Security Review Report
NM-0214-ORA-ERC7641-Token

NETHERMIND
SECURITY

(Jun 12, 2024)

Contents

1 Executive Summary 2

2 Audited Files 3

3 Summary of Issues 3

4 System Overview 4

5 Token design discussion 5

6 Risk Rating Methodology 6

7 Issues 7
- 7.1 [Low] Lack of input validation . . . 7
- 7.2 [Info] Possible division by zero in `claimableRevenue(...)` function . . . 7

8 Documentation Evaluation 8

9 Test Suite Evaluation 9
- 9.1 Compilation Output . . . 9
- 9.2 Tests Output . . . 9

10 About Nethermind 10

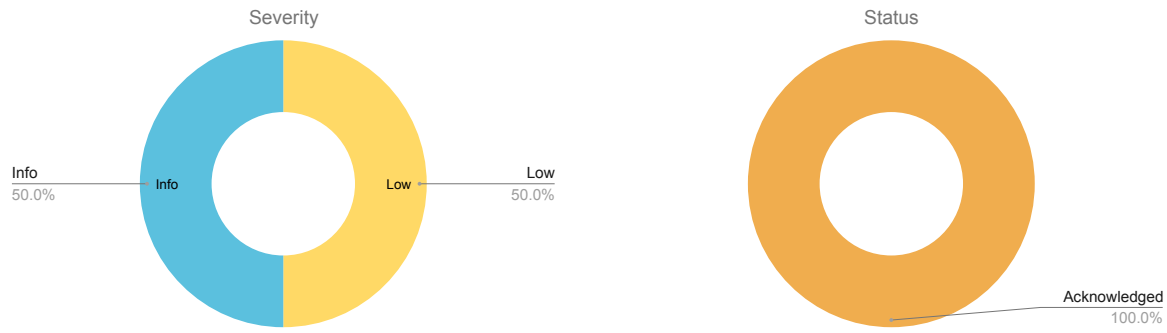
1 Executive Summary

This document outlines the security review conducted by Nethermind for the new token standard `ERC7641`. Holders of this token can claim the revenue generated by the AI model. This solution gives an opportunity for open-source AI models to compete with products created by closed-source, for-profit companies since contributors can be rewarded by holding the token and claiming revenue.

The audited code comprises 88 lines of code. The `ORA` team has provided documentation explaining the behavior of the token design and distribution. Aside from the provided documentation, the `ORA` and `Nethermind` teams have actively communicated to clarify any remaining questions about the expected behavior of the protocol.

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 `Low` 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 presents the system overview. Section 5 discusses the token design. Section 6 discusses the risk rating methodology adopted for this audit. Section 7 details the issues. Section 8 discusses the documentation provided by the client for this audit. Section 9 presents the compilation, tests, and automated tests. Section 10 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 (0), **High** (0), **Medium** (0), **Low** (1), **Undetermined** (0), **Informational** (1), **Best Practices** (0). **(b) Distribution of status: Fixed** (0), **Acknowledged** (2), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Apr 16, 2024
Final Report	Jun 12, 2024
Methods	Manual Review, Automated Analysis
Repository	IMO-token-contract
Commit Hash	54008c34b27d8f6ff87f052e83cd83e7e00b4dde
Final Commit Hash	0dbbb0f433d6b3d4bedc485234008f9434bab403
Documentation	-
Documentation Assessment	High
Test Suite Assessment	Medium

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	contracts/ERC7641.sol	79	63	79.7%	22	164
2	contracts/IERC7641.sol	9	28	311.1%	6	43
	Total	**88**	**91**	**103.4%**	**28**	**207**

3 Summary of Issues

	Finding	Severity	Update
1	Lack of input validation	Low	Acknowledged
2	Possible division by zero in `claimableRevenue(...)` function	Info	Acknowledged

4 System Overview

The **ERC7641** contract contains the revenue share logic. It is a token based on `ERC20Snapshot` standard. Snapshot feature allows anyone to call `snapshot(...)` function, which registers the current state of user balance. They can be taken with a predefined frequency. Based on the balance held during a snapshot, the revenue is distributed among the holders. Tokens can also be burned and transferred.

The contract uses the following storage variables:

```
uint256 constant public SNAPSHOT_CLAIMABLE_NUMBER = 2;
uint256 public lastSnapshotBlock;
uint256 immutable public percentClaimable;
uint256 immutable public snapshotInterval;
mapping (uint256 => uint256) private _claimableAtSnapshot;
mapping (uint256 => uint256) private _claimedAtSnapshot;
mapping (uint256 => mapping (address => bool)) private _hasClaimedAtSnapshot;
uint256 private _redeemPool;
uint256 private _redeemed;
```

The contract exposes the following public or external functions:

Constructor function

```
constructor(...)
```

Allows for receiving ETH

```
receive(...) external payable
```

Calculates the amount of ETH claimable by a token holder at certain snapshot

```
function claimableRevenue(...) public view
```

Claims revenue token based on the token balance at certain snapshot

```
function claim(...) public
```

Claim by a list of snapshot ids

```
function claimBatch(...) external
```

Records the deposited ETH amount at the time of the snapshot

```
function snapshot(...) external
```

Returns the amount of ETH redeemable by a token holder upon burn

```
function redeemableOnBurn(...) external view
```

Burns tokens and redeem the corresponding amount of revenue token

```
function burn(...) external
```

5 Token design discussion

The ERC7641 is designed to share the revenue generated by an open-sourced AI model between the token holders. The revenue is distributed proportionally to the user balance at a given snapshot. Snapshots are taken between defined periods, e.g., three months. Each snapshot captures the amount of new revenue (`ETH`) along with the amount of tokens held by users. Token holders can claim the amount of `ETH` from the contract proportionate to the number of tokens held when the snapshot was taken. The revenue can be claimed for the current and previous snapshots, i.e., two last snapshots. When the new snapshot is created, the unclaimed revenue from the last snapshot is treated as new revenue for the new snapshot. The token holder is incentivized to hold the token and periodically claim the revenue after creating the new snapshot.

The `ERC7641` token is initially distributed through the `IMO` with a predefined price, and later, it can be exchanged as any other `ERC20` token. The price is expected to change significantly around the time of snapshot creation. The new snapshot computes the amount of new revenue available for claim. Therefore, since the token holders can claim `ETH`, the value of tokens increases. Additionally, any unclaimed revenue from the last snapshot moves to the new snapshot, increasing the amount of claimable `ETH`. Those factors may incentivize users to buy tokens before the snapshot is taken and sell them immediately. In extreme cases, the actor may take a flash loan, buy many available tokens, call the snapshot function, and sell the tokens, all in one transaction. This way, those actors can claim the revenue and are not exposed to the potential risks of holding the token. This may cause inflation of the token price right before the snapshot since buying pressure is high due to upcoming new claimable revenue. After the snapshot, the price may drop due to the selling pressure of tokens worth less due to a lack of claimable revenue. The change in price would depend on the amount of new revenue and the amount of tokens being exchanged compared to tokens held by the same addresses. We recommend discussing the described possibility of price changes and evaluating their impact on token utility.

6 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.

7 Issues

7.1 [Low] Lack of input validation

File(s): ERC7641.sol

Description: The `percentClaimable` variable is bound by `100` in the constructor, yet there's no defined minimum value. Setting it to `0` would prevent claiming rewards.

Recommendation(s): Consider implementing checks to validate the values assigned to important variables.

Status: Acknowledged

Update from the client: Not going to implement. When `percentClaimable==0`, it means 0% claimable on rev share and 100% redeemable on burn, which is not an invalid input, just a special use case.

7.2 [Info] Possible division by zero in `claimableRevenue(...)` function

File(s): ERC7641.sol

Description: The `claimableRevenue(...)` function calculates the amount a user can claim within a specific snapshot. However, the current implementation lacks handling for the division by zero case when dividing by the `totalSupply`.

```
function claimableRevenue(...) public view returns (uint256) {
    // ...
    uint256 totalSupply = totalSupplyAt(snapshotId);
    uint256 ethClaimable = _claimableAtSnapshot[snapshotId];
    // @audit possible division by zero
    return _hasClaimedAtSnapshot[snapshotId][account] ? 0 : balance * ethClaimable / totalSupply;
}
```

Recommendation(s): Consider adding a check to handle this case separately and prevent the division operation if `totalSupply` is zero.

Status: Acknowledged

Update from the client: Not going to implement (for gas efficiency). The reported issue only trigger when the `totalSupply == 0`. The total supply is decreasing only since there's only _burn but no _mint in the token contract.

8 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 ORA documentation

Throughout the review process, the ORA team offered assistance during meetings to clarify the protocol and address any questions or concerns raised by the Nethermind Security team. However, the reviewed code lacks NatSpec documentation and inline comments explaining the implementation details.

9 Test Suite Evaluation

9.1 Compilation Output

```
npx hardhat compile
Compiled 11 Solidity files successfully (evm target: paris).
```

9.2 Tests Output

```
npx hardhat test

  ERC7641
    Deployment
       Should set the right name
       Should set the right symbol
       Should set the right total supply
       Should set the right last snapshot block
       Should set the right percent claimable
       Should set the right snapshot interval
       Should assign the total supply to the owner
    Deposit
       Should deposit ETH to the contract
    Snapshot
       Should not snapshot if 648,000 blocks have not passed
       Should snapshot if > 648,000 blocks have passed
    Burn
       Should burn tokens
       Should burn tokens and receive ETH
       Should snapshot and burn tokens
       Should snapshot, deposit, and burn
    Claim
       Should not claim if no snapshot has been taken
       Should claim after snapshot
       Should claim after snapshot and deposit
       Should claim correctly after snapshot with two holders (41ms)
       Should claim multiple snapshots correctly (44ms)
       Should not claim if not most recent three snapshots (67ms)
    Mixed operations
       deposit -> snapshot -> deposit -> burn -> deposit -> burn -> snapshot -> claim -> burn (83ms)


  21 passing (2s)
```

10 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.