Security Review Report
NM-0645 - Origin

NETHERMIND SECURITY

(October 16, 2025)

Contents

1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for Origin Protocol's Compounding Staking Strategy. **Origin Protocol's Compounding Staking Strategy** is a system for natively staking ETH on the Beacon Chain, designed to operate efficiently within the post-Pectra upgrade environment by minimizing the number of validators required and thus reducing operational overhead.

The strategy leverages the SSV Network for distributed validator operations and ensures the integrity of its accounting through on-chain cryptographic verification of Beacon Chain state using Merkle proofs, forgoing reliance on external data feeds. A privileged `Registrator` role is responsible for managing the validator lifecycle, which incorporates a secure, two-phase staking process to mitigate deposit front-running attacks. This lifecycle includes the initial deposit, on-chain credential verification, balance top-ups to leverage the increased maximum effective balance, and eventual validator exit and removal from the network.

The audit comprises 1313 lines of Solidity code.

The audit was performed using (a) manual analysis of the codebase, and (b) automated analysis tools.

Along this document, we report a single point of attention classified as `Informational` severity. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 and Section 7 provide details of the issues from intial and the new version. Section 8 discusses the documentation provided by the client for this audit. Section 9 presents the test suite evaluation and automated tools used. Section 10 concludes the document.

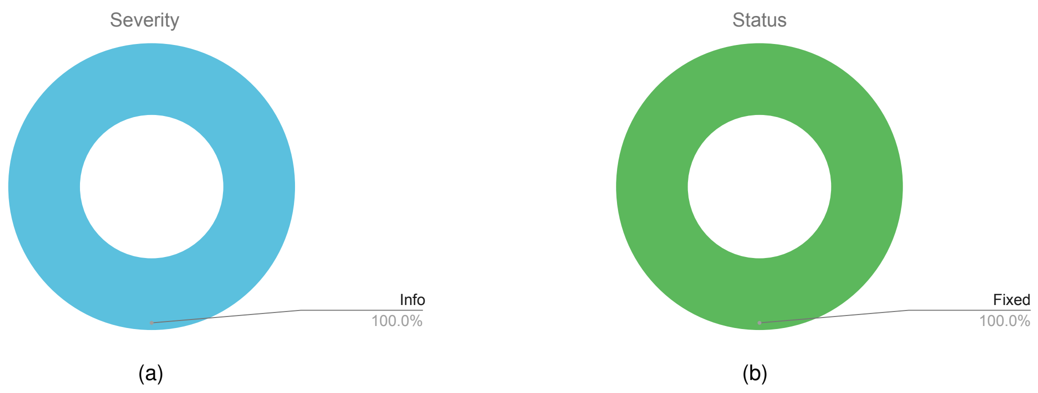

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (1), **Best Practices** (0).
Distribution of status: Fixed (1), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	October 10, 2025
Final Report	October 16, 2025
Initial Commit	0528675dcc0a2a6b773baf3d310b062a804bda7d
Final Commit	3e790c1dd9b59d0cb02570d2110e5eba004eeeff
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	contracts/beacon/Merkle.sol	85	53	62.4%	5	143
2	contracts/beacon/PartialWithdrawal.sol	22	16	72.7%	7	45
3	contracts/beacon/BeaconProofs.sol	115	62	53.9%	10	187
4	contracts/beacon/Endian.sol	35	18	51.4%	3	56
5	contracts/beacon/BeaconRoots.sol	16	15	93.8%	3	34
6	contracts/beacon/BeaconProofsLib.sol	250	143	57.2%	37	430
7	contracts/strategies/NativeStaking/CompoundingStakingView.sol	61	13	21.3%	8	82
8	contracts/strategies/NativeStaking/CompoundingStakingSSVStrategy.sol	120	71	59.2%	32	223
9	contracts/strategies/NativeStaking/CompoundingValidatorManager.sol	609	540	88.7%	131	1280
	Total	**1313**	**931**	**70.9%**	**236**	**2480**

3 Summary of Issues

	Finding	Severity	Update
1	Off-by-one error prevents reaching `MAX_VERIFIED_VALIDATORS` limit	Info	Fixed

4 System Overview

The Origin Protocol's Compounding Staking Strategy is a system for natively staking ETH on the Beacon Chain, designed to operate efficiently within the post-Pectra upgrade environment. By leveraging the increased maximum effective balance for validators, the strategy aims to minimize the number of validators required, thereby reducing operational overhead. The system utilizes the SSV Network for distributed validator operations and relies on on-chain verification of Beacon Chain state through Merkle proofs to ensure the integrity of its accounting.

4.1 Fund Deposit and Withdrawal

The primary entry point for capital is the `CompoundingStakingSSVStrategy` contract, which receives Wrapped ETH (WETH) from the main Origin Protocol Vault. Unlike strategies that immediately deploy assets, this contract holds the deposited WETH until it is actively used for staking.

- **Deposits**: When the Vault calls the `deposit` or `depositAll` functions, the strategy contract receives WETH. It updates an internal accounting variable, `depositedWethAccountedFor`, to track the amount of WETH that is available for staking but has not yet been deployed to a validator.
- **Withdrawals**: Users initiate withdrawals through the Vault, which in turn calls the `withdraw` or `withdrawAll` function on the strategy. The strategy fulfills the request by transferring its WETH balance. Any native ETH held by the contract, accumulated from validator exits or execution layer rewards, is first converted to WETH to satisfy the withdrawal demand.

4.2 Validator Lifecycle Management

The management of validators from creation to retirement is handled by a privileged account known as the `Registrator`. This role is responsible for executing the key state transitions in a validator's lifecycle.

- **Registration**: The lifecycle begins when the `Registrator` calls `registerSsvValidator`, providing a validator's public key and the operator details for the SSV Network. This action transitions the validator's state from `NON_REGISTERED` to `REGISTERED`.
- **Initial Staking and Security**: To protect against deposit front-running attacks, the protocol employs a two-phase staking process. The `Registrator` first stakes a minimal 1 ETH deposit, moving the validator to the `STAKED` state. Crucially, before any further capital is committed, the validator's withdrawal credentials must be proven correct via a successful on-chain call to the `verifyValidator` function. This verification ensures that funds are staked to a validator controlled by the strategy. To further minimize risk, the system only permits one unverified validator to exist at any given time, serializing the onboarding process.
- **Verification and Top-Up**: Once a validator's credentials are confirmed, its state becomes `VERIFIED`. The `Registrator` can then call `stakeEth` to deposit additional funds. This can be done to meet the 32 ETH activation threshold or to top up the validator's balance further, taking advantage of the increased maximum effective balance post-Pectra. The validator's state is updated to `ACTIVE` once its balance is confirmed to be above the activation threshold during a subsequent balance verification process.
- **Withdrawal, Exit, and Removal**: The `Registrator` can initiate withdrawals by calling `validatorWithdrawal`. Providing a specific amount triggers a partial withdrawal of excess balance. Providing a zero amount initiates a full validator exit, moving its state to `EXITING`. After a full exit is complete and the validator's balance is verified to be zero, its state becomes `EXITED`. Finally, the `Registrator` calls `removeSsvValidator` to decommission it from the SSV network, marking its final state as `REMOVED`.

4.3 On-Chain State Verification via Beacon Proofs

The protocol ensures the integrity of its accounting by forgoing reliance on off-chain data feeds. Instead, it employs a permissionless system of on-chain cryptographic verification using Merkle proofs. The EIP-4788 Beacon Block Roots oracle serves as the on-chain source of truth, providing the necessary state roots against which these proofs are validated.

Anyone can call the verification functions, but they are expected to be run by an off-chain keeper that generates the necessary proofs.

- **`verifyValidator`**: As part of the lifecycle, this function verifies a new validator's public key and, crucially, its withdrawal credentials against the Beacon Chain state. This step is mandatory before significant capital is deposited.
- **`verifyDeposit`**: This function is used to prove that a pending deposit has been successfully processed by the Beacon Chain. This transitions the value of the deposit from being "pending" to being part of a validator's balance in the protocol's internal accounting.
- **`verifyBalances`**: This function verifies the current ETH balance of every active validator against a specific Beacon Chain state root. It is the primary mechanism through which the protocol recognizes consensus layer rewards that have compounded on the validators' balances.

4.4 Balance and Rewards Accounting

The strategy's total value is composed of multiple components: idle WETH and ETH in the contract, ETH in pending deposits, and ETH held in validators on the Beacon Chain. A periodic two-step process ensures this value is accurately accounted for on-chain.

Consensus layer rewards are automatically compounded into the validator balances on the Beacon Chain. Execution layer rewards, such as MEV and priority fees, are sent directly to the strategy contract as native ETH.

- **Snapshot**: The process begins with a call to `snapBalances`. This function records a snapshot of the strategy contract's current native ETH balance and the corresponding parent Beacon Chain block root.
- **Verification**: Following the snapshot, an off-chain keeper provides proofs to the `verifyBalances` function. This function verifies the balance of every active validator and confirms that all pending deposits are still in the queue for the snapped block root. Upon successful verification, the protocol sums the snapped ETH balance, the total value of all pending deposits, and the total verified validator balances. This sum is stored in `lastVerifiedEthBalance`, which serves as the canonical, on-chain record of the strategy's total assets under management.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security 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 Security 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 [Info] Off-by-one error prevents reaching MAX_VERIFIED_VALIDATORS limit

File(s): contracts/strategies/NativeStaking/CompoundingValidatorManager.sol

Description: The `stakeEth(...)` function is used to perform the initial `1 ETH` deposit for a new validator that is in the `REGISTERED` state. The contract defines a constant `MAX_VERIFIED_VALIDATORS` set to `48`, which is intended to be the maximum number of validators the system can manage.

However, the check to enforce this limit contains an off-by-one error. The `require` statement uses a strict less-than (<) operator, which prevents the number of validators from reaching the intended maximum.

```
function stakeEth(...) external onlyRegistrator whenNotPaused {
    // ...
    bytes32 pubKeyHash = _hashPubKey(validatorStakeData.pubkey);
    ValidatorState currentState = validator[pubKeyHash].state;
    // ...
    if (currentState == ValidatorState.REGISTERED) {
        // ...
        // Limits the number of validator balance proofs to verifyBalances
        // @audit-issue This check should use `<=` to allow exactly 48 validators.
        require(
            verifiedValidators.length + 1 < MAX_VERIFIED_VALIDATORS,
            "Max validators"
        );

        // ...
        firstDeposit = true;
        validator[pubKeyHash].state = ValidatorState.STAKED;
    }
    // ...
}
```

When there are `47` validators in the `verifiedValidators` array, `verifiedValidators.length` is `47`. The check evaluates to `47 + 1 < 48`, which is `48 < 48`. This condition is `false`, causing the transaction to revert. As a result, it is impossible to stake for the 48th validator, effectively limiting the system to a maximum of `47` validators instead of the intended `48`.

Recommendation(s): Consider changing the strict inequality operator (<) to a less-than-or-equal-to operator (<=). This will ensure that the number of verified validators can reach the `MAX_VERIFIED_VALIDATORS` limit.

Status: Fixed

Update from the client: Fixed in #2683

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 they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about Origin Protocol's documentation

The Origin Protocol's Compounding Staking Strategy is **exceptionally well-documented**. The codebase is thoroughly commented, featuring detailed NatSpec documentation for every function. Extensive inline comments clarify the rationale behind specific implementations and thoroughly address potential edge cases. This high level of detail, combined with clear and consistent naming conventions for variables and functions, makes the codebase easy to follow and understand, despite its inherent complexity. Throughout the engagement, the Origin Protocol team was highly responsive, readily available for calls, and provided clear and comprehensive answers to all questions raised by the Nethermind Security auditors.

