Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: MoonLock
Website: https://planetmoon.io
Platform: Ethereum
Language: Solidity
Date: May 29th, 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 15
Our Methodology 16
Disclaimers 18
Appendix

- Code Flow Diagram 19
- Slither Results Log 20
- Solidity static analysis 21
- Solhint Linter 23

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 MoonLock to perform the Security audit of the MoonLock 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 29th, 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 MoonLock Contracts handle multiple contracts, and all contracts have different functions.
    - MoonLock: It handles proposals, votes, and updates social data.
    - MoonLockFactory: It handles updating membership management and withdrawing funds functionality.
- MoonLock is a voting smart contract that has functions like mint, start proposals, finalize proposals, update socials, receive funds, withdraw funds, etc.
- There is one smart contract that was included in the audit scope. And there were some standard library codes, such as OpenZepelin, that were excluded. Because those standard library codes are considered time-tested and community-audited, we can safely ignore them.

Email: audit@EtherAuthority.io

Audit scope

Name	Code Review and Security Analysis Report for MoonLock Smart Contracts
Platform	**Ethereum / Solidity**
File 1	MoonLock.sol
File 1 MD5 Hash	89DE746E9C2420860807F7B9F6C1AF8B
Updated File 1 MD5 Hash	9AA43367A8AE360DFED20E30D59F9447
Audit Date	May 29th, 2023
Revised Audit Date	June 13th, 2023

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
MoonLock.sol **<u>Owner has control over following functions:</u>** ● Added a Start a new unlock proposal. ● Cancel proposal. ● Finalize the latest unlock proposal. ● Update the social data. ● Update / remove the executor.	**YES, This is valid.**
MoonLockFactory.sol **<u>Owner has control over following functions:</u>** ● Set the membership manager addresses. ● Set the fees. ● Withdraw funds.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“Secured”**. Also, these contracts do 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 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, 1 high, 0 medium and 0 low and some very low level issues.
All the issues have been resolved 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.

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 1 smart contract file. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the MoonLock 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 MoonLock Protocol.

The MoonLock 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 MoonLock Protocol smart contract code in the form of a file. 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.

Another source of information was its official website: https://www.planetmoon.io/ which provided rich information about the project architecture and tokenomics.

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

MoonLock.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	isOwnershipLocked	read	Passed	No Issue
9	getProposals	read	Passed	No Issue
10	getLatestProposal	read	Passed	No Issue
11	startProposal	write	access only Effective Owner	No Issue
12	cancelProposal	write	access only Effective Owner	No Issue
13	castVote	write	Passed	No Issue
14	finalizeProposal	write	access only Effective Owner	No Issue
15	updateSocials	write	access only Effective Owner	No Issue
16	onlyEffectiveOwner	modifier	Passed	No Issue
17	getMoonLockInfo	read	Passed	No Issue
18	updateExecutor	write	access only Effective Owner	No Issue
19	removeExecutor	write	access only Effective Owner	No Issue
20	executor	read	Passed	No Issue
21	effetiveOwner	read	Passed	No Issue

MoonLockFactory.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	isOwnershipLocked	read	Passed	No Issue
9	createMoonLock	write	Passed	No Issue
10	getMoonLocks	read	Passed	No Issue
11	updateMembershipManager	write	access only Owner	No Issue
12	receive	external	Passed	No Issue
13	withdrawFunds	write	access only Owner	No Issue
14	updateFee	write	access only Owner	No Issue

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

No Low severity vulnerabilities were found in the contract code.

Very Low / Informational / Best practices:

No Informational severity vulnerabilities were found in the contract code.

Email: audit@EtherAuthority.io

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:

MoonLock.sol

- startProposal: Start a new unlock proposal added by the effective owner.
- cancelProposal: It is a conditional statement expressing that once a proposal is initiated, it cannot be canceled by anyone.
- finalizeProposal: Finalize the latest unlock proposal by the effective owner.
- updateSocials: Update the socials data by the effective owner.
- updateExecutor: Update the backup owner by the effective owner.
- removerExecutor: Remove previously set executor by the effective owner.

MoonLockFactory.sol

- updateMembershipManager: Membership manager address can be set by the owner.
- withdrawFunds: Withdraw funds by the owner.
- updateFee: Fee can be updated by the owner.

Conclusion

We were given a contract code in the form of a file. And we have used all possible tests based on given objects as files. We had observed 1 high severity issue and some informational issues in the smart contracts. All the issues have been resolved in the revised code. **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.

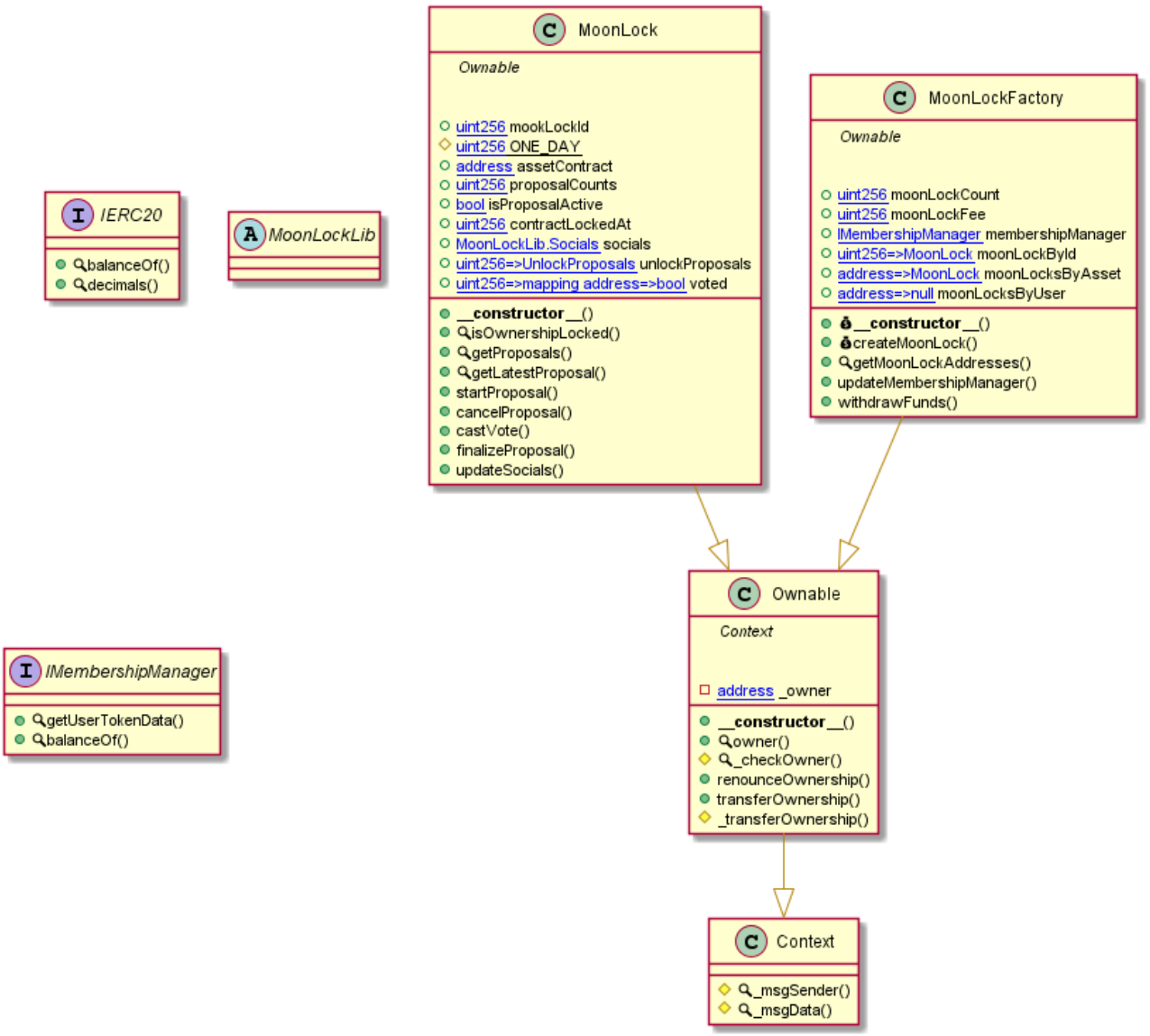
Technical Disclaimer

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 - MoonLock

MoonLock Diagram



Slither Results Log

Slither log >> MoonLock.sol

```
Reentrancy in MoonLock.finalizeProposal() (MoonLock.sol#295-324):
        External calls:
        - Ownable(assetContract).transferOwnership(owner()) (MoonLock.sol#314)
        Event emitted after the call(s):
        - OwnershipReleased(p.id) (MoonLock.sol#316)
        - ProposalFinalized(p.id,Status.SUCCESSFUL) (MoonLock.sol#315)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

MoonLock.startProposal(string,uint256,uint8) (MoonLock.sol#194-224) uses timestamp for comparisons
        Dangerous comparisons:
        - _startingAt <= block.timestamp (MoonLock.sol#205)
MoonLock.cancelProposal() (MoonLock.sol#226-242) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > p.startingAt (MoonLock.sol#234)
MoonLock.castVote(MoonLock.Vote) (MoonLock.sol#244-293) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp <= p.startingAt (MoonLock.sol#257)
        - block.timestamp > p.startingAt + (p.durationInDays * ONE_DAY) (MoonLock.sol#279)
MoonLock.finalizeProposal() (MoonLock.sol#295-324) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < p.startingAt + p.durationInDays * ONE_DAY (MoonLock.sol#304)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Context._msgData() (MoonLock.sol#10-12) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.9 (MoonLock.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter MoonLock.startProposal(string,uint256,uint8)._description (MoonLock.sol#194) is not in mixedCase
Parameter MoonLock.startProposal(string,uint256,uint8)._startingAt (MoonLock.sol#194) is not in mixedCase
Parameter MoonLock.startProposal(string,uint256,uint8)._durationInDays (MoonLock.sol#194) is not in mixedCase
Parameter MoonLock.updateSocials(MoonLockLib.Socials)._socials (MoonLock.sol#326) is not in mixedCase
Parameter MoonLockFactory.createMoonLock(address,MoonLockLib.Socials)._socials (MoonLock.sol#350) is not in mixedCase
Parameter MoonLockFactory.updateMembershipManager(address)._membershipManager (MoonLock.sol#398) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

MoonLockFactory.moonLockFee (MoonLock.sol#337) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

MoonLock.assetContract (MoonLock.sol#131) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
MoonLock.sol analyzed (7 contracts with 84 detectors), 17 result(s) found
```

Solidity Static Analysis

MoonLock.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in MoonLock.(uint256,address,address,struct MoonLockLib.Socials): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 88:4:

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: 246:12:

Gas & Economy

Gas costs:

Gas requirement of function MoonLock.getLatestProposal 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: 130:4:

Gas costs:

Gas requirement of function MoonLockFactory.getMoonLockAddresses 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: 316:4:

Email: audit@EtherAuthority.io

Miscellaneous

Constant/View/Pure functions:

MoonLock.getProposals(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 106:4:

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: 350:8:

Solhint Linter

MoonLock.sol

```
MoonLock.sol:7:6: Error: Parse error: missing 'constant' at
'CONTRACT_IS_NOT_LOCKED'
MoonLock.sol:7:28: Error: Parse error: missing '=' at '('
MoonLock.sol:8:6: Error: Parse error: missing 'constant' at
'PROPOSAL_ALREADY_ACTIVE'
MoonLock.sol:8:29: Error: Parse error: missing '=' at '('
MoonLock.sol:9:6: Error: Parse error: missing 'constant' at
'PROPOSAL_NOT_ACTIVE'
MoonLock.sol:9:25: Error: Parse error: missing '=' at '('
MoonLock.sol:10:6: Error: Parse error: missing 'constant' at
'STARTING_TIME_PASSED'
MoonLock.sol:10:26: Error: Parse error: missing '=' at '('
MoonLock.sol:11:6: Error: Parse error: missing 'constant' at
'PROPOSAL_NOT_STARTED_YET'
MoonLock.sol:11:30: Error: Parse error: missing '=' at '('
MoonLock.sol:12:6: Error: Parse error: missing 'constant' at
'PROPOSAL_EXPIRED'
MoonLock.sol:12:22: Error: Parse error: missing '=' at '('
MoonLock.sol:13:6: Error: Parse error: missing 'constant' at
'PROPOSAL_NOT_EXPIRED_YET'
MoonLock.sol:13:30: Error: Parse error: missing '=' at '('
MoonLock.sol:14:6: Error: Parse error: missing 'constant' at
'INSUFFICIENT_BALANCE_FOR_VOTING'
MoonLock.sol:14:37: Error: Parse error: missing '=' at '('
MoonLock.sol:15:6: Error: Parse error: missing 'constant' at
'ALREADY_VOTED'
MoonLock.sol:15:19: Error: Parse error: missing '=' at '('
MoonLock.sol:16:6: Error: Parse error: missing 'constant' at
'DURATION_NOT_ALLOWED'
MoonLock.sol:16:26: Error: Parse error: missing '=' at '('
MoonLock.sol:17:6: Error: Parse error: missing 'constant' at
'NOT_OWNER_OF_ASSET'
MoonLock.sol:17:24: Error: Parse error: missing '=' at '('
MoonLock.sol:18:6: Error: Parse error: missing 'constant' at
'OUT_OF_BOUND_REQUEST'
MoonLock.sol:18:26: Error: Parse error: missing '=' at '('
MoonLock.sol:19:6: Error: Parse error: missing 'constant' at
'INSUFFICIENT_FUNDS'
MoonLock.sol:19:24: Error: Parse error: missing '=' at '('
MoonLock.sol:20:6: Error: Parse error: missing 'constant' at
'NOT_A_PREMIUM_MEMBER'
MoonLock.sol:20:26: Error: Parse error: missing '=' at '('
MoonLock.sol:21:6: Error: Parse error: missing 'constant' at
'PROPOSAL_ALREADY_INPROGRESS'
MoonLock.sol:21:33: Error: Parse error: missing '=' at '('
MoonLock.sol:91:37: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:110:39: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:140:41: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:144:42: Error: Parse error: mismatched input '('
```

```
expecting {';', '='}
MoonLock.sol:148:39: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:152:39: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:171:38: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:177:46: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:190:41: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:194:38: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:200:43: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:204:38: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:214:50: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:218:32: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:222:35: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:241:38: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:247:43: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:295:37: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:299:37: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:303:39: Error: Parse error: mismatched input '('
expecting {';', '='}
MoonLock.sol:319:39: Error: Parse error: mismatched input '('
expecting {';', '='}
```

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