SCV SECURITY

AUDIT REPORT

Hydro Protocol

Loop Staking Contracts

Prepared by SCV-Security

On 17th November 2024

Table of Contents

Introduction

SCV has been engaged by Hydro Protocol to conduct a comprehensive security review with the goal of identifying potential security threats and vulnerabilities within the codebase. The purpose of this audit is to evaluate the security posture of the codebase and provide actionable recommendations to mitigate any identified risks. This report presents an overview of the findings from our security audit, outlining areas of concern and proposing effective measures to enhance the codebase's security.

Scope Functionality

The Hydro Loop Staking contracts comprise a leveraged farming protocol built on Injective. The protocol makes use of Neptune Finance, DojoSwap, and hINJ.

Users onboard to the protocol by staking INJ to the lending vault and receive a share token that corresponds to their share of the vault. The protocol then maximizes the lending vault yield by either minting an hINJ liquid staked derivative directly from the staked INJ or swapping the staked INJ for hINJ, depositing the hINJ as collateral on Neptune, and borrowing INJ from Neptune.

Users offboard from the protocol by sending their share token back to the lending vault. The protocol then repays the user's share of borrowed INJ back to Neptune, withdraws the hINJ collateral from Neptune, swaps the hINJ back to INJ, and sends the user their corresponding INJ vault share.

An off-chain Health Care Bot is utilized to recurrently calculate the health of the protocol and initiate rebalances by adjusting leveraging to maximize profit and avoid liquidation.

Submitted Codebase

loop-staking-contracts	
Repository	https://github.com/hydro-protocol/loop-staking-contracts
Commit	b7d1cc7f3c5a8d85e60a0da25ab78384a1f5f561
Contracts	fixed_swap, lending_vault, lending_vault_swap_proxy, lsd_mint_dojo_pair_swap_proxy, share_token
Branch	main

Revisions - Submitted Codebase

loop-staking-contracts	
Repository	https://github.com/hydro-protocol/loop-staking-contracts
Commit	b103da2e50051ce4b0a1f47cfc16cdb5961b6eb9
Contracts	fixed_swap, lending_vault, lending_vault_swap_proxy, lsd_mint_dojo_pair_swap_proxy, share_token
Branch	fix/audit

Methodologies

SCV performs a combination of automated and manual security testing based on the scope of testing. The testing performed is based on the extensive experience and knowledge of the auditor to provide the greatest coverage and value to Hydro Protocol Testing includes, but is not limited to, the following:

- Understanding the application and its functionality purpose.
- Deploying SCV in-house tooling to automate dependency analysis and static code review.
- Analyse each line of the code base and inspect application security perimeter.
- Review underlying infrastructure technologies and supply chain security posture.

Code Criteria

This section provides an evaluation of specific criteria aspects as described below:

- **Documentation:** Evaluating the presence and comprehensiveness of publicly available or provided explanatory information, diagram flowcharts, comments, and supporting documents to enhance code understanding.
- **Coverage:** Evaluating whether the code adequately addresses all necessary cases and scenarios, ensuring that the intended functionality or requirements are sufficiently covered.
- **Readability:** Assessing how easily the code can be understood and maintained, considering factors such as code structure, naming conventions, and overall organisation.
- **Complexity:** Evaluating the complexity of the code, including factors such as, number of lines, conditional statements, and nested structures.

The status of each criteria is categorised as either **SUFFICIENT** or **NOT-SUFFICIENT** based on the audit assessment. This categorisation provides insights to identify areas that may require further attention and improvement.

Criteria	Status	Notes
Documentation	**SUFFICIENT**	The team provided a technical document and flowchart diagrams as documentation.
Coverage	**NOT-SUFFICIENT**	There are no test cases in the codebase.
Readability	**SUFFICIENT**	N/A
Complexity	**SUFFICIENT**	The codebase integrates with Neptune Finance, DojoSwap, and hINJ. However, it is worth noting that Neptune Finance is a private repo at the time of writing.

