UMA Optimistic Governor Audit

OPENZEPPELIN SECURITY | JULY 21, 2022

Security Audits

This security assessment was prepared by **OpenZeppelin**, protecting the open economy.

Table of Contents

Summary

Type
: Governance/Oracles

Timeline
: From 2022-04-25
: To 2022-05-10

Languages
: Solidity

Total Issues
: 21 (18 resolved)

Critical Severity Issues
: 0 (0 resolved)

High Severity Issues
: 0 (0 resolved)

Medium Severity Issues
: 4 (3 resolved)

Low Severity Issues
: 5 (4 resolved)

Notes & Additional Information
: 12 (11 resolved)

Scope

commit `fca8e24275e928f7ddf660b5651eb93b87f70afb`.In scope were the following contracts:

- OptimisticGovernor.sol

2. The `UMAprotocol/protocol` repository PR #3880 at commit `acfb166ef521c1b6ab13a82acf5eee6e16ffd9e9`.In scope were the following contracts:
   - Lockable.sol
   - OptimisticOracle.sol
   - OptimisticOracleInterface.sol
3. After the initial audit UMA requested us to audit the `UMAprotocol/protocol` repository PR #3911 at commit `75bdffecdae5c3f0c7a7546e65c99b0be02be223`. We included PR in scope and audited it.

System Overview

The Optimistic Governor contract is meant to control an “avatar” that complies with the Gnosis’ Zodiac framework. The target use case is that the avatar is a Gnosis Safe that holds DAO funds and is capable of controlling DAO operations. The Optimistic Governor aims to allow anyone to 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.

The unrelated Optimistic Oracle PR we were asked to review adds a mechanism to suspend reentrancy guard protections to allow for callbacks in the body of `nonRentrant` functions (where such callbacks would not otherwise be permissable). Additionally, the Oracle was modified to support “event-based” price requests – that is price requests after some event in the future occurs, rather than at a specific point in time.

Privileged Roles

which a proposal is contestable), and set the identifier used by the Optimistic Oracle to support Optimistic Governance proposals. Finally, the owner can delete specific proposals at any time and can renounce and transfer ownership.

Security Considerations

The UMA team clearly spent time considering security implications of the Optimistic Governor as evidenced by the UMIP that is required for the Optimistic Oracle to support this new use case. However, not all security implications are touched on in the UMIP. Some additional considerations follow:

First and foremost, the theoretical possibility of corrupting the Optimistic Oracle itself, by essentially bribing UMA token holders to enrich themselves at the expense of Optimistically Governed DAO(s) can not be disregarded. The UMA team has a good understanding of how to increase the costs of such corruption, in fact their whitepaper covers the topic at length, but at this time it is unclear if all of the corruption mitigation mechanisms are fully in place and actively enforced. The risks of corruption, as well as any countermeasures in place, need to be well understood by DAOs looking to use the Optimistic Governor in production.

Additionally, because the protocol is so flexible, DAOs that use it need to be fully aware of what the protocol can guarantee and what it cannot. DAOs have an immense responsibility not just for making sure that their rules are as explicit and well-defined as possible, but also for choosing where to host such rules. The protocol allows for rules to be submitted as a URI, in which case the DAO needs to consider if those rules are immune from being tampered with or are at risk of being taken off line. The flexible nature of the “rule” argument also allows rules – or a subset of rules – to be stored on chain. This approach may be worth the associated costs depending on the individual DAO.

If the avatar is managed by some means other than the Optimistic Governor, for instance if it has an “owner”, then Optimistic Governance can only work as long as the other means of avatar management do not interfere. The ability to outright delete proposals and modify parameters for making proposals can undermine the entire Optimistic Governance model depending on how those permissions are managed independently of the Optimistic Governor.

a proposal involve upgradeable contracts, then the code that ends up being executed could well be different than that which existed at the time the proposal was made. Savvy or malicious actors could frontrun execution calls with contract logic upgrade calls. DAOs must be vigilant against this and should consider how to handle upgradeable code explicitly in their rules.

