BLOCKSEC

Security Audit

Report for Ref-Exchange Smart Contracts

Date: March 23rd, 2024 **Version:** 4.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Ref Finance
Target	Ref-Exchange Smart Contracts

Version History

Version	Date	Description
1.0	November 2nd, 2022	First Version
2.0	November 20th, 2022	Second Version
3.0	September 28th, 2023	Third Version
4.0	March 23rd, 2024	Fourth Version

Signature

Digitally signed by: BlockSec (https://blocksec.com)

Email: audit@blocksec.com

Date: 2024-03-28 02:53:27+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

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

The target of this audit is the code repository of Ref-Exchange Smart Contracts[1] of Ref Finance. Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include `ref-contracts/ref-exchange/src` folder contract only. Specifically, the files covered in this audit include:

```
1  custom_keys.rs
2  action.rs
3  owner.rs
4  account_deposit.rs
5  legacy.rs
6  token_receiver.rs
7  lib.rs
8  simple_pool.rs
9  storage_impl.rs
10 views.rs
11 errors.rs
12 pool.rs
13 multi_fungible_token.rs
14 shadow_actions.rs
15 unit_lpt_cumulative_infos.rs
16 utils.rs
17 admin_fee.rs
18 stable_swap/mod.rs
19 stable_swap/math.rs
20 rated_swap/linear_rate.rs
21 rated_swap/nearx_rate.rs
22 rated_swap/rate.rs
23 rated_swap/sfrax_rate.rs
24 rated_swap/mod.rs
25 rated_swap/math.rs
26 rated_swap/stnear_rate.rs
```

Listing 1.1: Audit Scope for this Report

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

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

Project	Version	Commit Hash
Ref-Exchange Smart Contracts	Version 1	536a60c842e018a535b478c874c747bde82390dd
	Version 2	19e98ec7e70b72d0a2bb1281eb8cd171cebcc931
	Version 3	edea28e1f9bb4f66f5f64eb8448f681f92ef3f10
	Version 4	422591c276224c6477cd638a88ab21807b4b5795
	Version 5	a708597e7333a6f7e2b335682af1cd4f01fb35c3
	Version 6	202ebb7d81ceed0c72b2434081ba7152e7c4075a

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.

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

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

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

Table 1.1: Vulnerability Severity Classification

Impact		Likelihood: *High*	Likelihood: *Low*
	High	High	Medium
	Low	Medium	Low

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

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

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

Chapter 2 Findings

In total, we find **four** potential issue. Besides, we also have **twelve** recommendations and **four** note.

- Medium Risk: 3
- Low Risk: 1
- Recommendation: 12
- Note: 4

ID	Severity	Description	Category	Status
1	Medium	Improper Account Unregistration	Software Security	Fixed
2	Medium	Lack of Storage Usage Check in function `ft_on_transfer`	Software Security	Fixed
3	Low	Unrestricted Referral Account	DeFi Security	Fixed
4	Medium	Incorrect Admin Fees Calculation in Simple Pool	DeFi Security	Fixed
5	-	Lack of Check on Guardians' Removal	Recommendation	Fixed
6	-	Two-Step Transfer of Privileged Account Ownership	Recommendation	Confirmed
7	-	Potential Elastic Supply Token Problem	Recommendation	Confirmed
8	-	Improper Check on the Admin Fees	Recommendation	Fixed
9	-	Lack of Check in `retrieve_unmanaged_token()`	Recommendation	Confirmed
10	-	Lack of Check on the Gas Used by `migrate()`	Recommendation	Fixed
11	-	Code Optimization (I)	Recommendation	Fixed
12	-	Code Optimization (II)	Recommendation	Fixed
13	-	Avoid Logging in View Functions	Recommendation	Fixed
14	-	Slippage Protection in Function `add_liquidity`	Recommendation	Fixed*
15	-	Spelling Errors	Recommendation	Fixed
16	-	Lack of Checks for Oracle Configuration	Recommendation	Fixed
17	-	Delayed Price in Rated Swap Pool	Note	Confirmed
18	-	Timely Triggering `update_token_rate()`	Note	Confirmed
19	-	Sensitive Functions Managed by DAO	Note	Confirmed
20	-	Reliability of Oracle	Note	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Improper Account Unregistration

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `storage_unregister()` allows users to unregister their accounts, and get back their deposits (i.e., `NEAR`s). However, it doesn't check whether the `legacy_tokens` of accounts are empty before the unregistration.

```
56  #[allow(unused_variables)]
57  #[payable]
58  fn storage_unregister(&mut self, force: Option<bool>) -> bool {
59      assert_one_yocto();
60      self.assert_contract_running();
61      let account_id = env::predecessor_account_id();
62      if let Some(account_deposit) = self.internal_get_account(&account_id) {
63          // TODO: figure out force option logic.
64          assert!(
65              account_deposit.tokens.is_empty(),
66              "{}", ERR18_TOKENS_NOT_EMPTY
67          );
68          self.accounts.remove(&account_id);
69          Promise::new(account_id.clone()).transfer(account_deposit.near_amount);
70          true
71      } else {
72          false
73      }
74  }
```

Listing 2.1: src/storage_impl.rs

Impact Users may lose tokens that are recorded in `legacy_tokens`.

Suggestion I Add the check to ensure `legacy_tokens` of accounts are empty before the removal.

Feedback from the Project Will fix this in the next accumulated contract upgrade.