8 Test Suite Evaluation

8.1 Tests Output

```
$ rm -rf deployments/hardhat && IS_TEST=true npx hardhat test test/strategies/compoundingSSVStaking.js
↪  test/beacon/beaconProofs.js && yarn test:fork -- test/beacon/beaconProofs.mainnet.fork-test.js
↪  test/beacon/beaconRoots.mainnet.fork-test.js test/beacon/partialWithdrawal.mainnet.fork-test.js


  Unit test: Compounding SSV Staking Strategy
    Governable behaviour
Running 000_mock deployment...
Deployer address 0x17BAd8cbCDeC350958dF0Bfe01E284dd8Fec3fcD
Governor address 0x3cECEAe65A70d7f5b7D579Ba25093A37A47706B3
000_mock deploy done.
Running 001_core deployment...
001_core deploy done.
       Should have governor set
       Should detect if governor set or not
       Should not allow transfer of arbitrary token by non-Governor
       Should allow governor to transfer governance
       Should not allow anyone to transfer governance
       Should not allow anyone to claim governance
       Should allow governor to transfer governance multiple times
    Strategy behaviour
       Should have vault configured
       Should be a supported asset
       Should NOT be a supported asset
       Should allow transfer of arbitrary token by Governor (39ms)
       Should not transfer supported assets from strategy
       Should not allow transfer of arbitrary token by non-Governor
       Should not allow transfer of supported token
       Should allow the harvester to be set by the governor
       Should not allow the harvester to be set by non-governor (42ms)
       Should allow reward tokens to be set by the governor
       Should not allow reward tokens to be set by non-governor
      with no assets in the strategy
         Should check asset balances
         Should be able to deposit each asset
         Should not allow deposit by non-vault
         Should be able to deposit all asset together
         Should not be able to deposit zero asset amount
         Should not allow deposit all by non-vault
         Should not be able to withdraw zero asset amount
         Should not allow withdraw by non-vault
         Should be able to call withdraw all by vault
         Should be able to call withdraw all by governor
         Should not allow withdraw all by non-vault or non-governor
      with assets in the strategy
         Should check asset balances
         Should be able to withdraw each asset to the vault
         Should be able to withdraw all assets
    Initial setup
       Should anyone to send ETH
       SSV network should have allowance to spend SSV tokens of the strategy
    Configuring the strategy
       Governor should be able to change the registrator address
       Non governor should not be able to change the registrator address
       Should support WETH as the only asset
       Should not collect rewards
       Should not set platform token
       Should not remove platform token
       Non governor should not be able to withdraw SSV
       Non governor should not be able to reset the first deposit flag
       Should revert reset of first deposit if there is no first deposit
       Registrator or governor should be the only ones to pause the strategy
    Register, stake, withdraw and remove validators
bigint: Failed to load bindings, pure JS will be used (try npm run rebuild?)
       Should stake to a validator: 1 ETH (237ms)
       Should stake 1 ETH then 2047 ETH to a validator (324ms)
       Should revert when first stake amount is not exactly 1 ETH
       Should revert registerSsvValidator when contract paused
       Should revert stakeEth when contract paused
       Should revert when registering a validator that is already registered
       Should revert when staking because of insufficient ETH balance
       Should revert when staking a validator that hasn't been registered
       Should exit a validator with no pending deposit (398ms)
       Should exit a validator that is already exiting (393ms)
       Should revert when validator's balance hasn't been confirmed to equal or surpass 32.25 ETH (315ms)
       Should revert partial withdrawal when validator's balance hasn't been confirmed to equal or surpass 32 ETH
       ↪  (3167ms)
       Should revert when exiting a validator with a pending deposit (493ms)
       Should revert when verifying deposit between snapBalances and verifyBalances (140ms)
       Should partial withdraw from a validator with a pending deposit (495ms)
       Should remove a validator when validator is registered
       Should revert when removing a validator that is not registered
       Should remove a validator when validator is exited (562ms)
       Should not remove a validator if it still has a pending deposit (705ms)
       Should revert when removing a validator that has been found (78ms)
    Verify deposits
       Should revert first pending deposit slot is zero
       Should revert when no deposit
       Should revert when deposit verified again (78ms)
       Should revert when processed slot is after snapped balances
       Should verify deposit with no snapped balances (70ms)
       Should verify deposit with processed slot 1 before the snapped balances slot (84ms)
       Should verify deposit with processed slot well before the snapped balances slot (81ms)
    Deposit/Withdraw in the strategy
       Should deposit ETH in the strategy
       Should depositAll ETH in the strategy when depositedWethAccountedFor is zero
       Should depositAll ETH in the strategy when depositedWethAccountedFor is not zero
       Should revert when depositing 0 ETH in the strategy
       Should withdraw ETH from the strategy, no ETH
       Should withdraw ETH from the strategy, withdraw some ETH
       Should revert when withdrawing other than WETH
       Should revert when withdrawing 0 ETH from the strategy
       Should revert when withdrawing to the zero address
       Should withdrawAll ETH from the strategy, no ETH
       Should withdrawAll ETH from the strategy, withdraw some ETH
    Strategy balances
      When no execution rewards (ETH), no pending deposits and no active validators
         Should verify balances with no WETH
         Should verify balances with some WETH transferred before snap
         Should verify balances with some WETH transferred after snap
         Should verify balances with some WETH transferred before and after snap (42ms)
         Should verify balances with one registered validator (44ms)
         Should verify balances with one staked validator (145ms)
         Should verify balances with one exited verified validator (230ms)
         Should not verify a validator with incorrect withdrawal credential validator type (83ms)
         Should not verify a validator with incorrect withdrawal zero padding (88ms)
         Should verify balances with one verified deposit (270ms)
      When an active validator does a
        partial withdrawal
           Should account for a pending partial withdrawal (94ms)
           Should account for a processed partial withdrawal (75ms)
        full withdrawal
           Should account for full withdrawal (128ms)
      When WETH, ETH, no pending deposits and 2 active validators
         consensus rewards are earned by the validators (105ms)
         execution rewards are earned as ETH in the strategy (100ms)
        when balances have been snapped
           Fail to verify balances with not enough validator leaves
           Fail to verify balances with too many validator leaves
           Fail to verify balances with not enough validator proofs
           Fail to verify balances with too many proofs
      With 21 active validators
         Should verify balances with some WETH, ETH and no deposits (994ms)
         Should verify balances with one validator exited with two pending deposits (3581ms)
         Should verify balances with one validator exited with two pending deposits and three deposits to non-exiting
         ↪  validators (1246ms)
    Compounding SSV Staking Strategy Mocked proofs
       Should be allowed 2 deposits to an exiting validator  (310ms)
       Should verify validator that has a front-run deposit (116ms)
       Should verify validator with incorrect type (67ms)
       Should verify validator with malformed credentials (73ms)
       Should fail to verify front-run deposit (151ms)
       Governor should reset first deposit after front-run deposit (92ms)
```

```
        Should remove a validator from SSV cluster when validator is invalid (95ms)
        Should fail to active a validator with a 32.25 ETH balance (170ms)
        Should active a validator with more than 32.25 ETH balance (169ms)
      When a verified validator is exiting after being slashed And a new deposit is made to the validator
