
421
422     if sr.op != "decrease" {
423         // try to repay remaining debt from margin
424         if mt.token_d_shares.0 > 0 && mt.token_d_id == mt.token_c_id {
425             let remain_debt_balance = asset_debt
426                 .margin_debt
427                 .shares_to_amount(mt.token_d_shares, true);
428             let margin_shares_to_repay = asset_debt
429                 .supplied
430                 .amount_to_shares(remain_debt_balance, true);
431             let (repay_debt_share, used_supply_share, repay_amount) =
432                 if margin_shares_to_repay <= mt.token_c_shares {
433                     (mt.token_d_shares, margin_shares_to_repay, remain_debt_balance)
434                 } else {
435                     // use all margin balance to repay
436                     let margin_balance = asset_debt
437                         .supplied
438                         .shares_to_amount(mt.token_c_shares, false);
439                     let repay_debt_shares = asset_debt
440                         .margin_debt
441                         .amount_to_shares(margin_balance, false);
442                     (repay_debt_shares, mt.token_c_shares, margin_balance)
443                 };
444             asset_debt
445                 .supplied
446                 .withdraw(used_supply_share, repay_amount);
447             asset_debt
448                 .margin_debt
449                 .withdraw(repay_debt_share, repay_amount);
450             mt.token_d_shares.0 -= repay_debt_share.0;
451             mt.token_c_shares.0 -= used_supply_share.0;
452         }
453     }
454
455     if sr.op == "forceclose" {
456         // try to use protocol reserve to repay remaining debt
```

```
457     if mt.token_d_shares.0 > 0 {
458         let remain_debt_balance = asset_debt
459             .margin_debt
460             .shares_to_amount(mt.token_d_shares, true);
461         if asset_debt.reserved > remain_debt_balance {
462             asset_debt.reserved -= remain_debt_balance;
463             asset_debt
464                 .margin_debt
465                 .withdraw(mt.token_d_shares, remain_debt_balance);
466             mt.token_d_shares.0 = 0;
467         }
468     }
469 }
470
471 mt.is_locking = false;
472 account
473     .margin_positions
474     .insert(sr.pos_id.clone(), mt.clone());
475
476
477 let event = EventDataMarginDecreaseResult {
478     account_id: account.account_id.clone(),
479     pos_id: sr.pos_id.clone(),
480     liquidator_id: sr.liquidator_id.clone(),
481     token_c_id: mt.token_c_id.clone(),
482     token_c_shares: mt.token_c_shares,
483     token_d_id: mt.token_d_id.clone(),
484     token_d_shares: mt.token_d_shares,
485     token_p_id: mt.token_p_id.clone(),
486     token_p_amount: mt.token_p_amount,
487     holding_fee: repay_hp_fee,
488 };
489
490 // try to settle this position
491 if mt.token_d_shares.0 == 0 {
492     // close this position and remaining asset goes back to user's inner account
493     // TODO: change to directly send assets back to user
494     if mt.token_c_shares.0 > 0 {
495         benefit_m_shares = mt.token_c_shares.0;
496     }
497     if mt.token_p_amount > 0 {
498         let position_shares = asset_position
499             .supplied
500             .amount_to_shares(mt.token_p_amount, false);
501         asset_position
502             .supplied
503             .deposit(position_shares, mt.token_p_amount);
504         asset_position.margin_position -= mt.token_p_amount;
505         benefit_p_shares = position_shares.0;
506     }
507     account.margin_positions.remove(&sr.pos_id);
508 } else {
509     if sr.op != "decrease" {
```

```
510         env::log_str(&format!(
511             "{} failed due to insufficient fund, user {}, pos_id {}",
512             sr.op.clone(),
513             account.account_id.clone(),
514             sr.pos_id.clone()
515         ));
516     }
517 }
518
519 // distribute benefits
520 if benefit_d_shares > 0 || benefit_m_shares > 0 || benefit_p_shares > 0 {
521     if sr.op == "liquidate" || sr.op == "forceclose" {
522         let mut liquidator_account =
523             if let Some(ref liquidator_account_id) = sr.liquidator_id {
524                 if let Some(x) = self.internal_get_margin_account(&liquidator_account_id) {
525                     x
526                 } else {
527                     self.internal_unwrap_margin_account(&self.internal_config().owner_id)
528                 }
529             } else {
530                 self.internal_unwrap_margin_account(&self.internal_config().owner_id)
531             };
532         if benefit_d_shares > 0 {
533             liquidator_account
534                 .deposit_supply_shares(&mt.token_d_id, &U128(benefit_d_shares));
535         }
536         if benefit_m_shares > 0 {
537             liquidator_account
538                 .deposit_supply_shares(&mt.token_c_id, &U128(benefit_m_shares));
539         }
540         if benefit_p_shares > 0 {
541             liquidator_account
542                 .deposit_supply_shares(&mt.token_p_id, &U128(benefit_p_shares));
543         }
544         self.internal_set_margin_account(
545             &liquidator_account.account_id.clone(),
546             liquidator_account,
547         );
548     } else {
549         if benefit_d_shares > 0 {
550             account.deposit_supply_shares(&mt.token_d_id, &U128(benefit_d_shares));
551         }
552         if benefit_m_shares > 0 {
553             account.deposit_supply_shares(&mt.token_c_id, &U128(benefit_m_shares));
554         }
555         if benefit_p_shares > 0 {
556             account.deposit_supply_shares(&mt.token_p_id, &U128(benefit_p_shares));
557         }
558     }
559 }
560
561 self.internal_set_asset(&mt.token_d_id, asset_debt);
562 self.internal_set_asset(&mt.token_p_id, asset_position);
```

```

563
564     event
565 }

```

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

Impact Users may lose the assets they obtained from the [Ref V1](#) or [Ref V2](#) contract, causing an inconsistency in the contract state. The corresponding margin position would also remain locked and unable to be unlocked.

Suggestion I Revise the corresponding logic.

2.1.7 Unreasonable check of reserves

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description When executing a force close operation in the function `on_decrease_trade_return()`, any remaining `debt` needs to be covered by the contract's `reserves`. However, when comparing the amount between the contract's `reserves` and the remaining `debt` amount, the check requires the `reserves` to be strictly greater than the remaining `debt`.

```

366 pub(crate) fn on_decrease_trade_return(
367     &mut self,
368     account: &mut MarginAccount,
369     amount: Balance,
370     sr: &SwapReference,
371 ) -> EventDataMarginDecreaseResult {
372     let mut mt = account.margin_positions.get(&sr.pos_id).unwrap().clone();
373     let mut asset_debt = self.internal_unwrap_asset(&mt.token_d_id);
374     let mut asset_position = self.internal_unwrap_asset(&mt.token_p_id);
375     let (mut benefit_m_shares, mut benefit_d_shares, mut benefit_p_shares) =
376         (0_u128, 0_u128, 0_u128);
377
378     // figure out actual repay amount and shares
379     // figure out how many debt_cap been repaid, and charge corresponding holding-position fee
380     // from repayment.
381     let debt_amount = asset_debt
382         .margin_debt
383         .shares_to_amount(mt.token_d_shares, true);
384     let hp_fee = u128_ratio(
385         mt.debt_cap,
386         asset_debt.unit_acc_hp_interest - mt.uahpi_at_open,
387         UNIT,
388     );
389     let repay_cap = u128_ratio(mt.debt_cap, amount, debt_amount + hp_fee);
390     let (repay_amount, repay_shares, left_amount, repay_hp_fee) = if repay_cap >= mt.debt_cap {
391         (debt_amount, mt.token_d_shares, amount - debt_amount - hp_fee, hp_fee)
392     } else {

```

```
393     let repay_hp_fee = u128_ratio(hp_fee, repay_cap, mt.debt_cap);
394     (
395         amount - repay_hp_fee,
396         asset_debt
397             .margin_debt
398             .amount_to_shares(amount - repay_hp_fee, false),
399         0,
400         repay_hp_fee,
401     )
402 };
403 asset_debt.margin_debt.withdraw(repay_shares, repay_amount);
404 mt.token_d_shares.0 -= repay_shares.0;
405 mt.debt_cap = if repay_cap >= mt.debt_cap {
406     0
407 } else {
408     mt.debt_cap - repay_cap
409 };
410 // distribute hp_fee
411 asset_debt.prot_fee += repay_hp_fee;
412
413 // handle possible leftover debt asset, put them into user's supply
414 if left_amount > 0 {
415     let supply_shares = asset_debt.supplied.amount_to_shares(left_amount, false);
416     if supply_shares.0 > 0 {
417         asset_debt.supplied.deposit(supply_shares, left_amount);
418         benefit_d_shares = supply_shares.0;
419     }
420 }
421
422 if sr.op != "decrease" {
423     // try to repay remaining debt from margin
424     if mt.token_d_shares.0 > 0 && mt.token_d_id == mt.token_c_id {
425         let remain_debt_balance = asset_debt
426             .margin_debt
427             .shares_to_amount(mt.token_d_shares, true);
428         let margin_shares_to_repay = asset_debt
429             .supplied
430             .amount_to_shares(remain_debt_balance, true);
431         let (repay_debt_share, used_supply_share, repay_amount) =
432             if margin_shares_to_repay <= mt.token_c_shares {
433                 (mt.token_d_shares, margin_shares_to_repay, remain_debt_balance)
434             } else {
435                 // use all margin balance to repay
436                 let margin_balance = asset_debt
437                     .supplied
438                     .shares_to_amount(mt.token_c_shares, false);
439                 let repay_debt_shares = asset_debt
440                     .margin_debt
441                     .amount_to_shares(margin_balance, false);
442                 (repay_debt_shares, mt.token_c_shares, margin_balance)
443             };
444         asset_debt
445             .supplied
```

