Security Audit Report for NEARx Smart Contract

Date: September 9, 2022

Version: 3.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Stader Labs
Target	NEARx Smart Contract

Version History

Version	Date	Description
1.0	August 8, 2022	First Release
2.0	August 17, 2022	Second Release
3.0	September 9, 2022	Third 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 NEARx Smart Contract [1].

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. Thus, there are multiple commit SHA values referred in this report. The commit SHA values before and after the audit are shown in the following.

Project		Commit SHA
NEARx Smart Contract	Version 1	89ee18b875adde6dd1a150c86f6d5a332d5e4404
	Version 2	7203ff19101de9ba4af0b1a91bd4f4bf187e8da5
	Version 3	108b4510305f8e4caf809b250db763f0b0e0f8a6
	Version 4	edbbdcb04af6de10cd96f5a66c5ed081a6a1151c

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

- contract/
  - empty_storage_spec.rs (moved to storage_spec.rs since Version 2)
  - internal.rs
  - metadata.rs
  - operator.rs
  - public.rs
  - upgrade.rs
  - util.rs
- fungible_token/
  - metadata.rs
  - nearx_internal.rs
  - nearx_token.rs
- constants.rs
- contract.rs
- errors.rs
- events.rs
- fungible_token.rs
- lib.rs
- state.rs
- utils.rs

[1] https://github.com/stader-labs/near-liquid-token

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 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.

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

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

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we find **ten** potential issues. We have **fifteen** recommendations and **one** notes.

- High Risk: 1
- Medium Risk: 2
- Low Risk: 7
- Recommendations: 15
- Notes: 1

ID	Severity	Description	Category	Status
1	Low	Incorrect Usage of u128	Software Security	Fixed
2	Low	Precision Loss Caused by Improper Rounding Implementation	Software Security	Confirmed
3	Low	Lack of Lower Bound for min_deposit_amount	Software Security	Fixed
4	Medium	Improper Value of MIN_BALANCE_FOR_STORAGE	Software Security	Fixed
5	Medium	Lack of Check on Account's Balance	Software Security	Fixed
6	Low	Mismatched Gas Assignment	Software Security	Fixed
7	Low	Rewards Fee can Exceed 10% due to Precision Loss	DeFi Security	Fixed
8	Low	User's Deposited Storage Fee May be Locked	DeFi Security	Fixed
9	High	Repeatable Refund of the Storage Fee	DeFi Security	Fixed
10	Low	User's Withdrawable Epoch Height can be Advanced	DeFi Security	Fixed
11	-	Improper Usage of saturating_sub	Recommendation	Fixed
12	-	Lack of Checking on Privileged Accounts	Recommendation	Fixed
13	-	Incomplete Checking on Privileged Accounts	Recommendation	Fixed
14	-	Improper Marco Usage	Recommendation	Confirmed
15	-	Potential Centralization Problem	Recommendation	Fixed
16	-	Unused Code (I)	Recommendation	Fixed
17	-	Unused Code (II)	Recommendation	Fixed
18	-	Redundant Code (I)	Recommendation	Confirmed
19	-	Redundant Code (II)	Recommendation	Confirmed
20	-	Redundant Code (III)	Recommendation	Confirmed
21	-	Redundant Code (IV)	Recommendation	Confirmed
22	-	Redundant Code (V)	Recommendation	Confirmed
23	-	Redundant Log Emissions	Recommendation	Confirmed
24	-	Improper Gas Value	Recommendation	Fixed
25	-	Improper Calculation of Operator Fee Shares	Recommendation	Confirmed
26	-	Only Staked Balance is Used to Calculate Rewards	Note	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Incorrect Usage of u128

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description JSON standard, which is used in NEAR protocol, can only work with primitive integer up to 53 bits. However, the value of the parameters (e.g., amount) or the returned values in the functions (e.g., set_min_deposit) can be larger than $2^{53} - 1$. In this case, overflow can occur.

```
315    #[payable]
316    pub fn set_min_deposit(&mut self, min_deposit: u128) {
317        self.assert_owner_calling();
318        assert_one_yocto();
319
320        require!(min_deposit < 100 * ONE_NEAR, ERROR_MIN_DEPOSIT_TOO_HIGH);
321
322        let old_min_deposit = self.min_deposit_amount;
323        self.min_deposit_amount = min_deposit;
324
325        Event::SetMinDeposit {
326            old_min_deposit: U128(old_min_deposit),
327            new_min_deposit: U128(self.min_deposit_amount),
328        }
329        .emit();
330    }
```

Listing 2.1: contracts/near-x/src/contract/public.rs

Impact Integer overflow.

Suggestion I Use `U128` instead of `u128` as U128 is a helper class for 128-bit integers provided by NEAR-SDK, which uses base-10 string.

2.1.2 Precision Loss Caused by Improper Rounding Implementation

Severity Low

Status Confirmed

Introduced by `Version 1`

Description Function `staked_amount_from_num_shares_rounded_up` is used when calculating `receive_amount`.

```
111    pub(crate) fn internal_unstake(&mut self, amount: u128) {
112        self.assert_unstaking_not_paused();
113
114        require!(amount > 0, ERROR_NON_POSITIVE_UNSTAKE_AMOUNT);
115
116        let account_id = env::predecessor_account_id();
117        let mut account = self.internal_get_account(&account_id);
118
119        require!(
```

```
            self.total_staked > 0,
            ERROR_NOT_ENOUGH_CONTRACT_STAKED_AMOUNT
        );

        let num_shares = self.num_shares_from_staked_amount_rounded_up(amount);
        require!(num_shares > 0, ERROR_NON_POSITIVE_UNSTAKING_SHARES);
        require!(
            account.stake_shares >= num_shares,
            ERROR_NOT_ENOUGH_STAKED_AMOUNT_TO_UNSTAKE
        );

        let receive_amount = self.staked_amount_from_num_shares_rounded_up(num_shares);
        require!(
            receive_amount > 0,
            ERROR_NON_POSITIVE_UNSTAKE_RECEVIE_AMOUNT
        );

        account.stake_shares -= num_shares;
        account.unstaked_amount += receive_amount;
        account.withdrawable_epoch_height =
            env::epoch_height() + self.get_unstake_release_epoch(amount);
        if self.last_reconcilation_epoch == env::epoch_height() {
            // The unstake request is received after epoch_reconcilation
            // so actual unstake will happen in the next epoch,
            // which will put withdraw off for one more epoch.
            account.withdrawable_epoch_height += 1;
        }

        self.internal_update_account(&account_id, &account);

        self.total_staked -= receive_amount;
        self.total_stake_shares -= num_shares;

        // Increase requested unstake amount within the current epoch
        self.user_amount_to_unstake_in_epoch += receive_amount;

        Event::Unstake {
            account_id: account_id.clone(),
            unstaked_amount: U128(amount),
            burnt_stake_shares: U128(num_shares),
            new_unstaked_balance: U128(account.unstaked_amount),
            new_stake_shares: U128(account.stake_shares),
            unstaked_available_epoch_height: account.withdrawable_epoch_height,
        }
        .emit();

        Event::FtBurn {
            account_id,
            amount: U128(num_shares),
        }
        .emit();
    }
```

Listing 2.2: contracts/near-x/src/contract/internal.rs

Impact Due to precision loss, the user may get extra NEARs, and the project may suffer a loss.

Suggestion I It is recommended to use `staked_amount_from_num_shares_rounded_down` to calculate `receive_amount`.

Feedback from the Project We wanted to give a similar interface so that we can integrate with external parties who are already integrated with staking pools.

2.1.3 Lack of Lower Bound for min_deposit_amount

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract doesn't require depositing storage fees. In this case, there should be a check for the lower bound of the `min_deposit_amount` in case it is rather small. Otherwise, the contract may be vulnerable to DoS attack as malicious users can run out of storage with low cost.

```
315     #[payable]
316     pub fn set_min_deposit(&mut self, min_deposit: u128) {
317         self.assert_owner_calling();
318         assert_one_yocto();
319
320         require!(min_deposit < 100 * ONE_NEAR, ERROR_MIN_DEPOSIT_TOO_HIGH);
321
322         let old_min_deposit = self.min_deposit_amount;
323         self.min_deposit_amount = min_deposit;
324
325         Event::SetMinDeposit {
326             old_min_deposit: U128(old_min_deposit),
327             new_min_deposit: U128(self.min_deposit_amount),
328         }
329         .emit();
330     }
```

Listing 2.3: contracts/near-x/src/contract/public.rs

Impact If the `min_deposit_amount` is rather small, the contract may be vulnerable to DoS attack.

Suggestion I Add a check for the lower bound of the `min_deposit_amount` in `set_min_deposit`.

2.1.4 Improper Value of MIN_BALANCE_FOR_STORAGE

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description `MIN_BALANCE_FOR_STORAGE` may not be enough to support the storage fee with the increasement of the users. In this case, `MIN_BALANCE_FOR_STORAGE` should not be a constant.

```
/// The contract keeps at least 50 NEAR in the account to avoid being transferred out to cover
/// contract code storage and some internal state.
pub const MIN_BALANCE_FOR_STORAGE: u128 = 50_000_000_000_000_000_000_000_000;
```

Listing 2.4: contracts/near-x/src/constants.rs

```
    // Make this return a promise
    pub(crate) fn internal_withdraw(&mut self, amount: Balance) {
        self.assert_withdraw_not_paused();

        let account_id = env::predecessor_account_id();

        require!(amount > 0, ERROR_NON_POSITIVE_WITHDRAWAL);

        let account = self.internal_get_account(&account_id);
        require!(
            account.unstaked_amount >= amount,
            ERROR_NOT_ENOUGH_UNSTAKED_AMOUNT_TO_WITHDRAW
        );
        require!(
            account.withdrawable_epoch_height <= env::epoch_height(),
            ERROR_UNSTAKED_AMOUNT_IN_UNBONDING_PERIOD
        );

        require!(
            env::account_balance().saturating_sub(MIN_BALANCE_FOR_STORAGE) >= amount,
            ERROR_NOT_ENOUGH_BALANCE_FOR_STORAGE
        );

        let mut account = self.internal_get_account(&account_id);
        account.unstaked_amount -= amount;
        self.internal_update_account(&account_id, &account);

        Event::Withdraw {
            account_id: account_id.clone(),
            amount: U128(amount),
            new_unstaked_balance: U128(account.unstaked_amount),
        }
        .emit();

        Promise::new(account_id).transfer(amount);
    }
```

Listing 2.5: contracts/near-x/src/contract/internal.rs

```
    // keep calling this method until false is return
    pub fn staking_epoch(&mut self) -> bool {
        self.assert_staking_epoch_not_paused();

        let min_gas = gas::STAKING_EPOCH
            + gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE
            + gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE_CB;
        require!(
```

```
22          env::prepaid_gas() >= min_gas,
23          format!("{}. require at least {:?}", ERROR_NOT_ENOUGH_GAS, min_gas)
24      );
25
26      self.epoch_reconcilation();
27      // after cleanup, there might be no need to stake
28      if self.reconciled_epoch_stake_amount == 0 {
29          log!("no need to stake, amount to settle is zero");
30          return false;
31      }
32
33      let validator_to_stake_info =
34          self.get_validator_to_stake(self.reconciled_epoch_stake_amount);
35      require!(
36          validator_to_stake_info.0.is_some(),
37          ERROR_NO_VALIDATOR_AVAILABLE_TO_STAKE
38      );
39
40      let validator = validator_to_stake_info.0.unwrap();
41
42      let amount_to_stake = validator_to_stake_info.1;
43
44      log!("amount to stake is {:?}", amount_to_stake);
45
46      require!(
47          env::account_balance() >= amount_to_stake + MIN_BALANCE_FOR_STORAGE,
48          ERROR_MIN_BALANCE_FOR_CONTRACT_STORAGE
49      );
50
51      // update internal state
52      self.reconciled_epoch_stake_amount = self
53          .reconciled_epoch_stake_amount
54          .saturating_sub(amount_to_stake);
55
56      // do staking on selected validator
57      ext_staking_pool::ext(validator.account_id.clone())
58          .with_attached_deposit(amount_to_stake)
59          .with_static_gas(gas::DEPOSIT_AND_STAKE)
60          .deposit_and_stake()
61          .then(
62              ext_staking_pool_callback::ext(env::current_account_id())
63                  .with_attached_deposit(NO_DEPOSIT)
64                  .with_static_gas(gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE)
65                  .on_stake_pool_deposit_and_stake(validator.account_id.clone(), amount_to_stake),
66          );
67
68      Event::StakingEpochAttempt {
69          validator_id: validator.account_id,
70          amount: U128(amount_to_stake),
71      }
72      .emit();
73
74      true
```

```
75    }
```

Listing 2.6: contracts/near-x/src/contract/operator.rs

Impact When the number of users increases, `MIN_BALANCE_FOR_STORAGE` may not be enough for the storage fee, resulting the denial of service of this contract.

Suggestion I Revise the value of `MIN_BALANCE_FOR_STORAGE`.

2.1.5 Lack of Check on Account's Balance

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When withdrawing, if the user does not withdraw all the balance, the user needs to keep a certain amount (i.e., `min_deposit_amount`) for the storage fee of the account. In current implementation, the user can retain a small balance (e.g. 1 yocto NEAR) and the account will not be removed, resulting in additional storage cost for the contract.

```
173    // Make this return a promise
174    pub(crate) fn internal_withdraw(&mut self, amount: Balance) {
175        self.assert_withdraw_not_paused();
176
177        let account_id = env::predecessor_account_id();
178
179        require!(amount > 0, ERROR_NON_POSITIVE_WITHDRAWAL);
180
181        let account = self.internal_get_account(&account_id);
182        require!(
183            account.unstaked_amount >= amount,
184            ERROR_NOT_ENOUGH_UNSTAKED_AMOUNT_TO_WITHDRAW
185        );
186        require!(
187            account.withdrawable_epoch_height <= env::epoch_height(),
188            ERROR_UNSTAKED_AMOUNT_IN_UNBONDING_PERIOD
189        );
190
191        require!(
192            env::account_balance().saturating_sub(MIN_BALANCE_FOR_STORAGE) >= amount,
193            ERROR_NOT_ENOUGH_BALANCE_FOR_STORAGE
194        );
195
196        let mut account = self.internal_get_account(&account_id);
197        account.unstaked_amount -= amount;
198        self.internal_update_account(&account_id, &account);
199
200        Event::Withdraw {
201            account_id: account_id.clone(),
202            amount: U128(amount),
203            new_unstaked_balance: U128(account.unstaked_amount),
204        }
```

```
205         .emit();
206
207         Promise::new(account_id).transfer(amount);
208     }
```

Listing 2.7: contracts/near-x/src/contract/internal.rs

```
272     pub(crate) fn internal_update_account(&mut self, account_id: &AccountId, account: &Account) {
273         if account.is_empty() {
274             self.accounts.remove(account_id);
275         } else {
276             self.accounts.insert(account_id, account); //insert_or_update
277         }
278     }
```

Listing 2.8: contracts/near-x/src/contract/internal.rs

Impact The contract may be vulnerable to DoS attack, malicious users can run out of storage with low cost.

Suggestion I Check whether the left balance is larger than a specified value (e.g.,`min_deposit_amount`) if not all the balance is withdrawn.

