UMA DVM 2.0 Audit

OPENZEPPELIN SECURITY | DECEMBER 2, 2022

Security Audits

August 16th, 2022 Update: October 14th, 2022

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

Table of Contents

Summary

Type
: DeFi – Oracles and Governance

Timeline
: From 2022-07-18
: To 2022-07-29

Languages
: Solidity

Total Issues
: 45 (33 resolved, 6 partially resolved)

Critical Severity Issues
: 1 (1 resolved)

High Severity Issues
: 1 (1 resolved)

Medium Severity Issues
: 5 (1 resolved, 2 partially resolved)

Low Severity Issues
: 10 (8 resolved, 2 partially resolved)

Notes & Additional Information
: 26 (20 resolved, 2 partially resolved)

Client-Reported Issues
: 2 (2 resolved)

Scope

We audited the UMAprotocol/protocol repository at the `7938617bf79854811959eb605237edf6bdccbc90` commit.

In scope were the following contracts:

```
├── DesignatedVotingV2Factory.sol
├── GovernorV2.sol
├── ProposerV2.sol
├── ResultComputationV2.sol
├── SlashingLibrary.sol
├── SpamGuardIdentifierLib.sol
├── Staker.sol
├── VoteTimingV2.sol
└── VotingV2.sol
```

System overview

The UMA Data Verification Mechanism (DVM) is the ultimate source of truth in the oracle and governance system of the UMA Protocol. UMA's purpose is to serve as a trustworthy oracle for price requests of arbitrary assets. While the majority of price requests are handled by the Optimistic Oracle component of the UMA ecosystem, any dispute regarding the suggested price leads to the price request being escalated to the DVM.

The DVM uses round-based voting in a commit-reveal scheme to resolve requested prices. A price will successfully resolve if the number of votes exceeds a predefined quorum and more than 50% of votes agree on the same price. Otherwise, a request is rolled to the next round.

The DVM additionally uses its voting mechanism to make governance decisions, which take the form of a sequence of calls to given addresses with given calldata to be executed in order. Any change of the parameterization of the DVM (e.g. the quorum or reward emission) must go through the governance system by proposing the change to the `ProposerV2` contract and in case of acceptance executing it on the `GovernorV2` contract.

Updates from Phase 1 Audit

The scope of this audit covered DVM 2.0, while the prior Open Zeppelin's audit covered DVM 1.0. The following changed between these two major versions:

- DVM 2.0 version migrated from a snapshot-based system, which enables UMA token holders to vote to a new system in which UMA tokens need to be staked within

rewards at a fixed rate per time unit and allocates them pro rata to stakers.

- DVM 2.0 version introduced a slashing mechanism on the staked balance, which transfers stake from minority voters or absentees to majority voters in the case of resolved requests to incentivize participation in the voting process and the correctness of the result.
- DVM 2.0 version introduced an additional feature to remove proposals that are perceived as spam.

Privileged roles

The `VotingV2` system is owned by the `GovernorV2` contract and every change of governance parameterization is protected with the `onlyOwner` modifier. This ensures that every parameter change needs to enter the system as a governance proposal via the `ProposerV2` contract and receive the majority of votes before taking effect. Privileged roles held by EOAs or multi-signature addresses are not present in the core system which excludes the `DesignatedVotingV2` contract.

Client-Reported Issues

On August 25th, 2022 as well as September 9th, 2022 the UMA team approached us about two additional issues related to VotingV2 contract that were not identified as part of the audit. We've included our assessment of these two issues, and the corresponding fixes, below to confirm that they have been resolved.

Medium: Parasitic staking term issue

This issue allows for a "parasitic" staker to accumulate rewards without having to participate on a vote and without ever being slashed. The root cause of this is that the slashing mechanism only takes into account the `activeStake` of a staker while deciding to slash a user, while the rewards mechanism takes both `activeStake` and `pendingStake` into account.

