# Security Review Report
# NM-0233 Summon

NETHERMIND
SECURITY

(May 27, 2024)

# Contents

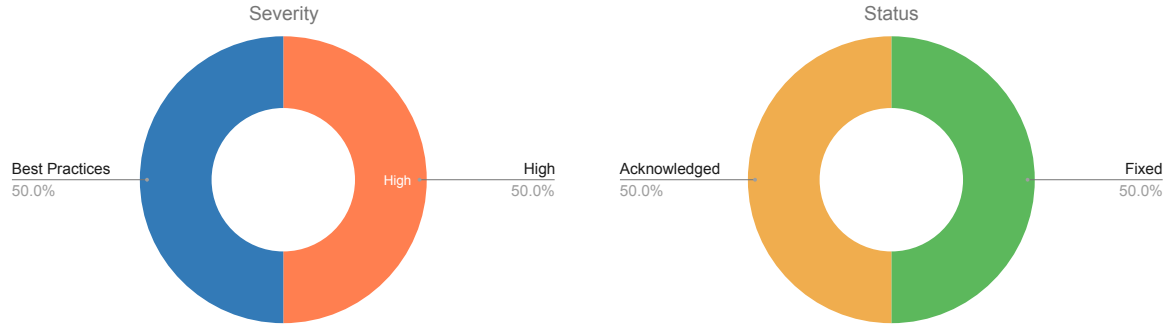
# 1 Executive Summary

This document outlines the security review conducted by Nethermind Security for Summon. The **Summon** system allows gamification of the participation and commitment of users towards a community, using playful reward mechanisms in the form of tokens to obtain the engagement of its members.

This code review focuses on the `AvatarBound.sol` contract, ERC-721 token representing the user's avatar. This contract was previously audited by **Nethermind security**, and this audit mainly focuses on newly introduced changes in the code. This report does not include findings reported previously for this contract.

**The audited code comprises** 371 lines of code in Solidity and introduced changes represent 50 lines of code in Solidity. The audit was performed using: (a) manual analysis of the codebase, (b) simulation of the smart contracts. **Along this document, we report** 2 points of attention, where one is classified as `High`, one is classified as `Best Practices`. 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 risk rating methodology adopted for this audit. 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.




(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** (1), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (0), **Best Practices** (1). **(b) Distribution of status: Fixed** (1), **Partially Fixed** (0), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

### Summary of the Audit

| | |
|---|---|
| **Audit Type** | Security Review |
| **Initial Report** | May 27, 2024 |
| **Final Report** | May 27, 2024 |
| **Methods** | Manual Review |
| **Repository** | achievo-contracts |
| **Commit Hash** | 1943ba3b45915b0bed21f5330094666328607800 |
| **Final Commit Hash** | 7d06cc286fdb34e715daa44e1e76b3740f764777 |
| **Documentation** | Not Available |
| **Documentation Assessment** | Low |
| **Test Suite Assessment** | Medium |

## 2 Audited Files

| | Contract | LoC | Comments | Ratio | Blank | Total |
|---|---|---|---|---|---|---|
| 1 | AvatarBound.sol | 371 | 20 | 5% | 61 | 452 |
| | **Total** | **371** | **20** | **5%** | **61** | **452** |

## 3 Summary of Issues

| | Finding | Severity | Update |
|---|---|---|---|
| 1 | Signature replay attack across different chains | High | Fixed |
| 2 | Lack of input validation might lead to unexpected behaviour | Best Practices | Acknowledged |

# 4 System Overview

The whole **Summon** system is part of the game where the user's assets are represented in the form of ERC-721 tokens. An example of such a token is the audited AvatarBound contract, which serves as the user's avatar. An avatar is soulbound to the user's address, making it non-transferable to other users. The avatar features various skins based on the ID chosen by the user during the minting process. Each skin is then uniquely visualised on the front end.

