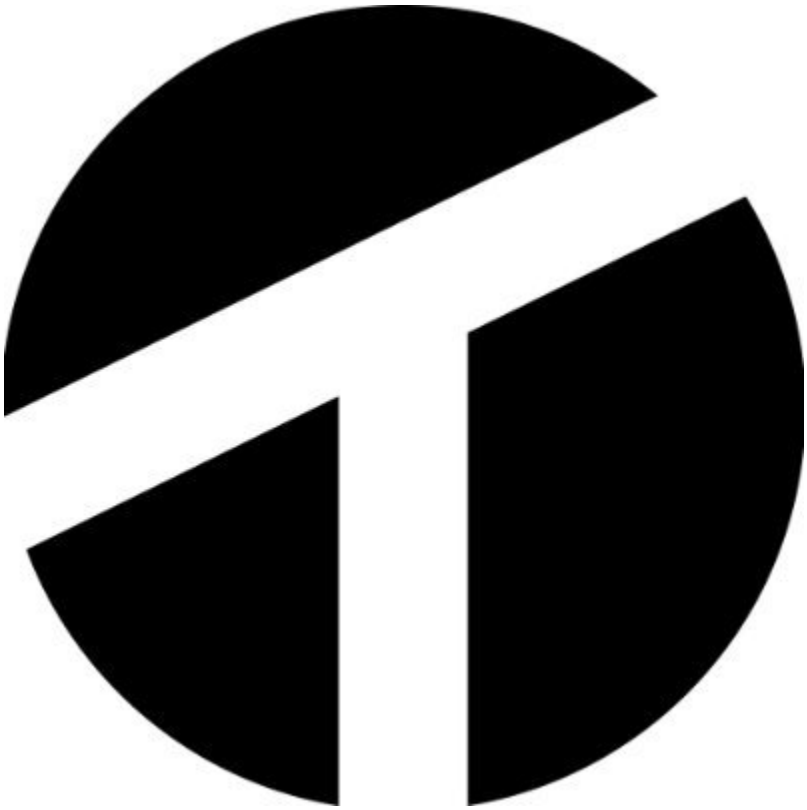
Othentic

Smart Contract Security Assessment

September 23, 2024



ABSTRACT

Dedaub was commissioned to perform a security audit of the Othentic protocol. Three low severity issues and one medium severity issue were identified by the audit team and resolved by the Othentic team.

BACKGROUND

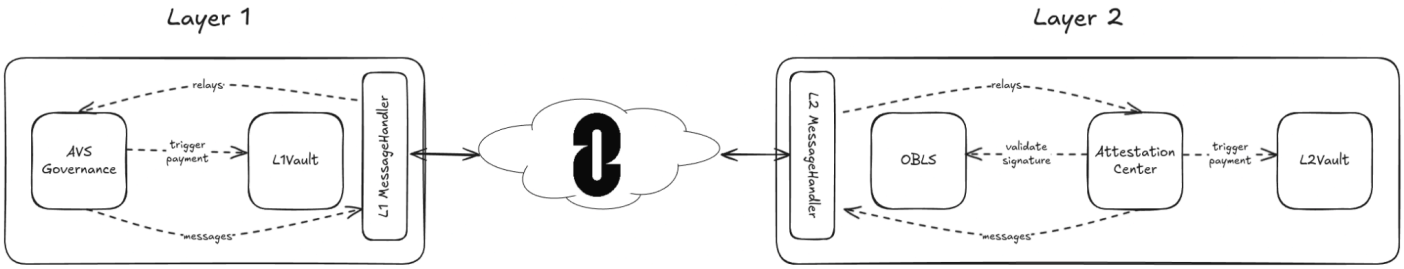
Othentic is building a platform to enable the creation of Actively Validated Services (AVSs) on top of Eigenlayer. Their stack consists of code for both on-chain and off-chain components.

The off-chain code which operates using Othentic's stack consists of a network of operators, which can be Performer, Attestator and Aggregator nodes. These are able to perform and validate tasks, and receive payment in return. These off-chain components were out of scope for this audit.

On the other hand Othentic's on-chain code consists of two subsystems, one operating on Layer 1 (L1) and the other on Layer 2 (L2). Only the following on-chain contracts were in scope:

- AVSGovernance: an L1 contract designed to allow operator registration and AVS management.
- AttestationCenter: an L2 contract which provides an interface to the off-chain components, and handles functionality such as task submission and definition, as well as payments to the operators.
- OBLS: an L2 support contract to conduct BLS signature aggregation and validation for the operators, while also accounting for operator vote weights.
- Vault (L1 & L2): an "escrow" contract where AVSs can preload funds to be paid out to operators (following task submission).

The contracts are arranged and deployed in the following manner, with LayerZero acting as the message relay between the L1 and L2.



SETTING & CAVEATS

The audit report mainly covers the contracts of the following private repository at commit number `23da97cdc43f901870b440cc4516d605462de65c`.

2 auditors worked on the codebase for 10 days (each) on the following contracts:

```
src
└── NetworkManagement
    ├── Common
    │   ├── OBLS.sol
    │   ├── OBLSStorage.sol
    │   ├── Vault.sol
    │   └── VaultStorage.sol
    ├── L1
    │   ├── AvsGovernance.sol
    │   ├── AvsGovernancesLibrary.sol
    │   ├── AvsGovernanceStorage.sol
    │   └── L1Vault.sol
```

```
└── L2
    ├── AttestationCenter.sol
    ├── AttestationCenterStorage.sol
    └── L2Vault.sol
```

Following this, the private repository was forked into a public repository and fixes for outstanding issues were applied, resulting in commit `459d1b0a58eef2b0a8bf42d5ac7e92a185e77ca0`. These fixes were audited again and verified for correctness by the Dedaub team.

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e. full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

PROTOCOL-LEVEL CONSIDERATIONS

ID	Description	STATUS
P1	Aggregators are not incentivised to collect all the attestors' signatures	**INFO**
The role of aggregators in the system is to collect and combine BLS signatures until 66% of attestors approve or 33% of attestors reject a task. Aggregators are paid a flat fee for submitting a task, which is not tied to how many signatures they collect.		

This can result in aggregators not collecting signatures from smaller attestors since it will require more computation to reach the signature threshold, as compared with collecting fewer signatures from larger attestors, thus making it more likely that someone else will have reached the signature threshold and submitted before them.

This can have a centralizing effect whereby smaller or less geographically central attestors are not ever picked up by aggregators, and focusing the rewards to a smaller set of operators. There are a number of different scenarios which can unfold here, with a lot of these problems also existing in the Ethereum consensus layer and Cosmos.

P2	Attestors are not paid proportionally to their stake	**INFO**

Reward calculations inside of `_submitTaskBussinesLogic` do not scale rewards based on the vote share of the attestor.

Comments:
The default behavior can be modified by AVS's by implementing `IFeeCalculator` and setting the `feeCalculator` inside the `AttestationCenter`. This design was decided on by the Othentic team following discussions with their users.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
LOW	Examples: • Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed" or "acknowledged" but no action taken, by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

ID	Description	STATUS
M1	Missing pauser role assignment in AvsGovernance contract	**RESOLVED**

The `AvsGovernance` contract does not assign the pauser role `PauserRolesLibrary.SET_AVS_LOGIC_FLOW` to a contract. This means that the `setAvsGovernanceLogic` function is not pausable.

Resolution:
In addition to resolving the issue above the team has added whenFlowNotPaused(PauserRolesLibrary.SET_AVS_LOGIC_FLOW) modifiers to the `setAvsLogic`, `setBeforePaymentsLogic` and `setFeeCalculator` functions, as these may need to be paused by contracts with the `SET_AVS_LOGIC_FLOW` pauser role.

LOW SEVERITY:

ID	Description	STATUS
L1	Insufficient transfer validation in Vault::withdrawRewards	**RESOLVED**

The `withdrawRewards` function of the `Vault` contract only checks whether the `success` value of its low level call is true or false. However this is not sufficient validation, as some `ERC20` tokens return a boolean value to indicate success or failure, instead of reverting.

`Vault:_withdrawRewards:101-121`

```
function _withdrawRewards(address _operator, uint256 _lastPayedTask,
uint256 _feeToClaim) internal returns (bool _success) {
    ...
        (_success, ) =
address(_token).call(abi.encodeWithSignature("transfer(address,uint256)",
_operator, _feeToClaim));
    }
    else {
        (_success,) = _operator.call{value: _feeToClaim}("");
    }

    // Dedaub: revert checks are not sufficient for all tokens, some use
boolean values to denote failure/success
    if (_success) {
        _sd.balance -= _feeToClaim;
        emit RewardWithdrawn(_operator, _lastPayedTask, _feeToClaim);
    }
    else {
        emit RewardWithdrawalFailed(_operator, _lastPayedTask,
_feeToClaim);
    }
}
```

Resolved:

The protocol now also validates `returndata` in the cases where it is returned.

L2	OBLS::_modifyOperatorVotingPower fails successfully	**RESOLVED**

The if-else statements inside of `_modifyOperatorVotingPower` function of the `OBLS` contract are rendered redundant due to a prior call to `_resetOperatorVotingPower`, which sets the `votingPower` to 0. This results in the if branch always executing (except in the zero case) and setting the `votingPower` to the new value.

`OBLS::_modifyOperatorVotingPower:261-268`

```
function _modifyOperatorVotingPower(uint256 _index, uint256 _votingPower,
OBLSStorageData storage _sd) internal {
    _resetOperatorVotingPower(_index, _sd);
    if (_sd.operators[_index].votingPower < _votingPower) {
        _increaseOperatorVotingPower(_index, _votingPower, _sd);
    } else {
        _decreaseOperatorVotingPower(_index, _votingPower, _sd);
    }
}
```

Resolution:
The function was removed from the codebase and the system was refactored to use `_increaseOperatorVotingPower` and `_decreaseOperatorVotingPower` instead.

L3	Incorrect initialisation in several contracts	**RESOLVED**

The `OBLS` contract inherits from `AccessControlUpgradable`, but does not call `__AccessControl_init` in its initializer.

The `Vault`, `AVSGovernance` and `AttestationCenter` functions inherit from `ReentrancyGuardUpgradable` but do not call `__ReentrancyGuard_init` in their initializer.

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

ID	Description	STATUS
N1	SharesSyncer as complete control over voting weights	**DISMISSED**

The `OBLS` contract defines a `SharesSyncer` role which is granted the permission to increase and decrease operator voting power. This functionality is needed for the protocol to function properly, as the operator's stake may increase or decrease over time and may need to be adjusted.

However, control of this role would allow the controller to also arbitrarily adjust the voting power of any operator, undermining the validation process.

Comments:
The Othentic team have clarified that this is a temporary solution for their initial version. In future versions of the protocol they plan to deprecate this and have share syncing be a "maintenance task" within the AVS. These "maintenance" tasks will be validated by the network in a decentralized way and submitted via the same mechanisms of AVS-defined tasks. This ensures the mechanism stays credibly neutral.

OTHER / ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Inconsistent Permissions in OBLS	**RESOLVED**

The functions used in the `OBLS` contract to update operator voting power are split into 3 types, `modify*`, `increase*` and `decrease*`. The `modify*` endpoints are used to set the absolute value of an operator's vote power, and are guarded by `onlyOblsManager`. The `increase*` and `decrease*` endpoints, which can be combined to act like the `modify*` endpoints are set as `onlyOblsManagerorSharesSyncer`. Hence the `SharesSyncer` is able to reconstruct the behavior of the `modify*` function despite not being able to call it.

Resolution:
The team have removed the `modify*` functions, and updated the permissions to use proper roles (`VOTING_POWER_SYNCER` and `OBLS_MANAGER`).

ID	Description	STATUS
A2	AttestationCenter::isSorted does not allow duplicate values	**DISMISSED**

The `isSorted` function of the `AttestationCenter` contract does not work properly if the `_arr` parameter contains duplicate entries.

`AttestationCenter::isSorted:540-549`

```
function _isSorted(uint[] memory _arr) private pure returns (bool) {
    uint _len = _arr.length;
```

```
    if (_len <= 1) return true;
    unchecked {
        for (uint i = 0; i < _len - 1; i++) {
            // Dedaub: fails on duplicates
            if (_arr[i] >= _arr[i + 1]) return false;
        }
    }
    return true;
}
```

Comments:
This is intended behavior as the supplied user input is trusted and intended to be a set, rather than an array.

A3	AttestationCenter::_verifyArraySubset implicitly expects two sorted arrays without validation	**DISMISSED**

The `verifyArraySubset` function of the `AttestationCenter` requires the two array parameters `_arr1` and `_arr2` to be sorted. If the arrays are not sorted it is possible for the `_verifyArraySubset` to report false negatives. That is, it can report that an array is not a subset of another array when this is in fact the case (for example `_arr1 = [1,2]` and `_arr2 = [2,1]`).

Also note that the arrays cannot contain any duplicates for the function to work properly. For instance, `_arr1 = [1,1,5]` and `_arr2 = [1,2,5]` will also report a false negative.

Comments:
The team has indicated that arrays will be provided in sorted order and without duplicates to the function. In addition the team is only concerned about false positives for this function.

A4	It is possible to submit tasks with taskDefinitionId set to zero	**DISMISSED**

The `_submitTask` function of the `AttestationCenter` contract allows users to submit tasks whose `taskDefinitionId` is set to zero. This triggers special case behavior inside this function, skipping certain validations.

Comments:
The Othentic team have clarified that this is intended behavior for the initial phase to aid users in setting up a simple test system. The team has informed us that they intend to remove this in future versions.

A5	Large constants are not declared using scientific notation	**RESOLVED**

Constants such as the `baseRewardFee` in `AttestationCenter` or the `MILLION_DENOMINATOR` in `Vault` are written in full. It is generally advised that these are written in scientific notation to reduce mistakes caused by typos and increase code clarity.

A6	Misleading Events	**RESOLVED**

The `_depositERC20Rewards` function of the `Vault` contract emits an event `RewardsDeposited(_sd.ownerVault, _amount)`, but the amount does not reflect the fact that fees were previously subtracted from the `_amount`.

Similarly, the `depositNative` function of the `Vault` contract emits an event `RewardsDeposited(_sd.ownerVault, msg.value)`, but the amount does not reflect the fact that fees were previously subtracted from the `msg.value`.

A7	Assignment of unused roles	**DISMISSED**

The `AVSGovernance` and `AttestationCenter` contracts grant the `RolesLibrary.COMMUNITY_MULTISIG` role, but this role is not needed to access any functions.		
A8	Dangerous role transfer mechanism	**DISMISSED**
The `transferAvsGovernanceMultisig` function of the `AVSGovernance` contract directly transfers the `RolesLibrary.AVS_GOVERNANCE_MULTISIG` to a new multisig. However if the sender makes a mistake when providing the new address, the governance functionality will be bricked. We recommend implementing a two-step transfer of ownership process instead, such as Ownable2Step.		
A9	Typos in various contracts	**RESOLVED**
A number of typos were found in the contracts which were in scope: • The `AttestationCenter` contract has a function called `_submitTaskBussinesLogic` instead of `_submitTaskBusinessLogic` • The `AvsGovernanceStorage` contract has a mapping called `isReqestPaymentPaused` instead of `isRequestPaymentPaused` • The `Vault` contract has a function called `getStorge` instead of `getStorage`		
A10	Compiler bugs	**INFO**
The code is compiled with Solidity version `0.8.25` which has no known bugs.		

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Security Suite.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.