Update: *Fixed as of commit* `fd8cecc6719a7a69d44287b2096866686313593c` *of pull request #4168.*

Low: Unresolvable price request issue

request was not staked before the request and they had staked during the reveal phase period over.

Then the price request can only be resolved if another voter who staked before the request participates once the request rolls over again.

Update: *Fixed as of commit* `2d4fad7cb3525a7faeb88f07953907d3c4515796` *of pull request #4139*.

Findings

Here we present our findings.

Critical Severity

Slashing mechanism grants exponentially more rewards than expected

In the `VotingV2` contract the function `updateTrackers` calls the internal function `_updateAccountSlashingTrackers` with parameter `indexTo` set to the `priceRequestIds.length` to process the slashing rewards and penalties associated with all resolved requests.

Within the `_updateAccountSlashingTrackers` function, a for-loop processes all requests starting with the last request that has not yet been considered for the respective voter up to `priceRequestIds = indexTo - 1`. During this process it is ensured that any `slash` changes throughout one voting round are first accumulated and only applied at the end of the round. This ensures that slash rewards and penalties within one round accumulate linearly instead of exponentially. More specifically, the value `activeStake` which is directly proportional to any penalties and rewards received is meant to be kept constant throughout each round, but should be altered by the amount `slash` between rounds. Finally, the function `_updateAccountSlashingTrackers` applies all remaining `slash` changes to the voter's `activeStake` regardless of round boundaries.

Additionally, the contract contains a function `updateTrackersRange` that is callable without authorization, with freely chosen parameters `voterAddress` and `indexTo` which are passed

Issue: Assume that a voter has committed and revealed multiple votes within one round, and that these votes are resolved and correct. Further assume the voter has not yet called `commitVote` in any of the following rounds (which will automatically call `_updateTrackers` internally). In this scenario, the voter can obtain exponentially more `slash` rewards than intended by applying the steps outlined in the **Strategy** section below. Moreover, this will lead to the total `slash` reward payout exceeding the captured `slash` penalties, in contrast to the expectation of both quantities always being equal. In effect, the staking balances stored in `VotingV2` will no longer be backed by actual UMA tokens contained in the same contract. This will render many stakers unable to withdraw their stake and equip exploiters with disproportionally large voting power thereby breaking the core functionality for the UMA ecosystem.

Strategy: To receive exponentially more rewards, a voter must avoid calling `updateTrackers` which linearly accumulates their rewards (by computing them against the same, round-wise fix value of `activeStake`) and instead perform multiple calls to `updateTrackersRange` with increasing values of `indexTo`, such that the function `_updateAccountSlashingTrackers` processes each request – in which the voter committed & revealed a correct vote in said round – individually. Because each call changes the voter's `activeStake`, this strategy leads to an exponential accumulation of slashing rewards instead of the intended roundwise-linear accumulation of rewards.

Consider modifying the respective `update` functions to ensure that `slash` values can only be applied round-wise. Additionally, consider implementing a monitoring solution that continuously checks whether the sum of all staked balances is equal to the amount of UMA contained within the contract and whether this value lies within an expected range. Finally, consider implementing an emergency mechanism to temporarily suspend the reward system without affecting the contract's functionality as a governance and voting system.

Update: *Fixed as of commit* `18aef110f2bc882a9bfe115bc4ca86f3681f2d4b` *in pull request #4067.*

High Severity

optimization that marks unresolved requests in a prior round (rolled votes) as deleted via an entry in the `deletedRequests` map. The intention is to reduce gas consumption as the function will be called for every staker in the system. The logic on line 860 enables future callers to skip an **isolated** rolled vote by setting `deletedRequests[requestIndex] = requestIndex`. It then appends the request identifier to the `priceRequestIds` array after updating the `lastVotingRound` and `priceRequestIndex` fields on the respective request struct. The intention behind adding a specific `requestIndex` to `deletedRequests` is to prevent it from ever being processed again and instead allow skipping ahead to a non-deleted element.

However, the request processing logic within `_updateAccountSlashingTrackers` is not able to correctly handle multiple rolled votes that appear sequentially in the request array. Consider the following scenario:

The `deletedRequests` map is initially empty and the previous voting round contained three votes (limiting to 3 just for simplicity). In this example we will number these request indexes as 1, 2, 3. Assume the the first two votes (index 1 and 2) have not been resolved and are considered rolled, while index 3 is considered resolved. The first voter to call the `_updateAccountSlashingTrackers` function will go through the array of requests to process and cause a modification of the `deletedRequests` mapping to

```
deletedRequests[1] = 1
deletedRequests[2] = 2
```

and re-append each request's identifier to the `priceRequestIds` array. This creates index 4 and 5 in our example.

The next voter that calls this same function will not process request 1 due to the logic in line 845. Instead, the `requestIndex` will be updated to `2`. However, there is no additional check ensuring that index 2 has not been deleted as well; the code assumes this is not the case and index 2 will be processed again instead of being skipped as intended.

statement on line 867. Consequently, the entire loop is terminated without increasing the voter's `lastRequestIndexConsidered` and none of the subsequent resolved requests within the same round are processed. The comment on lines 865-866 indicates that if the `break` statement on line 867 was reached, "all subsequent requests within the array must be in the same state", i.e. being actively voted on. But in this example, index 3 is a subsequent request that is resolved, and should have slashing computations applied to it, but instead the for-loop terminates. Additionally, the value of `voterStake.lastRequestIndexConsidered` is not updated on line 920.

Now, assuming both rolled requests are resolved in the following round, our previous voter calls (indirectly) this function again. Because `lastRequestIndexConsidered` was not updated, once again the processing starts at index 1, which is again skipped on line 845 when it is looked up in the `deletedRequests` map, and `requestIndex` is advanced to `2`. Execution continues to line 854, which will evaluate to true because the corresponding request has been resolved, so slashing will be applied for index 2. On the next cycles of the for-loop, indices 3, 4, and 5 will be processed. Recall however that index 4 and 5 are re-queued elements that reference the same identifiers as index 1 and 2. Therefore, the slashing penalties for index 2 will be processed twice in this example. In this example all the requests were resolved, but if that had not been the case, additional duplications would have occurred each time a vote was rolled to the next round.

The underlying bug is the incorrect assumption on line 845 that assumes `deletedRequests` won't contain consecutive unresolved votes. There are several ways the fix can be approached depending on the tradeoff between simplicity vs efficiency. Consider some possible options:

- A straightforward fix that sets `requestIndex = deletedRequests[requestIndex]` on line 845 (remove the `+ 1`) and then inserts a `continue` statement. This would return execution back to line 843 where `requestIndex` would be incremented as before, but ensures that the `deletedRequests[requestIndex] != 0` test on line 845 is applied to this new index.

Update: *Fixed as of commit* `8356375e8893d3b0375345b0464ef0fe10d15c26` *in pull request #4064 by adopting the first recommendation.*

Medium Severity

Incorrect refund of spamDeletionProposalBond

In the `VotingV2.sol` contract the `spamDeletionProposalBond` determines the amount of `votingToken` the caller transfers to the contract upon calling `signalRequestsAsSpamForDeletion`. An amount of `votingToken` determined by the same variable is either sent back to the initial requester or to the `OracleInterfaces.Store` contract upon calling `executeSpamDeletion`. Independently, the contract owner is able to call `setSpamDeletionProposalBond` anytime.

If the owner mistakenly calls `setSpamDeletionProposalBond` between a spam deletion request and its execution, the value of the refunded bond will differ from the originally submitted bond, thereby leading to a loss for either the original requester or the `VotingV2` contract.

Consider adding a new `uint256 bond` parameter to the `SpamDeletionRequest` struct to record the bond amount at the time a specific request was submitted, and using this value during the call to `executeSpamDeletion`.

Update: *Partially fixed in commit* `52b32d4e1f6f7262228b7c3d7f1fa00b95c4d4b2` *of pull request #4065. Our recommendation has been adopted in the case of a successful* `spamDeletionProposal`*. However, in the case of unsuccesful proposal, the amount sent to the store contract remains affected.*

Incorrect math in slashing library comments

In the `SlashingLibrary` contract the functions `calcWrongVoteSlashPerToken` and `calcNoVoteSlashPerToken` are meant to return constants representing a slashing percentage per wrong vote or no vote that would

More specifically, the formula outlined in the inline comments

```
0.2/(10*12)
```

uses basic interest instead of compound interest and assumes that a 20% increase will be negated with a subsequent 20% loss. However, the correct formula to counteract a 20% APY over 120 votes would be

```
 1 - (1 / 1.2)**(1/120)
```

While the current deviation between both values is negligible, applying the correct formula becomes crucially important for a higher APY and more yearly votes.

Consider updating the documentation to include both the correct mathematical formula and an advisory to check the inequality

```
(1 + APY) * ( 1 - calcNoVoteSlashPerToken() )**expected_yearly_vote
```

to prevent setting any incentives for staking without participating in votes.

Update: *Fixed as of commit* `fed9d90bff9ec5052ced798e52ff36502f3cac1e` *of pull request #4066*.

Stake can be withheld indefinitely

In the `Staker` contract the function `setUnstakeCoolDown` allows the contract owner to set the `unstakeCoolDown` variable to an arbitrarily large `uint64` value. This variable controls the time that needs to pass between successful calls to `requestUnstake` and `executeUnstake`. It retroactively applies changes to all users currently within the cooldown phase.

Giving the contract owner full control over setting `unstakeCoolDown` violates the trust assumptions typically present in a staking system. Namely, stakers expect to be able to retrieve

The intended use of this contract is to be owned by the `GovernorV2` contract thereby allowing the affected stakers to control the `unstakeCoolDown` through governance proposals, which reduce the likelihood of malice. However, voters who disagree with a legitimate majority vote to extend `unstakeCoolDown` will most likely not be able to leave the staking system before the changes take effect.

Consider validating the input of the `setUnstakeCoolDown` function against an acceptable maximum cooldown time. Also consider allowing the retroactive application of a new `unstakeCoolDown` value only in cases when it acts to decrease the cooldown time of users who are actively unstaking.

Update: *Acknowledged. UMA indicated that the economic incentives between stakeholders make this scenario unlikely. The increased complexity and gas cost of the suggested recommendation does not appear justified.*

Limitations of contract migration

The `VotingV2` contract includes a `setMigrated` administrative function that sets an associated `migratedAddress` which stores the address of a future version of the voting system (e.g. V3).

The contract also includes the `onlyIfNotMigrated` and `onlyRegisteredContract` modifiers which restrict its functionality in the event of a migration. The `commitVote` and `revealVote` functions are guarded by the `onlyIfNotMigrated` modifier, which prevents voting from occcuring once the `migrationAddress` has been set. The `requestPrice`, `hasPrice`, and `getPrice` functions are guarded by the `onlyRegisteredContract` modifier. In the event of migration, the modifier restricts calls to these functions to the new Voting contract. Otherwise, the modifier restricts calls to these functions to a set of contracts that are registered with the Registry contract.

However, there are several concerns with the existing migration design:

requests.

- Voter tokens are staked on the `VotingV2` contract and there is a GAT threshold for voting. If users are directed to migrate their staked tokens from `VotingV2` to the new Voting contract, there may be a transition period where not enough users are staked on either contract to reach the GAT threshold for voting.
- Ideally, contract migration would occur during a time window when there are no active requests. Attempting to upgrade the contract while price requests are active would result in it not being able to complete the voting process on those requests due to the disabled state of the commit and reveal functions. However a price request can occur at any time, and if demand for the voting system increases in the future, it will become more difficult to chance upon a time window where there is no activity in the voting system, and there is no guarantee that such a window will exist.

To avoid a situation where the voting system cannot be upgraded, consider introducing an upgrade mechanism that allows for migration in several distinct steps while ensuring that the voting system remains functional at each stage.

Update: *Partially fixed in commit `852b60531bb1888d581046a3ccda3a68d4856f33` of pull request #4105. Forwarding price requests to the old contract have been fixed, while UMA indicated that additional measures to facilitate migration will be addressed in a future version.*

Lack of emergency administration

The `VotingV2` contract has several functions that are protected by the `onlyOwner` modifier:

- `requestGovernanceAction`
- `setMigrated`
- `setGat`
- `setRewardsExpirationTimeout`
- `setSlashingLibrary`
- `setSpamDeletionProposalBond`
- `setEmissionRate` (inherited from `Staker`)

the `GovernorV2` contract. In order for this contract-to-contract call to work with the `onlyOwner` restriction in place, the `GovernorV2` contract must become the owner of the `VotingV2` contract. This action is performed by calling the Ownable base contract function `transferOwnership()`, as demonstrated in the `GovernorV2.js` test:

```
    // To work, the governorV2 must be the owner of the VotingV2 c
    // environment, so ownership must be transferred.

    await voting.methods.transferOwnership(governorV2.options.addre
```

This change of ownership means that the `GovernorV2` contract also becomes the only valid caller for the setter functions in the list above. In order to call any of these functions, a governance action must be proposed with `Transaction` data that will invoke the target function, and subsequently executed via the `executeProposal` function.

The concern with this approach is that the voting system is potentially a single point of failure for administrative governance actions. Taking the `setGat` function as an example, if voter participation becomes insufficient to meet the GAT threshold and the GAT value needs to be lowered, a governance proposal to call `setGat` may be prevented from execution if the existing GAT value prevents that vote from succeeding. Furthermore, future versions of the DVM with new administrative functions may encounter the same problem where the current state of the Voting contract prevents a new corrective state from being applied.

To address this issue, consider developing an out-of-band system that allows UMA governance to execute emergency corrective actions for the voting system that do not rely on the voting system itself in order to succeed. For this approach, it is recommended to use a multisig wallet to control access to the `onlyOwner`-protected functions. Also consider implementing minimum and maximum values for each setter function that restrict governance changes to a safe range.

Update: *Acknowledged. UMA acknowledges that emergency administration is a concern. However, due to the complexity and severity of changes it cannot be addressed as part of this audit.*

The `Staker` contract contains the following internal virtual functions:

- `inActiveReveal` which always returns `false` and is meant to return a boolean value that indicates whether or not the voting system is currently in a reveal phase
- `getStartingIndexForStaker` which always returns `0` and is meant to return a uint64 value indicating the index of the first request from which slashing rewards and penalties apply to a new staker

Neither function is supposed to modify the contract state despite lacking a `view` identifier.

Further, the documentation of the `inActiveReveal` function states "This function should be overridden by the child contract". This suggests the Staker contract is designed to always be inherited by a child contract which overrides both virtual functions, and never deployed as a standalone contract.

To enforce the intended design, prevent accidental deployments, and enhance the clarity of your code base, consider declaring the `Staker` contract with the `abstract` keyword and removing the function body of `inActiveReveal` and `getStartingIndexForStaker` in order to force their implementation by a derived contract. Additionally, consider adding the `view` modifier to both function declarations.

Update: *Fixed as of commit `8b3084adcd51fba381e733d05b75345fa3db5e4d` in pull request #4068.*

Duplicate function execution

In the `VotingV2` contract the overloaded function `commitAndEmitEncryptedVote` that is provided for backwards compatibility mistakenly calls `commitVote` twice: The first call to `commitVote` is performed directly within the function body on line 614, and an additional call is performed indirectly by the call to `commitAndEmitEncryptedVote` on line 616.

