Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Another World DAO
Platform: Ethereum
Language: Solidity
Date: May 23rd, 2023

Table of contents

Introduction 4
Project Background 4
Audit Scope 5
Claimed Smart Contract Features 6
Audit Summary 7
Technical Quick Stats 8
Code Quality 9
Documentation 9
Use of Dependencies 9
AS-IS overview 10
Severity Definitions 12
Audit Findings 13
Conclusion 17
Our Methodology 18
Disclaimers 20
Appendix

- Code Flow Diagram 21
- Slither Results Log 23
- Solidity static analysis 25
- Solhint Linter 28

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by Another World DAO to perform the Security audit of the Another World DAO smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 23rd, 2023.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The Another World DAO Contracts handle multiple contracts, and all contracts have different functions.
    - TreasureFragments: Treasure Fragments can be forged to mint another item.
    - MerkleDistributor: Distribute ERC-20 tokens based on Merkle proofs.

- Another World DAO is a NFT smart contract which has functions like withdraw, claim, _mintBatch, burn, burnBatch, etc.

Email: audit@EtherAuthority.io

Audit scope

Name	Code Review and Security Analysis Report for Another World DAO Smart Contracts
Platform	**Ethereum / Solidity**
File 1	MerkleDistributor.sol
File 1 MD5 Hash	B5C6297C189CDB0018BB4277068F668D
Updated File 1 MD5 Hash	B5235E9E471525341EBDEFA822496008
File 2	TreasureFragments.sol
File 2 MD5 Hash	E9FE5B85D69C81626832EFEC85FAD7C4
Github Commit Hash	2cb8921df01912d2d4071e2789279e1f5dde6f51
Updated Github Commit Hash	34627bb929a3c56865e2ef7483031e46bc70cfea
Audit Date	May 23rd, 2023

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 MerkleDistributor.sol **<u>Owner has control over following functions:</u>** • Set the merkle root values. • Set the reward token address. • Withdraw tokens. • Current owner can transfer ownership of the contract to a new account. • Deleting ownership will leave the contract without an owner, removing any owner-only functionality.	**YES, This is valid.**
File 2 TreasureFragments.sol • Name: TreasureFragments • Symbol: FRAG • Refine Fee: 0.01 ether **<u>Owner has control over following functions:</u>** • Set the mint contract address. • Set a refine fee. • Set a URI. • Set an airdrop value. • Set the vault operator address. • Withdraw tokens. • Current owner can transfer ownership of the contract to a new account. • Deleting ownership will leave the contract without an owner, removing any owner-only functionality.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“Secured”**. Also, these contracts do not contain owner control, which makes them fully decentralized.

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 2 low and some very low level issues.

All the issues have been resolved / acknowledged in the revised code.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Email: audit@EtherAuthority.io

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Code Quality

This audit scope has 2 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Another World DAO Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Another World DAO Protocol.

The Another World DAO team has provided unit test scripts, which helped to determine the integrity of the code in an automated way.

Code parts are not well commented on smart contracts.

Documentation

We were given a Another World DAO Protocol smart contract code in the form of a github web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are not well commented. But the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

TreasureFragments.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOwner	modifier	Passed	No Issue
3	owner	read	Passed	No Issue
4	_checkOwner	internal	Passed	No Issue
5	renounceOwnership	write	access only Owner	No Issue
6	transferOwnership	write	access only Owner	No Issue
7	_transferOwnership	internal	Passed	No Issue
8	supportsInterface	read	Passed	No Issue
9	uri	read	Passed	No Issue
10	balanceOf	read	Passed	No Issue
11	balanceOfBatch	read	Passed	No Issue
12	setApprovalForAll	write	Passed	No Issue
13	isApprovedForAll	read	Passed	No Issue
14	safeTransferFrom	write	Passed	No Issue
15	safeBatchTransferFrom	write	Passed	No Issue
16	_safeTransferFrom	internal	Passed	No Issue
17	_safeBatchTransferFrom	internal	Passed	No Issue
18	_setURI	internal	Passed	No Issue
19	_mint	internal	Passed	No Issue
20	_mintBatch	internal	Passed	No Issue
21	_burn	internal	Passed	No Issue
21	_burnBatch	internal	Passed	No Issue
22	_setApprovalForAll	internal	Passed	No Issue
23	_beforeTokenTransfer	internal	Passed	No Issue
24	_afterTokenTransfer	internal	Passed	No Issue
25	_doSafeTransferAcceptanceCheck	write	Passed	No Issue
26	_doSafeBatchTransferAcceptanceCheck	write	Passed	No Issue
27	_asSingletonArray	write	Passed	No Issue
28	burn	write	Passed	No Issue
29	burnBatch	write	Passed	No Issue
30	setMintContract	external	access only Owner	No Issue
31	toggleForge	external	access only Owner	No Issue
32	toggleRefinement	external	access only Owner	No Issue
33	setTokenForgingRequirement	external	access only Owner	No Issue
34	checkTokenForgingRequirement	read	Passed	No Issue
35	forge	external	Passed	No Issue
36	refine	external	Function is accepting more	Refer to audit findings

			payment than the require	
37	setRefineFee	external	Owner can set refineFee without any limit	Refer to audit findings
38	setURI	external	access only Owner	No Issue
39	setVaultOperator	external	access only Owner	No Issue
40	toggleAirdrop	external	access only Owner	No Issue
41	uri	read	Passed	No Issue
42	airdrop	external	Passed	No Issue
43	random	internal	Passed	No Issue
44	withdraw	external	access only Owner	No Issue

MerkleDistributor.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOwner	modifier	Passed	No Issue
3	owner	read	Passed	No Issue
4	_checkOwner	internal	Passed	No Issue
5	renounceOwnership	write	access only Owner	No Issue
6	transferOwnership	write	access only Owner	No Issue
7	_transferOwnership	internal	Passed	No Issue
8	updateMerkleRoot	external	access only Owner	No Issue
9	updateRewardToken	external	access only Owner	No Issue
10	claim	external	Passed	No Issue
11	withdraw	external	Owner can drain all the reward tokens	Refer to audit findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found in the contract code.

High Severity

No high severity vulnerabilities were found in the contract code.

Medium

No medium severity vulnerabilities were found in the contract code.

Low

(1) Owner can set refineFee without any limit: **TreasureFragments.sol**

Owner is able to set a refine fee without any maximum or minimum range.

Resolution: We suggest using some range for fees.

Status: **Acknowledged**

(2) Function is accepting more payment than the require: **TreasureFragments.sol**

```
function refine(uint256 tokenId, uint256 amount) external payable {
    require(canRefine, "refinement not open");
    require(amount > 0, "nothing to refine");
    require(
        balanceOf(msg.sender, tokenId) >= amount,
        "not enough to refine"
    ); // check refinement requirement
    require(
        msg.value >= refineFee * amount,
        "not enough to pay refinement"
    );
```

While refine, if an user send more amount than the refineFee * amount of tokens, then the contract does not send back the extra amount to the user.

Resolution: We suggest validating for an exact amount or send back the extra amount to the user.

Status: Acknowledged

Very Low / Informational / Best practices:

(1) Function input parameters lack of check:

TreasureFragments.sol

```
// owner can update refineFee
function setRefineFee(uint256 newFee) external onlyOwner {
    refineFee = newFee;
}

// owner can update metadata uri
function setURI(string memory newuri) external onlyOwner {
    baseUri = newuri;
}

// owner can update vault operator
function setVaultOperator(address newOperator) external onlyOwner {
    vaultOperator = newOperator;
}
```

MerkleDistributor.sol

```
function updateRewardToken(address token_) external onlyOwner {
    token = token_;
    emit RewardTokenUpdated(token, merkleRoot);
}
```

Above functions do not validate the input before resetting the global value.

