OpenZeppelin

Uma Audit Phase 6

OPENZEPPELIN SECURITY | JANUARY 7, 2022

Security Audits

Introduction

UMA is a platform that allows users to enter trust-minimized financial contracts on the Ethereum blockchain. We previously audited the decentralized oracle, a particular financial contract template, some ad hoc pull requests, the Perpetual Multiparty template, various incremental pull requests over a longer engagement and the insured bridge.

In this audit we reviewed a new governance proposal contract, a mechanism to extend the UMA ecosystem across multiple chains, a mechanism to distribute rewards to ERC721 token holders in accordance with an off-chain specification, and an update to the insured bridge to support WETH on the Optimism chain.

The audited commit is `0c4cea3c3d5e48da6f8984b8ba3afdfea4ce47cc` and the scope includes the following contracts:

- `oracle/implementation/Proposer.sol`
- `cross-chain-oracle/*` (excluding test and Polygon contracts)
- `financial-templates/optimistic-rewarder/*` (excluding test contracts)

We also reviewed the changes to solidity files in Pull Request 3611.

All external code and contract dependencies were assumed to work as documented.

System overview

that will be sacrificed if the proposal fails. There is no specific incentive for making proposals. The intention is to ensure that only the actions which are very likely to be accepted will be proposed.

The new cross-chain mechanism allows governance proposal to be passed from the Ethereum mainnet to the Optimism and Arbitrum chains. In this way, the UMA governance mechanism on layer 1 can be used to govern UMA contracts on the supported layer 2 chains. The mechanism also allows price requests and resolutions to be forwarded between the layers, so Optimistic Oracles on the layer 2 chains can be secured by the mainnet Data Verification Mechanism in the same way that the layer 1 Optimistic Oracle is secured by the DVM.

It is worth noting that these messages are sent using the native bridge mechanics, which means they are limited by the characteristics of the relevant layer 2 chains. In particular, the messages from layer 2 to layer 1 could take a week or longer to transit the bridge. Moreover, the UMA governance mechanism supports proposals that include multiple ordered transactions, but this merely restricts the order in which they can be added to the bridge. It's possible for some of those transactions to be executed in a different order, or not at all, on layer 2.

The Optimistic Rewarder contract simply mints ERC721 tokens for anyone who requests them. It also allows anyone to associate arbitrary data with any token, and to deposit various ERC20 tokens to be distributed as rewards. The interpretation of the arbitrary data and the expected distribution of rewards amongst the token holders are determined using an unspecified off-chain procedure. Anyone can make a claim that a specific ERC721 token is owed a set of rewards if they're willing to deposit a bond. The standard Optimistic Oracle mechanism is used to allow someone else to dispute the claim, which will be resolved by the DVM. Claims that are not disputed in time are assumed to be valid, and the contract distributes the rewards accordingly. The only restriction (to simplify accounting) is that the oracle bond token cannot be used as a reward token.

Lastly, PR3611 modifies the insured bridge mechanism to avoid sending WETH across the Optimism token bridge, which is not supported. Instead, any L2 WETH deposited in the Optimism deposit box is unwrapped to L2 ETH before transiting the bridge. On layer 1, the ETH is converted to WETH before being forwarded to the WETH bridge pool.

When disputing a reward request, the `OptimisticRewardBase` contract first triggers a proposal on the `SkinnyOptimisticOracle` and then disputes that proposal. However, the proposal sets the expiration time as an offset from the current (dispute) time, while the dispute specifies the expiration as an offset from the time of the original reward request. In most cases, this discrepancy will prevent the oracle from identifying the proposal to be disputed, which means valid disputes will not be processed and invalid reward requests will be accepted.

Consider updating the dispute invocation to correctly specify the proposal to be disputed.

Update: *Fixed as of commit `9e15557` in PR3690.*

[C02] Repeatedly resolve proposals

The `resolveProposal` function of the `Proposer` contract simply validates that the oracle has resolved, but does not check if the bond has been distributed. This means the same proposal can be resolved multiple times, resulting in duplicate bond payments. Consider flagging or deleting existing proposals when they are resolved.

Update: *Fixed as of commit `b152718` in PR3689.*

High severity

None.

Medium severity

[M01] Incorrect event parameters

The `OptimisticRewarderBase` contract defines a `Requested` event which is emitted from the `requestRedemption` function when a redemption is requested. This event is defined to emit the expiry time of the redemption as its last parameter. However, when the event is emitted, its last parameter is incorrectly set to the current time.

Given that this event can be used to trigger off-chain computations, consider updating the emitted value appropriately.

Update: *Fixed as of commit `f04eef9` in PR3694.*

Low severity

[L01] Lack of event emission after disputing a redemption

The `OptimisticRewarderBase` contract defines a `Disputed` event that is intended to be triggered if a redemption is disputed. However, this event is not emitted within or outside of the `OptimisticRewarderBase` contract.

Consider emitting the event after sensitive changes take place in the `dispute` function, to facilitate tracking and notify off-chain clients following the contracts' activity.

Update: *Fixed as of commit `c275e92` in PR3695.*

[L02] Inconsistent reentrancy guard

The `Optimism_ParentMessenger` and `Arbitrum_ParentMessenger` contracts inconsistently apply the `nonReentrant` modifier. Consider including it on all public functions.

Update: *Fixed as of commit `6275c39` in PR3677.*

[L03] Misleading comments

Here are some misleading comments we identified during our review:

- `ChildMessengerConsumerInterface.sol`:
  - Line 5 says "parent messenger" instead of "child messenger"
- `GovernorSpoke.sol`:
  - Lines 49-51 links to a `Gnosis` file even though the comment says the snippet was copied from `Governor.sol`. Additionally, the snippet is not identical to the one in `Governor.sol`

When requesting a price on the `OracleSpoke` contract, the provided ancillary data is stamped with the child chain identifier. However, the `hasPrice` and `getPrice` functions don't stamp the ancillary data when identifying the price request. This forces calling contracts to apply the stamp themselves, which causes an inconsistency between the price request and price retrieval mechanisms. Consider applying the stamp in the `hasPrice` and `getPrice` functions.

Update: *Fixed as of commit `fdb845d` in PR3668.*

[L05] Missing NatSpec parameter

Many functions in the `OptimisticRewarderBase` contract are missing the `@return` parameter in their Natural Specification comments. Consider including it for completeness.

Update: *Fixed as of commit `8920f38` in PR3679.*

[L06] Residual allowance

In order to invoke the Optimistic Oracle, the `OptimisticRewarderBase` contract grants it a token allowance, so it can pull the bond payments. If the proposal fails, the reward redemption is cancelled but the allowance is not reset. Therefore, the Optimistic Oracle will retain an unnecessary residual allowance until the next time a dispute is triggered. Consider revoking the allowance if the proposal fails.

Update: *Fixed as of commit `c2d444b` in PR3698.*

[L07] Invalid refund address

The refund L2 address of the `Arbitrum_ParentMessenger` is initialized to the contract owner, which should be the L1 Governor. Similarly, the `setRefundL2Address` has a comment stating it should be set to the governor. However, when passing messages over the bridge, this value is set as the L2 user, which is the address on Arbitrum that receives excess funds after the ticket is resolved. Since the L1 governor address will not be accessible on Arbitrum, any funds sent to this address will be lost.

Consider setting it to a valid L2 address.

The `GovernorSpoke` and `OracleSpoke` contract each initialize the child messenger in the constructor, with no mechanism to update it. This means that when the child messenger is changed, both spoke contracts become obsolete.

Since the spoke contract are likely more stable than the messengers, consider including a mechanism to update the messenger on the spokes.

Update: *Fixed as of commit* `7c9e061` *in PR3688.*

Notes & Additional Information

[N01] Change bond token

The `Proposer` contract includes a mechanism for the owner to change the size of the proposal bond. Consider whether they should also be able to change the bond token. Note that this would require a mechanism to identify the correct bond currency when existing proposals are resolved.

Update: *Not an issue. UMA's statement for this issue:*

> N01 recommends enabling the proposer contract to change the bond token to something other than UMA. We have no intention of supporting any token other than $UMA for this function and so have chosen to not make any changes for this issue. Moreover, a single token per contract keeps this logic as simple as possible. Lastly, If a change was needed (in the case of a token migration, for instance), we could just deploy a new proposer contract with the other token and initiate a proposal to migrate the system to use that one.