The duplicate call is an unnecessary gas expense and leads to an unintentional duplication of `VoteCommitted` event emission.

request #4069.

Imprecise function name

In the `DesignatedVotingV2` contract, the `retrieveRewards` function name suggests that the caller will gain possession of the rewards; this assumption is further reinforced by the docstring's description of the return value as "amount of rewards that the user should receive". However, instead of transferring rewards to the user, the `retrieveRewards` function restakes user's rewards.

To avoid confusion, consider renaming the `retrieveRewards` function to `withdrawAndRestakeRewards` or some other name that more accurately describes its behavior.

Update: *Fixed as of commit* `284c6842ed05b13cfdc82cb3b5dd897507696e8f` *in pull request #4071*.

Incorrect documentation for voting commit hash

In order to submit a vote on a price request, the voter must construct an off-chain hash by encoding the following data in order and then computing the keccak256 hash: `price`, `salt`, `voterAddress`, `time`, `ancillaryData`, `currentRoundId`, and `identifier`. This can be seen in lines 559-563 of the `VotingV2` contract's `revealVote` function, where the contract constructs a reveal hash that should match the one that was committed:

```
require(
    keccak256(abi.encodePacked(price, salt, msg.sender, time, ancil
        voteSubmission.commit,
    "Revealed data != commit hash"
);
```

The structure of the commit hash is described in the docstrings of several functions within the `DesignatedVotingV2` and `VotingV2` contracts. However, none of these docstrings

- line 56: The `commitVote` docstring describes the input `hash` as "the keccak256 hash of the price you want to vote for and a random integer salt value". This description omits the required `voterAddress`, `time`, `ancillaryData`, `currentRoundId`, and `identifier`.
- line 74: The `commitAndEmitEncryptedVote` docstring describes the input `hash` as "keccak256 hash of the price you want to vote for and a int256 salt". This description omits the required `voterAddress`, `time`, `ancillaryData`, `currentRoundId`, and `identifier`.
- line 91: The `revealVote` docstring describes the input `price` as "used along with the `salt` to produce the `hash` during the commit phase". This description is technically correct but implies these are the only two values required, omitting `voterAddress`, `time`, `ancillaryData`, `currentRoundId`, and `identifier`.
- line 92: The `revealVote` docstring describes the input `salt` as "used along with the `price` to produce the `hash` during the commit phase". This description is technically correct but implies these are the only two values required, omitting `voterAddress`, `time`, `ancillaryData`, `currentRoundId`, and `identifier`.

- In `VotingV2.sol`:
  - line 488 : The `commitVote` docstring describes the input `hash` as "keccak256 hash of the `price`, `salt`, voter `address`, `time`, current `roundId`, and `identifier`". This description omits the required `ancillaryData` variable.
  - line 526: The `revealVote` docstring states that "The revealed `price`, `salt`, `address`, `time`, `roundId`, and `identifier`, must hash to the latest `hash`". This comment omits the requires `ancillaryData` variable.

Furthermore, while the type of all the encoded values can be deduced by examining the input parameters and body of the `revealVote` function, this type information is not explicitly provided to voters in the documentation of the commit functions where the hash is inputted.

To avoid any misunderstanding about the hash encoding and the potential for unintentional wrong votes, consider carefully documenting the exact format of the commit hash in all docstrings where it is referenced.

Update: *Fixed as of commit* `4f8798aeb90b1b3419f6e6da3ae1531ef07c058b` *in pull request #4072.*

Lack of input verification in library functions

In the `SpamGuardIdentifierLib` contract, the function `_constructIdentifier` does not document or enforce the range of allowed input values for its function parameter `uint256 id`.

Supplying a value of `10^11` or greater as parameter `id` will lead to silent unexpected behavior and potentially cause the generation of duplicate identifiers, because the concatenation of the decimal string `AdminIdentifierLib._uintToUtf8(id)` and the identifier "SpamDeletionProposal " is limited to 32 bytes. While the current usage of `SpamGuardIdentifierLib` within `VotingV2` appears safe, this might change due to future modifications of the code base.