Resolution: We suggest validating like: numeric values should be greater than 0 and address type variables should not be address(0).

Status: Fixed for MerkleDistributor.sol

(2) Owner can drain all the reward tokens: **MerkleDistributor.sol**

Owner is able to withdraw all the rewards tokens from the contract.

Resolution: We suggest confirming if this is required or not.

Status: Acknowledged

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

TreasureFragments.sol

- setMintContract: Mint contract address can be updated by the owner.
- toggleForge: Forge can be toggled by the owner.
- toggleRefinement: Refinement can be toggled by the owner.
- setTokenForgingRequirement: Token forging requirement values can be set by the owner.
- setRefineFee: RefineFee can be set by the owner.
- setURI: URI can be set by the owner.
- setVaultOperator: Vault Operator address can be set by the owner.
- toggleAirdrop: Airdrop value can be toggled by the owner.
- airdrop: Airdrop value can be set by the owner.
- withdraw: Withdraw token by the owner.

MerkleDistributor.sol

- updateMerkleRoot: Merkle Root values can be updated by the owner.
- updateRewardToken: Reward token address can be updated by the owner.
- withdraw: Withdraw token by the owner.

Ownable.sol

- renounceOwnership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.
- _checkOwner: Throws if the sender is not the owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a github web link. And we have used all possible tests based on given objects as files. We had observed 2 low severity issues and some informational issues in the smart contracts, but those are not critical ones. One of the low issues has been resolved in the revised code and the rest are acknowledged. **So, the smart contracts are ready for the mainnet deployment**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed contract, based on standard audit procedure scope, is **"Secured"**.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

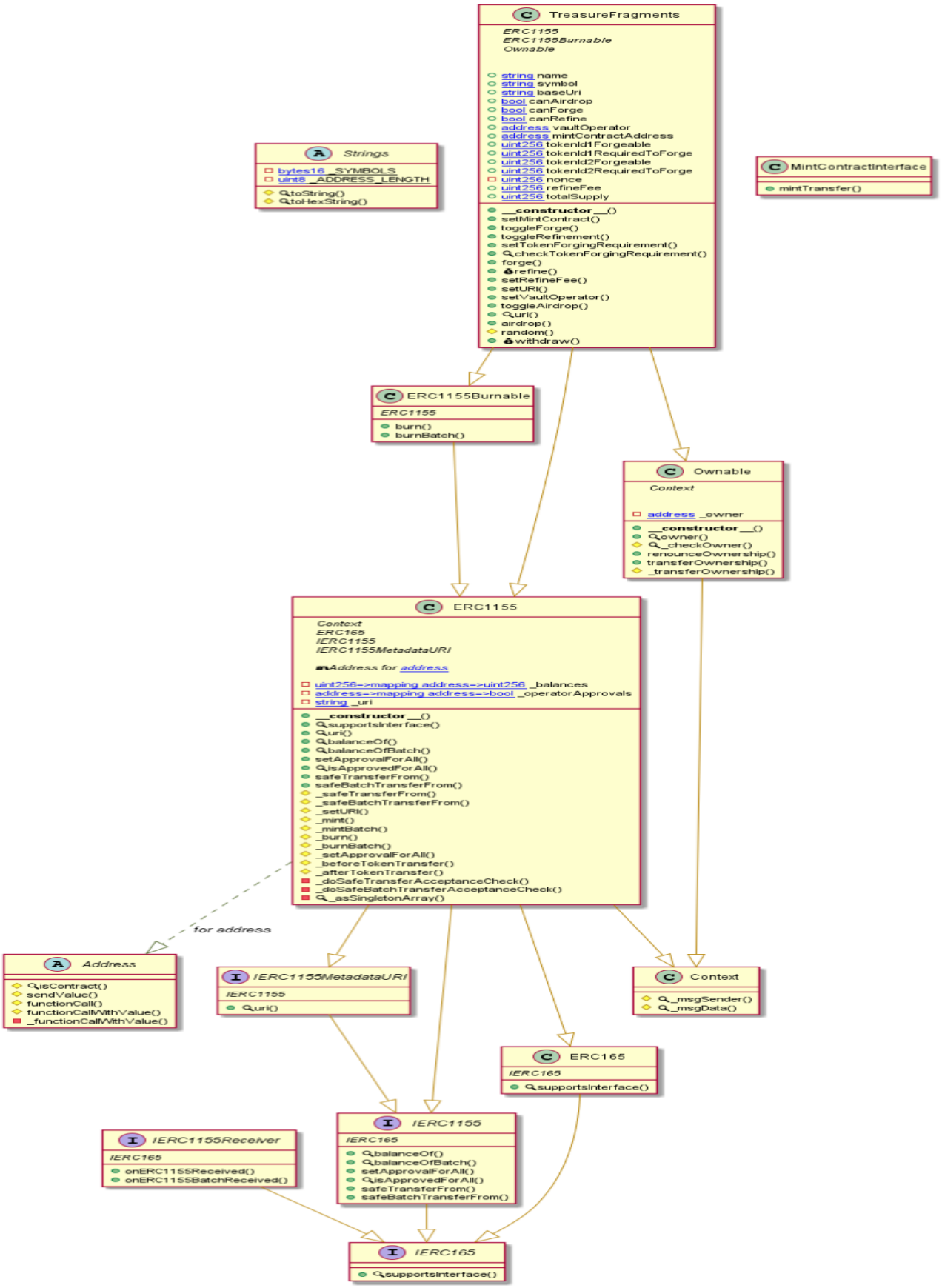
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

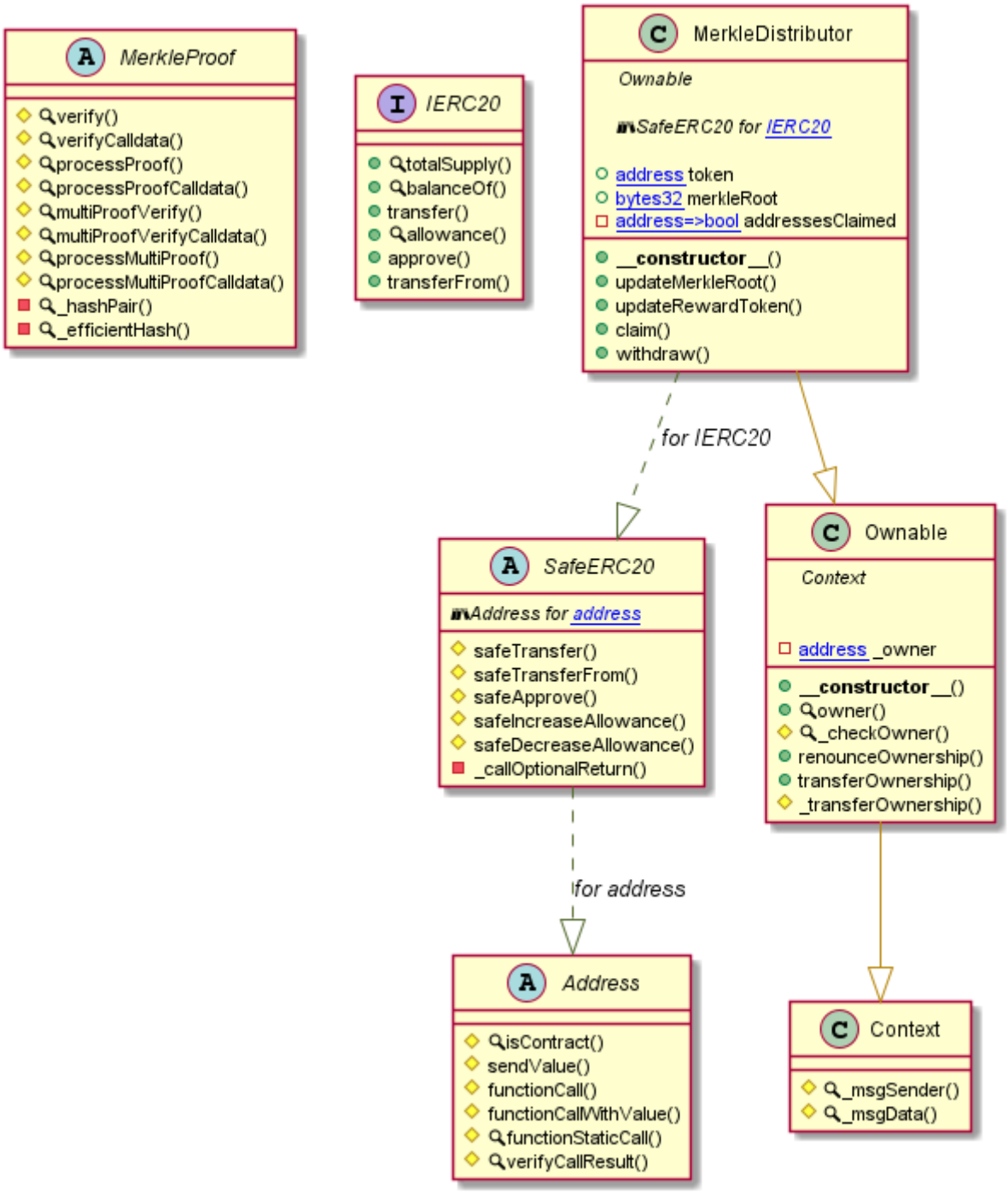
Appendix

Code Flow Diagram - Another World DAO

TreasureFragments Diagram



MerkleDistributor Diagram



Slither Results Log

Slither log >> TreasureFragments.sol

```
TreasureFragments.setTokenForgingRequirement(uint256,uint256,uint256,uint256) (TreasureFragments.sol#619-629) should emit an e
vent for:
        - tokenId1Forgeable = newTokenId1Forgeable (TreasureFragments.sol#625)
        - tokenId1RequiredToForge = newTokenId1RequiredToForge (TreasureFragments.sol#626)
        - tokenId2Forgeable = newTokenId2Forgeable (TreasureFragments.sol#627)
        - tokenId2RequiredToForge = newTokenId2RequiredToForge (TreasureFragments.sol#628)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

TreasureFragments.setMintContract(address).contractAddress (TreasureFragments.sol#607) lacks a zero-check on :
                - mintContractAddress = contractAddress (TreasureFragments.sol#608)
TreasureFragments.setVaultOperator(address).newOperator (TreasureFragments.sol#737) lacks a zero-check on :
                - vaultOperator = newOperator (TreasureFragments.sol#738)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).response (TreasureFragments.so
l#464)' in ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (TreasureFragments.sol#455-47
4) potentially used before declaration: response != IERC1155Receiver.onERC1155Received.selector (TreasureFragments.sol#465)
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).reason (TreasureFragments.sol#
468)' in ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (TreasureFragments.sol#455-474)
 potentially used before declaration: revert(string)(reason) (TreasureFragments.sol#469)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).response (TreasureFra
gments.sol#486)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (TreasureFr
agments.sol#476-497) potentially used before declaration: response != IERC1155Receiver.onERC1155BatchReceived.selector (Treasu
reFragments.sol#488)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).reason (TreasureFragm
ents.sol#491)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (TreasureFrag
ments.sol#476-497) potentially used before declaration: revert(string)(reason) (TreasureFragments.sol#492)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in TreasureFragments.airdrop(address,uint256) (TreasureFragments.sol#752-762):
        External calls:
        - _mint(account,id,1,) (TreasureFragments.sol#755)
                - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (TreasureFragments.sol#464-472)
        State variables written after the call(s):