Transactions could be crafted to look safe or even desirable when called, but then be completely malicious if they are delegate called. Which context a call is executed in 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 – essentially 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 inspection of proposals and of the Optimistic Governor on its own cannot anticipate.

Findings

Here we present our findings.

Medium Severity

Change of collateral could result in unintended bond value

The `OptimisticGovernor` contract requires users to provide a preconfigured quantity of an ERC20 token as bond to propose a set of transactions. If the set of transactions is rejected, the proposer will lose their bond.

To change the collateral token address and its amount the contract owner will generally have to call two separate functions, namely `setBond` to set the new amount and `setCollateral` to set the new ERC20 address of the bond token.

If the contract owner is an EOA, then `setBond` and `setCollateral` will be called in two separate transactions which allows a third party to call `proposeTransactions` in between.

Consider renaming `setCollateral` to `setCollateralAndBond` and updating both the bond value and the bond token address in the same function call.

Update: *Fixed as of commit* `5794c2040cc85aced20ef1145aa0329a1c8d8236` *in pull request #3912.*

Lack of event emission after sensitive actions

In the `OptimisticGovernor` contract,
the `setUp`, `setBond`, `setCollateral`, `setRules`, `setLiveness`,
and `setIdentifier` functions do not emit relevant events after executing sensitive actions.

Consider emitting events after sensitive changes take place (including in the constructor and/or initializer), to facilitate tracking and notify off-chain clients following the contracts' activity.

Update: *Fixed as of commit* `f6c3d17ae9e31d2f337d3f887647731959096663` *in pull request #3913 and commit* `9b5b6d3f4b1168157344c1b93a2c2aa695f19580` *in pull request #3914.*

Lack of input validation

The `OptimisticGovernor` contract has a general lack of input validation. For instance, the `setUp` function does not validate that the `_finder` argument is non-zero, which can lead to a non-functional module instance. Nor does `setUp` validate that the `_rules` argument is non-empty, which could lead to a loss of funds if optimistic governance proposals are the sole way to manage an Avatar.

In the same function, `_liveness` is checked to be greater than zero, in line with the Optimistic Oracle's lower bound requirement. However, `setUp` does not check that `_liveness` is less than 5200 weeks, which is the Optimistic Oracle's upper bound requirement.

The `setLiveness` and `setRules` functions have the same sort of lacking input validation as `setUp` does.

of the validation logic.

To avoid errors and unexpected system behavior, consider implementing require statements to validate all user-controlled input. Where zero-values are acceptable, consider leaving inline documentation to that effect to avoid ambiguity.

Update: *Fixed as of commit* `661b984edcbab12b7e0ed9f9e9739169cb732c33` *in pull request #3915, commit* `9b5b6d3f4b1168157344c1b93a2c2aa695f19580` *in pull request #3914, commit* `55e8f77e748619052b885ce191fee18984a44f29` *in pull request #3950 and commit* `f3d2431f5fb594b6dd5d3a92d6bb1b91d2a25cfe` *in pull request #3962.*

Mismatches between UMIP and implementation

In UMIP-152, the documentation of several data structures is significantly different from their implementation in `OptmisticGovenor.sol`.

Differences can be found in the `Transaction` and `Proposal` structs as well as the data format of `ancilliaryData`.

Consider updating the documentation to correspond to the implementation. Additionally, in light of the removal of the `module` address from `ancilliaryData`, consider adding further documentation to explain how uniqueness of a proposal identifier across all users of the Optimistic Oracle is maintained.

Update: *not fixed, UMA's reply: "will be fixed in separate PR updating the UMIP after code is finalized".*

Low Severity

Events lacking information

We identified two events which could benefit from being more complete. Specifically, the `ProposalDeleted` and `TransactionsProposed` events.

augmenting the event so that it also emits details about the proposal status and the `msg.sender` when a proposal is deleted.