## 4.1 Design of AvatarBound contract

The contract uses the following storage variables:

```
uint256[50] private __gap;
bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
uint256 private _tokenIdCounter;
uint256 public _baseSkinCounter;
uint256 private _specialItemId;
uint256 private defaultItemId;
string public baseTokenURI;
string public contractURI;
string public revealURI;
address public gatingNFTAddress;
address public itemsNFTAddress;
bool private mintNftGatingEnabled;
bool private mintNftWithoutGatingEnabled;
bool private mintRandomItemEnabled;
bool private mintSpecialItemEnabled;
bool private mintDefaultItemEnabled;
mapping(uint256 => string) public baseSkins;
```

The contract exposes following public functions:

Mints avatar token and additional item tokens (represented as ERC-1155 tokens)

```
function mintAvatar(...) public;
```

Mints avatar token for the owner of gating token

```
function mintAvatarNftGating(...) public;
```

Role-restricted avatar token minting

```
function adminMint(...) public;
function batchMint(...) public;
```

Role-restricted signature utilization

```
function adminVerifySignature(...) public;
```

Role-restricted pausing functions

```
function pause(...) public;
function unpause(...) public;
```

Role restricted setters

```
function batchSetTokenURI(...) public;
function setContractURI(...) public;
function setTokenURI(...) public;
function setBaseURI(...) public;
function setRevealURI(...) public;
function setBaseSkin(...) public;
function setMintRandomItemEnabled(...) public;
function setMintDefaultItemEnabled(...) public;
function setItemsNFTAddress(...) public;
function setNftGatingAddress(...) public;
function setSpecialItemId(...) public;
function setDefaultItemId(...) public;
function setMintNftGatingEnabled(...) public;
function setMintSpecialItemEnabled(...) public;
function setMintNftWithoutGatingEnabled(...) public;
function addWhitelistSigner(...) external;
function removeWhitelistSigner(...) external;
```

View functions

```
function getAllBaseSkins(...) public;
function tokenURI(...) public;
function decodeData(...) public;
function getSpecialId(...) public;
function getDefaultItem(...) public;
```

The contract inherits the following OpenZeppelin contracts:

```
    ERC721URIStorage,
    ERC721Enumerable,
    AccessControl,
    Pausable,
    ReentrancyGuard
```

And the following custom contracts:

```
Achievo721Soulbound,
ERCWhitelistSignature
```

The contract makes external calls to the following contracts:

```
   IOpenMint,
   IItemBound
```

# 5 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.

Other factors are also considered when defining the likelihood of a finding. 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.

Other factors are also considered when defining the impact of a finding. 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 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 [High] Signature replay attack across different chains

**File(s)**: AvatarBound.sol

**Description**: A user can mint Avatar NFT by two functions: `mintAvatar(...)` and `mintAvatarNftGating(...)`. Both of these functions require a signature created by a whitelisted signer (admin of the protocol).

The signature is verified inside a function `_verifySignature()`:

```
function _verifySignature(
    address to,
    uint256 nonce,
    bytes calldata data,
    bytes calldata signature
) internal virtual returns (bool) {
    if (usedSignatures[signature]) revert("AlreadyUsedSignature");

    address signer = _recoverAddress(to, nonce, data, signature);
    if (whitelistSigners[signer]) {
        usedSignatures[signature] = true;
        return true;
    } else {
        return false;
    }
}
```

However, this function is vulnerable to the following attacks:

- Cross-chain replay attack: The function lacks a check for `chainID` (the blockchain's ID) to verify the chain for which the signature was generated;
- Cross-contract replay attack: The function lacks a check for `address(this)` to confirm that the signature is intended for the current contract;

Since the intention is to implement the protocol across various blockchains and `ERCWhitelistSignature` contract is inherited by multiple contracts in the protocol, it becomes paramount to incorporate metadata within the signature to prevent these types of attacks between contract instances across distinct chains or within the same chain.

**Recommendation(s)**: Consider using a chain-specific signing scheme such as EIP-155, which includes the chain ID in the signed message.