        - totalSupply ++ (TreasureFragments.sol#760)
Reentrancy in TreasureFragments.forge() (TreasureFragments.sol#645-672):
```

```
Reentrancy in TreasureFragments.refine(uint256,uint256) (TreasureFragments.sol#674-727):
        External calls:
        - _mint(msg.sender,mintedId,1,) (TreasureFragments.sol#691)
                - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (TreasureFragments.sol#464-472)
        - _mint(msg.sender,tokenId,amount,) (TreasureFragments.sol#698)
                - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (TreasureFragments.sol#464-472)
        - _mint(msg.sender,mintedId_scope_0,1,) (TreasureFragments.sol#708)
                - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (TreasureFragments.sol#464-472)
        - _mint(msg.sender,mintedId_scope_1,mintedAmount,) (TreasureFragments.sol#719)
                - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (TreasureFragments.sol#464-472)
        Event emitted after the call(s):
        - RefinementMint(msg.sender,mintedId_scope_1,mintedAmount) (TreasureFragments.sol#720)
        - TransferSingle(operator,address(0),to,id,amount) (TreasureFragments.sol#356)
                - _mint(msg.sender,mintedId_scope_1,mintedAmount,) (TreasureFragments.sol#719)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

TreasureFragments.refine(uint256,uint256) (TreasureFragments.sol#674-727) uses timestamp for comparisons
        Dangerous comparisons:
        - random() % 1000 < 500 (TreasureFragments.sol#686)
        - random() % 1000 < 250 (TreasureFragments.sol#706)
        - random() % 1000 < 100 (TreasureFragments.sol#716)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Strings.toString(uint256) (TreasureFragments.sol#9-23) uses assembly
        - INLINE ASM (TreasureFragments.sol#12-13)
        - INLINE ASM (TreasureFragments.sol#16-18)
Address.isContract(address) (TreasureFragments.sol#48-55) uses assembly
        - INLINE ASM (TreasureFragments.sol#51-53)
Address._functionCallWithValue(address,bytes,uint256,string) (TreasureFragments.sol#94-116) uses assembly
        - INLINE ASM (TreasureFragments.sol#108-111)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address._functionCallWithValue(address,bytes,uint256,string) (TreasureFragments.sol#94-116) is never used and should be remove
d
Address.functionCall(address,bytes) (TreasureFragments.sol#64-66) is never used and should be removed
Address.functionCall(address,bytes,string) (TreasureFragments.sol#68-74) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (TreasureFragments.sol#76-82) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (TreasureFragments.sol#84-92) is never used and should be removed
Address.sendValue(address,uint256) (TreasureFragments.sol#57-62) is never used and should be removed
Context._msgData() (TreasureFragments.sol#191-193) is never used and should be removed
ERC1155._mintBatch(address,uint256[],uint256[],bytes) (TreasureFragments.sol#363-385) is never used and should be removed
Strings.toHexString(address) (TreasureFragments.sol#42-44) is never used and should be removed
Strings.toHexString(uint256) (TreasureFragments.sol#25-28) is never used and should be removed
Strings.toHexString(uint256,uint256) (TreasureFragments.sol#30-40) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.17 (TreasureFragments.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.1
2/0.7.6/0.8.16
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Low level call in Address.sendValue(address,uint256) (TreasureFragments.sol#57-62):
        - (success) = recipient.call{value: amount}() (TreasureFragments.sol#60)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (TreasureFragments.sol#94-116):
        - (success,returndata) = target.call{value: weiValue}(data) (TreasureFragments.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Variable TreasureFragments.refine(uint256,uint256).mintedId_scope_0 (TreasureFragments.sol#707) is too similar to TreasureFrag
ments.refine(uint256,uint256).mintedId_scope_1 (TreasureFragments.sol#717)
Variable TreasureFragments.setTokenForgingRequirement(uint256,uint256,uint256,uint256).newTokenId1Forgeable (TreasureFragments
.sol#620) is too similar to TreasureFragments.setTokenForgingRequirement(uint256,uint256,uint256,uint256).newTokenId2Forgeable
 (TreasureFragments.sol#622)
Variable TreasureFragments.setTokenForgingRequirement(uint256,uint256,uint256,uint256).newTokenId1RequiredToForge (TreasureFra
gments.sol#621) is too similar to TreasureFragments.setTokenForgingRequirement(uint256,uint256,uint256,uint256).newTokenId2Req
uiredToForge (TreasureFragments.sol#623)
Variable TreasureFragments.tokenId1Forgeable (TreasureFragments.sol#590) is too similar to TreasureFragments.tokenId2Forgeable
 (TreasureFragments.sol#592)
Variable TreasureFragments.tokenId1RequiredToForge (TreasureFragments.sol#591) is too similar to TreasureFragments.tokenId2Req
uiredToForge (TreasureFragments.sol#593)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

TreasureFragments.name (TreasureFragments.sol#576) should be immutable
TreasureFragments.symbol (TreasureFragments.sol#577) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
TreasureFragments.sol analyzed (13 contracts with 84 detectors), 59 result(s) found
```