The `TransactionsProposed` event currently emits the time at which a proposal is created, but it does not emit the liveness time the proposal is subject to or a timestamp that indicates when the proposal needs to be disputed by. As this is likely to be of interest at the time a proposal is created, consider emitting enough information to determine when a proposal must be disputed by.

***Update:** Fixed as of commit `c3ae271a3e9a10dd69fe33ef44417633e53043ec` in pull request #3916.*

Duplicated code

There are instances of duplicated code within the codebase. Duplicated code can lead to issues later in the development lifecycle and leaves the project more prone to the introduction of errors later if functionality changes are not replicated across all instances of code that should be identical.

Within the `OptimisticGovernor` contract the `setUp` function repeats code found in several of the `set*` "setter" functions.

Rather than duplicating code, consider reusing existing functions as needed or having just one contract or library containing the duplicated code and using it whenever the duplicated functionality is required.

***Update:** Fixed as of commit `9b5b6d3f4b1168157344c1b93a2c2aa695f19580` in pull request #3914.*

Misleading inline documentation

There are instances of misleading or imprecise documentation throughout the codebase.

In particular, in `OptimisticGovernor.sol`:

- The public `sync` function has a comment beneath it that reads, "Sync the oracle contract addresses as well as the final fee." In fact, the function merely makes a call to the

collateral type)". However, there is no comparison made.

- The NatSpec of the `_originalTime` parameter of the `deleteRejectedProposal` function is a copy paste error from the line above describing the `_proposalId` parameter.
- On line 246 and line 279 there is an inline comment that reads, "This will revert if the price has not settled". This is not as nuanced as it could be. The calls the comments refer to will revert if the price has not and can not currently be settled; the calls can actually settle the the request if the price has not yet been settled but is settle-able.

Additionally, in `OptimisticOracle.sol`:

- On line 172 the comment explains the inequality check that follows as: "This ensures that the ancillary data is below the OO limit". In fact, the inequality test that the ancillary data is less than *or equal to* the Optimistic Oracle (OO) ancillary data limit.

Clear inline documentation is fundamental to outline the intentions of the code. Mismatches between them and the implementation can lead to serious misconceptions about how the system is expected to behave. Therefore, consider fixing these errors to avoid potential confusion for developers, users, auditors alike.

***Update:** Fixed as of commit `6a3e00d72832e663f191920a796b5cbe52aea774` in pull request #3917 and commit `d1a6421e4331861708a5f5bb7b20072d042d17ff` in pull request #3963.*

Proposals can be deleted repeatedly

In the `OptimisticGovernor` contract there is no check that a proposal exists before it is deleted with the `deleteProposal` function. Similarly, a rejected proposal can be deleted repeatedly via the `deleteRejectedProposal` function.

Although there is no clear economic incentive to do delete a proposal numerous times – in fact it will waste gas – the repeated emission of identical `ProposalDeleted` events could be confusing for parties monitoring for such events.

request #3918.

The `deleteProposal` function may not work as expected with all avatars

The `deleteProposal` function allows the owner to delete a particular proposal so that it will not be executed. In the current implementation of the `OptimisticGovernor` contract the owner and the avatar are the same address. In general, an avatar does not necessarily have the ability to send arbitrary transactions without having enabled some module specifically for this purpose.

If an avatar were to have only the `OptimisticGovernor` module enabled, then it could not initiate a transaction in any way other than calling the `proposeTransactions` function. However, in this case the `deleteProposal` function would not work as expected.

Consider a scenario where an owner would like to delete an existing proposal via creating a new proposal:

If the owner creates a proposal to call `deleteProposal` via `proposeTransactions`, then the expiration time of the `deleteProposal` proposal will be greater than expire time of the original proposal which the owner wished to delete. Thus the owner would not be guaranteed to be able delete the original proposal because it could be executed before the deletion proposal passed the liveness threshold.

If the owner wanted to dispute the proposal they were trying to delete, then they could do so. But the assumption that only proposals which break the rules may not alway hold. Additionally, if the proposal “technically” followed the rules, but only elucidated how the rules themselves needed to be updated, any such rule update proposal would also run into the same sort of liveness delay dilemma.

Consider better documenting assumptions about the capabilities of the avatar and what may happen if those assumptions do not hold. Additionally, if having an avatar exclusively controlled by an `OptimisticGovernor` module is a reasonable use case, then consider allowing some other form of proposal deletion capabilities that can bypass the standard liveness condition in case of emergencies.

Notes & Additional Information

Commented out code

The `proposeTransactions` and `executeProposal` functions in the `OptimisticGovernor` contract include commented out lines of code.

As the purpose of these lines is unclear and may confuse future developers and external contributors, consider removing them from the codebase. If they are meant to provide alternate implementation options, then consider extracting them to a separate document where they can be accompanied by a more thorough explanation of their purpose.

Update: *Fixed as of commit* `e35c199cc774066c4b65bcec8f82cffcc5aeabd4` *in pull request #3919.*

Coding style deviates from Solidity Style Guide

In the `OptimisticGovernor` contract the `private` `_getOptimisticOracle` and `_isContract` functions are declared *before* the `internal` functions.

This function order deviates from the recommended order of: constructor, receive, fallback, external, public, internal, private.

To increase overall code readability, consider reordering these functions and conforming to the Solidity Style Guide where possible.

Update: *Fixed as of commit* `e35c199cc774066c4b65bcec8f82cffcc5aeabd4` *in pull request #3920.*

Disabling reentrancy protection is prone to error

In the `OptimisticOracle` contract all external functions are protected with a `nonReentrant` modifier. However, in designated places the user is allowed to perform a callback into the `OptimisticOracle` contract. This is achieved via the

These functions must be used in pairs to perform as expected; that process is manual and potentially error prone.

To reduce the likelihood of error, consider including a continuous integration or custom linter check for pairwise matching `_start` and `_end` functions and for containment within a function that uses a `nonReentrant` modifier.

Update: *acknowledged by UMA: "This sort of linting would be helpful, but at the moment, it's unclear how something like this could be implemented without a linter that can interpret solidity that also supports custom plugins."; no immediate code changes are needed.*

`immutable` value could be used

In the `OptimisticGovernor` contract the `finder` variable is only ever set in the `setUp` initialization function. The value is not modifiable after deployment.

In practice, the `finder` implementation may generally be modified solely by the UMA team. Only after such an update to the implementation would users want to update their `finder` values. Then, if users do wish to migrate to the new `finder` address, they will need to deploy a new `OptimisticGovernor` module anyway.

In this case, consider marking the `finder` `immutable` and setting it directly in the `constructor` of the module to better signal intent and to reduce users' operational gas costs. Note that this deployment scheme would require the master `OptimisticGovernor` module to be redeployed after `finder` implementation updates before users could redeploy their instances of the `OptimisticGovernor` module.

Update: *Fixed as of commit* `7aae2aa34eabf7b3d5896e3537a9cfc8b17b4e6c` *in pull request #3921.*

Some `public` functions could be `external`

The `setBond`, `setCollateral`, `setRules`, `setLiveness`, and `setIdentifier` functions are marked `public` despite the fact that they are never used internally and could therefor be declared as `external`.

Update: *Fixed as of commit* `3fdebf2d48263f5ec9a73255853874535141e220` *in pull request #3922.*

Suboptimal `struct` packing

In the `OptimisticGovernor` contract the `Transaction` `struct` declares an `Enum.Operation` member labeled `operation`. This member is implicitly of type `uint8` since there are only two potential values for the underlying `enum`.

