



BlockSec

Security Audit Report for PembRock Contracts

Date: June 27th, 2022

Version: 1.0

Contact: contact@blocksec.com

Contents

1	Introduction	1
1.1	About Target Contracts	1
1.2	Disclaimer	2
1.3	Procedure of Auditing	2
1.3.1	Software Security	2
1.3.2	DeFi Security	3
1.3.3	NFT Security	3
1.3.4	Additional Recommendation	3
1.4	Security Model	3
2	Findings	5
2.1	Software Security	5
2.1.1	Incorrect User Storage Assertion	5
2.1.2	Potential DoS Problem	6
2.1.3	Token's min_amount is not Configurable	7
2.1.4	Improper Check on Token's min_amount	8
2.2	DeFi Security	9
2.2.1	Tokens' last_accrue_time are not Maintained Correctly	9
2.2.2	Farm shares/value and Position farm_shares are not Properly Handled	10
2.2.3	Improper Withdrawn Failure Handling	12
2.2.4	Principal Token is not Accrued When Opening Position	13
2.2.5	Debt Token is not Accrued in Function calc_swap_action	14
2.2.6	Users may not Close the Position Permanently	15
2.3	Additional Recommendation	16
2.3.1	Potential Precision Loss	16
2.3.2	Missing Validations on the Contract Settings	17
2.3.3	Potential Unsupported SwapPool in Ref-exchange	19
2.3.4	Potential Elastic Supply Token Issue	19
2.3.5	Inconsistent Gas Reference for ft_transfer	19
2.3.6	Inconsistent Log Emission Pattern	20
2.3.7	Inconsistent Gas Reserved for Function call_transfer1/2	21
2.3.8	Inconsistent Comment and Code Implementation	23
2.3.9	Inconsistent Token Saving Procedure	23
2.3.10	Potential Centralization Problem	24
2.4	Additional Note	25
2.4.1	The Async Nature of NEAR Protocol	25

Report Manifest

Item	Description
Client	PembRock Finance
Target	PembRock Contracts

Version History

Version	Date	Description
1.0	June 27th, 2022	First Release

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

Chapter 1 Introduction

1.1 About Target Contracts

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

The repository that has been audited includes PembRock Contracts 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 code (in the following versions) to fix issues in the audit report.

Project	Version	Commit SHA
PembRock Contracts	Version 1	57b50ede59405e86980c7f28dff010d967158deb
	Version 2	b3db428471fc560ae28c96588eb5485e1d30cb89

Note that we did **NOT** audit all the code in the repository. The scope of this audit report **ONLY** include the following files under the directory **src**.

- `account.rs`
- `ft_callback.rs`
- `position_actions.rs`
- `reinvest_proxy.rs`
- `test_utils.rs`
- `views.rs`
- `callbacks.rs`
- `lib.rs`
- `position_callbacks.rs`
- `settings.rs`
- `token.rs`
- `events.rs`
- `owner.rs`
- `position_utils.rs`
- `storage.rs`
- `types.rs`
- `farm.rs`
- `position.rs`
- `ref_integration.rs`
- `storage_management.rs`
- `utils.rs`

¹<https://github.com/PembRock-Finance/contracts>

All the code in the other files, which are not mentioned above, of this repository is out of our audit scope.

1.2 Disclaimer

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

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

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

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

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

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation

- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

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

1.3.3 NFT Security

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

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style



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

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology 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.

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

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

³<https://cwe.mitre.org/>

Table 1.1: Vulnerability Severity Classification

Impact	High	High	Medium
	Low	Medium	Low
		High	Low
		Likelihood	

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

Chapter 2 Findings

In total, we find 10 potential issues in the smart contract. We also have 10 recommendations and 1 note, as follows:

- High Risk: 2
- Medium Risk: 4
- Low Risk: 4
- Recommendations: 10
- Notes: 1

ID	Severity	Description	Category	Status
1	Medium	<i>Incorrect User Storage Assertion</i>	Software Security	Fixed
2	Low	<i>Potential DoS Problem</i>	Software Security	Fixed
3	Low	<i>Token's min_amount is not Configurable</i>	Software Security	Fixed
4	Low	<i>Improper Check on Tokens' min_amount</i>	Software Security	Fixed
5	High	<i>Tokens' last_accrue_time are not Maintained Correctly</i>	DeFi Security	Fixed
6	High	<i>Farm shares/value and Position farm_shares are not Properly Handled</i>	DeFi Security	Fixed
7	Medium	<i>Improper Withdrawn Failure Handling</i>	DeFi Security	Fixed
8	Medium	<i>Principal Token is not Accrued When Opening Position</i>	DeFi Security	Fixed
9	Medium	<i>Debt Token is not Accrued in Function calc_swap_action</i>	DeFi Security	Fixed
10	Low	<i>Users may not Close the Position Permanently</i>	DeFi Security	Confirmed
11	-	<i>Potential Precision Loss</i>	Recommendation	Fixed
12	-	<i>Missing Validations on the Contract Settings</i>	Recommendation	Fixed
13	-	<i>Potential Unsupported SwapPool in Ref-exchange</i>	Recommendation	Confirmed
14	-	<i>Potential Elastic Supply Token Issue</i>	Recommendation	Confirmed
15	-	<i>Inconsistent Gas Reference for ft_transfer</i>	Recommendation	Fixed
16	-	<i>Inconsistent Log Emission Pattern</i>	Recommendation	Fixed
17	-	<i>Inconsistent Gas Reserved for Function call_transfer1/2</i>	Recommendation	Fixed
18	-	<i>Inconsistent Comment and Code Implementation</i>	Recommendation	Fixed
19	-	<i>Inconsistent Token Saving Procedure</i>	Recommendation	Fixed
20	-	<i>Potential Centralization Problem</i>	Recommendation	Confirmed
21	-	<i>The Async Nature of NEAR Protocol</i>	Note	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Incorrect User Storage Assertion

