Security Review Report
NM-0550 SKATE Claim and Stake Hook

NETHERMIND SECURITY

(June 3, 2025)

Contents

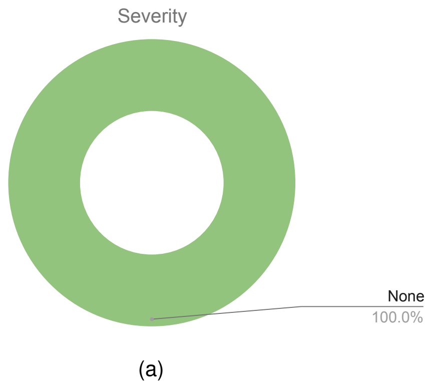
1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for Skate.

Skate is an infrastructure layer that enables users to interact seamlessly from their native chains by connecting them to Virtual machines across chains. It uses **LayerZero** for cross-chain communication. This audit engagement is focused on the integration of **Magna platform**'s protocol-level hooks with **Skate**. When the tokens on **Magna platform** are claimed by the user, using the protocol level hook of **Magna platform**, the **Skate** will automatically stake the claimed funds into the **Eigen Layer** in the same transaction. The implementation leverages the stake function with the signature of the Strategy Manager in **Eigen Layer** to achieve this. The claim-initiating function will capture the signature of the signer to process the transaction.

The audit comprises 74 lines of Solidity code. **The audit was performed using** (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases. The audit also consisted of verifying the post-deployment of the OFT Skate token on three chains: Ethereum, Binance Chain, and Solana. **No issues were found during the audit of the code in scope**.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 presents the system overview. Section 4 discusses the risk rating methodology. Section 5 discusses the documentation provided by the client for this audit. Section 6 presents the test suite evaluation and automated tools used. Section 7 concludes the document.



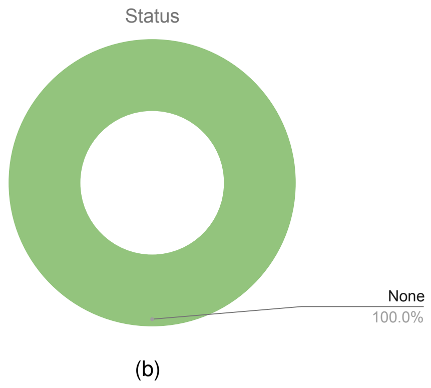


Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (0), **Best Practices** (0).
Distribution of status: Fixed (0), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Final Report	June 3, 2025
Repositories	Magna-Hook
Initial Commit	32d23ebec
Final Commit	32d23ebec
Documentation	Notion
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	ClaimAndStakeHandlerHook.sol	74	1	1.4%	13	88
	Total	**74**	**1**	**1.4%**	**13**	**88**

3 System Overview

The process is designed to allow users to claim their tokens from the Magna Platform and have them automatically staked into an EigenLayer strategy, all within a single atomic transaction.

The User (Staker) initiates a token claim on the Magna Platform. As part of this process, the Magna Platform collects the User's signature. This signature is intended to authorize not only the claim but also the subsequent staking action on EigenLayer.

Upon a successful claim, the Magna Platform's protocol level hook will be triggered. This hook calls the handlePostClaim function on the ClaimAndStakeHandlerHook, the handler contract implemented by Skate. When calling handlePostClaim, the Magna Platform transfers the claimed tokens to the ClaimAndStakeHandlerHook contract.

Magna Platform also passes the User's signature to the ClaimAndStakeHandlerHook Contract. This is typically done via extraData parameter of the hook function call.

The ClaimAndStakeHandlerHook contract receives the claimed tokens and the user's signature. Then it sends these funds to EigenLayer's strategy manager. To do this, it calls the depositIntoStrategyWithSignature function on the target EigenLayer strategy contract. The ClaimAndStakeHandlerHook uses the user's signature received from Magna Platform to authorize this deposit, effectively staking the tokens on behalf of the User.

If the signature is valid and all conditions are met, the EigenLayer strategy accepts the tokens, and they are staked on behalf of the user. The entire sequence, from claim to stake, occurs within the context of the initial transaction initiated by the user on the Magna Platform.

The following diagram provides a high-level illustration of the Skate - Magna Platform integration.