2.1.2 Lack of Storage Usage Check in function ft_on_transfer

Severity Medium

Status Fixed in `Version 4`

Introduced by `Version 3`

Description `HotZap` allows users to seamlessly swap tokens and provide liquidity to pools in a single transaction via the function `ft_on_transfer()`. However, adding liquidity may potentially increase the pool's storage usage. The function does not perform sufficient validation of user's storage usage.

```
116 TokenReceiverMessage::HotZap {
117     referral_id,
118     hot_zap_actions,
119     add_liquidity_infos
120 } => {
121     assert!(hot_zap_actions.len() > 0 && add_liquidity_infos.len() > 0);
122     let sender_id: AccountId = sender_id.into();
123     let mut account = self.internal_unwrap_account(&sender_id);
124     let referral_id = referral_id.map(|x| x.to_string());
```

```
125         let out_amounts = self.internal_direct_actions(
126             token_in,
127             amount.0,
128             referral_id,
129             &hot_zap_actions,
130         );
131 
132         let mut token_cache = TokenCache::new();
133         for (out_token_id, out_amount) in out_amounts {
134             token_cache.add(&out_token_id, out_amount);
135         }
136 
137         for add_liquidity_info in add_liquidity_infos {
138             let mut pool = self.pools.get(add_liquidity_info.pool_id).expect(ERR85_NO_POOL);
139             let tokens_in_pool = match &pool {
140                 Pool::SimplePool(p) => p.token_account_ids.clone(),
141                 Pool::RatedSwapPool(p) => p.token_account_ids.clone(),
142                 Pool::StableSwapPool(p) => p.token_account_ids.clone(),
143             };
144 
145             let mut add_liquidity_amounts = add_liquidity_info.amounts.iter().map(|v| v.0).collect
                    ();
146 
147             match pool {
148                 Pool::SimplePool(_) => {
149                     pool.add_liquidity(
150                         &sender_id,
151                         &mut add_liquidity_amounts,
152                         false
153                     );
154                     if let Some(min_amounts) = add_liquidity_info.min_amounts {
155                         // Check that all amounts are above request min amounts in case of front
                                running that changes the exchange rate.
156                         for (amount, min_amount) in add_liquidity_amounts.iter().zip(min_amounts.
                                iter()) {
157                             assert!(amount >= &min_amount.0, "{}", ERR86_MIN_AMOUNT);
158                         }
159                     }
160                 },
161                 Pool::StableSwapPool(_) | Pool::RatedSwapPool(_) => {
162                     let min_shares = add_liquidity_info.min_shares.expect("Need input min_shares");
163                     pool.add_stable_liquidity(
164                         &sender_id,
165                         &add_liquidity_amounts,
166                         min_shares.into(),
167                         AdminFees::new(self.admin_fee_bps),
168                         false
169                     );
170                 }
171             };
172 
173             for (cost_token_id, cost_amount) in tokens_in_pool.iter().zip(add_liquidity_amounts.
                    into_iter()) {
```

```
174                 token_cache.sub(cost_token_id, cost_amount);
175             }
176
177             self.pools.replace(add_liquidity_info.pool_id, &pool);
178         }
179
180         for (remain_token_id, remain_amount) in token_cache.0.iter() {
181             account.deposit(remain_token_id, *remain_amount);
182         }
183
184         self.internal_save_account(&sender_id, account);
185
186         env::log(
187             format!(
188                 "HotZap remain internal account assets: {:?}",
189                 token_cache.0
190             )
191             .as_bytes(),
192         );
193
194         PromiseOrValue::Value(U128(0))
195     }
```

Listing 2.2: src/token_receiver.rs

Impact The storage fees that should be claimed from users may be bypassed. Furthermore, users can claim extra fees with the function `remove_liquidity` or `remove_liquidity_by_tokens`.

Suggestion I Add storage check in the function `ft_on_transfer()`.

2.2 DeFi Security

2.2.1 Unrestricted Referral Account

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The protocol allows the user to provide a `referral` account for receiving a reward during the swap process. However, there is no restriction on this `referral` account, which allows the user to receive the `referral` fee as a reward by providing his/her own address.

```
62  #[allow(unreachable_code)]
63  fn ft_on_transfer(
64      &mut self,
65      sender_id: ValidAccountId,
66      amount: U128,
67      msg: String,
68  ) -> PromiseOrValue<U128> {
69      self.assert_contract_running();
70      let token_in = env::predecessor_account_id();
71      // feature frozenlist
```

```
72      self.assert_no_frozen_tokens(&[token_in.clone()]);
73      if msg.is_empty() {
74          // Simple deposit.
75          self.internal_deposit(sender_id.as_ref(), &token_in, amount.into());
76          PromiseOrValue::Value(U128(0))
77      } else {
78          // instant swap
79          let message =
80              serde_json::from_str::<TokenReceiverMessage>(&msg).expect(ERR28_WRONG_MSG_FORMAT);
81          match message {
82              TokenReceiverMessage::Execute {
83                  referral_id,
84                  actions,
85              } => {
86                  let referral_id = referral_id.map(|x| x.to_string());
87                  let out_amounts = self.internal_direct_actions(
88                      token_in,
89                      amount.0,
90                      referral_id,
91                      &actions,
92                  );
93                  for (token_out, amount_out) in out_amounts.into_iter() {
94                      self.internal_send_tokens(sender_id.as_ref(), &token_out, amount_out);
95                  }
96                  // Even if send tokens fails, we don't return funds back to sender.
97                  PromiseOrValue::Value(U128(0))
98              }
99          }
100     }
101 }
```

Listing 2.3: src/token_receiver.rs

Impact Users can earn the additional referral fee in the swap process, which is against the original design.

Suggestion I Ensure the `referral` account is different from the `sender_id`.

