QuillAudits

Audit Report
April, 2024

For

NFTs2Me

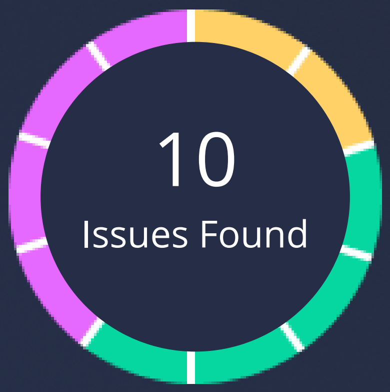
Table of Content

Table of Content

Executive Summary

Project Name	NFTs2Me
Overview	NFTs2Me contains multiple contracts. The main functionality in the audit was the factory and collection contract. N2MFactory allows anyone to create the NFT collection while initializing the created collection. The owner can add the new implementation for the implementation type. The N2MFactory contract allows creating collections apart from the specified implementation types.
Timeline	26th February 2024 - 18th March 2024
Updated Code Received	18th March 2024
Second Review	18th March 2024 - 19th March 2024
Third Review	1st April 2024 - 16th April 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	This audit aimed to analyze the NFTs2Me Codebase for quality, security, and correctness. 1. N2MFactory.sol 2. N2MERC721A.sol 3. ConsecutiveMinting.sol 4. Common.sol
Source Code	*https://github.com/nfts2me/nfts2me-contracts-audit/tree/main*
Branch	Main
Commit Hash	341261169350bb139ff87f19297ec198f41725d3
Fixed In	*a47bc2b17045f4e99a03f936390a3aca30402abe*

Number of Security Issues per Severity



🟥 High	🟨 Medium
🟩 Low	🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	**1**	**3**	**4**
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	**1**	**1**	0

Checked Vulnerabilities

✓ Re-entrancy	✓ Transfer forwards all gas
✓ Timestamp Dependence	✓ ERC20 API violation
✓ Gas Limit and Loops	✓ Compiler version not fixed
✓ DoS with Block Gas Limit	✓ Redundant fallback function
✓ Transaction-Ordering Dependence	✓ Send instead of transfer
✓ Use of tx.origin	✓ Style guide violation
✓ Exception disorder	✓ Unchecked external call
✓ Gasless send	✓ Unchecked math
✓ Balance equality	✓ Unsafe type inference
✓ Byte array	✓ 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-20 token 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

Hardhat, Foundry.

Types of Severity

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

High Severity Issues

A high severity issue or vulnerability means your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

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

Resolved

These issues were identified in the initial audit and 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.

Medium Severity Issues

1. Possibility of Denial Of Service

Path

Common.sol

Function

withdraw(), withdrawERC20

Description

_revenueInfo is an array that store all the Revenue Address based on their percentage of share.

Whenever withdraw/withdrawERC20 is called, it iterates over the length of the array and send them the amount which was defined during initialization.

The problem with this approach is that, there's no upper bound for the number of addresses that should be inside the array as long as the total sum is 100 percent.

There exist an edge case, where this array becomes so large that looping over it would exceed the block gas limit resulting in a state of DOS.

https://github.com/nfts2me/contracts/blob/6e92ed4aa28c8ae3c3b02a040ccb6e006fbb0822/Common.sol#L392

https://github.com/nfts2me/contracts/blob/6e92ed4aa28c8ae3c3b02a040ccb6e006fbb0822/Common.sol#L423

Recommendation

Consider applying an upper bound on the length of the array.

Status

Resolved

2. Use safeTransfer instead of Transfer

Path

Common.sol

Description

Tokens not compliant with the ERC20 specification could return ***false*** from the transfer function call to indicate the ***transfer*** fails, while the calling contract would not notice the failure if the return value is not checked.

https://github.com/nfts2me/contracts/blob/6e92ed4aa28c8ae3c3b02a040ccb6e006fbb0822/Common.sol#L396

https://github.com/nfts2me/contracts/blob/6e92ed4aa28c8ae3c3b02a040ccb6e006fbb0822/Common.sol#L367

Recommendation

Consider using safeTransfer or add a require statement to assert return value.

Status

Acknowledged

NFTs2Me Team's Comment

We did this on purpose. Doing the recommended task by QuillAudits would actually create a serious bug on the contracts. We intentionally use transfer instead of safeTransfer to avoid blocking the contract. Let's say that we have 3 different wallets on the array. If one of them was wrongly set, and fails, it would block the contract from withdrawing the ERC-20 funds when using safeTransfer instead of Transfer (as done now). However, if we use Transfer instead of SafeTransfer, at least the non-blocking wallets that don't fail when transfering the ERC-20 will get the funds, and the contract won't be blocked. That's the reason we are using try-catch instead of directly calling it.

Low Severity Issues

3. Timestamp for signature expiry can be used

Path

N2MFactory.sol

Description

delegatedCreation() function verifies the signed signature for the function call, which allows executing creation transaction on behalf of others. Currently, theres no timestamp getting used to check if the signature is expired or not.

Adding an expiry timestamp in the signature verification logic will make sure that the signature is getting used within the expected timeframe (i.e. before it expires).

Recommendation

In the function it can be checked that the current block.timestamp is less than or equal to the expiry timestamp else the transaction will be reverted.

Status

Acknowledged

4. Make signature verification more secure

Path

N2MFactory.sol#L247

Function

delegatedCreation()

Description

Currently the verifying contract's address is not getting included while creating hash on L247 with ECDSA.toEthSignedMessageHash(). In the case where the team decides to deprecate the current contract because of some reason, and after deploying the new contract. The same used signature can be reused on that contract to create a new collection on behalf of the owner/signer.

(Currently signature verification is only getting used to deploy the contract, so even though any bad actor can use the signature again on a different contract version on the same chain, this directly creates no risk to the signer because the worst case possible here would be to create the another contract on behalf of the original owner/signer. Still its good idea to take care of it.)

Additionally, this function is not checking that the nonce is different for the specific owner address, which creates the risk of using the same signature again on the same factory contract, but in this case, the deployment process will revert as the contract is already present on that address if someone replays it.

Recommendation

Use the verifying contract’s (address(this)) address while creating a hash which will prevent cross-contract signature replay attack on the new contract version on the same chain.

Status

Acknowledged

5. Redundant import

Path

N2MFactory.sol#L52

Description

import {IN2MSequential} from "./interfaces/IN2MSequential.sol"; is not getting used in N2MFactory contract and can be removed.

Recommendation

Remove the unused import.

Status

Resolved

6. The remaining value should be sent back

Path

N2MFactory.sol

Function

multicallMulticollection()

Description

The caller can send msg.value to some functions if required for transaction execution. It can happen that while executing a call with multicallMulticollection(), more msg.value is getting sent than what is required.

In these types of cases, the remaining value can be sent back to the msg.sender.

Recommendation

Send the remaining value back to the msg.sender.

NFTs2Me Team's Comment

The specific value that is needed for execution should be sent while calling the function so that there would be no remaining value. Additionally, to keep gas costs lower, the function does not keep the track of sent value.

Status

Acknowledged

Informational Issues

7. Function create3 does not return the address of deployed contract

Path

N2MFactory.sol#L318

Function

create3()

Description

Function create3() is not returning the value of the deployed contract.

Recommendation

The value of the deployed contract can be returned if required.

NFTs2Me Team's Comment

Because of reasons like the address can be predicted and to keep the gas cost low, the function is not returning the address.

Status

Acknowledged

8. Delegatecall in the loop can be used maliciously

Path

N2MFactory.sol#331, N2MFactory.sol#367

Function

multicallN2M_001Taw5z(), multicall()

Description

multicall() and multicallN2M_001Taw5z() are using delegatecall() in the loop. Here any native token balance that the contract has, can be used by using delegate calls and not sending enough value to execute the transaction but instead using the value/native token balance from the contract.

Example: User performs multicall that takes some value from the contract. Let's say multicall to execute createNewDynamic() 2 times. which will deploy a contract and will call the function specified in the data with the msg.value provided. But because of the same msg.value will be reused multiple times while using delegateCall() in the loop, the user may send the value required for first createNewDynamic() call and for the next call, the value stored in the contract i.e N2MFactory contract's balance will be used.

Recommendation

Here, the msg.value can be cached first by storing in a variable and the value can be subtracted, once used for the first delegatecall loop iteration, so while using for the next iteration the value stored in the variable can be checked to be greater than the value user wanted to send. According to the fix, there would be other changes to the function are necessary if its implemented.

Auditor's Comment

The issue is medium to high severity but according to the project team the N2MFactory contract won't be storing any native balance according to the business logic that's why this issue does not affect the contract and is moved in the informational category.

Status

Acknowledged

9. createCrossCollection will fail on zkSync

Path

N2MFactory.sol

Function

createCrossCollection()

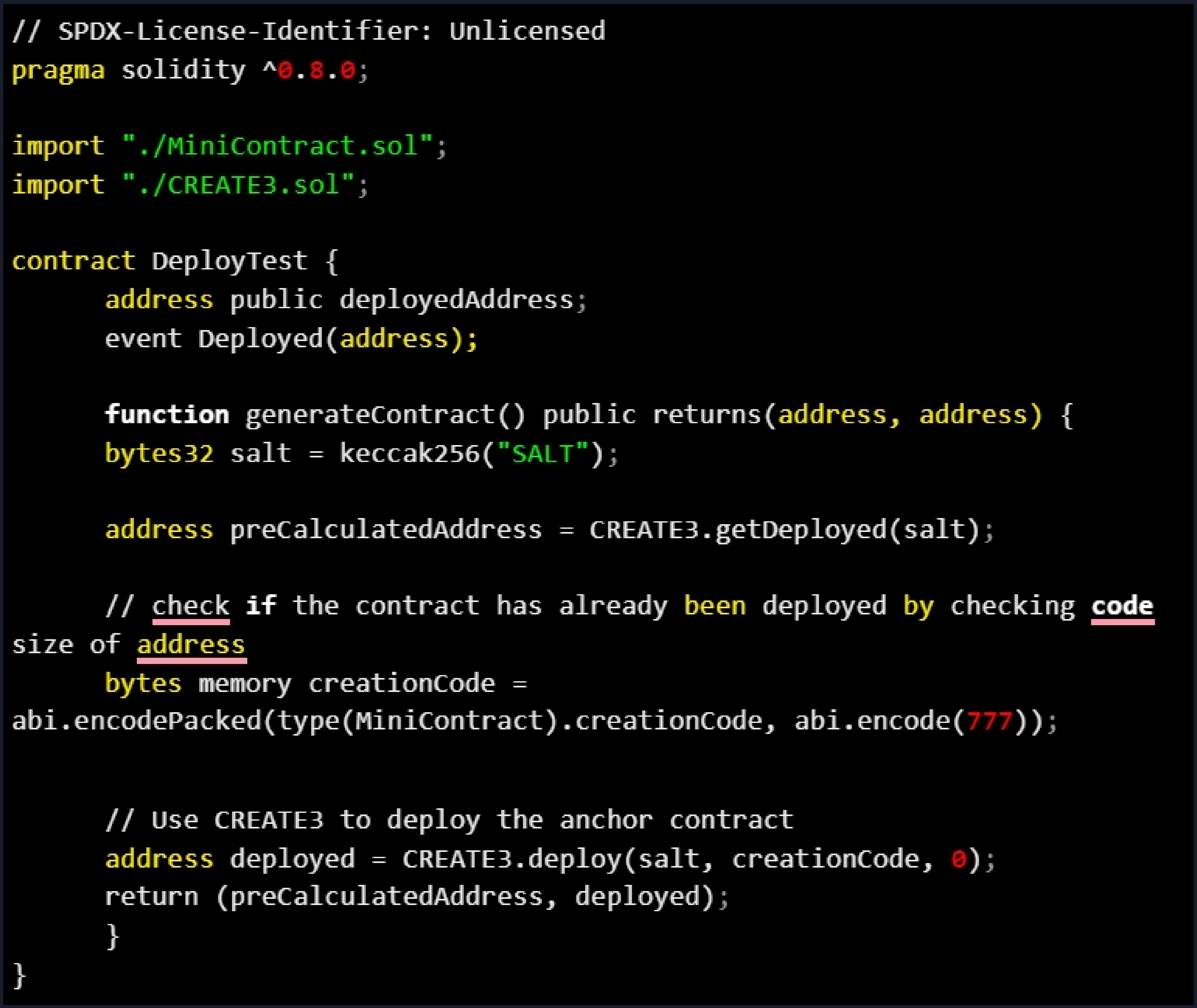
Description

The project is expected to be deployed on the zkSync network as well. However, the current implementation of createCrossCollection will not work as intended on zkSync due to CREATE3.

According to zkSync Era's **docs**, it states that create cannot be used for arbitrary code unknown to the compiler.

Here’s a sample POC code at zkSync Era Testnet

https://goerli.explorer.zksync.io/address/0x0f670f8AfcB09f4BC509Cb59D6e7CEC1A52BFA51#contract



```
// SPDX-License-Identifier: Unlicensed
pragma solidity ^0.8.0;

import "./MiniContract.sol";
import "./CREATE3.sol";

contract DeployTest {
      address public deployedAddress;
      event Deployed(address);

      function generateContract() public returns(address, address) {
      bytes32 salt = keccak256("SALT");

      address preCalculatedAddress = CREATE3.getDeployed(salt);

      // check if the contract has already been deployed by checking code
size of address
      bytes memory creationCode =
abi.encodePacked(type(MiniContract).creationCode, abi.encode(777));


      // Use CREATE3 to deploy the anchor contract
      address deployed = CREATE3.deploy(salt, creationCode, 0);
      return (preCalculatedAddress, deployed);
      }
}
```

Recommendation

This can be mitigated by directly using CREATE2 instead of CREATE3.

NFTs2Me Team’s Comment

Our factory for zkSync Era would be different.

Status

Acknowledged

10. Remove unused function

Path

Common.sol

Function

_setSoulBound

Description

The _setSoulBound internal function was not invoked within or outside the common.sol abstract contract. This function could have been used in the _mintSequential private function where the _soulBound mapping state variable sets a tokenId to true.

Recommendation

Remove this function since its original motive is performed in another function.

Status

Acknowledged

NFTs2Me Team's Comment

While this function is not used by N2MERC721A, other implementations that inherit from Common.sol use it. That's the reason why we keep it there. On N2MERC721A the compiler will just ignore it, leaving it irrelevant.

Functional Tests Cases

Some of the tests performed are mentioned below:

- ✓ Should be able to create a collection.
- ✓ Should be able to make multicall.
- ✓ Owner can add new contract implementation and signer.
- ✓ Should be able to create a collection on behalf of another user using signature.
- ✓ Should be able to transfer collection ownership.
- ✓ Should not be able to create collection with same signature.
- ✓ Should revert if the salt contains address other than msg.sender while creating collection.
- ✓ Should be able to execute multiple calls on multiple collections with msg.value.
- ✓ Should detect the critical state variables from the provided bytes values.
- ✓ Should verify that contract initialization is a one-time setup with packed data.
- ✓ Should check when a mint is within the end and drop date during public sale.
- ✓ Should confirm that only token owners can set traits and update users with expiry time.
- ✓ Should verify the updated owner when the expiry time is still valid.

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.

Closing Summary

In this report, we have considered the security of the NFTs2Me codebase. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in NFTs2Me smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of NFTs2Me smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the NFTs2Me to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2024

For

NFTs2Me

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com