

SMART CONTRACT

Security Audit Report

Project: Decentraland MANA
Website: decentraland.org
Platform: Ethereum
Language: Solidity
Date: May 9th, 2024

Table of contents

Introduction	4
Project Background	4
Audit Scope	5
Claimed Smart Contract Features	6
Audit Summary	7
Technical Quick Stats	8
Business Risk Analysis	9
Code Quality	10
Documentation	10
Use of Dependencies	10
AS-IS overview	11
Severity Definitions	13
Audit Findings	14
Conclusion	21
Our Methodology	22
Disclaimers	24
Appendix	
• Code Flow Diagram	25
• Slither Results Log	26
• Solidity static analysis	27
• Solhint Linter	29

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.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the smart contracts of MANA from decentraland.org were audited. 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 9th, 2024.

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

- Decentraland MANA Token smart contract is an ERC20-compliant token contract for the Decentraland MANA token. Here's a breakdown of its key features:
 - **Token Standards:** The contract implements the ERC20 token standard, providing basic token functionalities such as transfer, allowance, and approval.
 - **BurnableToken:** Allows tokens to be burned (destroyed).
 - **PausableToken:** Allows the contract owner to pause and unpaue token transfers.
 - **MintableToken:** Allows the contract owner to mint (create) new tokens.
- Overall, this contract provides standard ERC20 token functionalities with additional features for token burning, pausing transfers, and minting new tokens. It also implements security measures such as modifiers to control access to critical functions.

Audit scope

Name	Code Review and Security Analysis Report for Decentraland MANA Token Smart Contract
Platform	Ethereum
File	MANAToken.sol
Smart Contract Code	0x0f5d2fb29fb7d3cfee444a200298f468908cc942
Audit Date	May 9th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: <ul style="list-style-type: none">• Name: Decentraland MANA• Symbol: MANA• Decimals: 18	YES, This is valid.
Owner control: <ul style="list-style-type: none">• Mint new tokens by the owner.• The owner can stop minting new tokens.• The owner can trigger a stopped state.• The owner can return to a normal state.• The current owner can transfer the ownership.	YES, This is valid. We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.

Audit Summary

According to the standard audit assessment, the Customer's solidity-based smart contracts are "**Secured**". Also, these contracts contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a 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. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 1 low, and 7 very low level issues.

Investor Advice: A 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.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	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	Moderated
	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	Moderated
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

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	Not Detected
● Fee Check	No
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	No
● Pause Transfer?	No
● Max Tax?	No
● Is it Anti-whale?	No
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	Not Detected
● Blacklist Check	No
● Can Mint?	No
● Is it Proxy?	No
● Can Take Ownership?	Yes
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Decentraland MANA Token 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 Decentraland MANA Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Decentraland MANA Token smart contract code in the form of an [Etherscan](#) web link.

As mentioned above, code parts are well commented on. and 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 used in this smart contract infrastructure that is based on well known industry standard open source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	burn	write	Passed	No Issue
3	burn	write	Deprecated event invocation syntax burn function	Refer Audit Findings
4	transfer	write	Missing explicit visibility declaration in functions	Refer Audit Findings
5	transferFrom	write	Missing explicit visibility declaration in functions	Refer Audit Findings
6	canMint	modifier	Missing required error message	Refer Audit Findings
7	mint	write	Mintable, Missing explicit visibility declaration in functions Centralized risk	Refer Audit Findings
8	finishMinting	write	Missing explicit visibility declaration in functions Centralized risk	Refer Audit Findings
9	transferFrom	write	Passed	No Issue
10	approve	write	Passed	No Issue
11	allowance	write	Passed	No Issue
12	transfer	write	Missing explicit visibility declaration in functions	Refer Audit Findings
13	balanceOf	write	Missing explicit visibility declaration in functions	Refer Audit Findings
14	allowance	write	Missing explicit visibility declaration in functions	Refer Audit Findings
15	transferFrom	write	Missing explicit visibility declaration in functions	Refer Audit Findings
16	approve	write	Missing explicit visibility declaration in functions	Refer Audit Findings
17	whenNotPaused	modifier	Missing required error message	Refer Audit Findings
18	whenPaused	modifier	Missing required error message	Refer Audit Findings