2.1.6 Mismatched Gas Assignment

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The gas prepaid to `deposit_and_stake` and `on_stake_pool_deposit_and_stake` are mismatched. The correct ones should be `gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE` and `gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE_CB`, respectively.

```
14     // keep calling this method until false is return
15     pub fn staking_epoch(&mut self) -> bool {
16         self.assert_staking_epoch_not_paused();
17
18         let min_gas = gas::STAKING_EPOCH
19             + gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE
20             + gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE_CB;
21         require!(
22             env::prepaid_gas() >= min_gas,
23             format!("{}. require at least {:?}", ERROR_NOT_ENOUGH_GAS, min_gas)
24         );
25
26         self.epoch_reconcilation();
27         // after cleanup, there might be no need to stake
28         if self.reconciled_epoch_stake_amount == 0 {
29             log!("no need to stake, amount to settle is zero");
30             return false;
31         }
32
33         let validator_to_stake_info =
```

```
34              self.get_validator_to_stake(self.reconciled_epoch_stake_amount);
35          require!(
36              validator_to_stake_info.0.is_some(),
37              ERROR_NO_VALIDATOR_AVAILABLE_TO_STAKE
38          );
39
40          let validator = validator_to_stake_info.0.unwrap();
41
42          let amount_to_stake = validator_to_stake_info.1;
43
44          log!("amount to stake is {:?}", amount_to_stake);
45
46          require!(
47              env::account_balance() >= amount_to_stake + MIN_BALANCE_FOR_STORAGE,
48              ERROR_MIN_BALANCE_FOR_CONTRACT_STORAGE
49          );
50
51          // update internal state
52          self.reconciled_epoch_stake_amount = self
53              .reconciled_epoch_stake_amount
54              .saturating_sub(amount_to_stake);
55
56          // do staking on selected validator
57          ext_staking_pool::ext(validator.account_id.clone())
58              .with_attached_deposit(amount_to_stake)
59              .with_static_gas(gas::DEPOSIT_AND_STAKE)
60              .deposit_and_stake()
61              .then(
62                  ext_staking_pool_callback::ext(env::current_account_id())
63                      .with_attached_deposit(NO_DEPOSIT)
64                      .with_static_gas(gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE)
65                      .on_stake_pool_deposit_and_stake(validator.account_id.clone(), amount_to_stake),
66              );
67
68          Event::StakingEpochAttempt {
69              validator_id: validator.account_id,
70              amount: U128(amount_to_stake),
71          }
72          .emit();
73
74          true
75      }
```

Listing 2.9: contracts/near-x/src/contract/operator.rs

Impact The gas fee assigned to cross-contract invocation may not be enough.

Suggestion I Set the gas prepaid to `deposit_and_stake` and `on_stake_pool_deposit_and_stake` to `gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE` and `gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE_CB`, respectively.

2.2 DeFi Security

2.2.1 Rewards Fee can Exceed 10% due to Precision Loss

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The reward fee is expected no more than 10%. However, if `numerator` is 1_099_999 and `denominator` is 10_000_000, `(numerator * 100 / denominator)` is 10, but the rewards fee is actually 10.99999%, which is more than 10%.

```
299     #[payable]
300     pub fn set_reward_fee(&mut self, numerator: u32, denominator: u32) {
301         self.assert_owner_calling();
302         assert_one_yocto();
303         require!((numerator * 100 / denominator) <= 10); // less than or equal to 10%
304
305         let old_reward_fee = self.rewards_fee;
306         self.rewards_fee = Fraction::new(numerator, denominator);
307
308         Event::SetRewardFee {
309             old_reward_fee,
310             new_reward_fee: self.rewards_fee,
311         }
312         .emit();
313     }
```

Listing 2.10: contracts/near-x/src/contract/public.rs

Impact The rewards fee can be more than 10%.

Suggestion I Change `require!((numerator * 100 / denominator) <= 10);` to `require!(numerator * 10 <= denominator);`.

2.2.2 User's Deposited Storage Fee May be Locked

Severity Low

Status Fixed in `Version 3`

Introduced by `Version 2`

Description If the account is checked as empty when invoking the function `internal_update_account`, the account will be removed without refunding its storage fee.

```
313     pub(crate) fn internal_update_account(&mut self, account_id: &AccountId, account: &Account) {
314         if account.is_empty() {
315             self.accounts.remove(account_id);
316         } else {
317             self.accounts.insert(account_id, account); //insert_or_update
318         }
319     }
```

Listing 2.11: contracts/near-x/src/contract/internal.rs

```
149     pub fn is_empty(&self) -> bool {
150         self.stake_shares == 0 && self.unstaked_amount == 0
151     }
```

Listing 2.12: contracts/near-x/src/state.rs

Impact User's deposited storage fee may be locked and cannot be withdrawn.

Suggestion I Return user's deposited storage fee when the account is removed.

2.2.3 Repeatable Refund of the Storage Fee

Severity High

Status Fixed in `Version 4`

Introduced by `Version 3`

Description Assume that there are two accounts (`user_A` and `contract_B`) registered in this contract. If `user_A` invokes the function `ft_transfer_call()` to transfer all of his NEARx to `contract_B`, then the cross-contract invocation `ft_on_transfer()` will be executed in the next block.

```
50      #[payable]
51      fn ft_transfer_call(
52          &mut self,
53          receiver_id: AccountId,
54          amount: U128,
55          #[allow(unused)] memo: Option<String>,
56          msg: String,
57      ) -> PromiseOrValue<U128> {
58          assert_one_yocto();
59          self.assert_ft_transfer_call_not_paused();
60          let min_gas = gas::FT_TRANSFER + gas::FT_TRANSFER_RESOLVE;
61          assert!(
62              env::prepaid_gas() > min_gas,
63              "require at least {:?} gas",
64              min_gas
65          );
66
67          Event::FtTransferCall {
68              receiver_id: receiver_id.clone(),
69              sender_id: env::predecessor_account_id(),
70              msg: msg.clone(),
71              amount,
72          }
73          .emit();
74
75          self.internal_nearx_transfer(&env::predecessor_account_id(), &receiver_id, amount.0);
76
77          ext_ft_receiver::ext(receiver_id.clone())
78              .with_attached_deposit(NO_DEPOSIT)
79              .with_static_gas(env::prepaid_gas() - gas::FT_TRANSFER - gas::FT_TRANSFER_RESOLVE)
80              .ft_on_transfer(env::predecessor_account_id(), amount, msg)
81              .then(
```

```
82              ext_self::ext(env::current_account_id())
83                  .with_attached_deposit(NO_DEPOSIT)
84                  .with_static_gas(gas::FT_TRANSFER_RESOLVE)
85                  .ft_resolve_transfer(env::predecessor_account_id(), receiver_id, amount),
86          )
87          .into()
88  }
```

Listing 2.13: contracts/near-x/src/fungible_token/nearx_token.rs

After that, `user_A` can unregister his account immediately at the same block as his account is empty,. The account's storage fee, which is about 0.0025 NEAR, will also be refunded to `user_A` in the next block.

```
95  #[payable]
96  fn storage_unregister(&mut self, force: Option<bool>) -> bool {
97      assert_one_yocto();
98
99      if let Some(f) = force {
100         if f {
101             panic!("We don't support force unregister");
102         }
103     }
104
105     let account_id = env::predecessor_account_id();
106
107     if self.accounts.get(&account_id).is_none() {
108         return false;
109     }
110
111     let account = self.internal_get_account(&account_id);
112
113     // if account registered check if amount staked and unstaked is 0
114     if account.is_empty() {
115         self.accounts.remove(&account_id);
116         Promise::new(account_id).transfer(self.storage_balance_bounds().min.0);
117     } else {
118         panic!("Account is not empty!");
119     }
120
121     true
122 }
```

Listing 2.14: contracts/near-x/src/contract/storage_spec.rs

However, if `contract_B` does not implement the function `ft_on_transfer`, the `promise_result` will be returned as failed, and the account of `user_A` will be recovered in the callback function `ft_resolve_transfer` without depositing any storage fee (lines 59-61). In this case, `user_A` can unregister himself again and get the refund of storage fee again.

```
29  pub fn int_ft_resolve_transfer(
30      &mut self,
31      sender_id: &AccountId,
32      receiver_id: AccountId,
33      amount: U128,
```

```
34    ) -> (u128, u128) {
35        let receiver_id = receiver_id;
36        let amount: Balance = amount.into();
37
38        // Get the unused amount from the ft_on_transfer call result.
39        let unused_amount = match env::promise_result(0) {
40            PromiseResult::NotReady => unreachable!(),
41            PromiseResult::Successful(value) => {
42                if let Ok(unused_amount) = near_sdk::serde_json::from_slice::<U128>(&value) {
43                    std::cmp::min(amount, unused_amount.0)
44                } else {
45                    amount
46                }
47            }
48            PromiseResult::Failed => amount,
49        };
50
51        if unused_amount > 0 {
52            let mut receiver_acc = self.internal_get_account(&receiver_id);
53            let receiver_balance = receiver_acc.stake_shares;
54            if receiver_balance > 0 {
55                let refund_amount = std::cmp::min(receiver_balance, unused_amount);
56                receiver_acc.stake_shares -= refund_amount;
57                self.internal_update_account(&receiver_id, &receiver_acc);
58
59                let mut sender_acc = self.internal_get_account(sender_id);
60                sender_acc.stake_shares += refund_amount;
61                self.internal_update_account(sender_id, &sender_acc);
62
63                log!(
64                    "Refund {} from {} to {}",
65                    refund_amount,
66                    receiver_id,
67                    sender_id
68                );
69                return (amount - refund_amount, 0);
70            }
71        }
72        (amount, 0)
73    }
```

Listing 2.15: contracts/near-x/src/fungible_token/nearx_internal.rs

Impact All NEARs in the contract can be withdrawn by repeating the above steps as described.

Suggestion I Ensure that the account `sender_id` is not allowed to be unregistered while the cross contract invocation `ft_on_transfer` and the callback function `ft_resolve_transfer` are not finished.

2.2.4 User's Withdrawable Epoch Height can be Advanced

Severity Low

Status Fixed in `Version 4`

Introduced by Version 1

Description The `account.withdrawable_epoch_height` indicates when the user's unstaked NEARs will be available for withdrawal. If there are not enough NEARs as requested in current available validators, the `account.withdrawable_epoch_height` will be set to 8 epochs later in function `internal_unstake` (line 139).

However, the user's withdrawable epoch height can be advanced by unstaking a smaller amount of NEARs again. In this case, the `account.unstaked_amount` (line 138) will be accumulated, but the user's withdrawal time is reset to only 4 epochs later (line 139).

```
111    pub(crate) fn internal_unstake(&mut self, amount: u128) {
112        self.assert_unstaking_not_paused();
113
114        require!(amount > 0, ERROR_NON_POSITIVE_UNSTAKE_AMOUNT);
115
116        let account_id = env::predecessor_account_id();
117        let mut account = self.internal_get_account(&account_id);
118
119        require!(
120            self.total_staked > 0,
121            ERROR_NOT_ENOUGH_CONTRACT_STAKED_AMOUNT
122        );
123
124        let num_shares = self.num_shares_from_staked_amount_rounded_up(amount);
125        require!(num_shares > 0, ERROR_NON_POSITIVE_UNSTAKING_SHARES);
126        require!(
127            account.stake_shares >= num_shares,
128            ERROR_NOT_ENOUGH_STAKED_AMOUNT_TO_UNSTAKE
129        );
130
131        let receive_amount = self.staked_amount_from_num_shares_rounded_up(num_shares);
132        require!(
133            receive_amount > 0,
134            ERROR_NON_POSITIVE_UNSTAKE_RECEVIE_AMOUNT
135        );
136
137        account.stake_shares -= num_shares;
138        account.unstaked_amount += receive_amount;
139        account.withdrawable_epoch_height =
140            env::epoch_height() + self.get_unstake_release_epoch(amount);
141        if self.last_reconcilation_epoch == env::epoch_height() {
142            // The unstake request is received after epoch_reconcilation
143            // so actual unstake will happen in the next epoch,
144            // which will put withdraw off for one more epoch.
145            account.withdrawable_epoch_height += 1;
146        }
147
148        self.internal_update_account(&account_id, &account);
149
150        self.total_staked -= receive_amount;
151        self.total_stake_shares -= num_shares;
152
153        // Increase requested unstake amount within the current epoch
```

```
154         self.user_amount_to_unstake_in_epoch += receive_amount;
155
156         Event::Unstake {
157             account_id: account_id.clone(),
158             unstaked_amount: U128(amount),
159             burnt_stake_shares: U128(num_shares),
160             new_unstaked_balance: U128(account.unstaked_amount),
161             new_stake_shares: U128(account.stake_shares),
162             unstaked_available_epoch_height: account.withdrawable_epoch_height,
163         }
164         .emit();
165
166         Event::FtBurn {
167             account_id,
168             amount: U128(num_shares),
169         }
170         .emit();
171     }
```

Listing 2.16: contracts/near-x/src/fungible_token/nearx_internal.rs

```
371     #[private]
372     pub fn get_unstake_release_epoch(&self, amount: u128) -> EpochHeight {
373         let mut available_amount: Balance = 0;
374         let mut total_staked_amount: Balance = 0;
375         for validator in self.validator_info_map.values() {
376             total_staked_amount += validator.staked;
377
378             if !validator.paused() && !validator.pending_unstake_release() && validator.staked > 0
                    {
379                 available_amount += validator.staked;
380             }
381
382             // found enough balance to unstake from available validators
383             if available_amount >= amount {
384                 return NUM_EPOCHS_TO_UNLOCK;
385             }
386         }
387
388         // nothing is actually staked, all balance should be available now
389         // still leave a buffer for the user
390         if total_staked_amount == 0 {
391             return NUM_EPOCHS_TO_UNLOCK;
392         }
393
394         // no enough available validators to unstake
395         // double the unstake wating time
396         2 * NUM_EPOCHS_TO_UNLOCK
397     }
```

Listing 2.17: contracts/near-x/src/contract/internal.rs

Impact The user's withdrawable epoch height can be advanced, which may result in a lack of liquidity for planned withdrawal.

Suggestion I Add a check to ensure that the user's withdrawable epoch height can only be postponed when the user tries to update the amount of unstaked tokens.

2.3 Additional Recommendation

2.3.1 Improper Usage of saturating_sub

Status Fixed in `Version 2`

Introduced by `Version 1`

Description It is recommended to change `saturating_sub` to `checked_sub` to ensure that `amount_to_stake` is larger than `self.reconciled_epoch_stake_amount`.

```
14    // keep calling this method until false is return
15    pub fn staking_epoch(&mut self) -> bool {
16        self.assert_staking_epoch_not_paused();
17
18        let min_gas = gas::STAKING_EPOCH
19            + gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE
20            + gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE_CB;
21        require!(
22            env::prepaid_gas() >= min_gas,
23            format!("{}. require at least {:?}", ERROR_NOT_ENOUGH_GAS, min_gas)
24        );
25
26        self.epoch_reconcilation();
27        // after cleanup, there might be no need to stake
28        if self.reconciled_epoch_stake_amount == 0 {
29            log!("no need to stake, amount to settle is zero");
30            return false;
31        }
32
33        let validator_to_stake_info =
34            self.get_validator_to_stake(self.reconciled_epoch_stake_amount);
35        require!(
36            validator_to_stake_info.0.is_some(),
37            ERROR_NO_VALIDATOR_AVAILABLE_TO_STAKE
38        );
39
40        let validator = validator_to_stake_info.0.unwrap();
41
42        let amount_to_stake = validator_to_stake_info.1;
43
44        log!("amount to stake is {:?}", amount_to_stake);
45
46        require!(
47            env::account_balance() >= amount_to_stake + MIN_BALANCE_FOR_STORAGE,
48            ERROR_MIN_BALANCE_FOR_CONTRACT_STORAGE
49        );
50
51        // update internal state
52        self.reconciled_epoch_stake_amount = self
```

```
53              .reconciled_epoch_stake_amount
54              .saturating_sub(amount_to_stake);
55
56          // do staking on selected validator
57          ext_staking_pool::ext(validator.account_id.clone())
58              .with_attached_deposit(amount_to_stake)
59              .with_static_gas(gas::DEPOSIT_AND_STAKE)
60              .deposit_and_stake()
61              .then(
62                  ext_staking_pool_callback::ext(env::current_account_id())
63                      .with_attached_deposit(NO_DEPOSIT)
64                      .with_static_gas(gas::ON_STAKE_POOL_DEPOSIT_AND_STAKE)
65                      .on_stake_pool_deposit_and_stake(validator.account_id.clone(), amount_to_stake),
66              );
67
68          Event::StakingEpochAttempt {
69              validator_id: validator.account_id,
70              amount: U128(amount_to_stake),
71          }
72          .emit();
73
74          true
75      }
```

