#HashEx
BLOCKCHAIN SECURITY

Mad Metaverse Ethereum

smart contracts
final audit report

April 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Mad Metaverse team to perform an audit of their smart contracts. The audit was conducted between 25/04/2022 and 29/04/2022.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at the @MadMetaverse/mad_ethereum GitHub repository and was audited after the commit 52c6714.

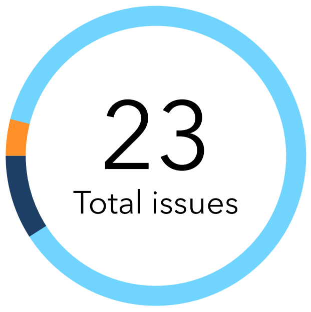
2.1 Summary

Project name	Mad Metaverse Ethereum
URL	https://madmetaverse.com
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
EIP712Base	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/meta-transactions/EIP712Base.sol
ERC721Tradable	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/ERC721Tradable.sol
Scientist	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/Scientist.sol
ScientistsFactory	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/ScientistsFactory.sol
IAllowance	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/interfaces/IAllowance.sol
ScientistData	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/libraries/ScientistData.sol
IRepository	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/interfaces/IRepository.sol
Repository	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/Repository.sol
Marketplace	https://github.com/MadMetaverse/mad_ethereum/blob/52c67145b34c9d1f03cd3c478633f23af33655dc/contracts/Marketplace.sol

3. Found issues



●	High	1 (4%)
●	Low	20 (87%)
●	Info	2 (9%)

C1. EIP712Base

ID	Severity	Title	Status
C1-01	● Info	Getting a chainId	Open
C1-02	● Info	Typos	Open

C2. ERC721Tradable

ID	Severity	Title	Status
C2-01	● Low	Constructor lacks validation of the input parameters	Open
C2-02	● Low	Gas optimization	Open
C2-03	● Low	Variable default visibility	Open

C3. Scientist

ID	Severity	Title	Status
C3-01	● Low	Gas optimization	Open

C4. ScientistsFactory

ID	Severity	Title	Status
C4-01	● Low	Lack validation of input parameters	Open
C4-02	● Low	Reason message in require()	Open
C4-03	● Low	Overcomparison	Open
C4-04	● Low	Gas optimization	Open
C4-05	● Low	Lack of events	Open
C4-06	● Low	Redundant validation for uint type	Open
C4-07	● Low	Using assert() instead of require()	Open

C5. IAllowance

ID	Severity	Title	Status
C5-01	● Low	Lack of events	Open
C5-02	● Low	Compare to a boolean constant	Open

C6. ScientistData

ID	Severity	Title	Status
C6-01	● Low	Unused variables	Open

C7. IRepository

ID	Severity	Title	Status
C7-01	● Low	Lack of events	Open

C8. Repository

ID	Severity	Title	Status
C8-01	● High	Malicious operator	Open
C8-02	● Low	Unused library	Open
C8-03	● Low	Redundant validation for uint type	Open

C9. Marketplace

ID	Severity	Title	Status
C9-01	● Low	Constructor lacks validation of input parameters	Open
C9-02	● Low	Gas optimization	Open
C9-03	● Low	Redundant validation for uint type	Open

4. Contracts

C1. EIP712Base

Overview

The contract has the base functionality of the EIP-712 standard. There are functions for changing or getting the DomainSeparator variable, hashing the messageHash, and getting the chain id.

Issues

C1-01 Getting a chainId ● Info Open

Starting with version 0.8 of the Solidity language, chainId can be obtained not only with `assembly`, but also with a global variable `block.chainid`.

C1-02 Typos ● Info Open

Typos reduce the code's readability.1) 23L 'domainSeperator' should be replaced with 'domainSeparator';

2) 29L '_setDomainSeperator' should be replaced with '_setDomainSeparator';

3) 32L '_setDomainSeperator' should be replaced with '_setDomainSeparator';

4) 33L 'domainSeperator' should be replaced with 'domainSeparator';

5) 44L 'getDomainSeperator' should be replaced with 'getDomainSeparator';

6) 45L 'domainSeperator' should be replaced with 'domainSeparator';

7) 70L 'getDomainSeperator' should be replaced with 'getDomainSeparator'

C2. ERC721Tradable

Overview

The abstract contract is inherited from the functionality of ERC721Enumerable, Ownable, and EIP-712 standard. It has the basic functionality of an NFT of this project.

Issues

C2-01 Constructor lacks validation of the input parameters ● Low ⓘ Open

The contract constructor does not check the addresses `proxyRegistryAddress` for a non-zero address.

C2-02 Gas optimization ● Low ⓘ Open

a. The functions `_getNextTokenId()`, `_incrementTokenId()`, `burn()` are never used and can be removed to save gas in deployment.

b. The functions `mintTo()`, `baseTokenURI()` can be declared as external to save gas.

C2-03 Variable default visibility ● Low ⓘ Open

The variable `proxyRegistryAddress` (L29) has default visibility. Labeling visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

C3. Scientist

Overview

The ERC721 contract inherited the ERC721Tradable contract.

Issues

C3-01 Gas optimization ● Low ⓘ Open

The functions `baseTokenURI()`, `contractURI()` can be declared as `external` to save gas.

C4. ScientistsFactory

Overview

This contract is responsible for the creation and minting of new NFT tokens. The contract owner or the `proxyRegistryAddress` have the ability to mint NFT tokens.

Issues

C4-01 Lack validation of input parameters ● Low ⓘ Open

a. The contract constructor does not check the addresses `proxyRegistryAddress` and `scientistNftAddress` for a non-zero address.

b. The function `setToken()` does not check the address `_token` for a non-zero address.

