OpenZeppelin

UMA Optimistic Governor Incremental Audit

OPENZEPPELIN SECURITY | JUNE 26, 2023

Security Audits

June 26, 2023

This security assessment was prepared by **OpenZeppelin**.

Table of Contents

Summary

Type	Governance
Timeline	From 2023-03-07 To 2023-03-13
Languages	Solidity
Total Issues	8 (8 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	4 (4 resolved)
Notes & Additional Information	4 (4 resolved)

Scope

We audited the UMAprotocol/protocol repository at the 3687d9b commit.

In scope were the following contracts:

```
 contracts
└── gnosis-zodiac
    └── implementation
        └── OptimisticGovernor.sol
```

System Overview

control DAO funds and activities as long as their proposed activities conform to a set of natural language rules that the DAO has made publicly available. This is facilitated via the UMA Optimistic Oracle, which becomes the final arbiter in case proposals are contested for not conforming to DAO rules. In the case where proposals are not contested during their "liveness" period, they then become executable by anyone.

Summary of Changes

The main change is the migration to the Optimistic Oracle v3. As a consequence, it is possible to specify an escalation manager which enables replacing Data Verification Mechanism (DVM) dispute resolution with a custom dispute resolution logic and whitelisting asserters and disputers.

Additional changes:

- Proposals are now deleted automatically and immediately upon a dispute. If the Optimistic Oracle is upgraded, anyone can delete all proposals that were submitted before the upgrade.
- Proposal explanations and governance rules are now emitted with the proposal event to facilitate disputes and votes on the Optimistic Oracle v3.
- Docstrings and styling were improved.

Security Model and Trust Assumptions

Some of the broader security concerns that were identified in the previous audit were addressed.

First and foremost, there is now less reliance on the DVM. Since proposals are deleted in case of a dispute, it is not possible to execute a malicious proposal even if the DVM is compromised.

Rules and explanations are now part of the proposal event, so the data availability of these items is no longer a concern.

However, there are some new security concerns, on top of those from the previous audit.

the governance. It is the DAO's responsibility to ensure the safety of any chosen escalation manager - not UMA's.

Transactions could be crafted to look safe or even desirable when called, but then be completely malicious if they are delegate-called. The context in which a call is executed comes down to a simple `uint` flag attached to a transaction. DAOs using the Optimistic Oracle need to be vigilant against potential transaction phishing attacks that use the wrong context for a call.

Finally, the Zodiac framework allows for additional modules, modifiers, and guards; it is essentially a smart contract middleware that can sit between EOA interactions with the Optimistic Governor and interactions with the avatar. These entire code chains must be well-understood, as they could potentially modify the final behavior of the avatar in ways that an inspection of the proposals and the Optimistic Governor cannot anticipate.

Privileged Roles

The Optimistic Governor contract has only one privileged role, which is that of the contract owner. The owner is able to set the bond amount and bond collateral token required to initiate a proposal. The owner can also set the rules that proposals must conform to, set the "liveness" time (the window during which a proposal is contestable), and set the identifier used by the Optimistic Oracle to support Optimistic Governance proposals. Finally, the owner can set the escalation manager and renounce or transfer ownership.

`ProposalDeleted` events can be emitted with non-existent `assertionId`

The `OptimisticGovernor` contract's `assertionDisputedCallback` function does not validate whether its `assertionId` argument is valid (i.e., whether it is associated with a proposal). This allows a `ProposalDeleted` event to be emitted for any `assertionId` value, not just values that correspond to actual proposals.

A user can accomplish this by interacting with the `OptimisticOracleV3` contract. The `OptimisticOracleV3` contract's `assertTruth` function may be called by any user to assert truths that will be accepted as true unless disputed. This will result in the generation of a new `assertionId` that will be stored in the oracle's `assertions` mapping.

If `assertTruth` is called with the `callbackRecipient` set to zero and the `escalationManager` set to the `OptimisticGovernor` address, a subsequent call to `disputeAssertion` will call `_callbackOnAssertionDispute`, which in turn will call the `OptimisticGovernor` contract's `assertionDisputedCallback` function, passing the disputed `assertionId`, which exists in the `assertions` mapping of the oracle, but not in the `proposalHashes` mapping of the `OptimisticGovernor`. Regardless, the `assertionId` value is not checked and execution will enter the `if` condition on line 369 because the caller is the `OptimisticOracle`, and ultimately on line 374 the `ProposalDeleted` event will be emitted.

Consider adding a check in the `assertionDisputedCallback` function that ensures the `assertionId` value maps to a non-zero `proposalHash`, which ensures the assertion is associated with a proposal.

Update: *Resolved in pull request #4486 at commit d880037.*

Lack of event emission

The internal `_sync` function changes the `OptimisticOracleV3` contract used by the `OptimisticGovernor`, but unlike other administrative setter functions, it does not emit an event when a change occurs.

Update: *Resolved in pull request #4487 at commit f3ea7a6.*

Lack of contract address check in `setEscalationManager`

The `setEscalationManager` administrative function in `OptimisticGovernor` does not perform any checks to validate that the `_escalationManager` address is a contract. Specifying an address where no contract code is deployed could result in unexpected behavior since the `OptimisticOracleV3` contract expects that all escalation managers implement `EscalationManagerInterface`.

Consider using the existing `_isContract` function to validate the `_escalationManager` argument.

Update: *Resolved in pull request #4488 at commit ed8f3fc.*

Unused event

In the `OptimisticGovernor` contract, the `SetCollateral` event, which can be used to emit an updated `collateral` value, is unused. The `setCollateralAndBond` function emits the `SetBond` event, which includes both the `collateral` token and the `bond` amount.

Consider renaming the `SetBond` event to `SetCollateralAndBond`, and removing the unused `SetCollateral` event.

Update: *Resolved in pull request #4489 at commit cf6b68f.*

Notes & Additional Information

Inconsistent variable naming

Some functions in the `OptimisticGovernor` contract use leading underscores to name either input parameters or local variables in cases where there is no name collision with the contract's storage variables:

- `deleteProposalOnUpgrade`: Input parameter `_proposalHash`
- `_constructClaim`: Input parameters `_proposalHash` and `_explanation`

For consistency and clarity, consider removing the leading underscore from these parameters, and also updating the corresponding docstrings. Alternatively, consider using the leading underscore consistently throughout the contract.

***Update:** Resolved in pull request #4490 at commit 8a2f6d5.*

Locked ETH

The `executeProposal` function has the `payable` modifier. However, any ETH sent is not used in the function and will be locked in the `OptimisticGovernor` contract.

Consider removing the `payable` attribute to avoid potentially locking ETH in the `OptimisticGovernor` contract.

***Update:** Resolved in pull request #4491 at commit f417256.*

Misleading variable names

The `proposalHashes` and `assertionIds` mapping names misleadingly reflect the contents of one another (i.e., `proposalHashes` stores `assertionId` values, and `assertionIds` stores `proposalHash` values).

To increase clarity, consider swapping the names of the `proposalHashes` and `assertionIds` mappings, so that the variable names correspond to the values being stored.

***Update:** Resolved in pull request #4492 at commit c1e996c.*

Redundant code

Consider making the following changes to eliminate redundant code:

- In `OptimisticGovernor.sol`, `msg.sender` is used on line 263 to transfer tokens from the proposer to the Optimistic Governor contract, but `proposer` has already been swapped, and `proposalHashes[_proposalhash]` inside the `require` statement can be replaced with `assertionId`.

- In `StakerInterface.sol`, the `setDelegate` and `setDelegator` functions are marked `virtual`, but interface functions are implicitly virtual, so this keyword can be removed.

Update: *Resolved in pull request #4493 at commit 835a6ff. The `StakerInterface.sol` file was not changed. The UMA team stated:*

> *The issue in `StakerInterface.sol` was not fixed as it is not used by the Optimistic Governor, while the dependent VotingV2 contract is already deployed in production.*

Four low-severity issues were found, and four notes were included to improve the quality of the codebase. Some changes were proposed regarding smart contract security best practices. In terms of system design, this iteration of the Optimistic Governor was found to have more flexibility and fewer security assumptions than the previous implementation.

Appendix

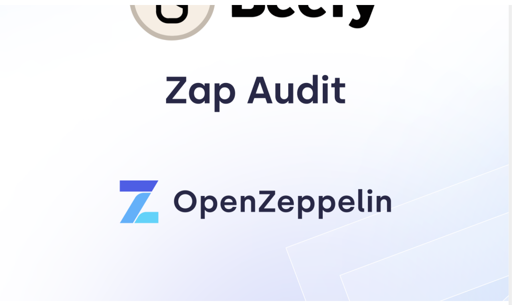
Monitoring Recommendations

While audits help in identifying potential security risks, the UMA team is encouraged to also incorporate automated monitoring of on-chain contract activity into their operations. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment.

To ensure no unexpected administrative actions are occurring, and to validate that correct values were used, consider monitoring all `OptimisticGovernor` administrative events (i.e., events emitted from functions with the `onlyOwner` modifier). In particular, consider monitoring:

- The `OwnershipTransferred` event for expected and unexpected changes of ownership
- The repeated emission of events that are expected to be unique (e.g., the occurrence of multiple `ProposalExecuted` events with identical `proposalHash` and `assertionId` values)
- Changes in the `Finder` contract state, specifically a change in address for the `Store`, `CollateralWhitelist`, `IdentifierWhitelist`, and `OptimisticOracleV3` contracts, which affect the behavior of the Optimistic Governor
- Unusually high or low values emitted in the `SetCollateralAndBond` and `SetLiveness` events, which may interfere with proper operation of the Governor

Related Posts



Beefy Zap Audit

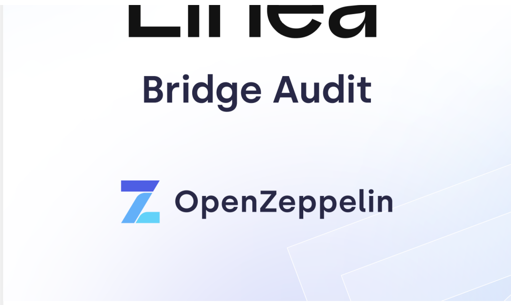
BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits



Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

© Zeppelin Group Limited 2023
Privacy | Terms of Use