Feedback from the Project The new rated referral fee feature would include a fix for it.

2.2.2 Incorrect Admin Fees Calculation in Simple Pool

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The shares minted to admin for Simple Pool are calculated as follows:

$$\texttt{Minted\_Share} = \texttt{Total\_Share} * \texttt{Admin\_Fee} * \frac{\sqrt{\texttt{k'}} - \sqrt{\texttt{k}}}{\sqrt{\texttt{k'}}}$$

The actual `Admin_Fee` is $\frac{\texttt{Admin\_Fee\_Amount}}{\texttt{Total\_Fee\_Amount}}$. The total value of the pool can be represented as $\sqrt{\texttt{k'}}$, and the `Total_Fee_Amount` can be represented as $\sqrt{\texttt{k'}} - \sqrt{\texttt{k}}$. Thus, `Admin_Fee_Amount` should be

$$\frac{\texttt{Minted\_Share}}{\texttt{Total\_Share} + \texttt{Minted\_Share}} * \sqrt{\texttt{k'}}$$

In this case, given the Minted_Share above, the actual Admin_Fee could be calculated as follows:

$$\text{Actual Admin\_Fee} = \frac{\text{Total\_Share} * \text{Admin\_Fee} * \frac{\sqrt{k'}-\sqrt{k}}{\sqrt{k'}}}{\text{Total\_Share} * \text{Admin\_Fee} * \frac{\sqrt{k'}-\sqrt{k}}{\sqrt{k'}} + \text{Total\_Share}} * \frac{\sqrt{k'}}{\sqrt{k'}-\sqrt{k}} = \frac{1}{\frac{1}{\text{Admin\_Fee}} + \frac{\sqrt{k'}-\sqrt{k}}{\sqrt{k'}}}$$

, which is always less than the Admin_Fee in the Simple Pool. That's to say, the calculation of the amount of shares minted for the admin is incorrect.

To ensure that the actual Admin_Fee is equal to Admin_Fee, we have the following equations:

$$\text{Actual Admin\_Fee} = \frac{\text{Admin\_Fee\_Amount}}{\text{Total\_Fee\_Amount}} = \frac{\text{Minted\_Share} * \sqrt{k'}}{\text{Total\_Share} + \text{Minted\_Share}} \div (\sqrt{k'}-\sqrt{k}) = \text{Admin\_Fee}$$

Given the formula above, the minted share should be calculated as following:

$$\text{Minted\_Share} = \text{Total\_Share} * \frac{\sqrt{k'}-\sqrt{k}}{(\frac{1}{\text{Admin\_Fee}} - 1) * \sqrt{k'} + \sqrt{k}}$$

```
269   pub fn swap(
270       &mut self,
271       token_in: &AccountId,
272       amount_in: Balance,
273       token_out: &AccountId,
274       min_amount_out: Balance,
275       admin_fee: &AdminFees,
276   ) -> Balance {
277       assert_ne!(token_in, token_out, "{}", ERR73_SAME_TOKEN);
278       let in_idx = self.token_index(token_in);
279       let out_idx = self.token_index(token_out);
280       let amount_out = self.internal_get_return(in_idx, amount_in, out_idx);
281       assert!(amount_out >= min_amount_out, "{}", ERR68_SLIPPAGE);
282       env::log(
283           format!(
284               "Swapped {} {} for {} {}",
285               amount_in, token_in, amount_out, token_out
286           )
287           .as_bytes(),
288       );
289
290       let prev_invariant =
291           integer_sqrt(U256::from(self.amounts[in_idx]) * U256::from(self.amounts[out_idx]));
292
293       self.amounts[in_idx] += amount_in;
294       self.amounts[out_idx] -= amount_out;
295
296       // "Invariant" is by how much the dot product of amounts increased due to fees.
297       let new_invariant =
298           integer_sqrt(U256::from(self.amounts[in_idx]) * U256::from(self.amounts[out_idx]));
299
300       // Invariant can not reduce (otherwise loosing balance of the pool and something it broken)
              .
301       assert!(new_invariant >= prev_invariant, "{}", ERR75_INVARIANT_REDUCE);
302       let numerator = (new_invariant - prev_invariant) * U256::from(self.shares_total_supply);
303
```

```
304         // Allocate exchange fee as fraction of total fee by issuing LP shares proportionally.
305         if admin_fee.exchange_fee > 0 && numerator > U256::zero() {
306             let denominator = new_invariant * FEE_DIVISOR / admin_fee.exchange_fee;
307             self.mint_shares(&admin_fee.exchange_id, (numerator / denominator).as_u128());
308         }
309
310         // If there is referral provided and the account already registered LP, allocate it % of LP
                rewards.
311         if let Some(referral_id) = &admin_fee.referral_id {
312             if admin_fee.referral_fee > 0
313                 && numerator > U256::zero()
314                 && self.shares.contains_key(referral_id)
315             {
316                 let denominator = new_invariant * FEE_DIVISOR / admin_fee.referral_fee;
317                 self.mint_shares(referral_id, (numerator / denominator).as_u128());
318             }
319         }
320
321         // Keeping track of volume per each input traded separately.
322         // Reported volume with fees will be sum of input, without fees will be sum of output.
323         self.volumes[in_idx].input.0 += amount_in;
324         self.volumes[in_idx].output.0 += amount_out;
325
326         amount_out
327     }
```

Listing 2.4: src/simple_pool.rs

Impact Simple Pool will always charge less admin fees than expected.

Suggestion I Use the equation listed above to calculate the shares minted for admins.