Slither log >> MerkleDistributor.sol

```
MerkleDistributor.updateRewardToken(address).token_ (MerkleDistributor.sol#367) lacks a zero-check on :
                - token = token_ (MerkleDistributor.sol#368)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in MerkleDistributor.claim(uint256,bytes32[]) (MerkleDistributor.sol#372-382):
        External calls:
        - IERC20(token).safeTransfer(msg.sender,amount) (MerkleDistributor.sol#380)
        Event emitted after the call(s):
        - Claimed(msg.sender,amount) (MerkleDistributor.sol#381)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

MerkleProof._efficientHash(bytes32,bytes32) (MerkleDistributor.sol#117-123) uses assembly
        - INLINE ASM (MerkleDistributor.sol#118-122)
Address.verifyCallResult(bool,bytes,string) (MerkleDistributor.sol#213-231) uses assembly
        - INLINE ASM (MerkleDistributor.sol#223-226)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
Address.functionCall(address,bytes) (MerkleDistributor.sol#162-164) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MerkleDistributor.sol#174-180) is never used and should be removed
Address.functionStaticCall(address,bytes) (MerkleDistributor.sol#195-197) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (MerkleDistributor.sol#199-208) is never used and should be removed
Address.sendValue(address,uint256) (MerkleDistributor.sol#155-160) is never used and should be removed
Context._msgData() (MerkleDistributor.sol#303-305) is never used and should be removed
MerkleProof.multiProofVerify(bytes32[],bool[],bytes32,bytes32[]) (MerkleDistributor.sol#29-36) is never used and should be rem
oved
MerkleProof.multiProofVerifyCalldata(bytes32[],bool[],bytes32,bytes32[]) (MerkleDistributor.sol#38-45) is never used and shoul
d be removed
MerkleProof.processMultiProof(bytes32[],bool[],bytes32[]) (MerkleDistributor.sol#47-78) is never used and should be removed
MerkleProof.processMultiProofCalldata(bytes32[],bool[],bytes32[]) (MerkleDistributor.sol#80-111) is never used and should be r
emoved
MerkleProof.processProofCalldata(bytes32[],bytes32) (MerkleDistributor.sol#21-27) is never used and should be removed
MerkleProof.verifyCalldata(bytes32[],bytes32,bytes32) (MerkleDistributor.sol#9-11) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (MerkleDistributor.sol#255-265) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (MerkleDistributor.sol#276-287) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (MerkleDistributor.sol#267-274) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (MerkleDistributor.sol#246-253) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version=0.8.17 (MerkleDistributor.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.1
2/0.7.6/0.8.16
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (MerkleDistributor.sol#155-160):
        - (success) = recipient.call{value: amount}() (MerkleDistributor.sol#158)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (MerkleDistributor.sol#182-193):
        - (success,returndata) = target.call{value: value}(data) (MerkleDistributor.sol#191)
Low level call in Address.functionStaticCall(address,bytes,string) (MerkleDistributor.sol#199-208):
        - (success,returndata) = target.staticcall(data) (MerkleDistributor.sol#206)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
MerkleDistributor.sol analyzed (7 contracts with 84 detectors), 25 result(s) found
```

