



SPADE SOLIDITY AUDITS

2omb Audit
January 26, 2022

For :
2omb Team

Website:
2omb.finance

Twitter:
twitter.com/2ombfinance



Twitter:
[@SpadeAudits](https://twitter.com/SpadeAudits)

Telegram:
t.me/spadeaudits



Disclaimer

Spade Solidity reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Spade to perform a security review.

Spade Solidity Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

Spade Solidity Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Spade Solidity Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Spade Solidity’s position is that each company and individual are responsible for their own due diligence and continuous security. Spade Solidity’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a Spade Solidity report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to Spade Solidity by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of Spade Solidity has indeed completed a round of auditing with the intention to increase the quality of the company/ product’s IT infrastructure and or source code.

OVERVIEW

Project Summary

| | |
|---------------------------|---|
| Project Name | 2omb |
| Description | DeFi |
| Platform | Fantom Network |
| 2ombtoken | https://ftmscan.com/address/0x7a6e4E3CC2ac9924605DCa4bA31d1831c84b44aE |
| masonry | https://ftmscan.com/address/0x627A83B6f8743c89d58F17F994D3F7f69c32F461#code |
| treasury | https://ftmscan.com/address/0x55e1978dbf46d66c356b2fd7d2d1d060b2e4e40d#code |
| oracle | https://ftmscan.com/address/0xcf80d06D38d63d4B4D0fc52D2F0CDe7EfA0fBf87#code |
| Two_ombGenesisReward Pool | https://ftmscan.com/address/0x683b16b985f37b0f1b227c2c0264d3f8d05eee37#code |

Audit Summary

| | |
|-----------------|---|
| Delivery Date | January 26 th , 2022 |
| Method of Audit | mythril tools for analyzing, Manual Review |
| Timeline | Story Points - 64 |

Vulnerability Summary

| | |
|---------------------|---|
| Total Issues | 0 |
| Total Critical | 0 |
| Total High | 0 |
| Total Medium | 0 |
| Total Low | 0 |
| Total Informational | 0 |

Executive Summary

Our detailed audit methodology was as follows:

Step 1

A manual line-by-line code review to ensure the logic behind each function is sound and safe from common attack vectors.

Step 2

Simulation of hundreds of thousands of Smart Contract Interactions on a test blockchain using a combination of automated test tools and manual testing to determine if any security vulnerabilities exist.


Step 3

Consultation with the project team on the audit report pre-publication to implement recommendations and resolve any outstanding issues.



Grading

The following grading structure was used to assess the level of vulnerability found within all Smart Contracts:



| Threat Level | Definition |
|---------------|---|
| Critical | Severe vulnerabilities which compromise the entire protocol and could result in immediate data manipulation or asset loss. |
| High | Significant vulnerabilities which compromise the functioning of the smart contracts leading to possible data manipulation or asset loss. |
| Medium | Vulnerabilities which if not fixed within in a set timescale could compromise the functioning of the smart contracts leading to possible data manipulation or asset loss. |
| Low | Low level vulnerabilities which may or may not have an impact on the optimal performance of the Smart contract. |
| Informational | Issues related to coding best practice which do not have any impact on the functionality of the Smart Contracts. |



About

2omb Finance (2OMB) is a cryptocurrency and operates on the Fantom platform.

2omb.finance is an incentive-oriented, high yield algorithmic stablecoin protocol that utilizes seigniorage mechanisms to bring \$2OMB to a 1:1 price peg with Fantom.

Unlike previous algorithmic tokens, \$2OMB is not pegged to a stable coin—it is instead pegged to \$FTM. Why is this? 2omb.Finance believes in the potential of Fantom Opera and has chosen to align its mission to both provide value to and derive value from \$FTM's future growth.

In addition to existing and future use cases such as FTMPad, \$2OMB aims to become the main medium of exchange on Fantom Opera: this will be achieved by providing a mirrored, liquid asset to \$FTM.

One of the primary shortcomings of past algorithmic tokens has been a lack of use cases, leaving no good reason for somebody to want to use or hold them. In order to successfully maintain the peg in the long run, the 2omb team will maintain a focus on innovation around enhanced functionality and use cases.



Protocol Overview

The \$2OMB token serves as an algorithmic stablecoin pegged to the price of 1 FTM. The protocol's underlying mechanism dynamically adjusts \$2OMB's supply, pushing its price up or down relative to the price of FTM.

Inspired by tomb.finance, which was originally inspired by the idea behind Basis as well as its predecessors 2omb.finance is a multi-token protocol that consists of the following three tokens: - 2omb Token (\$2OMB) - 2Shares (\$2SHARE) - 2Bonds (\$2BOND)

Tokens

2OMB – 2OMB token

0x7a6e4E3CC2ac9924605DCa4bA31d1831c84b44aE

2OMB token is designed to be used as a medium of exchange. The built-in stability mechanism in the protocol aims to maintain Tomb's peg to 1 Fantom (FTM) token in the long run.

2SHARE - 2OMB SHARES

0xc54A1684fD1bef1f077a336E6be4Bd9a3096a6Ca.

2OMB Shares (2SHARE) are one of the ways to measure the value of the 2OMB Protocol and shareholder trust in its ability to maintain 2OMB close to peg. During epoch expansions, the protocol mints 2OMB and distributes it proportionally to all 2SHARE holders who have staked their tokens in the **Masonry** (boardroom).

2SHARE holders have voting rights (governance) on proposals to improve the protocol and future use cases within the Tomb finance ecosystem.

2SHARE has a **maximum total supply of 70000** tokens distributed as follows:

Team Allocation: 4990 TSHARE vested linearly over 3 months.

Initial mint: 10 2SHARE minted upon contract creation for initial pool.

Remaining 65000 TSHARE are allocated for incentivizing Liquidity Providers in two shares pools for 12 months.

2BOND - 2omb Bonds

0x9fe05f56c9a644eb92ee8f6987fd4edacb6da87d.

2OMB Bonds (2BOND) main job is to help incentivize changes in 2OMB supply during an epoch contraction period. When the TWAP (Time Weighted Average Price) of 2OMB falls below 1 FTM, 2BONDS are issued and can be bought with 2OMB at the current price.

Exchanging 2OMB for 2BOND burns 2OMB tokens, taking them out of circulation (deflation) and helping to get the price back up to 1 FTM. These 2BOND can be redeemed for 2OMB when the price is above peg in the future, plus an extra incentive for the longer they are held above peg.

This amounts to inflation and sell pressure for 2OMB when it is above peg, helping to push it back toward 1 FTM.

All holders are able to redeem their 2BOND for 2OMB tokens as long as the Treasury has a positive 2OMB balance, which typically happens when the protocol is in epoch expansion periods.



Automation Testing

The audit proceess involves:

1. using mythrill tools for analyzing the exploit in the code → None found
2. Manual line by line reviewing of code to check for logical/calculation error → no issues found.

File: 2ombToken.sol

```
auditor@DESKTOP-3RP1MGS:~$ nano 2ombToken.sol
auditor@DESKTOP-3RP1MGS:~$ myth analyze /home/auditor/2ombToken.sol --solc 0.8.4
The analysis was completed successfully. No issues were detected.
auditor@DESKTOP-3RP1MGS:~$
```

No issues found.

File: Masonary.sol

```
auditor@DESKTOP-3RP1MGS:~$  
auditor@DESKTOP-3RP1MGS:~$  
auditor@DESKTOP-3RP1MGS:~$  
auditor@DESKTOP-3RP1MGS:~$ myth analyze /home/auditor/masonary.sol --solc 0.8.4  
The analysis was completed successfully. No issues were detected.  
auditor@DESKTOP-3RP1MGS:~$
```

No issues found.

File : Treasury.sol

```
auditor@DESKTOP-3RP1MGS:~$  
auditor@DESKTOP-3RP1MGS:~$ myth analyze /home/auditor/treasury.sol --solc 0.8.4  
The analysis was completed successfully. No issues were detected.  
auditor@DESKTOP-3RP1MGS:~$
```

No issues found.

File: 2ombRewardPool.sol

```
auditor@DESKTOP-3RP1MGS:~$  
auditor@DESKTOP-3RP1MGS:~$ myth analyze /home/auditor/2ombRewardPool.sol --solc 0.8.4  
The analysis was completed successfully. No issues were detected.  
auditor@DESKTOP-3RP1MGS:~$ █
```

No issues found.

Best Practices

1. Hardcoded addresses should be checked carefully before deployment.
2. Ownership transfer function - It is good practice to implement an acceptOwnership style to prevent ownership sent to invalid addresses by human error. Code flow similar to below.

```
function transferOwnership(address _newOwner) public  
onlyOwner
```

```
{
```

```
    newOwner = _newOwner;
```

```
}
```

```
function acceptOwnership() public
```

```
{
```

```
    require(msg.sender == newOwner, "Not authorized");
```

```
    emit OwnershipTransferred(owner,newOwner);
```

```
    owner = newOwner;
```

```
    newOwner = address(0);
```

```
}
```

Twitter:
[@SpadeAudits](https://twitter.com/SpadeAudits)

Website:
<https://spadetech.io>

Telegram:
t.me/spadeaudits

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in avulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a structassignment operation affecting an in-memory struct rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.