QuillAudits

Audit Report
August, 2024

For

OMA3

Table of Content

Executive Summary

Project Name	OMA3
Overview	OMA3 smart contracts encompass an ERC20 token, a token lock and staking contracts with different purposes. AccessControlEnumerable, ERC20 and SafeERC20 contracts from the standard Openzeppelin library were integrated into the OMA3 contracts. The OMALock contract allows for the creation and update of Lock details, staking, un-staking and slashing of locks. The OMABoardStaking is designed for allowing users to stake OMA tokens and un-stake. The OMABountyStaking shares similar features as the OMABoardStaking but instead of the unstake feature, users get their staked tokens when the release address invokes release function. The implementation across all of the contract allows some unique addresses to onboard users into the protocol.
Timeline	29th July 2024 - 13th August 2024
Updated Code Received	15th August 2024
Second Review	19th August 2024 - 20th August 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	This audit aimed to analyze the OMA3 Codebase for quality, security, and correctness. 1. OMA.sol 2. OMALock.sol 3. OMABountyStaking.sol 4. OMABoardStaking.sol
Source Code	https://github.com/oma3dao/token-ft-eth

Executive Summary

Branch	Main
Fixed In	https://github.com/oma3dao/token-ft-eth/pull/2 https://github.com/oma3dao/token-ft-eth/pull/1

Number of Security Issues per Severity

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	1	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	2

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Medium Severity Issues

1. Centralization Risk

Description

The current reward mechanism of the protocol and the user onboarding process are not conducted on-chain. This lack of on-chain implementation may lead to significant issues such as reduced transparency, limited traceability, and potential security vulnerabilities.

Recommendation

Transition both the reward mechanism and the user onboarding process to an on-chain implementation.

Status

Acknowledged

OMA3 Team's Comment

OMA3 is going to use a multi-sig wallet to execute the different admin functions, including the slash functions. So the on-chain "verification" are going to be done by this multi-sig wallet.

Low Severity Issues

2. Some conditions are dependent on an offchain activity

Path

OMABountyStaking.sol | OMALock.sol | OMABoardStaking.sol

Function

slash

Description

Slash function allows the slasher address to slash users' stakes on the OMA lock contract. Both staking contracts have the slashing attribute; however, the conditions for slashing users' stakes are not explicitly implemented in the contracts. This alongside some other conditions like evaluating the voting power mechanism of users, onboarding, etc.

Status

Acknowledged

Informational Issues

3. Possible out of gas error

Path

OMABountyStaking.sol | OMALock.sol | OMABoardStaking.sol

Function

remove, addLocks, allow, disAllow

```
/**
 * @notice Remove a bounty, requires the remove role
 *         The bounty must not be staked.
 * @param bountyIndex_ The bounty index
 */
function remove(uint256 bountyIndex_) external onlyRole(REMOVE_ROLE) {
    ...
    // remove the bounty index from the bounties index per wallet array
    uint256[] storage _bountiesIndex = bountiesIndexPerWallet[
        _bounty.wallet
    ];
    for (uint256 _i = 0; _i < _bountiesIndex.length; _i++) {
        if (_bountiesIndex[_i] == bountyIndex_) {
            _bountiesIndex[_i] = _bountiesIndex[_bountiesIndex.length - 1];
            _bountiesIndex.pop();
            break;
        }
    }

    // remove the bounty
    delete bounties[bountyIndex_];
}
```

Description

There is no limitation to how much a single wallet will have to bounty indexes. This implies that the creator can assign as many bounty indexes to a single wallet and when this array of bounty index grows so large, it is possible to encounter an out of gas error when the remover address invokes the remove function. Also, there are some functions that take in an array of parameters. If any of the assigned addresses, with insufficient native tokens to pay for gas, invoke these functions with large array inputs, there is a possibility of an out of gas error and the transaction will revert.

Recommendation

Add a limit to the bounty index per wallets to avoid a large bountiesIndex to loop through when removing a bounty index. Also, add a limit to input parameters.

Status

Resolved

(OMA3 Team has acknowledged other gas issues)

https://github.com/oma3dao/token-ft-eth/pull/2/commits/80655cfb8a0856c0c077013072cbb8b2e183869a

4. Use exact lock duration value

Path

OMABoardStaking.sol

Variable

lockDuration

```
 */
//@audit
// uint32 public immutable lockDuration = 365 days;
uint32 public immutable lockDuration = 1 minutes; // TODO: for testnet only
```

Description

For testing purposes during development, the exact lockDuration of the OMABoardStaking contract was changed to 1 minute.

Recommendation

Ensure to update the current codebase to the exact lockDuration before deployment.

Status

Resolved

https://github.com/oma3dao/token-ft-eth/pull/1

5. Address appearing more than once in array causes gas wastage

Path

OMABoardStaking.sol

Variable

allow

Description

The tendency of generating a large array of addresses and amounts, with an address reoccur twice, is high. If an address occurs more than once in the array of parameters passed into the allow function, it will set the value more than once, depending on how often it reoccur.

Recommendation

Although, dev cited the need to amend the wallet amount, supposing a user has not staked, it is better to have a function do this or redesign the allow function to avoid address repetition when it is invoked.

Status

Acknowledged

OMA3 Team's Comment

as this is an admin function (reduced users error compared to public function), and it doesn't create side effect, just gas wastage, to have multiple time the same wallet and amount in the parameter, and would require a notable complexity and gas increase to check a wallet doesn't appear multiple time in the parameter, we are going to leave it like this.

Functional Tests Cases

Some of the tests performed are mentioned below:

- Should test input parameters that addresses invoke with functions
- Should confirm that all tokens in a wallet lock is sent to recipient when slashed
- Should check the effect of updating lock cliffDate and lockEndDate to outdated period
- Should create multiple bounty indexes for a wallet address and remove one large index
- Should revert when restake is triggered for users with a slashed stake
- Should let the allow_role address reset previously allowed addresses
- Should test for allowing multiple address with repeated address
- Should release all tokens staked on contracts and from the OMALock

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the OMA3 codebase. We performed our audit according to the procedure described above.

Medium, Low and informational severity issues were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the end, OMA3 Team Resolved few issues and acknowledged other issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in OMA3 smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of OMA3 smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the OMA3 to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2024

For

OMA3

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com