```
446         .withdraw(used_supply_share, repay_amount);
447     asset_debt
448         .margin_debt
449         .withdraw(repay_debt_share, repay_amount);
450     mt.token_d_shares.0 -= repay_debt_share.0;
451     mt.token_c_shares.0 -= used_supply_share.0;
452 }
453 }
454
455 if sr.op == "forceclose" {
456     // try to use protocol reserve to repay remaining debt
457     if mt.token_d_shares.0 > 0 {
458         let remain_debt_balance = asset_debt
459             .margin_debt
460             .shares_to_amount(mt.token_d_shares, true);
461         if asset_debt.reserved > remain_debt_balance {
462             asset_debt.reserved -= remain_debt_balance;
463             asset_debt
464                 .margin_debt
465                 .withdraw(mt.token_d_shares, remain_debt_balance);
466             mt.token_d_shares.0 = 0;
467         }
468     }
469 }
470
471 mt.is_locking = false;
472 account
473     .margin_positions
474     .insert(sr.pos_id.clone(), mt.clone());
475
476
477 let event = EventDataMarginDecreaseResult {
478     account_id: account.account_id.clone(),
479     pos_id: sr.pos_id.clone(),
480     liquidator_id: sr.liquidator_id.clone(),
481     token_c_id: mt.token_c_id.clone(),
482     token_c_shares: mt.token_c_shares,
483     token_d_id: mt.token_d_id.clone(),
484     token_d_shares: mt.token_d_shares,
485     token_p_id: mt.token_p_id.clone(),
486     token_p_amount: mt.token_p_amount,
487     holding_fee: repay_hp_fee,
488 };
489
490 // try to settle this position
491 if mt.token_d_shares.0 == 0 {
492     // close this position and remaining asset goes back to user's inner account
493     // TODO: change to directly send assets back to user
494     if mt.token_c_shares.0 > 0 {
495         benefit_m_shares = mt.token_c_shares.0;
496     }
497     if mt.token_p_amount > 0 {
498         let position_shares = asset_position
```

```
499         .supplied
500         .amount_to_shares(mt.token_p_amount, false);
501     asset_position
502         .supplied
503         .deposit(position_shares, mt.token_p_amount);
504     asset_position.margin_position -= mt.token_p_amount;
505     benefit_p_shares = position_shares.0;
506 }
507 account.margin_positions.remove(&sr.pos_id);
508 } else {
509     if sr.op != "decrease" {
510         env::log_str(&format!(
511             "{} failed due to insufficient fund, user {}, pos_id {}",
512             sr.op.clone(),
513             account.account_id.clone(),
514             sr.pos_id.clone()
515         ));
516     }
517 }
518
519 // distribute benefits
520 if benefit_d_shares > 0 || benefit_m_shares > 0 || benefit_p_shares > 0 {
521     if sr.op == "liquidate" || sr.op == "forceclose" {
522         let mut liquidator_account =
523             if let Some(ref liquidator_account_id) = sr.liquidator_id {
524                 if let Some(x) = self.internal_get_margin_account(&liquidator_account_id) {
525                     x
526                 } else {
527                     self.internal_unwrap_margin_account(&self.internal_config().owner_id)
528                 }
529             } else {
530                 self.internal_unwrap_margin_account(&self.internal_config().owner_id)
531             };
532         if benefit_d_shares > 0 {
533             liquidator_account
534                 .deposit_supply_shares(&mt.token_d_id, &U128(benefit_d_shares));
535         }
536         if benefit_m_shares > 0 {
537             liquidator_account
538                 .deposit_supply_shares(&mt.token_c_id, &U128(benefit_m_shares));
539         }
540         if benefit_p_shares > 0 {
541             liquidator_account
542                 .deposit_supply_shares(&mt.token_p_id, &U128(benefit_p_shares));
543         }
544         self.internal_set_margin_account(
545             &liquidator_account.account_id.clone(),
546             liquidator_account,
547         );
548     } else {
549         if benefit_d_shares > 0 {
550             account.deposit_supply_shares(&mt.token_d_id, &U128(benefit_d_shares));
551         }
```



