Security Review Report
NM-0141 AVNU

NETHERMIND SECURITY

(Nov 2, 2023)

Contents

1 Executive Summary

This document outlines the security review conducted by Nethermind for the AVNU protocol. AVNU provides a DEX aggregator that scans and combines liquidity from various AMMs to find the best trading route, minimizing slippage and optimizing trade execution. By aggregating multiple liquidity sources, AVNU ensures that users have access to the most competitive prices available in the market. The protocol has integrations with well-recognized AMM's, such as ekubo, sithswap, jediswap, 10kswap, and Braavos mySwap.

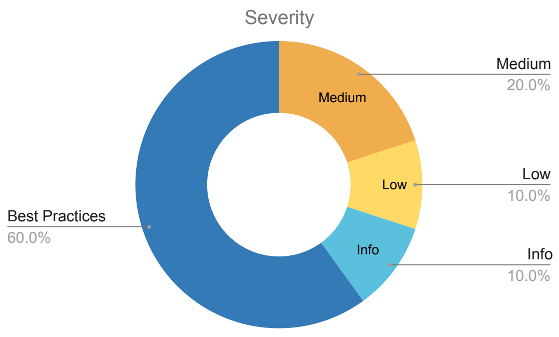
The audited code comprises 830 lines of Cairo code. During the audit, we observed that the code lacks validation for some variables within valid ranges. Although we have not currently identified any specific attack vectors associated with these missing validations, it is prudent to ensure that all variables consistently conform to valid ranges. This practice will significantly reduce the likelihood of encountering instances where invariants break.

Consequently, we recommend that the AVNU team undertake a comprehensive examination of the entire codebase to confirm that each function rigorously tests preconditions and postconditions. Moreover, it is of utmost importance to verify that each variable undergoes meticulous scrutiny to guarantee compliance with valid ranges. Specifically, percentages should not exceed 100%, fees should be subject to appropriate restrictions, addresses must not be null, and amounts should not be zero. Furthermore, routes should always contain valid data, and it is imperative to impose a maximum length on routes to prevent arbitrary sizes, which could potentially introduce vulnerabilities and opportunities for exploitation.

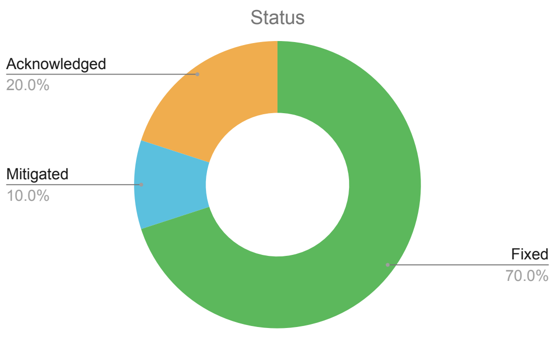
It is highly advisable to ensure that all variables remain within acceptable ranges across all AVNU contracts. In exploiting smart contracts, attackers seek to manipulate the system into an invalid state by compromising one or more invariants, leaving the system vulnerable. To mitigate the risk of such attacks, it is critical to maintain the system in a continuously valid state. This can be achieved through thorough validation of preconditions before executing functions and conducting comprehensive checks on post-conditions after execution. We strongly recommend the application of these principles to every function and variable within AVNU.

The audit was performed using: (a) manual analysis of the codebase and (b) simulation of the smart contracts. **Along this document, we report** 10 points of attention, where two are classified as `Medium`, one is classified as `Low`, and seven 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 assumptions for this audit. Section 5 presents the system overview. Section 6 discusses the risk rating methodology adopted for this audit. Section 7 details the issues. Section 8 discusses the documentation provided by the client for this audit. Section 9 presents the compilation, tests, and automated tests. Section 10 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 (0), **High** (0), **Medium** (2), **Low** (1), **Undetermined** (0), **Informational** (1), **Best Practices** (6). **(b) Distribution of status: Fixed** (7), **Acknowledged** (2), **Mitigated** (1), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	October 31, 2023
Final Report	November 2, 2023
Methods	Manual Review, Automated Analysis, Tests
Repository	AVNU Contracts V2
Commit Hash	6845406ad6a4134b7f3d64e8f578b4fd4c3e3560
Documentation	README.md
Documentation Assessment	Medium
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/interfaces.cairo	2	0	0.0%	0	2
2	src/exchange.cairo	327	38	11.6%	49	414
3	src/models.cairo	8	1	12.5%	1	10
4	src/adapters.cairo	19	1	5.3%	2	22
5	src/lib.cairo	6	1	16.7%	1	8
6	src/math.cairo	2	0	0.0%	0	2
7	src/math/sqrt_ratio.cairo	24	0	0.0%	0	24
8	src/math/muldiv.cairo	35	5	14.3%	8	48
9	src/adapters/ekubo_adapter.cairo	156	27	17.3%	23	206
10	src/adapters/myswapv2_adapter.cairo	57	11	19.3%	9	77
11	src/adapters/sithswap_adapter.cairo	54	7	13.0%	8	69
12	src/adapters/jediswap_adapter.cairo	42	6	14.3%	7	55
13	src/adapters/tenkswap_adapter.cairo	48	7	14.6%	8	63
14	src/adapters/myswap_adapter.cairo	37	4	10.8%	4	45
15	src/interfaces/locker.cairo	6	2	33.3%	1	9
16	src/interfaces/erc20.cairo	7	1	14.3%	1	9
	Total	**830**	**111**	**13.4%**	**122**	**1063**

3 Summary of Issues

	Finding	Severity	Update
1	Missing validation of token addresses in routes data	Medium	Fixed
2	Percentages in routes data may cause residual tokens	Medium	Fixed
3	Not using a two-step process for transferring ownership	Low	Mitigated
4	Unnecessary onchain deadline calculation	Info	Fixed
5	Lack of input validation in `multi_route_swap(...)`	Best Practices	Fixed
6	Missing boundaries for fees	Best Practices	Fixed
7	Missing check for ranges	Best Practices	Acknowledged
8	Missing validation for `Route.percentage`	Best Practices	Fixed
9	Unused parameter `token_to_amount`	Best Practices	Acknowledged
10	Unused variable `myswapv2_adapter.exact_input`	Best Practices	Fixed

4 Assumptions

During the course of our review for AVNU, we focused primarily on the smart contract code and operational logic specific to the aggregator itself. We did not review the smart contracts of the protocols that the aggregator integrates with. The following assumptions were made with regard to the integrated protocols:

Correct integration assumption

We assume the integration between AVNU and the integrated protocols is implemented correctly. This includes, but is not limited to, data retrieval, transaction forwarding, fee calculations, and any other interactions between the aggregator and the integrated protocols.

Reliability of integrated protocols

We assume the integrated protocols are secure, reliable, and function as intended. Any vulnerabilities or malfunctions within those protocols are beyond the scope of our audit and could have potential implications for the aggregator.

By clarifying these assumptions, we intend to provide a clear delineation of our assessment's scope and to ensure that stakeholders are aware of the boundaries of our review.

5 System Overview

AVNU allows for multi-route swaps using a set of well-established AMMs (Automated Market Makers). Each AMM can be plugged into AVNU using the abstraction called `adaptor`. The adaptor defines the communication interface between external AMMs and AVNU. The interface is reproduced below.

```
#[starknet::interface]
trait ISwapAdapter<TContractState> {
    fn swap(
        self: @TContractState,
        exchange_address: ContractAddress,
        token_from_address: ContractAddress,
        token_from_amount: u256,
        token_to_address: ContractAddress,
        token_to_min_amount: u256,
        to: ContractAddress,
        additional_swap_params: Array<felt252>,
    );
}
```

The core contract is exchange.cairo. The contract uses the following storage variables for fee data, owner address, and adapters' class hashes.

```
#[storage]
struct Storage {
    Ownable_owner: ContractAddress,
    AdapterClassHash: LegacyMap<ContractAddress, ClassHash>,
    fees_active: bool,
    fees_bps_0: u128,
    fees_bps_1: u128,
    fees_recipient: ContractAddress,
}
```

The contract emits events for `swaps` and `ownership transfers`.

```
#[event]
#[derive(starknet::Event, Drop, PartialEq)]
enum Event {
    Swap: Swap,
    OwnershipTransferred: OwnershipTransferred,
}
```

The struct `Swap` is used to emit events after the swap is completed.

```
#[derive(Drop, starknet::Event, PartialEq)]
struct Swap {
    taker_address: ContractAddress,
    sell_address: ContractAddress,
    sell_amount: u256,
    buy_address: ContractAddress,
    buy_amount: u256,
    beneficiary: ContractAddress,
}
```

Ownership transfers are recorded using the following structure.

```
#[derive(starknet::Event, Drop, PartialEq)]
struct OwnershipTransferred {
    previous_owner: ContractAddress,
    new_owner: ContractAddress,
}
```

The constructor sets the `owner` of the contract and the fee recipient.

```
#[constructor]
fn constructor(
    ref self: ContractState, owner: ContractAddress, fee_recipient: ContractAddress
) {
    // Set owner & fee collector address
    self._transfer_ownership(owner);
    self.fees_recipient.write(fee_recipient)
}
```

The contract also has a set of admin and operational functions defined in the trait `IExchange`.

```
trait IExchange<TContractState> {
    fn get_owner(self: @TContractState) -> ContractAddress;
    fn transfer_ownership(ref self: TContractState, new_owner: ContractAddress) -> bool;
    fn upgrade_class(ref self: TContractState, new_class_hash: ClassHash) -> bool;
    fn get_adapter_class_hash(
        self: @TContractState, exchange_address: ContractAddress
    ) -> ClassHash;
    fn set_adapter_class_hash(
        ref self: TContractState, exchange_address: ContractAddress, adapter_class_hash: ClassHash
    ) -> bool;
    fn get_fees_active(self: @TContractState) -> bool;
    fn set_fees_active(ref self: TContractState, active: bool) -> bool;
    fn get_fees_recipient(self: @TContractState) -> ContractAddress;
    fn set_fees_recipient(ref self: TContractState, recipient: ContractAddress) -> bool;
    fn get_fees_bps_0(self: @TContractState) -> u128;
    fn set_fees_bps_0(ref self: TContractState, bps: u128) -> bool;
    fn get_fees_bps_1(self: @TContractState) -> u128;
    fn set_fees_bps_1(ref self: TContractState, bps: u128) -> bool;
    fn multi_route_swap(
        ref self: TContractState,
        token_from_address: ContractAddress,
        token_from_amount: u256,
        token_to_address: ContractAddress,
        token_to_amount: u256,
        token_to_min_amount: u256,
        beneficiary: ContractAddress,
        integrator_fee_amount_bps: u128,
        integrator_fee_recipient: ContractAddress,
        routes: Array<Route>,
    ) -> bool;
}
```

The function `multi_route_swap(...)` implements the core use case. It receives the address and amount of the source token, the address and minimal amount required from the destination token. It can define another address to receive the destination tokens (i.e., the beneficiary). However, the current implementation requires the beneficiary to be the caller's address, as shown below.

```
// In the future, the beneficiary may not be the caller
// Check if beneficiary == caller_address
assert(beneficiary == caller_address, 'Beneficiary is not the caller');
```

The function `multi_route_swap` also defines fee parameters and the route for the trade. The route specifies the AMMs that will be used to perform the swap, including the percentage of tokens to be traded using each AMM.

6 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.

7 Issues

7.1 [Medium] Missing validation of token addresses in routes data

File(s): `exchange.cairo`

Description The `routes` array, within the `multi_route_swap(...)` function, holds the data required for token swaps across multiple pools using the `apply_routes(...)` function. However, it is important to note that there is no guarantee that the address of the first swapped token matches the `token_from_address` passed into the `before_swap(...)` function. Likewise, the address of the last token swapped does not necessarily align with the `token_to_address` used in the `after_swap(...)` function. This lack of alignment potentially introduces the possibility of disparate addresses, giving rise to the following scenarios:

- **Inaccurate Data in the `Swap` Event Emitted:** Consider a scenario where the `routes` consist of tokens `A` and `B` designated for swapping. A user invokes `multi_route_swap(...)` and provides the address of token `C` as the `token_from_address` (with `token_to_amount=0`). Simultaneously, the user correctly forwards the required amount of tokens `A` directly to the `Exchange` contract. This configuration allows for the accurate execution of swaps defined within the `routes`. However, it is noteworthy that the `Swap` event emitted in this process will contain incorrect data. Such discrepancies have the potential to impact monitoring activities and the user interface;

- **Reduced Fee for Complex Routing:** Consider a scenario where the `routes` define an intricate token swap from `A` to `B` involving multiple pools. A user initiates the `multi_route_swap(...)` function, providing the specified `routes` and designating token `C` as the `token_to_address`. In this sequence, the entire quantity of token `B` that was intended to be swapped remains within the `Exchange` contract, and no token `C` is transferred to the caller. Subsequently, in a separate call, the user invokes `multi_route_swap(...)` with an empty routing while specifying the token `B` address as the `token_to_address`. In this context, the current length of the `routes` array does not exceed one, leading to the interpretation that the routing is not complex. Consequently, a reduced fee, denoted as `fees_bps_0`, is collected;

Recommendation(s): It is advisable to incorporate checks within the system to verify that the initial token address involved in the first swap corresponds to the `token_from_address`. Likewise, it is essential to ensure that the token acquired through the final swap precisely matches the `token_to_address`. Additionally, the function `after_swap(...)` must check if all the tokens defined in the `route.token_from` and `route.token_to` are not left in the `Exchange` contract, except the `token_to_address`.

Status: Fixed.

Update from the client: Fixed in commit 01cbbd19278cdd956a799aea8fc7d62539225da3.

7.2 [Medium] Percentages in routes data may cause residual tokens

File(s): exchange.cairo

Description: The function apply_routes(...) has the capability to perform a partial swap of the originally deposited quantity, as indicated by the variable token_from_amount, to satisfy the condition of receiving a minimum number of tokens, as specified by the value of the variable token_to_min_amount. This could potentially allow the Exchange contract to retain some tokens in its possession, not executing the swap of all tokens deposited by the caller_address. The code with audit comments is reproduced below.

```
fn after_swap(...) {
    let token_to = IERC20Dispatcher { contract_address: token_to_address };
    let received_token_to = token_to.balanceOf(contract_address);
    let token_to_final_amount = self
        .collect_fees(
            token_to,
            received_token_to,
            integrator_fee_amount_bps,
            integrator_fee_recipient,
            route_len
        );

    assert(token_to_min_amount <= token_to_final_amount, 'Insufficient tokens received');

    ////////////////////////////////////////////////////////////////////
    // @audit Missing validation: Not checking if "route.token_from"
    // and "route.token_to" have been completely swapped.
    ////////////////////////////////////////////////////////////////////

    token_to.transfer(beneficiary, token_to_final_amount);
    ...
}
```

Recommendation(s): The function after_swap(...) must check if all the tokens defined in the route.token_from and route.token_to are not left in the Exchange contract, except the token_to_address.

Status: Fixed.

Update from the client: Fixed in commit 01cbbd19278cdd956a799aea8fc7d62539225da3.

7.3 [Low] Not using a two-step process for transferring ownership

File(s): exchange.cairo

Description: Currently, the ownership is directly transferred by calling transfer_ownership(...) by the protocol owner. However, this is error-prone, and accidentally transferring ownership to the incorrect address may result in loss of ownership over the protocol.

Recommendation(s): We strongly advise the implementation of a two-step ownership transfer mechanism that would facilitate the proposal of a new owner and the subsequent acceptance of ownership by the designated address.

Status: Mitigated.

Update from the client: The owner role will be held by a multi-sig account. Every action done from it will be carefully reviewed before being submitted.

7.4 [Info] Unnecessary on-chain deadline calculation

File(s): `sithswap_adapter.cairo`, `tenkswap_adapter.cairo`, `jediswap_adapter.cairo`

Description: In Automated Market Makers (AMMs), deadlines are commonly employed to prevent situations where a swap transaction lingers excessively in the mempool, potentially causing price fluctuations from when the transaction is dispatched. Nevertheless, on-chain computation of the deadline, which occurs when the transaction is executed, is superfluous. This is because the current block timestamp is utilized in the calculation, guaranteeing the perpetuity of the deadline's validity. The code with audit comments is reproduced below.

```
// Init deadline
let block_timestamp = get_block_timestamp();
// @audit No meaning when calculating deadline on-chain
let deadline = block_timestamp + 1000;
```

Recommendation(s): We recommend computing the deadline off-chain and passing it to each external swap function. Or if the deadline is intentionally missed, simply pass the current block timestamp.

Status: Fixed.

Update from the client: Fixed in commit 08673de6535366ea532b3e69be16f8e3edfc20da.

7.5 [Best Practices] Lack of input validation in `multi_route_swap(...)`

File(s): `exchange.cairo`

Description: The function `multi_route_swap(...)` lacks a verification for the condition where `token_from_amount` is greater than zero. It can save gas when the user inputs a wrong value.

```
fn multi_route_swap(...) -> bool {
    ...
    ///////////////////////////////////////////////////////////////////////////
    // @audit "token_from_amount" must be greater than zero
    ///////////////////////////////////////////////////////////////////////////
    ...
}
```

Recommendation(s): Revert the transaction if `token_from_amount` is zero.

```
fn multi_route_swap(...)
    ...
+   assert(token_from_amount > 0, 'Amount must be greater than zero');
    ...
}
```

Status: Fixed.

Update from the client: Fixed in commit 5bc4fafce484b8a58cbd8c6bbaafaac37fcc62d5.

7.6 [Best Practices] Missing boundaries for fees

File(s): `exchange.cairo`

Description: The protocol does not account for valid fee ranges within the functions `set_fees_bps_0(...)` and `set_fees_bps_1(...)`. Furthermore, there is an absence of restrictions on the ability to set fees by integrators.

Recommendation(s): We recommend defining valid ranges for fees.

Status: Fixed.

Update from the client: Fixed in commit 21eb34dc68e4745afce5ba47e7c8b0094239b7ac.

7.7 [Best Practices] Missing check for ranges

File(s): `src/*`

Description: The current codebase does not validate all variables for valid ranges. Even though there are no specific issues observed due to these missing validations, it is recommended to ensure all variables adhere to valid ranges, minimizing the risk of broken invariants.

Recommendation(s): Consider a thorough examination of the entire codebase to ensure each function checks proper preconditions and postconditions rigorously. Confirmation of each variable's compliance with valid ranges is crucial. Specifically, percentages should not exceed 100%, fees should not be unrestricted, addresses and amounts should not be zero, and routes should never be empty or allowed to have arbitrary sizes. **These principles should be applied to every function and variable within AVNU**.

Status: Acknowledged.

7.8 [Best Practices] Missing validation for `Route.percentage`

File(s): `exchange.cairo`

Description: The function `fn apply_routes(...)` lacks validation for ensuring that `route.percent.into()` is less than or equal to 100. While we currently do not observe a specific attack vector associated with this vulnerability, it is advisable to take measures to ensure that all variables adhere to valid ranges. This will help minimize the chances of encountering broken invariants. Below, you will find the function with audit comments included for reference.

```
fn apply_routes(
    ref self: ContractState, mut routes: Array<Route>, contract_address: ContractAddress
) {
    ...
    /////////////////////////////////////////////////////////////
    // @audit "route.percent" must be less or equal to 100
    // @audit "route.percent" must be greater than or equal to 0
    /////////////////////////////////////////////////////////////
    let (token_from_amount, overflows) = muldiv(
        token_from_balance, route.percent.into(), 100_u256, false
    );
    assert(overflows == false, 'Overflow: Invalid percent');
    ...
}
```

Recommendation(s): Ensure that `Route.percentage` is less than or equal to 100% and greater than 0%. Do not allow `Route.percentage` equals 0%.

Status: Fixed.

Update from the client: Fixed in commit c13a5acdfe440f1de4195665ac9c728c92d41527.

7.9 [Best Practices] Unused parameter `token_to_amount`

File(s): `exchange.cairo`

Description: The function `multi_route_swap(...)` does not use the input parameter `token_to_amount`. Instead, the logic relies on the `token_to_min_amount`.

Recommendation(s): Evaluate if the parameter `token_to_amount` is really required. It seems redundant with the parameter `token_to_min_amount`.

Status: Acknowledged.

7.10 [Best Practices] Unused variable `myswapv2_adapter.exact_input`

File(s): `myswapv2_adapter.cairo`

Description: The variable `exact_input` is set in the `myswapv2_adapter::swap(...)` but not used.

Recommendation(s): The AVNU team must decide if this variable should be used or removed from the code.

Status: Fixed.

Update from the client: Fixed in commit c16940f2775b81e17faab8027de4596dc1d800c1.

8 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 the AVNU documentation

AVNU has sufficient documentation on the website and inline comments. Moreover, the client provided additional detailed documentation for the audit team. During the audit process, the audit team remained in constant communication with the client.

9 Test Suite Evaluation

9.1 Compilation Output

```
$ scarb build
   Compiling avnu v1.0.0 (/NM-0141-AVNU/avnu-contracts-v2-private/Scarb.toml)
    Finished release target(s) in 2 seconds
```

9.2 Tests Output

```
$ scarb test
     Running cairo-test avnu
testing avnu ...
running 41 tests
test avnu::tests::adapters::myswap_adapter_test::Swap::should_fail_when_invalid_additional_swap_params ... ok (gas
↪  usage est.: 807070)
test avnu::tests::adapters::sithswap_adapter_test::Swap::should_fail_when_invalid_additional_swap_params ... ok (gas
↪  usage est.: 828560)
test avnu::tests::adapters::jediswap_adapter_test::Swap::should_fail_when_invalid_additional_swap_params ... ok (gas
↪  usage est.: 834330)
test avnu::tests::adapters::tenkswap_adapter_test::Swap::should_fail_when_invalid_additional_swap_params ... ok (gas
↪  usage est.: 834330)
test avnu::tests::exchange_test::GetFeesRecipient::should_return_recipient ... ok (gas usage est.: 687860)
test avnu::tests::exchange_test::TransferOwnership::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.:
↪  776680)
test avnu::tests::exchange_test::GetFeesActive::should_return_a_bool ... ok (gas usage est.: 684030)
test avnu::tests::exchange_test::SetAdapterClassHash::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.:
↪  737310)
test avnu::tests::exchange_test::SetFeesActive::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 723050)
test avnu::tests::exchange_test::SetFeesBps1::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 724160)
test avnu::tests::exchange_test::GetFeesBps0::should_return_bps ... ok (gas usage est.: 688150)
test avnu::tests::exchange_test::SetFeesBps0::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 724160)
test avnu::tests::exchange_test::SetFeesRecipient::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.:
↪  724060)
test avnu::tests::exchange_test::UpgradeClass::should_upgrade_class ... ok (gas usage est.: 833630)
test avnu::tests::adapters::myswap_adapter_test::Swap::should_call_myswap ... ok (gas usage est.: 992670)
test avnu::tests::adapters::jediswap_adapter_test::Swap::should_call_jediswap ... ok (gas usage est.: 1089170)
test avnu::tests::exchange_test::SetFeesBps1::should_set_fees_bps_1 ... ok (gas usage est.: 942890)
test avnu::tests::exchange_test::SetAdapterClassHash::should_set_adapter_class ... ok (gas usage est.: 968800)
test avnu::tests::math::sqrt_ratio_test::ComputeSqrtRatioLimit::should_return_max_when_u256_add_Overflow ... ok (gas
↪  usage est.: 19480)
test avnu::tests::exchange_test::SetFeesActive::should_set_fees_active ... ok (gas usage est.: 937660)
test avnu::tests::math::sqrt_ratio_test::ComputeSqrtRatioLimit::should_return_value_when_token0 ... ok (gas usage est.:
↪  19480)
test avnu::tests::math::sqrt_ratio_test::ComputeSqrtRatioLimit::should_return_min_when_u256_sub_Overflow ... ok (gas
↪  usage est.: 19480)
test avnu::tests::exchange_test::GetOwner::should_return_owner ... ok (gas usage est.: 687860)
test avnu::tests::exchange_test::MultiRouteSwap::should_throw_error_when_caller_balance_is_too_low ... ok (gas usage
↪  est.: 1791820)
test avnu::tests::exchange_test::GetAdapterClassHash::should_return_adapter_class_hash ... ok (gas usage est.: 700910)
test avnu::tests::exchange_test::UpgradeClass::should_fail_when_caller_is_not_the_owner ... ok (gas usage est.: 721560)
test avnu::tests::exchange_test::GetFeesBps1::should_return_bps ... ok (gas usage est.: 688150)
test avnu::tests::math::sqrt_ratio_test::ComputeSqrtRatioLimit::should_return_value_when_token1 ... ok (gas usage est.:
↪  19480)
test avnu::tests::exchange_test::TransferOwnership::should_fail_when_owner_is_0 ... ok (gas usage est.: 882850)
test avnu::tests::adapters::tenkswap_adapter_test::Swap::should_call_tenkswap ... ok (gas usage est.: 1089170)
test avnu::tests::adapters::sithswap_adapter_test::Swap::should_call_sithswap ... ok (gas usage est.: 1115420)
test avnu::tests::exchange_test::SetFeesRecipient::should_set_fees_recipient ... ok (gas usage est.: 942500)
test avnu::tests::exchange_test::SetFeesBps0::should_set_fees_bps_0 ... ok (gas usage est.: 942890)
test avnu::tests::exchange_test::TransferOwnership::should_change_owner ... ok (gas usage est.: 1001120)
test avnu::tests::exchange_test::MultiRouteSwap::should_fail_when_beneficiary_is_not_the_caller ... ok (gas usage est.:
↪  1270310)
test avnu::tests::exchange_test::MultiRouteSwap::should_call_swap ... ok (gas usage est.: 2863230)
test avnu::tests::exchange_test::MultiRouteSwap::should_fail_when_exchange_is_unknown ... ok (gas usage est.: 2153350)
test avnu::tests::exchange_test::MultiRouteSwap::should_fail_when_insufficient_tokens_received ... ok (gas usage est.:
↪  2559130)
test avnu::tests::exchange_test::MultiRouteSwap::should_call_swap_when_fees ... ok (gas usage est.: 4420030)
test avnu::tests::exchange_test::MultiRouteSwap::should_call_swap_when_fees_and_multiple_routes ... ok (gas usage est.:
↪  4871670)
test avnu::tests::exchange_test::MultiRouteSwap::should_call_swap_when_multiple_routes ... ok (gas usage est.: 5731090)
test result: ok. 41 passed; 0 failed; 0 ignored; 0 filtered out;
```

10 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.

Disclaimer