19	pause	write	Missing explicit visibility declaration in functions, Centralized risk	Refer Audit Findings
20	unpause	write	Missing explicit visibility declaration in functions, Centralized risk	Refer Audit Findings
21	Ownable	write	Passed	No Issue
22	onlyOwner	modifier	Missing required error message	Refer Audit Findings
23	transferOwnership	write	Missing explicit visibility declaration in functions, Centralized risk, Critical operation lacks event log	Refer Audit Findings
24	balanceOf	write	Missing explicit visibility declaration in functions	Refer Audit Findings
25	transfer	write	Missing explicit visibility declaration in functions	Refer 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.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Mintable: [BasicToken.sol](#)

```
function mint(address _to, uint256 _amount) onlyOwner canMint returns (bool) {
    totalSupply = totalSupply.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    Mint(_to, _amount);
    return true;
}
```

A large amount of this token can be minted by a private wallet or contract.

Resolution: Implement a cap or limit on the total supply that can be minted by a private wallet or contract, ensuring responsible token creation and preventing excessive minting.

Very Low / Informational / Best practices:

(1) Use the latest solidity version: [ERC20Basic.sol](#)

```
pragma solidity ^0.4.11;
```

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use 0.8.22 which is the latest version.

(2) Missing explicit visibility declaration in functions:

PausableToken.sol

```
function transfer(address _to, uint _value) whenNotPaused returns (bool) { infinite gas  
    return super.transfer(_to, _value);  
}  
  
function transferFrom(address _from, address _to, uint _value) whenNotPaused returns (bool) {  
    return super.transferFrom(_from, _to, _value);  
}
```

ERC20Basic.sol

```
function balanceOf(address who) constant returns (uint256);  
function transfer(address to, uint256 value) returns (bool);
```


Ownable.sol


```
function transferOwnership(address newOwner) onlyOwner { 20757 gas  
    if (newOwner != address(0)) {  
        owner = newOwner;  
    }  
}
```

Pausable.sol

```
function pause() onlyOwner whenNotPaused returns (bool) { 21914 gas  
    paused = true;  
    Pause();  
    return true;  
}  
  
/**  
 * @dev called by the owner to unpause, returns to normal state  
 */  
function unpause() onlyOwner whenPaused returns (bool) { 21867 gas  
    paused = false;  
    Unpause();  
    return true;  
}
```


BasicToken.sol

```
function transfer(address _to, uint256 _value) returns (bool) {  infinite gas
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(msg.sender, _to, _value);
    return true;
}
```


```
function balanceOf(address _owner) constant returns (uint256 balance) {  531 gas
    return balances[_owner];
}
```

MintableToken.sol

```
function mint(address _to, uint256 _amount) onlyOwner canMint returns (bool) {  infinite gas
    totalSupply = totalSupply.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    Mint(_to, _amount);
    return true;
}


/**
 * @dev Function to stop minting new tokens.
 * @return True if the operation was successful.
 */
function finishMinting() onlyOwner returns (bool) [ 21707 gas
    mintingFinished = true;
    MintFinished();
    return true;
]
```

StandardToken.sol

```
function transferFrom(address _from, address _to, uint256 _value) returns (bool) {  infinite gas
    var _allowance = allowed[_from][msg.sender];

    // Check is not needed because sub(_allowance, _value) will already throw if this condition is not met
    // require (_value <= _allowance);

    balances[_to] = balances[_to].add(_value);
    balances[_from] = balances[_from].sub(_value);
    allowed[_from][msg.sender] = _allowance.sub(_value);
    Transfer(_from, _to, _value);
    return true;
}
```

```
function allowance(address _owner, address _spender) constant returns (uint256 remaining) {  759 gas
    return allowed[_owner][_spender];
}
```



```
function approve(address _spender, uint256 _value) returns (bool) { 22732 gas

    // To change the approve amount you first have to reduce the addresses`
    // allowance to zero by calling `approve(_spender, 0)` if it is not
    // already 0 to mitigate the race condition described here:
    // https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
    require((_value == 0) || (allowed[msg.sender][_spender] == 0));

    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
}
```

Functions in the contract lack explicit visibility modifiers, causing potential ambiguity regarding access levels.

Resolution: Review and add clear visibility modifiers (e.g., "public," "internal") to each function, enhancing code clarity and adherence to best practices.

(3) Missing required error message:

Ownable.sol

```
modifier onlyOwner() {
    require(msg.sender == owner);
    _;
}
```

Pausable.sol

```
modifier whenNotPaused() {
    require(!paused);
    _;
}

/**
 * @dev modifier to allow actions only when the contract IS NOT paused
 */
modifier whenPaused {
    require(paused);
    _;
}
```

MintableToken.sol

```
modifier canMint() {  
    require(!mintingFinished);  
    _;  
}
```

There is no error message set in the required condition.

Resolution: We suggest setting relevant error messages to identify the failure of the transaction.

(4) Centralized risk:

Ownable.sol

The onlyOwner can call the transferOwnership function.

Pausable.sol

The onlyOwner can call pause and unpause functions.

MintableToken.sol

The onlyOwner can call mint and finishMinting functions.

Resolution: To make the smart contract 100% decentralized. We suggest renouncing ownership of the smart contract once its function is completed.

(5) Critical operation lacks event log: [Ownable.sol](#)

```
function transferOwnership(address newOwner) onlyOwner { 20757 gas  
    if (newOwner != address(0)) {  
        owner = newOwner;  
    }  
}
```

This contract is missing useful events with regard to access control changes.

Resolution: Introduce access control events to enhance transparency for access control changes, facilitating effective monitoring and auditing of role assignments and revocations.

(6) Explicit Visibility for State Variables:

BasicToken.sol

```
mapping(address => uint256) balances;
```

StandardToken.sol

```
mapping (address => mapping (address => uint256)) allowed;
```

The warning is related to the visibility of state variables in your Solidity code.

Resolution: We recommend updating the code to explicitly mark the visibility of state variables using the internal or public keyword, depending on the intended visibility.

(7) Deprecated event invocation syntax burn function: [BurnableToken.sol](#)

The code uses deprecated event invocation syntax without the "emit" prefix, which may lead to compatibility issues.

Resolution: Update the code to include the "emit" prefix for event invocations, ensuring alignment with current Solidity standards and maintaining code compatibility.

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:

MintableToken.sol

- mint: Mint new tokens by the owner.
- finishMinting: The owner can stop minting new tokens.

Pausable.sol

- pause: The owner can trigger a stopped state.
- unpaue: The owner can return to a normal state.

Ownable.sol

- transfer ownership: Allows the current owner to transfer control of the contract to a newOwner by the current owner.

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

Conclusion

We were given a contract code in the form of [Etherscan](#) web links. And we have used all possible tests based on given objects as files. We observed 1 low and 7 informational issues in the smart contracts. And those issues are not critical. So, **it's good to go for the production.**

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.

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

Security state of the reviewed smart 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).

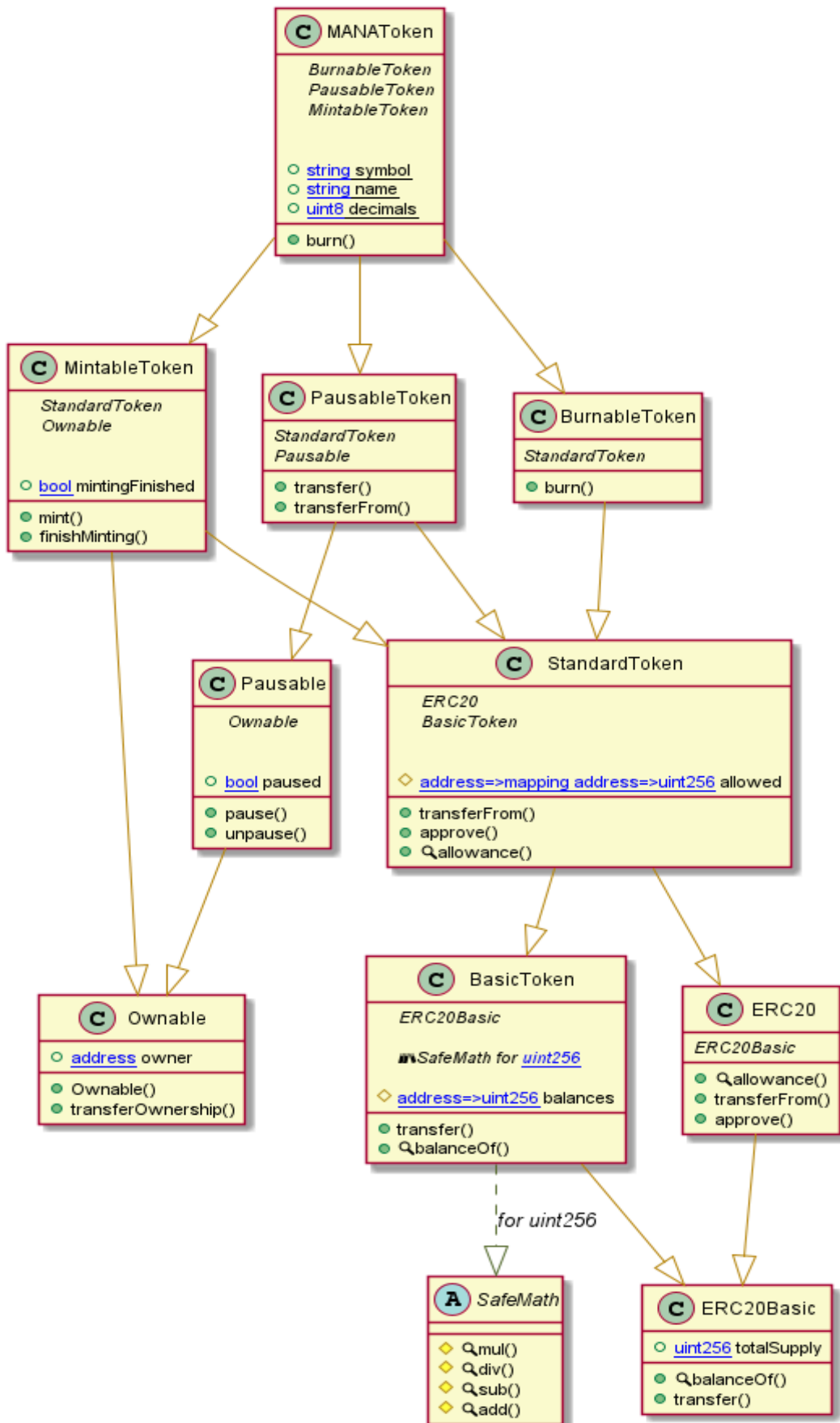
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 - Decentraland MANA Token



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> MANAToken.sol

```
Ownable.transferOwnership(address) (MANAToken.sol#40-44) should emit an event for:
- owner = newOwner (MANAToken.sol#42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

SafeMath.div(uint256,uint256) (MANAToken.sol#104-109) is never used and should be removed
SafeMath.mul(uint256,uint256) (MANAToken.sol#98-102) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.4.11 (MANAToken.sol#5) allows old versions
solc-0.4.11 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter BasicToken.transfer(address,uint256)._to (MANAToken.sol#133) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._value (MANAToken.sol#133) is not in mixedCase
Parameter BasicToken.balanceOf(address)._owner (MANAToken.sol#145) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (MANAToken.sol#162) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (MANAToken.sol#162) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (MANAToken.sol#162) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._spender (MANAToken.sol#180) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (MANAToken.sol#180) is not in mixedCase
Parameter StandardToken.allowance(address,address)._owner (MANAToken.sol#199) is not in mixedCase
Parameter StandardToken.allowance(address,address)._spender (MANAToken.sol#199) is not in mixedCase
Parameter MintableToken.mint(address,uint256)._to (MANAToken.sol#223) is not in mixedCase
Parameter PausableToken.transfer(address,uint256)._to (MANAToken.sol#243) is not in mixedCase
Parameter PausableToken.transfer(address,uint256)._value (MANAToken.sol#243) is not in mixedCase
Parameter PausableToken.transferFrom(address,address,uint256)._from (MANAToken.sol#247) is not in mixedCase
Parameter PausableToken.transferFrom(address,address,uint256)._to (MANAToken.sol#247) is not in mixedCase
Parameter PausableToken.transferFrom(address,address,uint256)._value (MANAToken.sol#247) is not in mixedCase
Parameter BurnableToken.burn(uint256)._value (MANAToken.sol#260) is not in mixedCase
Parameter MANAToken.burn(uint256)._value (MANAToken.sol#279) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
MANAToken.sol analyzed (11 contracts with 84 detectors), 24 result(s) found
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

MANAToken.sol

Gas costs:

Gas requirement of function MANAToken.transfer 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: 133:2:

Gas costs:

Gas requirement of function MANAToken.mint 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: 223:2:

Gas costs:

Gas requirement of function MANAToken.burn 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: 260:4:

Constant/View/Pure functions:

PausableToken.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 243:2:

Similar variable names:

BasicToken.balanceOf(address) : Variables have very similar names "balance" and "balances". Note: Modifiers are currently not considered by this static analysis.

Pos: 146:11:

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: 112:4:

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: 261:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 100:21:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

MANAToken.sol

```
Compiler version ^0.4.11 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:4
Explicitly mark visibility in function
Pos: 3:8
Explicitly mark visibility in function
Pos: 3:9
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 3:21
Provide an error message for require
Pos: 5:30
Explicitly mark visibility in function
Pos: 3:39
Provide an error message for require
Pos: 5:58
Provide an error message for require
Pos: 5:66
Explicitly mark visibility in function
Pos: 3:73
Explicitly mark visibility in function
Pos: 3:82
Explicitly mark visibility in function
Pos: 3:90
Explicitly mark visibility in function
Pos: 3:91
Explicitly mark visibility in function
Pos: 3:92
Explicitly mark visibility of state
Pos: 3:125
Explicitly mark visibility in function
Pos: 3:132
Explicitly mark visibility in function
Pos: 3:144
Explicitly mark visibility of state
Pos: 3:152
Explicitly mark visibility in function
Pos: 3:161
Explicitly mark visibility in function
Pos: 3:179
Provide an error message for require
Pos: 5:185
Explicitly mark visibility in function
```

```
Pos: 3:198
Provide an error message for require
Pos: 5:212
Explicitly mark visibility in function
Pos: 3:222
Explicitly mark visibility in function
Pos: 3:233
Explicitly mark visibility in function
Pos: 3:242
Explicitly mark visibility in function
Pos: 3:246
Provide an error message for require
Pos: 9:260
Constant name must be in capitalized SNAKE_CASE
Pos: 5:272
Constant name must be in capitalized SNAKE_CASE
Pos: 5:274
Constant name must be in capitalized SNAKE_CASE
Pos: 5:276
Visibility modifier must be first in list of modifiers
Pos: 49:278
```

Software analysis result:

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



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io