```
552         if benefit_m_shares > 0 {
553             account.deposit_supply_shares(&mt.token_c_id, &U128(benefit_m_shares));
554         }
555         if benefit_p_shares > 0 {
556             account.deposit_supply_shares(&mt.token_p_id, &U128(benefit_p_shares));
557         }
558     }
559 }
560
561 self.internal_set_asset(&mt.token_d_id, asset_debt);
562 self.internal_set_asset(&mt.token_p_id, asset_position);
563
564 event
565 }
```

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

Impact Margin positions that can be forced closed may fail to be closed.

Suggestion I Revise the check to be greater or equal.

2.1.8 Potential sandwich attack in force close position token swap

Severity Low

Status Confirmed

Introduced by [Version 1](#)

Description The force closing margin position operation swaps the liquidated user's [position](#) token into the [debt](#) token. Despite having slippage protection, there is still some room for a sandwich attack during this swap.

This issue is similar to issue-2, where both involve a potential sandwich attack on a token swap. However, issue-2 can be reliably triggered by an attacker, whereas under normal circumstances, the probability of a margin position reaching a force-closable state to trigger this swap is relatively low.

```
307 pub(crate) fn process_decrease_margin_position(
308     &mut self,
309     account: &mut MarginAccount,
310     pos_id: &String,
311     token_p_amount: Balance,
312     min_token_d_amount: Balance,
313     swap_indication: &SwapIndication,
314     prices: &Prices,
315     op: String,
316     liquidator_id: Option<AccountId>,
317 ) -> EventDataMarginDecrease {
318     let mut mt = account
319         .margin_positions
320         .get_mut(pos_id)
321         .expect("Position not exist");
322     assert!(
323         !mt.is_locking,
```

```
324         "Position is currently waiting for a trading result."
325     );
326     let pre_token_p_amount = mt.token_p_amount;
327     let mut asset_p = self.internal_unwrap_asset(&mt.token_p_id);
328     let asset_d = self.internal_unwrap_asset(&mt.token_d_id);
329     let margin_config = self.internal_margin_config();
330
331     // check swap_indication
332     let mut swap_detail = self.parse_swap_indication(swap_indication);
333     let ft_p_amount =
334         token_p_amount / 10u128.pow(asset_p.config.extra_decimals as u32);
335     assert!(
336         swap_detail.verify_token_in(&mt.token_p_id, ft_p_amount),
337         "token_in check failed"
338     );
339     let ft_d_amount = min_token_d_amount / 10u128.pow(asset_d.config.extra_decimals as u32);
340     assert!(
341         swap_detail.verify_token_out(&mt.token_d_id, ft_d_amount),
342         "token_out check failed"
343     );
344
345     // min_debt_amount reasonable
346     assert!(
347         is_min_amount_out_reasonable(
348             token_p_amount,
349             &asset_p,
350             prices.get_unwrap(&mt.token_p_id),
351             &asset_d,
352             prices.get_unwrap(&mt.token_d_id),
353             min_token_d_amount,
354             margin_config.max_slippage_rate,
355         ),
356         "min_debt_amount is too low"
357     );
358
359     if op == "close" || op == "liquidate" {
360         // ensure all debt would be repaid
361         // and take holding-position fee into account
362         let total_debt_amount = asset_d
363             .margin_debt
364             .shares_to_amount(mt.token_d_shares, true);
365         let hp_fee = u128_ratio(
366             mt.debt_cap,
367             asset_d.unit_acc_hp_interest - mt.uahpi_at_open,
368             UNIT,
369         );
370         if min_token_d_amount < total_debt_amount + hp_fee {
371             assert_eq!(
372                 mt.token_c_id, mt.token_d_id,
373                 "Can NOT trade under total debt when margin and debt asset are not the same"
374             );
375             let gap_shares = asset_d
376                 .supplied
```

