Security Audit Report for Ref DCL Contract

Date: December 9th, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.12.09 13:29:42 +08'00'

Contents

Report Manifest

Item	Description
Client	Ref-Finance
Target	Ref DCL Contract

Version History

Version	Date	Description
1.0	December 9th, 2022	First 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`	`0a4ec89884133a5b9ce258a618edddb9d70bd2fa`

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:

- common_math.rs
- event.rs
- lib.rs
- md.rs
- nft.rs
- oracle.rs (created since `Version 2`)
- point_info.rs
- slot_bitmap.rs
- swap_math.rs
- token_receiver.rs
- user_liquidity.rs
- user.rs
- view.rs
- errors.rs
- legacy.rs
- management.rs
- nft_approval.rs
- owner.rs
- pool.rs
- storage_impl.rs
- swap.rs

[1] https://github.com/ref-finance/ref-dcl

- user_asset.rs
- user_order.rs
- utils.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.

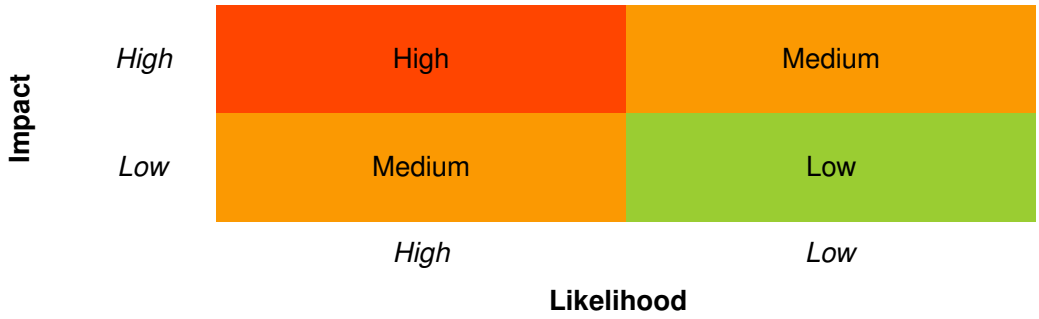
In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

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.

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

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

Table 1.1: Vulnerability Severity Classification



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

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

Chapter 2 Findings

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

- High Risk: 2
- Medium Risk: 2
- Low Risk: 1
- Recommendations: 9
- Notes: 3

ID	Severity	Description	Category	Status
1	High	Non-withdrawable Fees Charged by the Protocol	DeFi Security	Fixed
2	High	Incorrect sqrt_price_loc_96 Calculation in y_swap-_x_range_complete_desire()	DeFi Security	Fixed
3	Low	Liquidity on Endpoint Processed Before the Limit Order	DeFi Security	Fixed
4	Medium	Potential Failure in the Callback Function	DeFi Security	Fixed
5	Medium	Improper Rounding Implementation	DeFi Security	Fixed
4	-	Potential Elastic Supply Token Problem	Recommendation	Confirmed
5	-	Potential Centralization Problem	Recommendation	Confirmed
6	-	Redundant Code	Recommendation	Fixed
7	-	Gas Optimization	Recommendation	Fixed
8	-	Unused Code	Recommendation	Fixed
9	-	Repeated Variable Assignments	Recommendation	Fixed
10	-	Incomplete Implementation of Function cancel_order()	Recommendation	Fixed
11	-	Code Optimization	Recommendation	Confirmed
12	-	Unsupported Token Frozen List	Recommendation	Fixed
15	-	Assumption on the Secure Implementation of Contract Dependencies	Notes	Confirmed
16	-	Unsupported Increasement of Selling Tokens for Limit Orders	Notes	Confirmed
17	-	Unsupported Deposit of Native NEAR Tokens	Notes	Confirmed

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 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)]
```

```rust
#[serde(crate = "near_sdk::serde")]
pub struct Pool {
    pub pool_id: PoolId,
    pub token_x: AccountId,
    pub token_y: AccountId,
    pub fee: u32,
    pub point_delta: i32,

    pub current_point: i32,
    #[serde(skip_serializing)]
    pub sqrt_price_96: U256,
    #[serde(with = "u128_dec_format")]
    pub liquidity: u128,
    #[serde(with = "u128_dec_format")]
    pub liquidity_x: u128,
    #[serde(with = "u128_dec_format")]
    pub max_liquidity_per_point: u128,

    #[serde(skip_serializing)]
    pub fee_scale_x_128: U256, // token X fee per unit of liquidity
    #[serde(skip_serializing)]
    pub fee_scale_y_128: U256, // token Y fee per unit of liquidity

    #[serde(skip_serializing)]
    pub total_fee_x_charged: U256,
    #[serde(skip_serializing)]
    pub total_fee_y_charged: U256,

    #[serde(with = "u256_dec_format")]
    pub volume_x_in: U256,
    #[serde(with = "u256_dec_format")]
    pub volume_y_in: U256,
    #[serde(with = "u256_dec_format")]
    pub volume_x_out: U256,
    #[serde(with = "u256_dec_format")]
    pub volume_y_out: U256,

    #[serde(with = "u128_dec_format")]
    pub total_liquidity: u128,
    #[serde(with = "u128_dec_format")]
    pub total_order_x: u128,
    #[serde(with = "u128_dec_format")]
    pub total_order_y: u128,
    #[serde(with = "u128_dec_format")]
    pub total_x: u128,
    #[serde(with = "u128_dec_format")]
    pub total_y: u128,

    #[serde(skip_serializing)]
    pub point_info: PointInfo,
    #[serde(skip_serializing)]
    pub slot_bitmap: SlotBitmap,

```

```
57        pub state: RunningState,
58    }
```

Listing 2.1: contracts/dcl/src/pool.rs

Impact Protocol fees are locked in the contract.

Suggestion Implement the corresponding withdrawal functions.

2.1.2 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(),
          false).as_u128();
680     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_m1_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_m1_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 2.2: 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:

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

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()`.

2.1.3 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 2.3: 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.

2.1.4 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 2.4: 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 2.5: contracts/dcl/src/user_asset.rs

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

```rust
#[private]
pub fn callback_post_withdraw_near(
    &mut self,
    user_id: AccountId,
    amount: U128,
) -> bool {
    require!(
        env::promise_results_count() == 1,
        E001_PROMISE_RESULT_COUNT_INVALID
    );
    let amount: Balance = amount.into();
    match env::promise_result(0) {
        PromiseResult::NotReady => unreachable!(),
        PromiseResult::Successful(_) => {
            Promise::new(user_id).transfer(amount);
            true
        }
        PromiseResult::Failed => {
            // This reverts the changes from withdraw function.
            if let Some(mut user) = self.internal_get_user(&user_id) {
                user.add_asset(&self.data().wnear_id, amount);
                self.internal_set_user(&user_id, user);

                Event::Lostfound {
                    user: &user_id,
                    token: &self.data().wnear_id,
                    amount: &U128(amount),
                    locked: &false,
                }
                .emit();
            } else {
                Event::Lostfound {
                    user: &user_id,
                    token: &self.data().wnear_id,
                    amount: &U128(amount),
                    locked: &true,
                }
                .emit();
            }
            false
        }
    }
}
```

Listing 2.6: 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.

2.1.5 Improper Rounding Implementation

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In function `interrnal_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 2.7: 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 Additional Recommendation

2.2.1 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.

2.2.2 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`.

2.2.3 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 2.8: contracts/dcl/src/point_info.rs

Suggestion I It is suggested to remove the redundant checks.

2.2.4 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 2.9: 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.

2.2.5 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 2.10: contracts/dcl/src/utils.rs

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

2.2.6 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
                ), false).as_u128();
680         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_m1_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_m1_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 2.11: contracts/dcl/src/swap_math.rs

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

2.2.7 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 2.12: contracts/dcl/src/user_order.rs

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

2.2.8 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(
                pool_id)));
158         let mut pool_record = HashSet::new();
159         let protocol_fee_rate = self.data().protocol_fee_rate;
160         let (actual_output_token, actual_output_amount) = {
161             let mut next_input_token_or_last_output_token = input_token.clone();
162             let mut next_input_amount_or_actual_output = input_amount;
163             for pool_id in pool_ids {
164                 let mut pool = self.internal_unwrap_pool(&pool_id);
165                 let is_not_exist = pool_record.insert(format!("{}|{}", pool.token_x, pool.token_y))
                        ;
166                 require!(is_not_exist, E206_DUPLICATE_POOL);
167                 if next_input_token_or_last_output_token.eq(&pool.token_x) {
168                     let (actual_cost, out_amount, is_finished) =
169                         pool.internal_x_swap_y(protocol_fee_rate, next_input_amount_or_actual_output
                                , -799999, false);
170                     if !is_finished {
171                         env::panic_str(&format!("ERR_TOKEN_{}_NOT_ENOUGH", pool.token_y.to_string().
                                to_uppercase()));
172                     }
173 
174                     pool.total_x += actual_cost;
175                     pool.total_y -= out_amount;
176                     pool.volume_x_in += U256::from(actual_cost);
177                     pool.volume_y_out += U256::from(out_amount);
178 
179                     next_input_token_or_last_output_token = pool.token_y.clone();
180                     next_input_amount_or_actual_output = out_amount;
181                 } else if next_input_token_or_last_output_token.eq(&pool.token_y) {
182                     let (actual_cost, out_amount, is_finished) =
183                         pool.internal_y_swap_x(protocol_fee_rate, next_input_amount_or_actual_output
                                , 799999, false);
184                     if !is_finished {
185                         env::panic_str(&format!("ERR_TOKEN_{}_NOT_ENOUGH", pool.token_x.to_string().
                                to_uppercase()));
186                     }
187 
188                     pool.total_y += actual_cost;
189                     pool.total_x -= out_amount;
190                     pool.volume_y_in += U256::from(actual_cost);
191                     pool.volume_x_out += U256::from(out_amount);
192 
193                     next_input_token_or_last_output_token = pool.token_x.clone();
194                     next_input_amount_or_actual_output = out_amount;
```

```
195                 } else {
196                     env::panic_str(E404_INVALID_POOL_IDS);
197                 }
198                 self.internal_set_pool(&pool_id, pool);
199             }
200             (
201                 next_input_token_or_last_output_token,
202                 next_input_amount_or_actual_output,
203             )
204         };
205
206         require!(output_token == &actual_output_token, E212_INVALID_OUTPUT_TOKEN);
207         require!(actual_output_amount >= min_output_amount, E204_SLIPPAGE_ERR);
208
209         if actual_output_amount > 0 {
210             if output_token == &self.data().wnear_id {
211                 self.process_near_transfer(account_id, actual_output_amount);
212             } else {
213                 self.process_ft_transfer(account_id, output_token, actual_output_amount);
214             }
215         }
216         Event::Swap {
217             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 2.13: contracts/dcl/src/swap.rs

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

2.2.9 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 Notes

2.3.1 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 2.14: 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 2.15: contracts/dcl/src/nft_approval.rs

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

Listing 2.16: 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.

2.3.2 Unsupported Increasement 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) {
```

