



# AUDIT REPORT




---

August 2025

For



# Table of Content

Executive Summary	03
Number of Security Issues per Severity	05
Summary of Issues	06
Checked Vulnerabilities	07
Techniques and Methods	09
Types of Severity	11
Types of Issues	12
Severity Matrix	13
 <b>Medium Severity Issues</b>	14
1. Add a restriction for checking <code>_mintPrice</code> .	14
2. Centralization concern	14
 <b>Low Severity Issues</b>	15
3. Floating pragma	15
4. Redundant inheritance	15
 <b>Informational Issues</b>	16
5. Events can be added according to the requirement	16
6. Comments can be removed before deployment	16
Functional Tests	17
Automated Tests	17
Threat Model	18
Closing Summary & Disclaimer	19



# Executive Summary

<b>Project Name</b>	ByteNFT
<b>Protocol Type</b>	NFT
<b>Project URL</b>	<a href="https://www.bytenft.xyz/">https://www.bytenft.xyz/</a>
<b>Overview</b>	<p>ByteNFT is an NFT contract with the functionality to mint and burn tokens. The contract allows the functionality to grant and revoke the admin role.</p> <p>The admin in this case is a trusted address that can mint or burn according to the requirement. The mint function also allows receiving ERC20 tokens as a price for minting the NFT.</p>
<b>Audit Scope</b>	The scope of this Audit was to analyze the ByteNFT Smart Contracts for quality, security, and correctness.
<b>Source Code link</b>	<a href="https://github.com/Cooraez12/bytenft-smart-contract">https://github.com/Cooraez12/bytenft-smart-contract</a>
<b>Branch</b>	main
<b>Contracts in Scope</b>	ByteNFT.sol
<b>Commit Hash</b>	<a href="#">58b8b997d1a92c6a5b5608de6e80c933989b716d</a>
<b>Language</b>	Solidity
<b>Blockchain</b>	Base
<b>Method</b>	Manual Analysis, Functional Testing, Automated Testing
<b>Review 1</b>	5th August 2025 - 11th August 2025
<b>Updated Code Received</b>	14th August 2025
<b>Review 2</b>	18th August 2025
<b>Fixed In</b>	<a href="#">4655a1de0e0a17af9531967f4e6e23f01dfd4091</a>

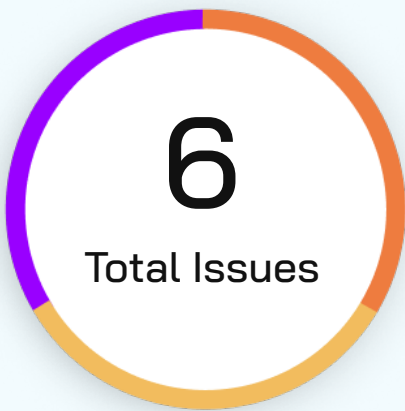


**Verify the Authenticity of Report on QuillAudits Leaderboard:**

<https://www.quillaudits.com/leaderboard>



# Number of Issues per Severity



Critical	0 (0.0%)
High	0 (0.0%)
Medium	2 (33.3%)
Low	2 (33.3%)
Informational	2 (33.3%)

		Severity				
		Critical	High	Medium	Low	Informational
Issues	Open	0	0	0	0	0
	Acknowledged	0	0	0	0	0
	Partially Resolved	0	0	0	0	0
	Resolved	0	0	2	2	2



# Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Add a restriction for checking _mintPrice.	Medium	Resolved
2	Centralization concern	Medium	Resolved
3	Floating pragma	Low	Resolved
4	Redundant inheritance	Low	Resolved
5	Events can be added according to the requirement	Informational	Resolved
6	Comments can be removed before deployment	Informational	Resolved



# Checked Vulnerabilities

✓ Access Management

✓ Arbitrary write to storage

✓ Centralization of control

✓ Ether theft

✓ Improper or missing events

✓ Logical issues and flaws

✓ Arithmetic Computations  
Correctness

✓ Race conditions/front running

✓ SWC Registry

✓ Re-entrancy

✓ Timestamp Dependence

✓ Gas Limit and Loops

✓ Exception Disorder

✓ Gasless Send

✓ Use of tx.origin

✓ Malicious libraries

✓ Compiler version not fixed

✓ Address hardcoded

✓ Divide before multiply

✓ Integer overflow/underflow

✓ ERC's conformance

✓ Dangerous strict equalities

✓ Tautology or contradiction

✓ Return values of low-level calls



✓ Missing Zero Address Validation

✓ Private modifier

✓ Revert/require functions

✓ Multiple Sends

✓ Using suicide

✓ Using delegatecall

✓ Upgradeable safety

✓ Using throw

✓ Using inline assembly

✓ Style guide violation

✓ Unsafe type inference

✓ Implicit visibility level



# Techniques and Methods

**Throughout the audit of smart contracts, care was taken to ensure:**

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

**The following techniques, methods, and tools were used to review all the smart contracts:**

## Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

## Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.



### Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

### Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

### Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.



# Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

## ■ **Critical: Immediate and Catastrophic Impact**

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

## ■ **High (H): Significant Risk of Major Loss or Compromise**

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

## ■ **Medium (M): Potential for Moderate Harm Under Specific Circumstances**

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

## ■ **Low (L): Minor Imperfections with Limited Repercussions**

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

## ■ **Informational (I): Opportunities for Improvement, Not Immediate Risks**

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.



# Types of Issues

## Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

## Resolved

These are the issues identified in the initial audit and have been successfully fixed.

## Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

## Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

# Severity Matrix

		Impact		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

## Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

## Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- **Medium** - only a conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.



# Medium Severity Issues

## Add a restriction for checking `_mintPrice`.

**Resolved**

### Path

ByteNFT.sol

### Path

`mint()`

### Description

According to the business logic, the intended behavior is not to allow minting when `_mintPrice` is passed as 0. This restriction will ensure that all minting transactions require a `_mintPrice` greater than zero.

But currently the `mint()` allows minting when the `_mintPrice` is passed as 0.

### Recommendation

In case it is required to strictly revert the transaction when the passed `_mintPrice` is 0, then a require statement to check that the `_mintPrice` is greater than 0 can be added.

It would look like this:

```
require(_mintPrice > 0, "Price should be greater than 0");
```

## Centralization concern

**Resolved**

### Path

ByteNFT.sol

### Path

`burn()`

### Description

The contract allows the admin or owner to burn the NFT without the current holder's/owner's approval.

### Recommendation

Consider using multisig to mitigate any centralization concerns.

### Audit Team's Comment

Fixed by making the `burn()` token holder specific.



# Low Severity Issues

## Floating pragma

**Resolved**

### Path

ByteNFT.sol

### Description

Contract is using version `^0.8.27` with a floating pragma instead of locking to a specific version. Floating pragmas allow the contract to be compiled with any version greater than or equal to the specified version for that major version. If the contract wasn't thoroughly tested with that version, this can introduce possible bugs.

### Recommendation

Consider using a fixed solidity version whenever possible.

## Redundant inheritance

**Resolved**

### Path

ByteNFT.sol

### Path

`burn()`

### Description

ByteNFT inherits `ERC721BurnableUpgradeable`, but `ByteNFT.burn()` does not use the `ERC721BurnableUpgradeable.burn()` even after overriding. In this case, the `ERC721BurnableUpgradeable` is unused, and the inheritance of the `ERC721BurnableUpgradeable` is redundant.

### Recommendation

Consider removing the inherited `ERC721BurnableUpgradeable`



# Informational Issues

## Events can be added according to the requirement

**Resolved**

### Path

ByteNFT.sol

### Description

Currently, there are no events in the ByteNFT smart contract functionality. Events help monitor updates off-chain. Events can be added according to the requirements.

### Recommendation

Add events based on the requirement.

## Comments can be removed before deployment

**Resolved**

### Path

ByteNFT.sol

### Path

`burn()` , `tokenURI()`

### Description

`burn()` , `tokenURI()` functions have comments regarding the **CORRECTED** code. These comments can be removed before deployment.

### Recommendation

Consider removing the comments related to the **CORRECTED** code.





# Functional Tests

**Some of the tests performed are mentioned below:**

- ✓ Admin or owner should be able to mint the NFT.
- ✓ Admin or owner should be able to burn the NFT.
- ✓ Only the owner should be able to grant the admin role.
- ✓ Only the owner should be able to revoke the admin role.
- ✓ Only the owner should be able to update the treasury address.
- ✓ Only the owner should be able to update the payment token address.
- ✓ Should be able to transfer the owned NFT.
- ✓ Should be able to transfer the approved NFT.
- ✓ Should be able to set approval for all tokens.

# Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.



# Threat Model

Contract	Function	Threats
ByteNFT	initialize	Failure in address initialization
	mint	Failure in the token transfer. Failure to update token URI, Access control failure
	burn	Failure to burn the token and reset the mapping values, Access control failure
	grantAdminRole	Failure to grant a role, Access control failure
	revokeAdminRole	Failure to revoke a role, Access control failure
	updateTreasuryAddress	Failure to update a treasury address, Access control failure
	updatePaymentTokenAddress	Failure to update a payment token address, Access control failure



# Closing Summary

In this report, we have considered the security of ByteNFT. We performed our audit according to the procedure described above.

Issues of Medium, Low, and Informational severity were found. ByteNFT team resolved all the issues mentioned.

# Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.



# About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers.

With seven years of expertise, we've secured over 1400 projects globally, averting over \$3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

**7+**

Years of Expertise

**1M+**

Lines of Code Audited

**50+**

Chains Supported

**1400+**

Projects Secured

Follow Our Journey



# AUDIT REPORT

---

August 2025

For



Canada, India, Singapore, UAE, UK

[www.quillaudits.com](http://www.quillaudits.com)

[audits@quillaudits.com](mailto:audits@quillaudits.com)