```
377         .amount_to_shares(total_debt_amount + hp_fee - min_token_d_amount, true);
378         assert!(
379             mt.token_c_shares.0 > gap_shares.0,
380             "Not all debt could be repaid"
381         );
382     }
383 }
384
385 if op == "liquidate" {
386     assert!(
387         self.is_mt_liquidatable(&mt, prices, margin_config.min_safety_buffer),
388         "Margin position is not liquidatable"
389     );
390 } else if op == "forceclose" {
391     assert!(
392         self.is_mt_forcecloseable(&mt, prices),
393         "Margin position is not forceclose-able"
394     );
395 }
396
397 // ensure enough position token to trade
398 if token_p_amount > mt.token_p_amount {
399     // try to add some of margin asset into trading
400     assert_eq!(
401         mt.token_c_id, mt.token_p_id,
402         "Not enough position asset balance"
403     );
404     let gap_shares = asset_p
405         .supplied
406         .amount_to_shares(token_p_amount - mt.token_p_amount, true);
407     mt.token_c_shares
408         .0
409         .checked_sub(gap_shares.0)
410         .expect("Not enough position asset balance");
411     asset_p
412         .supplied
413         .withdraw(gap_shares, token_p_amount - mt.token_p_amount);
414     asset_p.margin_position -= mt.token_p_amount;
415     mt.token_p_amount = 0;
416 } else {
417     asset_p.margin_position -= token_p_amount;
418     mt.token_p_amount -= token_p_amount;
419 }
420
421 // prepare to close
422 mt.is_locking = true;
423 self.internal_set_asset(&mt.token_p_id, asset_p);
424 // TODO: mt may be needed to change to store in an unorderedmap in user Account
425
426 let event = EventDataMarginDecrease {
427     account_id: account.account_id.clone(),
428     pos_id: pos_id.clone(),
429     liquidator_id: liquidator_id.clone(),
```

```
430         token_p_id: mt.token_p_id.clone(),
431         token_p_amount,
432         token_d_id: mt.token_d_id.clone(),
433         token_d_amount: min_token_d_amount,
434     };
435
436     // step 3: call dex to trade and wait for callback
437     // organize swap action
438     let swap_ref = SwapReference {
439         account_id: account.account_id.clone(),
440         pos_id: pos_id.clone(),
441         amount_in: token_p_amount.into(),
442         op,
443         liquidator_id,
444     };
445     swap_detail.set_client_echo(&swap_ref.to_msg_string());
446     let swap_msg = swap_detail.to_msg_string();
447     ext_fungible_token::ext(mt.token_p_id.clone())
448         .with_attached_deposit(1)
449         .with_static_gas(GAS_FOR_FT_TRANSFER_CALL)
450         .ft_transfer_call(
451             swap_indication.dex_id.clone(),
452             U128(ft_p_amount),
453             None,
454             swap_msg,
455         )
456         .then(
457             Self::ext(env::current_account_id())
458                 .with_static_gas(GAS_FOR_FT_TRANSFER_CALL_CALLBACK)
459                 .callback_dex_trade(
460                     account.account_id.clone(),
461                     pos_id.clone(),
462                     token_p_amount.into(),
463                     pre_token_p_amount.into(),
464                     format!("decrease"),
465                 ),
466         );
467     event
468 }
```

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

Impact An attacker can set the `min_amount_d_out` to be as small as possible within the allowed slippage protection range. This enables them to conduct a sandwich attack, forcing the contract to use more of its reserves to cover the liquidation.

Suggestion I The value of the parameter `max_slippage_rate` for slippage protection should be set more strictly (smaller) when it comes to force closing, distinguishing it from the values used for opening or other operations.

Feedback from the Project Accept this situation and try to ensure success when the force closing is initiated.

2.1.9 Unreasonable leverage rate computation

Severity Medium

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description The hold position fee is not included in leverage rate computation in the function `get_mtp_lr`.

```
160 pub(crate) fn get_mtp_lr(
161     &self,
162     mt: &MarginTradingPosition,
163     prices: &Prices,
164 ) -> Option<BigDecimal> {
165     if mt.token_c_shares.0 == 0 || mt.token_d_shares.0 == 0 {
166         None
167     } else {
168         Some(self.get_mtp_debt_value(&mt, prices) / self.get_mtp_collateral_value(&mt, prices))
169     }
170 }
```

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

Impact The computation of the leverage rate is inaccurate.

Suggestion I Add the hold position fee into the computation of the leverage rate.

2.2 Additional Recommendation

2.2.1 Automatically construct swap indication from the token information

Status Confirmed

Introduced by [Version 1](#)

Description In functions `internal_margin_open_position()` and `process_decrease_margin_position()`, a consistency check is required between the token information and the `swap_indication` provided by the user within functions `verify_token_in()` and `verify_token_out()`. However, both of them are specified by the user.

```
174 pub(crate) fn internal_margin_open_position(
175     &mut self,
176     ts: Timestamp,
177     account: &mut MarginAccount,
178     token_c_id: &AccountId,
179     token_c_amount: Balance,
180     token_d_id: &AccountId,
181     token_d_amount: Balance,
182     token_p_id: &AccountId,
183     min_token_p_amount: Balance,
184     swap_indication: &SwapIndication,
185     prices: &Prices,
186 ) -> EventDataMarginOpen {
```

```
187     let pos_id = format!("{}", ts);
188     assert!(
189         !account.margin_positions.contains_key(&pos_id),
190         "Margin position already exist"
191     );
192
193     let asset_c = self.internal_unwrap_asset(token_c_id);
194     let asset_p = self.internal_unwrap_asset(token_p_id);
195     let mut asset_d = self.internal_unwrap_asset(token_d_id);
196     let margin_config = self.internal_margin_config();
197
198     // check legitimacy: assets legal; swap_indication matches;
199     margin_config.check_pair(&token_d_id, &token_p_id, &token_c_id);
200     let mut swap_detail = self.parse_swap_indication(swap_indication);
201     let ft_d_amount = token_d_amount / 10u128.pow(asset_d.config.extra_decimals as u32);
202     assert!(
203         swap_detail.verify_token_in(token_d_id, ft_d_amount),
204         "token_in check failed"
205     );
206     let ft_p_amount =
207         min_token_p_amount / 10u128.pow(asset_p.config.extra_decimals as u32);
208     assert!(
209         swap_detail.verify_token_out(token_p_id, ft_p_amount),
210         "token_out check failed"
211     );
212
213     // check safty:
214     // min_position_amount reasonable
215     assert!(
216         is_min_amount_out_reasonable(
217             token_d_amount,
218             &asset_d,
219             prices.get_unwrap(&token_d_id),
220             &asset_p,
221             prices.get_unwrap(&token_p_id),
222             min_token_p_amount,
223             margin_config.max_slippage_rate,
224         ),
225         "min_position_amount is too low"
226     );
227     // margin_hf more than 1 + safty_buffer_rate(10%)
228     let mut mt = MarginTradingPosition::new(
229         ts,
230         token_c_id.clone(),
231         asset_c.supplied.amount_to_shares(token_c_amount, false),
232         token_d_id.clone(),
233         token_p_id.clone(),
234     );
235     mt.token_d_shares = asset_d.margin_debt.amount_to_shares(token_d_amount, true);
236     mt.token_p_amount = min_token_p_amount;
237     assert!(
238         !self.is_mt_liquidatable(&mt, prices, margin_config.min_safty_buffer),
239         "Debt is too much"
```

```
240     );
241     assert!(
242         !self.is_mt_forcecloseable(&mt, prices),
243         "Debt is too much"
244     );
245     // leverage rate less than max leverage rate
246     assert!(
247         self.get_mtp_lr(&mt, prices).unwrap()
248         <= BigDecimal::from(margin_config.max_leverage_rate as u32),
249         "Leverage rate is too high"
250     );
251
252     // passes all check, start to open
253     let event = EventDataMarginOpen {
254         account_id: account.account_id.clone(),
255         pos_id: pos_id.clone(),
256         token_c_id: token_c_id.clone(),
257         token_c_amount,
258         token_c_shares: mt.token_c_shares,
259         token_d_id: token_d_id.clone(),
260         token_d_amount,
261         token_p_id: token_p_id.clone(),
262         token_p_amount: min_token_p_amount,
263     };
264     account.withdraw_supply_shares(token_c_id, &mt.token_c_shares);
265     mt.token_d_shares.0 = 0;
266     mt.token_p_amount = 0;
267     asset_d.increase_margin_pending_debt(token_d_amount, margin_config.pending_debt_scale);
268     self.internal_set_asset(token_d_id, asset_d);
269     // TODO: may need to change to store in an unorderedmap in user Account
270     account.margin_positions.insert(pos_id.clone(), mt);
271
272     // step 4: call dex to trade and wait for callback
273     // organize swap action
274     let swap_ref = SwapReference {
275         account_id: account.account_id.clone(),
276         pos_id: pos_id.clone(),
277         amount_in: token_d_amount.into(),
278         op: format!("open"),
279         liquidator_id: None,
280     };
281     swap_detail.set_client_echo(&swap_ref.to_msg_string());
282     let swap_msg = swap_detail.to_msg_string();
283     ext_fungible_token::ext(token_d_id.clone())
284         .with_attached_deposit(1)
285         .with_static_gas(GAS_FOR_FT_TRANSFER_CALL)
286         .ft_transfer_call(
287             swap_indication.dex_id.clone(),
288             U128(ft_d_amount),
289             None,
290             swap_msg,
291         )
292         .then(
```

```
293         Self::ext(env::current_account_id())
294         .with_static_gas(GAS_FOR_FT_TRANSFER_CALL_CALLBACK)
295         .callback_dex_trade(
296             account.account_id.clone(),
297             pos_id.clone(),
298             token_d_amount.into(),
299             U128(0),
300             format!("open"),
301         ),
302     );
303     event
304 }
```

Listing 2.11: burrowland/contracts/contract/src/margin_position.rs

```
307 pub(crate) fn process_decrease_margin_position(
308     &mut self,
309     account: &mut MarginAccount,
310     pos_id: &String,
311     token_p_amount: Balance,
312     min_token_d_amount: Balance,
313     swap_indication: &SwapIndication,
314     prices: &Prices,
315     op: String,
316     liquidator_id: Option<AccountId>,
317 ) -> EventDataMarginDecrease {
318     let mut mt = account
319         .margin_positions
320         .get_mut(pos_id)
321         .expect("Position not exist");
322     assert!(
323         !mt.is_locking,
324         "Position is currently waiting for a trading result."
325     );
326     let pre_token_p_amount = mt.token_p_amount;
327     let mut asset_p = self.internal_unwrap_asset(&mt.token_p_id);
328     let asset_d = self.internal_unwrap_asset(&mt.token_d_id);
329     let margin_config = self.internal_margin_config();
330
331     // check swap_indication
332     let mut swap_detail = self.parse_swap_indication(swap_indication);
333     let ft_p_amount =
334         token_p_amount / 10u128.pow(asset_p.config.extra_decimals as u32);
335     assert!(
336         swap_detail.verify_token_in(&mt.token_p_id, ft_p_amount),
337         "token_in check failed"
338     );
339     let ft_d_amount = min_token_d_amount / 10u128.pow(asset_d.config.extra_decimals as u32);
340     assert!(
341         swap_detail.verify_token_out(&mt.token_d_id, ft_d_amount),
342         "token_out check failed"
343     );
344 }
```



```
345 // min_debt_amount reasonable
346 assert!(
347     is_min_amount_out_reasonable(
348         token_p_amount,
349         &asset_p,
350         prices.get_unwrap(&mt.token_p_id),
351         &asset_d,
352         prices.get_unwrap(&mt.token_d_id),
353         min_token_d_amount,
354         margin_config.max_slippage_rate,
355     ),
356     "min_debt_amount is too low"
357 );
358
359 if op == "close" || op == "liquidate" {
360     // ensure all debt would be repaid
361     // and take holding-position fee into account
362     let total_debt_amount = asset_d
363         .margin_debt
364         .shares_to_amount(mt.token_d_shares, true);
365     let hp_fee = u128_ratio(
366         mt.debt_cap,
367         asset_d.unit_acc_hp_interest - mt.uahpi_at_open,
368         UNIT,
369     );
370     if min_token_d_amount < total_debt_amount + hp_fee {
371         assert_eq!(
372             mt.token_c_id, mt.token_d_id,
373             "Can NOT trade under total debt when margin and debt asset are not the same"
374         );
375         let gap_shares = asset_d
376             .supplied
377             .amount_to_shares(total_debt_amount + hp_fee - min_token_d_amount, true);
378         assert!(
379             mt.token_c_shares.0 > gap_shares.0,
380             "Not all debt could be repaid"
381         );
382     }
383 }
384
385 if op == "liquidate" {
386     assert!(
387         self.is_mt_liquidatable(&mt, prices, margin_config.min_safety_buffer),
388         "Margin position is not liquidatable"
389     );
390 } else if op == "forceclose" {
391     assert!(
392         self.is_mt_forcecloseable(&mt, prices),
393         "Margin position is not forceclose-able"
394     );
395 }
396
397 // ensure enough position token to trade
```

```
398     if token_p_amount > mt.token_p_amount {
399         // try to add some of margin asset into trading
400         assert_eq!(
401             mt.token_c_id, mt.token_p_id,
402             "Not enough position asset balance"
403         );
404         let gap_shares = asset_p
405             .supplied
406             .amount_to_shares(token_p_amount - mt.token_p_amount, true);
407         mt.token_c_shares
408             .0
409             .checked_sub(gap_shares.0)
410             .expect("Not enough position asset balance");
411         asset_p
412             .supplied
413             .withdraw(gap_shares, token_p_amount - mt.token_p_amount);
414         asset_p.margin_position -= mt.token_p_amount;
415         mt.token_p_amount = 0;
416     } else {
417         asset_p.margin_position -= token_p_amount;
418         mt.token_p_amount -= token_p_amount;
419     }
420
421     // prepare to close
422     mt.is_locking = true;
423     self.internal_set_asset(&mt.token_p_id, asset_p);
424     // TODO: mt may be needed to change to store in an unorderedmap in user Account
425
426     let event = EventDataMarginDecrease {
427         account_id: account.account_id.clone(),
428         pos_id: pos_id.clone(),
429         liquidator_id: liquidator_id.clone(),
430         token_p_id: mt.token_p_id.clone(),
431         token_p_amount,
432         token_d_id: mt.token_d_id.clone(),
433         token_d_amount: min_token_d_amount,
434     };
435
436     // step 3: call dex to trade and wait for callback
437     // organize swap action
438     let swap_ref = SwapReference {
439         account_id: account.account_id.clone(),
440         pos_id: pos_id.clone(),
441         amount_in: token_p_amount.into(),
442         op,
443         liquidator_id,
444     };
445     swap_detail.set_client_echo(&swap_ref.to_msg_string());
446     let swap_msg = swap_detail.to_msg_string();
447     ext_fungible_token::ext(mt.token_p_id.clone())
448         .with_attached_deposit(1)
449         .with_static_gas(GAS_FOR_FT_TRANSFER_CALL)
450         .ft_transfer_call(
```

```

451         swap_indication.dex_id.clone(),
452         U128(ft_p_amount),
453         None,
454         swap_msg,
455     )
456     .then(
457         Self::ext(env::current_account_id())
458         .with_static_gas(GAS_FOR_FT_TRANSFER_CALL_CALLBACK)
459         .callback_dex_trade(
460             account.account_id.clone(),
461             pos_id.clone(),
462             token_p_amount.into(),
463             pre_token_p_amount.into(),
464             format!("decrease"),
465         ),
466     );
467     event
468 }

```

Listing 2.12: burrowland/contracts/contract/src/margin_position.rs

Suggestion I The contract can automatically construct the `swap_indication` based on the token information provided by the user.

Feedback from the Project The code will be optimized as appropriate in subsequent versions.

2.2.2 Use UnorderedMap for margin_positions instead of HashMap

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description The current implementation uses a `HashMap` for the `margin_positions` field in the struct `MarginAccount`. This means that every time the `MarginAccount` is deserialized, all the `margin_positions` have to be deserialized as well, resulting in a waste of gas.

```

3  #[derive(BorshSerialize, BorshDeserialize, Serialize, Clone)]
4  #[serde(crate = "near_sdk::serde")]
5  pub struct MarginAccount {
6      /// A copy of an account ID. Saves one storage_read when iterating on accounts.
7      pub account_id: AccountId,
8      /// A list of assets that are supplied by the account (but not used a collateral).
9      /// It's not returned for account pagination.
10     pub supplied: HashMap<TokenId, Shares>,
11     // margin trading related
12     pub margin_positions: HashMap<PosId, MarginTradingPosition>,
13     /// Tracks changes in storage usage by persistent collections in this account.
14     #[borsh_skip]
15     #[serde(skip)]
16     pub storage_tracker: StorageTracker,
17 }

```

Listing 2.13: burrowland/contracts/contract/src/margin_accounts.rs

Suggestion I Use `UnorderedMap` instead of `HashMap` for the `margin_positions` field in the struct `MarginAccount`.

2.2.3 Incorrect error message in `get_token_out()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `RefV2TokenReceiverMessage::get_token_out()`, `"RefV1TokenReceiverMessage"` is used as an error message, which is incorrect.

```
120 pub fn get_token_out(&self) -> (AccountId, Balance, Option<bool>) {
121     if let RefV2TokenReceiverMessage::Swap {
122         pool_ids: _,
123         output_token,
124         min_output_amount,
125         skip_unwrap_near,
126         client_echo: _,
127     } = self
128     {
129         (
130             output_token.clone(),
131             min_output_amount.0,
132             skip_unwrap_near.clone(),
133         )
134     } else {
135         env::panic_str("Invalid RefV1TokenReceiverMessage");
136     }
137 }
```

Listing 2.14: burrowland/contracts/contract/src/margin_trading.rs

Suggestion I Revise `"RefV1TokenReceiverMessage"` to `"RefV2TokenReceiverMessage"`.