[1] https://github.com/near/near-sdk-rs

```
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 2.17: 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(),
```

```
            original_deposit_amount: amount,
            swap_earn_amount,
            original_amount: amount - swapped_amount,
            created_at: env::block_timestamp(),
            last_acc_earn: U256::zero(),
            remain_amount: amount - swapped_amount,
            cancel_amount: 0_u128,
            bought_amount: 0_u128,
            unclaimed_amount: None,
        };

        let (token_x, token_y, _) = pool_id.parse();
        if token_x == (*token_id) {
            require!(buy_token == &token_y, E303_ILLEGAL_BUY_TOKEN);
            require!(point >= pool.current_point, E202_ILLEGAL_POINT); // greater or equal to
                current point
            require!(point <= RIGHT_MOST_POINT, E202_ILLEGAL_POINT);
            order.last_acc_earn = point_order.acc_earn_y;
            point_order.selling_x += amount - swapped_amount;
            pool.total_x += amount - swapped_amount;
            pool.total_order_x += amount - swapped_amount;
        } else {
            require!(buy_token == &token_x, E303_ILLEGAL_BUY_TOKEN);
            require!(point <= pool.current_point, E202_ILLEGAL_POINT); // less or equal to current
                point
            require!(point >= LEFT_MOST_POINT, E202_ILLEGAL_POINT);
            order.last_acc_earn = point_order.acc_earn_x;
            point_order.selling_y += amount - swapped_amount;
            pool.total_y += amount - swapped_amount;
            pool.total_order_y += amount - swapped_amount;
        }
        // update order
        user.order_keys.insert(&order_key, &order.order_id);
        self.internal_set_user(user_id, user);
        self.data_mut().user_orders.insert(&order.order_id, &order);

        // update pool info
        point_data.order_data = Some(point_order);
        pool.point_info.0.insert(&point, &point_data);
        pool.slot_bitmap.set_one(point, pool.point_delta);
        self.internal_set_pool(pool_id, pool);

        Event::OrderAdded {
            order_id: &order.order_id,
            created_at: &U64(env::block_timestamp()),
            owner_id: &order.owner_id,
            pool_id: &order.pool_id,
            point: &order.point,
            sell_token: &order.sell_token,
            buy_token: &order.buy_token,
            original_amount: &U128(order.original_amount),
            original_deposit_amount: &U128(order.original_deposit_amount),
            swap_earn_amount: &U128(order.swap_earn_amount),
```

```
568         }
569         .emit();
570 
571         order.order_id.clone()
572     }
573 }
```

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

2.3.3 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 2.19: 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.