Security Review Report
NM-0744 DeepNode

(December 09, 2025)

Contents

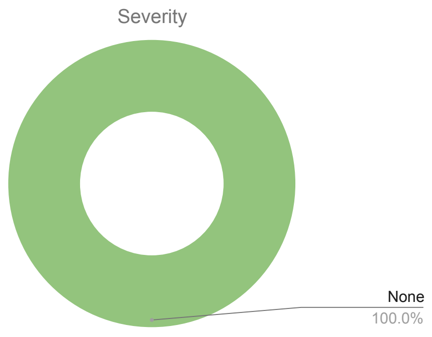
1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for DeepNode's ERC20 Token contract.

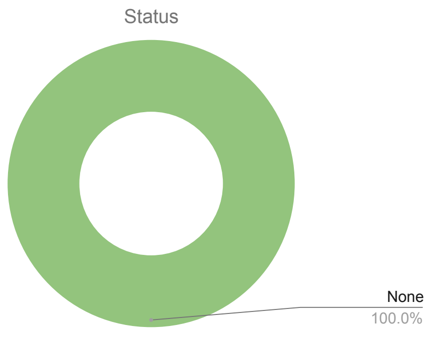
DeepToken is an ERC20 token that will be used as a staking reward token within the **DeepNode** ecosystem. The contract is an upgradable ERC20 that implements: role-based access control features, emergency pause capabilities, a banlist/blacklist mechanism, and **EIP2612 Permit** features.

The audit comprises 191 lines of the Solidity code. **The audit was performed using** (a) manual analysis of the codebase, and (b) automated analysis tools. **No issues were found in the audited contracts.**

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 details the issues. Section 6 discusses the documentation provided by the client for this audit. Section 7 presents the test suite evaluation and automated tools used. Section 8 concludes the document.



(a)



(b)

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
Initial Report	December 03, 2025
Final Report	December 09, 2025
Initial Commit	f675915
Final Commit	1b97f2c
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	DeepToken.sol	74	58	78.4%	18	150
2	base/BanlistControlUpgradeable.sol	49	34	69.4%	14	97
3	base/GuardControlUpgradeable.sol	21	20	95.2%	7	48
4	base/ERC20ControlUpgradeable.sol	47	33	70.2%	7	87
	Total	**191**	**145**	**75.9%**	**46**	**382**

3 System Overview

The DeepNode Protocol introduces `DeepToken` ($DN), an upgradeable ERC-20 token serving as the primary reward mechanism for the ecosystem's staking activities. The token is designed for deployment on Ethereum Mainnet with bridging capabilities to Base. The contract leverages OpenZeppelin's upgradeable standards to allow for future logic iterations while maintaining a persistent state.

The system logic is segmented into three primary functional areas: supply management, security restrictions, and access control.

3.1 Supply Management

The token supply is elastic but capped. Supply manipulation is strictly delegated to an address designated as the `EmissionsController`.

- **Minting**: Only the `EmissionsController` is authorized to mint new tokens. The contract enforces a global hard cap via the `MAX_TOTAL_SUPPLY_CAP` constant, preventing the total supply from ever exceeding 100,000,000 $DN.
- **Burning**: The `EmissionsController` acts as the exclusive burner, a function intended for reducing total supply using protocol-accrued fees or treasury funds. If the controller attempts to burn tokens held by another address, the contract enforces standard ERC-20 allowance checks before execution.

3.2 Security and Restrictions

To safeguard the protocol against malicious actors or unexpected anomalies, the token extends standard ERC-20 behavior with two restrictive modules. These checks are integrated directly into the `_update`, `_spendAllowance` and `_approve` hooks.

- **Guard Control**: Inheriting from `PausableUpgradeable`, this module allows the protocol to freeze all token transfers, minting, and burning. This acts as a circuit breaker during emergency scenarios.
- **Banlist Control**: This module maintains a registry of restricted addresses. If an address is added to the banlist, it is blocked from sending, receiving, or spending tokens. This is effectively used to isolate malicious wallets from the ecosystem.

3.3 Access Control

The system employs a strict separation of duties using `AccessControlUpgradeable`. Permissions are granular to prevent a single actor from having excessive power.

- **Admin Role**: Managing the overall configuration, including contract upgrades, setting the `EmissionsController` address, and managing role administration.
- **Guardian Role**: Restricted specifically to the `pause()` and `unpause()` functions within the Guard Control module.
- **Banlist Operator Role**: Restricted to the `ban()` and `unban()` functions within the Banlist Control module.

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 Issues

After careful review, no issues have been found in the codebase.

6 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 DeepNode's documentation

The development team provided an overview of the future use cases of the token contract. The development team was collaborative during the audit, providing answers to all questions raised by the Nethermind Security team during calls and through asynchronous communications.
The `DeepToken` contract has a comprehensive `README` that explains the features, behaviors, and roles of the contract.

7 Test Suite Evaluation

7.1 Tests Output

```
> yarn test

yarn run v1.22.22
$ hardhat test --network hardhat
[dotenv@17.2.3] injecting env (0) from .env -- tip:   enable debug logging with { debug: true }
Compiled 1 Solidity file successfully (evm target: paris).


  Deployment
    Implementation
       Should deploy implementation contract with proper address (1415ms)
       Should disable initializers on implementation contract
    Proxy admin
       Should deploy proxy admin contract with proper address
       Should deploy proxy admin contract with proper owner address
    Proxy
       Should deploy proxy contract with proper addresses
       Should set correct proxyAdmin address
       Should set correct implementation address
       Should have proper roles ids

  Initialization
    Initialize
       Should initialize with correct name
       Should initialize with correct symbol
       Should initialize with correct decimals
       Should initialize with correct admin of admin role
       Should initialize with correct admin of guardian role
       Should initialize with correct admin of banlist operator role
       Should initialize with correct admin address
       Should initialize with correct guardian address
       Should initialize with correct banlist operator address
       Should initialize with correct EmissionsController address
       Should emit EmissionsControllerUpdated event
       Should prevent re-initialization
       Should revert if EmissionsController address is not a contract
    Initial Mint
       Should mint initial supply to the initial receiver
       Should emit InitialMint event
       Should prevent initialization with non-zero initial amount and zero address as initial receiver
       Should prevent initialization with initial amount exceeding max supply cap
       Should initialize with zero initial amount and not change total supply

  ERC20 Operations
    Mint
       Should allow operator to mint tokens
       Should change recipient balance
       Should emit Mint event on successful minting
       Should emit Transfer event on successful minting
       Should increase total supply when minting
       Should prevent non-EmissionsController from minting tokens
       Should prevent minting beyond MAX_TOTAL_SUPPLY_CAP
       Should allow minting up to MAX_TOTAL_SUPPLY_CAP
       Should prevent minting to zero address
       Should revert if contract is paused
       Should allow minting after unpausing the contract
       Should revert if recipient is banned
       Should allow minting if recipient was unbanned
       Should prevent minting if EmissionsController address was changed
       Should allow new EmissionsController to mint tokens after update
    Burn
       Should allow EmissionsController to burn tokens
       Should allow EmissionsController to burn tokens from itself
       Should change user balance on burn
       Should emit Burn event on successful burn
       Should emit Transfer event on successful burn
       Should decrease total supply on burn
       Should prevent non-EmissionsController from burning tokens
```

```
      Should prevent burning more than user balance
      Should prevent burning more than user approved
      Should revert if contract is paused
      Should allow burning after unpausing the contract
      Should prevent burning if user is banned
      Should allow burning if user was unbanned
    Transfer
      Should transfer tokens between accounts
      Should emit Transfer event on successful transfer
      Should prevent transfer to zero address
      Should prevent transfer exceeding balance
      Should revert transfer if contract is paused
      Should allow transfer after unpausing the contract
      Should revert transfer if sender is banned
      Should revert transfer if recipient is banned
      Should allow transfer if sender was unbanned
      Should allow transfer if recipient was unbanned
    TransferFrom
      Should transfer tokens using allowance
      Should emit Transfer event on successful transferFrom
      Should spend allowance on transferFrom
      Should not spend allowance if allowance is unlimited (type(uint256).max)
      Should prevent transferFrom exceeding allowance
      Should prevent transferFrom exceeding balance
      Should prevent transferFrom to zero address
      Should revert transferFrom if contract is paused
      Should allow transferFrom after unpausing the contract
      Should revert transferFrom if `from` is banned
      Should revert transferFrom if `to` is banned
      Should revert transferFrom if spender is banned
      Should allow transferFrom if `from` was unbanned
      Should allow transferFrom if `to` was unbanned
    Approve
      Should set allowance with approve
      Should emit Approval event on approve
      Should prevent approve to zero address
      Should allow to change allowance to zero
      Should revert approve if contract is paused
      Should allow approve after unpausing the contract

  EIP-2612 Permit
    Should allow permit for approvals
    Should set allowance correctly
    Should increment nonce after permit
    Should revert permit with expired deadline
    Should revert permit with incorrect nonce
    Should revert permit with changed owner
    Should revert permit with changed spender
    Should revert permit with changed amount
    Should revert permit with changed deadline
    Should prevent permit if contract is paused
    Should allow permit if contract was unpaused

  Roles
    Role Assignments
      Should assign ADMIN_ROLE to admin
      Should assign GUARDIAN_ROLE to guardian
      Should assign BANLIST_OPERATOR_ROLE to banlistOperator
    Admin role
      Should be able to grant ADMIN_ROLE
      Should be able to revoke ADMIN_ROLE
      Should be able to renounce ADMIN_ROLE from itself
      Should be able to grant GUARDIAN_ROLE
      Should be able to revoke GUARDIAN_ROLE
      Should be able to grant BANLIST_OPERATOR_ROLE
      Should be able to revoke BANLIST_OPERATOR_ROLE
      Should be able to update EmissionsController address
      Should emit EmissionsControllerUpdated event when EmissionsController address is updated
    Guardian role
      Should be able to pause the contract
      Should be able to unpause the contract
      Should be able to renounce GUARDIAN_ROLE from itself
      Should not be able to renounce GUARDIAN_ROLE from another guardian
      Should not be able to grant ADMIN_ROLE
      Should not be able to revoke ADMIN_ROLE
```

```
        Should not be able to grant BANLIST_OPERATOR_ROLE
        Should not be able to revoke BANLIST_OPERATOR_ROLE
     Banlist operator role
        Should be able to ban an address
        Should emit Ban event when an address is banned
        Should not emit Ban event if an address was already banned
        Should be able to unban an address
        Should emit Unban event when an address is unbanned
        Should not emit Unban event when an address is unbanned
        Should be able to renounce BANLIST_OPERATOR_ROLE from itself
        Should not be able to renounce BANLIST_OPERATOR_ROLE from another banlist operator
        Should not be able to grant ADMIN_ROLE
        Should not be able to revoke ADMIN_ROLE
        Should not be able to grant GUARDIAN_ROLE
        Should not be able to revoke GUARDIAN_ROLE
     Access control
        Should revert when non-admin tries to grant an ADMIN role
        Should revert when non-admin tries to revoke an ADMIN role
        Should revert when non-admin tries to grant a GUARDIAN role
        Should revert when non-admin tries to revoke a GUARDIAN role
        Should revert when non-admin tries to grant a BANLIST_OPERATOR role
        Should revert when non-admin tries to revoke a BANLIST_OPERATOR role
        Should revert when non-admin tries to update EmissionsController address
        Should revert when non-guardian tries to pause the contract
        Should revert when non-guardian tries to unpause the contract
        Should revert when non-banlist operator tries to ban an address
        Should revert when non-banlist operator tries to unban an address


  138 passing (3s)

Done in 9.58s.
```

7.2 Automated Tools

7.2.1 AuditAgent

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.

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 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