Alethea AI | Smart Contract Audit | NFT Factory

tags: `alethea , audit`

Latest revision: November 21st
Miguel Palhas **mpalhas@gmail.com** (mailto:mpalhas@gmail.com)

Table of Contents

Overview

A permissioned NFT minting factory in Solidity

Dates

- **August 14th**: Start date
- **August 15th**: First report

Process

This document, and all suggestions detailed here, is meant to be scrutinized and
discussed between both parties, in an ongoing collaborative process after the
first report delivery. Each suggestion may not be needed

due to contextual
reasons, or limited understanding of external scope by the auditor.

Coverage

The following repository was considered in-scope for the review:
https://github.com/AletheaAI/alethea-contracts
(https://github.com/AletheaAI/alethea-contracts)
In particular, this audit focuses solely on the `NFTFactory.sol` contract and respective dependencies and tests. The revision used was `d156a57` .

Areas of Concern

The investigation focused on the following:

- Looking for attack vectors that could impact the intended behaviour of the
  smart contract
- Checking test coverage, particularly for potentially dangerous scenarios and
  edge-cases
- Interaction with 3rd party contracts
- Use of solidity best practices
- Permissions and roles after deploy script is executed

Findings

Each issue has an assigned severity:

- **Informational**
- **Minor**
  issues are subjective in nature. They are typically suggestions around best
  practices or readability. Code maintainers should use their judgement as to
  whether to address such issues.
- **Gas**
  issues are related to gas optimizations.
- **Medium**
  issues are objective in nature but are not security vulnerabilities. These

should be addressed unless there is a clear reason not to.

- **High**
  issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

Progress

This table helps tracks the auditor's progress through each section of the codebase, and is not indicative of final results.

`NFTFactory.sol`

identifier	progress
usedNonces	✅
DOMAIN_SEPARATOR	✅
DOMAIN_TYPEHASH	✅
MINT_WITH_AUTHORIZATION_TYPEHASH	✅
FEATURE_MINTING_WITH_AUTH	✅
ROLE_FACTORY_MINTER	✅
constructor	✅
mint	✅
__mint	✅
mintWithAuthorization	✅
authorizationState	✅
cancelAuthorization	✅
cancelAuthorization	✅
__deriveSigner	✅
__useNonce	✅

Findings Summary

Findings are listed in chronological order of discovery.

title	severity	status
1. Consider using OpenZeppelin's `EIP712` library	**Informational**	

Detailed Findings

1. Consider using OpenZeppelin's `EIP712` library

Severity: Informational

`NFTFactory`, much like previous contracts in the same codebase, relies on `EIP712` for signed approvals. Instead of repeatedly implementing the same logic across these contracts, the developers should consider relying on a battle-tested implementation such as the one from **OpenZeppelin** (https://docs.openzeppelin.com/contracts/3.x/api/drafts#EIP712)
One notable difference between OpenZeppelin's EIP712 and the current implementation is how the domain separator is built:

```
keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes("NFTFactoryV1")), block.cha
```

Notice that the string `"NFTFactoryV1"` is used, even though the contract is actually called `NFTFactory`. `EIP712.sol` handles this by including an additional `version` field. This could make the implementation more in line with known standards.