Security Review Report
NM-0292 ZERO TECH

NETHERMIND
SECURITY

(Aug 21, 2024)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind Security for ZERO TECH. This review focused on the `ZToken` smart contract that allows for scheduled token mints.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract, and (d) creation of test cases. **Along this document, we report** two points of attention, where two are classified as `Informational` or `Best Practices`. 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 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.

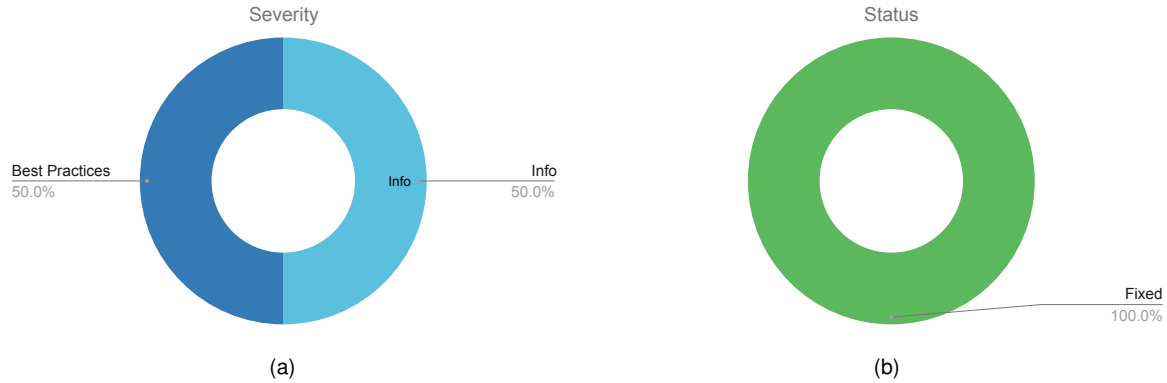

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (1), **Best Practices** (1). **Distribution of status: Fixed** (2), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Aug 21, 2024
Response from Client	Regular responses during audit engagement
Final Report	Aug 22, 2024
Repository	ZToken
Commit (Audit)	9d1905f3910a1b3a14e6bc2ef724c56f74299a96
Commit (Final)	6fdbb0f8baf273afd78b5a33211af8a9f937bcf4
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	DynamicToken.sol	116	98	84.5%	29	243
2	IZToken.sol	9	1	11.1%	7	17
3	IDynamicToken.sol	16	4	25.0%	12	32
4	ZToken.sol	42	24	57.1%	9	75
	Total	**183**	**127**	**69.4%**	**57**	**367**

3 Summary of Issues

	Finding	Severity	Update
1	Incorrect event	Info	Fixed
2	Access control	Best Practices	Fixed

4 System Overview

The reviewed smart contract is the implementation of the `ZToken`. The `ZToken` contract exposes the only two state-chainging restricted functions that: `mint(...)` and `setMintBeneficiary(...)`. The `DynamicToken` abstract contract inherited by `ZToken` contains internal logic and view functions. Other contracts inherited by `ZToken` are standard OpenZeppelin's `AccessControl` and `ERC20`.

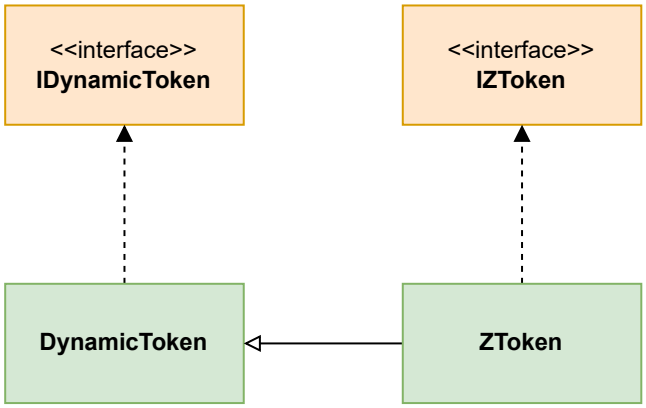

Fig. 2: Structural Diagram of Contracts

The `ZToken` defines two roles: 1) `DEFAULT_ADMIN_ROLE` who can call `setMintBeneficiary(...)` and set other roles and 2) `MINTER_ROLE` who can call `mint(...)`.

At the construction time, the rate schedule is provided which defines the maximum amount of tokens minted each year. The tokens are minted with `mint(...)` which uses the `calculateMintableTokens(...)` to calculate the correct amount of tokens.

The contract defines the following public/external functions:

- `mint(...)` - callable by the `MINTER_ROLE`. Mints new tokens to the beneficiary.
- `setMintBeneficiary(...)` - callable by `DEFAULT_ADMIN_ROLE`. Changes the minted tokens beneficiary.
- `baseSupply(...)` - returns the initial base supply, which is the amount of tokens minted at the construction time and is the base to compute the amount of tokens for each next year.
- `calculateMintableTokens(...)` - main logic calculating the amount that can be minted during each year.
- `yearSinceDeploy(...)` - returns the number of years passed since the deployment of the contract.
- `currentInflationRate(...)` - returns the percentage at the given index stored in `ANNUAL_INFLATION_RATES`.
- `tokensPerYear(...)` - returns the amount of tokens that can be minted at a given year.

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] Incorrect event

File(s): DynamicToken.sol

Description: The DynamicToken inherits OpenZepplin's standard ERC20 token functionality with one modification in the _update(...) function that results in burning tokens if they are transferred to the token contract itself. Such action would cause an emitting event reserved for burn action during the transfer, which may cause issues during the tracking of the token's on-chain activity.

Recommendation(s): Consider clearly documenting the fact that the ZToken would emit a burn event during a transfer to itself.

Status: Fixed

Update from the client: Added logic that emits one more Transfer event before calling super._update() with burn parameters to signal both operations happened during the call: 1. transfer to token address (first Transfer emission), 2. transfer to 0 address - burn (second Transfer emission). Added info about it to NatSpec for the function.

6.2 [Best Practice] Access control

File(s): ZToken.sol

Description: The ZToken contract inherits the AccessControl which allows for roles management. However, the recommended code for managing access is AccessControlDefaultAdminRules which introduces additional security measures for DEFAULT_ADMIN_ROLE like 2-step role transmission.

Recommendation(s): Consider using AccessControlDefaultAdminRules instead of AccessControl.

Status: Fixed

Update from the client: Switched logic to use AccessControlDefaultAdminRules, added another parameter to constructor to set initialDelay and some unit tests.

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 ZERO TECH documentation

The **ZERO TECH** team provided detailed in-line code documentation about each function. Well-written comments helped with the overall understanding of the contract. Additionally, the **ZERO TECH** team was available to address all questions or concerns the **Nethermind Security team** raised.

8 Test Suite Evaluation

8.1 Compilation Output

```
> npx hardhat compile
Generating typings for: 16 artifacts in dir: typechain for target: ethers-v6
Successfully generated 54 typings!
Compiled 14 Solidity files successfully (evm target: paris).
```

8.2 Test Suite Output

```
> npx hardhat test

ZToken Test
Deployment and Access Control
   should mint the provided initial supply to the beneficiary address upon deployment
   should revert if initial supply is passed as 0
   should revert when inflation rates array is empty
   should revert if inflation rates array does NOT start from 0
   should deploy with infation rates array of any length and return final rate correctly
   should revert if any of the addresses passed as 0x0 (44ms)
   should set the given address as the admin and minter of the contract
   should not set other addresses with roles on deployment
   should fail when an address that does not have the MINTER_ROLE tries to mint
   should be able to reassign the minter role to another address
AccessControl Default Admin Rules
   should be able to reassign DEFAULT_ADMIN_ROLE to another address with default delay
   should successfully change admin delay and change to new admin after
   should cancel the admin transfer during the delay period
#calculateMintableTokens()
   should revert when calculating tokens for time that is less than last mint time
   #calculateMintableTokens() should return 0 tokens when no time has passed
Helper math functions
   #currentInflationRate() should return the correct inflation rate for a given year and return final at the end of array
   #yearSinceDeploy() should return the correct year since deploy
   #yearSinceDeploy() should revert if time passed is less than deploy time
   #tokensPerYear() should return the correct tokens per year for a given year
Minting Scenarios on the same state. One after another.
   [1st year] middle of the first year
   [3rd year] 2 years + 260825 sec after
   [3rd + 4th year] end of 3rd year and 3 times in one 4th year
   [12th year] after 8 years and 8919854 seconds where the inflation plateaus
   [33rd year] during the inflation plateau
Burn on Transfer to Token Address
   should burn token upon transfer to token address
   should NOT burn tokens if transferred to any regular address
   should emit TWO Transfer events upon transfer to token address
#setMintBeneficiary()
   #setMintBeneficiary() should set the new address correctly
   #setMintBeneficiary() should revert if called by non-admin
   #setMintBeneficiary() should revert if called with 0x0 address
Minting scenarios on clean state.
 should mint the correct amount of tokens when minted every second (38ms)
 #calculateMintableTokens() should return correct years amount of tokens, increased each year
 should mint the correct amount of tokens per year exactly
 should correctly account totalSupply and mintable tokens when burning by transfer to token contract
34 passing (1s)
```

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 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

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.