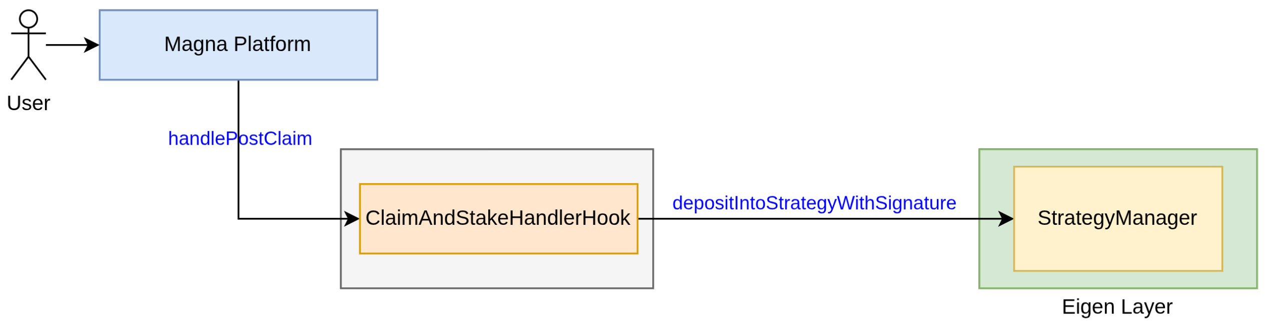


Fig. 2: Magna-Skate Integration

3.1 System Components

Magna Platform

The Magna Platform offers Protocol level hooks to handle uses cases related to handling of claimed tokens. In order to listen to the hook function, the handler contract implements IPostClaimHandler interface which has handlePostClaim hook function.

Skate

The Skate implemented ClaimAndStakeHandlerHook,a handler contract that will listen to the handlePostClaim hook function. The claimed funds are received into the handler contract, which then stakes the funds in favor of the last withdrawal address received as parameter to the hook. Magna Platform will pass the signature of the withdrawal address along with expiry in extraData field.

ClaimAndStakeHandlerHook has a reference to the StrategyManager. The handler contract stakes the received funds in favor of withdrawal address on the Eigen layer in the same transaction by calling depositIntoStrategyWithSignature function.

EigenLayer

EigenLayer's StrategyManager contract is the final destination for tokens claimed by users. It provides depositIntoStrategyWithSignature function that enables Skate to deposit and stake funds on behalf of the user using offchain signature.

3.2 Security Assumptions for the Skate-Magna Integration

The security and correct functioning of the integrated claiming (via Magna) and staking (via Skate into EigenLayer) process rely on the following core assumptions:

- **Standard ERC20 Token Behavior**: Skate is assumed to only interact with ERC20 tokens that adhere to the standard interface and behavior. Tokens with non-standard mechanics, such as "fee-on-transfer" tokens, rebasing tokens, or those with other weird behaviors that might alter transfer amounts or impede transfers, are explicitly not supported by this integration.
- **Origin of Signature**: The signature provided by the Magna Platform within the extraData parameter (which is passed to Skate's handlePostClaim function) is assumed to be a valid signature from the account designated as the Last withdrawal account. This implies that the signature directly authorizes actions on behalf of this specific user account for the staking operation on EigenLayer.
- **Exclusion of Multisig Wallets as Last Withdrawal Account**: The automated claim-and-stake flow facilitated by this Skate-Magna integration is not designed to support multisig wallets as the Last Withdrawal Account. The mechanism relies on a single signature, typically from an Externally Owned Account (EOA), and is not intended to handle the complexities of multisig authorization for this specific automated pathway.

3.3 Note regarding the post-deployment of OFT tokens

As part of their request, we reviewed their layer zero on-chain deployment for the OFT Skate token that was deployed on Ethereum, Binance, and Solana chains. The client used the default DVNs, receive, and send libraries of LayerZero. Two observations are worth mentioning:

- The trusted peers and connections to OFT deployments must be established only after configuring the DVN and Executor. This is because invoking `setPeer` opens the OFT deployments to receive messages from the messaging channel. Without prior DVN and Executor configuration, any bridge message will become stuck, as detailed in the LayerZero documentation. We noticed that in the transactions' configuration order, `setPeer` is the first transaction invoked, which doesn't align with the recommendation. The client acknowledged this and confirmed that they used the LayerZero CLI command to set all the configurations together, so the order mismatch came from the command execution rather than a manual configuration.
- Currently, only 1 DVN is configured, and the threshold is defaulted to zero; there is no problem with that, but we would recommend adding more DVNs and increasing the threshold accordingly.

4 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.

5 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 Skate documentation

The `Stake` team has provided a comprehensive walkthrough of the project in the kick-off call. The Notion page and the code comments include the explanation of the intended functionalities. Moreover, the team addressed the questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

6 Test Suite Evaluation

6.1 Compilation Output

```
> forge build
[] Compiling...
[] Compiling 47 files with Solc 0.8.24
[] Solc 0.8.24 finished in 452.61ms
Compiler run successful with warnings:
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> src/ClaimAndStakeHandlerHook.sol:68:9:
   |
68 |         address originalBeneficiary,
   |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

6.2 Tests Output

```
> forge test
Ran 4 tests for test/ClaimAndStakeHandlerHook.t.sol:ClaimAndStakeHandlerHookTest
[PASS] testCannotReinitialize() (gas: 22001)
[PASS] testDepositThroughHook() (gas: 325400)
[PASS] testTransferOwnership() (gas: 29467)
[PASS] testUpgrade() (gas: 1563481)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 3.74s (3.07s CPU time)

Ran 1 test suite in 3.74s (3.74s CPU time): 4 tests passed, 0 failed, 0 skipped (4 total tests)
```

7 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