Solidity Static Analysis

TreasureFragments.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in TreasureFragments.forge(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 119:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 254:24:

Low level calls:

Use of "send": "send" does not throw an exception when not successful, make sure you deal with the failure case accordingly. Use "transfer" whenever failure of the ether transfer should rollback the whole transaction. Note: if you "send/transfer" ether to a contract the fallback function is called, the callees fallback function is very limited due to the limited amount of gas provided by "send/transfer". No state changes are possible but the callee can log the event or revert the transfer. "send/transfer" is syntactic sugar for a "call" to the fallback function with 2300 gas and a specified ether value.

more

Pos: 264:16:

Gas & Economy

Gas costs:

Gas requirement of function TreasureFragments.withdraw is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 263:4:

Email: audit@EtherAuthority.io

Miscellaneous

Constant/View/Pure functions:

MintContractInterface.mintTransfer(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 22:4:

Similar variable names:

TreasureFragments.forge() : Variables have very similar names "tokenId1RequiredToForge" and "tokenId2RequiredToForge". Note: Modifiers are currently not considered by this static analysis.
Pos: 142:43:

No return:

MintContractInterface.mintTransfer(address): Defines a return type but never explicitly returns a value.
Pos: 22:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 264:8:

MerkleDistributor.sol

Gas & Economy

Gas costs:

Gas requirement of function MerkleDistributor.updateMerkleRoot is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 29:4:

Gas costs:

Gas requirement of function MerkleDistributor.withdraw is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 51:4:

Miscellaneous

Constant/View/Pure functions:

MerkleDistributor.withdraw() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 51:4:

Similar variable names:

MerkleDistributor.updateRewardToken(address) : Variables have very similar names "token" and "token_". Note: Modifiers are currently not considered by this static analysis.
Pos: 36:32:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 43:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 45:8:

Solhint Linter

TreasureFragments.sol

```
TreasureFragments.sol:137:9: Error: Parse error: mismatched input ';'
expecting '('
TreasureFragments.sol:139:18: Error: Parse error: missing ';' at '{'
TreasureFragments.sol:167:56: Error: Parse error: mismatched input
';' expecting '('
TreasureFragments.sol:169:22: Error: Parse error: missing ';' at '{'
TreasureFragments.sol:174:60: Error: Parse error: mismatched input
';' expecting '('
TreasureFragments.sol:176:22: Error: Parse error: missing ';' at '{'
TreasureFragments.sol:185:56: Error: Parse error: mismatched input
';' expecting '('
TreasureFragments.sol:187:22: Error: Parse error: missing ';' at '{'
TreasureFragments.sol:197:67: Error: Parse error: mismatched input
';' expecting '('
TreasureFragments.sol:199:22: Error: Parse error: missing ';' at '{'
TreasureFragments.sol:239:33: Error: Parse error: mismatched input
';' expecting '('
TreasureFragments.sol:241:18: Error: Parse error: missing ';' at '{'
TreasureFragments.sol:247:18: Error: Parse error: missing ';' at '{'
```

MerkleDistributor.sol

```
MerkleDistributor.sol:8:1: Error: Compiler version =0.8.17 does not
satisfy the r semver requirement
MerkleDistributor.sol:26:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
MerkleDistributor.sol:26:19: Error: Code contains empty blocks
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io