**Status**: Fixed

**Update from the client**: Add the changes here - 7d06cc286fdb34e715daa44e1e76b3740f764777

## 6.2 [Best Practices] Lack of input validation might lead to unexpected behaviour

**File(s)**: AvatarBound.sol

**Description**: The `getAllBaseSkins()` function is designed to return an array of `BaseSkinResponse` structs. This array is created by iterating over the `_baseSkinCounter` values with the assumption that the `skinIds` will be set in order, as shown below:

```
function getAllBaseSkins() public view returns (BaseSkinResponse[] memory) {
    BaseSkinResponse[] memory allBaseSkins = new BaseSkinResponse[](_baseSkinCounter);
    for (uint256 i = 0; i < _baseSkinCounter; i++) {
        BaseSkinResponse memory avatarBaseSkinResponse = BaseSkinResponse({
            baseSkinId: i, // @audit: This assumes that skinIds will be equal to i
            tokenUri: baseSkins[i]
        });
        allBaseSkins[i] = avatarBaseSkinResponse;
    }
    return allBaseSkins;
}
```

However, the `setBaseSkin()` function does not enforce the order of the `skinIds`, allowing them to be set in any order. This deviation from the expected order will disrupt the logic of the `getAllBaseSkins()` function, which relies on the `skinIds` being in sequence. Fixing unordered `skinIds` in the `setBaseSkin(...)` function could be challenging because it would involve filling the gaps from zero to the highest `skinId` value, ensuring that the sequence is maintained for the proper functioning of `getAllBaseSkins()`.

Notably, the below functions are also missing input validation:

- `constructor(...)`where all the passed addresses could be set to the zero address, and all mint operations could be turned off;
- `setRevealURI(...)` where `revealURI` could be set to an empty string;
- `setItemsNFTAddress(...)` where `itemsNFTAddress` could be set to the zero address;
- `setNftGatingAddress(...)` where `nftGatingAddress` could be set to the zero address;

**Recommendation(s)**: Consider applying proper input validation to avoid potential issues.

**Status**: Acknowledged

**Update from the client**:

# 7 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 Summon documentation**

No documentation has been provided. However, the **Summon** team was available to address any inquiries or concerns raised by the Nethermind Security team and explained all introduced changes to our team.

# 8 Test Suite Evaluation

## 8.1 Compilation Output

```
> forge compile --force
[] Compiling...
[] Compiling 183 files with 0.8.17
[] Solc 0.8.17 finished in 390.85s
Compiler run successful with warnings:
// Some warnings here
```

## 8.2 Tests Output

```
> forge test

Running 11 tests for test/AvatarBound.t.sol:AvatarBoundTest
[PASS] testAdminMint() (gas: 202186)
[PASS] testCompoundURI() (gas: 248025)
[PASS] testFailUnauthorizedTransfer() (gas: 34978)
[PASS] testMintAvatarNftGating() (gas: 1127132)
[PASS] testMintAvatarWithoutNftGating() (gas: 1062592)
[PASS] testPauseUnpause() (gas: 118472)
[PASS] testSetBaseSkin() (gas: 49597)
[PASS] testSetBaseURI() (gas: 34773)
[PASS] testSetContractURI() (gas: 26559)
[PASS] testSetTokenURI() (gas: 229079)
[PASS] testgetAllItems() (gas: 89759)
Test result: ok. 11 passed; 0 failed; 0 skipped; finished in 11.43ms

Running 11 tests for test/AvatarBoundV1.t.sol:AvatarBoundV1Test
[PASS] testAdminMint() (gas: 207470)
[PASS] testCompoundURI() (gas: 254862)
[PASS] testFailUnauthorizedTransfer() (gas: 43400)
[PASS] testMintAvatarNftGating() (gas: 1142056)
[PASS] testMintAvatarWithoutNftGating() (gas: 1077456)
[PASS] testPauseUnpause() (gas: 125518)
[PASS] testSetBaseSkin() (gas: 54841)
[PASS] testSetBaseURI() (gas: 40409)
[PASS] testSetContractURI() (gas: 31847)
[PASS] testSetTokenURI() (gas: 235534)
[PASS] testgetAllItems() (gas: 95422)
Test result: ok. 11 passed; 0 failed; 0 skipped; finished in 11.15ms

Ran 2 test suites: 22 tests passed, 0 failed, 0 skipped (22 total tests)
```

# 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 procehttps://www.overleaf.com/project/65c0e737f41a29601bda5c48ss, 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.