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description The user's total `storage_usage()` is accumulated twice in function `assert_requested_storage()` (line 173 and 177).

```
170    /// Returns 'Self' if enough storage to store additional requested amount, otherwise panics
171    pub(crate) fn assert_requested_storage(self, requested_storage_amount: StorageUsage) -> Self {
```



```
172     let requested_storage_usage_in_bytes = self
173         .storage_usage()
174         .checked_add(requested_storage_amount as Balance * env::storage_byte_cost())
175         .unwrap_or_else(|| env::panic_str("Requested too much storage"));
176
177     if self.storage_stake < self.storage_usage() + requested_storage_usage_in_bytes {
178         env::panic_str("Insufficient $NEAR storage deposit")
179     }
180
181     self
182 }
```

Listing 2.1: src/account.rs

Impact Users with enough `storage_stake` may not be able to `open_position` due to the incorrect user storage calculation.

Suggestion I Remove the redundant `self.storage_usage()` accumulation in line 177.

2.1.2 Potential DoS Problem

Status Fixed in `version 2`

Introduced by `version 1`

Description When changing the contract owner by invoking function `set_owner()`, it does not check whether the new `owner_id` is registered.

```
12  /// Change owner. Only can be called by owner.
13  #[payable]
14  pub fn set_owner(&mut self, owner_id: AccountId) {
15      assert_one_yocto();
16
17      self.assert_owner();
18
19      NewOwner {
20          old_owner: &self.owner_id,
21          new_owner: &owner_id,
22      }
23      .emit();
24
25      self.owner_id = owner_id;
26  }
```

Listing 2.2: src/owner.rs

Impact The contract may be locked when invoking the functions (e.g., `internal_accrue_token`) that have the owner participated in.

```
223 // accrue pending interest on a token
224 fn internal_accrue_token(&mut self, token_id: &TokenId) {
225     let now = env::block_timestamp();
226     // TODO: panic on overflow
227     let time_passed: u64 = match now.checked_sub(self.last_accrue_time) {
228         Some(t) if t != 0 => t,
```

```
229     _ => return,
230 };
231
232 let mut token: Token = self.try_get_token(token_id).into();
233
234 let mut owner: Account = self.try_get_account(&self.owner_id).into();
235
236 let interest = token.get_pending_interest(time_passed);
237 let fee = interest * u128::from(self.settings.borrow_fee) / BPS_DIVISOR as u128;
238 ...
```

Listing 2.3: src/lib.rs

Suggestion I Throw into a `panic` if the new owner has not be registered.

Suggestion II Register the new owner's account in function `set_owner()` if it has not been registered.

2.1.3 Token's `min_amount` is not Configurable

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description The `token.min_amount` sets the lower bound on the principal/debt tokens when opening a new position.

```
28 pub struct Token {
29     #[serde(deserialize_with = "parse_u128_str")]
30     pub(crate) lend_pool_value: u128,
31     #[serde(deserialize_with = "parse_u128_str")]
32     pub(crate) lend_pool_shares: u128,
33     #[serde(deserialize_with = "parse_u128_str")]
34     pub(crate) debt_pool_value: u128,
35     #[serde(deserialize_with = "parse_u128_str")]
36     pub(crate) debt_pool_shares: u128,
37     #[serde(deserialize_with = "parse_u128_str")]
38     pub(crate) min_amount: u128,
39 }
```

Listing 2.4: src/token.rs

However, it is set as zero by default in function `owner_add_tokens` (line 88) and the owner cannot change the token's `min_amount` elsewhere.

```
76// This function adds a token available for lending (only owner)
77//
78// Arguments:
79// * 'token_id': token id
80#[payable]
81pub fn owner_add_tokens(&mut self, token_ids: Vec<TokenId>) {
82    assert_one_yocto();
83
84    self.assert_owner();
85
86    token_ids.iter().for_each(|token_id| {
87        if self.tokens.get(token_id).is_none() {
```

```
88         self.tokens.insert(token_id, &Token::default().into());
89     }
90 });
91 }
```

Listing 2.5: src/owner.rs

Impact The `token.min_amount` is meaningless if its value remains zero and cannot be changed.

Suggestion I Initialize the newly added token with a reasonable value of `min_amount`.

Suggestion II Add an only-owner function for token's configuration on the value of `min_amount`.

