#HashEx
BLOCKCHAIN SECURITY

Mad Metaverse Scientist

smart contracts
final audit report

September 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 contract. The audit was conducted between 28/08/2022 and 01/09/2022.

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 ProxyAdmin and TransparentUpgradeableProxy contracts are the original Openzeppelin contracts.

The code is available at the @MadMetaverse/mad_ethereum GitHub repository and was audited after the commit c648fd3.

Audit scope:

IScientistCharacteristic.sol,
IScientistMintable.sol, IScientistRepository.sol, ERC721TradableUpgradeable.sol, AScientistRepository.sol, Scientist.sol, ProxyAdmin.sol, TransparentUpgradeableProxy.sol, TimelockAccess.sol, ITimelock.sol, Timelock.sol.

Update: the Mad Metaverse team has responded to this report. The updated code is located in the GitHub repository after the commit 06bb23f.

2.1 Summary

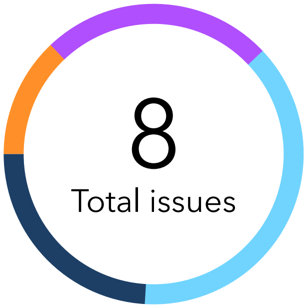
Project name	Mad Metaverse Scientist
URL	https://madmetaverse.com
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
ERC721TradableUpgradeable	https://github.com/MadMetaverse/mad_ethereum/blob/c648fd39fdc78bd40c0d2c87ed32908ccb326f23/contracts/scientist/ERC721TradableUpgradeable.sol
AScientistRepository	https://github.com/MadMetaverse/mad_ethereum/blob/c648fd39fdc78bd40c0d2c87ed32908ccb326f23/contracts/scientist/abstracts/AScientistRepository.sol
Scientist	https://github.com/MadMetaverse/mad_ethereum/blob/c648fd39fdc78bd40c0d2c87ed32908ccb326f23/contracts/scientist/Scientist.sol
ProxyAdmin	https://github.com/MadMetaverse/mad_ethereum/blob/c648fd39fdc78bd40c0d2c87ed32908ccb326f23/contracts/proxy/ProxyAdmin.sol
TransparentUpgradeableProxy	https://github.com/MadMetaverse/mad_ethereum/blob/c648fd39fdc78bd40c0d2c87ed32908ccb326f23/contracts/proxy/TransparentUpgradeableProxy.sol
TimelockAccess	https://github.com/MadMetaverse/mad_ethereum/blob/c648fd39fdc78bd40c0d2c87ed32908ccb326f23/contracts/helpers/timelock-access/TimelockAccess.sol

Timelock	https://github.com/MadMetaverse/mad_ethereum/blob/c648fd39fdc78bd40c0d2c87ed32908ccb326f23/contracts/helpers/timelock/Timelock.sol
IScientistCharacteristic	https://github.com/MadMetaverse/mad_ethereum/blob/c648fd39fdc78bd40c0d2c87ed32908ccb326f23/contracts/scientist/interfaces/IScientistCharacteristic.sol
IScientistMintable	
IScientistRepository	
ITimelock	

3. Found issues



●	High	1 (13%)
●	Medium	2 (25%)
●	Low	3 (38%)
●	Info	2 (24%)

C3. Scientist

ID	Severity	Title	Status
C3-01	● High	Malicious operator	✓ Acknowledged
C3-02	● Low	Few events	✓ Resolved
C3-03	● Low	Unused variable	✓ Resolved
C3-04	● Low	Functions lacks validation of input parameters	✓ Resolved
C3-05	● Info	Typos	✓ Resolved

C6. TimelockAccess

ID	Severity	Title	Status
C6-01	● Medium	Changing timelock address	✓ Resolved

C7. Timelock

ID	Severity	Title	Status
C7-01	● Medium	Low minimal delay	✓ Resolved

C8. IScientistCharacteristic

ID	Severity	Title	Status
C8-01	● Info	Typos	✓ Resolved

4. Contracts

C1. ERC721TradableUpgradeable

Overview

The abstract contract is inherited from the functionality of ERC721EnumerableUpgradeable and EIP-712 standard. It has the basic functionality of an NFT of this project.

C2. AScientistRepository

Overview

This contract implements functions for working with Scientists token data (adding, removing, updating).

C3. Scientist

Overview

The ERC721 contract inherited the ERC721TradableUpgradeable contract.

Issues

C3-01 Malicious operator ● High ✓ Acknowledged

The admin of the Timelock contract can add an unlimited amount of operators for the contract Scientist. Each of these operators:

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

b. can perform the `removeScientist()` function on any Scientist token. It can break transfers and remove ScientistData of that token.

c. can change the price of the token being sold or any other parameters using the `updateScientist()` function. This could potentially allow the token to be bought at zero price or disrupt the sale.

d. can mint an unlimited amount of tokens using `mint()` function. This can lead to unfair tokenomic (or game economic).

Recommendation

Consider restricting rights of operator. Or make the operator a single contract that can perform the above functions only when necessary.

Developer response

According to developer operators will be reliable and strictly controlled contracts.

Update

Based on the developer's answer, it should be noted that as described in the issue, the administrator **can make any new address the operator**. Thus, even if there is only one honest operator at the time of deploying the contracts, then nothing prevents the creation of other dishonest operators (including EOA accounts) in the future.

C3-02 Few events ● Low ✓ Resolved

The functions `addScientist()`, `removeScientist()`, `updateScientist()` don’t emit events, which complicates the tracking of important off-chain changes.

C3-03 Unused variable ● Low ✓ Resolved

The state variable `mintableAmount` is never used in the contract code and can be removed.

C3-04 Functions lacks validation of input parameters ● Low ✓ Resolved

The contract functions `initialize()` and `setProxyRegistryAddress()` do not check the input addresses against a zero address.

C3-05 Typos ● Info ✓ Resolved

Typos reduce the code's readability.1) L181 'traits.personalityDiorders' should be replaced with 'traits.personalityDi**s**orders'.

C4. ProxyAdmin

Overview

This is an auxiliary contract meant to be assigned as the admin of a TransparentUpgradeableProxy. The ProxyAdmin contract are the original Openzeppelin contract.

C5. TransparentUpgradeableProxy

Overview

This contract implements a proxy that is upgradeable by an admin. The TransparentUpgradeableProxy contract is the original Openzeppelin contract.

C6. TimelockAccess

Overview

This contract implements the `onlyTimelock()` modifier and the `setTimelock()` setter function.

Issues

C6-01 Changing timelock address ● Medium ✓ Resolved

The administrator of the Timelock contract can change the address of the TimeLock contract using the `setTimelock()` function. Behind the new address, there may be a dishonest new Timelock contract or EOA account, which can harm the users of the project.

Despite the fact that users will have some time to analyze the transaction that is in the queue (with a new timelock address), this still carries the risk that users might not have time to exit the project and save their funds.

Recommendation

We recommend restricting the possibility of changing the address of the `Timelock` contract

C7. Timelock

Overview

This contract implements a delayed call of functions by adding them to the queue with the desired delay and then calling after the end of the delay.

Issues

C7-01 Low minimal delay ● Medium ✓ Resolved

The timelock contract can set the `minimumDelay` of 1 second. The point of using a TimeLock contract is lost if such a small delay is used.

If the contract is initialized with such a low delay, or if such a transaction is queued, then the risks of using this project will increase many times over.

Recommendation

We recommend setting the minimumDelay variable for at least 24 hours.

C8. IScientistCharacteristic

Overview

This interface contains many enumerations for working with the characteristics of scientists.

Issues

C8-01 Typos ● Info ✓ Resolved

Typos reduce the code's readability.

1) L31 'PARANIOD' should be replaced with 'PARAN**OI**D';

2) L5 'scientitst' should be replaced with 'scient**is**t' (ScientistData.sol file).

C9. IScientistMintable

Overview

The interface for the Scientist contract.

C10. IScientistRepository

Overview

The interface for the AScientistRepository contract.

C11. ITimelock

Overview

The interface for the Timelock contract.

5. Conclusion

1 high, 2 medium, 3 low severity issues were found during the audit. 2 medium, 3 low issues were resolved in the update.

2 medium, 3 low and 2 infomational severity issues have been resolved in the update.

The reviewed contracts are highly dependent on the owner’s account. **Users using the project have to trust the owner (admin) and that the owner's (admin's) account is properly secured**.

We strongly suggest adding and fixing 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.
- **Informational.** Issues that do not impact the contract operation. Usually, informational 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