CERTIK

Security Assessment

Manifold - NFT2ERC20

May 12th, 2021

Summary

This report has been prepared for Manifold - NFT2ERC20 smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Majority of the findings are of informational nature with one medium finding. The medium finding comprise the incorrect setting of offset for an array when encoding the payload for a low level contract call.

Overview

Project Summary

Project Name	Manifold - NFT2ERC20
Description	The audited codebase comprise an ERC20, ERC721 Receiver contract and ERC1155 Receiver contract. The ERC20 contract allows receiving of `ERC721` and `ERC1155` tokens, burns them and in return mints ERC20 tokens for the sender depending upon the rate stored for that `ERC721` or `ERC1155` contract in `ASHRateEngine` contract. The receiver contract receive the tokens of their corresponding types, set approval to ERC20 token for transfer and then call `burnToken` function on ERC20 contract to burn corresponding NFT token and receive the ERC20 tokens.
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/manifoldxyz/nft2erc20-solidity/tree/732412f038685a7da54313bf6fe55ea2ba201bc1/contracts
Commits	1. https://github.com/manifoldxyz/nft2erc20-solidity/tree/732412f038685a7da54313bf6fe55ea2ba201bc1/contracts 2. https://github.com/manifoldxyz/nft2erc20-solidity/commit/adc49c19e3ac9d75c2d2ae81925c7639beebba26

Audit Summary

Delivery Date	May 12, 2021
Audit Methodology	Manual Review, Static Analysis
Key Components	

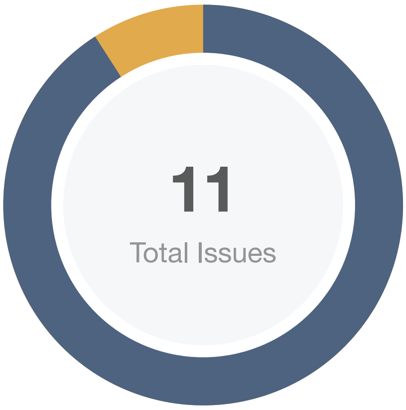
Vulnerability Summary

Total Issues	11
Critical	0
Major	0
Medium	1
Minor	0
Informational	10
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
INF	INFT2ERC20.sol	9ae3fee0a95db58b0069c855539cac2c951fe69232ca92a01fdd8fae02c4c160
MIG	Migrations.sol	4fd6092bdfa8b42f19d535c5ac69c4323b0b894717c699e58d5552eeabd04cd4
NFT	NFT2ERC20.sol	47df16c591e2fe428efc8b61aec06ebdd7fdc8a5534672e81ec7deeafd0948af
MER	mocks/MockERC1155.sol	09e4994ee864ca38e09bc89110cb6e243ac6e829e5ebbf0dcb4d6a6efac5753a
MEC	mocks/MockERC721.sol	f32a524e0ad9f21e05c58cc51dee4dc4721e1fb628f069c88da6430b97250023
MNF	mocks/MockNFT2ERC20RateEngine.sol	c9a2d012ba6c01759747eb3c929dd66d64cbac45d6646dd399f8417ce17bb1b0
INT	rates/INFT2ERC20RateEngine.sol	61ad6b9bd06d68486bc090e16ab9f581c7a4f90a626a60165cefd42b86ac4224
NFE	rates/NFT2ERC20RateEngine.sol	1e880247c69e597a9c5e8bc06f15cc3e81813d60ae9a08fb413124455e623fba
ERC	receivers/ERC1155NFT2ERC20Receiver.sol	afcfbf67298980b1aab742f1c6028bc652770d560cdea23efebff9a6c3e1ae23
ERN	receivers/ERC721NFT2ERC20Receiver.sol	858b3b188dff713ee4f7f4c2a4be856b4f8848f43dc4ca9f732a8c3947985d17

