Omni

AVS And Token Smart Contract Review

Version: 2.1

April, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Omni smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Omni smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Omni smart contracts.

Overview

The Omni token is an ERC20 token designed to be deployed on Ethereum L1. It uses open source libraries from OpenZeppelin for all functionality. It has a maximum total supply of 100 million tokens.

OmniAVS hooks into Eigenlayer for operator registration and ETH delegation. It keeps track of the current operator set and how much ETH is delegated to them. Additionally, it syncs operator set updates with the Omni chain.

Security Assessment Summary

Scope

The scope of this time-boxed review was strictly limited to the following files at commit 6c51393.

1. OmniAVS.sol
2. Omni.sol

Retesting was performed on commit b4e82eb.

Note: third party libraries and dependencies, such as OpenZeppelin, were excluded from the scope of this assessment.

Approach

The review was conducted on the files hosted on the Omni repository at commit 6c51393.

The manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team also utilised the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Coverage Limitations

Due to a time-boxed nature of this review, all documented vulnerabilities reflect best effort within the alloted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 5 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.
- Low: 2 issues.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Omni smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
OMI-01	Infinite Loop In `_getSelfDelegations()`	Critical	Resolved
OMI-02	`syncWithOmni()` Can Surpass Ethereum Block Gas Limit	Low	Closed
OMI-03	`syncWithOmni()` Can Surpass XMsg Max Gas Limit	Low	Closed
OMI-04	Gas Optimisations	Informational	Resolved
OMI-05	Miscellaneous General Comments	Informational	Resolved

OMI-01	Infinite Loop In `_getSelfDelegations()`		
Asset	`OmniAVS.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The `_getSelfDelegations()` function can suffer from an infinite loop if the operator has staked in a strategy that is not included in the `_strategyParams` array. This causes `syncWithOmni()` to fail if any operator has staked in an incompatible strategy.

```
551 for (uint256 i = 0; i < strategies.length;) {
        IStrategy strat = strategies[i];
553
        // find the strategy params for the strategy
555     StrategyParam memory params;
        for (uint256 j = 0; j < _strategyParams.length;) {
557         if (address(_strategyParams[j].strategy) == address(strat)) {
                params = _strategyParams[j];
559             break;
            }
561         unchecked {
                j++;
563         }
        }
565
        // if strategy is not found, do not consider it in stake
567     if (address(params.strategy) == address(0)) continue;

569     staked += _weight(shares[i], params.multiplier);
        unchecked {
571         i++;
        }
573 }
```

If the current strategy is not found inside `_strategyParams` in line [**567**], the outer for-loop will continue without incrementing the `i` iterator and the same incompatible strategy will be checked indefinitely until the transaction runs out of gas.

Recommendations

There are two proposed solutions to resolve the issue.

1. Move `i++` into the for-loop header in line [**551**]
2. Increment `i` inside the if-block before the `continue` keyword in line [**567**].

Resolution

The issue has been addressed in commit fc19c26 where `i++` has been added to the loop definition.

OMI-02	`syncWithOmni()` Can Surpass Ethereum Block Gas Limit		
Asset	`OmniAVS.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Due to the use of for-loops when calling the `_getOperators()` function, it is possible for calls to the `syncWithOmni()` function to use more gas than the Ethereum block gas limit (30 million), which results in the call reverting.

Tests demonstrate this is possible with 98 registered operators staking in 24 EigenLayer strategies each, where all strategies have strategy parameters in `OmniAVS`.

Recommendations

There are two proposed solutions to resolve the issue.

1. Reduce the gas usage of `_getTotalDelegations()` and `_getSelfDelegations()` and avoid iterating through arrays where possible, or
2. Limit the amount of strategy parameters and number of operators such that `syncWithOmni()` can no longer surpass the Ethereum block gas limit in gas usage.

Resolution

The project team has acknowledged the issue with the following comment:

> *"For our initial mainnet release, we will maintain a small list of allowed operators and strategies. Operators will not exceed 30, and supported strategies will not exceed 10. Gas optimizations would require non-trivial refactoring, adding complexity for (under these stricter conditions) marginal benefit. Gas optimizations will be considered in a future release when these conditions are less strict."*

OMI-03	`syncWithOmni()` Can Surpass XMsg Max Gas Limit		
Asset	`OmniAVS.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

In `OmniPortalConstants`, the max gas limit for `XMsg`s (`XMSG_MAX_GAS_LIMIT`) is set to 5 million gas units.

In `OmniAVS`, the base gas limit and gas limit per operator is initialised as 75,000 and 50,000 respectively. This means that the max number of operators before surpassing the `XMsg` gas limit is 98 operators.

If there are 99 or more operators registered in `OmniAVS`, all calls to `syncWithOmni()` will revert as the gas limit for the `XMsg` will be too high.

Recommendations

Consider doing one of the following:

1. Raising the XMsg max gas limit.
2. Reducing the gas limit per operator.
3. Setting the max operator count to less than 98 allowing for a buffer if gas prices change.
4. Allowing the gas limit to be configurable by admin.

Resolution

The project team has acknowledged this issue with the following comment:

> *"We do not plan to address this issue in this release due to the following reasons:*
>
> 1. *We do not plan on supporting a list of operators greater than 30 in the near-to-mid term.*
> 2. *We will likely be able to reduce the XMsg max gas limit per operator in the future."*

OMI-04	Gas Optimisations
Asset	`OmniAVS.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The following gas optimisation changes can be made to greatly reduce gas costs and risk of DoS from reaching the block gas limit:

1. The `_isOperator()` function can use the `_operatorPubkeys` mapping to check if the value corresponding to the operator's address is non-zero.
2. Avoid linear searches by using mappings to store the indexes of operators and strategy parameters in the `_operators` and `_strategyParams` arrays. Ensure that the index values are updated whenever changes to the arrays are made.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The project team has addressed the first gas optimization suggestion in commit 31e15e6.

The second suggestion has been acknowledged with the following comment:

> *"We looked into gas optimizing the contract using OpenZeppelin's EnumerableSet library. After analysis, we found that for reasonably small numbers of operators & strategies, the gas optimizations do not help that much. Given that the number of strategies we plan to support will not exceed 10, and likely remain closer to 5, optimizing for this case now is not necessary."*

OMI-05	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **`params.multiplier` validation**

   Related Asset(s): `OmniAVS.sol`

   In the `_setStrategyParams()` function, consider validating all strategy parameter multipliers so that they do not equal to zero.

2. **Zero-address check**

   Related Asset(s): `OmniAVS.sol`

   Consider adding zero address checks for critical setters such as `setOmniPortal()` and `setEthStakeInbox()`

3. **`_strategyParams` strategy limit**

   Related Asset(s): `OmniAVS.sol`

   The `_strategyParams` array is iterated through in various functions inside nested for-loops.

   Consider introducing a max limit for the number of strategies that can be added to the `_strategyParams` array to prevent high gas costs and potential DoS due to running out of gas.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team have acknowledged these findings, addressing them where appropriate as follows:

1. **`params.multiplier` validation:** Addressed in 763f0e3
2. **Zero-address check:** Addressed in b96525d
3. **`_strategyParams` strategy limit:** Acknowledged with comment - *"Given that we have admin control over the strategy parameters, we do not think enforcing a limit is required."*

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 3 tests for test/Omni.t.sol:OmniTest
[PASS] test_initialMint() (gas: 12039)
[PASS] test_transfer() (gas: 44494)
[PASS] test_transferFrom() (gas: 56436)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.21ms (441.92us CPU time)

Ran 25 tests for test/OmniAVS.t.sol:OmniAVSTest
[PASS] testFuzz_setMaxOperatorCount(uint32) (runs: 1001, u: 40578, ~: 40578)
[PASS] testFuzz_setMinOperatorStake(uint96) (runs: 1001, u: 40744, ~: 40744)
[PASS] testFuzz_setOmniChainId(uint64) (runs: 1001, u: 39223, ~: 39223)
[PASS] testFuzz_setXCallGasLimits(uint64,uint64) (runs: 1001, u: 41620, ~: 41620)
[PASS] test_allowlist_addAndRemove() (gas: 69457)
[PASS] test_allowlist_enableAndDisable() (gas: 58070)
[PASS] test_avsDirectory() (gas: 15137)
[PASS] test_canRegister() (gas: 93722900)
[PASS] test_deregisterOperator() (gas: 1619264)
[PASS] test_ejectOperator() (gas: 933567)
[PASS] test_feeForSync() (gas: 953755)
[PASS] test_getOperatorRestakedStrategies() (gas: 907328)
[PASS] test_isInAllowlist() (gas: 50017)
[PASS] test_operators() (gas: 1039091)
[PASS] test_pauseAndUnpause() (gas: 121657)
[PASS] test_registerOperator() (gas: 97228406)
[PASS] test_setEthStakeInbox() (gas: 40854)
[PASS] test_setMetadataURI() (gas: 50077)
[PASS] test_setOmniPortal() (gas: 52065)
[PASS] test_setStrategyParams() (gas: 142471)
[PASS] test_strategyParams_getRestakeableStrategies_equivalence() (gas: 26445)
[FAIL. Reason: assertion failed] test_syncWithOmni_enoughGas_Vuln() (gas: 639285853)
[PASS] test_syncWithOmni_fee() (gas: 110364)
[PASS] test_syncWithOmni_infiniteLoop_Vuln() (gas: 2323743)
[FAIL. Reason: revert: OmniPortal: gasLimit too high] test_syncWithOmni_xmsgGasLimitDoS_Vuln() (gas: 66232376)
Suite result: FAILED. 23 passed; 2 failed; 0 skipped; finished in 1.02s (2.22s CPU time)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

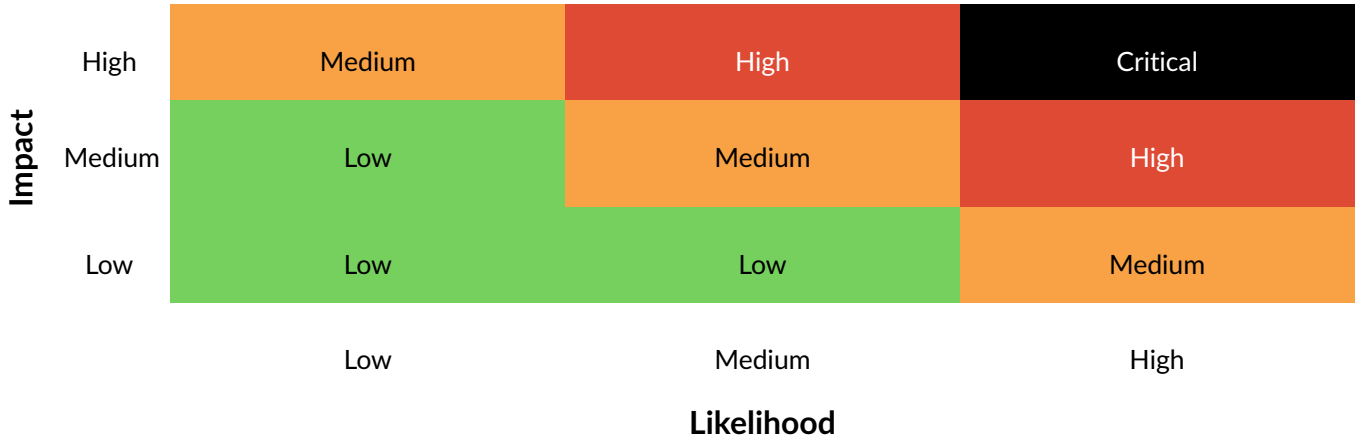

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].