// Security Assessment

04.03.2025 - 04.29.2025

Fastnear House of Stake

NEAR Foundation

HALBORN

Fastnear House of Stake - NEAR Foundation

Prepared by: HALBORN

Last Updated 05/26/2025

Date of Engagement: April 3rd, 2025 - April 29th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
14	0	0	0	3	11

TABLE OF CONTENTS

8. Automated Testing

1. Introduction

`FastNEAR` engaged **Halborn** to conduct a security assessment of their House-of-Stake (HoS) protocol for **NEAR** blockchain, beginning on April 3rd, 2025, and ending on April 29th, 2025. The security assessment was scoped to the repository listed with commit hash, and further details in the Scope section of this report.

The House of Stake (HoS) protocol is a NEAR-based governance system that allows users to lock their NEAR tokens to receive veNEAR (voting-escrowed NEAR), which grants them voting power in the protocol's governance decisions, while still enabling them to stake their locked NEAR to validators or liquid staking providers.

2. Assessment Summary

The team at Halborn assigned one full-time security engineer to verify the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing and smart-contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were addressed and mostly solved by the `FastNEAR team`. The main ones were the following:

- `Implement a pausability mechanism for the protocol contracts.`
- `Validate at the beginning of the internal_deploy_lockup function that the prepaid gas is sufficient to cover the total execution requirements.`
- `Implement a two-step ownership transfer process.`
- `Create low-privilege roles for executing routine tasks within the protocol to minimize the impact of potential compromises of these accounts.`

3. Test Approach And Methodology

Halborn performed a combination of the manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation, automated testing techniques help enhance the coverage of smart contracts. They can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual code reading and walkthrough.
- Manual assessment of the use and safety of critical Rust variables and functions to identify potential arithmetic vulnerabilities.
- Cross-contract call controls.
- Logical controls related to architecture.
- Scanning of Rust files for vulnerabilities using tools like `cargo audit`.
- Integration testing using the NEAR testing framework.

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: house-of-stake-contracts

(b) Assessed Commit ID: 8643d31

(c) Items in scope:

- common/src/events.rs
- common/src/types.rs
- common/src/account.rs
- common/src/global_state.rs
- common/src/venear.rs
- common/src/utils.rs
- common/src/lockup_update.rs
- common/src/lib.rs
- lockup-contract/src/lib.rs
- lockup-contract/src/owner.rs
- lockup-contract/src/owner_callbacks.rs
- lockup-contract/src/venear.rs
- lockup-contract/src/internal.rs
- lockup-contract/src/gas.rs
- lockup-contract/src/getters.rs
- lockup-contract/src/types.rs
- lockup-contract/src/venear_ext.rs
- merkle-tree/src/lib.rs
- venear-contract/src/lockup.rs
- venear-contract/src/account.rs
- venear-contract/src/storage.rs
- venear-contract/src/governance.rs
- venear-contract/src/upgrade.rs
- venear-contract/src/lib.rs
- venear-contract/src/delegation.rs
- venear-contract/src/token.rs
- venear-contract/src/config.rs
- venear-contract/src/snapshot.rs
- venear-contract/src/global_state.rs
- voting-contract/src/proposal.rs
- voting-contract/src/reviewer.rs
- voting-contract/src/votes.rs
- voting-contract/src/upgrade.rs
- voting-contract/src/governance.rs
- voting-contract/src/lib.rs
- voting-contract/src/metadata.rs

- voting-contract/src/config.rs

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- https://github.com/fastnear/house-of-stake-contracts/pull/35

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	3

INFORMATIONAL
11

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACKING PAUSABILITY MECHANISM	LOW	SOLVED - 05/21/2025
LACK OF PREPAID GAS VALIDATION MAY RESULT IN UNINTENDED CONSUMPTION OF GAS	LOW	SOLVED - 05/21/2025
SINGLE-STEP OWNERSHIP TRANSFER PROCESS	LOW	SOLVED - 05/21/2025
LACK OF VALIDATION FOR CRITICAL CONTRACT PARAMETERS	INFORMATIONAL	ACKNOWLEDGED - 05/21/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CENTRALIZATION RISK	INFORMATIONAL	ACKNOWLEDGED - 05/21/2025
MISSING DESCRIPTIVE ERROR MESSAGES	INFORMATIONAL	SOLVED - 05/21/2025
UNCHECKED MATH OPERATIONS	INFORMATIONAL	SOLVED - 05/21/2025
MISSING EVENT EMISSION	INFORMATIONAL	ACKNOWLEDGED - 05/21/2025
DUPLICATE LOGIC INTRODUCED VIA REPEATED FUNCTION	INFORMATIONAL	ACKNOWLEDGED - 05/21/2025
MINOR SPELLING INCONSISTENCIES IN DOCUMENTATION	INFORMATIONAL	SOLVED - 05/21/2025
SUBOPTIMAL INLINE DOCUMENTATION	INFORMATIONAL	SOLVED - 05/21/2025
PRESENCE OF TODO COMMENT	INFORMATIONAL	SOLVED - 05/21/2025
INEFFECTIVE MIN() EVALUATION DUE TO CONSTANT DOMINANCE	INFORMATIONAL	SOLVED - 05/21/2025
UNNECESSARY IMPLEMENTATION OF SET_VOTE_STORAGE_FEE	INFORMATIONAL	SOLVED - 05/21/2025

7. FINDINGS & TECH DETAILS

7.1 LACKING PAUSABILITY MECHANISM

// LOW

Description

The **venear** and **voting** contracts currently lack a pausability mechanism. This feature is crucial for smart contracts, especially those handling significant assets or performing critical operations. A pause functionality allows the contract owner or designated parties to temporarily halt contract operations in case of emergencies, such as discovered vulnerabilities, unexpected behavior, or during upgrades.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (2.5)

Recommendation

It is recommended to implement a pausability mechanism for the aforementioned contracts.

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue in the specified pull request.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.2 LACK OF PREPAID GAS VALIDATION MAY RESULT IN UNINTENDED CONSUMPTION OF GAS

// LOW

Description

Although the `internal_deploy_lockup` function performs two external calls, the initialization of the lockup contract and the `on_lockup_deployed` callback, it does not currently validate whether the caller has attached sufficient execution gas to complete these operations. As a result, if the provided gas is inadequate, the function proceeds until it fails, consuming the entire gas allocation.

This lack of an early validation step prevents the contract from reverting early, which could otherwise preserve some of the user's gas and improve the overall user experience.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:L/I:N/D:L/Y:N (2.1)

Recommendation

It is recommended to validate at the beginning of the `internal_deploy_lockup` function that the prepaid gas is sufficient to cover the total execution requirements. This should include, at a minimum, the sum of `LOCKUP_DEPLOY_MIN_GAS` and `ON_LOCKUP_DEPLOYED`, ensuring the function has adequate gas to complete both external calls reliably.
This precaution will help prevent unnecessary execution attempts that could exhaust all attached gas, ensuring a more efficient use of resources.

Remediation Comment

`SOLVED`: The `FastNEAR team` solved this issue in the specified pull request.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.3 SINGLE-STEP OWNERSHIP TRANSFER PROCESS

// LOW

Description

The ownership transfer of the **venear** and **voting** contracts is executed in a single step, which introduces a security risk. This issue is present in the `governance::set_owner_account_id` function, where the change of ownership lacks a verification step before finalization. As a result, there is a risk that the owner could make a mistake when executing the ownership transfer, such as setting an incorrect address, which could lead to the contract being locked or inaccessible.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:N/D:N/Y:N (2.0)

Recommendation

It is recommended to implement a two-step ownership transfer process: one function to establish the new owner's address (`propose_new_owner`) and another to accept the transfer (`accept_ownership`). The latter function should only be executable by the new owner.

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue in the specified pull request.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.4 LACK OF VALIDATION FOR CRITICAL CONTRACT PARAMETERS

// INFORMATIONAL

Description

It was identified that critical parameters within the **venear** and **voting** contracts lack proper validation before being updated. The affected parameters include:

- **venear** contract: `min_lockup_deposit`, `local_deposit`, `staking_pool_whitelist_account_id`, `unlock_duration_ns`, and `lockup_code_deployers`.
- **voting** contract: `venear_account_id`, `reviewer_ids`, `voting_duration_ns`, `base_proposal_fee`, and `max_number_of_voting_options`.

Without appropriate validation, incorrect or unintended values could be introduced by the contract owner, leading to the following potential issues:

- `min_lockup_deposit` and `local_deposit`: Values set too low or too high may result in expensive deployments due to misaligned minimum deposit requirements.
- `staking_pool_whitelist_account_id` and `venear_account_id`: Updating these parameters with the same account ID could lead to unnecessary updates and storage costs.
- `unlock_duration_ns`: An improperly configured unlock duration could cause user funds to be inaccessible for excessively long periods (delaying rewards), or unlock too early, preventing meaningful staking and undermining the intended vesting mechanism.
- `lockup_code_deployers`: Setting this vector to an empty or excessively large value may render the `prepare_lockup_code` function uncallable or introduce redundant entries in the `lockup_code_deployers` vector, leading to inefficiencies and potential maintainability issues.
- `reviewer_ids`: Similarly, this vector may contain duplicate entries, and if cleared, would temporarily disable the ability to approve or reject proposals.
- `voting_duration_ns`: Without well-defined lower and upper bounds, the voting phase may either end too quickly for users to participate or drag on unnecessarily long.
- `base_proposal_fee`: Without well-defined lower and upper bounds, the fee might be too low to cover storage costs or too high, potentially discouraging users from creating proposals.
- `max_number_of_voting_options`: Since the `create_proposal` function requires at least two voting options, setting this parameter below that threshold would prevent proposals from being created.

BVSS

AO:S/AC:L/AX:L/R:P/S:U/C:N/A:H/I:H/D:M/Y:N (1.1)

Recommendation

To mitigate these risks, it is recommended to implement validation checks for each of these parameters to ensure they fall within safe, logical, and contractually meaningful bounds.

Remediation Comment

`ACKNOWLEDGED` : The `FastNEAR team` acknowledged this issue.

7.5 CENTRALIZATION RISK

// INFORMATIONAL

Description

The **venear** contract designates an owner during installation, granting full control over the upgrade process and the ability to configure critical protocol parameters in `venear-contract/src/governance.rs` .

This violates the principle of separation of duties, introduces centralization risks, and grants the owner absolute control over the entire contract.

BVSS

AO:S/AC:L/AX:H/R:N/S:U/C:N/A:C/I:C/D:C/Y:N (1.0)

Recommendation

It is recommended to create low-privilege roles for executing routine tasks within the protocol to minimize the impact of potential compromises of these accounts.

Remediation Comment

`ACKNOWLEDGED` : The `FastNEAR team` acknowledged this issue. The team stated the following:

> It's working as intended. The owner ID is considered to be a contract or a DAO that controls the permissions of the contract. Most of the time teams are using Sputnik DAO to control the contract. The upgrade pattern is built with the Sputnik DAO in mind. There are now two extra roles for **venear** contract:
>
> - Lockup deployers - accounts that can upload new lockup code, but not activate it.
> - Guardians - accounts that can pause the contract.
>
> The rest is controlled by the owner account.

Although the aforementioned roles were introduced, the owner account still has the ability to pause the contract and retains control over several critical parameters.

7.6 MISSING DESCRIPTIVE ERROR MESSAGES

// INFORMATIONAL

Description

Multiple **require** statements without descriptive error messages were identified in the **venear** contract. As a result, any failure at this point triggers a generic panic without providing meaningful context, making it difficult for developers and users to understand the root cause.

Additionally, multiple instances of unchecked unwrap operations were observed throughout the contract. If any of these unwrap calls fail, they result in a generic "Trap: unreachable" error. These errors are ambiguous and provide no actionable information, significantly hindering debugging efforts and reducing overall developer and user experience.

Code Location

Below are the identified **require** statements within the `venear-contract/src/lockup.rs` file:

```
188 | require!(hash == contract_hash);
```

```
234 | require!(res == 1);
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to include clear, descriptive error messages in all require statements and to handle `Option` and `Result` types more gracefully, avoiding unchecked unwrap calls where possible. For example, `unwrap_or_else(|| env::panic_str("meaningful error message"))` can be used instead of `unwrap()` when the failure should never happen unless there's a logic bug or misconfiguration.

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue in the specified pull request.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.7 UNCHECKED MATH OPERATIONS

// INFORMATIONAL

Description

In computer programming, an overflow occurs when an arithmetic operation attempts to create a numeric value that is outside the range that can be represented with a given number of bits – either larger than the maximum or lower than the minimum representable value. This is a good practice, since having `overflow-checks` enabled in the workspace would avoid overflow situations in release mode.

The identified unchecked operations are listed below:

```
venear-contract/src/lockup.rs:
L162: global_state.total_venear_balance -= old_balance;
L163: global_state.total_venear_balance += account.balance;
L168: delegation_account.delegated_balance -= old_balance;
L169: delegation_account.delegated_balance += account.balance;
L197: + 1

venear-contract/src/governance.rs:
L64: unlock_duration_sec as u64 * 1_000_000_000

venear-contract/src/delegation.rs:
L25: delegation_account.delegated_balance += account.balance;
L50: delegation_account.delegated_balance -= account.balance;

venear-contract/src/account.rs:
L126: global_state.total_venear_balance += account.balance;
L171: total += account.balance;
L179: new_balance += account.balance;

merkle-tree/src/lib.rs
L197: for height in 0..self.tree_height() - 1 {

lockup-contract/src/getters.rs
L35: + self.get_known_deposited_balance().as_yoctonear()
L43: + self.get_known_deposited_balance().as_yoctonear()

lockup-contract/src/owner_callbacks.rs
L49:  + amount.as_yoctonear()
L194: + amount.as_yoctonear()

lockup-contract/src/venear.rs
L13: + self.get_known_deposited_balance().as_yoctonear()
L23: self.venear_unlock_timestamp = env::block_timestamp() + self.unlock_duration_ns
L27: self.lockup_update_nonce += 1;
L90: self.venear_locked_balance += amount;
L124: self.venear_locked_balance -= amount;
L157: self.venear_pending_balance -= amount;
L181: self.venear_pending_balance -= amount;
L182: self.venear_locked_balance += amount;

voting-contract/src/proposal.rs
L112: self.total_votes += 1;
L117: self.total_votes -= 1;
L129: if timestamp.0 >= self.voting_start_time_ns.unwrap().0 + self.voting_duration_ns.0
```

While the likelihood of an overflow issue arising from the identified occurrences is low, it's still best practice to handle overflow errors gracefully.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to utilize checked arithmetic operations to handle overflow errors gracefully.

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue in the specified pull request. The team has enabled the `overflow-checks` flag for the development and test profiles to detect and handle overflow errors effectively.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.8 MISSING EVENT EMISSION

// INFORMATIONAL

Description

The **venear** and **voting** contracts implement owner-only functions that modify critical state variables without emitting events. The affected functions are the following:

- `venear::set_lockup_contract`
- `venear::set_local_deposit`
- `venear::set_staking_pool_whitelist_account_id`
- `venear::set_owner_account_id`
- `venear::set_unlock_duration_sec`
- `venear::set_lockup_code_deployers`
- `voting::set_venear_account_id`
- `voting::set_reviewer_ids`
- `voting::set_voting_duration`
- `voting::set_base_proposal_fee`
- `voting::set_max_number_of_voting_options`
- `voting::set_owner_account_id`

The absence of events prevents external systems or users from tracking critical administrative changes via event logs. This lack of transparency diminishes visibility into owner actions that impact the protocol's behavior.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to emit an event whenever important contract state variables are updated.

Remediation Comment

`ACKNOWLEDGED` : The `FastNEAR team` acknowledged this issue. The team stated the following:

> Noted. We rely on the governance owner to notify the community about changes.

7.9 DUPLICATE LOGIC INTRODUCED VIA REPEATED FUNCTION

// INFORMATIONAL

Description

Within the **lockup** contract, the `get_owners_balance` function is called multiple times to return the balance of the account owner, including both vested and extra tokens that may have been deposited. However, it was identified that the `get_balance` function duplicates the logic of `get_owners_balance`, leading to redundant code, increased contract size, and higher deployment costs.

Code Location

Below is the implementation of the `get_owners_balance` function:

```
32 pub fn get_owners_balance(&self) -> NearToken {
33     NearToken::from_yoctonear(
34         env::account_balance().as_yoctonear()
35             + self.get_known_deposited_balance().as_yoctonear(),
36     )
37 }
```

Below is the implementation of the `get_balance` function:

```
40 pub fn get_balance(&self) -> NearToken {
41     NearToken::from_yoctonear(
42         env::account_balance().as_yoctonear()
43             + self.get_known_deposited_balance().as_yoctonear(),
44     )
45 }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to remove the `get_balance` function.

Remediation Comment

`ACKNOWLEDGED` : The `FastNEAR team` acknowledged this issue. The team chose to call the `get_balance` function from within `get_owners_balance`, opting to retain both functions due to frontend dependencies. They stated the following:

> Reused logic from one method; keeping the other, since there is already frontend work in progress.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.10 MINOR SPELLING INCONSISTENCIES IN DOCUMENTATION

// INFORMATIONAL

Description

The following typographical errors were identified within the **lockup** contract:

```
lockup-contract/src/owner_callbacks.rs
L138: Called after the request to get the current unstaked balance to withdraw everything by th
lockup-contract/src/owner.rs
L251: Tries to withdraws all unstaked balance from the staking pool
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to fix the aforementioned typographical errors as follows:

- th -> the
- withdraws -> withdraw

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue in the specified pull request.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.11 SUBOPTIMAL INLINE DOCUMENTATION

// INFORMATIONAL

Description

Incorporating documentation and comments into the codebase is essential for elucidating key aspects and facilitating comprehension of the developer's intentions by others. However, the codebase contains a misleading comment in the `transfer` function within the lockup contract, which states, ***"This requires transfers to be enabled within the voting contract".*** This contradicts the actual implementation, which lacks any transfer-enabling mechanism.

Additionally, the documentation for the `begin_unlock_near` function in the lockup contract contains an incomplete comment: ***"Starts the unlocking process of the in the lockup contract"***, which lacks clarity and fails to fully describe the function's behavior.

Code Location

Below is a code snippet of the `transfer` function:

```
411 /// This requires transfers to be enabled within the voting contract.
412 #[payable]
413 pub fn transfer(&mut self, amount: NearToken, receiver_id: AccountId) -> Promise {
```

Below is a code snippet of the `begin_unlock_near` function:

```
108 /// Starts the unlocking process of the in the lockup contract.
109 /// You specify the amount of near to unlock, or if you don't specify it, all the locked NEAR
110 /// will be unlocked.
111 /// (works similarly to unstaking from a staking pool).
112 #[payable]
113 pub fn begin_unlock_near(&mut self, amount: Option<NearToken>) {
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to remove the incorrect comment within the `transfer` function and complete the missing portion of the `begin_unlock_near` function's documentation. A suggested revision could be:

```
// Starts the unlocking process of the locked NEAR in the lockup contract.
```

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue in the specified pull request.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.12 PRESENCE OF TODO COMMENT

// INFORMATIONAL

Description

The following todo comment was identified within the **lockup** contract:

```
lockup-contract/src/gas.rs
L57:    // TODO: Deprecate
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to resolve the open ToDo.

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue by deprecating the relevant code and removing the associated TODO comment.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.13 INEFFECTIVE MIN() EVALUATION DUE TO CONSTANT DOMINANCE

// INFORMATIONAL

Description

Within the **lockup** contract, the `get_liquid_owners_balance` function is implemented to return the minimum of `get_owners_balance()` and `get_account_balance()`. However, this logic is redundant, as `get_account_balance()`, defined as `env::account_balance() - min_lockup_deposit`, will always return a value less than or equal to `get_owners_balance()`, which is defined as `env::account_balance() + deposit_amount`.

As a result, `get_liquid_owners_balance` will always return the same value as `get_account_balance()`, introducing unnecessary complexity and making the function itself redundant.

Code Location

Below is the implementation of the `get_liquid_owners_balance` function:

```
49 pub fn get_liquid_owners_balance(&self) -> NearToken {
50     std::cmp::min(self.get_owners_balance(), self.get_account_balance()).into()
51 }
```

Below is the implementation of the `get_account_balance` function:

```
11 pub fn get_account_balance(&self) -> NearToken {
12     NearToken::from_yoctonear(
13         env::account_balance()
14             .saturating_sub(self.min_lockup_deposit)
15             .as_yoctonear(),
16     )
17 }
```

Below is the implementation of the `get_owners_balance` function:

```
32 pub fn get_owners_balance(&self) -> NearToken {
33     NearToken::from_yoctonear(
34         env::account_balance().as_yoctonear()
35             + self.get_known_deposited_balance().as_yoctonear(),
36     )
37 }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to remove the `get_liquid_owners_balance` function and replace its usage with the `get_account_balance` function wherever applicable. Additionally, since `get_account_balance` is currently an internal function, unlike the publicly accessible

`get_liquid_owners_balance` , consider making it public to preserve external access where needed.

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue by keeping the `get_liquid_owners_balance` function and modifying it to return the result of `get_account_balance().`

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

7.14 UNNECESSARY IMPLEMENTATION OF SET_VOTE_STORAGE_FEE

// INFORMATIONAL

Description

The `set_vote_storage_fee` function, defined within the **voting** contract as an owner-only function, will always triggers a panic with the error message: ***"Vote storage fee cannot be changed, without contract upgrade".*** This renders the function effectively useless, as it cannot fulfill its intended purpose. Its presence not only increases the contract's bytecode size, leading to higher deployment gas costs, but also introduces unnecessary execution costs if mistakenly invoked by the owner. Therefore, unless future upgrades will enable this functionality, it would be more efficient to remove the function entirely.

Code Location

Below is the implementation of the `set_vote_storage_fee` function:

```
50 #[payable]
51 pub fn set_vote_storage_fee(&mut self, _vote_storage_fee: NearToken) {
52     // The reason for this restriction is that the contract needs to be upgraded to change the
53     // storage fee. Otherwise, the storage for the previous votes will be refunded with the
54     // new storage fee.
55     env::panic_str("Vote storage fee cannot be changed, without contract upgrade");
56 }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to remove the `set_vote_storage_fee` function.

Remediation Comment

`SOLVED` : The `FastNEAR team` solved this issue in the specified pull request.

Remediation Hash

https://github.com/fastnear/house-of-stake-contracts/pull/35

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. To better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the `cargo audit` output to better know the dependencies affected by unmaintained and vulnerable crates.

ID	CRATE	VERSION	TITLE	DATE	SOLUTION
RUSTSEC-2025-0022	openssl	0.10.71	Use-After-Free in `Md::fetch` and `Cipher::fetch`	2025-04-04	Upgrade to >=0.10.72
RUSTSEC-2022-0054	wee_alloc	0.4.5	wee_alloc is Unmaintained	2022-05-11	
RUSTSEC-2025-0023	tokio	1.44.1	Broadcast channel calls clone in parallel, but does not require `Sync`	2025-04-07	

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.