As `operation` is declared after a dynamic `bytes` declaration and at the end of the `struct` definition, if stored it would take an entire storage slot for itself. Consider declaring the `operation` member either before or after the `address` member labeled `to` in order to take advantage of more efficient `struct` packing in storage. Optimized data structures may be useful for future iterations of the system or for any systems that may integrate with the `OptimisticGovernor` contract and would like to store transactions.

Update: *Fixed as of commit* `556cd89217c3b96dde2f6dfc394f67a0742e48b3` *in pull request #3923.*

Typographical errors

The codebase contains the following typographical errors:

In `OptimisticGovernor.sol`:

- On line 37, `address need to` should be `address needs to`.
- On line 167, `proposals` should be `proposal's`.
- On line 292, `overriden` should be `overridden`.

Consider correcting these typos to improve the overall readability of the codebase.

Update: *Fixed as of commit* `745a64aeb0c0b61b214931310d08fbe8ac155f0f` *in pull request #3924.*

Undocumented implicit approval requirements

In favor of explicitness and to improve the overall clarity of the codebase, consider documenting all approval requirements in the relevant functions' inline documentation.

***Update:** Fixed as of commit* `c6bdb5e02b57d4f4135feca80d08671725141226` *in pull request #3925.*

Unexplained and unused constants

Throughout the `OptimisticGovernor` contract, to check if a proposal has been approved by the Optimistic Oracle the literal value `int256` `1e18` is used, where `1e18` signifies that a proposal was not rejected by the Optimistic Oracle.

Similarly, in the update to the `OptimisticOracle` contract the function `proposedPrice` uses a magic value `type(int256).min` to indicate that an event-based proposal cannot be resolved, because the event has not yet taken place.

Lastly, in the `OptimisticOracle` contract the `MAX_ADDED_ANCILLARY_DATA` constant is declared on line 129. On the next line the constant should be used, but instead the *value* of the constant is used directly to derive another constant.

To improve the overall readability of the codebase and to facilitate refactoring, consider defining a constant for every literal or magic value used, giving it a clear and self-explanatory name, and then using it in place of literal values. Also consider adding an inline comment explaining how literal values were calculated or why they were chosen.

***Update:** Fixed as of commit* `c7babc3d3082200b55901783f4ceabae82df1cea` *in pull request #3909.*

Unnecessary cast

In the `setUp` function of the `OptimisticGovernor` contract, `_collateral` is unnecessarily cast to an `address` type.

To improve the overall legibility of the codebase, consider removing this unnecessary cast.

Unnecessary imports

The codebase contains the following unnecessary imports:

In `OptimisticGovernor.sol`:

- On line 13 `OptimisticOracle.sol` is unnecessarily imported.

Consider removing unnecessary imports to improve code clarity.

***Update:** Fixed as of commit `5833fce724930ba27a01855dcbaf03bcfe7fa7a2` in pull request #3927.*

Unused “using for” directive

The `OptimisticOracle` contract includes the directive using `AncillaryData` for `bytes`, even though none of the library methods are ever used directly on a bytes value.

Consider removing the directive if it will remain unused.

***Update:** Fixed as of commit `2ee3f4b2affb7049a156c93705d9918ce5c3a670` in pull request #3910.*

Conclusions

0 critical and 0 high severity issues were found. Some changes were proposed to follow best practices and reduce the potential attack surface.

Appendix

Severity Levels

Critical Severity

The issue puts a large number of users’ sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client’s reputation or serious financial implications for client and users.

to catastrophic impact for client's reputation or serious financial implications for client and users.

Medium Severity

The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.

Low Severity

The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated its low impact in view of the client's business circumstances.

Notes & Additional Information

The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated its low impact in view of the client's business circumstances. It may also include non-security-relevant content for purely informational purposes.

Related Posts

Beefy Zap Audit

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

OpenBrush Contracts Library Security Review

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

Linea Bridge Audit

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

OpenZeppelin