Findings Summary

Summary Title	Risk Impact	Status
Price cache is not removed when the `max_action_count` threshold is reached	**CRITICAL**	**RESOLVED**
Share inflation attack can be triggered to steal funds	**SEVERE**	**ACKNOWLEDGED**
Sandwich attack possibility due to infinite slippage	**SEVERE**	**RESOLVED**
Slippage protection is disabled when fixed swaps fail	**SEVERE**	**RESOLVED**
Updating collateral and debt denom causes previous tokens to be stuck	**MODERATE**	**RESOLVED**
Share token configuration cannot be updated	**MODERATE**	**RESOLVED**
Registering and removing pairs does not account for reverse ordering	**LOW**	**RESOLVED**
Missing validations for liquidity deposits into the `fixed_swap` contract	**LOW**	**RESOLVED**
MarketContract queries all markets and collaterals	**LOW**	**RESOLVED**
Missing contract configuration validations	**LOW**	**RESOLVED**
Contracts can be migrated to previous versions	**INFO**	**RESOLVED**
Two-step ownership transfer is not implemented	**INFO**	**RESOLVED**
Typo in error messages	**INFO**	**RESOLVED**

Audit Observations

The audit observations section is intended to present potential findings that are related to the underlying design of the protocol and would require underlying design changes to remediate that may change the overall functioning of the protocol. SCV asks that the client formulate responses to add context to validate or invalidate the following concerns.

1. Potential stale prices from Neptune price oracle

Revision Notes

The team advised that the liquidator for Neptune uses the same price oracle contracts, ensuring consistency. Stale prices are not a concern as these prices are only used to assess the care contract's health.

–

In `contracts/lending_vault/src/states.rs:48-53`, the `PriceCache::new()` function queries the debt and collateral asset information from the Neptune price oracle contract. However, since the Neptune Finance codebase is a closed source (by the time of the audit), we cannot verify whether the oracle contract automatically checks for stale prices. Using stale prices may cause severe consequences, which attackers can weaponize to steal funds from the protocol.

Potential Remediation:

Consider verifying with the Neptune team whether stale prices are validated in the oracle price contract. If not, the function should implement validations to ensure the price is fresh according to Pyth's best practices documentation.

2. Potential liquidations from Health Care Bot inactivity

Revision Notes

The team advised that multiple health-care bots will be set up, each connecting to different Injective nodes with varying polling times. Additionally, bot malfunctions will be monitored with an emergency call system for quick response.

–

The Health Care Bot is a critical off-chain entity that is used to recurrently calculate the health of the protocol. If the protocol health score is below a certain threshold, then the Health Care Bot is responsible for initiating rebalances that adjust leveraging to maximize protocol profit and avoid liquidation. Since the Health Care Bot is an off-chain entity, the team should ensure that there are safeguards in place to guarantee bot liveness and redundancy.

Potential Remediation:

Consider using multiple bots that operate independently of each other in case one bot goes down or malfunctions and setting up alerting systems for protocol administrators to monitor bot health and manually initiate rebalances if necessary.

Findings Technical Details

1. Price cache is not removed when the `max_action_count` threshold is reached

RISK IMPACT: CRITICAL	STATUS: RESOLVED

Description

In `contracts/lending_vault/src/executions/rebalance.rs:23-33`, the `rebalance` function loads the `ActionContext` from the state and checks if the current action count is equal to or larger than the `config.max_action_count`. If yes, the `ActionContext` is removed from the state, and an `Ok(response)` will be returned. This mechanism prevents the transaction from failing due to an out-of-gas error when dispatching messages.

The issue is that the `PriceCache` is not removed from the state when finalizing the transaction with `Ok(response)`. In various entry points, the `PriceCache` is queried at the beginning of the transaction and stored in the state (e.g., `contracts/lending_vault/src/states.rs:69-71`) so the price can be reused throughout the transaction instead of querying the oracle contract every time. This approach reduces gas consumption for complex operations during the rebalance.

However, this also means that the `PriceCache` must be removed from the state at the end of the transaction. Failing to do this would result in the old `PriceCache` being left in storage, causing future messages to incorrectly use the old prices instead of the correct ones from the oracle contract (e.g., `contracts/lending_vault/src/states.rs:66-67`). This may allow an attacker to stake funds with a higher price than intended (thus minting more shares) or unstaking more funds than intended, resulting in a loss of funds.

Recommendation

Consider calling `PriceCache::remove(deps.storage)` when the `config.max_action_count` threshold is reached.

2. Share inflation attack can be triggered to steal funds

RISK IMPACT: SEVERE	**STATUS: ACKNOWLEDGED**

Revision Notes

The team advised that variable names were renamed for simplicity and readability. Additionally, further mitigations were implemented to prevent zero-share returns, enhancing the contract's security and clarity.

Description

In `contracts/lending_vault/src/executions/stake.rs:84-88`, the `stake_hook` function computes the `share_per_value` when minting shares to the user. However, attackers can purposely inflate the `filled_capacity_amount` value by transferring tokens directly to the contract, as the filled capacity includes the contract's balance (see `contracts/lending_vault/src/loan_state.rs:296`).

This allows attackers to carry out a share inflation attack to steal funds from the depositors:

1. The attacker becomes the first depositor and mints 1 share in the smallest unit possible. This causes the `total_share` and `filled_capacity_amount` to become 1.
2. The victim wants to deposit `10_000` assets to the contract.
3. The attacker notices it and front-runs the victim's transaction to transfer `10_000` assets to the contract.
4. When the victim transaction is executed, the contract will have `total_share` as 1 and `filled_capacity_amount` as `10_001`. This will cause the `share_per_value` variable to be `0.99990001` (`1/10_001*10_000`). However, as integer truncation will occur, the computation will result in `0` shares for the victim. This means the victim will receive nothing in return for their `10_000` assets.
5. The attacker unstakes all their shares to withdraw their deposited asset and the victim's asset, earning a profit of `10_000`.

6. The attacker repeats the attack for future depositors, continuously stealing funds from the protocol.

Recommendation

Consider minting “dead shares” to the contract when the total shares supply is zero.

3. Sandwich attack possibility due to infinite slippage

RISK IMPACT: SEVERE	**STATUS: RESOLVED**

Description

In several instances of the codebase, the `min_receive_amount` parameter is hardcoded to `None` during asset swaps. This is problematic because it allows infinite slippage when swapping assets, enabling attackers to steal funds via a sandwich attack.

- `contracts/fixed_swap/src/executions/swap.rs:112-119`
- `contracts/lending_vault/src/executions/deposit_collateral.rs:67-74`
- `contracts/lending_vault/src/replies/withdraw_collateral.rs:31-38`

Recommendation

Consider introducing a `min_receive` parameter for the caller to restrict the minimum receive amount in the above entry points: `Cw20HookMsg::Swap`, `ExecuteMsg::Swap`, `ExecuteMsg::DepositCollateral`, and `ExecuteMsg::WithdrawCollateral`.

4. Slippage protection is disabled when fixed swaps fail

RISK IMPACT: SEVERE	STATUS: RESOLVED

Description

In `contracts/lending_vault_swap_proxy/src/executions/swap_exact_output.rs:122-143`, the `swap_exact_output_hook` function assigns the swap operation to the fixed swap contract if the source and target denom is `hinj` or `inj`. The swap message is dispatched as `reply_on_error`, which means the `REPLY_FIXED_SWAP_FAILED` reply entry point will be entered if the fixed swap fails.

To prevent users from being sandwiched, slippage is handled via the `target_receive_amount` parameter, supplied from the `SwapExactOutput` message and passed to the `SwapExactOutputHook` message (e.g., `contracts/lending_vault_swap_proxy/src/executions/swap_exact_output.rs:44`). The `SwapExactOutputHook` message is responsible for ensuring that the `target_receive_amount` slippage parameter is satisfied throughout the transaction.

The issue is that if the fixed swap fails, the `REPLY_FIXED_SWAP_FAILED` reply handler will perform the swap via the swap proxy contract in `contracts/lending_vault_swap_proxy/src/replies/reply_fixed_swap_failed.rs:16-28` with the `min_receive_amount` parameter is set to `None`. This means that slippage protection is essentially ignored, allowing attackers to steal funds via a sandwich attack.

Recommendation

Consider setting the `min_receive_amount` parameter to `target_receive_amount` in the `swap_exact_output_by_native` and `swap_exact_output_by_cw20` functions.

5. Updating collateral and debt denom causes previous tokens to be stuck

RISK IMPACT: MODERATE	**STATUS: RESOLVED**

Description

In `contracts/lending_vault/src/executions/update_config.rs:49-52` and `59-62`, the contract admin has the ability to update the collateral and debt denom state values in the contract. This is problematic because updating these values would result in the previous collateral and debt denoms being stuck in the contract and left out.

Additionally, if liquidity providers decide to redeem shares after the admin has updated the tokens, they will receive different tokens from those they provided before, likely at a lesser value.

Recommendation

Consider removing the functionality for contract admin to update the collateral and debt denom.

6. Share token configuration cannot be updated

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

In `contracts/share_token/src/entrypoints.rs:34`, the `instantiate` function sets the `config.admin` to `info.sender`. This is problematic because the instantiator is the lending vault contract (`contracts/lending_vault/src/entrypoints.rs:61-68`), and it does not implement any entry points to call into the `UpdateConfig` message in `contracts/share_token/src/entrypoints.rs:134`.

As a result, the share token contract configurations cannot be updated.

Recommendation

Consider implementing a privileged entry point in the lending vault contract to update configurations in the share token contract.

7. Registering and removing pairs does not account for reverse ordering

RISK IMPACT: LOW	STATUS: RESOLVED

Description

In `contracts/lsd_mint_dojo_pair_swap_proxy/src/states.rs:75-81`, the `Pair::may_load` function is called to check whether existing pairs with (base denom, quote denom) and (quote denom, base denom) ordering is configured. This allows the pair address to be queried even if the denoms were provided in a different ordering, increasing efficiency.

However, when a pair address with (base denom, quote denom) is added, the reverse ordering is not automatically added in the `Pair::save` function. This is needed because the contract admin will be unable to add the reverse ordering due to the validation in `contracts/lsd_mint_dojo_pair_swap_proxy/src/executions/custom.rs:47`. This prevents pairs with the reverse ordering to be added, preventing the swap proxy from working as intended.

Similarly, this issue is present in the `remove_pair` function, as the (quote denom, base denom) ordering is not automatically removed from the `Pair::remove` function.

Recommendation

Consider automatically adding and removing (quote denom, base denom) ordering in the `Pair::save` and `Pair::remove` functions.

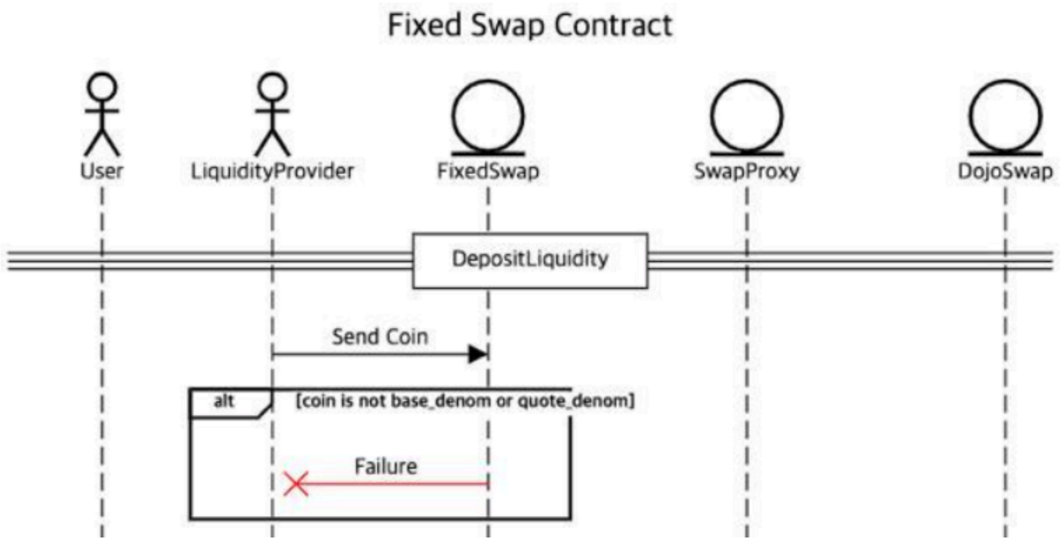
8. Missing validations for liquidity deposits into the `fixed_swap` contract

RISK IMPACT: LOW	STATUS: RESOLVED

Description

In the provided `fixed_swap` contract flowchart, any liquidity deposits that do not contain the base denomination or quote denomination are expected to fail when depositing.

However, there is no explicit message type for liquidity providers to deposit funds into the `fixed_swap` contract. This means that if a liquidity provider sends a coin that is not the specified `base_denom` or `quote_denom` to the contract, then it cannot be used for user swaps.



Recommendation

Consider adding a new `DepositLiquidity` message type to perform the denomination validations prior to accepting the liquidity deposit in the `fixed_swap` contract.

9. MarketContract queries all markets and collaterals

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The lending_vault contract uses a LoanState struct to store the current debt, collateral, and balance of the protocol. When creating a new LoanState struct, the debt info and collateral info for a particular asset are determined by querying the Neptune MarketContract. Unfortunately, the public Neptune API only explicitly supports querying all markets and collaterals and does not support querying for a specific asset instead.

In packages/neptune/src/contracts/market_contract.rs:39-65 and packages/neptune/src/contracts/market_contract.rs:67-93, all markets and collaterals are queried from the Neptune MarketContract, and then the market or collateral for a specific asset is extracted to build the LoanState.

However, there could be a significantly large amount of Neptune markets and collaterals to loop over that could potentially increase gas costs for the user.

Recommendation

Consider breaking from the Neptune market and collateral query loop early if the market or collateral for the expected asset_info is found. This will save on gas by avoiding appending the new query results if the desired market or collateral has already been found for the asset.

Alternatively, consider checking with the Neptune team to see if there are any query endpoints not included in the public Neptune API that could be used to query for a specific market or collateral by asset info instead of needing to query for and loop over all markets and collaterals.

10. Missing contract configuration validations

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The below contracts contain a few missing validations on configuration parameters that could lead to the protocol behaving in unexpected or undesirable ways:

- `lending_vault`
    - Validate that `admins` is not empty.
    - Validate that `max_action_count` is greater than the minimum action count required for all user-initiated messages to be successful.
- `fixed_swap`
    - Validate that `admins` is not empty.
    - Validate that `liquidity_providers` is not empty.
    - Validate that `swap_whitelist` is not empty.
- `lending_vault_swap_proxy`
    - Validate that `admins` is not empty.
- `lsd_mint_dojo_pair_swap_proxy`
    - Validate that `admins` is not empty.

Recommendation

Consider adding the above validations to the configuration parameters whenever instantiating a new contract or updating the corresponding contract configuration.

11. Contracts can be migrated to previous versions

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

In `contracts/fixed_swap/src/entrypoints.rs:146-150`, `contracts/lending_vault/src/entrypoints.rs:191-195`, `contracts/lending_vault_swap_proxy/src/entrypoints.rs:176-180`, `contracts/lsd_mint_dojo_pair_swap_proxy/src/entrypoints.rs:167-171`, and `contracts/share_token/src/entrypoints.rs:211-215`, a migration validation is done to ensure that the new version of the contract to migrate is not the same as the current version.

However, this validation does not account for contracts being incorrectly migrated to a previous contract version. This could cause potential discrepancies in which contract version is the latest and most secure version.

Recommendation

Consider enhancing the migration validations to enforce semver versioning and only allowing contracts to be migrated to future versions.

12. Two-step ownership transfer is not implemented

RISK IMPACT: INFORMATIONAL	**STATUS:** RESOLVED

Description

The codebase does not implement a two-step ownership transfer in the following instances:

- `contracts/fixed_swap/src/executions/update_config.rs:29-32`
- `contracts/lending_vault/src/executions/update_config.rs:35-42`
- `contracts/lending_vault_swap_proxy/src/executions/update_config.rs:19-25`
- `contracts/lsd_mint_dojo_pair_swap_proxy/src/executions/update_config.rs:22-28`

Using a two-step ownership transfer mechanism helps provide a window of opportunity for the current owner to cancel the transfer if they did not intend to initiate it or if there were any unintended actions.

As a result, the ownership will be lost and cannot be recovered if transferred to an incorrect address that no one owns.

Additionally, the `fixed_swap` contract allows for immediate `liquidity_providers` address updates without a timelock or acknowledgement. If an incorrect liquidity provider is accidentally set, they could immediately execute the `WithdrawLiquidity` message to drain all `fixed_swap` contract funds.

Recommendation

Consider implementing a two-step ownership transfer that proposes a new owner or liquidity provider in the first step and requires the proposed owner or liquidity provider to accept it in the second step.

13. Typo in error messages

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

In `contracts/lending_vault/src/executions/stake.rs:76`, there is a typo in the error message displayed if the staked denomination is not equal to the debt denomination set in the contract config. The current error message reads "*now allowed denom as stake*". However, this should instead read "*not allowed denom as stake*".

Recommendation

Consider updating the user-facing error message for the above case to say "*not allowed denom as stake*" instead.

Document Control

Version	Date	Notes
-	29th October 2024	Security audit commencement date.
0.1	14th November 2024	Initial report with identified findings delivered.
0.5	15th - 17th November 2024	Fixes remediations implemented and reviewed.
1.0	17th November 2024	Audit completed, final report delivered.

Appendices

A. Appendix - Risk assessment methodology

SCV-Security employs a risk assessment methodology to evaluate vulnerabilities and identified issues. This approach involves the analysis of both the LIKELIHOOD of a security incident occurring and the potential IMPACT if such an incident were to happen. For each vulnerability, SCV-Security calculates a risk level on a scale of 5 to 1, where 5 denotes the highest likelihood or impact. Consequently, an overall risk level is derived from combining these two factors, resulting in a value from 10 to 1, with 10 signifying the most elevated level of security risk

Risk Level	Range
CRITICAL	10
SEVERE	From 9 to 8
MODERATE	From 7 to 6
LOW	From 5 to 4
INFORMATIONAL	From 3 to 1

LIKELIHOOD and **IMPACT** would be individually assessed based on the below:

Rate	LIKELIHOOD	IMPACT
5	**Extremely Likely**	Could result in severe and irreparable consequences.
4	**Likely**	May lead to substantial impact or loss.
3	**Possible**	Could cause partial impact or loss on a wide scale.
2	**Unlikely**	Might cause temporary disruptions or losses.
1	**Rare**	Could have minimal or negligible impact.

B. Appendix - Report Disclaimer

THANK YOU FOR CHOOSING

SCV SECURITY

scv.services

contact@scv.services