2.1.4 Improper Check on Token's `min_amount`

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description It does not make sense to throw into a `panic` when the `principal_amount` or the `debt_amount` is exactly equal to the `token.min_amount` (line 73 and line 95).

```
39 pub fn open_position(
40     &mut self,
41     farm_id: FarmId,
42     principal_amount: U128,
43     debt_amount: U128,
44     flags: PositionFlags,
45 ) -> PromiseOrValue<PositionId, bool> {
46     assert_ten_yocto();
47
48     self.assert_contract_running();
49
50     require!(
51         !flags.intersects(!PositionFlags::OPEN_FLAGS),
52         "Invalid flags"
53     );
54
55     let account_id = env::predecessor_account_id();
56     let mut account = Account::assert_requested_storage(
57         self.try_get_account(&account_id).into(),
58         CONTRACT_POSITION_STORAGE,
59     );
60
61     let farm: Farm = self.try_get_farm(farm_id).into();
62     require!(farm.enabled, "Farm disabled");
63
64     let principal_is_t1 = flags.contains(PositionFlags::PRINCIPAL_IS_T1);
65     let debt_is_t1 = flags.contains(PositionFlags::DEBT_IS_T1);
66
67     let principal_token_id = farm.get_token_id(principal_is_t1);
68     let debt_token_id = farm.get_token_id(debt_is_t1);
69
70     self.internal_accrue_token(&debt_token_id);
71
72     let mut principal_token: Token = self.try_get_token(&principal_token_id).into();
```

```
73     if principal_token.min_amount >= principal_amount.0 {
74         env::panic_str("Not enough principal token amount");
75     }
76
77     let (principal_amount, principal_shares) =
78         principal_token.sub_lend_pool_shares_by_value(principal_amount.0);
79
80     account.sub_lend_shares(&principal_token_id, principal_shares);
81
82     // determinates if debt token should be saved
83     let mut should_save_debt_token = false;
84
85     let mut debt_token = if principal_is_t1 == debt_is_t1 {
86         // principal token is the same as debt, ignore saving principal to save gas, mark debt
            token for later save
87         should_save_debt_token = true;
88         principal_token
89     } else {
90         self.save_token(&principal_token_id, principal_token);
91         self.try_get_token(&debt_token_id).into()
92     };
93
94     // debt_amount can be 0, in this case no need to check
95     if debt_amount.0 != 0 && debt_token.min_amount >= debt_amount.0 {
96         env::panic_str("Not enough debt token amount");
97     }
```

Listing 2.6: src/position_actions.rs

Impact Improper logic on checking the token's `min_amount`.

Suggestion I Change the comparison operators mentioned above from `">="` to `">"`.

2.2 DeFi Security

2.2.1 Tokens' last_accrue_time are not Maintained Correctly

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description All tokens involved in this contract are sharing the same `Contract.last_accrue_time` for calculating the value `time_passed` in function `internal_accrue_token` (line 227), resulting incorrect calculation for the pending interest.

```
223 // accrue pending interest on a token
224 fn internal_accrue_token(&mut self, token_id: &TokenId) {
225     let now = env::block_timestamp();
226     // TODO: panic on overflow
227     let time_passed: u64 = match now.checked_sub(self.last_accrue_time) {
228         Some(t) if t != 0 => t,
229         _ => return,
230     };
231 }
```

```
232     let mut token: Token = self.try_get_token(token_id).into();
233
234     let mut owner: Account = self.try_get_account(&self.owner_id).into();
235
236     let interest = token.get_pending_interest(time_passed);
237     let fee = interest * u128::from(self.settings.borrow_fee) / BPS_DIVISOR as u128;
238
239     // mint fee shares to owner account
240     let fee_shares = token.lend_value_to_shares(fee);
241     owner.add_lend_shares(token_id, fee_shares);
242     token.lend_pool_shares += fee_shares;
243
244     // add interest after minting fee shares
245     token.debt_pool_value += interest;
246
247     self.accounts
248         .insert(&self.owner_id, &VAccount::Current(owner));
249
250     self.tokens.insert(token_id, &token.into());
251     self.last_accrue_time = now;
252 }
```

Listing 2.7: src/lib.rs

Impact The logic of accruing the pending interest is wrong.

Suggestion I An attribute `last_accrue_time` needs to be maintained for each token in separate to calculate the pending interest.

2.2.2 Farm shares/value and Position farm_shares are not Properly Handled

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description The `Farm.shares`, `Farm.value` and the `Position.farm_shares` are increased or updated in function `on_mft_transfer_call` (line 316-319). However, those values are not reduced or updated in function `on_withdraw_seed()` or elsewhere in this contract.

```
302 pub(crate) fn on_mft_transfer_call(
303     &mut self,
304     position_id: PositionId,
305     unused_amount: U128,
306 ) -> PromiseOrValue<(PositionId, bool)> {
307     let mut position = Position::from(self.try_get_position(position_id)).unlock();
308     if unused_amount.0 == position.ref_shares {
309         env::log_str("All tokens refunded");
310         return PromiseOrValue::Value((position_id, false));
311     }
312     require!(unused_amount.0 == 0, "Not all tokens transferred");
313
314     let mut farm: Farm = self.try_get_farm(position.farm_id).into();
315     let farm_shares = farm.value_to_shares(position.ref_shares);
316     farm.shares += farm_shares;
317     farm.value += position.ref_shares;
```

```
318     self.farms.insert(&position.farm_id, &farm.into());
319     position.farm_shares = farm_shares;
320
321     position.next_state().unwrap_or_else(panic_str);
322
323     self.push_position_internal(position_id, position, None)
324 }
```

Listing 2.8: src/position_callbacks.rs

```
328 pub fn on_withdraw_seed(
329     &mut self,
330     position_id: PositionId,
331     ref_shares: U128,
332 ) -> PromiseOrValue<(PositionId, bool)> {
333     let mut position = Position::from(self.try_get_position(position_id)).unlock();
334     position.ref_shares = ref_shares.0;
335
336     position.next_state().unwrap_or_else(panic_str);
337
338     self.push_position_internal(position_id, position, None)
339 }
```

Listing 2.9: src/position_callbacks.rs

Impact The results (e.g., the `ref_shares` in function `call_withdraw_seed` in line 582) calculated based on the above values will be wrong.

```
572 fn call_withdraw_seed(
573     &mut self,
574     position_id: PositionId,
575     position: Position,
576 ) -> PromiseResult<(PositionId, bool)> {
577     let farm: Farm = self.try_get_farm(position.farm_id).into();
578     let seed_id = SeedId::new(
579         self.settings.ref_exchange_address.clone(),
580         Some(farm.ref_pool_id),
581     );
582     let ref_shares = farm.shares_to_value(position.farm_shares);
583
584     let reserved_gas = Gas(21_000_000_000_000);
585     let callback_gas = try_calculate_gas(
586         GAS_FOR_WITHDRAW_SEED,
587         GAS_FOR_CALLBACK_MINIMUM,
588         reserved_gas,
589     )?;
590
591     self.positions
592         .insert(&position_id, &VPosition::Current(position.lock()));
593
594     Ok(ext_farming::withdraw_seed(
595         seed_id,
596         U128(ref_shares),
597         self.settings.ref_farming_address.clone(),
```

```
598     ONE_YOCTO,  
599     GAS_FOR_WITHDRAW_SEED,  
600 )  
601 .then(ext_self_cp::withdraw_seed_callback(  
602     position_id,  
603     U128(ref_shares),  
604     env::current_account_id(),  
605     NO_DEPOSIT,  
606     callback_gas,  
607 ))  
608 .into())  
609 }
```

Listing 2.10: src/position_actions.rs

Suggestion I When the seed is withdrawn, remove the farm's `shares/value` and update the position's `farm_shares`.

2.2.3 Improper Withdrawn Failure Handling

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description Function `withdraw_callback` is used to recover the token's `lend_pool_shares/value` and the account's `lend_shares` when the cross-contract invocation `ft_transfer` executed in the previous block is checked as failed. However, there may exist transactions between the `ft_transfer` and `withdraw_callback`, which can change the ratio of the token's `lend_pool_shares` and `lend_pool_value`.

Therefore, it is unreasonable to add the old `lend_shares` back to the token's `lend_pool_shares` and the account's `lend_shares` in this callback function.

```
19  #[private]  
20  pub fn withdraw_callback(  
21      &mut self,  
22      account_id: AccountId,  
23      token_id: TokenId,  
24      shares: U128,  
25      value: U128,  
26  ) -> U128 {  
27      if is_promise_success() {  
28          Withdraw {  
29              account_id: &account_id,  
30              token_id: &token_id,  
31              shares,  
32              value,  
33          }  
34          .emit();  
35          return value;  
36      }  
37  
38      self.internal_accrue_token(&token_id);  
39  
40      // undo balance changes on fail
```

```
41     let mut token: Token = self.try_get_token(&token_id).into();
42     token.lend_pool_shares += shares.0;
43     token.lend_pool_value += value.0;
44     self.save_token(&token_id, token);
45
46     let mut account: Account = self.try_get_account(&account_id).into();
47     account.add_lend_shares(&token_id, shares.0);
48     self.accounts
49         .insert(&account_id, &VAccount::Current(account));
50
51     U128(0)
52 }
```

Listing 2.11: src/callbacks.rs

Impact Users and tokens may be recovered with incorrect lend shares.

Suggestion I Re-calculate the lend shares in the function `withdraw_callback`.

2.2.4 Principal Token is not Accrued When Opening Position

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description The pending interest of the `principal_token` is not accrued before invoking the function `sub_lend_pool_shares_by_value` (line 78) when opening a new position.

```
39 pub fn open_position(
40     &mut self,
41     farm_id: FarmId,
42     principal_amount: U128,
43     debt_amount: U128,
44     flags: PositionFlags,
45 ) -> PromiseOrValue<(PositionId, bool)> {
46     assert_ten_yocto();
47
48     self.assert_contract_running();
49
50     require!(
51         !flags.intersects(!PositionFlags::OPEN_FLAGS),
52         "Invalid flags"
53     );
54
55     let account_id = env::predecessor_account_id();
56     let mut account = Account::assert_requested_storage(
57         self.try_get_account(&account_id).into(),
58         CONTRACT_POSITION_STORAGE,
59     );
60
61     let farm: Farm = self.try_get_farm(farm_id).into();
62     require!(farm.enabled, "Farm disabled");
63
64     let principal_is_t1 = flags.contains(PositionFlags::PRINCIPAL_IS_T1);
65     let debt_is_t1 = flags.contains(PositionFlags::DEBT_IS_T1);
```



```
66
67     let principal_token_id = farm.get_token_id(principal_is_t1);
68     let debt_token_id = farm.get_token_id(debt_is_t1);
69
70     self.internal_accrue_token(&debt_token_id);
71
72     let mut principal_token: Token = self.try_get_token(&principal_token_id).into();
73     if principal_token.min_amount >= principal_amount.0 {
74         env::panic_str("Not enough principal token amount");
75     }
76
77     let (principal_amount, principal_shares) =
78         principal_token.sub_lend_pool_shares_by_value(principal_amount.0);
79
80     account.sub_lend_shares(&principal_token_id, principal_shares);
```

Listing 2.12: src/position_actions.rs

Impact Incorrect `principal_shares` may be subtracted from the tokens' and the accounts' total lending shares.

Suggestion I Accrue the `principal_token` before invoking the function `sub_lend_pool_shares_by_value`.

2.2.5 Debt Token is not Accrued in Function `calc_swap_action`

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description The pending interest of the `debt_token` is not accrued before the `debt_token.debt_shares_to_value` is invoked in line 759.

```
721 fn calc_swap_action(
722     &self,
723     farm: &Farm,
724     position: &Position,
725     pool_info: &RefPoolInfo,
726 ) -> Option<(SwapAction, Balance)> {
727     let (
728         debt_token_id,
729         non_debt_token_id,
730         debt_token_amount,
731         non_debt_token_amount,
732         reserve_in,
733         reserve_out,
734     ) = if position.flags.contains(PositionFlags::DEBT_IS_T1) {
735         (
736             farm.token1_id.clone(),
737             farm.token2_id.clone(),
738             position.token1_amount,
739             position.token2_amount,
740             pool_info.amounts[1].0,
741             pool_info.amounts[0].0,
742         )
743     }
```

```
743     } else {
744         (
745             farm.token2_id.clone(),
746             farm.token1_id.clone(),
747             position.token2_amount,
748             position.token1_amount,
749             pool_info.amounts[0].0,
750             pool_info.amounts[1].0,
751         )
752     };
753
754     let swap_amount = if position.flags.contains(PositionFlags::MINIMIZE_TRADING) {
755         // TODO: accrue
756
757         // Calculate debt value from position debt shares
758         let debt_token: Token = self.try_get_token(&debt_token_id).into();
759         let debt_value = debt_token.debt_shares_to_value(position.debt_shares);
760
761         // Take into account that debt_value could grow while closing position
762         // add 1% to debt value
763         // TODO: separate setting for over amount
764         let debt_over_value = calc_over_amount(self.settings.max_slippage, debt_value);
```

Listing 2.13: src/position_actions.rs

Impact Incorrect debt value may be calculated from the position debt shares.

Suggestion I Accrue the `debt_token` before the function `debt_shares_to_value` for the `debt_token` is invoked.

2.2.6 Users may not Close the Position Permanently

Status Confirmed

Introduced by `version 1`

Description The requirements in function `on_close_position` (line 960 and line 969) may not always be met. In this case, users can not close the position permanently.

```
922 pub(crate) fn on_close_position(
923     &mut self,
924     position_id: PositionId,
925     position: Position,
926 ) -> PromiseResult<(PositionId, bool)> {
927     let account_id = position.account_id;
928     let mut account: Account = self.try_get_account(&account_id).into();
929     let farm: Farm = self.try_get_farm(position.farm_id).into();
930
931     let (debt_token_id, non_debt_token_id, mut debt_token_amount, non_debt_token_amount) =
932         if position.flags.contains(PositionFlags::DEBT_IS_T1) {
933             (
934                 farm.token1_id,
935                 farm.token2_id,
936                 position.token1_amount,
937                 position.token2_amount,
```

```
938         )
939     } else {
940         (
941             farm.token2_id,
942             farm.token1_id,
943             position.token2_amount,
944             position.token1_amount,
945         )
946     };
947
948     if position.debt_shares != 0 || debt_token_amount != 0 {
949         self.internal_accrue_token(&debt_token_id);
950
951         let mut debt_token: Token = self.try_get_token(&debt_token_id).into();
952
953         // return all debt token amount to lend pool
954         debt_token.lend_pool_value += debt_token_amount;
955
956         // Repay debt & liquidation fee
957         if position.debt_shares != 0 {
958             // swap amount slippage takes into account debt value increase while cross-contract
959             // calls
960             let debt_value = debt_token.debt_shares_to_value(position.debt_shares);
961             require!(debt_value <= debt_token_amount, "Cannot repay debt");
962
963             // remove debt from debt pool
964             debt_token.debt_pool_value -= debt_value;
965             debt_token.debt_pool_shares -= position.debt_shares;
966             account.sub_debt_shares(&debt_token_id, position.debt_shares);
967
968             if position.flags.contains(PositionFlags::LIQUIDATION) {
969                 // During liquidation non_debt_token_amount should be equal 0
970                 require!(non_debt_token_amount == 0, "Wrong non debt token amount");
971
972                 let liq_fee = std::cmp::min(
973                     debt_token_amount - debt_value, // amount after debt repay
974                     debt_token_amount * self.settings.kill_fee as u128 / BPS_DIVISOR as u128,
```

Listing 2.14: src/position_actions.rs

Impact There is a potential DoS problem in function `on_close_position`.

Suggestion I Additional error handlers should be introduced.

Feedback from the Project This is intentional. Any panic in position flow callbacks (implicit or explicit) will lock the position. Locked positions will be investigated and fixed manually. This may change in the future.

