Security Review Report
NM-0271 Avnu

NETHERMIND
SECURITY

(August 2, 2024)

Contents

1 Executive Summary

This document outlines the security review conducted by Nethermind Security for the Avnu Cairo contracts. Avnu is a decentralized exchange protocol designed to offer the best execution in DeFi for Layer 2. Their primary goal is to provide better pricing, zero slippage, MEV protection, and gasless trading, ensuring the most efficient and seamless operations in the DeFi space.

This review focuses on four different components:

- A set of components created to be reused across the different contracts that composed the Avnu ecosystem;
- A forwarder contract that allows users to pay execution fees using any different tokens;
- A set of contracts that can be used by users to create and execute DCA strategies;
- A new entry point added to the core contracts that permit the execution of trades for an exact amount of tokens.

The audited code comprises approximately 1100 lines of code. The Avnu has provided specific information for each component in addition to the official documentation on their website. Besides the written documentation, the Avnu team has been available to answer any questions the Nethermind Security team raised.

The audit was performed using (a) manual analysis of the codebase, and (b) simulation of the smart contract. Along with this document, we report 1 points of attention, classified as Low 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 describes 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.

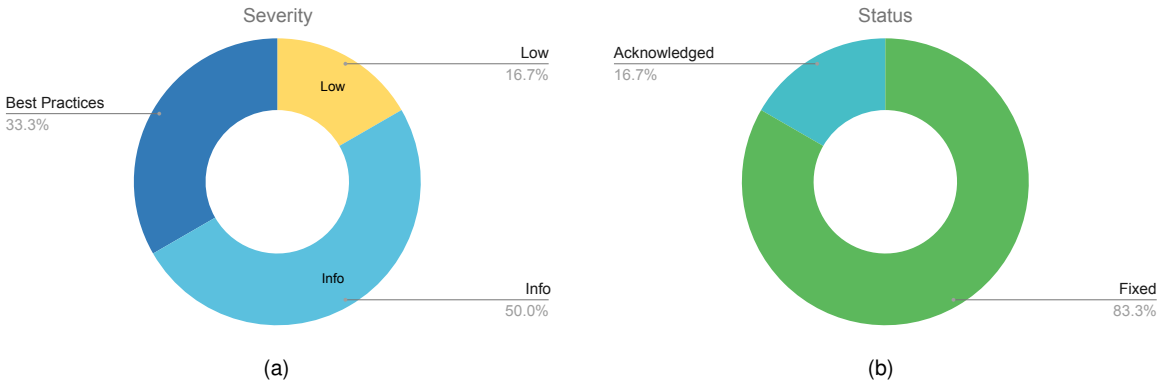

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (1), **Undetermined** (0), **Informational** (3), **Best Practices** (2). **Distribution of status: Fixed** (5), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0),

Summary of the Audit

Audit Type	Security Review
Initial Report	July 26, 2024
Final Report	August 2, 2024
Repositories	avnu-contracts-lib gasless-contracts-private dca-contracts-private avnu-contracts-v2-private
Documentation	Official Docs
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

2.1 avnu-contracts-lib

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/interfaces.cairo	1	0	0.0%	0	1
2	src/components.cairo	3	0	0.0%	0	3
3	src/lib.cairo	3	0	0.0%	1	4
4	src/math.cairo	1	0	0.0%	0	1
5	src/math/muldiv.cairo	25	2	8.0%	6	33
6	src/components/whitelist.cairo	26	6	23.1%	6	38
7	src/components/upgradable.cairo	30	7	23.3%	6	43
8	src/components/ownable.cairo	50	11	22.0%	10	71
9	src/interfaces/erc20.cairo	8	1	12.5%	1	10
	Total	**147**	**27**	**18.4%**	**30**	**204**

2.2 gasless-contracts

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/forwarder.cairo	81	21	25.9%	18	120
2	src/lib.cairo	1	0	0.0%	0	1
	Total	**82**	**21**	**25.6%**	**18**	**121**

2.3 dca-contracts

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/components.cairo	2	0	0.0%	0	2
2	src/orchestrator.cairo	172	33	19.2%	27	232
3	src/order.cairo	217	26	12.0%	30	273
4	src/lib.cairo	3	0	0.0%	1	4
5	src/components/ownable.cairo	30	7	23.3%	7	44
6	src/components/fee.cairo	60	9	15.0%	13	82
	Total	**484**	**75**	**15.5%**	**78**	**637**

2.4 avnu-contracts-v2

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/exchange.cairo	446	58	13.0%	73	577
	Total	**446**	**58**	**13.0%**	**73**	**577**

For this repository, only the diff between the commits *7c140b2af9d291995aabbc2d6811749419f56aa7* and *df7f0bb4694f135d4dc91392d3f8187eb4e19faa* was in scope.

3 Summary of Issues

	Finding	Severity	Update
1	The function `execute_batch(...)` may retrieve wrong tokens from traders	Low	Fixed
2	Additional `token_to` may be transferred to traders	Info	Fixed
3	DCA orders can be created with same `token_from_address` and `token_to_address`	Info	Fixed
4	Orders may remain `open` even when they are not executable	Info	Fixed
5	Traders can make `execute_batch(...)` function revert	Info	Acknowledged
6	Unused parameters	Best Practices	Fixed
7	Use of arbitrary calls for well defined interactions	Best Practices	Fixed

4 System Overview

This security review relies on four components:

- **Reusable Components**: A collection of reusable components designed for the various contracts within the AVNU ecosystem;
- **Gasless contract**: A forwarder contract enabling users to pay execution fees with different tokens;
- **Dollar Cost Averaging (DCA) Contracts**: A suite of contracts allowing users to create and execute DCA strategies;
- **AVNUExchange**: A new entry point added to the core contract, permitting the execution of trades for an exact amount of tokens.