Note After the patch, the fee mechanism among `Simple Pool`, `Stable Pool`, and `Rated Pool` are consistent. However, the actual admin fee rate is higher than the `admin_fee_bps` configured in `AdminFees`. The reason is that the `LP` fees are distributed to all shares in the pool including the newly minted shares for admins.

2.3 Additional Recommendation

2.3.1 Lack of Check on Guardians' Removal

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `owner` of the protocol can remove `guardians` via the function `remove_guardians()`. However, the existence of `guardians` is not checked. In this case, if the `guardians` do not exist, the program will not panic, which may mislead the `owner` and bring unexpected impact.

```
64     #[payable]
65     pub fn remove_guardians(&mut self, guardians: Vec<ValidAccountId>) {
66         assert_one_yocto();
```

```
67        self.assert_owner();
68        for guardian in guardians {
69            self.guardians.remove(guardian.as_ref());
70        }
71    }
```

Listing 2.5: src/owner.rs

Suggestion I Check the return value of function `remove_guardians()`.

Feedback from the Project Will fix it in the next accumulated contract upgrade.

2.3.2 Two-Step Transfer of Privileged Account Ownership

Status Confirmed

Introduced by `Version 1`

Description The contract uses `set_owner()` to configure the privileged account, which can conduct many sensitive operations (e.g., retrieve unmanaged tokens). In this case, when an incorrect new owner is provided, the contract is under the risk of attack and the privileged functions cannot be invoked.

```
14    #[payable]
15    pub fn set_owner(&mut self, owner_id: ValidAccountId) {
16        assert_one_yocto();
17        self.assert_owner();
18        self.owner_id = owner_id.as_ref().clone();
19    }
```

Listing 2.6: src/owner.rs

Suggestion I Implement a two-step approach for the `owner` update: `set_owner()` and `commit_owner()`.

Feedback from the Project To prevent human unintentional errors during the ownership transfer, we would have a safety design to ensure the next owner exists and is able to perform his duty (sign TX). For that purpose, we may leverage a relay baton process: Grant (by cur owner with a deadline blockheight or timestamp), Accept (by next owner to ensure he can sign TX within the deadline), Confirm (by cur owner and followed by the real ownership transfer) or Cancel (by cur owner)

2.3.3 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, and so forth. In the current implementation of protocol, 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 a small number of tokens will be burned by the token smart contract.

This inconsistency can lead to security impacts for the operations based on the transferred amount of tokens.

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

2.3.4 Improper Check on the Admin Fees

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the process of swapping, the user has to pay three different fees for the service, i.e., `exchange_fee`, `referral_fee`, and "`lp_fee`". The admin fee (i.e., `exchange_fee` and `referral_fee`) is adjusted with the function `modify_admin_fee()`. However, the maximum admin fee (the sum of `exchange_fee` and `referral_fee`) is allowed to be set as `FEE_DIVISOR` (i.e., 100%), which means all the fees collected from the user are kept to admin. In this case, the liquidity provider cannot get any profit, which is unfair.

```
137  #[payable]
138   pub fn modify_admin_fee(&mut self, exchange_fee: u32, referral_fee: u32) {
139       assert_one_yocto();
140       self.assert_owner();
141       assert!(exchange_fee + referral_fee <= FEE_DIVISOR, "{}", ERR101_ILLEGAL_FEE);
142       self.exchange_fee = exchange_fee;
143       self.referral_fee = referral_fee;
144   }
```

Listing 2.7: src/owner.rs

Suggestion I It is recommended to limit the sum up of `exchange_fee` + `referral_fee` with a reasonable value, which is less than `FEE_DIVISOR`.

Feedback from the Project Will fix it in the next accumulated contract upgrade.

2.3.5 Lack of Check in retrieve_unmanaged_token()

Status Confirmed

Introduced by `Version 1`

Description Function `retrieve_unmanaged_token()` enables the `owner` to transfer `NEP-141` tokens from the contract to the `owner`. The purpose is to retrieve the tokens accidentally transferred in by others. However, there is no limitation on the amount of tokens that are transferred out. In this case, users' assets may lose if the owner transfers more tokens than expected.

```
29 #[payable]
30   pub fn retrieve_unmanaged_token(&mut self, token_id: ValidAccountId, amount: U128) -> Promise
         {
31       self.assert_owner();
32       assert_one_yocto();
```

```
33      let token_id: AccountId = token_id.into();
34      let amount: u128 = amount.into();
35      assert!(amount > 0, "{}", ERR29_ILLEGAL_WITHDRAW_AMOUNT);
36      env::log(
37          format!(
38              "Going to retrieve token {} to owner, amount: {}",
39              &token_id, amount
40          )
41          .as_bytes(),
42      );
43      ext_fungible_token::ft_transfer(
44          self.owner_id.clone(),
45          U128(amount),
46          None,
47          &token_id,
48          1,
49          env::prepaid_gas() - GAS_FOR_BASIC_OP,
50      )
51  }
```

Listing 2.8: src/owner.rs

Suggestion I It is recommended to add the check to ensure the user's assets would not be transferred out.

Feedback from the Project Current safety policy includes two points: First, we only grant that sensitive interface to the owner's management, and he (the DAO) would be careful with the numbers according to relevant transfer TX. Second, this interface can only withdraw tokens to the owner's account, which gives the owner (the DAO) 2ed chance to inspect numbers.

2.3.6 Lack of Check on the Gas Used by migrate()

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There is no check on whether the `attached_gas` is enough for function `migrate()`.

```
309 #[no_mangle]
310 pub extern "C" fn upgrade() {
311     env::setup_panic_hook();
312     env::set_blockchain_interface(Box::new(near_blockchain::NearBlockchain {}));
313     let contract: Contract = env::state_read().expect(ERR103_NOT_INITIALIZED);
314     contract.assert_owner();
315     let current_id = env::current_account_id().into_bytes();
316     let method_name = "migrate".as_bytes().to_vec();
317     unsafe {
318         BLOCKCHAIN_INTERFACE.with(|b| {
319             // Load input into register 0.
320             b.borrow()
321                 .as_ref()
322                 .expect(BLOCKCHAIN_INTERFACE_NOT_SET_ERR)
323                 .input(0);
324             let promise_id = b
```

```
325                 .borrow()
326                 .as_ref()
327                 .expect(BLOCKCHAIN_INTERFACE_NOT_SET_ERR)
328                 .promise_batch_create(current_id.len() as _, current_id.as_ptr() as _);
329             b.borrow()
330                 .as_ref()
331                 .expect(BLOCKCHAIN_INTERFACE_NOT_SET_ERR)
332                 .promise_batch_action_deploy_contract(promise_id, u64::MAX as _, 0);
333             let attached_gas = env::prepaid_gas() - env::used_gas() - GAS_FOR_MIGRATE_CALL;
334             b.borrow()
335                 .as_ref()
336                 .expect(BLOCKCHAIN_INTERFACE_NOT_SET_ERR)
337                 .promise_batch_action_function_call(
338                     promise_id,
339                     method_name.len() as _,
340                     method_name.as_ptr() as _,
341                     0 as _,
342                     0 as _,
343                     0 as _,
344                     attached_gas,
345                 );
346         });
347     }
348 }
```

Listing 2.9: src/owner.rs

Suggestion I Check whether the `attached_gas` is larger than a specified value.

Feedback from the Project Will fix it in the next accumulated contract upgrade.

2.3.7 Code Optimization (I)

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `internal_unwrap_or_default_account()` is used to get the stored `Account` in the contract with the `AccountId`. If the `AccountId` is not registered, the function will return a default `Account`. This function is improperly used in the functions listed below (i.e., `add_liquidity()`, `add_stable_liquidity()`, `remove_liquidity()`, and `remove_liquidity_by_tokens()`). Take the function `add_liquidity()` as an example, if the `Account` of the sender doesn't exist (line 266), the withdrawal of the deposited tokens in the newly created `Account` (lines 269 - 271) will always fail.

```
237 #[payable]
238 pub fn add_liquidity(
239     &mut self,
240     pool_id: u64,
241     amounts: Vec<U128>,
242     min_amounts: Option<Vec<U128>>,
243 ) -> U128 {
244     self.assert_contract_running();
```

```
245         assert!(
246             env::attached_deposit() > 0,
247             "{}", ERR35_AT_LEAST_ONE_YOCTO
248         );
249         let prev_storage = env::storage_usage();
250         let sender_id = env::predecessor_account_id();
251         let mut amounts: Vec<u128> = amounts.into_iter().map(|amount| amount.into()).collect();
252         let mut pool = self.pools.get(pool_id).expect(ERR85_NO_POOL);
253         // feature frozenlist
254         self.assert_no_frozen_tokens(pool.tokens());
255         // Add amounts given to liquidity first. It will return the balanced amounts.
256         let shares = pool.add_liquidity(
257             &sender_id,
258             &mut amounts,
259         );
260         if let Some(min_amounts) = min_amounts {
261             // Check that all amounts are above request min amounts in case of front running that
                    changes the exchange rate.
262             for (amount, min_amount) in amounts.iter().zip(min_amounts.iter()) {
263                 assert!(amount >= &min_amount.0, "{}", ERR86_MIN_AMOUNT);
264             }
265         }
266         let mut deposits = self.internal_unwrap_or_default_account(&sender_id);
267         let tokens = pool.tokens();
268         // Subtract updated amounts from deposits. This will fail if there is not enough funds for
                any of the tokens.
269         for i in 0..tokens.len() {
270             deposits.withdraw(&tokens[i], amounts[i]);
271         }
272         self.internal_save_account(&sender_id, deposits);
273         self.pools.replace(pool_id, &pool);
274         self.internal_check_storage(prev_storage);
275 
276         U128(shares)
277     }
```

Listing 2.10: src/lib.rs

```
284     #[payable]
285     pub fn add_stable_liquidity(
286         &mut self,
287         pool_id: u64,
288         amounts: Vec<U128>,
289         min_shares: U128,
290     ) -> U128 {
291         self.assert_contract_running();
292         assert!(
293             env::attached_deposit() > 0,
294             "{}", ERR35_AT_LEAST_ONE_YOCTO
295         );
296         let prev_storage = env::storage_usage();
297         let sender_id = env::predecessor_account_id();
298         let amounts: Vec<u128> = amounts.into_iter().map(|amount| amount.into()).collect();
```

```
299         let mut pool = self.pools.get(pool_id).expect(ERR85_NO_POOL);
300         // feature frozenlist
301         self.assert_no_frozen_tokens(pool.tokens());
302         // Add amounts given to liquidity first. It will return the balanced amounts.
303         let mint_shares = pool.add_stable_liquidity(
304             &sender_id,
305             &amounts,
306             min_shares.into(),
307             AdminFees::new(self.exchange_fee),
308         );
309         let mut deposits = self.internal_unwrap_or_default_account(&sender_id);
310         let tokens = pool.tokens();
311         // Subtract amounts from deposits. This will fail if there is not enough funds for any of
                the tokens.
312         for i in 0..tokens.len() {
313             deposits.withdraw(&tokens[i], amounts[i]);
314         }
315         self.internal_save_account(&sender_id, deposits);
316         self.pools.replace(pool_id, &pool);
317         self.internal_check_storage(prev_storage);
318
319         mint_shares.into()
320     }
```