2.3 Additional Recommendation

2.3.1 Potential Precision Loss

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description In line 90 of function `check_leverage`, division is performed before multiplication when calculating variable `debt_amount`, which may result in precision loss.

```
54 // check if leverage not exceeds max_leverage
55 // leverage = 1 + debt / principal
56 // leverage base = 1000 (10x = 10000; max_leverage >= 1000)
57 pub fn check_leverage(
58     max_leverage: u16,
59     principal_is_t1: bool,
60     debt_is_t1: bool,
61     principal_amount: Balance,
62     debt_amount: Balance,
63     reserve1: Balance,
64     reserve2: Balance,
65 ) -> bool {
66     let max_leverage = max_leverage as u128;
67
68     log!(
69         "check_leverage: max_leverage={}, principal_amount={}, debt_amount={}, reserves={}/{}",
70         max_leverage,
71         principal_amount,
72         debt_amount,
73         reserve1,
74         reserve2
75     );
76
77     if principal_is_t1 == debt_is_t1 {
78         return 1000 * debt_amount / principal_amount <= max_leverage - 1000;
79     }
80
81     let (principal_res, debt_res) = if principal_is_t1 {
82         (reserve1, reserve2)
83     } else {
84         (reserve2, reserve1)
85     };
86
87     // check if [debt / principal <= max_leverage - 1] with different reserves
88     // leverage base = 1000 (10x = 10000; max_leverage >= 1000)
89     // 1000 * (debt_amount / debt_res) / (principal_amount / principal_res) <= max_leverage -
90     // 1000
91     U256::from(debt_amount) * U256::from(principal_res) / principal_amount * U256::from(1000)
92     / U256::from(debt_res)
93     <= U256::from(max_leverage - 1000)
94 }
```

Listing 2.15: `src/utils.rs`

Suggestion I Modify this calculation to perform multiplication before division.

2.3.2 Missing Validations on the Contract Settings

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description When the owner configures the contract, the validation of the newly set value is not checked in these functions listed below.

```
51  /// Changes contract's borrow fee setting, panics if not an owner
52  #[payable]
53  pub fn owner_set_borrow_fee(&mut self, borrow_fee: Bps) {
54      assert_one_yocto();
55
56      self.assert_owner();
57
58      self.settings.borrow_fee = borrow_fee;
59  }
```

Listing 2.16: src/settings.rs

```
66  /// Changes contract's reinvest fee setting, panics if not an owner
67  #[payable]
68  pub fn owner_set_reinvest_fee(&mut self, reinvest_fee: Bps) {
69      assert_one_yocto();
70
71      self.assert_owner();
72
73      self.settings.reinvest_fee = reinvest_fee;
74  }
```

Listing 2.17: src/settings.rs

```
81  /// Changes contract's kill fee setting, panics if not an owner
82  #[payable]
83  pub fn owner_set_kill_fee(&mut self, kill_fee: Bps) {
84      assert_one_yocto();
85
86      self.assert_owner();
87
88      self.settings.kill_fee = kill_fee;
89  }
```

Listing 2.18: src/settings.rs

```
96  /// Changes contract's 'max_slippage' setting, panics if not an owner
97  #[payable]
98  pub fn owner_set_max_slippage(&mut self, max_slippage: Bps) {
99      assert_one_yocto();
100
101      self.assert_owner();
102
103      self.settings.max_slippage = max_slippage;
104  }
```

Listing 2.19: src/settings.rs

Suggestion I It is recommended to check that the `borrow_fee`, `reinvest_fee`, `kill_fee`, and `max_slippage` are the valid ones when invoking the functions listed above.

2.3.3 Potential Unsupported SwapPool in Ref-exchange

Status Confirmed

Introduced by `version 1`

Description Ref-finance supports several different kinds of pools including `stable_swap pool`, `rated_swap pool`, and the normal pools that utilize the constant product market maker model. This contract now only supports the pools utilizing the constant product market maker model.

```
169  #[payable]
170  pub fn owner_add_ref_pool(&mut self, ref_pool_id: u64) {
171      assert_one_yocto();
172
173      self.assert_owner();
174
175      self.ref_pool_whitelist.insert(&ref_pool_id);
176  }
177
178  #[payable]
179  pub fn owner_remove_ref_pool(&mut self, ref_pool_id: u64) {
180      assert_one_yocto();
181
182      self.assert_owner();
183
184      self.ref_pool_whitelist.remove(&ref_pool_id);
185  }
```

Listing 2.20: `src/owner.rs`

Suggestion I Do not add `stable_swap pool` or `rated_swap pool` to the `ref_pool_whitelist`.

2.3.4 Potential Elastic Supply Token Issue

Status Confirmed

Introduced by `version 1`

Description Elastic supply tokens (e.g., deflation tokens) could dynamically adjust the supply or user's balance. For example, if the token is a deflation token, there will be a difference between the transferred amount of tokens and the actual received amount of tokens.

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

Suggestion I Do not append the elastic supply tokens into the whitelist.