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**1** (9.09%)
Minor	**0** (0.00%)
Informational	**10** (90.91%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
ERC-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
ERC-02	Unspecified state variable visibility	Language Specific	Informational	Resolved
ERC-03	Comparison with literal `true`	Gas Optimization	Informational	Resolved
ERC-04	Incorrect offset is provided for the `data` bytes array	Volatile Code	Medium	Resolved
ERN-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
ERN-02	Unspecified state variable visibility	Language Specific	Informational	Resolved
ERN-03	Comparison with literal `true`	Gas Optimization	Informational	Resolved
INF-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
INT-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
NFE-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
NFT-01	Unlocked Compiler Version	Language Specific	Informational	Resolved

ERC-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	receivers/ERC1155NFT2ERC20Receiver.sol: 3	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.7.0` the contract should contain the following line: `pragma solidity 0.8.2;`.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26`.

ERC-02 | Unspecified state variable visibility

Category	Severity	Location	Status
Language Specific	Informational	receivers/ERC1155NFT2ERC20Receiver.sol: 15	Resolved

Description

The `_nft2erc20` state variable in the aforementioned contract should have its visibility specified.

Recommendation

Consider specifying the visibility of the `_nft2erc20` state variable in the aforementioned contract as internal or public.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26` .

ERC-03 | Comparison with literal `true`

Category	Severity	Location	Status
Gas Optimization	Informational	receivers/ERC1155NFT2ERC20Receiver.sol: 33, 57	Resolved

Description

The aforementioned lines perform comparison with literal `true` which can substituted with the negation of the expression to save gas and increase the legibility of the codebase.

Recommendation

We advise to substitute the comparison with literal `true` on the aforementioned lines with the negation of the expression.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26`.

ERC-04 | Incorrect offset is provided for the `data` bytes array

Category	Severity	Location	Status
Volatile Code	Medium	receivers/ERC1155NFT2ERC20Receiver.sol: 40, 62	Resolved

Description

The aforementioned lines set incorrect offset `96` for the `data` bytes array. The signature of transfer function for `ERC1155` is `safeTransferFrom(address from, address to, uint256 id, uint256 amount, byte memory data);` where offset `96` will point to the data stored in `amount` which incorrectly acts as length of the `data` array. The correct offset for the `data` array is `160` (5 * 32 bytes) as length of the `data` array is stored in `32 bytes` next to variable `data` which stores the `data` array's offset.

Recommendation

We advise to rectify the offset set on the aforementioned lines for `data` bytes array to `160`.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26`.

ERN-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	receivers/ERC721NFT2ERC20Receiver.sol: 3	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.7.0` the contract should contain the following line: `pragma solidity 0.8.2;`.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26`.

ERN-02 | Unspecified state variable visibility

Category	Severity	Location	Status
Language Specific	Informational	receivers/ERC721NFT2ERC20Receiver.sol: 15	Resolved

Description

The `_nft2erc20` state variable in the aforementioned contract should have its visibility specified.

Recommendation

Consider specifying the visibility of the `_nft2erc20` state variable in the aforementioned contract as internal or public.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26` .

ERN-03 | Comparison with literal `true`

Category	Severity	Location	Status
Gas Optimization	● Informational	receivers/ERC721NFT2ERC20Receiver.sol: 32	Resolved

Description

The aforementioned line performs comparison with literal `true` which can substituted with the negation of the expression to save gas and increase the legibility of the codebase.

Recommendation

We advise to substitute the comparison with literal `true` on the aforementioned line with the negation of the expression.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26` .

INF-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	INFT2ERC20.sol: 3	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.7.0` the contract should contain the following line: `pragma solidity 0.8.2;`.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26`.

INT-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	rates/INFT2ERC20RateEngine.sol: 3	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.7.0` the contract should contain the following line: `pragma solidity 0.8.2;`.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26`.

NFE-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	rates/NFT2ERC20RateEngine.sol: 3	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.7.0` the contract should contain the following line: `pragma solidity 0.8.2;`.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26` .

NFT-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	NFT2ERC20.sol: 3	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.7.0` the contract should contain the following line: `pragma solidity 0.8.2;`.

Alleviation

Alleviations were applied as of commit hash `adc49c19e3ac9d75c2d2ae81925c7639beebba26` .

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK