eOracle

Smart Contract Security Assessment

June 21, 2024

ABSTRACT

Dedaub was commissioned to perform a security audit of target contracts of eOracle. The audit covers the two primary contracts, the `EOFeedManager` and the `EOFeedVerifier`, and the feed adapter contracts.

BACKGROUND

The eOracle target contracts consist of two primary smart contracts, the `EOFeedManager` and the `EOFeedVerifier`.

The `EOFeedManager` is responsible for receiving feed updates from whitelisted publishers, verifying them using `EOFeedVerifier`, and storing the verified data for access by other smart contracts.

The `EOFeedVerifier` handles the verification process of update payloads, ensuring the integrity and authenticity of the price feed updates. The update payload includes a Merkle root signed by eOracle validators and a Merkle path to the leaf containing the data. The verifier stores the current validator set in its storage and ensures that the Merkle root is signed by a subset of this validator set with sufficient voting power.

SETTING & CAVEATS

This audit report mainly covers the contracts of the at-the-time private repository Eoracle/target-contracts of the eOracle protocol at commit `e2b4f479ff510bb1bf2d2eaf93f4805b87591f20`.

Two auditors worked on the codebase for 4 days on the following contracts:

```
src/
├── EOFeedManager.sol
```

```
├── EOFeedVerifier.sol
├── adapters/
│   ├── EOFeedAdapter.sol
│   ├── EOFeedRegistryAdapter.sol
│   ├── EOFeedRegistryAdapterBase.sol
│   ├── EOFeedRegistryAdapterClone.sol
│   ├── factories/
│   │   ├── EOFeedFactoryBase.sol
│   │   ├── EOFeedFactoryBeacon.sol
│   │   └── EOFeedFactoryClone.sol
│   └── interfaces/*
├── interfaces/*
└── libraries/
    └── Denominations.sol
```

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.

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:

ID	Description	STATUS
L1	Implementation contracts' initializers are not disabled	**RESOLVED**

The initializer functions of the `EOFeedAdapter`, `EOFeedRegistryAdapterBase`, `EOFeedManager` and `EOFeedVerifier` implementation contracts should be disabled as suggested in the OpenZeppelin best practices. This can be done by calling the `Initializable::_disableInitializers()` function in the constructors of the aforementioned contracts.

```
constructor() {
    _disableInitializers();
}
```

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. We list issues that could arise if the protocol owner abuses their powers 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	EOFeedVerifier::setNewValidatorSet is owner-controlled	**ACKNOWLEDGED**

The audit initially found that the `EOFeedVerifier::setNewValidatorSet` function, which is owner-controlled, does not verify the validity of the provided public keys and their belonging to the E2 elliptic curve group and its subgroup (G2) as suggested in draft-irtf-cfrg-bls-signature-05. Also, `setNewValidatorSet` did not enforce a minimum validator set size or a minimum set voting power.

The protocol team informed us that all the aforementioned validations are performed by the eOracle middleware contracts and on the eOracle chain. A minimum validator set size was also implemented in `setNewValidatorSet`.

Therefore, we think that the main concern with `setNewValidatorSet` lies in the fact that it is owner-controlled. This is a centralization risk that according to the protocol team's words "is actively worked on".

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	`EOFeedVerifier::setNewValidatorSet` might leave junk behind	**RESOLVED**

The function `EOFeedVerifier::setNewValidatorSet` loops over `newValidatorSet` and copies the validator data into the `_currentValidatorSet` mapping.

`EOFeedVerifier::setNewValidatorSet:127`

```
function setNewValidatorSet(
    Validator[] calldata newValidatorSet
) external override onlyOwner {
    uint256 length = newValidatorSet.length;
    _currentValidatorSetLength = length;
    _currentValidatorSetHash = keccak256(abi.encode(newValidatorSet));
    uint256 totalPower = 0;
    for (uint256 i = 0; i < length; i++) {
        if (newValidatorSet[i]._address == address(0))
            revert InvalidAddress();
        uint256 votingPower = newValidatorSet[i].votingPower;
        if (votingPower == 0) revert VotingPowerIsZero();
        totalPower += votingPower;
        _currentValidatorSet[i] = newValidatorSet[i];
    }
    _totalVotingPower = totalPower;
    emit ValidatorSetUpdated(_currentValidatorSetLength,
        _currentValidatorSetHash, _totalVotingPower);
}
```

If the new set is smaller than the previous one, the data of validators whose index is greater than the current `newValidatorSet.length` will remain as junk in the `_currentValidatorSet` mapping, whereas their deletion would entail a gas refund. Another related issue is that the junk validator data will be accessible via the `currentValidatorSet` function, which does not compare the requested index to the length of the current set.

`EOFeedVerifier::currentValidatorSet:198`

```
function currentValidatorSet(
    uint256 index
) external view returns (Validator memory) {
    return _currentValidatorSet[index];
}
```

A2	Redundant `override` in `setNewValidatorSet`	**RESOLVED**

In the EOFeedVerifier contract, the `setNewValidatorSet` function uses an unnecessary `override` which does not stem from any of the interfaces used.

A3	`EOFeedManager` does not need to inherit from `Initializable`	**RESOLVED**

`EOFeedManager` does not need to inherit from the OZ `Initializable` contract, as the `OwnableUpgradeable` contract already does so.

A4	Missing validation checks	**RESOLVED**

The functions `setSupportedFeeds` and `whitelistPublishers` of the `EOFeedManager` contract do not check that the lengths of the provided arrays are equal, i.e., `feedIds.length == isSupported.length` and `publishers.length == isWhitelisted.length`.

A5	Missing `__gap` arrays from some of the upgradeable contracts	**RESOLVED**

Most of the contracts of the protocol are upgradeable and some of them define the usual: `uint256[50] private __gap` array to allow for future addition of variables avoiding issues with the storage expansion. However, not all the upgradeable contracts define this array which introduces an inconsistency which may lead to issues if it is not intentional. More specifically, the `EOFeedManager` and `EOFeedAdapter` contracts are meant to be upgradeable, but they do not declare this array compared to the `EOFeedVerifier` contract which does.

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.