Listing 2.11: src/lib.rs

```
333     #[payable]
334     pub fn remove_liquidity(&mut self, pool_id: u64, shares: U128, min_amounts: Vec<U128>) -> Vec<
            U128> {
335         assert_one_yocto();
336         self.assert_contract_running();
337         let prev_storage = env::storage_usage();
338         let sender_id = env::predecessor_account_id();
339         let mut pool = self.pools.get(pool_id).expect(ERR85_NO_POOL);
340         // feature frozenlist
341         self.assert_no_frozen_tokens(pool.tokens());
342         let amounts = pool.remove_liquidity(
343             &sender_id,
344             shares.into(),
345             min_amounts
346                 .into_iter()
347                 .map(|amount| amount.into())
348                 .collect(),
349         );
350         self.pools.replace(pool_id, &pool);
351         let tokens = pool.tokens();
352         let mut deposits = self.internal_unwrap_or_default_account(&sender_id);
353         for i in 0..tokens.len() {
354             deposits.deposit(&tokens[i], amounts[i]);
355         }
356         // Freed up storage balance from LP tokens will be returned to near_balance.
357         if prev_storage > env::storage_usage() {
358             deposits.near_amount +=
```

```
359                 (prev_storage - env::storage_usage()) as Balance * env::storage_byte_cost();
360         }
361         self.internal_save_account(&sender_id, deposits);
362
363         amounts
364             .into_iter()
365             .map(|amount| amount.into())
366             .collect()
367     }
```

Listing 2.12: src/lib.rs

```
373     #[payable]
374     pub fn remove_liquidity_by_tokens(
375         &mut self, pool_id: u64,
376         amounts: Vec<U128>,
377         max_burn_shares: U128
378     ) -> U128 {
379         assert_one_yocto();
380         self.assert_contract_running();
381         let prev_storage = env::storage_usage();
382         let sender_id = env::predecessor_account_id();
383         let mut pool = self.pools.get(pool_id).expect(ERR85_NO_POOL);
384         // feature frozenlist
385         self.assert_no_frozen_tokens(pool.tokens());
386         let burn_shares = pool.remove_liquidity_by_tokens(
387             &sender_id,
388             amounts
389                 .clone()
390                 .into_iter()
391                 .map(|amount| amount.into())
392                 .collect(),
393             max_burn_shares.into(),
394             AdminFees::new(self.exchange_fee),
395         );
396         self.pools.replace(pool_id, &pool);
397         let tokens = pool.tokens();
398         let mut deposits = self.internal_unwrap_or_default_account(&sender_id);
399         for i in 0..tokens.len() {
400             deposits.deposit(&tokens[i], amounts[i].into());
401         }
402         // Freed up storage balance from LP tokens will be returned to near_balance.
403         if prev_storage > env::storage_usage() {
404             deposits.near_amount +=
405                 (prev_storage - env::storage_usage()) as Balance * env::storage_byte_cost();
406         }
407         self.internal_save_account(&sender_id, deposits);
408
409         burn_shares.into()
410     }
```

Listing 2.13: src/lib.rs

Suggestion I Replace the function `internal_unwrap_or_default_account()` with the function `internal_unwrap_account()` in above functions.

Feedback from the Project Will fix it in the next accumulated contract upgrade.

2.3.8 Code Optimization (II)

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `ft_on_transfer()` is a callback function which is used to receive tokens. It will check whether the token transferred in is frozen for both the operation of `deposit` and the operation of `swap`. There also exist checks in the operation of `swap` to make sure the token swapped out is not frozen as well. However, this check will be done for each `swap`. The problem comes when a sequence of swaps executes, and there is a frozen token in the middle of the sequence. In this case, the execution will not fail until it reaches the middle, and the gas is wasted for executing the previous swap actions.

```
62  #[allow(unreachable_code)]
63  fn ft_on_transfer(
64      &mut self,
65      sender_id: ValidAccountId,
66      amount: U128,
67      msg: String,
68  ) -> PromiseOrValue<U128> {
69      self.assert_contract_running();
70      let token_in = env::predecessor_account_id();
71      // feature frozenlist
72      self.assert_no_frozen_tokens(&[token_in.clone()]);
73      if msg.is_empty() {
74          // Simple deposit.
75          self.internal_deposit(sender_id.as_ref(), &token_in, amount.into());
76          PromiseOrValue::Value(U128(0))
77      } else {
78          // instant swap
79          let message =
80              serde_json::from_str::<TokenReceiverMessage>(&msg).expect(ERR28_WRONG_MSG_FORMAT);
81          match message {
82              TokenReceiverMessage::Execute {
83                  referral_id,
84                  actions,
85              } => {
86                  let referral_id = referral_id.map(|x| x.to_string());
87                  let out_amounts = self.internal_direct_actions(
88                      token_in,
89                      amount.0,
90                      referral_id,
91                      &actions,
92                  );
93                  for (token_out, amount_out) in out_amounts.into_iter() {
94                      self.internal_send_tokens(sender_id.as_ref(), &token_out, amount_out);
95                  }
```

```
96                  // Even if send tokens fails, we don't return funds back to sender.
97                  PromiseOrValue::Value(U128(0))
98              }
99          }
100     }
101 }
```

Listing 2.14: src/token_receiver.rs

Suggestion I Check all the tokens listed in `actions` before the swapping to make sure no frozen tokens exist.

