Cyberscope

Audit Report

NeonSwap Staking

March 2025

Repository	https://github.com/neonswapfi/neonswap-contracts
Commit	ec95985de65ccc2124c400187d826c9da6df847d
Audited by	© cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/neonswapfi/neonswap-contracts
Commit	ec95985de65ccc2124c400187d826c9da6df847d

Audit Updates

Initial Audit	19 Mar 2025

Source Files

Filename	SHA256
neoswap_staking/src/contract.rs	37a0a527ae554d1e85b592185362e5a2bf6d35d8d483f3e5dc2c3b3558337c94
neoswap_staking/src/lib.rs	25dbff7189f00e838ea3e3f42dce3751094b289e8c65fe500a7a725296e66a16
neoswap_staking/src/querier.rs	fa0fa7e92ddf716a7af1f5b8827c13dbfb24100a2a22aeabec02e5cc5988fa66
neoswap_staking/src/state.rs	98034e96f2a6b4b0de916e0ab0b69264f4f1b6ca959ed9b5ec37c0208def36ee

Overview

Staking Contract

The `Staking` contract is designed to facilitate staking of tokens, allowing users to bond their tokens in exchange for rewards. This contract supports core staking functionalities, including bonding, unbonding, reward distribution, and governance updates. It ensures a fair and transparent staking mechanism by enabling periodic reward distribution while preventing unauthorized access through governance control mechanisms.

Bonding Functionality

Users can bond their tokens to participate in the staking program. The `bond` function allows users to stake a specific amount of tokens. The contract verifies that only the staking token contract can execute this function. Upon bonding, the user's stake is recorded, and their reward distribution index is updated. The contract ensures that staked tokens are properly accounted for within the contract's state.

Unbonding Functionality

The `unbond` function allows users to withdraw their staked tokens. To prevent abuse, the contract verifies that users cannot unbond more tokens than they have staked. Additionally, before executing an unbonding request, the contract computes the user's rewards and updates their staking information. If a user fully unbonds their stake and has no pending rewards, their staking record is removed from the contract's storage.

Reward Distribution Functionality

The contract periodically distributes staking rewards according to a predefined schedule. The `compute_reward` function calculates the amount of rewards to be distributed at each time interval. The global reward index is updated accordingly, ensuring that all stakers receive their fair share of the rewards. Users can claim their pending rewards through the `withdraw` function, which transfers the calculated reward amount to the staker.

Governance and Configuration Updates

The contract includes governance features that allow administrators to modify key parameters. The `update_config` function enables the modification of the staking distribution schedule, ensuring flexibility in adjusting rewards over time. The `update_gov` function allows for updating the governance address, ensuring that only authorized entities can make changes. Additionally, the `set_contract_whitelist` function controls which contracts can interact with the staking contract, enhancing security by preventing unauthorized access.

Staking Migration

To support contract upgrades, the `migrate_staking` function allows migrating staking balances to a new staking contract. This function ensures that staked tokens and remaining rewards are transferred to the new contract securely. The migration process includes updating distribution schedules, computing remaining rewards, and ensuring that stakers retain their balances.

Security and Safety Measures

The contract includes various safety mechanisms to protect user funds and prevent misuse. Unauthorized contract interactions are blocked unless explicitly whitelisted. Reward calculations and updates are performed securely to prevent over-distribution or reward manipulation. Additionally, the use of governance controls ensures that only trusted entities can modify critical parameters, maintaining the integrity of the staking mechanism.

Roles

Admins

Administrators with governance privileges can interact with the following functions:

- `update_config(distribution_schedule)`
- `update_gov(gov)`
- `set_contract_whitelist(address, enabled)`
- `migrate_staking(new_staking_contract)`

Users

Users participating in staking can interact with the following functions:

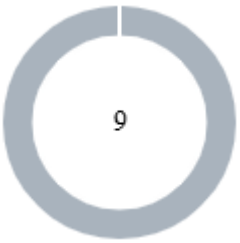
- `bond(sender_addr, amount)`
- `unbond(amount)`
- `withdraw()`

Retrieval Functions

The following functions can be used to retrieve staking-related information:

- `query_config()`
- `query_state(block_time)`
- `query_staker_info(staker, block_time)`
- `query_gov()`

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	0	9	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PRDM	Potential Reward Distribution Manipulation	Acknowledged
●	COC	Commented Out Code	Acknowledged
●	CCR	Contract Centralization Risk	Acknowledged
●	PLNCE	Potential Large Number Conversion Error	Acknowledged
●	PMM	Potential Migration Manipulation	Acknowledged
●	PPLOF	Potential Permanent Loss of Funds	Acknowledged
●	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
●	RF	Redundant Functionality	Acknowledged
●	TSI	Tokens Sufficiency Insurance	Acknowledged

PRDM - Potential Reward Distribution Manipulation

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L239,482
Status	Acknowledged

Description

The contract allows governance to update the reward distribution schedule at any time. This means governance can change the reward rate, including reducing future rewards or modifying the timeline for distribution. This could lead to unfair outcomes, especially if governance lowers the rewards right before stakers are set to claim them. Such unrestricted control over reward distribution creates trust issues and potential financial losses for stakers.

```
pub fn update_config(
    deps: DepsMut,
    _env: Env,
    info: MessageInfo,
    distribution_schedule: Vec<(u64, u64, Uint128)>,
) -> StdResult<Response> {
    //...
    assert_new_schedules(&config, &state,
distribution_schedule.clone())?;

    let new_config = Config {
        neon_token: config.neon_token,
        staking_token: config.staking_token,
        distribution_schedule,
    };
    store_config(deps.storage, &new_config)?;
    Ok(Response::new().add_attributes(vec![("action",
"update_config")]))
}

pub fn assert_new_schedules(
    config: &Config,
    state: &State,
    distribution_schedule: Vec<(u64, u64, Uint128)>,
) -> StdResult<()> {
    let mut existing_counts: BTreeMap<(u64, u64, Uint128), u32>
= BTreeMap::new();
    for schedule in config.distribution_schedule.clone() {
        let counter =
existing_counts.entry(schedule).or_insert(0);
        *counter += 1;
    }
    let mut new_counts: BTreeMap<(u64, u64, Uint128), u32> =
BTreeMap::new();
    for schedule in distribution_schedule {
        let counter = new_counts.entry(schedule).or_insert(0);
        *counter += 1;
    }
    Ok(())
}
```

Recommendation

To protect stakers from unexpected reductions in rewards, the contract should enforce constraints on how governance can modify the reward schedule. This can be done by checking for already active reward periods to prevent modifications.

The team should carefully manage the private keys of the governance's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract administrative functions.

The team may consider:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

COC - Commented Out Code

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L487,505
Status	Acknowledged

Description

In `assert_new_schedules` function there are code segments that are commented out. Commented out code reduces code readability.

```
// if distribution_schedule.len() <
config.distribution_schedule.len() {
//     return Err(StdError::generic_err(
//         "cannot update; the new schedule must support all of
the previous schedule",
//     ));
// }
//...
// for (schedule, count) in existing_counts.into_iter() {
//...
// for (schedule, count) in new_counts.into_iter() {
//     if count > 0 && schedule.0 <= state.last_distributed {
//         return Err(StdError::generic_err(
//             "new schedule adds an already started
distribution",
//         ));
//     }
// }
```

Recommendation

It is recommended to remove commented code to enhance code readability.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L221,251,265,285
Status	Acknowledged

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
pub fn update_config(**params**) -> {
    //...
    if sender_addr_raw !=
deps.api.addr_canonicalize(&gov_addr)? {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
pub fn update_gov(**params**) -> {
    //...
    if sender_addr_raw !=
deps.api.addr_canonicalize(&gov_addr)? {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
pub fn set_contract_whitelist(**params**) -> {
    //...
    if sender_addr_raw !=
deps.api.addr_canonicalize(&gov_addr)? {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
pub fn migrate_staking(**params**) -> {
    //...
    if sender_addr_raw !=
deps.api.addr_canonicalize(&gov_addr)? {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

PLNCE - Potential Large Number Conversion Error

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L4011
Status	Acknowledged

Description

The contract calculates a staker's pending rewards using 256-bit arithmetic for higher precision, but eventually converts the result back into a 128-bit integer using `Uint128::from_str(&pending_reward.to_string())`. If the computed reward exceeds the maximum value that a `uint128` can hold, this conversion will fail and trigger an error. As a result, the entire transaction will revert, preventing the user from updating their staking state or claiming rewards.

```
fn compute_staker_reward(state: &State, staker_info: &mut
StakerInfo) -> StdResult<()> {
    //...
    staker_info.pending_reward +=
Uint128::from_str(&pending_reward.to_string())?;
    Ok(())
}
```

Recommendation

It is recommended that the contract includes validation before attempting to convert large `uint256` reward values into `uint128`.

PMM - Potential Migration Manipulation

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L285
Status	Acknowledged

Description

The contract allows governance to migrate the staking contract, effectively moving all staked funds and rewards to a new contract. While this is intended for upgrading or improving staking mechanisms, it also means that governance has the power to halt rewards and modify staking rules at will. This creates a centralization risk, as users who have already staked may lose access to expected rewards.

```
pub fn migrate_staking(
    deps: DepsMut,
    env: Env,
    info: MessageInfo,
    new_staking_contract: String,
) -> StdResult<Response> {
    //...
    Ok(Response::new()
        .add_messages(vec![CosmosMsg::Wasm(WasmMsg::Execute {
            contract_addr: neon_token.to_string(),
            msg: to_binary(&Cw20ExecuteMsg::Transfer {
                recipient: new_staking_contract,
                amount: remaining_anc,
            })?,
            funds: vec![],
        })])
        .add_attributes(vec![
            ("action", "migrate_staking"),
            ("distributed_amount",
&distributed_amount.to_string()),
            ("remaining_amount", &remaining_anc.to_string()),
        ]))
}
```

Recommendation

The team should carefully manage the private keys of the governance. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Possible Solutions:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the governance, which will eliminate the threats but it is non-reversible.

PPLOF - Potential Permanent Loss of Funds

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L109
Status	Acknowledged

Description

It is possible that tokens are sent to the contract without issuing the `Bond` hook. This can happen if the `transfer` is used instead of `send`. These tokens have no way of being retrieved and they are permanently stacked inside the contract unless migrated.

```
pub fn receive_cw20(
    **args**
    cw20_msg: Cw20ReceiveMsg,
) -> StdResult<Response> {
    //...
    match from_binary(&cw20_msg.msg) {
        Ok(Cw20HookMsg::Bond {}) => {
            //..
            if config.staking_token !=
deps.api.addr_canonicalize(info.sender.as_str())? {
                return Err(StdError::generic_err("unauthorized
token"));
            }
            let cw20_sender =
deps.api.addr_validate(&cw20_msg.sender)?;
            bond(deps, env, cw20_sender, cw20_msg.amount)
        }
        Err(_) => Err(StdError::generic_err("data should be
given")),
    }
}
```

Recommendation

It is recommended that the team considers the possibility of tokens being transferred in the contract without the use of `send` with the proper `Bond` hook. The team could implement a governance-controlled recovery function that allows the retrieval of untracked tokens.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L109,359
Status	Acknowledged

Description

The CW20 tokens have functionality that enable transfers of specified amounts of tokens to an address. The fee or tax is an amount that is charged to the sender of a CW20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

In this case the contract may expect a certain amount of but receive less. The contract does not account for potential fees.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
fn increase_bond_amount(state: &mut State, staker_info: &mut
StakerInfo, amount: Uint128) {
    state.total_bond_amount += amount;
    staker_info.bond_amount += amount;
}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that a CW20 transfer tax is not a standard feature of the CW20 specification, and it is not universally implemented by all CW20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

RF - Redundant Functionality

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L482
Status	Acknowledged

Description

The contract uses the `assert_new_schedules` in the `update_config` function but it does not provide any checks for the schedules. Therefore it is redundant.

```
pub fn assert_new_schedules(
    config: &Config,
    state: &State,
    distribution_schedule: Vec<(u64, u64, Uint128)>,
) -> StdResult<()> {
    let mut existing_counts: BTreeMap<(u64, u64, Uint128), u32>
= BTreeMap::new();
    for schedule in config.distribution_schedule.clone() {
        let counter =
existing_counts.entry(schedule).or_insert(0);
        *counter += 1;
    }

    let mut new_counts: BTreeMap<(u64, u64, Uint128), u32> =
BTreeMap::new();
    for schedule in distribution_schedule {
        let counter = new_counts.entry(schedule).or_insert(0);
        *counter += 1;
    }
    Ok(())
}
```

Recommendation

It is recommended to remove redundant code to increase the contracts readability and general optimization.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	neonswap_staking/src/contract.rs#L180,188,189,208
Status	Acknowledged

Description

The reward tokens must be provided externally, creating a dependency on the administrator’s actions, which introduces centralization risks and potential issues. For example, stakers can stake their tokens while the contract holds no rewards, leaving them reliant on the administrator to supply the rewards. If the administrator fails to do so, stakers will not receive any rewards, undermining the reliability of the system.

```
pub fn withdraw(deps: DepsMut, env: Env, info: MessageInfo) ->
StdResult<Response> {
    //...
    // Compute global reward & staker reward
    compute_reward(&config, &mut state,
env.block.time.seconds());
    compute_staker_reward(&state, &mut staker_info)?;
    let amount = staker_info.pending_reward;
    staker_info.pending_reward = Uint128::zero();
    //...
    // Store updated state
    store_state(deps.storage, &state)?;
    Ok(Response::new()
        .add_messages(vec![CosmosMsg::Wasm(WasmMsg::Execute {
            contract_addr:
deps.api.addr_humanize(&config.neon_token)?.to_string(),
            msg: to_binary(&Cw20ExecuteMsg::Transfer {
                recipient: info.sender.to_string(),
                amount,
            })?,
            funds: vec![],
        })])
        .add_attributes(vec![
            ("action", "withdraw"),
            ("owner", info.sender.as_str()),
            ("amount", amount.to_string().as_str()),
        ]))
}
```

Recommendation

It is recommended to consider enabling staking after tokens have been added to the contract. If the contract guarantees that rewards are available it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

Summary

NeonSwap contract implements a staking mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io