Security Review Report
NM-0266 Haiko Solver

(July 17, 2024)

Contents

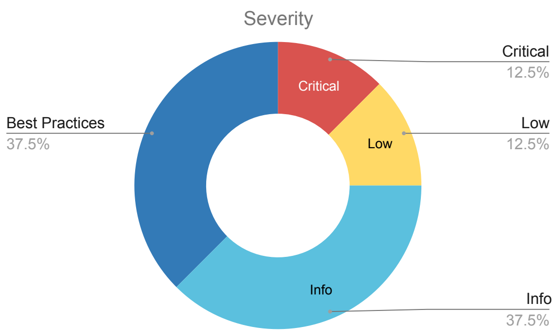
1 Executive Summary

This document outlines the security review conducted by Nethermind Security for the Haiko Solvers. Haiko is a liquidity management protocol that offers strategies to automatically manage liquidity on its AMM. Solvers are a new framework for building liquidity strategies. Solver smart contracts generate quotes and execute swaps on demand without interacting with or relying on an underlying AMM. Instead, solvers directly compute and quote swap amounts and execute trades automatically against depositors' positions. Rather than storing liquidity positions in a state, solvers maintain a stateless architecture, making them more flexible and gas-efficient than regular AMMs.

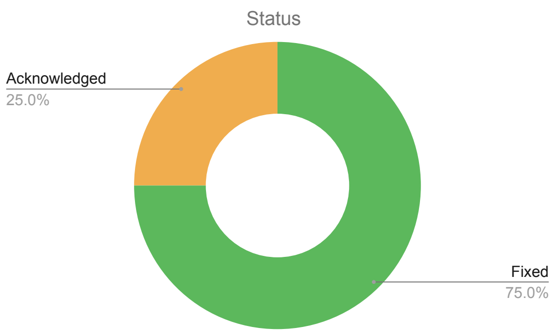
The audited repository contains a Cairo component `SolverComponent` that includes the basic functionality of all Solvers, such as the creation of markets, deposits, withdrawals, and swaps. Additionally, the repository consists of an implementation of a Solver called `ReplicatingSolver`, which contains the logic for calculating quotes for swaps.

The audited code comprises 1640 lines of code in Cairo. The **Haiko** team has provided documentation that explains the purpose and functionality behind audited contracts. The audit was performed using: (a) manual analysis of the codebase, (b) simulation of the smart contracts. **Along this document, we report** 8 points of attention, where one is classified as `Critical`, one is classified as `Low`, and six are classified as `Informational` or `Best Practice`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology adopted for this audit. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.



(a) distribution of issues according to the severity



(b) distribution of issues according to the status

Fig 1: (a) Distribution of issues: Critical (1), **High** (0), **Medium** (0), **Low** (1), **Undetermined** (0), **Informational** (3), **Best Practices** (3). **(b) Distribution of status: Fixed** (6), **Acknowledged** (2), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	July 9, 2024
Final Report	July 17, 2024
Methods	Manual Review
Repository	solver
Commit Hash	a1e28fe6f20811a58adc85dbe78549101b25e463
Final Commit Hash	66e5066729c2338255aadc3cb282b68f41110635
Documentation	README for solver
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	packages/replicating/src/types.cairo	16	25	156.2%	4	45
2	packages/replicating/src/libraries/store_packing.cairo	45	8	17.8%	9	62
3	packages/replicating/src/libraries/swap_lib.cairo	154	69	44.8%	16	239
4	packages/replicating/src/libraries/spread_math.cairo	59	62	105.1%	15	136
5	packages/replicating/src/contracts/replicating_solver.cairo	209	103	49.3%	37	349
6	packages/replicating/src/interfaces/IReplicatingSolver.cairo	14	28	200.0%	7	49
7	packages/replicating/src/interfaces/pragma.cairo	219	34	15.5%	32	285
8	packages/core/src/types.cairo	31	39	125.8%	7	77
9	packages/core/src/libraries/erc20_versioned_call.cairo	42	12	28.6%	1	55
10	packages/core/src/libraries/id.cairo	10	13	130.0%	2	25
11	packages/core/src/libraries/store_packing.cairo	29	8	27.6%	7	44
12	packages/core/src/contracts/vault_token.cairo	62	19	30.6%	15	96
13	packages/core/src/contracts/solver.cairo	680	328	48.2%	108	1116
14	packages/core/src/interfaces/IVaultToken.cairo	6	1	16.7%	1	8
15	packages/core/src/interfaces/ISolver.cairo	64	192	300.0%	32	288
	Total	**1640**	**941**	**57.4%**	**293**	**2874**

3 Summary of Issues

	Finding	Severity	Update
1	Anyone can withdraw from private vaults	Critical	Fixed
2	Market owner can influence the swap price to steal from liqudity providers	Low	Fixed
3	Incorrect value range of the `i32` type	Info	Acknowledged
4	Pause check in the `quote(...)` function doesn't prevent swaps in the core `SolverComponent`	Info	Fixed
5	Unhandled underflow during subtraction	Info	Fixed
6	Inconsistency between comment and function name	Best Practices	Fixed
7	Missing validations of base and quote tokens within the `create_market(...)` function	Best Practices	Acknowledged
8	Unnecessary storage read in `create_market(...)`	Best Practices	Fixed

4 System Overview

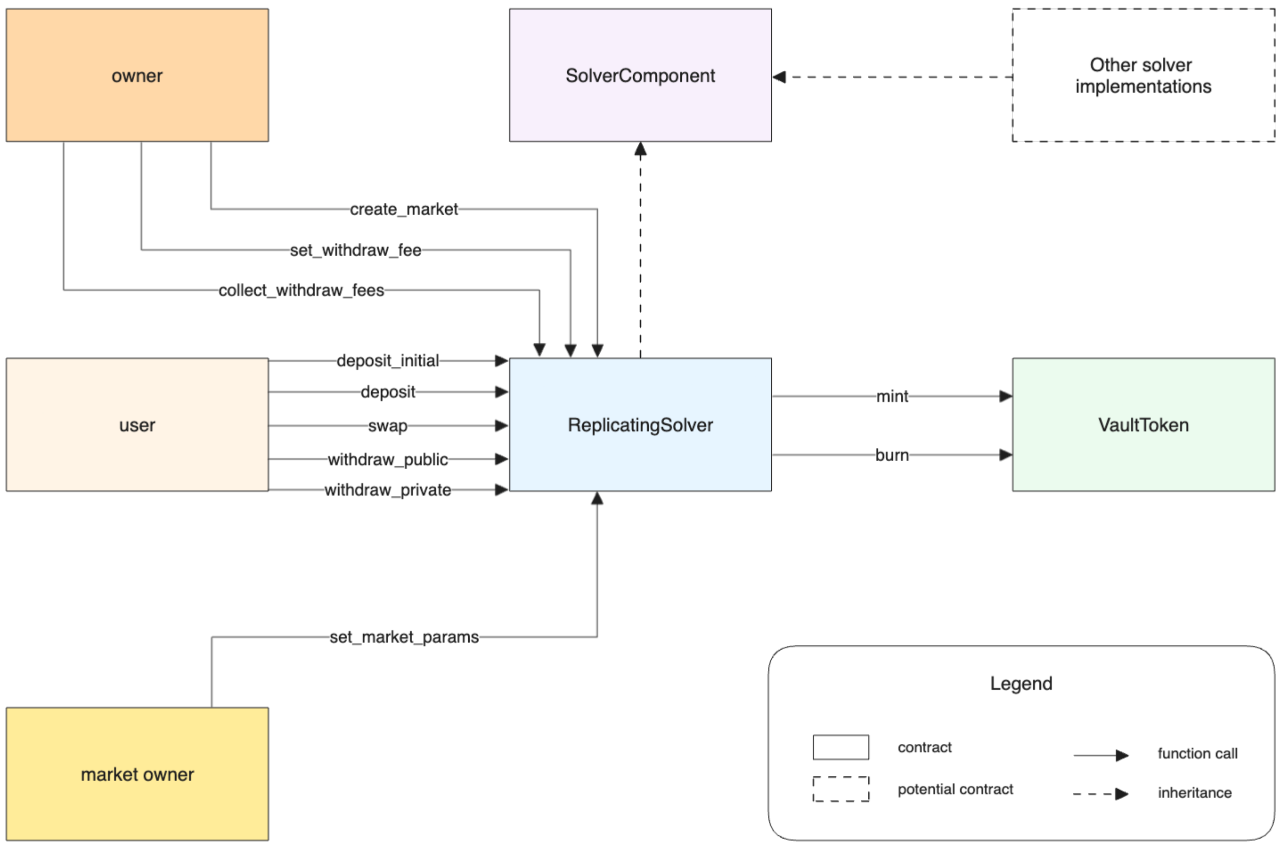


Fig. 2: Haiko solver overview, including most important interactions.

Haiko has introduced a new set of smart contracts to enhance liquidity management through the Haiko Solvers contracts. These independent contracts are designed to efficiently generate swap quotes and execute swaps with minimal gas usage.

The system comprises the core solver library (`SolverComponent`) and individual solver implementations, which include the `SolverHooks` interface. The Solvers smart contracts accept deposits from liquidity providers (LPs), generate swap quotes using a quoting strategy, and then execute swaps with traders. Unlike Haiko's Strategies, which interact with its AMM, Solvers do not deposit or interact with an underlying AMM protocol. Instead, Solvers directly calculate swap amounts and automatically execute trades against dynamically created positions.

4.1 Contracts

The contracts are part of a mono repository, which is divided into core solver libraries and contracts (in the `core` package) and solver implementations (in the `replicating` package).

4.1.1 Solver Core (`core` package)

The `SolverComponent` in the `core` package implements most of the core functionality of a `Solver` contract, including:

- Creating and managing new solver markets (which comprise a `base_token` and `quote_token` pair and an `owner` and it's visibility).
- Managing deposits and withdrawals from solver markets, which are tracked using ERC20 vault tokens representing shares of the market.
- Swapping assets through a solver market.
- Managing and collecting withdrawal fees (if enabled).
- Admin actions such as pausing and unpausing, upgrading, and transferring contract ownership.

Solvers currently support two market types:

- **Private Markets**: Managed only by the market owner who is able to deposit and withdraw liquidity and is responsible for keeping the correct exchange ratio.
- **Public Markets**: Open to any depositors who can manage their liquidity represented by ERC20 vault tokens, which work as market shares.

4.1.2 Solver Implementation (`replicating` package)

Every solver implementation must:

- Inherit the base functionality of `SolverComponent`.
- Implement `SolverHooks` interface, which contains methods for generating quotes and setting the market parameters that determine how swaps are executed and which strategies are used for quote calculation.

The `replicating` package contains the first developed solver called `Replicating Solver`. It creates a market for any token pair by providing bids and asks quotes based on an oracle price feed provided by Pragma oracle. It allows LPs to provide liquidity programmatically without actively managing their positions.

The solver executes swap orders based on the virtual bid and asks positions placed by each market. These positions are 'virtual' in that they are calculated in real-time for each swap. They are not stored in any contract states.

4.2 Creating and Configuring Solver Market

Each solver is a singleton contract that allows for the creation of multiple solver markets, all managed by the same contract. A solver market is created by calling `create_market(...)` through the `ISolver` interface.

```
fn create_market(
    ref self: TContractState, market_info: MarketInfo
) -> (felt252, Option<ContractAddress>);
```

This creates a market and assigns a `market_id`, which is the Poseidon chain hash of the members of the `MarketInfo` struct. Duplicate markets are disallowed.

Each market must define specific parameters (provided in the `MarketParams` struct) after market creation by calling the `set_market_params(...)` function. These parameters determine how the market will handle swaps from traders and the earnings for Liquidity Providers (LPs). The market owner can adjust these parameters later. To minimize the risk of malicious behavior by the market owner, any changes to market parameters are subject to a delay window. This means every parameter change requires first executing `queued_market_params(...)`, and only after a specified period can `set_market_params(...)` be executed.

```
fn queued_market_params(self: @TContractState, market_id: felt252) -> MarketParams;

fn set_market_params(ref self: TContractState, market_id: felt252);
```

Finally, each market's state is encapsulated within a `MarketState` struct. This struct includes the reserves for both tokens, the address of its corresponding vault token, and a boolean value indicating whether the market is paused. Markets are isolated from one another; they can only affect their own `MarketState` struct, and their reserves are independent of the balances of the singleton contract.

4.3 Depositing and Withdrawing

Deposits and withdrawals differ based on the type of solver market. Public markets allow third-party deposits and withdrawals. During a deposit, the liquidity provider deposits base and quote tokens according to the current reserve ratio and receives shares represented by ERC20 vault tokens. The number of shares is always calculated in favor of the token pair with higher reserves in the market. The equation for shares calculation is:

`shares = (total supply of vault tokens * pair token amount) / pair token reserves.`

During a withdrawal, the shares are burned, and the user receives a corresponding amount of base and quote tokens based on the current reserve ratio. Both deposits and withdrawals for public vaults must always maintain the current market reserve ratio.

Private markets only allow the market owner to deposit and withdraw assets. Since no other depositors are involved, the owner can deposit and withdraw assets at any ratio.

Liquidity providers can deposit to a solver market by calling `deposit(...)` or `deposit_initial(...)` if no deposits have been made yet. The initial deposit creates the first reserve ratio between the base and quote token.

```
fn deposit_initial(
    ref self: TContractState, market_id: felt252, base_amount: u256, quote_amount: u256
) -> (u256, u256, u256);

fn deposit(
    ref self: TContractState, market_id: felt252, base_amount: u256, quote_amount: u256
) -> (u256, u256, u256);
```

Withdrawals can be made by calling `withdraw_public(...)` for public vaults or `withdraw_private(...)` for private vaults.

```
fn withdraw_public(
    ref self: TContractState, market_id: felt252, shares: u256
) -> (u256, u256);

fn withdraw_private(
    ref self: TContractState, market_id: felt252, base_amount: u256, quote_amount: u256
) -> (u256, u256);
```

Note: During the review of this protocol, the functions were renamed. The function `withdraw_at_ratio(...)` *was used for public vaults, while* `withdraw(...)` *was designated for private vaults.*

4.4 Swapping and Quoting

A swap is executed when a swapper calls the `swap(...)` function in the `Solver` contract. Under the hood, this calls `quote(...)` to obtain a quote for the swap and then executes the swap.