          Should fail verify deposit when first pending deposit slot before the withdrawable epoch
          Should verify deposit when the pending deposit queue is empty
          Should verify deposit when the first pending deposit slot equals the withdrawable epoch
          Should verify deposit when the first pending deposit slot is after the withdrawable epoch
        When deposit has been verified to an exiting validator
            Should verify balances

  Beacon chain proofs
    Should calculate generalized index
        from height and index
        for BeaconBlock.slot
        for BeaconBlock.parentRoot
        for BeaconBlock.body
        for BeaconBlock.BeaconBlockBody.randaoReveal
        for BeaconBlock.BeaconState.balances
        for BeaconBlock.body.executionPayload.blockNumber
    Should merkleize
        pending deposit
        BLS signature
    Balances container to beacon block root proof
        Should verify
        Fail to verify with zero beacon block root
        Fail to verify with invalid beacon block root
        Fail to verify with zero padded proof
        Fail to verify with invalid proof
        Fail to verify with invalid beacon container root
    Validator balance to balances container proof
        Should verify with balance
        Fail to verify with incorrect balance
        Fail to verify with zero container root
        Fail to verify with incorrect container root
        Fail to verify with zero padded proof
        Fail to verify with no balance
    Validator public key to beacon block root proof
        Should verify a 0x02 validator
        Should verify a 0x01 validator
        Fail to verify with zero beacon block root
        Fail to verify with invalid beacon block root
        Fail to verify with zero padded proof
        Fail to verify with invalid withdrawal address
        Fail to verify when the validator type does not match
        Fail to verify with withdrawal credential prefix 0x0210000000000000000000000
        Fail to verify with withdrawal credential prefix 0x0201000000000000000000000
        Fail to verify with withdrawal credential prefix 0x0200000000001000000000000
        Fail to verify with withdrawal credential prefix 0x0200000000000000000000010
        Fail to verify with withdrawal credential prefix 0x0200000000000000000000001
    Validator withdrawable epoch to beacon block root proof
      when validator is not exiting
          Should verify
          Fail to verify with zero beacon block root
          Fail to verify with invalid block root
          Fail to verify with invalid validator index
          Fail to verify with invalid withdrawable epoch
          Fail to verify with zero padded withdrawable epoch proof
      when validator is exiting
          Should verify
          Fail to verify with invalid withdrawable epoch
    Pending deposit container to beacon block root proof
        Should verify
        Fail to verify with zero beacon block root
        Fail to verify with invalid beacon block root
        Fail to verify with zero padded proof
        Fail to verify with invalid proof
        Fail to verify with invalid pending deposit container root
    Pending deposit in pending deposit container proof
        Should verify
        Fail to verify with incorrect pending deposit root
        Fail to verify with zero container root
        Fail to verify with incorrect container root
        Fail to verify with zero padded proof
        Fail to verify with invalid deposit index
```

```
       Fail to verify a pending deposit index that is too big
    First pending deposit to beacon block root proof
      for verifyDeposit which only checks the deposit slot
        with pending deposit
           Should verify
           Fail to verify with zero beacon block root
           Fail to verify with invalid beacon block root
           Fail to verify with zero padded proof
           Fail to verify with incorrect slot
        with no pending deposit
           Should verify with zero slot
           Should verify with non-zero slot
           Fail to verify with zero beacon root
           Fail to verify with invalid beacon root
           Fail to verify with zero padded proof


  182 passing (1m)

yarn run v1.22.22
warning From Yarn 1.0 onwards, scripts don't require "--" for options to be forwarded. In a future version, any explicit
→  "--" will be forwarded as-is to the scripts.
$ ./fork-test.sh test/beacon/beaconProofs.mainnet.fork-test.js test/beacon/beaconRoots.mainnet.fork-test.js
→  test/beacon/partialWithdrawal.mainnet.fork-test.js
Fork Network: mainnet
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
curl: (7) Failed to connect to localhost port 8545 after 3 ms: Couldn't connect to server
No running node detected spinning up a fresh one
Test params: test/beacon/beaconProofs.mainnet.fork-test.js test/beacon/beaconRoots.mainnet.fork-test.js
→  test/beacon/partialWithdrawal.mainnet.fork-test.js
Running fork tests...
Starting a fresh node on block: undefined


  ForkTest: Beacon Proofs
bigint: Failed to load bindings, pure JS will be used (try npm run rebuild?)
    1) "before all" hook for "Should verify validator public key"

  ForkTest: Beacon Roots
     Should get the latest beacon root (521ms)
     Should to get beacon root from the current block (1071ms)
     Should to get beacon root from the previous block (1575ms)
     Should get beacon root from old block (1294ms)
     Fail to get beacon root from block older than the buffer (1183ms)

  ForkTest: Partial Withdrawal
Running 150_vault_upgrade deployment...
INFO: Deployed OETHVaultCore Address: 0x268f59D9A3a77fD3EDe23FCa2fc9Ca5c91d656b1 Gas Used: 3694451
INFO: Sending the governance proposal to xOGN governance
INFO: Submitting proposal for Upgrade VaultCore and set Vault as the recipient of the WETH rewards on the Simple
→  harvester
INFO: Args: [
  [
    "0x39254033945AA2E4809Cc2977E7087BEE48bd7Ab",
    "0x6D416E576eECBB9F897856a7c86007905274ed04"
  ],
  [
    {
      "type": "BigNumber",
      "hex": "0x00"
    },
    {
      "type": "BigNumber",
      "hex": "0x00"
    }
  ],
  [
    "upgradeTo(address)",
    "setDripper(address)"
  ],
  [
    "0x000000000000000000000000268f59d9a3a77fd3ede23fca2fc9ca5c91d656b1",
    "0x00000000000000000000000039254033945aa2e4809cc2977e7087bee48bd7ab"
  ]
]
INFO: Reducing required voting delay to 1 block and voting period to 60 blocks vote extension on late vote to 0 and
↪ timelock min delay to 5 seconds
INFO: Impersonated Guardian at 0xbe2AB3d3d8F6a32b96414ebbd865dBD276d3d899
INFO: Submitted governance proposal to xOGN governance
↪ 89774398822204939588376738490052434220515812229305241959075531139837105320408
INFO: Executing the proposal
INFO: Impersonated Guardian at 0xbe2AB3d3d8F6a32b96414ebbd865dBD276d3d899
INFO: Reducing required voting delay to 1 block and voting period to 60 blocks vote extension on late vote to 0 and
↪ timelock min delay to 5 seconds
INFO: Advancing 2 blocks to make transaction for from Pending to Active
INFO: Advancing 10 blocks to make transaction for from Active to Succeeded
INFO: Proposal queued
INFO: preparing to execute
INFO: Advancing to the end of the queue period (on the timelock): 30
INFO: Proposal id: 89774398822204939588376738490052434220515812229305241959075531139837105320408 executed
INFO: Proposal executed.
150_vault_upgrade deploy done.
Running 153_upgrade_native_staking deployment...
INFO: Deployed NativeStakingSSVStrategy Address: 0x2495e8E4FBE36C30EAAD4D473F674d947C0d9A42 Gas Used: 4663952
Deployed 2nd NativeStakingSSVStrategy 0x2495e8E4FBE36C30EAAD4D473F674d947C0d9A42
INFO: Deployed NativeStakingSSVStrategy Address: 0xc4444C5D9e7C1a5A0a01c5E4b11692d589DcAF22 Gas Used: 4663952
Deployed 3rd NativeStakingSSVStrategy 0xc4444C5D9e7C1a5A0a01c5E4b11692d589DcAF22
INFO: Deployed BeaconProofs Address: 0x452bB04D113478578dD23026E4f86A8FFb0118ca Gas Used: 1354130
INFO: Deployed CompoundingStakingSSVStrategy Address: 0x14A94976536c79cb4D7F7188F46602D75660d0ee Gas Used: 5357068
INFO: Deployed CompoundingStakingStrategyView Address: 0xa621CE592E528f2B640cbBD1d99465D9C80d8B0f Gas Used: 534965
INFO: Sending the governance proposal to xOGN governance
INFO: Submitting proposal for Upgrade the existing Native Staking Strategies and deploy new Compounding Staking Strategy
INFO: Args: [
  [
    "0x4685dB8bF2Df743c861d71E6cFb5347222992076",
    "0xE98538A0e8C2871C2482e1Be8cC6bd9F8E8fFD63",
    "0x39254033945AA2E4809Cc2977E7087BEE48bd7Ab",
    "0x840081c97256d553A8F234D469D797B9535a3B49",
    "0x840081c97256d553A8F234D469D797B9535a3B49"
  ],
  [
    {
      "type": "BigNumber",
      "hex": "0x00"
    },
    {
      "type": "BigNumber",
      "hex": "0x00"
    },
    {
      "type": "BigNumber",
      "hex": "0x00"
    },
    {
      "type": "BigNumber",
      "hex": "0x00"
    },
    {
      "type": "BigNumber",
      "hex": "0x00"
    }
  ],
  [
    "upgradeTo(address)",
    "upgradeTo(address)",
    "approveStrategy(address)",
    "setHarvesterAddress(address)",
    "setRegistrator(address)"
  ],
  [
    "0x0000000000000000000000002495e8e4fbe36c30eaad4d473f674d947c0d9a42",
    "0x000000000000000000000000c4444c5d9e7c1a5a0a01c5e4b11692d589dcaf22",
    "0x000000000000000000000000840081c97256d553a8f234d469d797b9535a3b49",
    "0x0000000000000000000000004b91827516f79d6f6a1f292ed99671663b09169a",
    "0x0000000000000000000000004b91827516f79d6f6a1f292ed99671663b09169a"
  ]
]
INFO: Reducing required voting delay to 1 block and voting period to 60 blocks vote extension on late vote to 0 and
↪ timelock min delay to 5 seconds
```

```
INFO: Impersonated Guardian at 0xbe2AB3d3d8F6a32b96414ebbd865dBD276d3d899
INFO: Submitted governance proposal to xOGN governance
↪ 79749484602467943882470626433043585806856595621023635180361278414920774084624
INFO: Executing the proposal
INFO: Impersonated Guardian at 0xbe2AB3d3d8F6a32b96414ebbd865dBD276d3d899
INFO: Reducing required voting delay to 1 block and voting period to 60 blocks vote extension on late vote to 0 and
↪ timelock min delay to 5 seconds
INFO: Advancing 2 blocks to make transaction for from Pending to Active
INFO: Advancing 10 blocks to make transaction for from Active to Succeeded
INFO: Proposal queued
INFO: preparing to execute
INFO: Advancing to the end of the queue period (on the timelock): 30
INFO: Proposal id: 79749484602467943882470626433043585806856595621023635180361278414920774084624 executed
INFO: Proposal executed.
153_upgrade_native_staking deploy done.
INFO: Deployed MockOracleRouterNoStale Address: 0xFB84996A72EF5A4fd4F0242c028450D0A5d7Fbc2 Gas Used: 747820
INFO: Deployed MockOETHOracleRouterNoStale Address: 0x412E1c7C39081063f1ee4febaAb44A496385c143 Gas Used: 540870
INFO: Deployed ExecutionLayerConsolidation Address: 0xFB5D83b455eFe511616908010C77c4ceeE75fCc1 Gas Used: 312059
INFO: Deployed ExecutionLayerWithdrawal Address: 0xB21A919544334f07215C2d70786B0E8E84a8e902 Gas Used: 329152
     Should get consolidation fee
     Should request a partial withdrawal (1255ms)


  7 passing (2m)
  1 failing

  1) ForkTest: Beacon Proofs
       "before all" hook for "Should verify validator public key":
     Error: Must set at least 1 URL in HttpClient opts
      at new HttpClient
       ↪ (file:///.../NM-0645-ORIGIN/origin-dollar/contracts/node_modules/@lodestar/api/src/utils/client/httpClient.ts
:128:13)
      at getClient
       ↪ (file:///.../NM-0645-ORIGIN/origin-dollar/contracts/node_modules/@lodestar/api/src/beacon/client/index.ts:33
:44)
      at configClient (utils/beacon.js:176:24)
      at getSlot (utils/beacon.js:42:18)
      at Context.<anonymous> (test/beacon/beaconProofs.mainnet.fork-test.js:29:25)



Done in 106.75s.
```

8.2 Automated Tools

8.2.1 AuditAgent

All the relevant issues raised by the AuditAgent have been incorporated into this report. The AuditAgent is an AI-powered smart contract auditing tool that analyses code, detects vulnerabilities, and provides actionable fixes. It accelerates the security analysis process, complementing human expertise with advanced AI models to deliver efficient and comprehensive smart contract audits. Available at https://app.auditagent.nethermind.io.

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 conducts cutting-edge internal research and collaborates closely with external partners on cryptographic protocols, consensus design, succinct arguments and folding schemes, elliptic curve-based STARK protocols, post-quantum security and zero-knowledge proofs (ZKPs). Our research has led to influential contributions, including Zinc (Crypto '25), Mova, FLI (Asiacrypt '24), and foundational results in Fiat-Shamir security and STARK proof batching. Complementing this theoretical work, our engineering expertise is demonstrated through implementations such as the Latticefold aggregation scheme, the Labrador proof system, zkvm-benchmarks, and Plonk Verifier in Cairo. This combined strength in theory and engineering enables us to deliver cutting-edge cryptographic solutions to 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.

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.