4.1 AVNUExchange

The new entry point added to the core contract implements the feature **Exact amount out swap**

This feature allows users to specify the exact amount of tokens they want to purchase. ANVU's solvers calculate the minimal amount of another token required to get that amount.

Supported Liquidity Sources: This feature primarily works with liquidity sources that support it at the contract level. However, this approach can limit available liquidity, potentially resulting in poor quotes and non-tradable tokens. Despite these constraints, AVNU's system is designed to support all sources, even those that don't natively support this type of swap.

Slippage: Slippage has a different meaning in the exact amount out of swap context. While in the standard trade, slippage reflects the tolerated negative price variation, in the context of the swap exact, considering the slippage on the amount out is illogical. This is because *swap exact* means the user wants a precise amount out without any variation. Allowing this amount to fluctuate would violate the agreement. Instead, slippage is applied to the input amount (X) to show the user's willingness to trade more if the initial input isn't sufficient to obtain the desired output amount (Y).

Fees: this feature constrains the value to the exact amount the user wants, taking any surplus as a fee.

4.2 Reusable components

They represent a collection of components that are used within the AVNU ecosystem. For example:

- `Ownable`: a component used to manage ownership of a contract;
- `Upgradable`: this component allows to upgrade the contract code;
- `Whitelist`: this component enables contracts to manage a whitelist of addresses.

4.3 Dollar Cost Averaging (DCA) Contracts

This part consists of contracts that are used to provide functionalities related to DCA. The contracts

- `Orchestrator`: this contract manages the DCA process. Only authorized callers can initiate DCA batches. During batch execution, it collects sell tokens from all DCA orders, swaps them, sends the swapped tokens to beneficiaries, and ensures compliance with pricing strategies.
- `Order`: This contract represents a DCA order containing details such as the token to buy, the token to sell, the amount to buy, the amount to sell, and the pricing strategy. It ensures the correct execution of the order.

4.4 Gasless Contracts

This is a repository that contains the `Forwarder` contract that provides two main functions:

- Function `execute` It checks if the caller is authorized (only whitelisted relayers can execute user calls), executes the user calls and collects the user's gas tokens.
- Function `execute_no_fee`: performs the same actions as execute but does not collect users' gas tokens.

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 the finding is to be uncovered and exploited by an attacker. 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 pass to the client formally;

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 [Low] The function `execute_batch(...)` may retrieve wrong tokens from traders

File(s): src/orchestrator.cairo, src/order.cairo

Description: The whitelisted caller invokes the function `execute_batch(...)` to perform swaps between `token_from` and `token_to` for multiple traders. Before executing the swap, the function calls `retrieve_token_from(...)` to transfer the `token_from` of all traders to the DCA orchestrator.

```
fn execute_batch(...) -> bool {
    // ...
    assert(self.whitelist.is_whitelisted(caller_address), 'Caller is not whitelisted');

    // ...
    // @audit-info transfer token_from of all traders to the DCA orchestrator
    let traders = self.retrieve_token_from(contract_address, token_from, traders.span());
    let token_from_amount = token_from.balanceOf(contract_address);

    // Execute Swap
    while let Option::Some(call) = calls
        .pop_front() {
            call_contract_syscall(call.to, call.selector, call.calldata).unwrap_syscall();
        };
     // ...
}
```

As presented below, the function `retrieve_token_from` receives the `contract_address` and the `token_from` instance. For each trader, the function calls `order.initiate_order_execution`. However, the `contract_address` and the `token_from`are not passed to the order contract to be validated.

```
fn retrieve_token_from(//ok
  ref self: ContractState, contract_address: ContractAddress, token_from: IERC20Dispatcher, mut traders: Span<Trader>
) -> Array<TraderWithTokenFromAmount> {
  // @audit unused parameters: token_from and contract_address
  let mut traders_with_token_from_amount: Array<TraderWithTokenFromAmount> = array![];
  while let Option::Some(trader) = traders
      .pop_front() {
          let token_from_amount = IDcaOrderDispatcher { contract_address: *trader.order_address
           ↪  }.initiate_order_execution();
          traders_with_token_from_amount
              .append(
                  TraderWithTokenFromAmount { address: *trader.address, order_address: *trader.order_address,
                   ↪  token_from_amount }
              )
      };
  traders_with_token_from_amount
}
```

The issue arises in `order.initiate_order_execution` since the `token_from` address is not verified with `order.token_from_address`. Thus, the expected `token_from` defined in `execute_batch` may be different from the one in `order.token_from_address`.

```
fn initiate_order_execution(ref self: ContractState) -> u256 {
    self.assert_only_orchestrator();
    let trader_address = self.ownable.get_owner();
    let contract_address = get_contract_address();
    let mut order = self.order.read();
    let token_from = IERC20Dispatcher { contract_address: order.token_from_address };
    //...
}
```

Recommendation(s): Check in the initiate_order_execution if the `token_from` address is the same as `order.token_from_address`.

Status: Fixed.

Update from the client: Fixed at commit 520e87e0f71f37fc146ebe4888226d191f233206.

6.2 [Info] Additional `token_to` may be transferred to traders

File(s): `src/orchestrator.cairo`

Description: The function `execute_batch` performs swaps from `token_from` to `token_to` for multiple traders. The function checks the contract balance for `token_from` before collecting all tokens from traders. When the `token_from_amount` is greater than zero, the contract's remaining `token_from` amount is transferred to the `fees_recipient`.

```
fn execute_batch(
    ref self: ContractState,
    traders: Array<Trader>,
    token_from_address: ContractAddress,
    token_to_address: ContractAddress,
    token_to_min_amount: u256,
    mut calls: Array<Call>
) -> bool {
    // ...

    // Collect remaining token from
    let token_from_amount = token_from.balanceOf(contract_address);
    if token_from_amount > 0 {
        let fees_recipient = self.fee.get_fees_recipient();
        assert(!fees_recipient.is_zero(), 'Fees recipient is zero');
        token_from.transfer(fees_recipient, token_from_amount);
    }

    // Retrieve all token from
    let traders = self.retrieve_token_from(contract_address, token_from, traders.span());
    let token_from_amount = token_from.balanceOf(contract_address);

    // @audit the function does not check if the contract balance for token_to before executing swap

    // Execute Swap
    while let Option::Some(call) = calls
        .pop_front() {
            call_contract_syscall(call.to, call.selector, call.calldata).unwrap_syscall();
        };
    // ...
}
```

However, the same operation is not applied to `token_to`; i.e., the function does not collect the remaining `token_to` from the `fees_recipient`. If the balance is greater than zero, then all the `token_to` will be transferred to the traders.

Recommendation(s): Ensure that all remaining `token_to` tokens are transferred to the `fees_recipient` before executing swap.

Status: Fixed.

Update from the client: Fix at commit c27c053a3e162a6bc8f7c0b1cd45e97655ca086a.

6.3 [Info] DCA orders can be created with same `token_from_address` `token_to_address`

File(s): `src/order.cairo`

Description: The `order.cairo` contract allows a user to create a DCA order mechanism to invest at regular intervals based on specific conditions. The contract constructor contains multiple validation checks to ensure that each parameter is correct. However, it does not prevent the creation of an order where the `token_from_address` and `token_to_address` are the same, which would result in an impractical order.

Recommendation(s): Consider adding a validation rule to prevent the creation of a DCA order with identical `token_from_address` and `token_to_address` to ensure robustness and maintain the protocol's integrity.

Status: Fixed.

Update from the client: Fix at commit 752bb5218d86c2b372b1431047278cc8d7455e86.

6.4 [Info] Orders may remain `open` even when they are not executable

File(s): src/order.cairo

Description: To create an `Order`, users specify multiple parameters, such as the total amount they want to trade, the amount they want to trade in each cycle, the start date of the `Order`, and the cycle frequency of the trades. Using these parameters, the contract computes the end date of the `Order`.

The `Order` contract uses the function `assert_next_trade_available` to check if the `Order` can be executed at the current time.

```
fn assert_next_trade_available(...) {
    // The order must be open
    assert(*order.open == true, 'Order is not open');

    // The order must be open
    assert(block_timestamp <= *order.end_date, 'End date has passed');

    // Check if next cycle is available
    assert(block_timestamp >= *order.next_cycle_at, 'Next cycle not yet available');

    // Allowance should be high enough
    let token_from_allowance = token_from.allowance(trader_address, contract_address);
    assert(token_from_allowance >= *order.token_from_amount_per_cycle, 'Allowance is too low');

    // User's balance should be high enough
    let token_from_balance = token_from.balanceOf(trader_address);
    assert(token_from_balance >= *order.token_from_amount_per_cycle, 'Balance is too low');
}
```

As can be seen from the code, one requirement to execute a trade is that the end date must not be reached yet. However, this end date was computed assuming all the cycles for the `Order` will be executed, meaning that if at least one cycle is skipped, the end date will be reached without swapping the full amount of the `Order`.

The main problem with the described state is that `Orders` are closed when the full amount is swapped. This could result in `Orders` that cannot be executed because their end time was reached, but they are never closed because their full amount was not swapped. These `Orders` can only be closed manually by their owners.

Recommendation(s): Consider closing `Orders` when their end time is reached.

Status: Fixed.

Update from the client: Fix at commit 376a7142d32ef5fce391ba01a818b2e6671d2333.

6.5 [Info] Traders can make `execute_batch(...)` function revert

File(s): src/orchestrator.cairo

Description: To create `Orders` traders do not need to commit any assets, they just set the correct allowance for the `Order` contract. To call `execute_batch(...)` the caller will create off-chain the set of `Orders` that will be part of the batch and check that these `Orders` can actually be executed and the correct allowance is set. However, if a trader removes its allowance between the time of these checks and the time of actual execution it would cause the `execute_batch(...)` function to fail. This could be abuse by traders to make calls to `execute_batch(...)` to revert. Note that traders do not have direct incentives for causing this issue, also in the current state it may not be feasible for traders to frontrun the call to `execute_batch(...)` in order to remove the allowance.

Recommendation(s): Consider forcing trades to commit funds for at least the next cycle in order to avoid this issue.

Status: Acknowledged.

6.6 [Best Practice] Unused parameters

File(s): src/orchestrator.cairo

Description: The function `retrieve_token_from(...)` contains two parameters that are never used.

```
fn retrieve_token_from(
    ref self: ContractState,
    contract_address: ContractAddress,
    token_from: IERC20Dispatcher,
    mut traders: Span<Trader>
) -> Array<TraderWithTokenFromAmount> {
    // @audit unused parameter token_from and contract_address
    // ...
}
```

In function `send_token_to` the parameter `contract_address` is never used.

```
fn send_token_to(
    ref self: ContractState,
    contract_address: ContractAddress,
    token_from_initial_amount: u256,
    token_to: IERC20Dispatcher,
    token_to_amount: u256,
    mut traders: Span<TraderWithTokenFromAmount>
) -> Array<TraderWithTokenFromAndToAmounts> {
    // @audit contract_address parameter is never used
    // ...
}
```

Recommendation(s): Consider using `token_from` in the `initiate_order_execution` to verify if the `order.token_from_address` is the same as `token_from` and remove `contract_address` parameters for both functions.

Status: Fixed.

Update from the client: Fix at commit 572237a7bd0814d59783c8b35c7096f56f711778.

6.7 [Best Practice] Use of arbitrary calls for well defined interactions

File(s): src/orchestrator.cairo

Description: The `orchestrator` contract purpose is to execute the `DCA` orders created by users. For executing these, a whitelisted caller from `AVNU` provides a set of trades to apply through the `AVNU` protocol.

```
fn execute_batch(...) -> bool {
    ...
    // Execute Swap
    while let Option::Some(call) = calls
        .pop_front() {
            // @audit - Use of arbitrary calls
            call_contract_syscall(call.to, call.selector, call.calldata).unwrap_syscall();
        };
    ...
}
```

As the snippet of code shows, these actions can be totally arbitrary even if they are expected to be traded on the `AVNU` protocol.

Recommendation(s): Consider replacing the arbitrary calls for calls to the `AVNU` protocol to decrease the probability of unexpected executions.

Status: Fixed.

Update from the client: Fix at commit 97e5798b2ec8c14a681daef5a75281330df5537a.

7 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. 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 provide a reference for developers who need to modify or maintain it in the future.

Remarks about the Avnu documentation

The documentation for Avnu is public available on their website. Beyond this, the AVNU team has provided extra documentation for each reviewed component in the format of Readme files. The team has been available to answer any questions the Nethermind Security team raised.

8 Test Suite Evaluation

8.1 Contracts Compilation

avnu-contracts-lib

```
scarb --version; scarb build
scarb 2.6.3 (e6f921dfd 2024-03-13)
cairo: 2.6.3 (https://crates.io/crates/cairo-lang-compiler/2.6.3)
sierra: 1.5.0

   Compiling lib(avnu_lib) avnu_lib v0.1.0 (.../avnu-contracts-lib/Scarb.toml)
   Compiling starknet-contract(avnu_lib) avnu_lib v0.1.0 (.../avnu-contracts-lib/Scarb.toml)
    Finished release target(s) in 3 seconds
```

gasless-contracts

```
scarb 2.6.3 (e6f921dfd 2024-03-13)
cairo: 2.6.3 (https://crates.io/crates/cairo-lang-compiler/2.6.3)
sierra: 1.5.0

   Compiling avnu v0.1.0 (.../gasless-contracts-private/Scarb.toml)
    Finished release target(s) in 2 seconds
```

dca-contracts

```
scarb 2.6.3 (e6f921dfd 2024-03-13)
cairo: 2.6.3 (https://crates.io/crates/cairo-lang-compiler/2.6.3)
sierra: 1.5.0

   Compiling avnu_dca v0.1.0 (.../dca-contract-private/Scarb.toml)
    Finished release target(s) in 2 seconds
```

avnu-contracts-v2

```
scarb 2.6.3 (e6f921dfd 2024-03-13)
cairo: 2.6.3 (https://crates.io/crates/cairo-lang-compiler/2.6.3)
sierra: 1.5.0

   Compiling avnu v1.0.0 (.../avnu-contracts-v2-private/Scarb.toml)
    Finished release target(s) in 3 seconds
```

8.2 Tests Output

avnu-contracts-lib

```
scarb --version; scarb test
scarb 2.6.3 (e6f921dfd 2024-03-13)
cairo: 2.6.3 (https://crates.io/crates/cairo-lang-compiler/2.6.3)
sierra: 1.5.0

     Running cairo-test avnu_lib
   Compiling test(avnu_lib_unittest) avnu_lib v0.1.0 (.../avnu-contracts-lib/Scarb.toml)
   Compiling test(avnu_lib_tests) avnu_lib_tests v0.1.0 (.../avnu-contracts-lib/Scarb.toml)
    Finished release target(s) in 3 seconds
testing avnu_lib ...
running 25 tests
test avnu_lib_tests::math::muldiv_test::test_muldiv_div_by_zero ... ok (gas usage est.: 48900)
test avnu_lib_tests::math::muldiv_test::test_muldiv_up_div_by_zero ... ok (gas usage est.: 48900)
test avnu_lib_tests::math::muldiv_test::test_muldiv_max_inputs ... ok (gas usage est.: 49900)
test avnu_lib_tests::math::muldiv_test::test_muldiv_overflows_by_more ... ok (gas usage est.: 50100)
test avnu_lib_tests::math::muldiv_test::test_muldiv_up_div_by_zero_no_overflow ... ok (gas usage est.: 48900)
test avnu_lib_tests::components::ownable_test::TransferOwnership::should_fail_when_caller_is_not_the_owner ... ok (gas
↪  usage est.: 322450)
test avnu_lib_tests::components::whitelist_test::IsWhitelisted::should_return_a_bool ... ok (gas usage est.: 279580)
test avnu_lib_tests::components::upgradable_test::UpgradeClass::should_fail_when_caller_is_not_the_owner ... ok (gas
↪  usage est.: 311040)
test avnu_lib_tests::math::muldiv_test::test_muldiv_up_max_inputs_no_rounding ... ok (gas usage est.: 49900)
test avnu_lib_tests::math::muldiv_test::test_muldiv_overflows_exactly ... ok (gas usage est.: 49900)
test avnu_lib_tests::components::ownable_test::GetOwner::should_return_owner ... ok (gas usage est.: 272040)
test avnu_lib_tests::components::whitelist_test::SetWhitelistedCaller::should_fail_when_caller_is_not_the_owner ... ok
↪  (gas usage est.: 306790)
test avnu_lib_tests::math::muldiv_test::test_muldiv_fits ... ok (gas usage est.: 50100)
test avnu_lib_tests::math::muldiv_test::test_muldiv_phantom_overflow ... ok (gas usage est.: 49900)
test avnu_lib_tests::components::upgradable_test::UpgradeClass::should_fail_when_new_class_hash_is_zero ... ok (gas
↪  usage est.: 305040)
test avnu_lib_tests::math::muldiv_test::test_muldiv_up_fits_no_rounding ... ok (gas usage est.: 50100)
test avnu_lib_tests::math::muldiv_test::test_muldiv_up_phantom_overflow_no_rounding ... ok (gas usage est.: 49900)
test avnu_lib_tests::math::muldiv_test::test_muldiv_up_overflow_with_rounding ... ok (gas usage est.: 49300)
test avnu_lib_tests::components::ownable_test::TransferOwnership::should_fail_when_owner_is_0 ... ok (gas usage est.:
↪  410610)
test avnu_lib_tests::math::muldiv_test::test_div ... ok (gas usage est.: 72480)
test avnu_lib_tests::math::muldiv_test::test_muldiv_up_no_overflow_rounding_min ... ok (gas usage est.: 49900)
test avnu_lib_tests::components::ownable_test::TransferOwnership::should_change_owner ... ok (gas usage est.: 429010)
test avnu_lib_tests::components::whitelist_test::SetWhitelistedCaller::should_set_whitelisted_addresses ... ok (gas
↪  usage est.: 415290)
test avnu_lib_tests::components::upgradable_test::UpgradeClass::should_upgrade_class ... ok (gas usage est.: 331160)
test avnu_lib_tests::components::upgradable_test::UpgradeClass::should_fail_when_class_does_not_exist ... ok (gas usage
↪  est.: 316040)
test result: ok. 25 passed; 0 failed; 0 ignored; 0 filtered out;

running 0 tests
test result: ok. 0 passed; 0 failed; 0 ignored; 0 filtered out;
```

gasless-contracts

```
scarb --version; scarb test
scarb 2.6.3 (e6f921dfd 2024-03-13)
cairo: 2.6.3 (https://crates.io/crates/cairo-lang-compiler/2.6.3)
sierra: 1.5.0

     Running cairo-test avnu
   Compiling test(avnu_unittest) avnu v0.1.0 (.../gasless-contracts-private/Scarb.toml)
   Compiling test(avnu_tests) avnu_tests v0.1.0 (.../gasless-contracts-private/Scarb.toml)
    Finished release target(s) in 3 seconds
testing avnu ...
running 7 tests
test avnu_tests::forwarder_test::ExecuteNoFee::should_fail_when_caller_is_not_whitelisted ... ok (gas usage est.:
↪  357440)
test avnu_tests::forwarder_test::SetGasFessRecipient::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.:
↪  329260)
test avnu_tests::forwarder_test::Execute::should_fail_when_caller_is_not_whitelisted ... ok (gas usage est.: 428120)
test avnu_tests::forwarder_test::GetGasFessRecipient::should_return_gas_fess_recipient ... ok (gas usage est.: 303050)
test avnu_tests::forwarder_test::SetGasFessRecipient::should_set_gas_fess_recipient ... ok (gas usage est.: 524480)
test avnu_tests::forwarder_test::ExecuteNoFee::should_execute ... ok (gas usage est.: 751340)
test avnu_tests::forwarder_test::Execute::should_execute ... ok (gas usage est.: 1974140)
test result: ok. 7 passed; 0 failed; 0 ignored; 0 filtered out;

running 0 tests
test result: ok. 0 passed; 0 failed; 0 ignored; 0 filtered out;
```

dca-contracts

```
scarb test
     Running test avnu_dca (
   Compiling test(avnu_dca_unittest) avnu_dca v0.1.0 (.../dca-contract-private/dca-contract-private/Scarb.toml)
   Compiling test(avnu_dca_tests) avnu_dca_tests v0.1.0 (.../dca-contract-private/dca-contract-private/Scarb.toml)
    Finished release target(s) in 28 seconds
testing avnu_dca ...
running 44 tests
test avnu_dca_tests::components::fee_test::GetFessRecipient::should_return_fees_recipient ... ok (gas usage est.:
↪  308220)
test avnu_dca_tests::order_test::Constructor::should_fail_when_token_from_amount_per_cycle_is_zero ... ok (gas usage
↪  est.: 1298320)
test avnu_dca_tests::components::fee_test::SetFessRecipient::should_set_fess_recipient ... ok (gas usage est.: 435490)
test avnu_dca_tests::components::fee_test::SetFessRecipient::should_fail_when_caller_is_not_the_owner ... ok (gas usage
↪  est.: 334430)
test
↪  avnu_dca_tests::order_test::Constructor::should_fail_when_token_from_amount_per_cycle_higher_than_token_from_amount
↪  ... ok (gas usage est.: 1298320)
test avnu_dca_tests::components::fee_test::GetFeesBps::should_return_bps ... ok (gas usage est.: 308620)
test avnu_dca_tests::order_test::Constructor::should_fail_when_too_many_iterations ... ok (gas usage est.: 1298320)
test avnu_dca_tests::components::fee_test::SetFeesBps0::should_set_fees_bps_0 ... ok (gas usage est.: 437820)
test avnu_dca_tests::components::fee_test::SetFeesBps0::should_fail_when_fees_are_too_high ... ok (gas usage est.:
↪  336360)
test avnu_dca_tests::order_test::Constructor::should_fail_when_not_enough_iterations ... ok (gas usage est.: 1298320)
test avnu_dca_tests::components::fee_test::SetFeesBps0::should_fail_when_caller_is_not_the_owner ... ok (gas usage
↪  est.: 336360)
test avnu_dca_tests::components::ownable_test::GetOwner::should_return_owner ... ok (gas usage est.: 272040)
test avnu_dca_tests::order_test::Constructor::should_fail_when_cycle_frequency_is_too_low ... ok (gas usage est.:
↪  1297750)
test avnu_dca_tests::orchestrator_test::GetAvnuExchangeAddress::should_return_the_avnu_exchange_address ... ok (gas
↪  usage est.: 519620)
test avnu_dca_tests::order_test::Constructor::should_fail_when_cycle_frequency_is_too_high ... ok (gas usage est.:
↪  1299260)
test avnu_dca_tests::orchestrator_test::SetAvnuExchangeAddress::should_change_address ... ok (gas usage est.: 551870)
test avnu_dca_tests::orchestrator_test::SetAvnuExchangeAddress::should_fail_when_caller_is_not_the_owner ... ok (gas
↪  usage est.: 451310)
test avnu_dca_tests::order_test::Constructor::should_fail_when_allows_is_too_low ... ok (gas usage est.: 1104860)
test avnu_dca_tests::orchestrator_test::SetAvnuExchangeAddress::should_fail_when_address_is_0 ... ok (gas usage est.:
↪  445310)
```

```
test
avnu_dca_tests::order_test::IsNextTradeAvailable::should_return_true_when_next_trade_is_available ... ok (gas usage
↪  est.: 2281430)
test avnu_dca_tests::order_test::IsNextTradeAvailable::should_fail_when_order_is_not_open ... ok (gas usage est.:
↪  3110860)
test avnu_dca_tests::order_test::IsNextTradeAvailable::should_fail_when_next_cycle_not_available ... ok (gas usage
↪  est.: 2080620)
test avnu_dca_tests::order_test::IsNextTradeAvailable::should_fail_when_date_is_after_end_date ... ok (gas usage est.:
↪  2061990)
test avnu_dca_tests::order_test::CloseOrder::should_close_order ... ok (gas usage est.: 3072810)
test avnu_dca_tests::order_test::CloseOrder::should_close_order_when_end_date_has_passed_and_call_is_not_the_owner ...
↪  ok (gas usage est.: 2814840)
test avnu_dca_tests::order_test::CloseOrder::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 2312300)
test avnu_dca_tests::order_test::CloseOrder::should_fail_when_allowance_is_not_zero ... ok (gas usage est.: 2427610)
test avnu_dca_tests::orchestrator_test::Execute::should_execute_a_dca_batch_with_multiple_traders ... ok (gas usage
↪  est.: 18566870)
test avnu_dca_tests::order_test::CloseOrder::should_fail_when_order_is_already_close ... ok (gas usage est.: 3097240)
test avnu_dca_tests::order_test::FulfillOrderExecution::should_check_pricing_strategy ... ok (gas usage est.: 2071130)
test avnu_dca_tests::order_test::FulfillOrderExecution::should_fail_when_token_to_amount_is_too_low ... ok (gas usage
↪  est.: 2070230)
test avnu_dca_tests::order_test::FulfillOrderExecution::should_fail_when_token_to_amount_is_too_high ... ok (gas usage
↪  est.: 2070230)
test avnu_dca_tests::orchestrator_test::Execute::should_execute_a_dca_batch ... ok (gas usage est.: 7409950)
test avnu_dca_tests::order_test::FulfillOrderExecution::should_fail_when_caller_is_not_the_orchestrator ... ok (gas
↪  usage est.: 1980230)
test avnu_dca_tests::order_test::InitiateOrderExecution::should_close_order_when_execute_latest_iteration ... ok (gas
↪  usage est.: 5215290)
test avnu_dca_tests::order_test::InitiateOrderExecution::should_throw_error_when_invalid_token_from ... ok (gas usage
↪  est.: 3722820)
test
↪  avnu_dca_tests::orchestrator_test::Execute::should_execute_a_dca_batch_and_send_remaining_token_from_to_fee_recipient
↪  ... ok (gas usage est.: 7943610)
test avnu_dca_tests::orchestrator_test::Execute::should_fail_when_caller_is_not_whitelisted ... ok (gas usage est.:
↪  2434110)
test avnu_dca_tests::orchestrator_test::Execute::should_fail_when_token_from_address_is_0 ... ok (gas usage est.:
↪  2563220)
test avnu_dca_tests::orchestrator_test::Execute::should_fail_when_token_to_amount_is_not_sufficient ... ok (gas usage
↪  est.: 5533020)
test avnu_dca_tests::orchestrator_test::Execute::should_fail_when_token_to_address_is_0 ... ok (gas usage est.: 2563220)
test avnu_dca_tests::orchestrator_test::Execute::should_fail_when_traders_is_empty ... ok (gas usage est.: 1473390)
test avnu_dca_tests::orchestrator_test::Execute::should_fail_when_calls_is_empty ... ok (gas usage est.: 2553540)
test avnu_dca_tests::order_test::Constructor::should_init_order ... ok (gas usage est.: 3744340)
test result: ok. 44 passed; 0 failed; 0 ignored; 0 filtered out;
```

avnu-contracts-v2

```
scarb --version; scarb test
scarb 2.6.3 (e6f921dfd 2024-03-13)
cairo: 2.6.3 (https://crates.io/crates/cairo-lang-compiler/2.6.3)
sierra: 1.5.0

     Running cairo-test avnu
   Compiling test(avnu_unittest) avnu v1.0.0
    ↪  (/Users/mauricio/Work/Nethermind/Audits/NM-0271-Avnu-Eagle/avnu-contracts-v2-private/Scarb.toml)
   Compiling test(avnu_tests) avnu_tests v1.0.0
    ↪  (/Users/mauricio/Work/Nethermind/Audits/NM-0271-Avnu-Eagle/avnu-contracts-v2-private/Scarb.toml)
    Finished release target(s) in 5 seconds
testing avnu ...
running 67 tests
test avnu_tests::adapters::jediswap_adapter_test::Swap::should_fail_when_invalid_additional_swap_params ... ok (gas
↪  usage est.: 872440)
test avnu_tests::exchange_test::SetAdapterClassHash::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.:
↪  592760)
test avnu_tests::exchange_test::GetOwner::should_return_owner ... ok (gas usage est.: 558200)
test avnu_tests::exchange_test::SetSwapExactTokenToFeesBps::should_fail_when_fees_are_too_high ... ok (gas usage est.:
↪  678700)
```

```
test avnu_tests::adapters::myswap_adapter_test::Swap::should_call_myswap ... ok (gas usage est.: 1062760)
test avnu_tests::adapters::jediswap_adapter_test::Swap::should_call_jediswap ... ok (gas usage est.: 1115380)
test avnu_tests::adapters::sithswap_adapter_test::Swap::should_call_sithswap ... ok (gas usage est.: 1126730)
test avnu_tests::exchange_test::MultiRouteSwap::should_fail_when_beneficiary_is_not_the_caller ... ok (gas usage est.:
↪  1435960)
test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_token_from_amount_is_0 ... ok (gas usage est.:
↪  1574760)
test avnu_tests::exchange_test::SetFeesBps0::should_set_fees_bps_0 ... ok (gas usage est.: 780160)
test avnu_tests::exchange_test::SwapExactTokenTo::should_throw_error_when_routes_is_empty ... ok (gas usage est.:
↪  1931800)
test avnu_tests::adapters::myswap_adapter_test::Swap::should_fail_when_invalid_additional_swap_params ... ok (gas usage
↪  est.: 853610)
test avnu_tests::exchange_test::GetFeesActive::should_return_a_bool ... ok (gas usage est.: 556380)
test avnu_tests::exchange_test::SetSwapExactTokenToFeesBps::should_fail_when_caller_is_not_the_owner ... ok (gas usage
↪  est.: 584540)
test avnu_tests::adapters::sithswap_adapter_test::Swap::should_fail_when_invalid_additional_swap_params ... ok (gas
↪  usage est.: 868970)
test avnu_tests::exchange_test::SwapExactTokenTo::should_fail_when_beneficiary_is_not_the_caller ... ok (gas usage
↪  est.: 1374160)
test avnu_tests::math::sqrt_ratio_test::ComputeSqrtRatioLimit::should_return_min_when_u256_sub_Overflow ... ok (gas
↪  usage est.: 10850)
test avnu_tests::math::sqrt_ratio_test::ComputeSqrtRatioLimit::should_return_max_when_u256_add_Overflow ... ok (gas
↪  usage est.: 10850)
test avnu_tests::exchange_test::TransferOwnership::should_change_owner ... ok (gas usage est.: 816040)
test avnu_tests::math::sqrt_ratio_test::ComputeSqrtRatioLimit::should_return_value_when_token0 ... ok (gas usage est.:
↪  10850)
test avnu_tests::math::sqrt_ratio_test::ComputeSqrtRatioLimit::should_return_value_when_token1 ... ok (gas usage est.:
↪  10850)
test avnu_tests::exchange_test::SetFeesBps0::should_fail_when_fees_are_too_high ... ok (gas usage est.: 678700)
test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_route_percent_is_higher_is_0 ... ok (gas usage
↪  est.: 2126630)
test avnu_tests::exchange_test::SetFeesActive::should_set_fees_active ... ok (gas usage est.: 777700)
test avnu_tests::adapters::tenkswap_adapter_test::Swap::should_call_tenkswap ... ok (gas usage est.: 1115380)
test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_caller_balance_is_too_low ... ok (gas usage
↪  est.: 1680930)
test avnu_tests::exchange_test::SwapExactTokenTo::should_throw_error_when_token_from_amount_is_0 ... ok (gas usage
↪  est.: 1512960)
test avnu_tests::exchange_test::TransferOwnership::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.:
↪  614820)
test avnu_tests::exchange_test::SetFeesBps0::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 584540)
test avnu_tests::exchange_test::SwapExactTokenTo::should_throw_error_when_first_token_from_is_not_token_from ... ok
↪  (gas usage est.: 2212480)
test avnu_tests::adapters::tenkswap_adapter_test::Swap::should_fail_when_invalid_additional_swap_params ... ok (gas
↪  usage est.: 872440)
test avnu_tests::exchange_test::SetFeesActive::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 584300)
test avnu_tests::exchange_test::GetFeesBps1::should_return_bps ... ok (gas usage est.: 558600)
test avnu_tests::exchange_test::TransferOwnership::should_fail_when_owner_is_0 ... ok (gas usage est.: 708980)

test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_routes_is_empty ... ok (gas usage est.: 1993600)
test avnu_tests::exchange_test::SwapExactTokenTo::should_throw_error_when_caller_balance_is_too_low ... ok (gas usage
↪  est.: 1619130)
test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_first_token_from_is_not_token_from ... ok (gas
↪  usage est.: 2274280)
test avnu_tests::exchange_test::GetFeesRecipient::should_return_recipient ... ok (gas usage est.: 558200)
test avnu_tests::exchange_test::MultiRouteSwap::should_call_swap ... ok (gas usage est.: 3737270)
test avnu_tests::exchange_test::MultiRouteSwap::should_call_swap_when_fees_and_multiple_routes ... ok (gas usage est.:
↪  5762300)
test avnu_tests::exchange_test::UpgradeClass::should_upgrade_class ... ok (gas usage est.: 685180)
test avnu_tests::exchange_test::SetFeesBps1::should_set_fees_bps_1 ... ok (gas usage est.: 780160)
test avnu_tests::exchange_test::SwapExactTokenTo::should_throw_error_when_last_token_to_is_not_token_to ... ok (gas
↪  usage est.: 2855100)
test avnu_tests::exchange_test::GetAdapterClassHash::should_return_adapter_class_hash ... ok (gas usage est.: 566450)
test avnu_tests::exchange_test::SetFeesRecipient::should_set_fees_recipient ... ok (gas usage est.: 779630)
```

```
test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_route_percent_is_higher_than_100 ... ok (gas
↪  usage est.: 2126630)
test avnu_tests::exchange_test::UpgradeClass::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 584010)
test avnu_tests::exchange_test::SetFeesBps1::should_fail_when_fees_are_too_high ... ok (gas usage est.: 678700)
test avnu_tests::exchange_test::SetFeesRecipient::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.:
↪  584410)
test avnu_tests::exchange_test::SwapExactTokenTo::should_swap_when_setting_token_to ... ok (gas usage est.: 6270980)
test avnu_tests::exchange_test::SetAdapterClassHash::should_set_adapter_class ... ok (gas usage est.: 796230)
test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_last_token_to_is_not_token_to ... ok (gas usage
↪  est.: 2916900)
test avnu_tests::exchange_test::SetFeesBps1::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 584540)
test avnu_tests::exchange_test::MultiRouteSwap::should_call_swap_when_fees ... ok (gas usage est.: 4892700)
test avnu_tests::exchange_test::SwapExactTokenTo::should_throw_error_fees_are_not_active ... ok (gas usage est.:
↪  3361600)
test avnu_tests::exchange_test::GetFeesBps0::should_return_bps ... ok (gas usage est.: 558600)
test avnu_tests::exchange_test::GetSwapExactTokenToFeesBps::should_return_bps ... ok (gas usage est.: 558600)
test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_residual_tokens ... ok (gas usage est.: 3463580)
test avnu_tests::exchange_test::MultiRouteSwap::should_fail_when_exchange_is_unknown ... ok (gas usage est.: 2237800)
test avnu_tests::exchange_test::SwapExactTokenTo::should_fail_when_not_enough_token_from ... ok (gas usage est.:
↪  3098470)
test avnu_tests::exchange_test::SetSwapExactTokenToFeesBps::should_set_swap_exact_token_to_fees_bps ... ok (gas usage
↪  est.: 780160)
test avnu_tests::exchange_test::MultiRouteSwap::should_fail_when_insufficient_tokens_received ... ok (gas usage est.:
↪  3030030)
test avnu_tests::exchange_test::SwapExactTokenTo::should_throw_error_fee_recipient_is_empty ... ok (gas usage est.:
↪  3713390)
test avnu_tests::exchange_test::MultiRouteSwap::should_throw_error_when_integrator_fees_are_too_high ... ok (gas usage
↪  est.: 3109690)
test avnu_tests::exchange_test::SwapExactTokenTo::should_throw_error_when_residual_tokens ... ok (gas usage est.:
↪  4146940)
test avnu_tests::exchange_test::MultiRouteSwap::should_call_swap_when_multiple_routes ... ok (gas usage est.: 9199260)
test avnu_tests::exchange_test::SwapExactTokenTo::should_swap_when_setting_token_to_when_no_fees ... ok (gas usage
↪  est.: 5856660)
test result: ok. 67 passed; 0 failed; 0 ignored; 0 filtered out;
```

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 process, 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.