



BlockSec

Security Audit Report for Ref DCL Contract

Date: September 28th, 2023

Version: 4.0

Contact: contact@blocksec.com

Contents

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	4
2	Findings	5
2.1	Software Security	7
2.1.1	Incorrect Storage Check in batch_update_liquidity	7
2.1.2	Lack of Check on the withdraw_amount	8
2.2	DeFi Security	9
2.2.1	Non-withdrawable Fees Charged by the Protocol	9
2.2.2	Incorrect sqrt_price_loc_96 Calculation in y_swap_x_range_complete_desire()	10
2.2.3	Liquidity on Endpoint Processed Before the Limit Order	13
2.2.4	Potential Failure in the Callback Function	15
2.2.5	Improper Rounding Implementation	17
2.2.6	Improper Implementation of internal_mft_transfer()	20
2.2.7	Lack of Check on Remaining Mft when Updating Farming Contract	20
2.2.8	Inappropriate Limitation of mft_assets for Farming Contract	21
2.2.9	Lack of Pausable Feature	21
2.2.10	Liquidity on Endpoint Processed Before the Limit Order	33
2.2.11	Lack of Check on Repeated Liquidity in internal_check_remove_liquidity_infos	35
2.2.12	Unchecked Received Token in internal_add_order	36
2.3	Additional Recommendation	38
2.3.1	Potential Elastic Supply Token Problem	38
2.3.2	Potential Centralization Problem	39
2.3.3	Redundant Code	39
2.3.4	Gas Optimization	40
2.3.5	Unused Code	41
2.3.6	Repeated Variable Assignments	41
2.3.7	Incomplete Implementation of Function cancel_order()	42
2.3.8	Code Optimization	44
2.3.9	Unsupported Token Frozen List	46
2.3.10	Redundant Clone in nft_transfer_call()	46
2.3.11	Redundant Information in MftId	47
2.3.12	Lack of Check on Duplicate Tokens in Frozen List	47

2.3.13	Potential Failure of NEAR Transfer	48
2.3.14	Skipped Transfer in Function storage_deposit and storage_deposit	49
2.3.15	Lack of Check on Empty Argument	51
2.3.16	Spelling Error	52
2.3.17	Redundant Event Emission in View Functions	52
2.4	Notes	53
2.4.1	Assumption on the Secure Implementation of Contract Dependencies	53
2.4.2	Unsupported Increase of Selling Tokens for Limit Orders	54
2.4.3	Unsupported Deposit of Native NEAR Tokens	58

Report Manifest

Item	Description
Client	Ref-Finance
Target	Ref DCL Contract

Version History

Version	Date	Description
1.0	December 5th, 2022	First Release
2.0	February 10th, 2023	Second Release
3.0	July 10th, 2023	Third Release
4.0	September 28th, 2023	Fourth Release

About BlockSec The **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 5 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 repository that has been audited includes the **Ref DCL** contract ¹.

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 (**Version 1**), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Ref DCL Contract	Version 1	0b96a993d6b463ef172f27606c903fe4fc5aaa9c
	Version 2	876326a1f09bc1ba37cca372196eb3215700d99e
	Version 3	3e1e1cf814f3ea6321de341dd42200e9bedd19fd
	Version 4	0c4617f1f1b24348ffd08237f3d9c573dc12fe11
	Version 5	0564f9926c2aad2892671210a30e3b61f09116bc
	Version 6	edd130f0a60209b80028b045e460638503af1dc9
	Version 7	1d301ab8b8e822fb41fdbccb13f0581168555fa9

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include **contracts/dcl/src** folder contract only. Specifically, the file covered in this audit include:

- lib.rs
- user_asset.rs
- user.rs
- legacy.rs
- global_config.rs
- utils.rs
- event.rs
- owner.rs
- errors.rs
- api/dcl_liquidity_api.rs
- api/dcl_liquidity_mft.rs
- api/mod.rs
- api/token_receiver.rs
- api/view.rs
- api/dcl_order_api.rs
- api/management.rs
- api/user_asset_api.rs

¹<https://github.com/ref-finance/ref-dcl>

- api/nft_approval.rs
- api/nft.rs
- api/storage_api.rs
- api/dcl_api.rs
- dcl/pool.rs
- dcl/dcl_md.rs
- dcl/mod.rs
- dcl/slot_bitmap.rs
- dcl/common_math.rs
- dcl/point_info.rs
- dcl/swap.rs
- dcl/user_mft_asset.rs
- dcl/oracle.rs
- dcl/user_liquidity.rs
- dcl/utlis.rs
- dcl/swap_math.rs
- dcl/user_order.rs

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 ² and Common Weakness Enumeration ³. 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	High	High	Medium
	Low	Medium	Low
		High	Low
		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.

²https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

³<https://cwe.mitre.org/>

Chapter 2 Findings

In total, we find **fourteen** potential issues. We also have **seventeen** recommendations and **three** notes as follows:

- High Risk: 5
- Medium Risk: 5
- Low Risk: 4
- Recommendations: 17
- Notes: 3

The details are provided in the following sections.

ID	Severity	Description	Category	Status
1	High	Incorrect Storage Check in batch_update_liquidity	Software Security	Fixed
2	Medium	Lack of Check on the withdraw_amount	Software Security	Confirmed
3	High	Non-withdrawable Fees Charged by the Protocol	DeFi Security	Fixed
4	High	Incorrect sqrt_price_loc_96 Calculation in y_swap_x_range_complete_desire()	DeFi Security	Fixed
5	Low	Liquidity on Endpoint Processed Before the Limit Order	DeFi Security	Fixed
6	Medium	Potential Failure in the Callback Function	DeFi Security	Fixed
7	Medium	Improper Rounding Implementation	DeFi Security	Fixed
8	Low	Improper Implementation of internal_mft_transfer()	DeFi Security	Fixed
9	Medium	Lack of Check on Remaining Mft when Updating Farming Contract	DeFi Security	Fixed
10	Medium	Inappropriate Limitation of mft_assets for Farming Contract	DeFi Security	Fixed
11	Low	Lack of Pausable Feature	DeFi Security	Confirmed
12	Low	Liquidity on Endpoint Processed Before the Limit Order	DeFi Security	Fixed
13	High	Lack of Check on Repeated Liquidity in internal_check_remove_liquidity_infos	DeFi Security	Fixed
14	High	Unchecked Received Token in internal_add_order	DeFi Security	Fixed
15	-	Potential Elastic Supply Token Problem	Recommendation	Confirmed
16	-	Potential Centralization Problem	Recommendation	Confirmed
17	-	Redundant Code	Recommendation	Fixed
18	-	Gas Optimization	Recommendation	Fixed
19	-	Unused Code	Recommendation	Fixed
20	-	Repeated Variable Assignments	Recommendation	Fixed
21	-	Incomplete Implementation of Function cancel_order()	Recommendation	Fixed
22	-	Code Optimization	Recommendation	Confirmed
23	-	Unsupported Token Frozen List	Recommendation	Fixed
24	-	Redundant Clone in nft_transfer_call()	Recommendation	Fixed
25	-	Redundant Information in MftId	Recommendation	Confirmed
26	-	Lack of Check on Duplicate Tokens in Frozen List	Recommendation	Confirmed
27	-	Potential Failure of NEAR Transfer	Recommendation	Confirmed
28	-	Skipped Transfer in Function storage_deposit and storage_deposit	Recommendation	Fixed
29	-	Lack of Check on Empty Argument	Recommendation	Fixed
30	-	Spelling Error	Recommendation	Fixed
31	-	Redundant Event Emission in View Functions	Recommendation	Fixed
32	-	Assumption on the Secure Implementation of Contract Dependencies	Notes	Confirmed
33	-	Unsupported Increase of Selling Tokens for Limit Orders	Notes	Confirmed
34	-	Unsupported Deposit of Native NEAR Tokens	Notes	Confirmed

2.1 Software Security

2.1.1 Incorrect Storage Check in batch_update_liquidity

Severity High

Status Fixed in [Version 5](#)

Introduced by [Version 4](#)

Description Function `batch_update_liquidity()` allows the user to add `liquidity` and remove `liquidity` in a batch processing. Since the storage used by the user may be changed in this process, the function verifies that the user's available slot plus the `remove_liquidity_infos.len()` is greater than the `add_liquidity_infos.len()`. However, it is worth noting that removing liquidity may not necessarily lead to an increase in user's available slots. In this case, this check may not be entirely accurate.

```
92 pub fn batch_update_liquidity(  
93     &mut self,  
94     remove_liquidity_infos: Vec<RemoveLiquidityInfo>,  
95     add_liquidity_infos: Vec<AddLiquidityInfo>,  
96     skip_unwrap_near: Option<bool>  
97 ) {  
98     require!(remove_liquidity_infos.len() > 0 && add_liquidity_infos.len() > 0);  
99     self.assert_contract_running();  
100    let user_id = env::predecessor_account_id();  
101    let mut user = self.internal_unwrap_user(&user_id);  
102    let global_config = self.internal_get_global_config();  
103    require!(user.get_available_slots(global_config.storage_price_per_slot, global_config.  
        storage_for_asset) + remove_liquidity_infos.len() as u64 >= add_liquidity_infos.len()  
        as u64, E107_NOT_ENOUGH_STORAGE_FOR_SLOTS);  
104  
105    let mut pool_cache = HashMap::new();  
106    let mut liquiditys = vec![];  
107    let remove_mft_details = self.internal_check_remove_liquidity_infos(&mut user, &mut  
        liquiditys, &mut pool_cache, &remove_liquidity_infos);  
108    for (mft_id, v_liquidity) in remove_mft_details {  
109        self.internal_decrease_mft_supply(&mft_id, v_liquidity);  
110    }  
111    let refund_tokens = self.internal_batch_remove_liquidity(&user_id, &mut pool_cache, &mut  
        liquiditys, remove_liquidity_infos);  
112    for (token_id, amount) in refund_tokens.into_iter() {  
113        user.add_asset(&token_id, amount);  
114    }  
115    self.internal_update_or_burn_liquiditys(&mut user, liquiditys);  
116  
117    let mut lpt_ids = vec![];  
118    let mut inner_id = self.data_mut().latest_liquidity_id;  
119    self.internal_check_add_liquidity_infos(&mut user, &mut lpt_ids, &mut pool_cache, &mut  
        inner_id, &add_liquidity_infos);  
120    self.data_mut().latest_liquidity_id = inner_id;  
121  
122  
123    let (refund_tokens, liquiditys) = self.internal_batch_add_liquidity(&user_id, &lpt_ids, &  
        mut pool_cache, add_liquidity_infos);
```

```
124
125     for (token_id, amount) in refund_tokens {
126         self.process_transfer(&user_id, &token_id, amount, skip_unwrap_near);
127     }
128
129     for (pool_id, pool) in pool_cache {
130         self.internal_set_pool(&pool_id, pool);
131     }
132
133     self.internal_mint_liquiditys(user, liquiditys);
134 }
```

Listing 2.1: contracts/dcl/src/api/dcl_liquidity_api.rs

Impact Users can bypass the limit on the number of added `liquidity`.

Suggestion Check the available slots correctly.

2.1.2 Lack of Check on the `withdraw_amount`

Severity Medium

Status Confirmed

Introduced by Version 4

Description In the function `storage_deposit()`, the minimum amount of `NEAR` to be deposited for new users is set as `STORAGE_BALANCE_MIN_BOUND`. However, users are allowed to withdraw `NEARS` via the function `storage_withdraw()` from their accounts, even if the remaining balance is less than `STORAGE_BALANCE_MIN_BOUND`.

```
103     #[payable]
104     fn storage_withdraw(
105         &mut self,
106         amount: Option<U128>,
107     ) -> StorageBalance {
108         assert_one_yocto();
109         self.assert_contract_running();
110
111         let account_id = env::predecessor_account_id();
112         let mut user = self.internal_unwrap_user(&account_id);
113         let receiver_id = user.sponsor_id.clone();
114         let global_config = self.internal_get_global_config();
115         let storage_price_per_slot = global_config.storage_price_per_slot;
116         let available_slots = user.get_available_slots(storage_price_per_slot, global_config.
            storage_for_asset);
117
118         let max_amount = available_slots as u128 * storage_price_per_slot;
119         let withdraw_amount = if let Some(a) = amount {
120             if a.0 > max_amount { max_amount } else { a.0 }
121         } else {
122             max_amount
123         };
124
125         user.locked_near_for_storage -= withdraw_amount;
```

```
126
127     Event::WithdrawUserStorage {
128         operator: &account_id,
129         receiver: &receiver_id,
130         amount: &U128(withdraw_amount),
131         remain: &U128(user.locked_near_for_storage),
132     }.emit();
133
134     self.internal_set_user(&account_id, user);
135
136     if withdraw_amount > 0 {
137         Promise::new(receiver_id).transfer(withdraw_amount);
138     }
139
140     self.storage_balance_of(account_id).unwrap()
141 }
```

Listing 2.2: contracts/dcl/src/api/storage_api.rs

Impact User's storage fee can be less than `STORAGE_BALANCE_MIN_BOUND`.

Suggestion Add a check to ensure the deposited amount of the user will be at least `STORAGE_BALANCE_MIN_BOUND` after the withdrawal.

Feedback from the Project The contract's design enables the removal of all idle slot fees, including some slot fees that were pre-deposited during registration.

2.2 DeFi Security

2.2.1 Non-withdrawable Fees Charged by the Protocol

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description `total_fee_x_charged` and `total_fee_y_charged` (lines 27-30) are used to record the charged protocol fees during the swap actions. However, the protocol fee can not be withdrawn due to the lack of corresponding functions.

```
3     #[derive(BorshSerialize, BorshDeserialize, Serialize)]
4     #[serde(crate = "near_sdk::serde")]
5     pub struct Pool {
6         pub pool_id: PoolId,
7         pub token_x: AccountId,
8         pub token_y: AccountId,
9         pub fee: u32,
10        pub point_delta: i32,
11
12        pub current_point: i32,
13        #[serde(skip_serializing)]
14        pub sqrt_price_96: U256,
15        #[serde(with = "u128_dec_format")]
```

```
16     pub liquidity: u128,
17     #[serde(with = "u128_dec_format")]
18     pub liquidity_x: u128,
19     #[serde(with = "u128_dec_format")]
20     pub max_liquidity_per_point: u128,
21
22     #[serde(skip_serializing)]
23     pub fee_scale_x_128: U256, // token X fee per unit of liquidity
24     #[serde(skip_serializing)]
25     pub fee_scale_y_128: U256, // token Y fee per unit of liquidity
26
27     #[serde(skip_serializing)]
28     pub total_fee_x_charged: U256,
29     #[serde(skip_serializing)]
30     pub total_fee_y_charged: U256,
31
32     #[serde(with = "u256_dec_format")]
33     pub volume_x_in: U256,
34     #[serde(with = "u256_dec_format")]
35     pub volume_y_in: U256,
36     #[serde(with = "u256_dec_format")]
37     pub volume_x_out: U256,
38     #[serde(with = "u256_dec_format")]
39     pub volume_y_out: U256,
40
41     #[serde(with = "u128_dec_format")]
42     pub total_liquidity: u128,
43     #[serde(with = "u128_dec_format")]
44     pub total_order_x: u128,
45     #[serde(with = "u128_dec_format")]
46     pub total_order_y: u128,
47     #[serde(with = "u128_dec_format")]
48     pub total_x: u128,
49     #[serde(with = "u128_dec_format")]
50     pub total_y: u128,
51
52     #[serde(skip_serializing)]
53     pub point_info: PointInfo,
54     #[serde(skip_serializing)]
55     pub slot_bitmap: SlotBitmap,
56
57     pub state: RunningState,
58 }
```

Listing 2.3: contracts/dcl/src/pool.rs

Impact Protocol fees are locked in the contract.

Suggestion Implement the corresponding withdrawal functions.

2.2.2 Incorrect `sqrt_price_loc_96` Calculation in `y_swap_x_range_complete_desire()`

Severity High

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In function `y_swap_x_range_complete_desire()`, the calculation of `sqrt_price_loc_96` is wrong. According to the current implementation, the `result.loc_pt` calculated from the `sqrt_price_loc_96` is the offset relative to the `left_point`. However, the correct `result.loc_pt` should be the offset relative to the point zero. In this case, the numerator in line 692 should be `sqrt_price_r_96` instead of `sqrt_price_pr_pl_96`.

```
662 /// try to swap from right to left in range [left_point, right_point) with all liquidity used.
663 /// @param liquidity: liquidity of each point in the range
664 /// @param sqrt_price_l_96: sqrt of left point price in 2^96 power
665 /// @param left_point: left point of this range
666 /// @param sqrt_price_r_96: sqrt of right point price in 2^96 power
667 /// @param right_point: right point of this range
668 /// @param desire_x: amount of token X as swap-out
669 /// @return Y2XRangeCompRetDesire
670 pub fn y_swap_x_range_complete_desire(
671     liquidity: u128,
672     sqrt_price_l_96: U256,
673     left_point: i32,
674     sqrt_price_r_96: U256,
675     right_point: i32,
676     desire_x: u128
677 ) -> Y2XRangeCompRetDesire {
678     let mut result = Y2XRangeCompRetDesire::default();
679     let max_x = get_amount_x(liquidity, left_point, right_point, sqrt_price_r_96, sqrt_rate_96(),
680         false).as_u128();
681     if max_x <= desire_x {
682         // maxX <= desireX <= uint128.max
683         result.acquire_x = max_x;
684         result.cost_y = get_amount_y(liquidity, sqrt_price_l_96, sqrt_price_r_96, sqrt_rate_96(),
685             true);
686         result.complete_liquidity = true;
687         return result;
688     }
689     let sqrt_price_pr_pl_96 = get_sqrt_price(right_point - left_point);
690     let sqrt_price_pr_m1_96 = sqrt_price_r_96.mul_fraction_floor(pow_96(), sqrt_rate_96());
691     let div = sqrt_price_pr_pl_96 - U256::from(desire_x).mul_fraction_floor(sqrt_price_r_96 -
692         sqrt_price_pr_m1_96, U256::from(liquidity));
693     let sqrt_price_loc_96 = sqrt_price_pr_pl_96.mul_fraction_floor(pow_96(), div);
694     result.complete_liquidity = false;
695     result.loc_pt = get_log_sqrt_price_floor(sqrt_price_loc_96);
696     result.loc_pt = std::cmp::max(left_point, result.loc_pt);
697     result.loc_pt = std::cmp::min(right_point - 1, result.loc_pt);
698     result.sqrt_loc_96 = get_sqrt_price(result.loc_pt);
699     if result.loc_pt == left_point {
700         result.acquire_x = 0;
701     }
```

```

703     result.cost_y = Default::default();
704     return result;
705 }
706 result.complete_liquidity = false;
707 result.acquire_x = std::cmp::min(
708     get_amount_x(liquidity, left_point, result.loc_pt, result.sqrt_loc_96, sqrt_rate_96(),
709         false).as_u128(),
710     desire_x);
711 result.cost_y = get_amount_y(liquidity, sqrt_price_l_96, result.sqrt_loc_96, sqrt_rate_96(),
712     true);
713 result
714 }
```

Listing 2.4: contracts/dcl/src/swap_math.rs

For example, we have a liquidity whose range is from the `left_point` (A) to the `result.loc_pt` (B), L denotes the amount of liquidity and X denotes the desired amount for token X.

Now we have:

$$\frac{L}{\sqrt{1.0001}^A} + \frac{L}{\sqrt{1.0001}^{A+1}} + \frac{L}{\sqrt{1.0001}^{A+2}} \dots + \frac{L}{\sqrt{1.0001}^{B-1}} = X$$

With D = 1.0001, the formula (a) can be simplified as follows :

$$L * \frac{1 - D^{A-B}}{D^A - D^{A-1}} = X$$

$$L * D^{A-B} = L - X(D^A - D^{A-1})$$

$$D^{B-A} = \frac{L}{L - X(D^A - D^{A-1})}$$

For `result.loc_pt`, we have:

$$B = \log_D \frac{L}{L - X(D^A - D^{A-1})} + A$$

However, the current implementation of Ref-DCL for calculating `result.loc_pt` is:

$$B = \log_D \frac{D^{C-A}}{D^{C-A} - \frac{X}{L} * (D^C - D^{C-1})}$$

where C denotes the `right_point`

$$B = \log_D \frac{L * D^{C-A}}{L * D^{C-A} - X * (D^C - D^{C-1})}$$

$$B = \log_D \frac{L}{L - X * (D^A - D^{A-1})}$$

The `result.loc_pt` calculated from Ref-DCL is incorrect, and the correct calculation should follow the equation (e).

Impact There won't be enough `token_x` swapped out due to the incorrect calculation described above.

Suggestion Replace the `sqrt_price_pr_pl_96` with the `sqrt_price_r_96` when calculating the `sqrt_price_loc_96` in function `y_swap_x_range_complete_desired()`.

2.2.3 Liquidity on Endpoint Processed Before the Limit Order

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description Function `internal_x_swap_y()` is to swap `token_x` to `token_y`. During the swapping process, the liquidity will be processed before the limit order. In this case, when the point stops at the `next_point`, which is an `endpoint`, and the amount of `token_x` is not fully swapped, the liquidity can be used up while the order is not processed. This is inconsistent with the original design.

```
209  /// Process x_swap_y in range
210  /// @param protocol_fee_rate
211  /// @param input_amount: amount of token X
212  /// @param low_boundary_point
213  /// @param is_quote: whether the quote function is calling
214  /// @return (consumed_x, gained_y, is_finished)
215  pub fn internal_x_swap_y(&mut self, protocol_fee_rate: u32, input_amount: u128,
    low_boundary_point: i32, is_quote: bool) -> (u128, u128, bool) {
216      let boundary_point = std::cmp::max(low_boundary_point, LEFT_MOST_POINT);
217      let mut amount = input_amount;
218      let mut amount_x = 0;
219      let mut amount_y = 0;
220      let mut is_finished = false;
221      let mut current_order_or_endpt = self.point_info.get_point_type_value(self.current_point,
        self.point_delta);
222
223      while boundary_point <= self.current_point && !is_finished {
224          if current_order_or_endpt & 2 > 0 {
225              // process limit order
226              let mut point_data = self.point_info.0.get(&self.current_point).unwrap();
227              let mut order_data = point_data.order_data.take().unwrap();
228              let process_ret = self.process_limit_order_y(protocol_fee_rate, &mut order_data,
                amount);
229              is_finished = process_ret.0;
230              (amount, amount_x, amount_y) = (amount-process_ret.1, amount_x+process_ret.1,
                amount_y+process_ret.2);
231
232              self.update_order(&mut point_data, order_data, is_quote);
233
234              if is_finished {
235                  break;
236              }
237          }
238
239          let search_start = self.current_point - 1;
240
241          if current_order_or_endpt & 1 > 0 {
242              // current point is an liquidity endpoint, process liquidity
```

```
243     let process_ret = self.process_liquidity_y(protocol_fee_rate, amount, self.  
        current_point);  
244     is_finished = process_ret.0;  
245     (amount, amount_x, amount_y) = (amount-process_ret.1, amount_x+process_ret.1,  
        amount_y+process_ret.2);  
246  
247     if !is_finished {  
248         // pass endpoint  
249         self.pass_endpoint(self.current_point, is_quote, true);  
250         // move one step to the left  
251         self.current_point -= 1;  
252         self.sqrt_price_96 = get_sqrt_price(self.current_point);  
253         self.liquidity_x = 0;  
254     }  
255     if is_finished || self.current_point < boundary_point {  
256         break;  
257     }  
258 }  
259  
260 // process range liquidity  
261 let next_pt= match self.slot_bitmap.get_nearest_left_valued_slot(search_start, self.  
    point_delta, boundary_point / self.point_delta){  
262     Some(point) => {  
263         if point < boundary_point {  
264             boundary_point  
265         } else {  
266             point  
267         }  
268     },  
269     None => { boundary_point }  
270 };  
271  
272 let process_ret = self.process_liquidity_y(protocol_fee_rate, amount, next_pt);  
273 is_finished = process_ret.0;  
274 (amount, amount_x, amount_y) = (amount-process_ret.1, amount_x+process_ret.1, amount_y+  
    process_ret.2);  
275  
276 if self.current_point == next_pt {  
277     current_order_or_endpt = self.point_info.get_point_type_value(next_pt, self.  
        point_delta);  
278 } else {  
279     current_order_or_endpt = 0;  
280 }  
281  
282  
283 if self.current_point <= boundary_point {  
284     if self.current_point == boundary_point && !is_finished && current_order_or_endpt &  
        2 > 0 {  
285         // this final point should check if there is limit order to trade  
286         let mut point_data = self.point_info.0.get(&self.current_point).unwrap();  
287         let mut order_data = point_data.order_data.take().unwrap();  
288         let process_ret = self.process_limit_order_y(protocol_fee_rate, &mut order_data,  
            amount);
```

```
289         is_finished = process_ret.0;
290         (_, amount_x, amount_y) = (amount-process_ret.1, amount_x+process_ret.1,
                                     amount_y+process_ret.2);
291
292         if !is_quote {
293             point_data.order_data = Some(order_data);
294             self.point_info.0.insert(&self.current_point, &point_data);
295             if order_data.selling_x == 0 && order_data.selling_y == 0 &&
                current_order_or_endpt & 1 == 0 {
296                 self.slot_bitmap.set_zero(self.current_point, self.point_delta);
297             }
298         }
299     }
300     break;
301 }
302 }
303 (amount_x, amount_y, is_finished)
304 }
```

Listing 2.5: contracts/dcl/src/pool.rs

Impact Liquidity on the endpoint may be swapped out before the limit order on the same `endpoint`.

Suggestion Process the `liquidity_y` that ranges from the `current_point` to the `next_point+1` first, if there're still some `token_x` left, move to the `next_point`, and handle the limit order before the liquidity on the point.

2.2.4 Potential Failure in the Callback Function

Severity Medium

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In function `callback_post_withdraw_asset()`, if the `PosmiseResult` is checked as `Failed` and the number of the user's assets has reached the threshold, this callback function will panic in line 5 of function `add_asset()`. In this case, the `Event::Lostfound` will not be emitted.

```
91     #[private]
92     pub fn callback_post_withdraw_asset(
93         &mut self,
94         token_id: AccountId,
95         user_id: AccountId,
96         amount: U128,
97     ) -> bool {
98         require!(
99             env::promise_results_count() == 1,
100             E001_PROMISE_RESULT_COUNT_INVALID
101         );
102         let amount: Balance = amount.into();
103         match env::promise_result(0) {
104             PromiseResult::NotReady => unreachable!(),
105             PromiseResult::Successful(_) => {
```

```
106         true
107     }
108     PromiseResult::Failed => {
109         // This reverts the changes from withdraw function.
110         if let Some(mut user) = self.internal_get_user(&user_id) {
111             user.add_asset(&token_id, amount);
112             self.internal_set_user(&user_id, user);
113
114             Event::Lostfound {
115                 user: &user_id,
116                 token: &token_id,
117                 amount: &U128(amount),
118                 locked: &false,
119             }
120             .emit();
121         } else {
122             Event::Lostfound {
123                 user: &user_id,
124                 token: &token_id,
125                 amount: &U128(amount),
126                 locked: &true,
127             }
128             .emit();
129         }
130         false
131     }
132 }
133 }
```

Listing 2.6: contracts/dcl/src/user_asset.rs

```
4 pub fn add_asset(&mut self, token_id: &AccountId, amount: Balance) {
5     require!(self.assets.len() < DEFAULT_MAX_USER_ASSET_COUNT || self.assets.get(token_id).is_some
        (), "ERR_USER_ASSET_COUNT_EXCEEDED");
6     self.assets.insert(
7         token_id,
8         &(amount + self.assets.get(token_id).unwrap_or(0_u128)).clone(),
9     );
10 }
```

Listing 2.7: contracts/dcl/src/user_asset.rs

The same problem exists in the function `callback_post_withdraw_near()`.

```
135 #[private]
136 pub fn callback_post_withdraw_near(
137     &mut self,
138     user_id: AccountId,
139     amount: U128,
140 ) -> bool {
141     require!(
142         env::promise_results_count() == 1,
143         E001_PROMISE_RESULT_COUNT_INVALID
144     );
```

```
145 let amount: Balance = amount.into();
146 match env::promise_result(0) {
147     PromiseResult::NotReady => unreachable!(),
148     PromiseResult::Successful(_) => {
149         Promise::new(user_id).transfer(amount);
150         true
151     }
152     PromiseResult::Failed => {
153         // This reverts the changes from withdraw function.
154         if let Some(mut user) = self.internal_get_user(&user_id) {
155             user.add_asset(&self.data().wnear_id, amount);
156             self.internal_set_user(&user_id, user);
157
158             Event::Lostfound {
159                 user: &user_id,
160                 token: &self.data().wnear_id,
161                 amount: &U128(amount),
162                 locked: &false,
163             }
164             .emit();
165         } else {
166             Event::Lostfound {
167                 user: &user_id,
168                 token: &self.data().wnear_id,
169                 amount: &U128(amount),
170                 locked: &true,
171             }
172             .emit();
173         }
174         false
175     }
176 }
177 }
```

Listing 2.8: contracts/dcl/src/user_asset.rs

Impact Users' assets may be lost due to the potential failure of the callback function.

Suggestion If the function `add_asset()` is called by the callback function and the number of the user's assets has reached the threshold (i.e., 64), emit an `Event::Lostfound` instead of throwing into a panic.

2.2.5 Improper Rounding Implementation

Severity Medium

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In function `internal_update_order()`, the amount of the `token_x` or `token_y` earned by the user in lines 304-313 is rounded up with function `mul_fraction_ceil()`, which is inconsistent with the calculation in lines 349-358.

```
279 /// Sync user order with point order, try to claim as much earned as possible
280 /// @param ue: user order
```

```
281 /// @param po: point order
282 /// @return earned amount this time
283 pub fn internal_update_order(ue: &mut UserOrder, po: &mut OrderData) -> u128 {
284     let is_earn_y = ue.is_earn_y();
285     let sqrt_price_96 = get_sqrt_price(ue.point);
286     let (total_earn, total_legacy_earn, acc_legacy_earn, cur_acc_earn) = if is_earn_y {
287         (
288             po.earn_y,
289             po.earn_y_legacy,
290             po.acc_earn_y_legacy,
291             po.acc_earn_y,
292         )
293     } else {
294         (
295             po.earn_x,
296             po.earn_x_legacy,
297             po.acc_earn_x_legacy,
298             po.acc_earn_x,
299         )
300     };
301
302     if ue.last_acc_earn < acc_legacy_earn {
303         // this order has been fully filled
304         let mut earn = if is_earn_y {
305             let liquidity =
306                 U256::from(ue.remain_amount).mul_fraction_ceil(sqrt_price_96, pow_96());
307             liquidity.mul_fraction_ceil(sqrt_price_96, pow_96())
308         } else {
309             let liquidity =
310                 U256::from(ue.remain_amount).mul_fraction_ceil(pow_96(), sqrt_price_96);
311             liquidity.mul_fraction_ceil(pow_96(), sqrt_price_96)
312         }
313         .as_u128();
314
315         // update po
316         if earn > total_legacy_earn {
317             // just protect from some rounding errors
318             earn = total_legacy_earn;
319         }
320         if is_earn_y {
321             po.earn_y_legacy -= earn;
322         } else {
323             po.earn_x_legacy -= earn;
324         }
325
326         // update ue
327         ue.last_acc_earn = cur_acc_earn;
328         ue.remain_amount = 0;
329         ue.bought_amount += earn;
330         ue.unclaimed_amount = Some(U128(earn));
331
332         earn
333     } else {
```

```
334 // this order needs to compete earn
335 let mut earn = min((cur_acc_earn - ue.last_acc_earn).as_u128(), total_earn);
336
337 let mut sold = if is_earn_y {
338     let liquidity = U256::from(earn).mul_fraction_ceil(pow_96(), sqrt_price_96);
339     liquidity.mul_fraction_ceil(pow_96(), sqrt_price_96)
340 } else {
341     let liquidity = U256::from(earn).mul_fraction_ceil(sqrt_price_96, pow_96());
342     liquidity.mul_fraction_ceil(sqrt_price_96, pow_96())
343 }
344 .as_u128();
345
346 // actual sold should less or equal to remaining, adjust sold and earn if needed
347 if sold > ue.remain_amount {
348     sold = ue.remain_amount;
349     earn = if is_earn_y {
350         let liquidity =
351             U256::from(sold).mul_fraction_floor(sqrt_price_96, pow_96());
352         liquidity.mul_fraction_floor(sqrt_price_96, pow_96())
353     } else {
354         let liquidity =
355             U256::from(sold).mul_fraction_floor(pow_96(), sqrt_price_96);
356         liquidity.mul_fraction_floor(pow_96(), sqrt_price_96)
357     }
358     .as_u128();
359 }
360
361 // update po
362 if earn > total_earn {
363     // just protect from some rounding errors
364     earn = total_earn;
365 }
366 if is_earn_y {
367     po.earn_y -= earn;
368 } else {
369     po.earn_x -= earn;
370 }
371
372 // update ue
373 ue.last_acc_earn = cur_acc_earn;
374 ue.remain_amount -= sold;
375 ue.bought_amount += earn;
376 ue.unclaimed_amount = Some(U128(earn));
377
378 earn
379 }
380 }
```

Listing 2.9: contracts/dcl/src/user_order.rs

Impact Some users may earn more tokens while others can not withdraw all the tokens.

Suggestion Use function `mul_fraction_floor()` instead of `mul_fraction_ceil()` when calculating the users' earned tokens in lines 304-313.

2.2.6 Improper Implementation of `internal_mft_transfer()`

Severity Low

Status Fixed in [Version 3](#)

Introduced by [Version 2](#)

Description The internal function `internal_mft_transfer()` is implemented to transfer the `mft` tokens between `sender` and `receiver`. However, it does not consider the situation that the `sender` and `receiver` can be the same account. In this case, the `sender/receiver` (e.g., farming contract) can mint infinite `mft` tokens by setting `receiver` as the `sender`.

```
91 fn internal_mft_transfer(  
92     &mut self,  
93     token_id: String,  
94     sender_id: &AccountId,  
95     receiver_id: &AccountId,  
96     amount: u128,  
97     memo: Option<String>,  
98 ) {  
99     let mut sender = self.internal_unwrap_user(sender_id);  
100    let mut receiver = self.internal_unwrap_user(receiver_id);  
101  
102    sender.sub_mft_asset(&token_id, amount);  
103    receiver.add_mft_asset(&token_id, amount);  
104  
105    self.internal_set_user(sender_id, sender);  
106    self.internal_set_user(receiver_id, receiver);  
107  
108    if let Some(memo) = memo {  
109        log!("Memo: {}", memo);  
110    }  
111 }
```

Listing 2.10: `contracts/dcl/src/multi_fungible_token.rs`

Impact Although it can only be done by the `farming` contract, the implementation of the above `internal_mft_transfer` is improper.

Suggestion Add the check to ensure the `sender` and the `receiver` are not the same account.

2.2.7 Lack of Check on Remaining Mft when Updating Farming Contract

Severity Medium

Status Fixed in [Version 3](#)

Introduced by [Version 2](#)

Description The function `set_farming_contract_id()` is used to set and update the `farming` contract. However, there is no check on whether there still exists some `mft` tokens in the previous `farming` contract. In this case, the `mft` tokens can be locked in the previous `farming` contract.

```
86 #[payable]  
87 pub fn set_farming_contract_id(&mut self, farming_contract_id: AccountId) {
```



```
88     assert_one_yocto();
89     self.assert_owner();
90
91     if !self.data().users.contains_key(&farming_contract_id) {
92         self.data_mut().users.insert(
93             &farming_contract_id,
94             &User::new(&farming_contract_id, &env::current_account_id()).into(),
95         );
96         self.data_mut().user_count += 1;
97     }
98
99     self.data_mut().farming_contract_id = farming_contract_id;
100 }
```

Listing 2.11: contracts/dcl/src/owner.rs

Impact Users' `mft` can be locked in the previous `farming` contract.

Suggestion Add the check to ensure no `mft` tokens left in the previous `farming` contract before updating.

2.2.8 Inappropriate Limitation of `mft_assets` for Farming Contract

Severity Medium

Status Fixed in [Version 3](#)

Introduced by [Version 2](#)

Description There is a check in the function `add_mft_asset()` to ensure the amount of the user's `mft` assets will be no larger than `DEFAULT_MAX_USER_ASSET_COUNT` (i.e. 64). However, the `farming` contract, which should hold much more assets than the normal user, is also limited by this number, which is inappropriate.

```
4     pub fn add_mft_asset(&mut self, mft_id: &MftId, amount: Balance) {
5         require!(self.mft_assets.len() < DEFAULT_MAX_USER_ASSET_COUNT || self.mft_assets.get(mft_id)
6             .is_some(), "ERR_USER_ASSET_COUNT_EXCEEDED");
7         self.add_mft_asset_uncheck(mft_id, amount);
8     }
```

Listing 2.12: contracts/dcl/src/user_mft_asset.rs

Impact The `farming` contract will not be able to receive `mft` tokens after the amount of the `mft` assets reaches the cap.

Suggestion The amount of the `mft` tokens for the `farming` contract should be limited by a different value.

2.2.9 Lack of Pausable Feature

Severity Low

Status Confirmed

Introduced by [Version 1](#)

Description In current implementation, even if one of the pools is paused, the user can still add liquidity, append liquidity, merge liquidity, remove liquidity, add order, and cancel order to the paused pool.

```
504 pub fn internal_add_order(  
505     &mut self,  
506     client_id: String,  
507     user_id: &AccountId,  
508     token_id: &AccountId,  
509     amount: Balance,  
510     pool_id: &PoolId,  
511     point: i32,  
512     buy_token: &AccountId,  
513     swapped_amount: Balance,  
514     swap_earn_amount: Balance,  
515 ) -> OrderId {  
516     let mut pool = self.internal_get_pool(pool_id).unwrap();  
517     self.assert_no_frozen_tokens(&[pool.token_x.clone(), pool.token_y.clone()]);  
518     require!(point % pool.point_delta as i32 == 0, E202_ILLEGAL_POINT);  
519     require!(client_id.len() <= MAX_USER_ORDER_CLIENT_ID_LEN, E306_INVALID_CLIENT_ID);  
520     require!(amount - swapped_amount > 0, E307_INVALID_SELLING_AMOUNT);  
521  
522     let mut user = self.internal_unwrap_user(user_id);  
523     let order_key = gen_user_order_key(pool_id, point);  
524     require!(  
525         user.order_keys.get(&order_key).is_none(),  
526         E301_ACTIVE_ORDER_ALREADY_EXIST  
527     );  
528     require!(  
529         user.order_keys.len() < DEFAULT_MAX_USER_ACTIVE_ORDER_COUNT,  
530         E302_USER_ACTIVE_ORDER_NUM_EXCEEDED  
531     );  
532  
533     let mut point_data = pool.point_info.0.get(&point).unwrap_or_default();  
534     let prev_active_order = point_data.has_active_order();  
535     let mut point_order: OrderData = point_data.order_data.unwrap_or_default();  
536  
537     let order_id = gen_order_id(pool_id, &mut self.data_mut().latest_order_id);  
538     let mut order = UserOrder {  
539         client_id,  
540         order_id: order_id.clone(),  
541         owner_id: user_id.clone(),  
542         pool_id: pool_id.clone(),  
543         point,  
544         sell_token: token_id.clone(),  
545         buy_token: buy_token.clone(),  
546         original_deposit_amount: amount,  
547         swap_earn_amount,  
548         original_amount: amount - swapped_amount,  
549         created_at: env::block_timestamp(),  
550         last_acc_earn: U256::zero(),  
551         remain_amount: amount - swapped_amount,  
552         cancel_amount: 0_u128,  
553         bought_amount: 0_u128,  
554         unclaimed_amount: None,  
555     };
```

```
556
557 let (token_x, token_y, _) = pool_id.parse_pool_id();
558 if token_x == (*token_id) {
559     require!(buy_token == &token_y, E303_ILLEGAL_BUY_TOKEN);
560     require!(point >= pool.current_point, E202_ILLEGAL_POINT); // greater or equal to
        current point
561     require!(point <= RIGHT_MOST_POINT, E202_ILLEGAL_POINT);
562     order.last_acc_earn = point_order.acc_earn_y;
563     point_order.selling_x += amount - swapped_amount;
564     pool.total_x += amount - swapped_amount;
565     pool.total_order_x += amount - swapped_amount;
566 } else {
567     require!(buy_token == &token_x, E303_ILLEGAL_BUY_TOKEN);
568     require!(point <= pool.current_point, E202_ILLEGAL_POINT); // less or equal to current
        point
569     require!(point >= LEFT_MOST_POINT, E202_ILLEGAL_POINT);
570     order.last_acc_earn = point_order.acc_earn_x;
571     point_order.selling_y += amount - swapped_amount;
572     pool.total_y += amount - swapped_amount;
573     pool.total_order_y += amount - swapped_amount;
574 }
575 point_order.user_order_count += 1;
576 // update order
577 user.order_keys.insert(&order_key, &order.order_id);
578 self.internal_set_user(user_id, user);
579
580 // update pool info
581 point_data.order_data = Some(point_order);
582 pool.point_info.0.insert(&point, &point_data);
583 if !prev_active_order && !point_data.has_active_liquidity() {
584     pool.slot_bitmap.set_one(point, pool.point_delta);
585 }
586 self.internal_set_pool(pool_id, pool);
587
588 Event::OrderAdded {
589     order_id: &order.order_id,
590     created_at: &U64(env::block_timestamp()),
591     owner_id: &order.owner_id,
592     pool_id: &order.pool_id,
593     point: &order.point,
594     sell_token: &order.sell_token,
595     buy_token: &order.buy_token,
596     original_amount: &U128(order.original_amount),
597     original_deposit_amount: &U128(order.original_deposit_amount),
598     swap_earn_amount: &U128(order.swap_earn_amount),
599 }
600 .emit();
601 self.internal_set_user_order(&order_id, order);
602
603 order_id
604 }
```