Listing 2.18: contracts/near-x/src/contract/operator.rs

```
288     #[private]
289     pub fn get_validator_to_stake(&self, amount: Balance) -> (Option<ValidatorInfo>, Balance) {
290         let mut selected_validator = None;
291         let mut amount_to_stake: Balance = 0;
292
293         for validator in self.validator_info_map.values() {
294             let target_amount = self.get_validator_expected_stake(&validator);
295             if validator.staked < target_amount {
296                 let delta = std::cmp::min(target_amount - validator.staked, amount);
297                 if delta > amount_to_stake {
298                     amount_to_stake = delta;
299                     selected_validator = Some(validator);
300                 }
301             }
302         }
303
304         if amount_to_stake > 0 && amount - amount_to_stake <= ONE_NEAR {
305             amount_to_stake = amount;
306         }
307
308         // Note that it's possible that no validator is available
309         (selected_validator, amount_to_stake)
310     }
```

Listing 2.19: contracts/near-x/src/contract/internal.rs

Suggestion I Change `saturating_sub` to `checked_sub` in function `staking_epoch`.

2.3.2 Lack of Checking on Privileged Accounts

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When setting owner, operator_id, or treasury_id, there is no check whether `env::current_account_id()`, `owner_account_id`, `operator_account_id` and `treasury_account_id` are different.

```
184    // Owner update methods
185    #[payable]
186    pub fn set_owner(&mut self, new_owner: AccountId) {
187        assert_one_yocto();
188        require!(
189            env::predecessor_account_id() == self.owner_account_id,
190            ERROR_UNAUTHORIZED
191        );
192        self.temp_owner = Some(new_owner.clone());
193        Event::SetOwner {
194            old_owner: self.owner_account_id.clone(),
195            new_owner,
196        }
197        .emit();
198    }
```

Listing 2.20: contracts/near-x/src/contract/public.rs

```
221    #[payable]
222    pub fn set_operator_id(&mut self, new_operator_account_id: AccountId) {
223        assert_one_yocto();
224        self.assert_owner_calling();
225
226        Event::UpdateOperator {
227            old_operator: self.operator_account_id.clone(),
228            new_operator: new_operator_account_id.clone(),
229        }
230        .emit();
231
232        self.operator_account_id = new_operator_account_id;
233    }
234
235    #[payable]
236    pub fn set_treasury_id(&mut self, new_treasury_account_id: AccountId) {
237        assert_one_yocto();
238        self.assert_owner_calling();
239
240        Event::UpdateTreasury {
241            old_treasury_account: self.treasury_account_id.clone(),
242            new_treasury_account: new_treasury_account_id.clone(),
243        }
244        .emit();
245
246        self.treasury_account_id = new_treasury_account_id;
```

```
247     }
```

Listing 2.21: contracts/near-x/src/contract/public.rs

Impact The privileged accounts can be the same, leading to centralization problem.

Suggestion I Add checks in functions `set_owner`, `set_operator_id` and `set_treasury_id` to ensure that `env::current_account_id()`, `owner_account_id`, `operator_account_id` and `treasury_account_id` will not be the same account.

2.3.3 Incomplete Checking on Privileged Accounts

Status Fixed in `Version 3`

Introduced by `Version 2`

Description For the initialization function `new`, there is no check whether the `env::current_account_id` differs from the other privileged accounts.

```
 9    #[near_bindgen]
10    impl NearxPool {
11       #[init]
12       pub fn new(
13           owner_account_id: AccountId,
14           operator_account_id: AccountId,
15           treasury_account_id: AccountId,
16       ) -> Self {
17           require!(
18               owner_account_id != operator_account_id,
19               ERROR_OWNER_OPERATOR_SAME
20           );
21           require!(
22               owner_account_id != treasury_account_id,
23               ERROR_OWNER_TREASURY_SAME
24           );
25           require!(
26               operator_account_id != treasury_account_id,
27               ERROR_OPERATOR_TREASURY_SAME
28           );
```

Listing 2.22: contracts/near-x/src/contract/public.rs

Impact The privileged accounts can be the same, leading to centralization problem.

Suggestion I Add checks in functions `new` to ensure that `env::current_account_id()`, `owner_account_id`, `operator_account_id` and `treasury_account_id` will not be the same account.

2.3.4 Improper Marco Usage

Status Confirmed

Introduced by `Version 1`

Description Macro `#[private]` is usually used by callback functions. Function `get_validator_to_stake`, `get_validator_to_unstake`, `get_unstake_release_epoch` and `epoch_reconcilation` are not callback functions and macro `#[private]` should not be used.

```
288     #[private]
289     pub fn get_validator_to_stake(&self, amount: Balance) -> (Option<ValidatorInfo>, Balance) {
290         let mut selected_validator = None;
291         let mut amount_to_stake: Balance = 0;
292 
293         for validator in self.validator_info_map.values() {
294             let target_amount = self.get_validator_expected_stake(&validator);
295             if validator.staked < target_amount {
296                 let delta = std::cmp::min(target_amount - validator.staked, amount);
297                 if delta > amount_to_stake {
298                     amount_to_stake = delta;
299                     selected_validator = Some(validator);
300                 }
301             }
302         }
303 
304         if amount_to_stake > 0 && amount - amount_to_stake <= ONE_NEAR {
305             amount_to_stake = amount;
306         }
307 
308         // Note that it's possible that no validator is available
309         (selected_validator, amount_to_stake)
310     }
311 
312     #[private]
313     pub fn get_validator_to_unstake(&self) -> Option<ValidatorInfo> {
314         let mut max_validator_stake_amount: u128 = 0;
315         let mut current_validator: Option<ValidatorInfo> = None;
316 
317         for validator in self.validator_info_map.values() {
318             if !validator.pending_unstake_release()
319                 && !validator.paused()
320                 && validator.staked.gt(&max_validator_stake_amount)
321             {
322                 max_validator_stake_amount = validator.staked;
323                 current_validator = Some(validator)
324             }
325         }
326 
327         current_validator
328     }
329 
330     #[private]
331     pub fn get_unstake_release_epoch(&self, amount: u128) -> EpochHeight {
332         let mut available_amount: Balance = 0;
333         let mut total_staked_amount: Balance = 0;
334         for validator in self.validator_info_map.values() {
335             total_staked_amount += validator.staked;
336 
337             if !validator.paused() && !validator.pending_unstake_release() && validator.staked > 0
                    {
338                 available_amount += validator.staked;
```

```
339             }
340
341             // found enough balance to unstake from available validators
342             if available_amount >= amount {
343                 return NUM_EPOCHS_TO_UNLOCK;
344             }
345         }
346
347         // nothing is actually staked, all balance should be available now
348         // still leave a buffer for the user
349         if total_staked_amount == 0 {
350             return NUM_EPOCHS_TO_UNLOCK;
351         }
352
353         // no enough available validators to unstake
354         // double the unstake wating time
355         2 * NUM_EPOCHS_TO_UNLOCK
356     }
```

Listing 2.23: contracts/near-x/src/contract/internal.rs

```
438     #[private]
439     pub fn epoch_reconcilation(&mut self) {
440         if self.last_reconcilation_epoch == env::epoch_height() {
441             return;
442         }
443         self.last_reconcilation_epoch = env::epoch_height();
444
445         // here we use += because cleanup amount might not be 0
446         self.reconciled_epoch_stake_amount += self.user_amount_to_stake_in_epoch;
447         self.reconciled_epoch_unstake_amount += self.user_amount_to_unstake_in_epoch;
448         self.user_amount_to_stake_in_epoch = 0;
449         self.user_amount_to_unstake_in_epoch = 0;
450
451         let reconciled_stake_amount = self
452             .reconciled_epoch_stake_amount
453             .saturating_sub(self.reconciled_epoch_unstake_amount);
454         let reconciled_unstake_amount = self
455             .reconciled_epoch_unstake_amount
456             .saturating_sub(self.reconciled_epoch_stake_amount);
457
458         self.reconciled_epoch_stake_amount = reconciled_stake_amount;
459         self.reconciled_epoch_unstake_amount = reconciled_unstake_amount;
460
461         Event::EpochReconcile {
462             actual_epoch_stake_amount: U128(self.user_amount_to_stake_in_epoch),
463             actual_epoch_unstake_amount: U128(self.user_amount_to_unstake_in_epoch),
464             reconciled_stake_amount: U128(self.reconciled_epoch_stake_amount),
465             reconciled_unstake_amount: U128(self.reconciled_epoch_unstake_amount),
466         }
467         .emit();
468     }
```

Listing 2.24: contracts/near-x/src/contract/operator.rs

Suggestion I Revise the code accordingly.

2.3.5 Potential Centralization Problem

Status Fixed in `Version 2`

Introduced by `Version 1`

Description This project has potential centralization problems. The project owner needs to ensure the security of the private key of `NearxPool.owner_account_id`, `NearxPool.operator_account_id` and `NearxPool.treasury_account_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 multisignature or a public DAO.

Feedback from the Project The account will be multi-signature.

2.3.6 Unused Code (I)

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There are some unused code in the contract. Some of them are listed below.

```
4pub fn assert_min_balance(amount: u128) {
5    require!(amount > 0, ERROR_DEPOSIT_SHOULD_BE_GREATER_THAN_ZERO);
6    require!(
7        env::account_balance() >= MIN_BALANCE_FOR_STORAGE
8            && env::account_balance() - MIN_BALANCE_FOR_STORAGE > amount,
9        ERROR_MIN_BALANCE_FOR_CONTRACT_STORAGE
10   );
11}
12
13pub fn assert_callback_calling() {
14   require!(env::predecessor_account_id() == env::current_account_id());
15}
```

Listing 2.25: contracts/near-x/src/utils.rs

```
37#[near_bindgen]
38#[derive(BorshDeserialize, BorshSerialize, PanicOnDefault)]
39pub struct Contract {
40    metadata: LazyOption<FungibleTokenMetadata>,
41
42    pub accounts: LookupMap<AccountId, Balance>,
43    pub total_supply: Balance,
44    pub account_storage_usage: StorageUsage,
45}
```

Listing 2.26: contracts/near-x/src/fungible_token/nearx_token.rs

Suggestion I It is recommended to tidy up all the unused code and variables.

2.3.7 Unused Code (II)

Status Fixed in `Version 3`

Introduced by `Version 2`

Description There are some unused code in the contract. Some of them are listed below.

```
22    pub fn assert_operator_calling(&self) {
23        require!(
24            env::predecessor_account_id() == self.operator_account_id,
25            ERROR_UNAUTHORIZED
26        );
27    }
28
29    pub fn assert_treasury_calling(&self) {
30        require!(
31            env::predecessor_account_id() == self.treasury_account_id,
32            ERROR_UNAUTHORIZED
33        );
34    }
```

Listing 2.27: contracts/near-x/src/utils.rs

```
39pub const ERROR_INSUFFICIENT_FUNDS_FOR_STORAGE_RESERVE: &str =
40    "Need to send 50N for storage reserve";
```

Listing 2.28: contracts/near-x/src/errors.rs

Suggestion I It is recommended to remove the unused code and variables.

2.3.8 Redundant Code (I)

Status Confirmed

Introduced by `Version 1`

Description If `validator_info.is_empty()` is true, then `validator_info.weight` must be 0. Therefore, `self.total_validator_weight -= validator_info.weight;` is redundant.

```
123    #[payable]
124    pub fn remove_validator(&mut self, validator: AccountId) {
125        self.assert_operator_or_owner();
126        assert_one_yocto();
127
128        let validator_info = self.internal_get_validator(&validator);
129
130        require!(validator_info.is_empty(), ERROR_INVALID_VALIDATOR_REMOVAL);
131
132        self.total_validator_weight -= validator_info.weight;
133        self.validator_info_map.remove(&validator);
134
135        Event::ValidatorRemoved {
136            account_id: validator,
137        }
138        .emit();
```

```
139     }
```

Listing 2.29: contracts/near-x/src/contract/public.rs

```
94      pub fn is_empty(&self) -> bool {
95          self.paused()
96              && !self.pending_unstake_release()
97              && self.staked == 0
98              && self.unstaked_amount == 0
99      }
```

Listing 2.30: contracts/near-x/src/state.rs

```
117     pub fn paused(&self) -> bool {
118         self.weight == 0
119     }
```

Listing 2.31: contracts/near-x/src/state.rs

Suggestion I It is recommended to remove `self.total_validator_weight -= validator_info.weight;`.

Feedback from the Project Acknowledged. We will not go forward with this change as there are no user funds at stake and no user security at stake.

2.3.9 Redundant Code (II)

Status Confirmed

Introduced by `Version 1`

Description Function `staked_amount_from_num_shares_rounded_down` will check whether `self.total_staked` or `self.total_stake_shares` is 0. In this case, function `get_nearx_price` does not need to check repeatedly.

```
427     pub fn get_nearx_price(&self) -> U128 {
428         if self.total_staked == 0 || self.total_stake_shares == 0 {
429             return U128(ONE_NEAR);
430         }
431
432         let amount = self.staked_amount_from_num_shares_rounded_down(ONE_NEAR);
433         if amount == 0 {
434             U128(ONE_NEAR)
435         } else {
436             U128(amount)
437         }
438     }
```

Listing 2.32: contracts/near-x/src/contract/public.rs

```
247     pub(crate) fn staked_amount_from_num_shares_rounded_down(&self, num_shares: u128) -> Balance {
248         if self.total_staked == 0 || self.total_stake_shares == 0 {
249             return num_shares;
250         }
251
```

```
252         (U256::from(self.total_staked) * U256::from(num_shares)
253             / U256::from(self.total_stake_shares))
254         .as_u128()
255     }
```

Listing 2.33: contracts/near-x/src/contract/internal.rs

Suggestion I It is recommended to remove the check of whether `self.total_staked` or `self.total_stake_shares` is 0 in function `get_nearx_price`.

Feedback from the Project Acknowledged. We will not go forward with fixing this.

2.3.10 Redundant Code (III)

Status Confirmed

Introduced by `Version 1`

Description The `burned_amount` returned from `int_ft_resolve_transfer` is always 0. In this case, checking whether `burned_amount` is greater than 0 in function `ft_resolve_transfer` is redundant. In addition, since `burned_amount` is always 0, this item can be deleted in the return value of function `int_ft_resolve_transfer`.

```
29    pub fn int_ft_resolve_transfer(
30        &mut self,
31        sender_id: &AccountId,
32        receiver_id: AccountId,
33        amount: U128,
34    ) -> (u128, u128) {
35        let receiver_id = receiver_id;
36        let amount: Balance = amount.into();
37
38        // Get the unused amount from the `ft_on_transfer` call result.
39        let unused_amount = match env::promise_result(0) {
40            PromiseResult::NotReady => unreachable!(),
41            PromiseResult::Successful(value) => {
42                if let Ok(unused_amount) = near_sdk::serde_json::from_slice::<U128>(&value) {
43                    std::cmp::min(amount, unused_amount.0)
44                } else {
45                    amount
46                }
47            }
48            PromiseResult::Failed => amount,
49        };
50
51        if unused_amount > 0 {
52            let mut receiver_acc = self.internal_get_account(&receiver_id);
53            let receiver_balance = receiver_acc.stake_shares;
54            if receiver_balance > 0 {
55                let refund_amount = std::cmp::min(receiver_balance, unused_amount);
56                receiver_acc.stake_shares -= refund_amount;
57                self.internal_update_account(&receiver_id, &receiver_acc);
58
59                let mut sender_acc = self.internal_get_account(sender_id);
```

```
60              sender_acc.stake_shares += refund_amount;
61              self.internal_update_account(sender_id, &sender_acc);
62
63              log!(
64                  "Refund {} from {} to {}",
65                  refund_amount,
66                  receiver_id,
67                  sender_id
68              );
69              return (amount - refund_amount, 0);
70          }
71      }
72      (amount, 0)
73  }
```

Listing 2.34: contracts/near-x/src/fungible_token/nearx_internal.rs

```
118 #[private]
119 fn ft_resolve_transfer(
120     &mut self,
121     sender_id: AccountId,
122     receiver_id: AccountId,
123     amount: U128,
124 ) -> U128 {
125     let (used_amount, burned_amount) =
126         self.int_ft_resolve_transfer(&sender_id, receiver_id, amount);
127     if burned_amount > 0 {
128         log!("{} tokens burned", burned_amount);
129     }
130     used_amount.into()
131 }
```

Listing 2.35: contracts/near-x/src/fungible_token/nearx_token.rs

Suggestion I Revise the code accordingly.

Feedback from the Project Acknowledged. Since this is a very critical piece of code, we do not want to touch it. Since this recommendation has no impact on user funds and security of the smart contract, we acknowledge and appreciate the recommendation.

2.3.11 Redundant Code (IV)

Status Confirmed

Introduced by `Version 1`

Description In function `unstaking_epoch`, the `amount_to_unstake` is the minimum value of `validator_info.staked` and `self.reconciled_epoch_unstake_amount`. In this case, `amount_to_unstake` must be less than or equal to `validator_info.staked`. Therefore, checking whether `amount_to_unstake` is less than or equal to `validator_info.staked` (line 240 to line 243) is redundant.

```
210 pub fn unstaking_epoch(&mut self) -> bool {
211     self.assert_unstaking_epoch_not_paused();
212
```

```
213         let min_gas =
214             gas::UNSTAKING_EPOCH + gas::ON_STAKE_POOL_UNSTAKE + gas::ON_STAKE_POOL_UNSTAKE_CB;
215         require!(
216             env::prepaid_gas() >= min_gas,
217             format!("{}. require at least {:?}", ERROR_NOT_ENOUGH_GAS, min_gas)
218         );
219
220         self.epoch_reconcilation();
221
222         // after cleanup, there might be no need to unstake
223         if self.reconciled_epoch_unstake_amount == 0 {
224             log!("No amount to unstake");
225             return false;
226         }
227
228         let validator_to_unstake = self.get_validator_to_unstake();
229
230         require!(
231             validator_to_unstake.is_some(),
232             ERROR_NO_VALIDATOR_AVAILABLE_FOR_UNSTAKE
233         );
234
235         let mut validator_info = validator_to_unstake.unwrap();
236
237         let amount_to_unstake =
238             std::cmp::min(validator_info.staked, self.reconciled_epoch_unstake_amount);
239
240         require!(
241             amount_to_unstake <= validator_info.staked,
242             ERROR_CANNOT_UNSTAKED_MORE_THAN_STAKED_AMOUNT
243         );
244
245         self.reconciled_epoch_unstake_amount -= amount_to_unstake;
246         validator_info.staked -= amount_to_unstake;
247         validator_info.last_unstake_start_epoch = validator_info.unstake_start_epoch;
248         validator_info.unstake_start_epoch = env::epoch_height();
249
250         self.internal_update_validator(&validator_info.account_id, &validator_info);
251
252         ext_staking_pool::ext(validator_info.account_id.clone())
253             .with_static_gas(gas::ON_STAKE_POOL_UNSTAKE)
254             .with_attached_deposit(NO_DEPOSIT)
255             .unstake(U128(amount_to_unstake))
256             .then(
257                 ext_staking_pool_callback::ext(env::current_account_id())
258                     .with_attached_deposit(NO_DEPOSIT)
259                     .with_static_gas(gas::ON_STAKE_POOL_UNSTAKE_CB)
260                     .on_stake_pool_unstake(validator_info.account_id.clone(), amount_to_unstake),
261             );
262
263         Event::UnstakingEpochAttempt {
264             validator_id: validator_info.account_id,
265             amount: U128(amount_to_unstake),
```

```
266         }
267         .emit();
268
269         true
270     }
```

Listing 2.36: contracts/near-x/src/contract/operator.rs

Suggestion I It is recommended to remove the code from line 240 to line 243 in function `unstaking_epoch`.

2.3.12 Redundant Code (V)

Status Confirmed

Introduced by `Version 1`

Description In function `on_stake_pool_get_account`, if the absolute difference between `account.staked_balance.0` and `validator.staked` is less than or equal to 5000 and the absolute difference between `account.unstaked_balance.0` and `validator.unstaked_amount` is also less than or equal to 5000, then the absolute difference between `new_total_balance` and `validator.total_balance()` must be less than or equal to 10000. Therefore, using `abs_diff_eq` to check `new_total_balance` and `validator.total_balance()` is redundant.

```
399     #[private]
400     pub fn on_stake_pool_get_account(
401         &mut self,
402         validator_id: AccountId,
403         #[callback] account: HumanReadableAccount,
404     ) {
405         let mut validator = self.internal_get_validator(&validator_id);
406
407         let new_total_balance = account.staked_balance.0 + account.unstaked_balance.0;
408         require!(
409             abs_diff_eq(new_total_balance, validator.total_balance(), 10000),
410             ERROR_VALIDATOR_TOTAL_BALANCE_OUT_OF_SYNC
411         );
412
413         require!(
414             abs_diff_eq(account.staked_balance.0, validator.staked, 5000),
415             ERROR_VALIDATOR_STAKED_BALANCE_OUT_OF_SYNC
416         );
417         require!(
418             abs_diff_eq(account.unstaked_balance.0, validator.unstaked_amount, 5000),
419             ERROR_VALIDATOR_UNSTAKED_BALANCE_OUT_OF_SYNC
420         );
421
422         Event::BalanceSyncedFromValidator {
423             validator_id: validator_id.clone(),
424             old_staked_balance: U128(validator.staked),
425             old_unstaked_balance: U128(validator.unstaked_amount),
426             staked_balance: account.staked_balance,
427             unstaked_balance: account.unstaked_balance,
428         }
```

```
429         .emit();
430
431         // update balance
432         validator.staked = account.staked_balance.0;
433         validator.unstaked_amount = account.unstaked_balance.0;
434
435         self.internal_update_validator(&validator_id, &validator);
436     }
```

Listing 2.37: contracts/near-x/src/contract/operator.rs

Suggestion I It is recommended to remove the code from line 407 to line 411 in function `on_stake_pool_get_account`.

2.3.13 Redundant Log Emissions

Status Confirmed

Introduced by `Version 1`

Description Function `log!` is used very often in the contract and some of them are redundant.

```
146     #[private]
147     pub fn on_get_sp_staked_balance_for_rewards(
148         &mut self,
149         #[allow(unused_mut)] mut validator_info: ValidatorInfo,
150         #[callback] total_staked_balance: U128,
151     ) -> PromiseOrValue<bool> {
152         validator_info.last_redeemed_rewards_epoch = env::epoch_height();
153
154         //new_total_balance has the new staked amount for this pool
155         let new_total_balance = total_staked_balance.0;
156         log!("total staked balance is {}", total_staked_balance.0);
157
158         //compute rewards, as new balance minus old balance
159         let rewards = new_total_balance.saturating_sub(validator_info.staked);
160
161         log!(
162             "validator account:{} old_balance:{} new_balance:{} rewards:{}",
163             validator_info.account_id,
164             validator_info.staked,
165             new_total_balance,
166             rewards
167         );
168
169         self.internal_update_validator(&validator_info.account_id, &validator_info);
170
171         Event::AutocompoundingEpochRewards {
172             validator_id: validator_info.account_id.clone(),
173             old_balance: U128(validator_info.staked),
174             new_balance: U128(new_total_balance),
175             rewards: U128(rewards),
176         }
177         .emit();
```

```
178
179         if rewards > 0 {
180             //updated accumulated_staked_rewards value for the contract
181             self.accumulated_staked_rewards += rewards;
182             //updated new "staked" value for this pool
183             validator_info.staked = new_total_balance;
184
185             let operator_fee = rewards * self.rewards_fee;
186             log!("operator fee is {}", operator_fee);
187             self.total_staked += rewards;
188             let treasury_account_shares =
189                 self.num_shares_from_staked_amount_rounded_down(operator_fee);
190
191             self.internal_update_validator(&validator_info.account_id, &validator_info);
192
193             if treasury_account_shares > 0 {
194                 // Mint shares for the treasury account
195                 let treasury_account_id = self.treasury_account_id.clone();
196                 let mut treasury_account = self.internal_get_account(&treasury_account_id);
197                 treasury_account.stake_shares += treasury_account_shares;
198                 self.total_stake_shares += treasury_account_shares;
199                 self.internal_update_account(&treasury_account_id, &treasury_account);
200
201                 PromiseOrValue::Value(true)
202             } else {
203                 PromiseOrValue::Value(false)
204             }
205         } else {
206             PromiseOrValue::Value(false)
207         }
208     }
```

Listing 2.38: contracts/near-x/src/contract/operator.rs

Suggestion I It is recommended to use NEP-297 Events Standard instead of using `log!` function when logging.

Feedback from the Project It is useful for us in some places. We will leave it as it is as it doesn't harm user funds nor impacts gas fees.

2.3.14 Improper Gas Value

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The values of `ONE_T_GAS`, `FIVE_T_GAS`, and `TEN_T_GAS` are not used. Furthermore, the return value of `base_gas(1)` is 25T, but it is thought as 1T by the contract. For example, the real values of `ONE_T_GAS`, `FIVE_T_GAS`, and `TEN_T_GAS` are 25T, 125T, 250T, which are incorrect.

```
91     pub const ONE_T_GAS: Gas = base_gas(1);
92
93     pub const FIVE_T_GAS: Gas = base_gas(5);
94
```

```
95    pub const TEN_T_GAS: Gas = base_gas(10);
96
97    const fn base_gas(n: u64) -> Gas {
98        Gas(1_000_000_000_000 * 25 * n)
99    }
100
101   const fn tera(n: u64) -> Gas {
102       Gas(1_000_000_000_000 * n)
103   }
```

Listing 2.39: contracts/near-x/src/constants.rs

Suggestion I Revise the value of the constants that are related to `base_gas()`.

2.3.15 Improper Calculation of Operator Fee Shares

Status Confirmed

Introduced by `Version 1`

Description According to the reward distribution implemented in function `on_get_sp_staked_balance_for_rewards`, the corresponding shares of operator fee (NEARx) minted to the treasury account are less than expected. This is because the `treasury_account_shares` (line 189) is calculated based on the `self.total_staked`, which already includes the amount of the newly added rewards (line 187).

```
146   #[private]
147   pub fn on_get_sp_staked_balance_for_rewards(
148       &mut self,
149       #[allow(unused_mut)] mut validator_info: ValidatorInfo,
150       #[callback] total_staked_balance: U128,
151   ) -> PromiseOrValue<bool> {
152       validator_info.last_redeemed_rewards_epoch = env::epoch_height();
153
154       //new_total_balance has the new staked amount for this pool
155       let new_total_balance = total_staked_balance.0;
156       log!("total staked balance is {}", total_staked_balance.0);
157
158       //compute rewards, as new balance minus old balance
159       let rewards = new_total_balance.saturating_sub(validator_info.staked);
160
161       log!(
162           "validator account:{} old_balance:{} new_balance:{} rewards:{}",
163           validator_info.account_id,
164           validator_info.staked,
165           new_total_balance,
166           rewards
167       );
168
169       self.internal_update_validator(&validator_info.account_id, &validator_info);
170
171       Event::AutocompoundingEpochRewards {
172           validator_id: validator_info.account_id.clone(),
173           old_balance: U128(validator_info.staked),
174           new_balance: U128(new_total_balance),
```

```
175             rewards: U128(rewards),
176         }
177         .emit();
178
179         if rewards > 0 {
180             //updated accumulated_staked_rewards value for the contract
181             self.accumulated_staked_rewards += rewards;
182             //updated new "staked" value for this pool
183             validator_info.staked = new_total_balance;
184
185             let operator_fee = rewards * self.rewards_fee;
186             log!("operator fee is {}", operator_fee);
187             self.total_staked += rewards;
188             let treasury_account_shares =
189                 self.num_shares_from_staked_amount_rounded_down(operator_fee);
190
191             self.internal_update_validator(&validator_info.account_id, &validator_info);
192
193             if treasury_account_shares > 0 {
194                 // Mint shares for the treasury account
195                 let treasury_account_id = self.treasury_account_id.clone();
196                 let mut treasury_account = self.internal_get_account(&treasury_account_id);
197                 treasury_account.stake_shares += treasury_account_shares;
198                 self.total_stake_shares += treasury_account_shares;
199                 self.internal_update_account(&treasury_account_id, &treasury_account);
200
201                 PromiseOrValue::Value(true)
202             } else {
203                 PromiseOrValue::Value(false)
204             }
205         } else {
206             PromiseOrValue::Value(false)
207         }
208     }
```

Listing 2.40: contracts/near-x/src/contract/operator.rs

Suggestion I Deduct the `operator_fee` from the pending rewards before it is added to the `self.total_staked` (line 187), and then calculate the `treasury_account_shares` separately.

Feedback from the Project We are mainly fine with it since the rewards are autocompounded.

2.4 Additional Notes

2.4.1 Only Staked Balance is Used to Calculate Rewards

Status Confirmed

Introduced by `Version 1`

Description In the validator staking pool contract, each account has two kinds of balances, which are staked balance and unstaked balance. Function `get_account_staked_balance` only returns the staked

balance of an account, while function `get_account_total_balance` returns the sum of staked balance and unstaked balance. The contract only uses staked balance to calculate rewards.

```
101    pub fn autocompounding_epoch(&mut self, validator: AccountId) {
102        self.assert_autocompounding_epoch_not_paused();
103
104        let min_gas = gas::AUTOCOMPOUNDING_EPOCH
105            + gas::ON_STAKE_POOL_GET_ACCOUNT_STAKED_BALANCE
106            + gas::ON_STAKE_POOL_GET_ACCOUNT_STAKED_BALANCE_CB;
107        require!(
108            env::prepaid_gas() >= min_gas,
109            format!("{}. require at least {:?}", ERROR_NOT_ENOUGH_GAS, min_gas)
110        );
111
112        let validator_info = self.internal_get_validator(&validator);
113
114        let epoch_height = env::epoch_height();
115
116        if validator_info.staked == 0 {
117            return;
118        }
119
120        if validator_info.last_redeemed_rewards_epoch == epoch_height {
121            return;
122        }
123
124        log!(
125            "Fetching total balance from the staking pool {}",
126            validator_info.account_id
127        );
128
129        ext_staking_pool::ext(validator_info.account_id.clone())
130            .with_attached_deposit(NO_DEPOSIT)
131            .with_static_gas(gas::ON_STAKE_POOL_GET_ACCOUNT_STAKED_BALANCE)
132            .get_account_staked_balance(env::current_account_id())
133            .then(
134                ext_staking_pool_callback::ext(env::current_account_id())
135                    .with_attached_deposit(NO_DEPOSIT)
136                    .with_static_gas(gas::ON_STAKE_POOL_GET_ACCOUNT_STAKED_BALANCE_CB)
137                    .on_get_sp_staked_balance_for_rewards(validator_info),
138            );
139
140        Event::AutocompoundingEpochRewardsAttempt {
141            validator_id: validator,
142        }
143        .emit();
144    }
```

Listing 2.41: contracts/near-x/src/contract/operator.rs

Feedback from the Project Autocompounding rewards in the validator stake pool contracts are only accrued to the staked balance of the account and not to the unstaked balance. Unstaked balance would be same in the stake pool contract and it would be redundant to fetch it.