Security Review Report
NM-0067 DECENTRALAND PERIODIC TOKEN VESTING

NETHERMIND

(Dec 5, 2022)

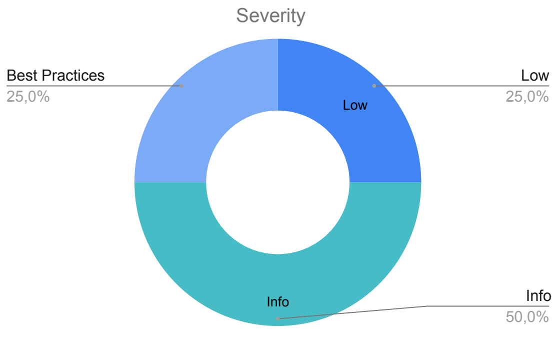
Contents

1 Executive Summary

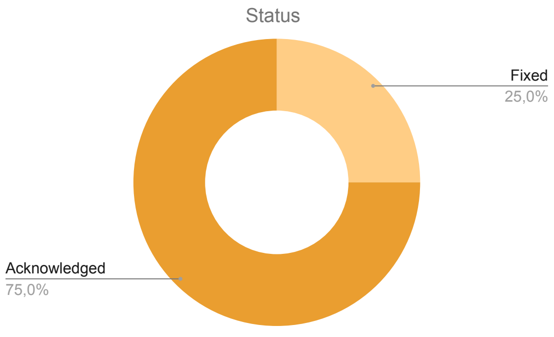
This document presents the security review of Decentraland periodic token vesting implementation performed by Nethermind. The vesting generator allows vesting a token in varying quantities over consecutive periods of the same length. Common vestings would obtain vested tokens linearly over a certain period, occasionally with a cliff to restrict these tokens from being released for a fraction of that time. Decentraland vesting builder allows one to set periods, define how many tokens each period vests, and construct a program in which a variable number of tokens become releasable as each period proceeds.

The contract under security review implements 80 unit tests. The source code consists of 258 lines of Solidity with code coverage of 100%. The audit was completed primarily with manual analysis of the codebase, as well as using automated analysis tools. **The application has good technical documentation.** The documentation is easy to follow and sufficient for this audit purpose. It includes a high level overview of the application, how to test it, use case scenarios, and access to the application (although the link for the test application is not working to create a token vesting at the time of this report). **Overall the code is well-written, structured, commented, and easy to comprehend.** Inputs and variables are validated to ensure the data are correct which is ideal for security.

During the initial audit, we point out 5 potential issues. During the reaudit, the Decentraland team has acknowledged 3 issues, fixed 1 issue and 1 issue was defined as false-positive. These numbers are summarized in the figure below. **This document is organized as follows.** Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 discusses the risk rating methodology adopted for this audit. Section 5 details the issues. Section 6 discusses the documentation provided for this audit. Section 7 presents the compilation, tests, coverage and automated tests. Section 8 concludes the document.



(a)



(b)

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

Summary of the Audit

Audit Type	Security Review
Initial Report	Nov. 15, 2022
Response from Client	Nov. 28, 2022
Final Report	Dec. 5, 2022
Methods	Manual Review, Automated Analysis
Repository	Periodic Token Vesting Contract
Commit Hash (Initial Audit)	0950e1eab55c707fea4f4999699e4429cd65ef1c
Documentation	README.md
Documentation Assessment	High
Test Suite Assessment	High

2 Contracts

	Contract	Lines of Code	Lines of Comments	Comments Ratio	Blank Lines	Total Lines
1	./contracts/PeriodicTokenVesting.sol	258	108	41.9%	85	451
	Total	**258**	**108**	**41.9%**	**85**	**451**

3 Summary of Issues

	Finding	Severity	Update
1	Changing beneficiary is an atomic process	Low	Acknowledged
2	Total vested amount can be calculated in `initialize(...)`	Info	Acknowledged
3	Total vested amount not guaranteed to be released to beneficiary	Info	Acknowledged
4	Gas optimization for function `getTotal()`	Best Practices	Fixed

4 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 is a measure of 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 the finding were to be exploited by an attacker. 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, 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.

5 Issues

5.1 PeriodicTokenVesting.sol

5.1.1 [Low] Changing beneficiary is an atomic process

File(s): `contracts/PeriodicTokenVesting.sol`

Description: Changing the beneficiary with the function `setBenificiary(...)` is an atomic process. Once changed only the new beneficiary can make any further changes. If the current beneficiary passes an incorrect non-zero value then access to the contract will be lost.

Recommendation(s): Consider changing the process of setting a new beneficiary to set-then-claim to ensure that the beneficiary has set the correct address.

Status: Acknowledged

Update from the client: It is a great suggestion but we will not change it for the moment as we think that if we apply this pattern to `setBeneficiary`, it would be appropriate to change it on all setters and we can't do it for inherited OZ contracts.

5.1.2 [Info] Total vested amount can be calculated in `initialize(...)`

File(s): `contracts/PeriodicTokenVesting.sol`

Description: The function `getTotal(...)` is used to calculate the total amount to be vested over all time periods. Since the contents of `vestedPerPeriod` cannot change, the total can be calculated once in the `initialize(...)` function to save gas.

Recommendation(s): Consider calculating the total amount to be vested in the `initialize(...)` function rather than calculating the total on every call to `getTotal(...)`.

Status: Acknowledged

Update from the client: We won't apply as it considerably increases the gas consumption of the initialize function.

5.1.3 [Info] Total vested amount not guaranteed to be released to beneficiary

File(s): `contracts/PeriodicTokenVesting.sol`

Description: The vesting contract is used to gradually release tokens based on some vesting schedule. However, it is not guaranteed that the owner has transferred enough funds to the contract to cover the entire vested amount. The beneficiary may be unable to release vested funds due to insufficient contract token balances.

Recommendation(s): Consider checking in the `initialize(...)` function that the contract's balance of vesting tokens is greater than or equal to the total amount to be vested over all periods. This will ensure that the contract has enough tokens for the beneficiary.

Status: Acknowledged

Update from the client: This is done by design. The one in charge of funding the contract might not be the deployer so it has to be funded manually after the deployment.

5.1.4 [Best Practices] Gas optimization for function `getTotal()`

File(s): `contracts/PeriodicTokenVesting.sol`

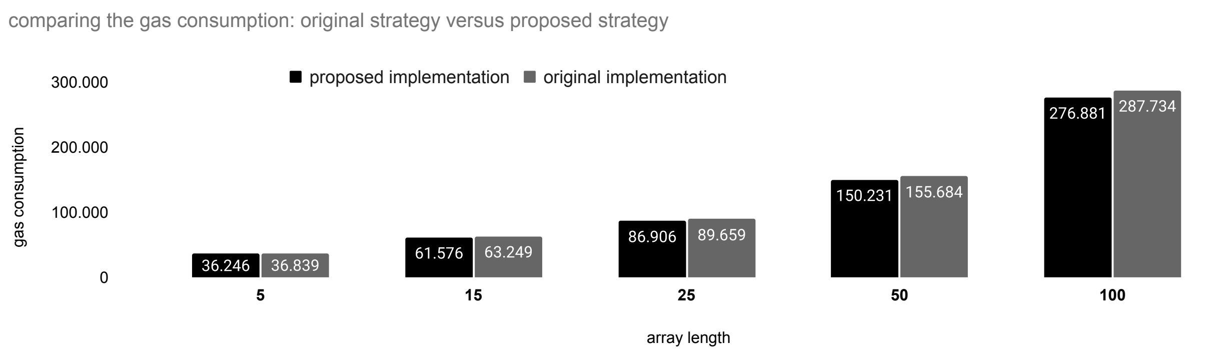
Description: The function `getTotal()` can be optimized by storing the length of the array `vestedPerPeriod` into a temporary variable using the same strategy adopted by the developers in function `getVested()`. The original code is shown below.

```
function getTotal() public view returns (uint256) {
   uint256 total;

   //////////////////////////////////////////////////////////////////////
   // @audit store "vestedPerPeriod.length" into a temp var
   //////////////////////////////////////////////////////////////////////
   for (uint i = 0; i < vestedPerPeriod.length; ) {
       total += vestedPerPeriod[i];
       unchecked {
           ++i;
       }
   }
   return total;
}
```