2.3.5 Inconsistent Gas Reference for `ft_transfer`

Status Fixed in `version 2`

Introduced by `version 1`

Description The value of `GAS_FOR_FT_TRANSFER_CALL` is `Gas(35_000_000_000_000 + 1)`, defined in `src/-ref_integration.rs`, which is associated with the calculation of `callback_gas`. However, the actual gas prepared for the cross-contract invocation `ft_transfer` is `GAS_FOR_FT_TRANSFER`, which is defined in `src/types.rs` with a different value of `Gas(10_000_000_000_000)`.

```
164 let reserved_gas = Gas(20_000_000_000_000);
165 let callback_gas = try_calculate_gas(
166     GAS_FOR_FT_TRANSFER_CALL,
167     GAS_FOR_CALLBACK_MINIMUM,
168     reserved_gas,
169 )
170 .unwrap_or_else(panic_str);
171
172 ext_fungible_token::ft_transfer(
173     account_id.clone(),
174     U128(withdraw_value),
175     None,
176     token_id.clone(),
177     ONE_YOCTO,
178     GAS_FOR_FT_TRANSFER,
179 )
180 .then(ext_self::withdraw_callback(
181     account_id,
182     token_id,
183     U128(withdraw_shares),
184     U128(withdraw_value),
185     env::current_account_id(),
186     NO_DEPOSIT,
187     callback_gas,
188 ))
```

Listing 2.21: src/lib.rs

Suggestion I Revise the code accordingly.

2.3.6 Inconsistent Log Emission Pattern

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description The event log for function `owner_change_state` is emitted by macro `log!` instead of the `NearEvent` pattern.

```
33 /// Change state of contract, Only can be called by owner or guardians.
34 #[payable]
35 pub fn owner_change_state(&mut self, state: RunningState) {
36     assert_one_yocto();
37
38     self.assert_owner_or_backend();
39
40     if self.running_state != state {
41         if state == RunningState::Running {
42             // only owner can resume the contract
43             self.assert_owner();
44         }
45         log!(
46             "Contract state changed from {} to {} by {}",
47             self.running_state,
```

```
48         state,
49         env::predecessor_account_id()
50     );
51     self.running_state = state;
52 }
53 }
```

Listing 2.22: src/owner.rs

Suggestion I It is recommended to emit a `NearEvent` in function `owner_change_state`, as is done in function `set_owner`.

```
12/// Change owner. Only can be called by owner (line 19-23).
13#[payable]
14pub fn set_owner(&mut self, owner_id: AccountId) {
15    assert_one_yocto();
16
17    self.assert_owner();
18
19    NewOwner {
20        old_owner: &self.owner_id,
21        new_owner: &owner_id,
22    }
23    .emit();
24
25    self.owner_id = owner_id;
26}
```

Listing 2.23: src/owner.rs

2.3.7 Inconsistent Gas Reserved for Function `call_transfer1/2`

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description The gas reserved for function `call_transfer1` (line 287) is inconsistent with another function `call_transfer2` (line 328);

```
274 fn call_transfer1(
275     &mut self,
276     position_id: PositionId,
277     position: Position,
278 ) -> PromiseResult<(PositionId, bool)> {
279     let token1_amount = position.token1_amount;
280
281     if token1_amount == 0 {
282         return Ok(self.on_transfer_1(position_id, 0.into()));
283     }
284
285     let farm: Farm = self.try_get_farm(position.farm_id).into();
286
287     let reserved_gas = Gas(25_000_000_000_000);
288     let callback_gas = try_calculate_gas(
```



```
289         GAS_FOR_FT_TRANSFER_CALL,
290         GAS_FOR_CALLBACK_MINIMUM,
291         reserved_gas,
292     )?;
293
294     self.positions
295         .insert(&position_id, &VPosition::Current(position.lock()));
296
297     Ok(ext_fungible_token::ft_transfer_call(
298         self.settings.ref_exchange_address.clone(),
299         U128(token1_amount),
300         None,
301         "".to_owned(),
302         farm.token1_id,
303         ONE_YOCTO,
304         GAS_FOR_FT_TRANSFER_CALL,
305     )
306     .then(ext_self_op::ft_1_callback(
307         position_id,
308         env::current_account_id(),
309         NO_DEPOSIT,
310         callback_gas,
311     ))
312     .into())
313 }
314
315 fn call_transfer2(
316     &mut self,
317     position_id: PositionId,
318     position: Position,
319 ) -> PromiseResult<(PositionId, bool)> {
320     let token2_amount = position.token2_amount;
321
322     if token2_amount == 0 {
323         return Ok(self.on_transfer_2(position_id, 0.into()));
324     }
325
326     let farm: Farm = self.try_get_farm(position.farm_id).into();
327
328     let reserved_gas = Gas(20_000_000_000_000);
329     let callback_gas = try_calculate_gas(
330         GAS_FOR_FT_TRANSFER_CALL,
331         GAS_FOR_CALLBACK_MINIMUM,
332         reserved_gas,
333     )?;
334
335     self.positions
336         .insert(&position_id, &VPosition::Current(position.lock()));
337
338     Ok(ext_fungible_token::ft_transfer_call(
339         self.settings.ref_exchange_address.clone(),
340         U128(token2_amount),
341         None,
```

```
342         ".to_owned(),
343         farm.token2_id,
344         ONE_YOCTO,
345         GAS_FOR_FT_TRANSFER_CALL,
346     )
347     .then(ext_self_op::ft_2_callback(
348         position_id,
349         env::current_account_id(),
350         NO_DEPOSIT,
351         callback_gas,
352     ))
353     .into()
354 }
```

Listing 2.24: src/position_actions.rs

Suggestion I It is recommended to unify the value of gas reserved for function `call_transfer1` and `call_transfer2`.