The `quote(...)` function is part of the `SolverHooks` interface and should be implemented by each `Solver` contract based on its desired quoting logic and strategy.

```
fn quote(self: @TContractState, market_id: felt252, swap_params: SwapParams) -> (u256, u256);

fn swap(ref self: TContractState, market_id: felt252, swap_params: SwapParams) -> (u256, u256);
```

The swap is performed using the Uniswap's V3 position concept. However, the position creation is dynamic and depends on a provided current price, as well as particular quoting implementation and the initially set parameters: **min_spread**, **range**, **max_delta**, **max_skew**.

In the `ReplicatingSolver`, the price comes from the Pragma oracle. This provided price, called `limit`, is used as a beginning or the border of the position range. Then, the `limit` is adjusted by given parameters. First, the `skew` is computed by calculating the difference between the provided oracle price and the price given the current reserves in the market. The `skew` is a percentage value and represents how much of `max_delta` parameter should be applied to `limit`. Then, the `limit` is moved in the direction of a swap by the `min_spread`. Next, the `delta` is applied to the `limit`. From the modified `limit`, the position is created, and its length is defined by the `range` parameter. The swap is performed within this position. After the swap, the new skew is calculated and compared with the `max_skew`(When the skew is more significant than `max_skew`, the function reverts.). The moving of the `limit` by the spread and delta affects the swap either in favor of the swappers or the liquidity providers. If the `limit` is moved in the direction of a swap, then the LPs' shares accrue value (fees). If the `limit` moves in the opposite direction, the price is more attractive to the swapper even to the point where the swap price is lower than the provided oracle price, which results in a discount for the swapper but loss of fees for the LPs.

4.5 Setting and Collecting Withdrawal Fees

Solvers are deployed with a contract `owner` with permission to set and collect withdrawal fees for each solver market. Fees are set as a percentage of the withdrawn amount and can be set and collected by calling `set_withdraw_fee(...)` and `collect_withdraw_fees(...)` respectively.

```
fn collect_withdraw_fees(
    ref self: TContractState, receiver: ContractAddress, token: ContractAddress
) -> u256;

fn set_withdraw_fee(ref self: TContractState, market_id: felt252, fee_rate: u16);
```

4.6 Pausing and Unpausing Markets

Market owners can pause and unpause their markets. This functionality is implemented through the `pause(...)` and `unpause(...)` functions, which can only be called by the market owners. Pausing is used in emergency situations to restrict new deposits and swaps; however, withdrawals remain active to allow liquidity providers to recover their tokens.

```
fn pause(ref self: TContractState, market_id: felt252);

fn unpause(ref self: TContractState, market_id: felt252);
```

5 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Critical] Anyone can withdraw from private vaults

File(s): `core/src/contracts/solver.cairo`

Description: Markets are divided into public and private ones. In a private market, only the market owner controls its liquidity and is authorized to deposit and withdraw from it.

The `withdraw(...)` function allows a market owner to withdraw liquidity from a private market. This function includes checks to ensure that the market exists and is private:

```
fn withdraw(...) -> (...) {
  // ...
  assert(market_info.base_token != contract_address_const::<0x0>(), 'MarketNull');
  assert(!market_info.is_public, 'UseWithdrawAtRatio');
  // ...
}
```

Note: The name of the `withdraw(...)` *function and error message of the second assert statement were changed to* `withdraw_private(...)` *and* `'UseWithdrawPublic'` *in the final commit.*

However, the function lacks an essential check to verify that the caller is the market owner, allowing anyone to call the function and withdraw all liquidity from the private market, stealing all its funds.

Recommendation(s): Add an access control check to prevent any unauthorized withdrawal and ensure that only the market owner can withdraw liquidity from a private market.

Status: Fixed.

Update from the client: Fixed by commit 44c68c2. We implement the recommended fix by adding a `assert_market_owner()` check to prevent non-owner withdrawals. We also add a new test case to prevent future regression of this access control check.

6.2 [Low] Market owner can influence the swap price to steal from liquidity providers

File(s): `replicating/src/contracts/replicating_solver.cairo`

Description: The market owner can set the market parameter with the function `set_market_params(...)`. One of the parameters is `max_delta`, which defines how many ticks the start price can be moved when accounting for a skew. The skew represents the difference between the value of base assets in the market and the oracle price, expressed as a percentage.

The `max_delta` parameter allows the market owner to influence the price direction depending on the skew:

- Positive Direction: Increases the swap price, raising fees for liquidity providers (LPs).
- Negative Direction: Decreases the swap price, potentially creating a discount for swappers but reducing fees for LPs.

A malicious market owner could exploit this mechanism by setting a high `max_delta` after detecting a negative skew and then performing a swap. This manipulation would move the price significantly below the oracle price, creating a large discount for swappers and drastically reducing the value of LP shares. This could lead to substantial losses for LPs, including their initially deposited capital.

Currently, only the protocol owner can create markets, limiting this exploit. However, if this functionality is extended to a broader group in the future, the risk of exploitation would increase significantly.

Recommendation(s): Consider checking if the `max_delta` is within a reasonable range when setting the market parameters via the `set_market_params(...)` function. Additionally, consider implementing a two-step process for changing market parameters. This could involve requiring an initial parameters proposal that would only be enforced after a certain delay, allowing users to review the changes and decide whether to withdraw their funds.

Status: Fixed.

Update from the client:

- Fixed by commit f35bbdc. We add a new timelock mechanism to address the risk of swap price manipulation. In particular, the contract admin will now be able to set a delay (in seconds), which must pass before queued market parameter updates are set. We expect this feature will not be used until we introduce permissionless market listing.
- Further updated by commit 668ee67. We skip the delay if the market parameters are being initialized for the first time.

6.3 [Info] Incorrect value range of the `i32` type

File(s): `replicating/src/libraries/spread_math.cairo`

Description: The type `i32` imported from the `haiko_lib` is used in the `spread_math` file. However, its value range differs from Solidity's `int32` type:

- Range of `i32` in `haiko_lib`: `[-4294967295, 4294967295]`;
- Range of `int32` in Solidity: `[-2147483648, 2147483647]`;

This discrepancy arises due to the lack of boundary checks during the creation of the `i32` type:

```
fn new(val: u32, sign: bool) -> i32 {
    // @audit: val is u32
    i32 { val, sign: if val == 0 {
        false
    } else {
        sign
    } }
}
```

Recommendation(s): Consider checking the correct boundaries of `val` while creating the `i32` type number to maintain the intended range.

Status: Acknowledged.

Update from the client: Fixed by commit 84cf7ce. We don't update the `i32` implementation, but instead add inline documentation to explain that it has a non-standard range.

6.4 [Info] Pause check in the `quote(...)` function doesn't prevent swaps in the core `SolverComponent`

File(s): `core/src/contracts/solver.cairo`

Description: The documentation specifies that a market owner can pause a market to prevent deposits and swaps. While the `deposit` and `deposit_initial` functions correctly implement these pause checks, the `swap` function relies on the solver implementation (in this case, the `ReplicatingSolver` inside `quote(...)` function) to perform the pause check. This could lead to inconsistencies where a solver implementation might allow swaps even when the market is paused, contrary to the documentation.

Recommendation(s): Consider adding a pause check in the `swap` function within the core `SolverComponent` for consistency. This ensures that all solver implementations will prevent swaps while their market is paused, regardless of their quote implementation, and align with the documentation.

Status: Fixed.

Update from the client: Fixed by commit 8da868f. To ensure consistency, we migrate all validity checks from the `quote()` function back to `swap()`. This includes both the pause check mentioned above as well as a second check for uninitialised markets.

6.5 [Info] Unhandled underflow during subtraction

File(s): `replicating/src/libraries/spread_math.cairo`

Description: The function `get_virtual_position_range(...)` in the `spread_math` library does not validate the values before performing subtraction operations.

```
if is_bid {
    limit -= min_spread;
} else {
    limit += min_spread;
}

if delta.sign {
    limit -= delta.val;
} else {
    limit += delta.val;
}
```

In both scenarios, the `limit` could be less than `min_spread` or `delta.val`, leading to an underflow and resulting in an unclear error message.

Recommendation(s): Consider checking the values before the subtractions to ensure the correct error message when the subtraction will result in underflow.

Status: Fixed.

Update from the client: Fixed by commit 50613ec. We implement the recommended fix by adding a range check and emitting an error message in the event of underflow.

6.6 [Best Practices] Inconsistency between comment and function name

File(s): `core/src/contracts/solver.cairo`

Description: The function `withdraw_at_ratio(...)` has a comment indicating that the function `withdraw_amount` should be used for private vaults. However, the actual function name used for private withdrawals is `withdraw`, leading to an inconsistency between the comment and the function name.

Here is the relevant section of the code:

```
// Burn pool shares and withdraw funds from market.
// Called for public vaults. For private vaults, use `withdraw_amount`.
// ...
fn withdraw_at_ratio(
    ref self: ComponentState<TContractState>, market_id: felt252, shares: u256
) -> (u256, u256) {...}
```

Recommendation(s): To maintain clarity and consistency, consider adjusting either the comment or the function name.

Status: Fixed.

Update from the client:

- Superseded by commit f676015. For clarity, we rename `withdraw()` -> `withdraw_private()` and `withdraw_at_ratio()` -> `withdraw_-public()`. We update all comments to reflect this change.
- Further updated in commit 39f0ae7. We update the error messages for consistency with the new function names, plus add a missing test cases for withdrawing from private vaults.

6.7 [Best Practices] Missing validations of base and quote tokens within the `create_market(...)` function

File(s): `core/src/contracts/solver.cairo`

Description: The `create_market(...)` function allows the owner of the solver contract to create markets. However, this function lacks input validations that could preserve specific invariants and ensure the integrity of the protocol:

1. Same Token for Base and Quote: The function does not prevent the creation of a market where the base and quote tokens are the same, which would result in an impractical market;
2. Duplicate Markets with Swapped Tokens: As per the documentation, a market owner is expected to maintain only one market for the same token pair. However, the current implementation allows the market owner to create two markets for the same pair by swapping the base and quote tokens, leading to different hashes in the Poseidon hashing mechanism.

Recommendation(s): Consider adding validation rules for these cases to the `create_market(...)` function to ensure robustness and maintain the protocol's integrity.

Status: Acknowledged.

Update from the client: Partially fixed by 2d9985b. We address issue (1) above by adding a new check and test case for non-equality between the base and quote tokens. We do not apply any fix for the issue (2) because this is best addressed through off-chain validation (at the level of the indexer and frontend interface).

6.8 [Best Practices] Unnecessary storage read in `create_market(...)`

File(s): `core/src/contracts/solver.cairo`

Description: When setting up a public market, a vault token is deployed, and its address is set in the market state struct for the newly created `market_id`. This process includes retrieving an empty `MarketState` from storage, modifying it, and writing it back to storage as shown below:

```
fn create_market(...) -> (...) {
    // ...
    let mut vault_token: Option<ContractAddress> = Option::None(());
    if market_info.is_public {
        let vault_token_addr = self._deploy_vault_token(market_info);
        vault_token = Option::Some(vault_token_addr);
        // @audit This storage read will always be empty
        // Can initialize empty `MarketState` struct instead
        let mut state: MarketState = self.market_state.read(market_id);
        state.vault_token = vault_token_addr;
        self.market_state.write(market_id, state);
    }
    // ...
}
```

When calling `create_market(...)` for a new and unique `market_id`, the storage read will always return an empty `MarketState` struct. Interacting with storage is more costly in terms of execution than using memory. Because the `MarketState` will always be empty initially, it can be efficiently initialized directly in memory.

Recommendation(s): Consider initializing an empty `MarketState` in memory instead of loading it from storage when writing the vault token data for public markets to reduce execution costs.

Status: Fixed.

Update from the client: Fixed by commit 5a7e900. We implement the recommended fix by initialising an empty struct rather than reading from state.

7 Documentation Evaluation

Software documentation refers to the written or visual information describing software's functionality, architecture, design, and implementation. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about Haiko documentation

The **Haiko** team has provided documentation about their protocol through detailed in-line comments within the code, and a README file. Additionally, the Haiko team was available to address any questions or concerns from the Nethermind Security team.

8 Test Suite Evaluation

8.1 Compilation Output

```
> scarb build
   Compiling lib(haiko_solver_core) haiko_solver_core v1.0.0 (/home/audits/nethermind/NM-0266/packages/core/Scarb.toml)
   Compiling starknet-contract(haiko_solver_core) haiko_solver_core v1.0.0
    ↪  (/home/audits/nethermind/NM-0266/packages/core/Scarb.toml)
   Compiling haiko_solver_replicating v1.0.0 (/home/audits/nethermind/NM-0266/packages/replicating/Scarb.toml)

Finished release target(s) in 43 seconds
```

8.2 Tests Output

```
> snforge test

Collected 113 test(s) from haiko_solver_core package
Running 113 test(s) from src/
[PASS] haiko_solver_core::tests::libraries::test_id::test_market_id (gas: ~3)
[PASS] haiko_solver_core::tests::solver::test_create_market::test_create_market_emits_events (gas: ~4646)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_both_amounts_zero_fails (gas: ~3826)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_uninitialised_market_fails (gas:
 ↪  ~3822)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_private_quote_token_only (gas:
 ↪  ~3374)
[PASS] haiko_solver_core::tests::solver::test_create_market::test_create_market_with_null_base_token_fails (gas: ~3819)
[PASS] haiko_solver_core::tests::solver::test_create_market::test_create_market_with_null_owner_fails (gas: ~3819)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_with_referrer_emits_event (gas:
 ↪  ~4268)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_emits_event (gas: ~4265)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_public_base_token_only (gas: ~4150)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_public_fail_for_private_vault (gas: ~3497)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_private_both_tokens (gas: ~3517)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_emits_event (gas: ~4308)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_market_uninitialised (gas: ~4270)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_with_referrer_emits_event (gas: ~4313)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_allowed_even_if_paused (gas: ~4414)
[PASS] haiko_solver_core::tests::libraries::test_store_packing::test_store_packing_market_state (gas: ~299)
[PASS] haiko_solver_core::tests::solver::test_withdraw_fees::test_set_withdraw_fees_emits_event (gas: ~3885)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_remaining_shares_from_public_vault (gas: ~4038)
[PASS] haiko_solver_core::tests::solver::test_deploy::test_deploy_solver_and_vault_token_initialises_immutables (gas:
 ↪  ~3827)
[PASS] haiko_solver_core::tests::solver::test_withdraw_fees::test_withdraw_fee_overflow (gas: ~3818)
[PASS] haiko_solver_core::tests::solver::test_deploy::test_deploy_vault_token_initialises_immutables (gas: ~3825)
[PASS] haiko_solver_core::tests::solver::test_create_market::test_create_duplicate_market_fails (gas: ~3819)
[PASS] haiko_solver_core::tests::solver::test_vault_token::test_burn_vault_token_fails_for_non_owner (gas: ~3817)
[PASS] haiko_solver_core::tests::solver::test_pause::test_pause_prevents_owner_deposits (gas: ~3894)
[PASS] haiko_solver_core::tests::solver::test_create_market::test_create_market_with_null_quote_token_fails (gas:
 ↪  ~3819)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_private_base_only (gas: ~3408)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_not_approved (gas: ~2810)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_public_quote_token_only (gas:
 ↪  ~4150)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_private_market_for_non_owner_caller (gas: ~3497)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_private_quote_only (gas: ~3408)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_private_partial_amounts (gas: ~3543)
[PASS] haiko_solver_core::tests::solver::test_withdraw_fees::test_set_withdraw_fees (gas: ~3885)
[PASS] haiko_solver_core::tests::solver::test_withdraw_fees::test_collect_withdraw_fees_emits_event (gas: ~3372)
[PASS] haiko_solver_core::tests::solver::test_vault_token::test_change_vault_token_class_fails_if_not_owner (gas:
 ↪  ~3817)
[PASS] haiko_solver_core::tests::solver::test_vault_token::test_change_vault_token_class_fails_if_unchanged (gas:
 ↪  ~3817)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_public_both_tokens (gas: ~4302)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_private_vault_both_tokens_at_arbitrary_ratio (gas:
 ↪  ~3552)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_private_vault_base_token_only (gas: ~3543)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_private_base_token_only (gas:
 ↪  ~3373)
```

```
[PASS] haiko_solver_core::tests::solver::test_vault_token::test_change_vault_token_class_fails_if_zero_address (gas:
↪ ~3817)
[PASS] haiko_solver_core::tests::solver::test_upgrade::test_upgrade_solver (gas: ~3822)
[PASS] haiko_solver_core::tests::solver::test_withdraw_fees::test_collect_withdraw_fees (gas: ~3352)
[PASS] haiko_solver_core::tests::solver::test_withdraw_fees::test_withdraw_fee_unchanged (gas: ~3820)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_no_existing_deposits (gas: ~3824)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_on_market_with_existing_deposits
↪ (gas: ~4268)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_partial_shares_from_public_vault (gas: ~4449)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_market_uninitialised (gas: ~3192)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_public_vault_both_tokens_above_base_ratio (gas:
↪ ~4369)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_public_vault_both_tokens_above_quote_ratio (gas:
↪ ~4369)
[PASS] haiko_solver_core::tests::solver::test_pause::test_unpause_after_pause_reenables_deposits (gas: ~4275)
[PASS] haiko_solver_core::tests::solver::test_withdraw_fees::test_withdraw_fee_not_owner (gas: ~3817)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_public_vault_quote_token_only (gas: ~4249)
[PASS] haiko_solver_core::tests::solver::test_ownership::test_transfer_ownership_fails_if_unchanged (gas: ~3820)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_public_vault_both_tokens_above_available (gas:
↪ ~4068)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_both_amounts_zero_fails (gas:
↪ ~3826)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_public_vault_both_tokens_at_ratio (gas: ~4409)
[PASS]
↪ haiko_solver_core::tests::solver::test_ownership::test_transfer_ownership_fails_if_accepting_from_non_owner_address
↪ (gas: ~3884)
[PASS] haiko_solver_core::tests::solver::test_get_balances::test_get_user_balances_array_length_mismatch (gas: ~3817)
[PASS] haiko_solver_core::tests::solver::test_ownership::test_transfer_ownership_fails_not_owner (gas: ~3817)
[PASS] haiko_solver_core::tests::solver::test_ownership::test_transfer_ownership_works (gas: ~3824)
[PASS] haiko_solver_core::tests::solver::test_create_market::test_create_market_with_same_token_fails (gas: ~3819)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_public_vault_base_token_only (gas: ~4249)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_private_vault_quote_token_only (gas: ~3544)
[PASS] haiko_solver_core::tests::solver::test_get_balances::test_get_user_balances_array (gas: ~4394)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_public_vault_multiple_lps_capped_at_available (gas:
↪ ~4390)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_swap_buy_below_threshold_amount (gas: ~3508)
[PASS] haiko_solver_core::tests::solver::test_pause::test_pause_fails_if_already_paused (gas: ~3890)
[PASS] haiko_solver_core::tests::solver::test_vault_token::test_mint_vault_token_fails_for_non_owner (gas: ~3817)
[PASS] haiko_solver_core::tests::solver::test_pause::test_unpause_emits_event (gas: ~3830)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_private_uninitialised_market (gas: ~3195)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_private_not_owner (gas: ~3196)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_more_than_available_correctly... (gas: ~3257)
[PASS] haiko_solver_core::tests::solver::test_swap::test_after_swap_fails_for_non_solver_caller (gas: ~3189)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_private_zero_amounts (gas: ~3498)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_private_emits_event (gas: ~3259)
[PASS] haiko_solver_core::tests::solver::test_ownership::test_transfer_then_update_owner_before_accepting_works (gas:
↪ ~3827)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_custom_amounts_fail_for_public_vault (gas: ~4270)
[PASS] haiko_solver_core::tests::libraries::test_erc20_versioned_call::test_erc20_felt252 (gas: ~257)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_all_remaining_balances_from_private_vault (gas:
↪ ~3287)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_public_emits_event (gas: ~3918)
[PASS] haiko_solver_core::tests::solver::test_ownership::test_transfer_ownership_emits_event (gas: ~3824)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_public_zero_shares (gas: ~4271)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_swap_buy_with_zero_liquidity (gas: ~3365)
[PASS] haiko_solver_core::tests::libraries::test_erc20_versioned_call::test_erc20_bytearray (gas: ~362)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_swap_sell_below_threshold_amount (gas: ~3508)
[PASS]
↪ haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_on_private_market_for_non_owner_caller
↪ (gas: ~3194)
[PASS] haiko_solver_core::tests::solver::test_create_market::test_create_public_market_initialises... (gas: ~4660)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_sell_exact_out (gas: ~3540)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_not_approved (gas: ~2673)
[PASS] haiko_solver_core::tests::solver::test_pause::test_pause_prevents_non_owner_deposits (gas: ~4341)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_should_emit_event (gas: ~3540)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_not_approved (gas: ~2810)
```

```
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_market_paused (gas: ~3564)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_public_more_shares_than_available (gas: ~4274)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_buy_exact_in (gas: ~3540)
[PASS] haiko_solver_core::tests::solver::test_vault_token::test_change_vault_token_class_emits_event (gas: ~3820)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_amount_zero (gas: ~3507)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_sell_exact_in (gas: ~3540)
[PASS] haiko_solver_core::tests::solver::test_upgrade::test_upgrade_solver_not_owner (gas: ~3817)
[PASS] haiko_solver_core::tests::solver::test_create_market::test_create_private_market_initialises_immutables (gas:
 ↪ ~4021)
[PASS] haiko_solver_core::tests::solver::test_deposit_initial::test_deposit_initial_on_paused_market (gas: ~3892)
[PASS] haiko_solver_core::tests::solver::test_get_balances::test_get_balances_array (gas: ~4266)
[PASS] haiko_solver_core::tests::solver::test_pause::test_pause_fails_if_already_unpaused (gas: ~3819)
[PASS] haiko_solver_core::tests::solver::test_vault_token::test_change_vault_token_class_works (gas: ~3818)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_swap_sell_with_zero_liquidity (gas: ~3365)
[PASS] haiko_solver_core::tests::solver::test_get_balances::test_get_balances (gas: ~4265)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_fails_if_min_amount_out_zero (gas: ~3508)
[PASS] haiko_solver_core::tests::solver::test_deposit::test_deposit_paused (gas: ~4341)
[PASS] haiko_solver_core::tests::solver::test_pause::test_pause_allows_withdraws (gas: ~4415)
[PASS] haiko_solver_core::tests::solver::test_withdraw::test_withdraw_public_uninitialised_market (gas: ~3824)
[PASS] haiko_solver_core::tests::solver::test_pause::test_pause_emits_event (gas: ~3887)
[PASS] haiko_solver_core::tests::solver::test_swap::test_swap_buy_exact_out (gas: ~3540)
Tests: 113 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out

Collected 104 test(s) from haiko_solver_replicating package
Running 104 test(s) from src/
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_buy_exact_output... (gas:
 ↪ ~13)
[PASS] haiko_solver_replicating::tests::libraries::test_spread_math::test_get_virtual_position_range_oracle_price_zero
 ↪ (gas: ~1)
[PASS] haiko_solver_replicating::tests::libraries::test_store_packing::test_store_packing_market_params (gas: ~303)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_output_eq_quote_reserves
 ↪ (gas: ~5)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_output_eq_base_reserves
 ↪ (gas: ~5)
[PASS] haiko_solver_replicating::tests::libraries::test_spread_math::test_get_delta (gas: ~136)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_sell_price_overflow (gas:
 ↪ ~5)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_buy_exact_input_filled_max
 ↪ (gas: ~78)
[PASS]
 ↪ haiko_solver_replicating::tests::libraries::test_spread_math::test_get_virtual_position_range_oracle_price_overflow
 ↪ (gas: ~417)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_cases (gas: ~36)
[PASS]
 ↪ haiko_solver_replicating::tests::libraries::test_spread_math::test_get_virtual_position_range_bid_limit_underflow
 ↪ (gas: ~414)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_in_price_overflow (gas: ~5)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_in_cases (gas: ~45)
Test Case 1) Full range liq, price 1, no spread
Swap Case 1) is_buy: true, exact_input: true
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_sell_exact_output... (gas:
 ↪ ~14)
Test Case 11) Swap with low oracle price
Swap Case 1) is_buy: true, exact_input: true
Test Case 6) Concentrated liq, price 1, 50000 spread
Swap Case 1) is_buy: true, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_oracle::test_change_oracle_unchanged (gas: ~4396)
base_deposit_init: 100000000000000000000, quote_deposit_init: 1000000000000000000000, shares_init:
 ↪ 126840253109708511593711
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_delay (gas: ~4460)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_if_max_age_zero (gas:
 ↪ ~4675)
base_deposit: 50000000000000000000, quote_deposit: 500000000000000000000, shares: 63420126554854255796855
[PASS]
 ↪ haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_if_base_currency_id_zero
 ↪ (gas: ~4611)
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_change_in_oracle_price_above_max_skew_prevents_swap
 ↪ (gas: ~5233)
```

```
[PASS]
↪ haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_none_queued_null_params
↪ (gas: ~4557)
amount in: 100000000000000000000, amount out: 90909090909090909146
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_before_delay_complete
↪ (gas: ~4738)
[PASS]
↪ haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_if_quote_currency_id_zero
↪ (gas: ~4611)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_price_0 (gas: ~2)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_if_not_owner (gas:
↪ ~4675)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_buy_price_overflow (gas: ~6)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_liq_0 (gas: ~2)
Test Case 1) Full range liq, price 1, no spread
Swap Case 2) is_buy: false, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_if_unchanged (gas:
↪ ~4471)
[PASS] haiko_solver_replicating::tests::libraries::test_spread_math::test_get_virtual_position_range_cases (gas: ~1863)
[PASS] haiko_solver_replicating::tests::solver::test_deposit_initial::test_deposit_initial_public_mismatched... (gas:
↪ ~7056)
amount_in: 10000000000000000000, amount_out: 999463519687203401
amount in: 10263437372398, amount out: 9999999999999992346
base_withdraw: 494185112659704108 71, quote_withdraw: 500816666666666666666
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_that_improves_skew_is_allowed (gas: ~5237)
amount in: 100000000000000000000, amount out: 6064393018672684143
base_fees: 248334227467187995, quote_fees: 251666666666666667
base_reserves: 993336909868751977 33, quote_reserves: 1006666666666666666667
base_reserves_exp: 99333690986875197733, quote_reserves_exp: 1006666666666666666667
Test Case 11) Swap with low oracle price
Swap Case 2) is_buy: false, exact_input: true
amount in: 100000000000000000000, amount out: 90909090909090909146
[PASS] haiko_solver_replicating::tests::solver::test_e2e::test_solver_e2e_public_market (gas: ~7496)
Test Case 6) Concentrated liq, price 1, 50000 spread
Swap Case 2) is_buy: false, exact_input: true
Test Case 1) Full range liq, price 1, no spread
Swap Case 3) is_buy: true, exact_input: false
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_public_vault_both_tokens_above_quote_ratio
↪ (gas: ~5958)
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_public_vault_both_tokens_above_base_ratio
↪ (gas: ~5958)
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_public_vault_both_tokens_above_available
↪ (gas: ~5692)
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_private_vault_quote_token_only (gas: ~6251)
amount in: 111111111111111111028, amount out: 100000000000000000000
amount in: 100000000000000000000, amount out: 597579323442496790518
amount in: 100000000000000000000, amount out: 998993359216
Test Case 1) Full range liq, price 1, no spread
Swap Case 4) is_buy: false, exact_input: false
[PASS]
↪ haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_sell_exact_output_reached...
↪ (gas: ~18)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_buy_exact_output_filled_max
↪ (gas: ~80)
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_fails_if_limit_overflows (gas: ~4916)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_sell_exact_input_reached...
↪ (gas: ~16)
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_fails_if_swap_buy_below_threshold_amount (gas:
↪ ~4951)
Test Case 6) Concentrated liq, price 1, 50000 spread
Swap Case 3) is_buy: true, exact_input: false
Test Case 11) Swap with low oracle price
Swap Case 3) is_buy: true, exact_input: false
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_public_vault_base_token_only (gas: ~7696)
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_private_vault_both_tokens_at_arbitrary_ratio
↪ (gas: ~6249)
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_public_vault_both_tokens_at_ratio (gas:
↪ ~8160)
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_private_vault_base_token_only (gas: ~6238)
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_buy_above_max_skew_is_disallowed (gas: ~4357)
```

```
[PASS] haiko_solver_replicating::tests::solver::test_deposit::test_deposit_public_vault_quote_token_only (gas: ~7659)
amount in: 111111111111111111028, amount out: 100000000000000000000
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_fails_if_swap_sell_below_threshold_amount (gas:
 ↪  ~5202)
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_fails_if_limit_underflows (gas: ~4525)
[PASS] haiko_solver_replicating::tests::solver::test_oracle::test_change_oracle_emits_event (gas: ~4396)
Test Case 2) Full range liq, price 0.1, no spread
Swap Case 1) is_buy: true, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_oracle::test_change_oracle (gas: ~4396)
amount in: 1652803511080475795529, amount out: 100000000000000000000
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_sell_above_max_skew_is_disallowed (gas: ~4355)
Test Case 6) Concentrated liq, price 1, 50000 spread
Swap Case 4) is_buy: false, exact_input: false
amount in: 1003480936229, amount out: 100000000000000000000
Test Case 11) Swap with low oracle price
Swap Case 4) is_buy: false, exact_input: false
amount in: 10000000000000000000, amount out: 49999834853317669644
Test Case 2) Full range liq, price 0.1, no spread
Swap Case 2) is_buy: false, exact_input: true
amount in: 16528088195378414841, amount out: 100000000000000000000
Test Case 7) Concentrated liq, price 1, 100 spread, 500 max delta
Swap Case 1) is_buy: true, exact_input: true
amount in: 10034852567983843807646684605, amount out: 100000000000000000000
amount in: 10000000000000000000, amount out: 998997611702025556
Test Case 12) Swap buy capped at threshold price
Swap Case 1) is_buy: true, exact_input: true
amount in: 100000000000000000000, amount out: 99245582637747914747
Test Case 2) Full range liq, price 0.1, no spread
Swap Case 3) is_buy: true, exact_input: false
Test Case 7) Concentrated liq, price 1, 100 spread, 500 max delta
Swap Case 2) is_buy: false, exact_input: true
amount in: 21850483612303829530, amount out: 20823204527740984512
Test Case 12) Swap buy capped at threshold price
Swap Case 2) is_buy: true, exact_input: false
amount in: 100000000000000000000, amount out: 99819404970139967371
amount in: 1111118450987902275, amount out: 10000000000000000000
Test Case 7) Concentrated liq, price 1, 100 spread, 500 max delta
Swap Case 3) is_buy: true, exact_input: false
Test Case 2) Full range liq, price 0.1, no spread
Swap Case 4) is_buy: false, exact_input: false
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_get_swap_amounts_bid_threshold_sqrt_price (gas:
 ↪  ~81)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_get_swap_amounts_over_zero_liquidity (gas: ~51)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_buy_exact_input_reaches...
 ↪  (gas: ~65)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_get_swap_amounts_ask_threshold_sqrt_price (gas:
 ↪  ~83)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_buy_exact_output_reaches...
 ↪  (gas: ~71)
amount in: 21850483612303829530, amount out: 20823204527740984512
amount in: 100763924334713808580, amount out: 100000000000000000000
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_queue_and_set_market_params_with_delay...
 ↪  (gas: ~4467)
Test Case 13) Swap sell capped at threshold price
Swap Case 1) is_buy: false, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_after_swap_fails_for_non_solver_caller (gas: ~3765)
Test Case 7) Concentrated liq, price 1, 100 spread, 500 max delta
Swap Case 4) is_buy: false, exact_input: false
amount in: 101010443847319896837, amount out: 10000000000000000000
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_emits_event (gas: ~4425)
Test Case 3) Full range liq, price 10, no spread
Swap Case 1) is_buy: true, exact_input: true
amount in: 24240352373112801070, amount out: 22984922158048704819
amount in: 100181369566420499920, amount out: 100000000000000000000
[PASS] haiko_solver_replicating::tests::solver::test_withdraw::test_withdraw_private_base_only (gas: ~6006)
[PASS] haiko_solver_replicating::tests::solver::test_withdraw::test_withdraw_private_quote_only (gas: ~6018)
Test Case 13) Swap sell capped at threshold price
Swap Case 2) is_buy: false, exact_input: false
[PASS] haiko_solver_replicating::tests::solver::test_withdraw::test_withdraw_allowed_even_if_paused (gas: ~5986)
```

```
[PASS] haiko_solver_replicating::tests::solver::test_deposit_initial::test_deposit_initial_public_quote_token_only
↪ (gas: ~7453)
[PASS] haiko_solver_replicating::tests::solver::test_deposit_initial::test_deposit_initial_public_base_token_only (gas:
↪ ~7478)
[PASS] haiko_solver_replicating::tests::solver::test_deposit_initial::test_deposit_initial_public_both_tokens (gas:
↪ ~7813)
Test Case 8) Concentrated liq, price 0.1, 100 spread, 20000 max delta
Swap Case 1) is_buy: true, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_deposit_initial::test_deposit_initial_private_base_token_only
↪ (gas: ~5753)
[PASS] haiko_solver_replicating::tests::solver::test_deposit_initial::test_deposit_initial_private_both_tokens (gas:
↪ ~5969)
[PASS] haiko_solver_replicating::tests::solver::test_withdraw::test_withdraw_partial_shares_from_public_vault (gas:
↪ ~8192)
[PASS] haiko_solver_replicating::tests::solver::test_withdraw::test_withdraw_private_partial_amounts (gas: ~6233)
[PASS]
↪ haiko_solver_replicating::tests::solver::test_withdraw::test_withdraw_more_than_available_correctly_caps_amount...
↪ (gas: ~3834)
[PASS] haiko_solver_replicating::tests::solver::test_withdraw::test_withdraw_all_remaining_balances_from_private_vault
↪ (gas: ~5726)
amount in: 100000000000000000000, amount out: 9900956827006555844
amount in: 24240352373112801070, amount out: 22984922158048704819
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_sell_exact_input_filled_max
↪ (gas: ~72)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_queue_and_set_market_params_no_delay (gas:
↪ ~4423)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_update_queued_market_params (gas: ~4673)
[PASS] haiko_solver_replicating::tests::solver::test_withdraw::test_withdraw_remaining_shares_from_public_vault (gas:
↪ ~7515)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_in_price_0 (gas: ~2)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_cancel_queued_market_params (gas: ~4481)
[PASS]
↪ haiko_solver_replicating::tests::libraries::test_swap_lib::test_compute_swap_amounts_sell_exact_output_filled_max
↪ (gas: ~30)
Test Case 3) Full range liq, price 10, no spread
Swap Case 2) is_buy: false, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_queue_market_params_emits_event (gas: ~4662)
[PASS] haiko_solver_replicating::tests::solver::test_oracle::test_change_oracle_not_owner (gas: ~4393)
Test Case 14) Swap capped at threshold amount, exact input
Swap Case 1) is_buy: true, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_delay_emits_event (gas: ~4460)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_queue_and_set_market_params_with_delay (gas:
↪ ~4495)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_output_gt_base_reserves
↪ (gas: ~5)
amount in: 61495781354774112620, amount out: 4999999999999999997
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_in_liq_0 (gas: ~2)
Test Case 8) Concentrated liq, price 0.1, 100 spread, 20000 max delta
Swap Case 2) is_buy: false, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_queue_market_params_fails_if_not_market_owner
↪ (gas: ~4402)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_if_range_zero (gas:
↪ ~4675)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_queue_market_params_fails_if_params_unchanged
↪ (gas: ~4403)
[PASS] haiko_solver_replicating::tests::solver::test_deposit_initial::test_deposit_initial_private_quote_token_only
↪ (gas: ~5741)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_if_min_sources_zero
↪ (gas: ~4675)
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_unset_delay (gas: ~4401)
amount in: 100000000000000000000, amount out: 99654250398634336911
amount in: 100000000000000000000, amount out: 9090903631499880357 7
Test Case 14) Swap capped at threshold amount, exact input
Swap Case 2) is_buy: false, exact_input: true
Test Case 3) Full range liq, price 10, no spread
[PASS] haiko_solver_replicating::tests::solver::test_deploy::test_deploy_replicating_solver_initialises_immutables
↪ (gas: ~4400)
```

```
Swap Case 3) is_buy: true, exact_input: false
amount in: 100000000000000000000, amount out: 119678665348291756880
Test Case 8) Concentrated liq, price 0.1, 100 spread, 20000 max delta
Swap Case 3) is_buy: true, exact_input: false
amount in: 100000000000000000000, amount out: 99654250398634336911
Test Case 15) Swap capped at threshold amount, exact output
Swap Case 1) is_buy: true, exact_input: false
amount in: 1111114882320518862800, amount out: 1000000000000000000000
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_next_sqrt_price_out_output_gt_quote_reserves
 ↪  (gas: ~5)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib::test_get_swap_amounts_succeeds (gas: ~72)
[PASS] haiko_solver_replicating::tests::libraries::test_spread_math::test_get_virtual_position_range_ask_limit_overflow
 ↪  (gas: ~415)
Test Case 3) Full range liq, price 10, no spread
Swap Case 4) is_buy: false, exact_input: false
amount in: 12055018117706607775, amount out: 100000000000000000000
[PASS] haiko_solver_replicating::tests::solver::test_market_params::test_set_market_params_fails_none_queued (gas:
 ↪  ~4470)
amount in: 100347807913318736435, amount out: 100000000000000000000
Test Case 8) Concentrated liq, price 0.1, 100 spread, 20000 max delta
Swap Case 4) is_buy: false, exact_input: false
[PASS] haiko_solver_replicating::tests::libraries::test_spread_math::test_get_virtual_position_cases (gas: ~1635)
[PASS] haiko_solver_replicating::tests::solver::test_e2e::test_solver_e2e_private_market (gas: ~5261)
Test Case 15) Swap capped at threshold amount, exact output
Swap Case 2) is_buy: false, exact_input: false
amount in: 466588818773008687984714281083890117, amount out: 99999999999999999998
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_fails_if_invalid_oracle_price (gas: ~4090)
Test Case 4) Concentrated liq, price 1, no spread
Swap Case 1) is_buy: true, exact_input: true
amount in: 837391432418576103405, amount out: 100000000000000000000
amount in: 100347807913318736435, amount out: 100000000000000000000
Test Case 9) Swap with liquidity exhausted
Swap Case 1) is_buy: true, exact_input: true
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_cases_3 (gas: ~63780)
amount in: 100000000000000000000, amount out: 99753708432456984326
Test Case 4) Concentrated liq, price 1, no spread
Swap Case 2) is_buy: false, exact_input: true
amount in: 102634081505001697490, amount out: 99999999999999999999
Test Case 9) Swap with liquidity exhausted
Swap Case 2) is_buy: false, exact_input: true
amount in: 100000000000000000000, amount out: 99753708432456984326
Test Case 4) Concentrated liq, price 1, no spread
Swap Case 3) is_buy: true, exact_input: false
amount in: 102634081505001697490, amount out: 99999999999999999999
Test Case 10) Swap with high oracle price
Swap Case 1) is_buy: true, exact_input: true
amount in: 100247510763823131035, amount out: 100000000000000000000
Test Case 4) Concentrated liq, price 1, no spread
Swap Case 4) is_buy: false, exact_input: false
amount in: 100000000000000000000, amount out: 99899
amount in: 100247510763823131035, amount out: 100000000000000000000
Test Case 10) Swap with high oracle price
Swap Case 2) is_buy: false, exact_input: true
Test Case 5) Concentrated liq, price 1, 100 spread
Swap Case 1) is_buy: true, exact_input: true
amount in: 100000000000000000000, amount out: 99654250398634336911
Test Case 5) Concentrated liq, price 1, 100 spread
Swap Case 2) is_buy: false, exact_input: true
amount in: 1026351, amount out: 99999999999999345153
Test Case 10) Swap with high oracle price
Swap Case 3) is_buy: true, exact_input: false
amount in: 100000000000000000000, amount out: 99654250398634336911
Test Case 5) Concentrated liq, price 1, 100 spread
Swap Case 3) is_buy: true, exact_input: false
amount in: 100347899379444510663740995390777206, amount out: 100000000000000000000
amount in: 100347807913318736435, amount out: 100000000000000000000
Test Case 10) Swap with high oracle price
Swap Case 4) is_buy: false, exact_input: false
Test Case 5) Concentrated liq, price 1, 100 spread
Swap Case 4) is_buy: false, exact_input: false
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib_invariants::test_compute_swap_amounts_invariants
 ↪  (runs: 256, gas: {max: ~27, min: ~10, mean: ~19.00, std deviation: ~6.52})
```

```
amount in: 100347807913318736435, amount out: 1000000000000000000000
amount in: 100349, amount out: 1000000000000000000000
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_cases_1 (gas: ~108033)
[PASS] haiko_solver_replicating::tests::solver::test_swap::test_swap_cases_2 (gas: ~105548)
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib_invariants::test_next_sqrt_price_output_invariants
↪  (runs: 256, gas: {max: ~17, min: ~5, mean: ~8.00, std deviation: ~1.17})
[PASS] haiko_solver_replicating::tests::libraries::test_swap_lib_invariants::test_next_sqrt_price_input_invariants
↪  (runs: 256, gas: {max: ~16, min: ~4, mean: ~12.00, std deviation: ~3.60})
Tests: 104 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out
Fuzzer seed: 10641026143009073523
```

Remarks about Haiko test suite

The **Haiko** team built a comprehensive test suite that checks the actor's workflows, including liquidity management and swapping. The test suite is broad and covers various edge cases. Furthermore, it includes fuzzing tests for specific protocol invariants that should be maintained during swapping operations.

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development procehttps://www.overleaf.com/project/65c0e737f41a29601bda5c48ss, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.