[N02] Incomplete interface

The `ChildMessengerInterface` does not specify a `processMessageFromCrossChainParent` function, even though it is assumed to exist by parent messengers. Consider including it for completeness.

Update: *Not fixed. UMA's statement for this issue:*

> internal method is called called _processMessageFromRoot.

[N03] Incorrect interface

The `GovernorSpoke` contract incorrectly uses the `ChildMessengerConsumerInterface` type to describe its `messenger` variable. Consider using the `ChildMessengerInterface` instead.

Update: *Fixed as of commit `f31a527` in PR3680.*

[N04] Pull tokens to Store

In a previous audit we questioned the purpose of the `Store` contract's `payOracleFeesErc20` function (in issue L19). The UMA team opted to keep the function to standardized the interface for potential future modifications. Since the purpose of the function is not fully specified, it is unclear whether it should be triggered when the `Proposer` contract confiscates a bond. It likely should be used when the `OracleHub` pays for a price request. Consider whether the function should be used in either scenario.

Update: *Acknowledged. UMA's statement for this issue:*

> N04 recommends using the Store's payOracleFeeErc20 method for paying fees in both the Proposer and OracleHub contracts to be consistent with the Store usage. We've opted to not use this function as it would mean needing to import an additional interface (for the store) and require casting of the bond amount to a FixedPoint (which would also require an additional import. To keep the code simple and clean we've opted to not do this. The OZ feedback on payOracleFeeErc20 in audit phase 1 in April 2020 was valid that this method is not really useful, making this kind of integration harder to reason about.

[N05] TODOs in code

There are "TODO" comments in the code base that should be tracked in the project's issues backlog. For example:

- Line 37 of `Arbitrum_ParentMessenger` contract

During development, having well described "TODO" comments will make the process of tracking and solving them easier. Without that information, these comments might tend to rot and important information for the security of the system might be forgotten by the time it is released to production.

These TODO comments should have a brief description of the task pending to do, and a link to the corresponding issue in the project repository.

Consider updating the TODO comments to add this information. For completeness and traceability, a signature and a timestamp can be added. For example:

`// TODO: point this at an interface instead.`

`// https://github.com/UMAprotocol/protocol/issues/XXXX`

`// --mrice32 - 20211209`

Update: *Fixed as of commit `5d57b5b` in PR3684.*

[N06] Typographical errors

The codebase contains the following typographical errors:

- In the `Admin_ChildMessenger` contract, `impleenting` should be `implementing`
- In the `OptimisticRewarderBase` contract, `timestap` should be `timestamp`.
- In the `OptimisticRewarderBase` contract, `liveness liveness` should be `liveness`.
- In the `GovernorSpoke` contract, `only called` should be `only be called`.
- In the `Optimism_ChildMessenger` contract:
  - `onlyCrossDomainAccount` should be `onlyFromCrossDomainAccount`.
  - `addresses on the L1` should be `addresses on the L2`

Update: *Fixed as of commit `9b92b0b` in PR3681.*

[N07] Unused imports

contract
- The `OptimisticRewarderCreator` contract imports unused `ERC721` and `Testable` contracts
- the `OptimisticStaker` contract imports usused `OptimisticRewarder` and `OptimisticRewarderToken` contracts

Update: *Fixed as of commit `40b7221` in PR3682.*

[N08] L2 transaction ordering

The `Governor` ensures transactions within a proposal are executed in order. However, when those transactions involve cross-chain transactions, this merely guarantees that they arrive at the L1 bridge contract in the correct order. In the Arbitrum case, they may be reordered before they are finalized on L2. Therefore, governance proposals should be constructed to permit the possibility of reordered L2 transactions.

Update: *Fixed as of commit `0fb2e7b` in PR3703. The `GovernorHub` can now relay an array of L2 transactions.*

Conclusion

Two critical issues were found in the codebase. One medium severity issue and several minor vulnerabilities have been found, and recommendations for fixes have been suggested.

Related Posts

Beefy

Zap Audit

BRUSHFAM

OpenBrush Contracts Library Security Review

Linea

Bridge Audit

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