2.3.8 Inconsistent Comment and Code Implementation

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description There is no parameter named `swap_amount`, which is specified in the annotation (line 35), for function `open_position`.

```
29  /// Opens new position
30  ///
31  /// Arguments:
32  /// * 'farm_id': farm id
33  /// * 'principal_amount': coins amount to be withdrawn from user's lend pool
34  /// * 'debt_amount': tokens going to be borrowed
35  /// * 'swap_amount': amount of tokens we need to swap
36  ///
37  /// Returns position id
38  #[payable]
39  pub fn open_position(
40      &mut self,
41      farm_id: FarmId,
42      principal_amount: U128,
43      debt_amount: U128,
44      flags: PositionFlags,
45  ) -> PromiseOrValue<PositionId, bool> {
```

Listing 2.25: src/position_actions.rs

Suggestion I Revise the annotation.

2.3.9 Inconsistent Token Saving Procedure

Status Fixed in [version 2](#)

Introduced by [version 1](#)

Description If the principal token is not the same as the debt token when opening a position, the debt token will be finally saved in line 110 without checking that the token has been accrued before. Hence, the token saving procedures for `principal_token` and `debt_token` are inconsistent.

```
85 let mut debt_token = if principal_is_t1 == debt_is_t1 {
86     // principal token is the same as debt, ignore saving principal to save gas, mark debt
      token for later save
87     should_save_debt_token = true;
88     principal_token
89 } else {
90     self.save_token(&principal_token_id, principal_token);
91     self.try_get_token(&debt_token_id).into()
92 };
93
94 // debt_amount can be 0, in this case no need to check
95 if debt_amount.0 != 0 && debt_token.min_amount >= debt_amount.0 {
96     env::panic_str("Not enough debt token amount");
97 }
98
99 let debt_shares =
100     self.borrow_tokens(&mut account, &debt_token_id, &mut debt_token, debt_amount.0);
101
102 if debt_shares > 0 {
103     // some tokens borrowed, mark token to be saved
104     should_save_debt_token = true;
105 }
106
107 let debt_amount = debt_token.debt_shares_to_value(debt_shares);
108
109 if should_save_debt_token {
110     self.tokens.insert(&debt_token_id, &debt_token.into());
111 }
```

Listing 2.26: src/position_actions.rs

Suggestion I It is recommended to save the debt token by invoking function `save_token`.

2.3.10 Potential Centralization Problem

Status Confirmed

Introduced by `version 1`

Description This project has potential centralization problems. The project owner needs to ensure the security of the private key of the `Contract.owner_id` and use a multi-signature scheme to reduce the risk of single-point failure.

Suggestion I It is recommended to introduce a decentralization design in the contract, such as a multi-signature or a public DAO.

Feedback from the Project This smart contract was originally designed with the ability to manage it through a public DAO. Furthermore, contract design guarantees that the owner (DAO) does not have direct access to users' funds and has only limited editing options for positions stuck in the opening/closing process.

2.4 Additional Note

2.4.1 The Async Nature of NEAR Protocol

Status Confirmed

Introduced by [version 1](#)

Description Given the async nature of NEAR protocol, one transaction on NEAR protocol may be executed in several blocks.

For example, the `pool_info` retrieved from the external ref-exchange contract by invoking function `get_ref_pool_info` may only reflect the state of the ref-exchange pool in the last blocks ($N - x$ block, $x \geq 1$). Therefore, the `pool_info` may be outdated for functions like `check_leverage` which are executed in the N block.

```
394 pub(crate) fn call_swap(
395     &mut self,
396     position_id: PositionId,
397     position: Position,
398     pool_info: Option<RefPoolInfo>,
399 ) -> PromiseResult<(PositionId, bool)> {
400     let farm: Farm = self.try_get_farm(position.farm_id).into();
401
402     let pool_info = match pool_info {
403         Some(pool_info) => pool_info,
404         _ => return self.get_ref_pool_info(position_id, position, farm.ref_pool_id),
405     };
406
407     let farm: Farm = self.try_get_farm(position.farm_id).into();
408     let principal_is_t1 = position.flags.contains(PositionFlags::PRINCIPAL_IS_T1);
409     let debt_is_t1 = position.flags.contains(PositionFlags::DEBT_IS_T1);
410     let debt_token: Token = self
411         .try_get_token(if debt_is_t1 {
412             &farm.token1_id
413         } else {
414             &farm.token2_id
415         })
416         .into();
417     let debt_amount = debt_token.debt_shares_to_value(position.debt_shares);
418
419     require!(
420         check_leverage(
421             farm.max_leverage,
422             principal_is_t1,
423             debt_is_t1,
424             position.principal,
425             debt_amount,
426             pool_info.amounts[0].into(),
427             pool_info.amounts[1].into()
428         ),
429         "Over leverage"
430     );
431 }
```

```
432     let reserves = (pool_info.amounts[0].0, pool_info.amounts[1].0);
433
434     let tokens_amounts = (position.token1_amount, position.token2_amount);
435     let swap_in = optimal_deposit(tokens_amounts, reserves, pool_info.total_fee);
```

Listing 2.27: src/position_actions.rs