C4-02 Reason message in require() ● Low ⓘ Open

Reason message should be included as it can simplify understanding of what errors are occurring. Error names omitted in L179.

C4-03 Overcomparison ● Low ⓘ Open

On L90 the `marketplace` is compared to zero-address, but this comparison was already included in L89.

C4-04 Gas optimization ● Low ⓘ Open

a. The `fireTransferEvents()` function emits the `Transfer` event 1000 times (in deployment configuration considered 10000 times) in deployment or in performing the `transferOwnership()` function. This can lead to huge gas costs (approximately 2.2M gas for 1000 events). We recommend reconsidering the need to use event or for example, use event only for first and last id.

b. The variable `SCIENTISTS_NUM_OPTIONS` is read at each step of the loop at L62 and L71. Consider using a local variable instead of reading the storage every time to save gas.

c. The `INITIAL_AMOUNT` state variable is only used in the `createScientist()` function and serves the same purpose as `SCIENTISTS_NUM_OPTIONS`. In this case, before executing the `createScientist()` function, the possibility of creating a new token is checked by the `canMint()` function on L179. Thus, the `INITIAL_AMOUNT` variable is redundant and can be removed from the contract. It can save gas in deployment and executions.

d. The state variable `MAX_LEVEL_OF_EVOLUTION` has default `private` visibility and never used in the contract. It can be removed to save gas in deployment.

e. The state variable `lastIndex` can be declared as local variable inside the `addTokens()` function.

f. The state variable `proxyRegistryAddress` can be declared as `immutable` to save gas.

g. The `numOptions()`, `supportsFactoryInterface()`, `addTokens()`, `setRepository()` functions can be declared as external to save gas.

C4-05 Lack of events ● Low ⓘ Open

The functions `setRepository()`, `setMarketplace()`, `setToken()` don't emit events, which complicates the tracking of important off-chain changes.

Recommendation

Create events for these functions.

C4-06 Redundant validation for uint type ● Low ⓘ Open

On L118, there is redundant validation because the `uint` variable >= 0 by default.

C4-07 Using assert() instead of require() ● Low ⓘ Open

For checking, we recommend using `require()` statements instead of `assert()` (L175) to conform to the best practices in smart contracts development. This saves gas if a wrong parameter is passed to a function: the `assert()` statement uses all the gas provided in the transaction while the `require()` statement uses only gas spent before a failing statement is reached.

C5. IAllowance

Overview

The contract is responsible for the creation and minting of new NFT tokens. At the time of minting it also creates data about the token in the Repository contract.

Issues

C5-01 Lack of events ● Low ⓘ Open

The functions `_setAllowance()`, `_removeAllowance()` don't emit events, which complicates the tracking of important off-chain changes.

C5-02 Compare to a boolean constant ● Low ⓘ Open

On L8 used redundant comparison with `true`.

```
modifier allowedOperator() {
        require(allowedOperators[msg.sender] == true, "Caller is not allowed");
        _;
}
```

C6. ScientistData

Overview

The abstract contract stores additional information (ScientistData) about issued NFT tokens.

Issues

C6-01 Unused variables ● Low ⓘ Open

The variables `level` and `tokenUri` of the structure `Scientist` are never used.

C7. IRepository

Overview

The abstract contract stores information about issued NFT tokens.

Issues

C7-01 Lack of events ● Low ⓘ Open

The functions `_addScientist()`, `_removeScientist()`, `_updateScientist()` don't emit events, which complicates the tracking of important off-chain changes.

Recommendation

Create events for these functions.

C8. Repository

Overview

The contract implements the IRepository contract with management functions. Using this contract, the operator can add, delete or update information about a token.

Issues

C8-01 Malicious operator ● High ⓘ Open

The first contract operator has the ability to add an unlimited amount of other operators. Each of these operators:

a. can break the minting of new NFT tokens by ScientistsFactory if it performs the `addScientist()` function with `tokenId` that does not yet exist;

b. can change the price of the token being sold or any other parameters using `updateScientist()` function. This allows buying the token at zero price or breaking the sale.

Recommendation

It is necessary to set only one operator in the contract constructor - ScientistsFactory.

C8-02 Unused library ● Low ⓘ Open

The functionality of the imported `Ownable` library is not used in this contract. Thus, it can be removed to save gas in deployment.

C8-03 Redundant validation for uint type ● Low ⓘ Open

On L32, L41, L49 there is redundant validation because the `uint` variable >= 0 by default.

Recommendation

It is recommended to remove this check.

C9. Marketplace

Overview

The contract allows to sell and buy NFT tokens for Ether.

Issues

C9-01 Constructor lacks validation of input parameters ● Low ⓘ Open

The contract constructor does not check the addresses `repository`, `factory` and `scientistToken` for a non-zero address.

C9-02 Gas optimization ● Low ⓘ Open

a. The state variables `repository`, `factory`, `scientistToken` should be declared as immutable to save gas.

b. The private state variable `factory` is never used and should be removed.

C9-03 Redundant validation for uint type ● Low ⓘ Open

In 108L, 113L, 138L there is redundant validation because the `uint` variable >= 0 by default.

Recommendation

It is recommended to remove this check.

5. Conclusion

1 high, 21 low, and 2 informational severity issues were found.

The contracts Scientist and Repository are highly dependent on the owner's account. After deployment, the ownership of the Scientist contract must be transferred to the ScientistsFactory contract.

We strongly suggest adding unit and functional tests for all contracts.

We also recommend using pragma fixed to the version the contracts have been tested and are intended to be deployed with. This helps to avoid deploying using an outdated compiler version and shields from possible bugs in future solidity releases.

This audit includes recommendations on improving the code and preventing potential attacks.

Appendix A. Issues severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Info.** Issues that do not impact the contract operation. Usually, info severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY