Eoracle

Smart Contract Security Assessment

March 8, 2024

ABSTRACT

Dedaub was commissioned to perform a security audit of eoracle's middleware component. The audit was over smart contract code. Eoracle is a middleware service to launch over the Eigenlayer infrastructure. The eoracle service is a forked version of Eigenlayer's middleware component with small modifications. The intent is to register operators not just with Eigenlayer but also with the eoracle "chain manager", a new entity notified of operator registration actions.

SETTING & CAVEATS

This audit report mainly covers the contracts of the public repository https://github.com/Eoracle/eoracle-middleware of the protocol at commit `b2cb7ef19c205d90aaab7b02f1f785ef7e912878`.

2 auditors worked on the codebase for 2 days on the following contracts:

```
src/
├── BLSSignatureChecker.sol
├── EOBLSApkRegistry.sol
├── EOBLSApkRegistryStorage.sol
├── EOIndexRegistry.sol
├── EOIndexRegistryStorage.sol
├── EORegistryCoordinator.sol
├── EORegistryCoordinatorStorage.sol
├── EOServiceManager.sol
├── EOStakeRegistry.sol
├── EOStakeRegistryStorage.sol
├── OperatorStateRetriever.sol
├── ServiceManagerBase.sol
├── interfaces/
│   ├── IBLSSignatureChecker.sol
│   ├── IEOBLSApkRegistry.sol
│   ├── IEOChainManager.sol
│   ├── IEOIndexRegistry.sol
│   ├── IEORegistryCoordinator.sol
```

```
│   ├── IEOStakeRegistry.sol
│   ├── IRegistry.sol
│   ├── IServiceManager.sol
│   └── ISocketUpdater.sol
└── libraries/
    ├── BN254.sol
    └── BitmapUtils.sol
```

The implementation of the newly-introduced chain manager functionality was not available at the time of this report, so it was assumed that it would function properly. Among others, the implementation should anticipate problems such as:

- deregistering an operator that has not been registered with the Eoracle chain manager;
- gracefully handling possible front-running of operations, such as calls to EORegistryCoordinator::setChainManager, e.g., to register an operator while no chain manager is set, or while an older chain manager is still in control.

Importantly, at the time of the initial report, the fixes related to our Eigenlayer Middleware Report - Feb'24 have not been applied to the forked codebase. These fixes are in the current dev branch of the eigenlayer-middleware repo. We highly encouraged (also informally, before delivery of this audit report) the eoracle protocol developers to incorporate them, especially commit 6e010fe, and do not include the issues in this report.
UPDATE: the commits from the eigenlayer-middleware repo have now been updated, as of commit `05d8c24e19a6a0b80272b4cfa240202a390f0fc2`.

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.

DESIGN-LEVEL CONSIDERATIONS

ID	Description	STATUS
P1	Forking eigenlayer-middleware is a drastic step	**INFO / ACKNOWLEDGED**
The codebase is a fork of eigenlayer-middleware, with small changes other than renamings of identifiers and modifications to message strings. This raises the question of whether the protocol could have used Eigenlayer's middleware as library functionality or as a base contract, without forking the entire codebase. UPDATE: the developers acknowledged and justify the need to fork for reasons of extensibility and flexibility, especially given the contract size limitations and lack of inheritance support (e.g., virtual functions) in the original.		

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:

[No medium severity issues]

LOW SEVERITY:

[No low severity issues]

CENTRALIZATION ISSUES:

The codebase does not diverge much from Eigenlayer's middleware codebase, so naturally the same centralization considerations apply. In addition, the chain manager is a new entity, set by the RegistryCoordinator owner, although its functionality is merely to receive callbacks when an operator is registered/deregistered.

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	The typical storage gap convention for upgradable contracts is not followed	**INFO (Resolved, commit aad9eb5)**
The standard upgradability convention is for contracts to reserve a total of 50 storage slots. This is usually achieved by maintaining a fixed-sized array state variable and adjusting its size accordingly when new state variables are introduced. The		

`EORegistryCoordinatorStorage` contract unexpectedly breaks the above-mentioned convention by having 9+41+1 = 51 storage slots.

`EORegistryCoordinatorStorage`

```
abstract contract EORegistryCoordinatorStorage is IEORegistryCoordinator {
    uint8 public quorumCount;
    mapping(uint8 => OperatorSetParam) internal _quorumParams;
    mapping(bytes32 => QuorumBitmapUpdate[]) internal _operatorBitmapHistory;
    mapping(address => OperatorInfo) internal _operatorInfo;
    mapping(bytes32 => bool) public isChurnApproverSaltUsed;
    mapping(uint8 => uint256) public quorumUpdateBlockNumber;
    address[] public registries;
    address public churnApprover;
    address public ejector;

    //@Dedaub: This storage variable is new
    IEOChainManager public chainManager;
    ...
    uint256[41] private __GAP;
}
```

Breaking the convention for a single contract might prove error-prone, so we recommend that the `__GAP` array be made to have a fixed-size length of 40.

A2	New field added, with unclear function	**INFO ACKNOWLEDGED**

Since the chain manager functionality is not yet complete, we cannot be certain regarding the use of new data. However, it is particularly unclear what is the purpose of the new field `chainValidatorSignature` used in operator registration:

`IEOBLSApkRegistry:30`

```
struct PubkeyRegistrationParams {
       BN254.G1Point pubkeyRegistrationSignature;
       BN254.G1Point chainValidatorSignature;  // Dedaub: new field
       BN254.G1Point pubkeyG1;
       BN254.G2Point pubkeyG2;
```

```
}
```

If the field is meant to represent a signature that ensures the validity of the operator sign-up request, then it seems to be subsumed by `pubkeyRegistrationSignature`. If, instead, it is a public key in Group 1 of the pairings structure, it seems subsumed by `pubkeyG1`.

A3	License of file differs from eigenlayer	**INFO (Resolved, commit 05d8c24)**

File BN254.sol has a license identifier that includes both BUSL-1.1 and MIT. It is unclear that this combination of licenses is valid. If this is a carry-over from eigenlayer code, it seems that currently the eigenlayer-middleware has updated the license to be just MIT. Since the code is originally from eigenlayer-middleware (and from earlier sources), it is not useful to restrict its license anyway.

`BN254:1`

```
// SPDX-License-Identifier: BUSL-1.1 AND MIT
```

A4	Typehash differs, is incompatible with function signature	**INFO (Resolved, commit 05d8c24)**

The value of `OPERATOR_CHURN_APPROVAL_TYPEHASH` differs from that in the eigenlayer-middleware code (possibly because of copying an older version). Although the possibility of collision or misunderstanding is extremely small, developers could consider updating, to also reflect the function and structure signature.

`EORegistryCoordinatorStorage:18`

```
bytes32 public constant OPERATOR_CHURN_APPROVAL_TYPEHASH =
   keccak256("OperatorChurnApproval(address registeringOperator,bytes32
registeringOperatorId,OperatorKickParam[]
operatorKickParams)OperatorKickParam(address operator,bytes32[]
operatorIdsToSwap)");
```

In eigenlayer-middleware:

```
bytes32 public constant OPERATOR_CHURN_APPROVAL_TYPEHASH =
       keccak256("OperatorChurnApproval(address registeringOperator,bytes32
registeringOperatorId,OperatorKickParam[] operatorKickParams,bytes32 salt,uint256
expiry)OperatorKickParam(uint8 quorumNumber,address operator)");
```

The latter unique typehash seems to correctly reflect the signatures in the code:

```
function calculateOperatorChurnApprovalDigestHash(
       address registeringOperator,
       bytes32 registeringOperatorId,
       OperatorKickParam[] memory operatorKickParams,
       bytes32 salt,
       uint256 expiry
)...

struct OperatorKickParam {
       uint8 quorumNumber;
       address operator;
}
```

A5	Compiler bugs	**INFO**

The code is compiled with Solidity `0.8.12`. This version has some known bugs, which we do not believe affect the correctness of the contracts.

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.