Listing 2.13: contracts/dcl/src/user_order.rs

```
165 pub fn cancel_order(&mut self, order_id: OrderId, amount: Option<U128>) -> (U128, U128) {
166     self.assert_contract_running();
167     let mut order = self.internal_unwrap_user_order(&order_id);
168
169     let user_id = env::predecessor_account_id();
170     require!(order.owner_id == user_id, E300_NOT_ORDER_OWNER);
171
172     let mut pool = self.internal_get_pool(&order.pool_id).unwrap();
173     self.assert_no_frozen_tokens(&[pool.token_x.clone(), pool.token_y.clone()]);
174     let mut point_data = pool.point_info.0.get(&order.point).unwrap();
175     let mut point_order: OrderData = point_data.order_data.unwrap();
176
177     let earned = internal_update_order(&mut order, &mut point_order);
178
179     // do cancel
180     let actual_cancel_amount = if let Some(expected_cancel_amount) = amount {
181         min(expected_cancel_amount.into(), order.remain_amount)
182     } else {
183         order.remain_amount
184     };
185     order.cancel_amount += actual_cancel_amount;
186     order.remain_amount -= actual_cancel_amount;
187
188     // update point_data
189     if order.is_earn_y() {
190         pool.total_x -= actual_cancel_amount;
191         pool.total_y -= earned;
192         pool.total_order_x -= actual_cancel_amount;
193         point_order.selling_x -= actual_cancel_amount;
194     } else {
195         pool.total_x -= earned;
196         pool.total_y -= actual_cancel_amount;
197         pool.total_order_y -= actual_cancel_amount;
198         point_order.selling_y -= actual_cancel_amount;
199     }
200     point_data.order_data = if order.remain_amount == 0 {
201         point_order.user_order_count -= 1;
202         if point_order.user_order_count == 0 {
203             pool.total_order_x -= point_order.selling_x;
204             pool.total_order_y -= point_order.selling_y;
205             pool.total_x -= point_order.selling_x;
206             pool.total_y -= point_order.selling_y;
207             None
208         } else {
209             Some(point_order)
210         }
211     } else {
212         Some(point_order)
213     };
214     if !point_data.has_active_liquidity() && !point_data.has_active_order() {
215         pool.slot_bitmap.set_zero(order.point, pool.point_delta);
216     }
```

```
217     if point_data.has_order() || point_data.has_liquidity() {
218         pool.point_info.0.insert(&order.point, &point_data);
219     } else {
220         pool.point_info.0.remove(&order.point);
221     }
222     self.internal_set_pool(&order.pool_id, pool);
223
224     Event::OrderCancelled {
225         order_id: &order.order_id,
226         created_at: &U64(order.created_at),
227         cancel_at: &U64(env::block_timestamp()),
228         owner_id: &order.owner_id,
229         pool_id: &order.pool_id,
230         point: &order.point,
231         sell_token: &order.sell_token,
232         buy_token: &order.buy_token,
233         request_cancel_amount: &amount,
234         actual_cancel_amount: &U128(actual_cancel_amount),
235         original_amount: &U128(order.original_amount),
236         cancel_amount: &U128(order.cancel_amount),
237         remain_amount: &U128(order.remain_amount),
238         bought_amount: &U128(order.bought_amount),
239     }
240     .emit();
241
242     // transfer token to user
243     if earned > 0 {
244         if order.buy_token == self.data().wnear_id {
245             self.process_near_transfer(&order.owner_id, earned);
246         } else {
247             self.process_ft_transfer(&order.owner_id, &order.buy_token, earned);
248         }
249     }
250
251     if actual_cancel_amount > 0 {
252         if order.sell_token == self.data().wnear_id {
253             self.process_near_transfer(&order.owner_id, actual_cancel_amount);
254         } else {
255             self.process_ft_transfer(&order.owner_id, &order.sell_token, actual_cancel_amount);
256         }
257     }
258
259     // deactivate order if needed
260     if order.remain_amount == 0 {
261         // completed order move to user history
262         let order_key = gen_user_order_key(&order.pool_id, order.point);
263         let mut user = self.internal_unwrap_user(&user_id);
264         user.order_keys.remove(&order_key);
265         if user.completed_order_count < DEFAULT_USER_ORDER_HISTORY_LEN {
266             user.history_orders.push(&order);
267         } else {
268             let index = user.completed_order_count % DEFAULT_USER_ORDER_HISTORY_LEN;
269             user.history_orders.replace(index, &order);
```

```
270     }
271     user.completed_order_count += 1;
272     self.internal_set_user(&user_id, user);
273     self.data_mut().user_orders.remove(&order_id);
274     Event::OrderCompleted {
275         order_id: &order.order_id,
276         created_at: &U64(order.created_at),
277         completed_at: &U64(env::block_timestamp()),
278         owner_id: &order.owner_id,
279         pool_id: &order.pool_id,
280         point: &order.point,
281         sell_token: &order.sell_token,
282         buy_token: &order.buy_token,
283         original_amount: &U128(order.original_amount),
284         original_deposit_amount: &U128(order.original_deposit_amount),
285         swap_earn_amount: &U128(order.swap_earn_amount),
286         cancel_amount: &U128(order.cancel_amount),
287         bought_amount: &U128(order.bought_amount),
288     }
289     .emit();
290 } else {
291     self.internal_set_user_order(&order_id, order);
292 }
293
294 (actual_cancel_amount.into(), earned.into())
295 }
```

Listing 2.14: contracts/dcl/src/user_order.rs

```
81 pub fn add_liquidity(
82     &mut self,
83     pool_id: PoolId,
84     left_point: i32,
85     right_point: i32,
86     amount_x: U128,
87     amount_y: U128,
88     min_amount_x: U128,
89     min_amount_y: U128,
90 ) -> LptId {
91     self.assert_contract_running();
92     let user_id = env::predecessor_account_id();
93     let mut user = self.internal_unwrap_user(&user_id);
94     require!(
95         user.liquidity_keys.len() < DEFAULT_MAX_USER_LIQUIDITY_COUNT,
96         E217_USER_LIQUIDITY_COUNT_EXCEEDED
97     );
98
99     let mut pool = self.internal_unwrap_pool(&pool_id);
100     self.assert_no_frozen_tokens(&[pool.token_x.clone(), pool.token_y.clone()]);
101     require!(left_point % pool.point_delta == 0 && right_point % pool.point_delta == 0,
102             E200_INVALID_ENDPOINT);
103     require!(right_point > left_point, E202_ILLEGAL_POINT);
104     require!(right_point - left_point < RIGHT_MOST_POINT, E202_ILLEGAL_POINT);
```

```
104     require!(left_point >= LEFT_MOST_POINT && right_point <= RIGHT_MOST_POINT,
105               E202_ILLEGAL_POINT);
106
107     let (new_liquidity, need_x, need_y, acc_fee_x_in_128, acc_fee_y_in_128) = pool.
108         internal_add_liquidity(left_point, right_point, amount_x.0, amount_y.0, min_amount_x
109                               .0, min_amount_y.0);
110
111     user.sub_asset(&pool.token_x, amount_x.0);
112     user.sub_asset(&pool.token_y, amount_y.0);
113
114     let lpt_id = gen_lpt_id(&pool_id, &mut self.data_mut().latest_liquidity_id);
115     let liquidity = UserLiquidity {
116         lpt_id: lpt_id.clone(),
117         owner_id: user_id.clone(),
118         pool_id: pool_id.clone(),
119         left_point,
120         right_point,
121         last_fee_scale_x_128: acc_fee_x_in_128,
122         last_fee_scale_y_128: acc_fee_y_in_128,
123         amount: new_liquidity,
124         mft_id: String::new(),
125         v_liquidity: 0,
126         unclaimed_fee_x: None,
127         unclaimed_fee_y: None,
128     };
129
130     pool.total_liquidity += new_liquidity;
131     pool.total_x += need_x;
132     pool.total_y += need_y;
133
134     let refund_x = amount_x.0 - need_x;
135     let refund_y = amount_y.0 - need_y;
136     if refund_x > 0{
137         if pool.token_x == self.data().wnear_id {
138             self.process_near_transfer(&user_id, refund_x);
139         } else {
140             self.process_ft_transfer(&user_id, &pool.token_x, refund_x);
141         }
142     }
143     if refund_y > 0{
144         if pool.token_y == self.data().wnear_id {
145             self.process_near_transfer(&user_id, refund_y);
146         } else {
147             self.process_ft_transfer(&user_id, &pool.token_y, refund_y);
148         }
149     }
150
151     self.internal_set_pool(&pool_id, pool);
152     Event::LiquidityAdded {
153         lpt_id: &lpt_id,
154         owner_id: &user_id,
155         pool_id: &pool_id,
156         left_point: &left_point,
157         right_point: &right_point,
```

```
154         added_amount: &U128(new_liquidity),
155         cur_amount: &U128(liquidity.amount),
156         paid_token_x: &U128(need_x),
157         paid_token_y: &U128(need_y),
158     }
159     .emit();
160     self.internal_mint_liquidity(user, liquidity);
161     lpt_id
162 }
```

Listing 2.15: contracts/dcl/src/user_liquidity.rs

```
170 pub fn append_liquidity(
171     &mut self,
172     lpt_id: LptId,
173     amount_x: U128,
174     amount_y: U128,
175     min_amount_x: U128,
176     min_amount_y: U128,
177 ) {
178     self.assert_contract_running();
179     let user_id = env::predecessor_account_id();
180     let mut user = self.internal_unwrap_user(&user_id);
181     let mut liquidity = self.internal_unwrap_user_liquidity(&lpt_id);
182     require!(!liquidity.is_mining(), E218_USER_LIQUIDITY_IS_MINING);
183     require!(user_id == liquidity.owner_id, E215_NOT_LIQUIDITY_OWNER);
184     let mut pool = self.internal_unwrap_pool(&liquidity.pool_id);
185     self.assert_no_frozen_tokens(&[pool.token_x.clone(), pool.token_y.clone()]);
186
187     let (new_liquidity, need_x, need_y, acc_fee_x_in_128, acc_fee_y_in_128) = pool.
        internal_add_liquidity(liquidity.left_point, liquidity.right_point, amount_x.0,
        amount_y.0, min_amount_x.0, min_amount_y.0);
188     user.sub_asset(&pool.token_x, amount_x.0);
189     user.sub_asset(&pool.token_y, amount_y.0);
190
191     liquidity.get_unclaimed_fee(acc_fee_x_in_128, acc_fee_y_in_128);
192     let new_fee_x = liquidity.unclaimed_fee_x.unwrap_or(U128(0)).0;
193     let new_fee_y = liquidity.unclaimed_fee_y.unwrap_or(U128(0)).0;
194
195     pool.total_liquidity += new_liquidity;
196     pool.total_x += need_x;
197     pool.total_y += need_y;
198     pool.total_x -= new_fee_x;
199     pool.total_y -= new_fee_y;
200
201     // refund
202     let refund_x = amount_x.0 - need_x + new_fee_x;
203     let refund_y = amount_y.0 - need_y + new_fee_y;
204     if refund_x > 0 {
205         if pool.token_x == self.data().wnear_id {
206             self.process_near_transfer(&user_id, refund_x);
207         } else {
208             self.process_ft_transfer(&user_id, &pool.token_x, refund_x);

```



```
209     }
210 }
211 if refund_y > 0{
212     if pool.token_y == self.data().wnear_id {
213         self.process_near_transfer(&user_id, refund_y);
214     } else {
215         self.process_ft_transfer(&user_id, &pool.token_y, refund_y);
216     }
217 }
218 // update lpt
219 liquidity.amount += new_liquidity;
220 liquidity.last_fee_scale_x_128 = acc_fee_x_in_128;
221 liquidity.last_fee_scale_y_128 = acc_fee_y_in_128;
222 self.internal_set_user(&user.user_id.clone(), user);
223 self.internal_set_pool(&liquidity.pool_id, pool);
224 Event::LiquidityAppend {
225     lpt_id: &lpt_id,
226     owner_id: &user_id,
227     pool_id: &liquidity.pool_id,
228     left_point: &liquidity.left_point,
229     right_point: &liquidity.right_point,
230     added_amount: &U128(new_liquidity),
231     cur_amount: &U128(liquidity.amount),
232     paid_token_x: &U128(need_x),
233     paid_token_y: &U128(need_y),
234 }
235 .emit();
236 self.internal_set_user_liquidity(&lpt_id, liquidity);
237 }
```

Listing 2.16: contracts/dcl/src/user_liquidity.rs

```
242 pub fn merge_liquidity(
243     &mut self,
244     lpt_id: LptId,
245     lpt_id_list: Vec<LptId>
246 ) {
247     self.assert_contract_running();
248     require!(lpt_id_list.len() > 0, E216_INVALID_LPT_LIST);
249     let user_id = env::predecessor_account_id();
250     let mut retain_liquidity = self.internal_unwrap_user_liquidity(&lpt_id);
251     require!(!retain_liquidity.is_mining(), E218_USER_LIQUIDITY_IS_MINING);
252     require!(retain_liquidity.owner_id == user_id, E215_NOT_LIQUIDITY_OWNER);
253     let mut pool = self.internal_unwrap_pool(&retain_liquidity.pool_id);
254     self.assert_no_frozen_tokens(&[pool.token_x.clone(), pool.token_y.clone()]);
255
256     let mut remove_token_x = 0;
257     let mut remove_token_y = 0;
258     let mut remove_fee_x = 0;
259     let mut remove_fee_y = 0;
260
261     let mut merge_lpt_ids = String::new();
262     for item in lpt_id_list.iter() {
```

```
263     merge_lpt_ids = format!("{}", merge_lpt_ids, if merge_lpt_ids.is_empty() { "" }
264         else { "," }, item);
265     let user = self.internal_unwrap_user(&user_id);
266     let mut liquidity = self.internal_unwrap_user_liquidity(item);
267     require!(item != &lpt_id &&
268         liquidity.owner_id == retain_liquidity.owner_id &&
269         liquidity.pool_id == retain_liquidity.pool_id &&
270         liquidity.left_point == retain_liquidity.left_point &&
271         liquidity.right_point == retain_liquidity.right_point &&
272         !liquidity.is_mining(), E216_INVALID_LPT_LIST);
273
274     let (remove_x, remove_y, acc_fee_x_in_128, acc_fee_y_in_128) =
275         pool.internal_remove_liquidity(liquidity.amount, liquidity.left_point,
276             liquidity.right_point, 0, 0);
277
278     liquidity.get_unclaimed_fee(acc_fee_x_in_128, acc_fee_y_in_128);
279     let fee_x = liquidity.unclaimed_fee_x.unwrap_or(U128(0)).0;
280     let fee_y = liquidity.unclaimed_fee_y.unwrap_or(U128(0)).0;
281
282     remove_token_x += remove_x;
283     remove_token_y += remove_y;
284     remove_fee_x += fee_x;
285     remove_fee_y += fee_y;
286
287     pool.total_liquidity -= liquidity.amount;
288     pool.total_x -= remove_x + fee_x;
289     pool.total_y -= remove_y + fee_y;
290     self.internal_burn_liquidity(user, &liquidity);
291 }
292
293 let (new_liquidity, need_x, need_y, acc_fee_x_in_128, acc_fee_y_in_128) =
294     pool.internal_add_liquidity(retain_liquidity.left_point, retain_liquidity.right_point,
295         remove_token_x, remove_token_y, 0, 0);
296 retain_liquidity.get_unclaimed_fee(acc_fee_x_in_128, acc_fee_y_in_128);
297 let new_fee_x = retain_liquidity.unclaimed_fee_x.unwrap_or(U128(0)).0;
298 let new_fee_y = retain_liquidity.unclaimed_fee_y.unwrap_or(U128(0)).0;
299
300 pool.total_liquidity += new_liquidity;
301 pool.total_x += need_x;
302 pool.total_y += need_y;
303 pool.total_x -= new_fee_x;
304 pool.total_y -= new_fee_y;
305
306 let refund_x = remove_token_x - need_x + new_fee_x + remove_fee_x;
307 let refund_y = remove_token_y - need_y + new_fee_y + remove_fee_y;
308
309 if refund_x > 0{
310     if pool.token_x == self.data().wnear_id {
311         self.process_near_transfer(&user_id, refund_x);
312     } else {
313         self.process_ft_transfer(&user_id, &pool.token_x, refund_x);
314     }
315 }
```

```
314     if refund_y > 0{
315         if pool.token_y == self.data().wnear_id {
316             self.process_near_transfer(&user_id, refund_y);
317         } else {
318             self.process_ft_transfer(&user_id, &pool.token_y, refund_y);
319         }
320     }
321
322     retain_liquidity.amount += new_liquidity;
323     retain_liquidity.last_fee_scale_x_128 = acc_fee_x_in_128;
324     retain_liquidity.last_fee_scale_y_128 = acc_fee_y_in_128;
325
326     self.internal_set_pool(&retain_liquidity.pool_id, pool);
327     Event::LiquidityMerge {
328         lpt_id: &lpt_id,
329         merge_lpt_ids: &merge_lpt_ids,
330         owner_id: &user_id,
331         pool_id: &retain_liquidity.pool_id,
332         left_point: &retain_liquidity.left_point,
333         right_point: &retain_liquidity.right_point,
334         added_amount: &U128(new_liquidity),
335         cur_amount: &U128(retain_liquidity.amount),
336         paid_token_x: &U128(need_x),
337         paid_token_y: &U128(need_y),
338     }
339     .emit();
340     self.internal_set_user_liquidity(&lpt_id, retain_liquidity);
341 }
```