Feedback from the Project Will fix it in the next accumulated contract upgrade.

2.3.9 Avoid Logging in View Functions

Status Fixed in `Version 4`

Introduced by `Version 3`

Description The function shown below will always emit the logs regardless of whether the `is_view` argument is `true` or `false`, which can lead to inaccuracies in off-chain statistics and analytics. Though logs emitted in most of the functions can be differentiated by originating from a view account or not, logs emitted in function `swap` cannot be differentiated between view and non-view usage.

File	Function
simple_pool.rs	`add_liquidity`
simple_pool.rs	`remove_liquidity`
simple_pool.rs	`swap`
rated_swap/mod.rs	`add_liquidity`
rated_swap/mod.rs	`remove_liquidity_by_shares`
rated_swap/mod.rs	`remove_liquidity_by_tokens`
rated_swap/mod.rs	`swap`
stable_swap/mod.rs	`add_liquidity`
stable_swap/mod.rs	`remove_liquidity_by_shares`
stable_swap/mod.rs	`remove_liquidity_by_tokens`
stable_swap/mod.rs	`swap`

Suggestion I Avoid emitting logs when `is_view` is true.

2.3.10 Slippage Protection in Function add_liquidity

Status Fixed* in `Version 4`

Introduced by `Version 3`

Description When providing liquidity to `simple pool`s via the operation `HotZap`, users can optionally specify the parameter `min_amounts` to control slippage. However, when providing liquidity to `stable pool`s or `rated pool`s, the function `add_liqudity()` requires users to provide a `mint_shares`.

Considering the consistency and risk of frontrunning, it's recommended to check that `min_amounts` has to be provided.

```
250   #[payable]
251   pub fn add_liquidity(
252       &mut self,
253       pool_id: u64,
254       amounts: Vec<U128>,
255       min_amounts: Option<Vec<U128>>,
256   ) -> U128 {
257       self.assert_contract_running();
258       assert!(env::attached_deposit() > 0, "{}", ERR35_AT_LEAST_ONE_YOCTO);
259       let prev_storage = env::storage_usage();
260       let sender_id = env::predecessor_account_id();
261       let mut amounts: Vec<u128> = amounts.into_iter().map(|amount| amount.into()).collect();
262       let mut pool = self.pools.get(pool_id).expect(ERR85_NO_POOL);
263       // feature frozenlist
264       self.assert_no_frozen_tokens(pool.tokens());
265       // Add amounts given to liquidity first. It will return the balanced amounts.
266       let shares = pool.add_liquidity(&sender_id, &mut amounts, false);
267       if let Some(min_amounts) = min_amounts {
268           // Check that all amounts are above request min amounts in case of front running that
                changes the exchange rate.
269           for (amount, min_amount) in amounts.iter().zip(min_amounts.iter()) {
270               assert!(amount >= &min_amount.0, "{}", ERR86_MIN_AMOUNT);
271           }
272       }
273       // [AUDITION_AMENDMENT] 2.3.7 Code Optimization (I)
274       let mut deposits = self.internal_unwrap_account(&sender_id);
275       let tokens = pool.tokens();
276       // Subtract updated amounts from deposits. This will fail if there is not enough funds for
            any of the tokens.
277       for i in 0..tokens.len() {
278           deposits.withdraw(&tokens[i], amounts[i]);
279       }
280       self.internal_save_account(&sender_id, deposits);
281       self.pools.replace(pool_id, &pool);
282       self.internal_check_storage(prev_storage);
283
284
285       U128(shares)
286   }
```

Listing 2.15: src/lib.rs

```
116   TokenReceiverMessage::HotZap {
117       referral_id,
118       hot_zap_actions,
119       add_liquidity_infos
120   } => {
121       assert!(hot_zap_actions.len() > 0 && add_liquidity_infos.len() > 0);
122       let sender_id: AccountId = sender_id.into();
123       let mut account = self.internal_unwrap_account(&sender_id);
124       let referral_id = referral_id.map(|x| x.to_string());
```

```
125         let out_amounts = self.internal_direct_actions(
126             token_in,
127             amount.0,
128             referral_id,
129             &hot_zap_actions,
130         );
131 
132         let mut token_cache = TokenCache::new();
133         for (out_token_id, out_amount) in out_amounts {
134             token_cache.add(&out_token_id, out_amount);
135         }
136 
137         for add_liquidity_info in add_liquidity_infos {
138             let mut pool = self.pools.get(add_liquidity_info.pool_id).expect(ERR85_NO_POOL);
139             let tokens_in_pool = match &pool {
140                 Pool::SimplePool(p) => p.token_account_ids.clone(),
141                 Pool::RatedSwapPool(p) => p.token_account_ids.clone(),
142                 Pool::StableSwapPool(p) => p.token_account_ids.clone(),
143             };
144 
145             let mut add_liquidity_amounts = add_liquidity_info.amounts.iter().map(|v| v.0).collect
                    ();
146 
147             match pool {
148                 Pool::SimplePool(_) => {
149                     pool.add_liquidity(
150                         &sender_id,
151                         &mut add_liquidity_amounts,
152                         false
153                     );
154                     if let Some(min_amounts) = add_liquidity_info.min_amounts {
155                         // Check that all amounts are above request min amounts in case of front
                                running that changes the exchange rate.
156                         for (amount, min_amount) in add_liquidity_amounts.iter().zip(min_amounts.
                                iter()) {
157                             assert!(amount >= &min_amount.0, "{}", ERR86_MIN_AMOUNT);
158                         }
159                     }
160                 },
161                 Pool::StableSwapPool(_) | Pool::RatedSwapPool(_) => {
162                     let min_shares = add_liquidity_info.min_shares.expect("Need input min_shares");
163                     pool.add_stable_liquidity(
164                         &sender_id,
165                         &add_liquidity_amounts,
166                         min_shares.into(),
167                         AdminFees::new(self.admin_fee_bps),
168                         false
169                     );
170                 }
171             };
172 
173             for (cost_token_id, cost_amount) in tokens_in_pool.iter().zip(add_liquidity_amounts.
                    into_iter()) {
```

```
174                 token_cache.sub(cost_token_id, cost_amount);
175             }
176
177             self.pools.replace(add_liquidity_info.pool_id, &pool);
178         }
179
180         for (remain_token_id, remain_amount) in token_cache.0.iter() {
181             account.deposit(remain_token_id, *remain_amount);
182         }
183
184         self.internal_save_account(&sender_id, account);
185
186         env::log(
187             format!(
188                 "HotZap remain internal account assets: {:?}",
189                 token_cache.0
190             )
191             .as_bytes(),
192         );
193
194         PromiseOrValue::Value(U128(0))
195     }
```

