Security Review Report
NM-0179 CLAYSTACK

NETHERMIND SECURITY

(Jan 25, 2024)

Contents

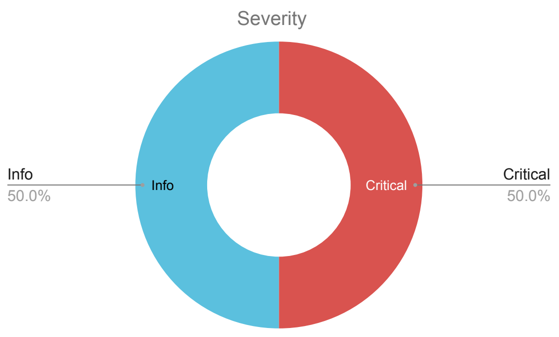
1 Executive Summary

This document presents the security review conducted by Nethermind for the ClayStack ETH Staking Contracts. `ClayStack` is a platform that enables ETH staking on the Ethereum blockchain. It allows users to participate in the staking process by delegating their stake to validators on the Ethereum Beacon chain. This staking process involves users locking up their ETH in exchange for a token called csETH. This csETH token represents their staked amount and is minted on the Ethereum network. The validator pool on `ClayStack` is managed by the platform, and those who operate and manage the validators are rewarded for their efforts. These operators can either be direct nodes or operators using a DVT based approach. In the audited version of `ClayStack`, validators are trusted/controlled entities that are not likely to be malicious.

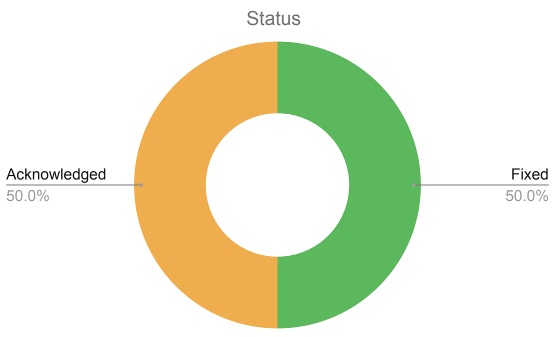
This assessment focuses on the recent update to the `ClayStack` protocol that implements integration with the EigenLayer protocol, enabling the restaking of native ETH. Notably, registered validators can designate a withdrawal address pointing to the `EigenPod` contract linked with ClayStack's `NodeManager`. The contract has been modified to incorporate new functions that enable the withdrawal and claiming processes directly from the `EigenPod`. Additionally, when oracles report the exit of a validator associated with the `EigenPod`, a claiming process is initiated to claim expired withdrawal requests. The claimed amount is then reflected in the validator's balance. The node is considered exited only if all withdrawal requests have been claimed. Otherwise, it will be subtracted from the total number of exited nodes.

The audit was performed using: (a) manual analysis of the codebase, (b) automated analysis tools, and (c) simulation of the smart contracts. **Along this document, we report** 2 points of attention, where one is classified as `Critical` and one is classified as `Informational`. 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 discusses the risk rating methodology adopted for this audit. Section 5 details the issues. Section 6 discusses the documentation provided by the client for this audit. Section 7 presents the compilation, tests, and automated tests. Section 8 concludes the document.



(a) distribution of issues according to the severity



(b) distribution of issues according to the status

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

Summary of the Audit

Audit Type	Security Review
Initial Report	Jan 25, 2024
Final Report	Jan 25, 2024
Methods	Manual Review, Automated Analysis
Repository	claystack-ethereum
Commit Hash	0de890799faef9df406c69361e83fe479e59ba7f
Commit Hash	184f9a9e8c90d19021640340a485edb8e4b68b48
Documentation	Docs
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/ClayMain.sol	571	170	29.8%	148	889
2	src/NodeManager.sol	431	152	35.3%	111	694
	Total	**1002**	**322**	**32.1%**	**259**	**1583**

3 Summary of Issues

	Finding	Severity	Update
1	Claiming withdrawals on EigenLayer can be done without updating internal accounting	Critical	Fixed
2	Updating Eigen Layer addresses could result in erroneous accounting	Info	Acknowledged

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 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.

5 Issues

5.1 [Critical] Claiming withdrawals on EigenLayer can be done without updating internal accounting

File(s): NodeManager.sol

Description: The `claimEigenLayer(...)` function shown below is claiming withdrawals within EigenLayer and updating the accounting in `eigenLayerWithdraws` variable.

```
function claimEigenLayer(uint256 maxNumberOfWithdrawalsToClaim) public {
    uint256 balanceBefore = address(this).balance;
    // @audit call to EigenLayer to claim withdrawals
    eigenDelayedWithdrawalRouter.claimDelayedWithdrawals(maxNumberOfWithdrawalsToClaim);
    uint256 balanceAfter = address(this).balance;
    if (balanceAfter > balanceBefore) {
        uint256 claimedAmount = balanceAfter - balanceBefore;
        // @audit update of the internal accounting
        eigenLayerWithdraws -= claimedAmount;
        emit LogClaimEigenLayer(claimedAmount);
    }
}
```

However, anyone can bypass this function and execute the claim directly on EigenLayer side, by calling the `claimDelayedWithdrawals(...)` function. This function allows the caller to claim withdrawals on behalf of a specified recipient, with the recipient address provided as an input. If the `NodeManager` contract address is utilized as the `recipient`, the claim is made on its behalf.

Consequently, the `eigenLayerWithdraws` variable won't be updated, resulting in wrong accounting within the contract. Despite funds being claimed and nodes considered exited, they are still accounted for in the active validators' balance, resulting in higher rewards.

Recommendation(s): Consider modifying the way the contract handles claims in EigenLayer, by moving the accounting for claimed amounts to the `receive` function, and filtering the funds received from the `eigenDelayedWithdrawalRouter` contract only.

Status: Fixed

Update from the client: updated with commit: 184f9a9e8c90d19021640340a485edb8e4b68b48

5.2 [Info] Updating Eigen Layer addresses could result in erroneous accounting

File(s): NodeManager.sol

Description: The `setEigenLayer(...)` function allows updating EigenLayer related addresses. This update carries a risk of affecting the balances accounting within `NodeManager` contract. That is possible because, after the update, the contract loses access to the queue of pending and completed withdrawals of the current validators, in addition to the current balance of the EigenPod.

Recommendation(s): Before updating EigenLayer addresses in the contract, consider reviewing the status of validators and withdrawal claims in the previous router contract. Confirm that all withdrawals have been successfully claimed and all nodes are exited.

Status: Acknowledged

Update from the client: -

6 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 ClayStack documentation

The ClayStack contracts' documentation is presented through inline comments within the code, providing explanations for the objectives and operations of various functions and formulas. Additionally, the README file, accessible at this link, outlines the deposit and withdrawal processes and delineates different roles within the system. Furthermore, the `ClayStack` team remained available to address any inquiries or concerns raised by the Nethermind auditors.

7 Test Suite Evaluation

7.1 Contracts Compilation

```
> forge compile
[] Compiling...
[] Compiling 222 files with 0.8.18
[] Solc 0.8.18 finished in 54.12s
Compiler run successful with warnings.
```

7.2 Tests Output

```
> forge test
[] Compiling...
No files changed, compilation skipped

Running 5 tests for test/operators/Account.t.sol:Account
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 11.57ms

Running 10 tests for test/staking/RoleManager.t.sol:TestRoleManager
Test result: ok. 10 passed; 0 failed; 0 skipped; finished in 13.63ms

Running 6 tests for test/staking/RateProtection.t.sol:RateProtection
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 28.20ms

Running 6 tests for test/eigenlayer/EigenPod.t.sol:EigenPodBase
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 50.34ms

Running 3 tests for test/staking/RegisterValidator.t.sol:TestRegisterValidator
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 27.36ms

Running 3 tests for test/staking/Setters.t.sol:TestSetters
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 69.55ms

Running 2 tests for test/operators/RewardDistributor.t.sol:RewardDistributorTest
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 4.32ms

Running 4 tests for test/operators/Rewards.t.sol:Rewards
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 11.39ms

Running 16 tests for test/staking/AutoBalance.t.sol:AutoBalance
Test result: ok. 16 passed; 0 failed; 0 skipped; finished in 111.93ms

Running 2 tests for test/operators/ssv/Staking.SSV.t.sol:SSV
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 21.85ms

Running 5 tests for test/staking/TimeLock.t.sol:TestTimeLock
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 10.10ms

Running 6 tests for test/staking/InstantWithdrawal.t.sol:TestInstantWithdraw
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 111.12ms

Running 8 tests for test/staking/PreDeposits.t.sol:PreDeposits
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 20.06ms

Running 5 tests for test/staking/NodeManager.t.sol:TestNodeManager
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 36.96ms

Running 5 tests for test/deposits/prestaking/PreStaking.t.sol:TestPreStaking
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 30.52ms

Running 3 tests for test/staking/UserClaimsView.t.sol:TestUserClaimsView
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 92.58ms

Running 8 tests for test/staking/CsToken.t.sol:TestCsToken
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 268.98ms
```

```
Running 15 tests for test/staking/Oracle.t.sol:Oracle
Test result: ok. 15 passed; 0 failed; 0 skipped; finished in 65.87ms

Running 19 tests for test/deposits/stETH.t.sol:stETHTest
Test result: ok. 19 passed; 0 failed; 0 skipped; finished in 39.20ms

Running 18 tests for test/deposits/rETH.t.sol:rETHTest
Test result: ok. 18 passed; 0 failed; 0 skipped; finished in 8.55ms

Running 34 tests for test/deposits/wstETH.t.sol:wstETH)
Test result: ok. 34 passed; 0 failed; 0 skipped; finished in 34.36ms

Running 8 tests for test/staking/Penalties.t.sol:Penalties
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 157.94ms

Running 17 tests for test/staking/OracleNetwork.t.sol:OracleNetwork
Test result: ok. 17 passed; 0 failed; 0 skipped; finished in 300.30ms

Running 8 tests for test/staking/SelfDestructAttack.t.sol:SelfDestructAttack
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 10.66ms

Running 10 tests for test/staking/Withdraw.t.sol:TestWithdraw
Test result: ok. 10 passed; 0 failed; 0 skipped; finished in 418.28ms

Running 4 tests for test/staking/Donations.test.sol:TestDonations
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 503.00ms

Running 8 tests for test/staking/SetFee.t.sol:TestSetFee
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 1.82s

Running 10 tests for test/staking/Deposit.t.sol:TestDeposit
Test result: ok. 10 passed; 0 failed; 0 skipped; finished in 2.00s

Running 7 tests for test/operators/Direct.t.sol:Direct
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 2.96s

Running 8 tests for test/operators/ssv/SSV.t.sol:SSV
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 3.59s

Running 18 tests for test/staking/Claim.t.sol:TestClaim
Test result: ok. 18 passed; 0 failed; 0 skipped; finished in 35.56s

Ran 31 test suites: 281 tests passed, 0 failed, 0 skipped (281 total tests)
```

7.3 Code Coverage

```
> forge coverage
```

The relevant output is presented below.

```
| File                                    | % Lines            | % Statements        | % Branches         | % Funcs           |
|-----------------------------------------|--------------------|---------------------|--------------------|-------------------|
| src/ClayMain.sol                        | 99.16% (354/357)   | 99.33% (445/448)    | 85.56% (154/180)   | 100.00% (44/44)   |
| src/NodeManager.sol                     | 99.50% (198/199)   | 99.63% (270/271)    | 79.82% (91/114)    | 96.43% (27/28)    |
```

8 About Nethermind

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

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

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

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

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

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

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

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

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;

- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;

- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

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

Disclaimer