Listing 2.17: contracts/dcl/src/user_liquidity.rs

```
352 pub fn remove_liquidity(
353     &mut self,
354     lpt_id: LptId,
355     amount: U128,
356     min_amount_x: U128,
357     min_amount_y: U128,
358 ) -> (U128, U128) {
359     self.assert_contract_running();
360     let user_id = env::predecessor_account_id();
361     let user = self.internal_unwrap_user(&user_id);
362     let mut liquidity = self.internal_unwrap_user_liquidity(&lpt_id);
363     require!(user_id == liquidity.owner_id, E215_NOT_LIQUIDITY_OWNER);
364     let mut pool = self.internal_unwrap_pool(&liquidity.pool_id);
365     self.assert_no_frozen_tokens(&[pool.token_x.clone(), pool.token_y.clone()]);
366
367     let remove_liquidity = if amount.0 < liquidity.amount { amount.0 } else { liquidity.amount };
368     if remove_liquidity > 0 {
369         require!(!liquidity.is_mining(), E218_USER_LIQUIDITY_IS_MINING);
370     }
371     let (remove_x, remove_y, acc_fee_x_in_128, acc_fee_y_in_128) = pool.
        internal_remove_liquidity(remove_liquidity, liquidity.left_point, liquidity.
```

```
        right_point, min_amount_x.0, min_amount_y.0);
372 liquidity.get_unclaimed_fee(acc_fee_x_in_128, acc_fee_y_in_128);
373
374 let new_fee_x = liquidity.unclaimed_fee_x.unwrap_or(U128(0)).0;
375 let new_fee_y = liquidity.unclaimed_fee_y.unwrap_or(U128(0)).0;
376
377 liquidity.amount -= remove_liquidity;
378
379 let refund_x = remove_x + new_fee_x;
380 let refund_y = remove_y + new_fee_y;
381 if refund_x > 0{
382     if pool.token_x == self.data().wnear_id {
383         self.process_near_transfer(&user_id, refund_x);
384     } else {
385         self.process_ft_transfer(&user_id, &pool.token_x, refund_x);
386     }
387 }
388 if refund_y > 0{
389     if pool.token_y == self.data().wnear_id {
390         self.process_near_transfer(&user_id, refund_y);
391     } else {
392         self.process_ft_transfer(&user_id, &pool.token_y, refund_y);
393     }
394 }
395
396 pool.total_liquidity -= remove_liquidity;
397 pool.total_x -= refund_x;
398 pool.total_y -= refund_y;
399
400 self.internal_set_pool(&liquidity.pool_id, pool);
401
402 Event::LiquidityRemoved {
403     lpt_id: &lpt_id,
404     owner_id: &user_id,
405     pool_id: &liquidity.pool_id,
406     left_point: &liquidity.left_point,
407     right_point: &liquidity.right_point,
408     removed_amount: &U128(remove_liquidity),
409     cur_amount: &U128(liquidity.amount),
410     refund_token_x: &U128(refund_x),
411     refund_token_y: &U128(refund_y),
412 }
413 .emit();
414
415 if liquidity.amount > 0 {
416     liquidity.last_fee_scale_x_128 = acc_fee_x_in_128;
417     liquidity.last_fee_scale_y_128 = acc_fee_y_in_128;
418     self.internal_set_user(&user.user_id.clone(), user);
419     self.internal_set_user_liquidity(&lpt_id, liquidity);
420 } else {
421     self.internal_burn_liquidity(user, &liquidity);
422 }
423
```

```
424         (refund_x.into(), refund_y.into())
425     }
```

Listing 2.18: contracts/dcl/src/user_liquidity.rs

Impact The whole contract instead of affected pools has to be paused in case of emergency.

Suggestion Implement `assert_pool_running()` in above functions.

Feedback from the Project It's a design purpose that we only hold any token exchange (all actions that swap involves) when pausing a pool. So, add/remove order actions would be still active in that case.

2.2.10 Liquidity on Endpoint Processed Before the Limit Order

Severity Low

Status Fixed in [Version 3](#)

Introduced by [Version 1](#)

Description Function `internal_x_swap_y_desire_y()` is designed to swap `token_x` to the desired amount of `token_y`. During the swapping process, the liquidity should be processed after the limit order. However, when the point stops at the `next_point`, which is an endpoint, and the amount of `token_x` is not FULLY swapped, the liquidity may be used up while the order is not processed. This is inconsistent with the original design.

```
550     pub fn internal_x_swap_y_desire_y(&mut self, protocol_fee_rate: u32, desire_y: u128,
551         low_boundary_point: i32, is_quote: bool) -> (u128, u128, bool) {
552         require!(desire_y > 0, E205_INVALID_DESIRE_AMOUNT);
553         let boundary_point = std::cmp::max(low_boundary_point, LEFT_MOST_POINT);
554         let mut is_finished = false;
555         let mut amount_x = 0;
556         let mut amount_y = 0;
557         let mut desire_y = desire_y;
558         let mut current_order_or_endpt = self.point_info.get_point_type_value(self.current_point,
559             self.point_delta);
560         while boundary_point <= self.current_point && !is_finished {
561             if current_order_or_endpt & 2 > 0 {
562                 // process limit order
563                 let mut point_data = self.point_info.0.get(&self.current_point).unwrap();
564                 let mut order_data = point_data.order_data.take().unwrap();
565                 let process_ret = self.process_limit_order_y_desire_y(protocol_fee_rate, &mut
566                     order_data, desire_y);
567                 is_finished = process_ret.0;
568                 (desire_y, amount_x, amount_y) = (if desire_y <= process_ret.2 { 0 } else {
569                     desire_y - process_ret.2 }, amount_x + process_ret.1, amount_y + process_ret.2)
570                 ;
571             }
572             self.update_point_order(&mut point_data, order_data, is_quote);
573         }
```

```
574     let search_start = self.current_point - 1;
575
576     if current_order_or_endpt & 1 > 0 {
577         let process_ret = self.process_liquidity_y_desire_y(protocol_fee_rate, desire_y,
578             self.current_point);
579         is_finished = process_ret.0;
580         (desire_y, amount_x, amount_y) = (desire_y - std::cmp::min(desire_y, process_ret.2)
581             , amount_x+process_ret.1, amount_y+process_ret.2);
582
583         if !is_finished {
584             self.pass_endpoint(self.current_point, is_quote, true);
585             // move one step to the left
586             self.current_point -= 1;
587             self.sqrt_price_96 = get_sqrt_price(self.current_point);
588             self.liquidity_x = 0;
589         }
590     }
591
592     if is_finished || self.current_point < boundary_point {
593         break;
594     }
595
596     let next_pt= match self.slot_bitmap.get_nearest_left_valued_slot(search_start, self.
597         point_delta, boundary_point / self.point_delta){
598         Some(point) => {
599             if point < boundary_point {
600                 boundary_point
601             } else {
602                 point
603             }
604         },
605         None => { boundary_point }
606     };
607     let next_val = self.point_info.get_point_type_value(next_pt, self.point_delta);
608
609     if self.liquidity == 0 {
610         // no liquidity in the range [next_pt, st.currentPoint)
611         self.current_point = next_pt;
612         self.sqrt_price_96 = get_sqrt_price(self.current_point);
613         current_order_or_endpt = next_val;
614     } else {
615         let process_ret = self.process_liquidity_y_desire_y(protocol_fee_rate, desire_y,
616             next_pt);
617         is_finished = process_ret.0;
618         (desire_y, amount_x, amount_y) = (desire_y - std::cmp::min(desire_y, process_ret.2)
619             , amount_x+process_ret.1, amount_y+process_ret.2);
620
621         if self.current_point == next_pt {
622             current_order_or_endpt = next_val;
623         } else {
624             current_order_or_endpt = 0;
625         }
626     }
627 }
```

```
622         if self.current_point <= boundary_point {
623             break;
624         }
625     }
626     (amount_x, amount_y, is_finished)
627 }
```

Listing 2.19: contracts/dcl/src/pool.rs

Impact Liquidity on the endpoint may be swapped out before the limit order on the same endpoint.

Suggestion Process the `liquidity_y` that ranges from the `current_point` to the `next_point+1` first, if there're still some `token_x` left, move to the `next_point`, and handle the limit order before the liquidity on the point.

2.2.11 Lack of Check on Repeated Liquidity in `internal_check_remove_liquidity_infos`

Severity High

Status Fixed in [Version 5](#)

Introduced by [Version 4](#)

Description The function `batch_remove_liquidity()` and `batch_update_liquidity()` are used to remove multiple `liquidity`s of the user in one transaction. The `liquidity` to be removed is passed by the parameter `remove_liquidity_infos`, and validated in the function `internal_check_remove_liquidity_infos()`.

However, if the parameter contains a repeated `liquidity` that is not in the status of mining, the validation will be bypassed, and the repeated `liquidity` will be pushed to the vector `liquidity`s. As a result, every duplicated `liquidity` to be removed will actually be removed only once, while users will be able to receive returned assets corresponding to all these removed `liquidity`s.

```
169 pub fn internal_check_remove_liquidity_infos(
170     &self,
171     user: &mut User,
172     liquiditys: &mut Vec<UserLiquidity>,
173     pool_cache: &mut HashMap<String, Pool>,
174     remove_liquidity_infos: &Vec<RemoveLiquidityInfo>,
175 ) -> HashMap<MftId, u128> {
176     let mut remove_mft_details = HashMap::new();
177     remove_liquidity_infos.iter().for_each(|remove_liquidity_info| {
178         let mut liquidity = self.internal_unwrap_user_liquidity(&remove_liquidity_info.lpt_id);
179         require!(user.user_id == liquidity.owner_id, E215_NOT_LIQUIDITY_OWNER);
180         if remove_liquidity_info.amount.0 > 0 {
181             if liquidity.is_mining() {
182                 if user.mft_assets.get(&liquidity.mft_id).unwrap_or_default() >= liquidity.
183                     v_liquidity {
184                     user.sub_mft_asset(&liquidity.mft_id, liquidity.v_liquidity);
185                     remove_mft_details.entry(liquidity.mft_id).and_modify(|v| *v += liquidity.
186                         v_liquidity).or_insert(liquidity.v_liquidity);
187                     liquidity.mft_id = String::new();
188                     liquidity.v_liquidity = 0;
189                 }else {
```

```
188         env::panic_str(E218_USER_LIQUIDITY_IS_MINING);
189     }
190 }
191 }
192 if !pool_cache.contains_key(&liquidity.pool_id) {
193     let pool = self.internal_unwrap_pool(&liquidity.pool_id);
194     self.assert_no_frozen_tokens(&[pool.token_x.clone(), pool.token_y.clone()]);
195     pool_cache.insert(liquidity.pool_id.clone(), pool);
196 }
197 liquiditys.push(liquidity);
198 });
199 remove_mft_details
200 }
```

Listing 2.20: contracts/dcl/src/dcl/user_liquidity. rs

Impact Repeated `liquidity` in the function `batch_remove_liquidity()` and `batch_update_liquidity()` may drain the pool.

Suggestion Check the `liquidity` to be removed accordingly.

2.2.12 Unchecked Received Token in internal_add_order

Severity High

Status Fixed in [Version 5](#)

Introduced by [Version 1](#)

Description The function `internal_add_order()` enables users to add orders to a specific pool. Users can indicate which token they would like to add with the specific parameter `token_id`. However, the function does not verify the `token_id` properly. When `token_id` does not match the pool's `token_x`, it is automatically assumed to be the `token_y`, which is incorrect. The `token_id` parameter is entirely under user control, and it could be neither `token_x` nor `token_y`, but instead, a spurious and worthless token.

```
357 pub fn internal_add_order(
358     &mut self,
359     client_id: String,
360     user_id: &AccountId,
361     token_id: &AccountId,
362     amount: Balance,
363     pool_id: &PoolId,
364     point: i32,
365     buy_token: &AccountId,
366     swapped_amount: Balance,
367     swap_earn_amount: Balance,
368 ) -> OrderId {
369     let mut pool = self.internal_get_pool(pool_id).unwrap();
370     self.assert_no_frozen_tokens(&[pool.token_x.clone(), pool.token_y.clone()]);
371     require!(point % pool.point_delta as i32 == 0, E202_ILLEGAL_POINT);
372     require!(client_id.len() <= MAX_USER_ORDER_CLIENT_ID_LEN, E306_INVALID_CLIENT_ID);
373     require!(amount - swapped_amount > 0, E307_INVALID_SELLING_AMOUNT);
374
375 }
```



```
376     let mut user = self.internal_unwrap_user(user_id);
377     let order_key = gen_user_order_key(pool_id, point);
378     require!(
379         user.order_keys.get(&order_key).is_none(),
380         E301_ACTIVE_ORDER_ALREADY_EXIST
381     );
382
383
384     let global_config = self.internal_get_global_config();
385     require!(user.get_available_slots(global_config.storage_price_per_slot, global_config.
386         storage_for_asset) > 0, E107_NOT_ENOUGH_STORAGE_FOR_SLOTS);
387
388     let mut point_data = pool.point_info.0.get(&point).unwrap_or_default();
389     let prev_active_order = point_data.has_active_order();
390     let mut point_order: OrderData = point_data.order_data.unwrap_or_default();
391
392
393     let order_id = gen_order_id(pool_id, &mut self.data_mut().latest_order_id);
394     let mut order = UserOrder {
395         client_id,
396         order_id: order_id.clone(),
397         owner_id: user_id.clone(),
398         pool_id: pool_id.clone(),
399         point,
400         sell_token: token_id.clone(),
401         buy_token: buy_token.clone(),
402         original_deposit_amount: amount,
403         swap_earn_amount,
404         original_amount: amount - swapped_amount,
405         created_at: env::block_timestamp(),
406         last_acc_earn: U256::zero(),
407         remain_amount: amount - swapped_amount,
408         cancel_amount: 0_u128,
409         bought_amount: 0_u128,
410         unclaimed_amount: None,
411     };
412
413
414     let (token_x, token_y, _) = pool_id.parse_pool_id();
415     if token_x == (*token_id) {
416         require!(buy_token == &token_y, E303_ILLEGAL_BUY_TOKEN);
417         require!(point >= pool.current_point, E202_ILLEGAL_POINT); // greater or equal to
418             current point
419         require!(point <= RIGHT_MOST_POINT, E202_ILLEGAL_POINT);
420         order.last_acc_earn = point_order.acc_earn_y;
421         point_order.selling_x += amount - swapped_amount;
422         pool.total_x += amount - swapped_amount;
423         pool.total_order_x += amount - swapped_amount;
424     } else {
425         require!(buy_token == &token_x, E303_ILLEGAL_BUY_TOKEN);
426         require!(point <= pool.current_point, E202_ILLEGAL_POINT); // less or equal to current
427             point
```

```
426         require!(point >= LEFT_MOST_POINT, E202_ILLEGAL_POINT);
427         order.last_acc_earn = point_order.acc_earn_x;
428         point_order.selling_y += amount - swapped_amount;
429         pool.total_y += amount - swapped_amount;
430         pool.total_order_y += amount - swapped_amount;
431     }
432     point_order.user_order_count += 1;
433     // update order
434     user.order_keys.insert(&order_key, &order.order_id);
435     self.internal_set_user(user_id, user);
436
437
438     // update pool info
439     point_data.order_data = Some(point_order);
440     pool.point_info.0.insert(&point, &point_data);
441     if !prev_active_order && !point_data.has_active_liquidity() {
442         pool.slot_bitmap.set_one(point, pool.point_delta);
443     }
444     self.internal_set_pool(pool_id, pool);
445
446
447     Event::OrderAdded {
448         order_id: &order.order_id,
449         created_at: &U64(env::block_timestamp()),
450         owner_id: &order.owner_id,
451         pool_id: &order.pool_id,
452         point: &order.point,
453         sell_token: &order.sell_token,
454         buy_token: &order.buy_token,
455         original_amount: &U128(order.original_amount),
456         original_deposit_amount: &U128(order.original_deposit_amount),
457         swap_earn_amount: &U128(order.swap_earn_amount),
458     }
459     .emit();
460     self.internal_set_user_order(&order_id, order);
461
462
463     order_id
464 }
```

Listing 2.21: contracts/dcl/src/dcl/user_order.rs

Impact Users can add orders to the liquidity pool with no cost, except for the gas fee, and drain the pool after the swap.

Suggestion Ensure that `token_id` is either `token_x` or `token_y`.

2.3 Additional Recommendation

2.3.1 Potential Elastic Supply Token Problem

Status Confirmed

Introduced by [Version 1](#)

Description Elastic supply tokens could dynamically adjust their price, supply, user's balance, etc. For example, inflation tokens, deflation tokens, rebasing tokens, etc.

In the current contract implementation, elastic supply tokens are not supported. If the token is a deflation token, there will be a difference between the recorded amount of transferred tokens to this smart contract (as a parameter of function `ft_on_transfer()`) and the actual number of transferred tokens (the token smart contract itself). That's because the token smart contract will burn a small number of tokens.

Suggestion I Do not add elastic supply tokens to the whitelist.

2.3.2 Potential Centralization Problem

Status Confirmed

Introduced by [Version 1](#)

Description This project has potential centralization problems. The `ContractData.owner_id` has the privilege to configure several system parameters (e.g., the `ContractData.protocol_fee_rate`) and pause or resume the contract & pools.

Suggestion I Introducing a decentralization design in the contract is recommended, such as a [multi-signature](#) or a public DAO.

2.3.3 Redundant Code

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In function `update_endpoint()`, if the signed integer `liquidity_data` is checked to be greater than zero, the `liquid_acc_after` will never be less than or equal to the `liquid_acc_before`. Therefore, it is not necessary to have the check in line 162. Similarly, the check in line 169 is also redundant.

```
147 pub fn update_endpoint(  
148     &mut self,  
149     endpoint: i32,  
150     is_left: bool,  
151     current_point: i32,  
152     liquidity_delta: i128,  
153     max_liquidity_per_point: u128,  
154     fee_scale_x_128: U256,  
155     fee_scale_y_128: U256  
156 ) -> bool {  
157     let mut point_data = self.0.remove(&endpoint).unwrap_or_default();  
158     let mut liquidity_data = point_data.liquidity_data.take().unwrap_or_default();  
159     let liquid_acc_before = liquidity_data.liquidity_sum;  
160     let liquid_acc_after = if liquidity_delta > 0 {  
161         let liquid_acc_after = liquid_acc_before + liquidity_delta as u128;  
162         require!(liquid_acc_after > liquid_acc_before);  
163         liquid_acc_after  
164     } else {  
165         let liquid_acc_after = liquid_acc_before - (-liquidity_delta) as u128;  
166         require!(liquid_acc_after < liquid_acc_before);
```

```

167         liquid_acc_after
168     };
169     require!(liquid_acc_after <= max_liquidity_per_point, E203_LIQUIDITY_OVERFLOW);
170     liquidity_data.liquidity_sum = liquid_acc_after;
171
172     if is_left {
173         liquidity_data.liquidity_delta += liquidity_delta;
174     } else {
175         liquidity_data.liquidity_delta -= liquidity_delta;
176     }
177
178     let mut new_or_erase = false;
179     if liquid_acc_before == 0 {
180         new_or_erase = true;
181         if endpoint >= current_point {
182             liquidity_data.acc_fee_x_out_128 = fee_scale_x_128;
183             liquidity_data.acc_fee_y_out_128 = fee_scale_y_128;
184         }
185     } else if liquid_acc_after == 0 {
186         new_or_erase = true;
187     }
188     point_data.liquidity_data = Some(liquidity_data);
189     self.0.insert(&endpoint, &point_data);
190     new_or_erase
191 }

```

Listing 2.22: contracts/dcl/src/point_info.rs

Suggestion I It is suggested to remove the redundant checks.

2.3.4 Gas Optimization

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In function `storage_unregister()`, if the `user.sponsor_id` is the contract itself (`env::current_account_id()`), there is no need to send the native `NEAR` tokens back to itself.

```

52     #[payable]
53     fn storage_unregister(&mut self, #[allow(unused_variables)] force: Option<bool>) -> bool {
54         assert_one_yocto();
55         self.assert_contract_running();
56
57         // force option is useless, leave it for compatible consideration.
58         // User can NOT unregister if there is still have liquidity, order and asset remain!
59         let account_id = env::predecessor_account_id();
60         if let Some(user) = self.internal_get_user(&account_id) {
61             require!(user.is_empty(), E103_STILL_HAS_REWARD);
62             self.data_mut().users.remove(&account_id);
63             self.data_mut().user_count -= 1;
64             Promise::new(user.sponsor_id).transfer(STORAGE_BALANCE_MIN_BOUND);
65             true
66         } else {

```

```
67         false
68     }
69 }
```

Listing 2.23: `cocontracts/dcl/src/storage_impl.rs`

Suggestion I If the `sponsor_id` is the contract itself, the transfer of the storage fee is suggested to be skipped.

2.3.5 Unused Code

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description Function `gen_liquidity_info_key()` is not used in this contract.

```
180 pub type LiquidityInfoKey = String;
181 pub fn gen_liquidity_info_key(left_point: i32, right_point: i32) -> LiquidityInfoKey {
182     format!("{}", left_point, LIQUIDITY_INFO_KEY, right_point)
183 }
```

Listing 2.24: `contracts/dcl/src/utlis.rs`

Suggestion I It is suggested to remove the unused function `gen_liquidity_info_key()`.

2.3.6 Repeated Variable Assignments

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In function `y_swap_x_range_complete_desire()`, the variable `result.complete_liquidity` is assigned twice in line 694 and line 706.

```
662 /// try to swap from right to left in range [left_point, right_point) with all liquidity used.
663 /// @param liquidity: liquidity of each point in the range
664 /// @param sqrt_price_l_96: sqrt of left point price in 2^96 power
665 /// @param left_point: left point of this range
666 /// @param sqrt_price_r_96: sqrt of right point price in 2^96 power
667 /// @param right_point: right point of this range
668 /// @param desire_x: amount of token X as swap-out
669 /// @return Y2XRangeCompRetDesire
670 pub fn y_swap_x_range_complete_desire(
671     liquidity: u128,
672     sqrt_price_l_96: U256,
673     left_point: i32,
674     sqrt_price_r_96: U256,
675     right_point: i32,
676     desire_x: u128
677 ) -> Y2XRangeCompRetDesire {
678     let mut result = Y2XRangeCompRetDesire::default();
679     let max_x = get_amount_x(liquidity, left_point, right_point, sqrt_price_r_96, sqrt_rate_96
680         (), false).as_u128();
681     if max_x <= desire_x {
```

```

681      // maxX <= desireX <= uint128.max
682      result.acquire_x = max_x;
683      result.cost_y = get_amount_y(liquidity, sqrt_price_l_96, sqrt_price_r_96, sqrt_rate_96
        (), true);
684      result.complete_liquidity = true;
685      return result;
686  }
687
688  let sqrt_price_pr_pl_96 = get_sqrt_price(right_point - left_point);
689  let sqrt_price_pr_ml_96 = sqrt_price_r_96.mul_fraction_floor(pow_96(), sqrt_rate_96());
690  let div = sqrt_price_pr_pl_96 - U256::from(desire_x).mul_fraction_floor(sqrt_price_r_96 -
        sqrt_price_pr_ml_96, U256::from(liquidity));
691
692  let sqrt_price_loc_96 = sqrt_price_pr_pl_96.mul_fraction_floor(pow_96(), div);
693
694  result.complete_liquidity = false;
695  result.loc_pt = get_log_sqrt_price_floor(sqrt_price_loc_96);
696
697  result.loc_pt = std::cmp::max(left_point, result.loc_pt);
698  result.loc_pt = std::cmp::min(right_point - 1, result.loc_pt);
699  result.sqrt_loc_96 = get_sqrt_price(result.loc_pt);
700
701  if result.loc_pt == left_point {
702      result.acquire_x = 0;
703      result.cost_y = Default::default();
704      return result;
705  }
706  result.complete_liquidity = false;
707  result.acquire_x = std::cmp::min(
708      get_amount_x(liquidity, left_point, result.loc_pt, result.sqrt_loc_96, sqrt_rate_96(),
        false).as_u128(),
709      desire_x);
710
711  result.cost_y = get_amount_y(liquidity, sqrt_price_l_96, result.sqrt_loc_96, sqrt_rate_96()
        , true);
712  result
713  }

```

Listing 2.25: contracts/dcl/src/swap_math.rs

Suggestion I It is suggested to remove the repeated assignment of variable `result.complete_liquidity` in line 706.

2.3.7 Incomplete Implementation of Function `cancel_order()`

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In lines 194-199 of function `cancel_order()`, the logic mentioned in the Todo comments has not been implemented yet.

```

141  /// @param order_id
142  /// @param amount: max cancel amount of selling token

```

```
143 /// @return (actual removed sell token, bought token till last update)
144 /// Note: cancel_order with 0 amount means claim
145 pub fn cancel_order(&mut self, order_id: OrderId, amount: U128) -> (U128, U128) {
146     self.assert_contract_running();
147     let mut order = self
148         .data()
149         .user_orders
150         .get(&order_id)
151         .expect(E304_ORDER_NOT_FOUND);
152
153     let user_id = env::predecessor_account_id();
154     require!(order.owner_id == user_id, E300_NOT_ORDER_OWNER);
155
156     let mut pool = self.internal_get_pool(&order.pool_id).unwrap();
157     self.assert_pool_running(&pool);
158     let mut point_data = pool.point_info.0.get(&order.point).unwrap();
159     let mut point_order: OrderData = point_data.order_data.unwrap();
160
161     let earned = internal_update_order(&mut order, &mut point_order);
162
163     // do cancel
164     let expected_cancel_amount: Balance = amount.into();
165     let actual_cancel_amount = min(expected_cancel_amount, order.remain_amount);
166     order.cancel_amount += actual_cancel_amount;
167     order.remain_amount -= actual_cancel_amount;
168
169     // update point_data
170     if order.is_earn_y() {
171         pool.total_x -= actual_cancel_amount;
172         pool.total_y -= earned;
173         pool.total_order_x -= actual_cancel_amount;
174         pool.total_order_y -= earned;
175         point_order.selling_x -= actual_cancel_amount;
176     } else {
177         pool.total_x -= earned;
178         pool.total_y -= actual_cancel_amount;
179         pool.total_order_x -= earned;
180         pool.total_order_y -= actual_cancel_amount;
181         point_order.selling_y -= actual_cancel_amount;
182     }
183     if point_order.selling_x == 0 && point_order.selling_y == 0
184     && point_order.earn_y == 0 && point_order.earn_x == 0
185     && point_order.earn_y_legacy == 0 && point_order.earn_x_legacy == 0 {
186         point_data.order_data = None;
187     }
188
189     if point_order.selling_x == 0 && point_order.selling_y == 0 {
190         // update slot_bitmap
191         if !pool.point_info.is_endpoint(order.point, pool.point_delta) {
192             pool.slot_bitmap.set_zero(order.point, pool.point_delta);
193         }
194         // TODO: will implement remove logic on prod env
195         // // see if we can remove point_order
```

```
196         // if point_order.earn_y == 0 && point_order.earn_x == 0
197         // && point_order.earn_y_legacy == 0 && point_order.earn_x_legacy == 0 {
198         //     point_data.order_data = None;
199         // }
200     } else {
201         point_data.order_data = Some(point_order);
202     }
203     pool.point_info.0.insert(&order.point, &point_data);
```

Listing 2.26: contracts/dcl/src/user_order.rs

Suggestion I It is suggested to implement the function `cancel_order()` completely.

2.3.8 Code Optimization

Status Confirmed

Introduced by Version 1

Description When a sequence of swap actions is executed in function `internal_swap()`, there is no check on duplicated pools. If a pool with the duplicated pair of `token_x` and `token_y` is involved in the middle of the sequence, the execution of the swap sequence will not fail until it reaches the middle. In this case, the gas is wasted for executing the previous successful swaps.

```
141     /// @param account_id
142     /// @param pool_ids: all pools participating in swap
143     /// @param input_token: the swap-in token, must be in pool_ids[0].tokens
144     /// @param input_amount: the amount of swap-in token
145     /// @param output_token: the swap-out token, must be in pool_ids[-1].tokens
146     /// @param min_output_amount: minimum number of swap-out token to be obtained
147     /// @return actual got output token amount
148     pub fn internal_swap(
149         &mut self,
150         account_id: &AccountId,
151         pool_ids: Vec<PoolId>,
152         input_token: &AccountId,
153         input_amount: Balance,
154         output_token: &AccountId,
155         min_output_amount: Balance,
156     ) -> Balance {
157         pool_ids.iter().for_each(|pool_id| self.assert_pool_running(&self.internal_unwrap_pool(
158             pool_id)));
159         let mut pool_record = HashSet::new();
160         let protocol_fee_rate = self.data().protocol_fee_rate;
161         let (actual_output_token, actual_output_amount) = {
162             let mut next_input_token_or_last_output_token = input_token.clone();
163             let mut next_input_amount_or_actual_output = input_amount;
164             for pool_id in pool_ids {
165                 let mut pool = self.internal_unwrap_pool(&pool_id);
166                 let is_not_exist = pool_record.insert(format!("{}", pool.token_x, pool.token_y));
167                 require!(is_not_exist, E206_DUPLICATE_POOL);
168                 if next_input_token_or_last_output_token.eq(&pool.token_x) {
```



```
169         pool.internal_x_swap_y(protocol_fee_rate, next_input_amount_or_actual_output
170             , -799999, false);
171         if !is_finished {
172             env::panic_str(&format!("ERR_TOKEN_{}_NOT_ENOUGH", pool.token_y.to_string().
173                 to_uppercase()));
174         }
175
176         pool.total_x += actual_cost;
177         pool.total_y -= out_amount;
178         pool.volume_x_in += U256::from(actual_cost);
179         pool.volume_y_out += U256::from(out_amount);
180
181         next_input_token_or_last_output_token = pool.token_y.clone();
182         next_input_amount_or_actual_output = out_amount;
183     } else if next_input_token_or_last_output_token.eq(&pool.token_y) {
184         let (actual_cost, out_amount, is_finished) =
185             pool.internal_y_swap_x(protocol_fee_rate, next_input_amount_or_actual_output
186                 , 799999, false);
187         if !is_finished {
188             env::panic_str(&format!("ERR_TOKEN_{}_NOT_ENOUGH", pool.token_x.to_string().
189                 to_uppercase()));
190         }
191
192         pool.total_y += actual_cost;
193         pool.total_x -= out_amount;
194         pool.volume_y_in += U256::from(actual_cost);
195         pool.volume_x_out += U256::from(out_amount);
196
197         next_input_token_or_last_output_token = pool.token_x.clone();
198         next_input_amount_or_actual_output = out_amount;
199     } else {
200         env::panic_str(E404_INVALID_POOL_IDS);
201     }
202     self.internal_set_pool(&pool_id, pool);
203 }
204 (
205     next_input_token_or_last_output_token,
206     next_input_amount_or_actual_output,
207 )
208 };
209
210 require!(output_token == &actual_output_token, E212_INVALID_OUTPUT_TOKEN);
211 require!(actual_output_amount >= min_output_amount, E204_SLIPPAGE_ERR);
212
213 if actual_output_amount > 0 {
214     if output_token == &self.data().wnear_id {
215         self.process_near_transfer(account_id, actual_output_amount);
216     } else {
217         self.process_ft_transfer(account_id, output_token, actual_output_amount);
218     }
219 }
220
221 Event::Swap {
222     swapper: account_id,
```

```
218         token_in: input_token,
219         token_out: output_token,
220         amount_in: &U128(input_amount),
221         amount_out: &U128(actual_output_amount),
222     }
223     .emit();
224     actual_output_amount
225 }
```

Listing 2.27: contracts/dcl/src/swap.rs

Suggestion I Check all the pools listed in `pool_ids` before the swap to ensure no duplicate pools exist.

2.3.9 Unsupported Token Frozen List

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description According to the current management of the contract, the contract owner (perhaps a public DAO) can not directly freeze a specified token for some potential emergency.

Suggestion I It is suggested to introduce a feature that can manage the status of tokens as frozen or unfrozen independently.

2.3.10 Redundant Clone in `nft_transfer_call()`

Status Fixed in [Version 3](#)

Introduced by [Version 1](#)

Description In function `nft_transfer_call()`, the input parameters `prev_owner`, `receiver_id`, and `token_id` will not be used in the function `nft_transfer_call()` after the callback function (i.e., `nft_resolve_transfer()`). In this case, there is no need to clone them for saving gas.

```
145     #[payable]
146     fn nft_transfer_call(
147         &mut self,
148         receiver_id: AccountId,
149         token_id: TokenId,
150         approval_id: Option<u64>,
151         memo: Option<String>,
152         msg: String,
153     ) -> PromiseOrValue<bool> {
154         assert_one_yocto();
155         require!(
156             env::prepaid_gas() > GAS_FOR_NFT_TRANSFER_CALL,
157             E501_MORE_GAS_IS_REQUIRED
158         );
159         self.assert_contract_running();
160         let sender_id = env::predecessor_account_id();
161         let (prev_owner, old_approvals) = self.internal_transfer(&token_id, &sender_id, &
            receiver_id, approval_id, memo);
162         // Initiating receiver's call and the callback
```

```

163     ext_receiver::ext(receiver_id.clone())
164     .with_attached_deposit(NO_DEPOSIT)
165     .with_static_gas(env::prepaid_gas() - GAS_FOR_NFT_TRANSFER_CALL)
166     .nft_on_transfer(sender_id.clone(), prev_owner.clone(), token_id.clone(), msg)
167     .then(
168         Self::ext(env::current_account_id())
169         .with_static_gas(GAS_FOR_RESOLVE_TRANSFER)
170         .nft_resolve_transfer(
171             prev_owner.clone(),
172             receiver_id.clone(),
173             token_id.clone(),
174             old_approvals,
175         ),
176     )
177     .into()
178 }

```

Listing 2.28: `contracts/dcl/src/nft.rs`

Suggestion I Remove the function `clone()` for the above mentioned parameters.

2.3.11 Redundant Information in MftId

Status Confirmed

Introduced by [Version 2](#)

Description Function `gen_mft_id()` is used to generate the `MftId` for the corresponding `mft` token, which consists of the `FarmingType`, `pool_id`, `left_point` and `right_point`. However, the `FarmingType` already contains the `left_point` and the `right_point` of the `mft`, which is duplicate.

```

178 pub type MftId = String;
179 pub fn gen_mft_id(pool_id: &PoolId, farming_type: &FarmingType) -> MftId {
180     match farming_type{
181         FarmingType::FixRange { left_point, right_point } => {
182             format!("{}", near_sdk::serde_json::to_string(farming_type).unwrap(),
183                     MFT_ID_BREAK, pool_id, MFT_ID_BREAK, left_point, MFT_ID_BREAK, right_point)
184         }
185     }
186 }

```

Listing 2.29: `contracts/dcl/src/utils.rs`

Suggestion I Delete the redundant information (i.e., `left_point` and `right_point`) in `MftId`.

Feedback from the Project This redundancy is designed to be like this, according to frontend development requests.

2.3.12 Lack of Check on Duplicate Tokens in Frozen List

Status Confirmed

Introduced by [Version 1](#)

Description The `owner` and `operators` can freeze tokens via the function `extend_frozenlist_tokens()`. However, the duplicate tokens in the input are not checked. In this case, the token which is supposed to be added in the list may be omitted.

```
38  #[payable]
39  pub fn extend_frozenlist_tokens(&mut self, tokens: Vec<AccountId>) {
40      assert_one_yocto();
41      require!(self.is_owner_or_operators(), E002_NOT_ALLOWED);
42      for token in tokens {
43          self.data_mut().frozenlist.insert(&token);
44      }
45  }
```

Listing 2.30: contracts/dcl/src/management.rs

Suggestion I Check the return value of function `insert()` in the for loop.

Feedback from the Project As a management interface, operators would check the execution result and corresponding effect to make sure the right tokens are correctly set.

2.3.13 Potential Failure of NEAR Transfer

Status Confirmed

Introduced by Version 1

Description In the callback function `callback_post_withdraw_near()`, if the `PromiseResult` is checked as `Successful`, the contract will transfer `NEARS` to the user. However, the transfer may fail due to the unregistration of the user's `NEAR` account.

```
140  #[private]
141  pub fn callback_post_withdraw_near(
142      &mut self,
143      user_id: AccountId,
144      amount: U128,
145  ) -> bool {
146      require!(
147          env::promise_results_count() == 1,
148          E001_PROMISE_RESULT_COUNT_INVALID
149      );
150      let amount: Balance = amount.into();
151      match env::promise_result(0) {
152          PromiseResult::NotReady => unreachable!(),
153          PromiseResult::Successful(_) => {
154              Promise::new(user_id).transfer(amount);
155              true
156          }
157          PromiseResult::Failed => {
158              // This reverts the changes from withdraw function.
159              if let Some(mut user) = self.internal_get_user(&user_id) {
160                  user.add_asset_uncheck(&self.data().wnear_id, amount);
161                  self.internal_set_user(&user_id, user);
162              }
163              Event::Lostfound {
```

```
164         user: &user_id,
165         token: &self.data().wnear_id,
166         amount: &U128(amount),
167         locked: &false,
168     }
169     .emit();
170 } else {
171     Event::Lostfound {
172         user: &user_id,
173         token: &self.data().wnear_id,
174         amount: &U128(amount),
175         locked: &true,
176     }
177     .emit();
178 }
179 false
180 }
181 }
182 }
```

Listing 2.31: contracts/dcl/src/user_asset.rs

Suggestion I It's suggested to print a log for the potential failure, which is similar to the implementation when `PromiseResult` returned as `Failed`.

Feedback from the Project Although it is a really rare condition, if the account was deleted before transfer, it could be taken as a donation cause even if we record this kind of transfer failure, we could not re-transfer it when the account is back online. As we are unable to tell if this new account owner is the one before.

2.3.14 Skipped Transfer in Function `storage_deposit` and `storage_deposit`

Status Fixed in [Version 5](#)

Introduced by [Version 4](#)

Description In function `storage_unregister()`, NEAR transfer is skipped if the `sponsor_id` matches `env::current_account_id`. This helps to save gas. However, this check is not present in the function `storage_withdraw()` and `storage_deposit()`.

```
44  #[payable]
45  fn storage_deposit(
46      &mut self,
47      account_id: Option<AccountId>,
48      registration_only: Option<bool>,
49  ) -> StorageBalance {
50      self.assert_contract_running();
51
52      let amount = env::attached_deposit();
53      let account_id = account_id.unwrap_or_else(env::predecessor_account_id);
54      let caller_id = env::predecessor_account_id();
55      let already_registered = self.data().users.contains_key(&account_id);
56      let registration_only = registration_only.unwrap_or_default();
```

```
57     if amount < STORAGE_BALANCE_MIN_BOUND && !already_registered {
58         env::panic_str(E102_INSUFFICIENT_STORAGE);
59     }
60
61     if already_registered {
62         if amount > 0 {
63             let mut user = self.internal_unwrap_user(&account_id);
64
65             if caller_id == account_id && account_id != user.sponsor_id {
66                 require!(amount >= user.locked_near_for_storage);
67                 Promise::new(user.sponsor_id).transfer(user.locked_near_for_storage);
68                 user.sponsor_id = caller_id;
69                 user.locked_near_for_storage = amount;
70             } else {
71                 user.locked_near_for_storage += amount;
72             }
73             Event::AppendUserStorage {
74                 operator: &env::predecessor_account_id(),
75                 user: &account_id,
76                 amount: &U128(amount),
77             }.emit();
78             self.internal_set_user(&account_id, user);
79         }
80     } else {
81         let actual_amount =
82         if registration_only {
83             self.internal_set_user(&account_id, User::new(&account_id, &caller_id,
84                 STORAGE_BALANCE_MIN_BOUND));
85             let refund = amount - STORAGE_BALANCE_MIN_BOUND;
86             if refund > 0 {
87                 Promise::new(env::predecessor_account_id()).transfer(refund);
88             }
89             STORAGE_BALANCE_MIN_BOUND
90         } else {
91             self.internal_set_user(&account_id, User::new(&account_id, &caller_id, amount));
92             amount
93         };
94         self.data_mut().user_count += 1;
95         Event::InitUserStorage {
96             operator: &env::predecessor_account_id(),
97             user: &account_id,
98             amount: &U128(actual_amount),
99         }.emit();
100     }
101     self.storage_balance_of(account_id).unwrap()
102 }
103
104 #[payable]
105 fn storage_withdraw(
106     &mut self,
107     amount: Option<U128>,
108 ) -> StorageBalance {
109     assert_one_yocto();
110 }
```

```
109     self.assert_contract_running();
110
111     let account_id = env::predecessor_account_id();
112     let mut user = self.internal_unwrap_user(&account_id);
113     let receiver_id = user.sponsor_id.clone();
114     let global_config = self.internal_get_global_config();
115     let storage_price_per_slot = global_config.storage_price_per_slot;
116     let available_slots = user.get_available_slots(storage_price_per_slot, global_config.
        storage_for_asset);
117
118     let max_amount = available_slots as u128 * storage_price_per_slot;
119     let withdraw_amount = if let Some(a) = amount {
120         if a.0 > max_amount { max_amount } else { a.0 }
121     } else {
122         max_amount
123     };
124
125     user.locked_near_for_storage -= withdraw_amount;
126
127     Event::WithdrawUserStorage {
128         operator: &account_id,
129         receiver: &receiver_id,
130         amount: &U128(withdraw_amount),
131         remain: &U128(user.locked_near_for_storage),
132     }.emit();
133
134     self.internal_set_user(&account_id, user);
135
136     if withdraw_amount > 0 {
137         Promise::new(receiver_id).transfer(withdraw_amount);
138     }
139
140     self.storage_balance_of(account_id).unwrap()
141 }
```

Listing 2.32: contracts/dcl/src/api/storage_api.rs

Suggestion I Add the corresponding check in the function `storage_withdraw()` and `storage_deposit()`.

2.3.15 Lack of Check on Empty Argument

Status Fixed in [Version 5](#)

Introduced by [Version 4](#)

Description The function `set_vip_user()` is used to configure the discounts of swap fees for certain pools of the user. However, if the `discount` is empty, it is meaningless to save the user in the contract.

```
57     #[payable]
58     pub fn set_vip_user(&mut self, user: AccountId, discount: HashMap<PoolId, u32>) {
59         assert_one_yocto();
60         require!(self.is_owner_or_operators(), E002_NOT_ALLOWED);
```

```

61     require!(discount.iter().all(|(_, &v)| v as u128 <= BP_DENOM),
        E011_INVALID_VIP_USER_DISCOUNT);
62     self.data_mut().vip_users.insert(&user, &discount);
63 }

```

Listing 2.33: contracts/dcl/src/api/management.rs

Suggestion I Add a check in the function `set_vip_user()` to verify whether the `discount` is empty.

2.3.16 Spelling Error

Status Fixed in [Version 5](#)

Introduced by [Version 1](#)

Description The spelling of these variables is inappropriate.

File	Variable
contracts/dcl/src/errors.rs, line 22	<code>E104_INSURFFICIENT_DEPOIST</code>
contracts/dcl/src/errors.rs, line 72	<code>E505_SNERER_NOT_APPROVED</code>

Suggestion I Correct the spelling mistakes.

2.3.17 Redundant Event Emission in View Functions

Status Fixed in [Version 7](#)

Introduced by [Version 6](#)

Description Function `internal_add_liquidity` will always emit the `Event::LiquidityAdded` even when `is_view` set as true. While this does not modify the contract's state, it may lead to inaccuracies in off-chain statistics and analytics.

```

71 pub fn internal_add_liquidity(
72     &self,
73     pool: &mut Pool,
74     user_id: &AccountId,
75     lpt_id: LptId,
76     left_point: i32,
77     right_point: i32,
78     amount_x: U128,
79     amount_y: U128,
80     min_amount_x: U128,
81     min_amount_y: U128,
82     is_view: bool
83 ) -> (u128, u128, UserLiquidity) {
84     let (new_liquidity, need_x, need_y, acc_fee_x_in_128, acc_fee_y_in_128) = pool.
        internal_add_liquidity(left_point, right_point, amount_x.0, amount_y.0, min_amount_x
        .0, min_amount_y.0, is_view);
85     let liquidity = UserLiquidity {
86         lpt_id: lpt_id.clone(),
87         owner_id: user_id.clone(),
88         pool_id: pool.pool_id.clone(),
89         left_point,

```



```
90         right_point,
91         last_fee_scale_x_128: acc_fee_x_in_128,
92         last_fee_scale_y_128: acc_fee_y_in_128,
93         amount: new_liquidity,
94         mft_id: String::new(),
95         v_liquidity: 0,
96         unclaimed_fee_x: None,
97         unclaimed_fee_y: None,
98     };
99
100     pool.total_liquidity += new_liquidity;
101     pool.total_x += need_x;
102     pool.total_y += need_y;
103
104     Event::LiquidityAdded {
105         lpt_id: &lpt_id,
106         owner_id: &user_id,
107         pool_id: &pool.pool_id,
108         left_point: &left_point,
109         right_point: &right_point,
110         added_amount: &U128(new_liquidity),
111         cur_amount: &U128(liquidity.amount),
112         paid_token_x: &U128(need_x),
113         paid_token_y: &U128(need_y),
114     }
115     .emit();
116
117     (need_x, need_y, liquidity)
118 }
```

Listing 2.34: contracts/dcl/src/dcl/user_liquidity.rs

Suggestion I Avoid emitting `Event::LiquidityAdded` when `is_view` is true.

2.4 Notes

2.4.1 Assumption on the Secure Implementation of Contract Dependencies

Status Confirmed

Introduced by [Version 1](#)

Description The `Ref_DCL_Contract` is built based on the crates `NEAR-SDK` (version 4.0.0) and `near-contract-standards` (version 4.0.0).

```
3     use near_contract_standards::non_fungible_token::core::NonFungibleTokenCore;
4     use near_contract_standards::non_fungible_token::core::NonFungibleTokenResolver;
5     use near_contract_standards::non_fungible_token::enumeration::NonFungibleTokenEnumeration;
6     use near_contract_standards::non_fungible_token::events::NftTransfer;
7     use near_contract_standards::non_fungible_token::metadata::{
8         NFTContractMetadata, NonFungibleTokenMetadataProvider, NFT_METADATA_SPEC,
9     };
10    use near_contract_standards::non_fungible_token::{Token, TokenId};
```

Listing 2.35: contracts/dcl/src/nft.rs

```
2 use near_contract_standards::non_fungible_token::approval::NonFungibleTokenApproval;
3 use near_contract_standards::non_fungible_token::approval::ext_nft_approval_receiver;
4 use near_contract_standards::non_fungible_token::TokenId;
```

Listing 2.36: contracts/dcl/src/nft_approval.rs

```
2 use near_contract_standards::storage_management::{
3     StorageBalance, StorageBalanceBounds, StorageManagement,
4 };
```

Listing 2.37: contracts/dcl/src/storage_impl.rs

The required interfaces and the basic functionality listed below are provided in the contract:

- * [NEP-171](#) (Non-Fungible Token Core Functionality)
- * [NEP-178](#) (Non-Fungible Token Approval Management)
- * [NEP-181](#) (Non-Fungible Token Enumeration)
- * [NEP-177](#) (Non-Fungible Token Metadata Standard)
- * [NEP-297](#) (Events Standard)
- * [NEP-145](#) (Storage Management)

In this audit, we assume the standard library provided by [NEAR-SDK-RS](#)¹ (i.e., `near_contract_standards`) has no security issues.

2.4.2 Unsupported Increase of Selling Tokens for Limit Orders

Status Confirmed

Introduced by [Version 1](#)

Description Users can reduce the amount of selling tokens for a specific limit order by invoking the function `cancel_order()`.

```
141 /// @param order_id
142 /// @param amount: max cancel amount of selling token
143 /// @return (actual removed sell token, bought token till last update)
144 /// Note: cancel_order with 0 amount means claim
145 pub fn cancel_order(&mut self, order_id: OrderId, amount: U128) -> (U128, U128) {
146     self.assert_contract_running();
147     let mut order = self
148         .data()
149         .user_orders
150         .get(&order_id)
151         .expect(E304_ORDER_NOT_FOUND);
152
153     let user_id = env::predecessor_account_id();
154     require!(order.owner_id == user_id, E300_NOT_ORDER_OWNER);
155
156     let mut pool = self.internal_get_pool(&order.pool_id).unwrap();
```

¹<https://github.com/near/near-sdk-rs>

```
157     self.assert_pool_running(&pool);
158     let mut point_data = pool.point_info.0.get(&order.point).unwrap();
159     let mut point_order: OrderData = point_data.order_data.unwrap();
160
161     let earned = internal_update_order(&mut order, &mut point_order);
162
163     // do cancel
164     let expected_cancel_amount: Balance = amount.into();
165     let actual_cancel_amount = min(expected_cancel_amount, order.remain_amount);
166     order.cancel_amount += actual_cancel_amount;
167     order.remain_amount -= actual_cancel_amount;
168
169     // update point_data
170     if order.is_earn_y() {
171         pool.total_x -= actual_cancel_amount;
172         pool.total_y -= earned;
173         pool.total_order_x -= actual_cancel_amount;
174         pool.total_order_y -= earned;
175         point_order.selling_x -= actual_cancel_amount;
176     } else {
177         pool.total_x -= earned;
178         pool.total_y -= actual_cancel_amount;
179         pool.total_order_x -= earned;
180         pool.total_order_y -= actual_cancel_amount;
181         point_order.selling_y -= actual_cancel_amount;
182     }
183     if point_order.selling_x == 0 && point_order.selling_y == 0
184     && point_order.earn_y == 0 && point_order.earn_x == 0
185     && point_order.earn_y_legacy == 0 && point_order.earn_x_legacy == 0 {
186         point_data.order_data = None;
187     }
188
189     if point_order.selling_x == 0 && point_order.selling_y == 0 {
190         // update slot_bitmap
191         if !pool.point_info.is_endpoint(order.point, pool.point_delta) {
192             pool.slot_bitmap.set_zero(order.point, pool.point_delta);
193         }
194         // TODO: will implement remove logic on prod env
195         // // see if we can remove point_order
196         // if point_order.earn_y == 0 && point_order.earn_x == 0
197         // && point_order.earn_y_legacy == 0 && point_order.earn_x_legacy == 0 {
198         //     point_data.order_data = None;
199         // }
200     } else {
201         point_data.order_data = Some(point_order);
202     }
203     pool.point_info.0.insert(&order.point, &point_data);
```

Listing 2.38: contracts/dcl/src/user_order.rs

However, on the contrary, no function can be used to increase the amount of selling tokens in a limit order.

```
474     /// Place order at given point
```

```
475 /// @param user_id: the owner of this order
476 /// @param token_id: the selling token
477 /// @param amount: the amount of selling token for this order
478 /// @param pool_id: pool of this order
479 /// @param buy_token: the token this order want to buy
480 /// @return OrderId
481 pub fn internal_add_order(
482     &mut self,
483     user_id: &AccountId,
484     token_id: &AccountId,
485     amount: Balance,
486     pool_id: &PoolId,
487     point: i32,
488     buy_token: &AccountId,
489     swapped_amount: Balance,
490     swap_earn_amount: Balance,
491 ) -> OrderId {
492     let mut pool = self.internal_get_pool(pool_id).unwrap();
493     self.assert_pool_running(&pool);
494     require!(point % pool.point_delta as i32 == 0, E202_ILLEGAL_POINT);
495
496     let mut user = self.internal_unwrap_user(user_id);
497     let order_key = gen_user_order_key(pool_id, point);
498     require!(
499         user.order_keys.get(&order_key).is_none(),
500         E301_ACTIVE_ORDER_ALREADY_EXIST
501     );
502     require!(
503         user.order_keys.len() < DEFAULT_MAX_USER_ACTIVE_ORDER_COUNT,
504         E302_USER_ACTIVE_ORDER_NUM_EXCEEDED
505     );
506
507     let mut point_data = pool.point_info.0.get(&point).unwrap_or_default();
508     let mut point_order: OrderData = point_data.order_data.unwrap_or_default();
509
510     let mut order = UserOrder {
511         order_id: gen_order_id(pool_id, &mut self.data_mut().latest_order_id),
512         owner_id: user_id.clone(),
513         pool_id: pool_id.clone(),
514         point,
515         sell_token: token_id.clone(),
516         buy_token: buy_token.clone(),
517         original_deposit_amount: amount,
518         swap_earn_amount,
519         original_amount: amount - swapped_amount,
520         created_at: env::block_timestamp(),
521         last_acc_earn: U256::zero(),
522         remain_amount: amount - swapped_amount,
523         cancel_amount: 0_u128,
524         bought_amount: 0_u128,
525         unclaimed_amount: None,
526     };
527
```

```
528     let (token_x, token_y, _) = pool_id.parse();
529     if token_x == (*token_id) {
530         require!(buy_token == &token_y, E303_ILLEGAL_BUY_TOKEN);
531         require!(point >= pool.current_point, E202_ILLEGAL_POINT); // greater or equal to
                                current point
532         require!(point <= RIGHT_MOST_POINT, E202_ILLEGAL_POINT);
533         order.last_acc_earn = point_order.acc_earn_y;
534         point_order.selling_x += amount - swapped_amount;
535         pool.total_x += amount - swapped_amount;
536         pool.total_order_x += amount - swapped_amount;
537     } else {
538         require!(buy_token == &token_x, E303_ILLEGAL_BUY_TOKEN);
539         require!(point <= pool.current_point, E202_ILLEGAL_POINT); // less or equal to current
                                point
540         require!(point >= LEFT_MOST_POINT, E202_ILLEGAL_POINT);
541         order.last_acc_earn = point_order.acc_earn_x;
542         point_order.selling_y += amount - swapped_amount;
543         pool.total_y += amount - swapped_amount;
544         pool.total_order_y += amount - swapped_amount;
545     }
546     // update order
547     user.order_keys.insert(&order_key, &order.order_id);
548     self.internal_set_user(user_id, user);
549     self.data_mut().user_orders.insert(&order.order_id, &order);
550
551     // update pool info
552     point_data.order_data = Some(point_order);
553     pool.point_info.0.insert(&point, &point_data);
554     pool.slot_bitmap.set_one(point, pool.point_delta);
555     self.internal_set_pool(pool_id, pool);
556
557     Event::OrderAdded {
558         order_id: &order.order_id,
559         created_at: &U64(env::block_timestamp()),
560         owner_id: &order.owner_id,
561         pool_id: &order.pool_id,
562         point: &order.point,
563         sell_token: &order.sell_token,
564         buy_token: &order.buy_token,
565         original_amount: &U128(order.original_amount),
566         original_deposit_amount: &U128(order.original_deposit_amount),
567         swap_earn_amount: &U128(order.swap_earn_amount),
568     }
569     .emit();
570
571     order.order_id.clone()
572 }
573 }
```

Listing 2.39: contracts/dcl/src/user_order.rs

2.4.3 Unsupported Deposit of Native NEAR Tokens

Status Confirmed

Introduced by [Version 1](#)

Description When processing the `wNEAR` transfer, the unwrapped native `NEAR` tokens will be transferred instead of the `wNEAR`.

```
203 pub fn process_near_transfer(&mut self, user_id: &AccountId, amount: Balance) -> Promise {
204     ext_wrap_near::ext(self.data().wnear_id.clone())
205         .with_attached_deposit(1)
206         .with_static_gas(GAS_FOR_NEAR_WITHDRAW)
207         .near_withdraw(amount.into())
208         .then(
209             Self::ext(env::current_account_id())
210                 .with_static_gas(GAS_FOR_RESOLVE_NEAR_WITHDRAW)
211                 .callback_post_withdraw_near(
212                     user_id.clone(),
213                     amount.into(),
214                 ),
215         )
216 }
```

Listing 2.40: `contracts/dcl/src/user_asset.rs`

However, on the contrary, this contract does not accept native `NEAR` tokens as deposits, which may cause inconvenience to the users.