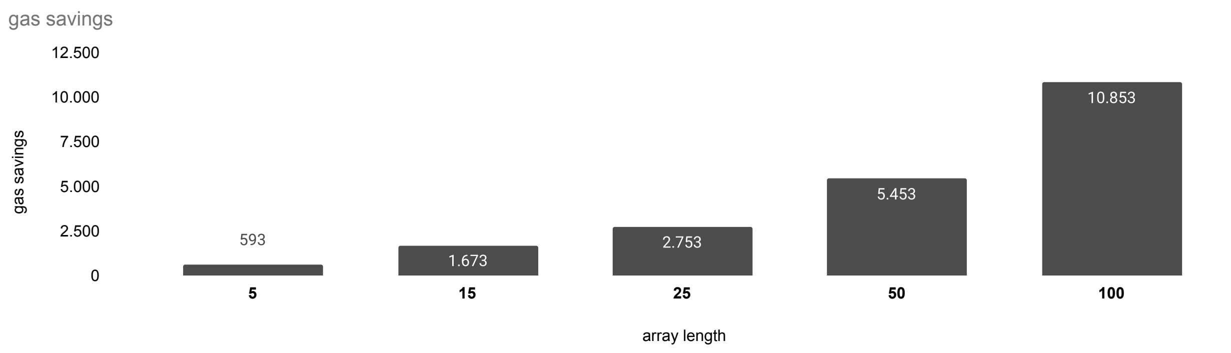
The proposed code is presented below.

```
function getTotal_new_implementation() public view returns (uint256)
{
   uint256 total;

   ///////////////////////////////////////////////////////////////////
   // @audit store "vestedPerPeriod.length" into a temp var
   ///////////////////////////////////////////////////////////////////
   uint256 size = vestedPerPeriod.length;
   for (uint i = 0; i < size; ) {
       total += vestedPerPeriod[i];
       unchecked {
           ++i;
       }
   }
   return total;
}
```

The gas consumption for both strategies is compared in the figure below. The `x-axis` shows the array length, while the `y-axis` shows the gas consumption.



The gas consumption reduction promoted by the new strategy is presented in the next figure.



Recommendation(s): Consider using the proposed strategy.

Status: Fixed

Update from the client: We will apply the following optimization in the `getTotal(...)` function as it saves even more gas due to not having to access storage `vestedPerPeriod[i]` each iteration.

```
/// @notice Get the total amount of tokens that will be vested in this contract.
/// @return The total amount of tokens that will be vested in this contract.
function getTotal() public view returns (uint256) {
   uint256 total;
   uint256[] memory mVestedPerPeriod = vestedPerPeriod;

   // Sum all the tokens vested per period to obtain the total amount.
   for (uint256 i = 0; i < mVestedPerPeriod.length; ) {
       total += mVestedPerPeriod[i];
       unchecked {
           ++i;
       }
   }
   return total;
}
```

6 Documentation Evaluation

Technical documentation is created to explain what the software product does. This way, developers and stakeholders can easily follow the purpose and the underlying functionality of each file/function/line. Documentation can come not only in the form of a README.md but also using **code as documentation** (to write clear code), **unit tests** (provide documentation and facilitate good design), diagrams, websites, research papers, videos and external documentation. Specifically, code as documentation is a software engineering practice where the code is written in such a way that it serves as its own explanatory documentation. It can be achieved by using meaningful names for variables, functions and methods following an intuitive design that can be easily understood by readers.

The codebase contains a README file with sufficient information for the purposes of this audit. It includes a high level overview of the application, use case scenarios, and a description of how to run unit tests. There is also a link to use the application to create vesting (although, we could not create a vesting through the provided link). Adequate technical documentation improves the efficiency of audits in addition to being a good programming practice. Consequently, less time is required to comprehend the contract under review and more time can be addressed to audit activities, thus, improving the efficiency and overall audit output. And for this end, the documentation is easy to follow and clarifies the most important features of the protocol.

Concerning code as documentation, we observed that all functions are properly documented by implementing NatSpec in-line documentation structure to make it easy to understand what a function should do as well as its inputs and outputs.

7 Test Suite Evaluation

Decentraland presented 80 unit tests addressed to functions implemented in the contract `PeriodicTokenVesting`, with the code coverage of 100%.

7.1 Contracts Compilation Output

Below, we present the compilation process output. As we can see, there are no warnings and all Solidity files were successfully compiled.

```
$ npx hardhat compile
Compiled 16 Solidity files successfully
```

7.2 Test Output

```
$ npx hardhat test
Compiled 16 Solidity files successfully

  PeriodicTokenVesting
    initialize
       should have uninitialized values before initialization
       should initialize values
       should support 1250 periods with 1 million to vest each
       reverts when initializing twice
       reverts when initializing the implementation
       reverts when owner is 0x0
       reverts when beneficiary is 0x0
       reverts when token is 0x0
       reverts when period duration is 0
       reverts when vested per period length is 0
    getTotal
       should return the total amount of tokens to be vested by the contract
    setBeneficiary
       should change the beneficiary
       should emit a BeneficiaryUpdated event
       reverts when sender is not the current beneficiary
       reverts when beneficiary is 0x0
    getVested
       should return 0 if the vesting has not started
       should return 0 if the cliff has not passed
       should return 0 if the vesting was revoked before start
       should return 0 if the vesting was revoked before cliff
       should return 0 if the vesting was paused before start
       should return 0 if the vesting was paused before cliff
       should return total if all periods have passed
       should return total if all periods + 1 have passed
       should return total if all periods + 1 have passed and the vesting is linear
       should return 0 when half of the first period has elapsed
       should return half the vested tokens of the first period when half the first period has elapsed and the vesting
      ↪   is linear
       should return 1/4 the vested tokens of the first period when 1/4 of the first period has elapsed and the vesting
      ↪   is linear
       should return 3/4 the vested tokens of the first period when 3/4 of the first period has elapsed and the vesting
      ↪   is linear
      should return all the vested tokens of the first period when the first period has elapsed and the vesting is linear
       should return the tokens vested by the first period when a period elapses and the cliff lasts 1 period
       should return the tokens vested by both the first and second period when 2 periods elapse and the cliff lasts 2
      ↪   periods
       should return correctly when the vesting is linear and the cliff is shorter than a period
       should return correctly when the vesting is linear and the cliff is longer than a period
    release
       should be able to release all tokens once all periods elapse
       should be able to release tokens vested up to the revoke
       should be able to release tokens vested up to the paused timestamp and release all when unpaused and all periods
    ↪  have elapsed
       should emit a Released event
       should be able to release all tokens when all periods + 1 have elapsed
       should update the released variable with the amount released
       should be able to release all when the vesting ends and there were 1250 periods with 1 million to vest each
       reverts when amount is 0
       reverts when amount is greater than what is releasable
       reverts when caller is not the beneficiary
       reverts when the contract does not have funds
       reverts when the receiver is 0x0
    revoke
       should update the stop timestamp variable
       should update the isRevoked variable
       should emit a Revoked event
       reverts when caller is not the owner
       reverts when contract is not revocable
       reverts when contract has been already revoked
    releaseForeignToken
       should transfer a determined amount of foreign tokens to the receiver
       should emit a ReleasedForeign event
       reverts when the receiver is 0x0
       reverts when the caller is not the owner
       reverts when amount is 0
       reverts when contract balance is lower than amount
       reverts when token is the same as the one vested
    releaseSurplus
       should release an amount of surplus tokens to the receiver
       should emit a ReleasedSurplus event
       should be able to release tokens not vested after revoke as surplus
       should be able to release only surplus tokens when paused
       reverts when amount is 0
       reverts when receiver is 0x0
       reverts when amount is higher than the surplus there is no surplus
       reverts when the caller is not the owner
       reverts when the contract balance is lower or equal than the non surplus
    pause
       should pause the vesting
       should emit a Paused event
       should update the stop timestamp
       reverts when the contract is already paused
       reverts when the caller is not the owner
       reverts when the vesting is revoked
       reverts when the vesting is non pausable
    unpause
       should unpause the vesting
       should emit an Unpaused event
       should update the stop timestamp
       reverts when the contract is not paused
       reverts when the caller is not the owner
       reverts when the vesting is revoked
```

```
·-----------------------------------------------|--------------------------|-------------|--------------------------·
|            Solc version: 0.8.17               ·  Optimizer enabled: true  ·  Runs: 200  ·  Block limit: 30000000 gas │
························································|··························|·············|··························
|  Methods
························|······················|············|·············|·············|···············|·············
|  Contract             ·  Method              ·  Min        ·  Max        ·  Avg        ·  # calls      ·  usd (avg)
························|······················|············|·············|·············|···············|·············
|  MockToken            ·  transfer            ·      51611  ·      51635  ·      51627  ·           16  ·          -
························|······················|············|·············|·············|···············|·············
|  PeriodicTokenVesting ·  initialize          ·     398886  ·   28185552  ·    1011354  ·           91  ·          -
························|······················|············|·············|·············|···············|·············
|  PeriodicTokenVesting ·  pause               ·          -  ·          -  ·      76097  ·           12  ·          -
························|······················|············|·············|·············|···············|·············
|  PeriodicTokenVesting ·  release             ·      82990  ·    3106783  ·     445048  ·            9  ·          -
························|······················|············|·············|·············|···············|·············
|  PeriodicTokenVesting ·  releaseForeignToken ·      62906  ·      67706  ·      64506  ·            3  ·          -
························|······················|············|·············|·············|···············|·············
|  PeriodicTokenVesting ·  releaseSurplus      ·      91234  ·      95066  ·      94300  ·            5  ·          -
························|······················|············|·············|·············|···············|·············
|  PeriodicTokenVesting ·  revoke              ·          -  ·          -  ·      56510  ·           11  ·          -
························|······················|············|·············|·············|···············|·············
|  PeriodicTokenVesting ·  setBeneficiary      ·          -  ·          -  ·      33130  ·            3  ·          -
························|······················|············|·············|·············|···············|·············
|  PeriodicTokenVesting ·  unpause             ·          -  ·          -  ·      33388  ·            6  ·          -
························|······················|············|·············|·············|···············|·············
|  Deployments                                  ·                                          ·  % of limit   ·
················································|············|·············|·············|···············|·············
|  MockProxy                                    ·     114832  ·     114856  ·     114855  ·        0.4 %  ·          -
················································|············|·············|·············|···············|·············
|  MockToken                                    ·     627560  ·     627596  ·     627593  ·        2.1 %  ·          -
················································|············|·············|·············|···············|·············
|  PeriodicTokenVesting                         ·          -  ·          -  ·    1653540  ·        5.5 %  ·          -
·-----------------------------------------------|------------|-------------|-------------|---------------|------------·

  80 passing (15s)
```

7.3 Code Coverage

The tests present a full code coverage (100%), i.e., all code lines were tested.

```
---------------------------|----------|----------|----------|----------|----------------|
File                       |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
---------------------------|----------|----------|----------|----------|----------------|
 contracts/                |      100 |      100 |      100 |      100 |                |
  PeriodicTokenVesting.sol |      100 |      100 |      100 |      100 |                |
---------------------------|----------|----------|----------|----------|----------------|
All files                  |      100 |      100 |      100 |      100 |                |
---------------------------|----------|----------|----------|----------|----------------|
```

7.4 Echidna

The file `config.yaml` was defined as:

```
testMode: assertion
```

The following command was used for running `Echidna`.

```
echidna-test test/AssertVesting.sol --contract AssertVesting --test-mode assertion
```

```
Echidna 2.0.3
Tests found: 28
Seed: -3550879629966329213                                                           Unique
↪  instructions: 2023
Unique codehashes: 1
Corpus size: 15

---Tests---
AssertionFailed(..): PASSED!
assertion in AssertTokenPeriodVested(): PASSED!
assertion in releaseSurplus(address,uint256): PASSED!
assertion in getReleased(): PASSED!
assertion in transferOwnership(address): PASSED!
assertion in getStop(): PASSED!
assertion in getIsPausable(): PASSED!
assertion in getStart(): PASSED!
assertion in releaseForeignToken(address,address,uint256): PASSED!
assertion in revoke(): PASSED!
assertion in getIsLinear(): PASSED!
assertion in owner(): PASSED!
assertion in pause(): PASSED!
assertion in getVestedPerPeriod(): PASSED!
assertion in getTotal(): PASSED!
assertion in renounceOwnership(): PASSED!
assertion in getCliff(): PASSED!
assertion in paused(): PASSED!

---Echidna 2.0.3---
Tests found: 28
Seed: -3550879629966329213
Unique instructions: 2023
Unique codehashes: 1
Corpus size: 15

---Tests---
AssertionFailed(..): PASSED!
assertion in AssertTokenPeriodVested(): PASSED!
assertion in releaseSurplus(address,uint256): PASSED!
assertion in getReleased(): PASSED!
assertion in transferOwnership(address): PASSED!
assertion in getStop(): PASSED!
assertion in getIsPausable(): PASSED!
assertion in getStart(): PASSED!
assertion in releaseForeignToken(address,address,uint256): PASSED!
assertion in revoke(): PASSED!
assertion in getIsLinear(): PASSED!
assertion in owner(): PASSED!
assertion in pause(): PASSED!
assertion in getVestedPerPeriod(): PASSED!
assertion in getTotal(): PASSED!
assertion in renounceOwnership(): PASSED!
assertion in getCliff(): PASSED!
assertion in paused(): PASSED!
assertion in getBeneficiary(): PASSED!
assertion in unpause(): PASSED!
assertion in getToken(): PASSED!
assertion in getReleasable(): PASSED!
assertion in getPeriod(): PASSED!
assertion in setBeneficiary(address): PASSED!
assertion in getIsRevocable(): PASSED!
assertion in getVested(): PASSED!
assertion in getIsRevoked(): PASSED!
assertion in release(address,uint256): PASSED!
Campaign complete, C-c or esc to exit
```

8 About Nethermind

Founded in 2017 by a small team of world-class technologists, Nethermind builds Ethereum solutions for developers and enterprises. Boosted by a grant from the Ethereum Foundation in August 2018, our team has worked tirelessly to deliver the fastest Ethereum client in the market. Our flagship Ethereum client is all about performance and flexibility. Built on .NET core, a widespread, enterprise-friendly platform, Nethermind makes integration with existing infrastructures simple, without losing sight of stability, reliability, data integrity, and security. **Nethermind is made up of several engineering teams across various disciplines, all collaborating to realize the Ethereum roadmap, by conducting research and building high-quality tools**. Teams focus on specific areas of the Ethereum problem space. Each consists of specialists and experienced developers working alongside interns, learning the ropes in the Nethermind Internship Program. **Our mission is to gather passionate talent from around the world, and to tackle some of the blockchain's most complex problems**. Nethermind provides software solutions and services for developers and enterprises building the Ethereum ecosystem. We offer security reviews to projects built on EVM compatible chains and StarkNet. We have expertise in multiple areas of the Ethereum ecosystem, including protocol design, smart contracts (written in Solidity and Cairo), MEV, etc. We develop some of the most used tools on Starknet and one of the most used Ethereum clients. Learn more about us at `https://nethermind.io`.