Listing 2.16: src/token_receiver.rs

Suggestion I Revise the logic accordingly.

Feedback from the Project The `add_liquidity` function has not been updated for compatibility.

2.3.11 Spelling Errors

Status Fixed in `Version 6`

Introduced by `Version 5`

Description There are some spelling errors in the code, as shown in the table below

File & Line	Spelling Error
src/utils.rs #line 162,165,166,167	pirce
src/sfrax_rate.rs #line 76,173	Oralce

Suggestion I Revise the corresponding typos.

2.3.12 Lack of Checks for Oracle Configuration

Status Fixed in `Version 6`

Introduced by `Version 5`

Description The functions `SfraxRate::update_extra_info()` and `SfraxRate::new()` configure the `frax` and `sfrax` oracle with `extra_info_string` provided by the `admin`, but there is no validation for the critical system variables such as `pyth_price_valid_duration_sec`, `maximum_recency_duration_sec`, and `maximum_staleness_duration_sec`.

```rust
pub struct PriceOracle {
    pub oracle_id: AccountId,
    pub base_contract_id: AccountId,
    /// The maximum number of seconds expected from the oracle price call.
    pub maximum_recency_duration_sec: u32,
    /// Maximum staleness duration of the price data timestamp.
    /// Because NEAR protocol doesn't implement the gas auction right now, the only reason to
    /// delay the price updates are due to the shard congestion.
    /// This parameter can be updated in the future by the owner.
    pub maximum_staleness_duration_sec: u32,
}
```

Listing 2.17: src/rated_swap/sfrax_rate.rs

```rust
pub struct PythOracle {
    pub oracle_id: AccountId,
    pub base_price_identifier: PriceIdentifier,
    pub rate_price_identifier: PriceIdentifier,
    /// The valid duration to pyth price in seconds.
    pub pyth_price_valid_duration_sec: u32,
}
```

Listing 2.18: src/rated_swap/sfrax_rate.rs

```rust
impl SfraxRate {
    pub fn new(contract_id: AccountId, extra_info_string: String) -> Self {
        let extra_info =
               near_sdk::serde_json::from_str::<SfraxExtraInfo>(&extra_info_string).expect(
                   ERR128_INVALID_EXTRA_INFO_MSG_FORMAT);
        Self {
            stored_rates: PRECISION,
            rates_updated_at: 0,
            contract_id,
            extra_info,
        }
    }

    pub fn update_extra_info(&mut self, extra_info_string: String) {
        let extra_info =
               near_sdk::serde_json::from_str::<SfraxExtraInfo>(&extra_info_string).expect(
                   ERR128_INVALID_EXTRA_INFO_MSG_FORMAT);
        self.extra_info = extra_info;
    }
}
```

Listing 2.19: src/rated_swap/sfrax_rate.rs

Suggestion I There should be a maximum value limit imposed on `PythOracle::pyth_price_valid_duration_sec` and `PriceOracle::maximum_staleness_duration_sec`.

2.4 Notes

2.4.1 Delayed Price in Rated Swap Pool

Status Confirmed
Introduced by `version 1`

Description Given the async nature of `NEAR` protocol, one transaction on the `NEAR` protocol may be executed in several blocks. The price of tokens in the `Rated Swap Pool` may not be the latest. Therefore, it should be noted that the token added to the `Rated Swap Pool` should be as stable as possible.

2.4.2 Timely Triggering update_token_rate()

Status Confirmed
Introduced by `version 1`

Description Function `update_token_rate()` is used to get the newest rates of tokens from the token contracts and update them in the contract for further use. It's important for the team to make sure that the function will be triggered by the team timely.

2.4.3 Sensitive Functions Managed by DAO

Status Confirmed
Introduced by `version 1`

Description Privileged functions in `Ref-Exchange` are controlled by `DAO` (i.e., `ref-finance.sputnik-dao.near`). The `DAO` has the privilege to configure system parameters, change the state of the contract (pause and unpause), upgrade the contract, etc. The community should manage the `DAO` carefully.

2.4.4 Reliability of Oracle

Status Confirmed
Introduced by `version 5`

Description The prices of the tokens `frax` and `sfrax` are supplied by two different external oracles, and the specific oracle to be used is determined and configured by the `admin` during registration and activation of the corresponding pool through the `extra_info` parameter. To ensure the normal operation of the `frax-sfrax` rated pool, it is essential to have reasonable prices. Therefore, a stable and reliable oracle is a necessary requirement in this case.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS