Cyberscope

Audit Report

NeonSwap

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_factory/src/contract.rs	72b91ca03e648b07d85d366cc50cf2dc41465b4a855e2b08fa2f984698919020
neoswap_factory/src/lib.rs	81a4747a4fb04ce30155b4a2275ee04b1abb4637afb4f7626ae82b4961a842f1
neoswap_factory/src/response.rs	a818010e387f8a52a2589d427d55b856729f5117d9e48cdeca7916171d4aa272
neoswap_factory/src/state.rs	371f02e0790181f16bea82aa7680d4e2502566708d87e9a6f6cb4a18cf18ae2e
neoswap_pair/src/contract.rs	dadb20d3ac5cdd016c470aac7ca74a70e292fc82a592df7c95f0bad8211f093c
neoswap_pair/src/lib.rs	09903627c9489dff2c42c29c81db2924415f6772aa8038f997123a616329b9bc
neoswap_pair/src/response.rs	a818010e387f8a52a2589d427d55b856729f5117d9e48cdeca7916171d4aa272
neoswap_pair/src/state.rs	1b5a6eeb1c76088d8859c19c054a6d02379bfe99fcc3245018575854506e6271

neoswap_pair/src/errors.rs	078978e7ccc7c1afaabdd79868af01a0f46e4d12cbbaa2bfbf328d253854b4df
neoswap_router/src/contract.rs	d035f730e2842f4165a70919d6ff297e9e1eff553b05fd7b1aac4c65ad238025
neoswap_router/src/lib.rs	c3a20cf7cc88d4c9024fdbc14a82a55354dd133fdb75a7957690e4e05b72bf7b
neoswap_router/src/operations.rs	e6147f701e4f536ecb004cd3d44c063f55206ef82b98296f9741f531e25f6e26
neoswap_router/src/state.rs	ab13759f99acee3b8d3d65c5370ee9032997e92dd463acb41f29c5e8f632b30d

Overview

Neonswap Factory Contract

The `Neonswap Factory` contract acts as the central registry and management point for Neonswap trading pairs. It is responsible for initializing the core configuration of the protocol, creating trading pairs, managing native token decimals, and administering key access control features such as whitelisting and withdrawals.

The contract is instantiated with references to a token code ID and a pair code ID, which define the base contracts used for liquidity tokens and trading pairs respectively. Owners of the contract can update these configurations and designate a new owner. Through the `CreatePair` functionality, users can deploy new pair contracts between two distinct assets, ensuring asset compatibility, preventing duplication, and storing temporary pair info for further setup.

Administrative functions enable control over the configuration of existing pairs, registration of native token decimals for accurate asset handling, and controlled withdrawals of native or CW20 tokens from the contract. Whitelisting functionality provides permissioned access to sensitive actions like adding native token decimals. The contract ensures security via strict ownership checks and validation logic. On successful pair creation, the factory automatically triggers the pair's instantiation and liquidity provisioning if initial assets are included.

Neonswap Pair Contract

The `Neonswap Pair` contract represents a single trading pair between two assets. It facilitates decentralized swaps and liquidity provisioning using the constant product market maker model. Upon instantiation, it deploys a custom liquidity token for the pair and initializes asset metadata such as token addresses and decimal information.

The pair contract allows users to provide liquidity by depositing both assets in the correct ratio. The contract calculates and mints LP tokens proportionally, accounting for initial and subsequent liquidity scenarios. For withdrawals, users burn LP tokens to retrieve their share of the pool, with protections in place to prevent front-running or inaccurate token amounts.

Swaps can be performed with either native or CW20 tokens. The contract ensures only the correct token is offered and calculates the return amount, commission, and spread accordingly. A fixed commission rate of 0.3% is applied to all trades, with half of it sent to a predefined fee collector. The contract provides advanced slippage protection by validating belief price and max spread inputs during swaps.

Additionally, the pair contract supports queries for pool state, swap simulations, and administrative updates by the fee collector, including reconfiguring the pair's metadata. It emphasizes mathematical precision and security, implementing safe arithmetic and custom error handling.

Neonswap Router Contract

The `Neonswap Router` contract enables complex swap routing logic, allowing users to execute multi-step trades through a sequence of swap operations. It is instantiated with a reference to the `Neonswap Factory` contract and is designed to streamline token exchanges across different trading pairs on the platform.

The router supports CW20 token transfers and native token operations. It can handle chained swap operations via the `ExecuteSwapOperations` message, where users define a series of swap steps. The contract internally breaks these steps into discrete sub-messages and tracks the final balance to enforce minimum output conditions. It also allows single-swap execution and direct assertions about received amounts after swaps.

For simulations, the router can provide estimates on the amount that will be received or required across multiple swap steps, using `SimulateSwapOperations` and `ReverseSimulateSwapOperations`. This enables frontends or dApps to offer users reliable swap previews.

The router includes validation for swap operation chains to ensure that only a single final output token is targeted. It tightly integrates with the Neonswap Pair and Factory contracts to provide accurate data and maintain trading integrity.

Roles

Neonswap Factory Contract

Owner

The contract owner can interact with the following functions:

- `execute_update_config(...)`
- `execute_admin_config(...)`
- `execute_admin_withdraw_denom(...)`
- `execute_admin_withdraw_token(...)`
- `execute_set_whitelist(...)`
- `execute_migrate_pair(...)`

Whitelisted / Native Token Owners

Whitelisted addresses or native token denom owners can interact with:

- `execute_add_native_token_decimals(...)`

Public Users

Anyone can interact with:

- `execute_create_pair(...)`
- `query_*` : Retrieve contract configuration, pair info, native token decimals, and registered pairs.

Neonswap Pair Contract

Fee Collector

The fee collector can interact with:

- `admin_configure(...)`

Liquidity Providers

Liquidity providers can interact with:

- `provide_liquidity(...)`
- `withdraw_liquidity(...)`

Swappers

Users swapping tokens interact with:

- `swap(...)`
- `receive_cw20(...)`

Retrieval Functions

Anyone can use the following queries:

- `query_pair_info()`
- `query_pool()`
- `query_simulation(...)`
- `query_reverse_simulation(...)`

Neonswap Router Contract

Owner (at initialization)

The owner sets the `neonswap_factory` address during contract instantiation.

Users

Users can interact with:

- `execute_swap_operations(...)`
- `execute_swap_operation(...)`
- `assert_minimum_receive(...)`

CW20 Token Senders

CW20 tokens can interact via hook messages:

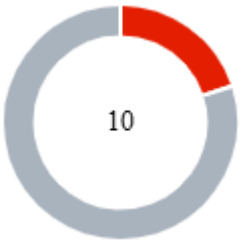
- `receive_cw20(...)`

Retrieval Functions

Anyone can query:

- `query_config()`
- `simulate_swap_operations(...)`
- `reverse_simulate_swap_operations(...)`

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	2	0	0
Medium	0	0	0	0
Minor / Informative	0	8	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	ITBT	Improper Token Balance Tracking	Acknowledged
Critical	PDAI	Potential Decimals Alteration Inconsistencies	Acknowledged
Minor / Informative	MPMVV	Missing Pair Migrate Versioning and Validation	Acknowledged
Minor / Informative	CCR	Contract Centralization Risk	Acknowledged
Minor / Informative	HFC	Hardcoded Fee Collector	Acknowledged
Minor / Informative	MSR	Missing Swap Restrictions	Acknowledged
Minor / Informative	PSU	Potential Subtraction Underflow	Acknowledged
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
Minor / Informative	SCZNS	Slippage Case Zero Not Supported	Acknowledged
Minor / Informative	UTC	Unused Testing Code	Acknowledged

ITBT - Improper Token Balance Tracking

Criticality	Critical
Location	neonswap_pair/src/contract.rs#L293,456,515,624,646,677
Status	Acknowledged

Description

The pair contract calculates the current pool state dynamically by querying token balances at the time of a function call. While this approach simplifies pool state management, it introduces a critical vulnerability: it relies on the actual token balances held by the contract, which can be manipulated via direct `transfer` calls to the contract address.

```
let mut pools: [Asset; 2] =
    pair_info.query_pools(&deps.querier, deps.api,
env.contract.address.clone())?;
```

Recommendation

To mitigate this risk, the contract should explicitly track pool balances in storage and update them only during controlled state-changing operations. This ensures the contract logic operates on verified, internally consistent state rather than relying on external token balances, which are subject to external interference.

Team Update

Issue acknowledged. This works as intended.

PDAI - Potential Decimals Alteration Inconsistencies

Criticality	Critical
Location	neonswap_factory/src/contract.rs#L89 neonswap_pair/src/contract.rs#L148
Status	Acknowledged

Description

The pair contract is able to update the decimals of the tokens used to define the pair contract. Updating the decimals of a token can create inconsistencies during pair calculations especially when liquidity has already been provided to the pair. Additionally the update process involves two separate calls of updating the pair and factory. Since the calls are not being processed in the same transaction there might be inconsistencies in information used externally or by other decentralized applications.

```
// factory
pub fn execute_admin_config(
    deps: DepsMut,
    _env: Env,
    info: MessageInfo,
    asset_infos: [AssetInfo; 2],
    asset_decimals: [u8; 2],
) -> StdResult<Response> {
    let config: Config = CONFIG.load(deps.storage)?;

    // permission check
    if deps.api.addr_canonicalize(info.sender.as_str())? !=
config.owner {
        return Err(StdError::generic_err("unauthorized"));
    }

    let raw_infos = [
        asset_infos[0].to_raw(deps.api)?,
        asset_infos[1].to_raw(deps.api)?,
    ];

    let pair_key = pair_key(&raw_infos);
    let pair_info: PairInfoRaw = PAIRS.load(deps.storage,
&pair_key)?;

    PAIRS.save(
        deps.storage,
        &pair_key,
        &PairInfoRaw {
            liquidity_token: pair_info.liquidity_token,
            contract_addr: pair_info.contract_addr,
            asset_infos: raw_infos,
            asset_decimals,
        },
    )?;

    Ok(Response::new().add_attribute("action",
"update_config"))
}
// pair
pub fn admin_configure(
    deps: DepsMut,
    env: Env,
    info: MessageInfo,
    assets: [AssetInfo; 2],
    asset_decimals: [u8; 2],
) -> Result<Response, ContractError> {
    // permission check
    if info.sender.as_str() != FEE_COLLECTOR {
        return Err(ContractError::Unauthorized {});
```

```
    }

    let config: PairInfoRaw = PAIR_INFO.load(deps.storage)?;

    let pair_info: &PairInfoRaw = &PairInfoRaw {
        contract_addr:
deps.api.addr_canonicalize(env.contract.address.as_str())?,
        liquidity_token: config.liquidity_token,
        asset_infos: [assets[0].to_raw(deps.api)?,
assets[1].to_raw(deps.api)?],
        asset_decimals,
    };

    PAIR_INFO.save(deps.storage, pair_info)?;

    Ok(Response::new().add_attributes(vec![("action",
"admin_configure")]))
}
```

Recommendation

It is recommended to not allow the change of asset decimals inconsistencies may arise during production. This is especially dangerous if liquidity has already been provided.

The team should carefully manage the private keys of authorities' accounts. 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.

Team Update

Issue acknowledged. This works as intended.

MPMVV - Missing Pair Migrate Versioning and Validation

Criticality	Minor / Informative
Location	neonswap_pair/src/contract.rs#L912
Status	Acknowledged

Description

The migrate function in the pair contract lacks any internal permission checks or version validation logic. While CosmWasm enforces that only the contract admin can call migrate, the contract itself does not implement any safeguards, such as validating the migration message, checking the current contract version, or confirming compatibility with the new logic.

Furthermore, the current implementation does not utilize the `migrate_version()` function, which is a standard utility in CosmWasm for safely updating contract versions and ensuring proper version history is maintained.

```
const TARGET_CONTRACT_VERSION: &str = "0.2.0";
#[cfg_attr(not(feature = "library"), entry_point)]
pub fn migrate(deps: DepsMut, _env: Env, _msg: MigrateMsg) ->
Result<Response, ContractError> {
    Ok(Response::default())
}
```

Recommendation

It is recommended to enhance the migrate function with proper safety checks. Specifically: Use `migrate_version()` to track and verify version upgrades. Optionally, validate the incoming `MigrateMsg` for expected structure or constraints.

Team Update

Issue acknowledged. This works as intended.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	neonswap_factory/src/contract.rs#L89,126,238,267,284,307,337,374 neonswap_pair/src/contract.rs#L148
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.

```
// factory
pub fn execute_admin_config(**args**) -> StdResult<Response> {
    //...
    if deps.api.addr_canonicalize(info.sender.as_str())? !=
config.owner {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
pub fn execute_update_config(**args**) -> StdResult<Response> {
    //...
    if deps.api.addr_canonicalize(info.sender.as_str())? !=
config.owner {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
pub fn execute_add_native_token_decimals(**args**) ->
StdResult<Response> {
    //...
    if !is_owner && !is_denom_owner && !is_whitelisted {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
pub fn execute_set_whitelist(**args**) -> StdResult<Response> {
    //...
    if deps.api.addr_canonicalize(info.sender.as_str())? !=
config.owner {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
pub fn execute_admin_withdraw_denom(**args**) ->
StdResult<Response> {
    //...
    if deps.api.addr_canonicalize(info.sender.as_str())? !=
config.owner {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
pub fn execute_admin_withdraw_token(**args**) ->
StdResult<Response> {
    //...
    if deps.api.addr_canonicalize(info.sender.as_str())? !=
config.owner {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
```

```
}
pub fn execute_migrate_pair(**args**) -> StdResult<Response> {
    //...
    if deps.api.addr_canonicalize(info.sender.as_str())? !=
config.owner {
        return Err(StdError::generic_err("unauthorized"));
    }
    //...
}
// pair
pub fn admin_configure(**args**) -> Result<Response,
ContractError> {
    if info.sender.as_str() != FEE_COLLECTOR {
        return Err(ContractError::Unauthorized {});
    }
    //…
}
```

Additionally, in `execute_add_native_token_decimals` whitelisted addresses are able to bypass the denom contract's balance check.

```
if !is_whitelisted {
    if balance.is_zero() {
        return Err(StdError::generic_err(
            "a balance greater than zero is required by the
factory for verification",
        ));
    }
}
```

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.

Team Update

Issue acknowledged. This works as intended.

HFC - Hardcoded Fee Collector

Criticality	Minor / Informative
Location	neonswap_pair/src/contract.rs#L40
Status	Acknowledged

Description

The pair contract has a hardcoded `FEE_COLLECTOR` used to collect the fees implemented in the contract. This `FEE_COLLECTOR` address also has the authority to use admin functions like `admin_configure`. Hardcoding this address may create a problem if it needs to be updated. Furthermore if the intention is that the factory contract should be declared as the `FEE_COLLECTOR` then additional functionality needs to be added in both factory and pair in order to support the functionality of the pair contract.

```
const FEE_COLLECTOR: &str =
"mantra1fcsp67yecrxygtq0urtxaquua3u9rgksmuse8z";
```

Recommendation

The team should consider declaring the `FEE_COLLECTOR` when the contract is instantiated. Additionally if the intention is to declare the factory contract as `FEE_COLLECTOR` the team should adjust both contract's functionality so that the factory contract supports the functionality of the pair contract.

Team Update

Issue acknowledged. This works as intended.

MSR - Missing Swap Restrictions

Criticality	Minor / Informative
Location	neonswap_pair/src/contract.rs#L498
Status	Acknowledged

Description

In the current configuration of the pair contract there is no restriction about who can call the `swap` function. This allows users to directly use `swap` instead of first going through the router contract. Direct swap with the pair may create inconsistencies during calculations if token swapped implements fees since the pair does not account for them.

```
#[allow(clippy::too_many_arguments)]
pub fn swap(
    deps: DepsMut,
    env: Env,
    info: MessageInfo,
    sender: Addr,
    offer_asset: Asset,
    belief_price: Option<Decimal>,
    max_spread: Option<Decimal>,
    to: Option<Addr>,
    deadline: Option<u64>,
) -> Result<Response, ContractError> {
    //…
}
```

Recommendation

The team is advised to consider the possibility of allowing swaps only through the router contract.

Team Update

Issue acknowledged. This works as intended.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	neonswap_pair/src/contract.rs#L383
Status	Acknowledged

Description

The contract subtracts two values, the second value may be greater than the first value. As a result, the subtraction may underflow and cause the execution to revert.

```
remain_amount = deposits[i] - desired_amount;
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

Team Update

Issue acknowledged. This works as intended.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	neonswap_pair/src/contract.rs#L402,546
Status	Acknowledged

Description

The `transferFrom()` function is used to transfer a specified amount 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.

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

```
else if let AssetInfo::Token { contract_addr, .. } = &pool.info
{
    messages.push(CosmosMsg::Wasm(WasmMsg::Execute {
        contract_addr: contract_addr.to_string(),
        msg: to_binary(&Cw20ExecuteMsg::TransferFrom {
            owner: info.sender.to_string(),
            recipient: env.contract.address.to_string(),
            amount: desired_amount,
        })?,
        funds: vec![],
    }));
}
```

The case is similar for the `swap` function if users do direct swaps with the pair instead of using router. The pair uses `offer_asset.amount` directly trusting that the tokens will not have any fees.

```
let offer_amount = offer_asset.amount;
let (return_amount, spread_amount, commission_amount) =
compute_swap(offer_pool.amount, ask_pool.amount,
offer_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`

Team Update

Issue acknowledged. This works as intended.

SCZNS - Slippage Case Zero Not Supported

Criticality	Minor / Informative
Location	neonswap_pair/src/contract.rs#L384
Status	Acknowledged

Description

In `provide_liquidity` function the `slippage_tolerance` is used to ensure that the difference between the user's deposited amount and the amount actually used in the liquidity pool does not exceed a user-defined threshold. However if the user adds zero as `slippage_tolerance` the `remain_amount` will always be bigger and the function will return an error.

```
if let Some(slippage_tolerance) = slippage_tolerance {
    if remain_amount > deposits[i] * slippage_tolerance {
        return Err(ContractError::MaxSlippageAssertion {});
    }
}
```

Recommendation

It is recommended that the team handles the possibility of users adding zero `slippage_tolerance` .

Team Update

Issue acknowledged. This works as intended.

UTC - Unused Testing Code

Criticality	Minor / Informative
Location	neonswap_factory/src/state.rs#L83 neonswap_pair/src/contract.rs#L732 neonswap_router/src/contract.rs#L371
Status	Acknowledged

Description

The contracts have testing code left in the scripts. While very useful for testing purposes it decreases code readability.

```
#[cfg(test)]
mod allow_native_token {
    //...
}

#[test]
fn test_compute_swap_with_huge_pool_variance() {
    let offer_pool = Uint128::from(395451850234u128);
    let ask_pool = Uint128::from(317u128);

    assert_eq!(
        compute_swap(offer_pool, ask_pool,
Uint128::from(1u128))
            .unwrap()
            .0,
        Uint128::zero()
    );
}

#[test]
fn test_invalid_operations() {
    // empty error
    assert!(assert_operations(&[]).is_err());

    // uluna output
    assert!(assert_operations(&[
        SwapOperation::SpiritSwap {
            offer_asset_info: AssetInfo::NativeToken {
                denom: "ukrw".to_string(),
            },
            ask_asset_info: AssetInfo::Token {
                contract_addr: "asset0001".to_string(),
            },
        },
        SwapOperation::SpiritSwap {
            offer_asset_info: AssetInfo::Token {
                contract_addr: "asset0001".to_string(),
            },
            ask_asset_info: AssetInfo::NativeToken {
                denom: "uluna".to_string(),
            },
        }
    ])
    .is_ok());

    // asset0002 output
    assert!(assert_operations(&[
        SwapOperation::SpiritSwap {
            offer_asset_info: AssetInfo::NativeToken {
                denom: "ukrw".to_string(),
```

```
                },
                ask_asset_info: AssetInfo::Token {
                    contract_addr: "asset0001".to_string(),
                },
            },
            SwapOperation::SpiritSwap {
                offer_asset_info: AssetInfo::Token {
                    contract_addr: "asset0001".to_string(),
                },
                ask_asset_info: AssetInfo::NativeToken {
                    denom: "uluna".to_string(),
                },
            },
            SwapOperation::SpiritSwap {
                offer_asset_info: AssetInfo::NativeToken {
                    denom: "uluna".to_string(),
                },
                ask_asset_info: AssetInfo::Token {
                    contract_addr: "asset0002".to_string(),
                },
            },
        ])
        .is_ok());
}
```

Recommendation

It is recommended to remove code that is not used in production to enhance code readability.

Team Update

Issue acknowledged. This works as intended.

Summary

NeonSwap contract implements an exchange 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