Consider both explicitly documenting the valid range of the input parameter and including a `require` statement that enforces this range.

Update: *Fixed as of commit* `97ca84895e094a825c55c4b905e262c771a6c80a` *in pull request #4073 by reducing the input to* `uint32`.

Missing documentation

Many of the functions, events, and variables within the codebase lack documentation.

Contracts that do not have a docstring:

- `SlashingLibrary`
- `SpamGuardIdentifierLib`

Functions that do not have a docstring:

- `updateTrackers`, `getStartingIndexForStaker` in `Staker.sol`
- `computeRoundToVoteOnPriceRequest` in `VoteTimingV2.sol`
- `getNumberOfPriceRequests`, `_getPriceRequest`, `_encodePriceRequest`, `_freezeRoundVariables`, `_priceRequestResolved`, `_computeGat`, `_getRequestStatus`, `unsafe_inc`, `unsafe_inc_64`, `_getIdentifierWhitelist`, `_requireNotMigrated`, `_requireRegisteredContract` in `VotingV2.sol`

Docstrings that are incomplete:

- `commitVote` in `DesignatedVotingV2.sol` is missing the @param for `ancillaryData`
- `_constructIdentifier` in `SpamGuardIdentifierLib.sol` is missing the @param for `id`
- `init` in `VoteTimingV2.sol` is missing the @param documentation for all input parameters
- `getVoterFromDelegate` in `VotingV2.sol` is missing the @param for `caller`
- `constructor` in `VotingV2.sol` is missing the @param for `_spamDeletionProposalBond` and `_startingRequestIndex`

Many events and storage variables throughout the codebase also lack explanatory comments.

Finally, some useful documentation was recently removed:

- Documentation for the `NewProposal` and `ProposalExexcuted` events in `GovernorV2.sol`
- Documentation explaining the `propose` function's public visibility in `GovernorV2.sol`

Missing documentation hinder reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec). Also consider providing explanatory comments for all events, structs, and storage variables that indicate their intended purpose.

Update: *Partially fixed as of commit* `3d68ad7fa471d4d763ab3ae9e6e5052bad1dce8e` *in pull request #4077. The following functions still lack docstrings:*

- `getIdentifierWhitelist` *in* `GovernorV2.sol`
- `updateTrackers`, `getStartingIndexForStaker` *in* `Staker.sol`
- `_requireRegisteredContract` *in* `VotingV2.sol`

Missing error messages in require statements

Within the codebase, there are multiple `require` statements that lack error messages.

- In `VotingV2.sol`:
  - The `require` statement on line 552
  - The `require` statement on line 771
  - The `require` statement on line 810
  - The `require` statement on line 961
  - The `require` statement on line 993
  - The `require` statement on line 999
  - The `require` statement on line 1188
  - The `require` statement on line 1192
- In `VoteTimingV2.sol`:
  - The `require` statement on line 24
  - The `require` statement on line 25

Consider including specific, informative error messages in `require` statements to improve overall code clarity and to facilitate troubleshooting whenever a requirement is not satisfied.

Update: *Fixed as of commit* `f29085860f7a32a480825e06b8f69c3456ea19dc` *in pull request #4074.*

meant to allow setting and providing the time via an additional external contract. During deployment to a production environment the inheritance of `Testable` is meant to be kept while disabling the external time setter by passing the zero address to the testable constructor.

This testing methodology increases both the code size and the risk of deployment mistakes. Further, it allows multiple time sources for different contracts within the project's test cases, thereby increasing the likelihood of erroneous tests.

Consider replacing the `Testable` contract with equivalent functionality provided by the project's test framework. For example, consider using `evm_setNextBlockTimestamp` in Hardhat or `vm.warp` in Foundry.

If maintaining a dedicated function for time setting in the contract code is desired, consider reversing the order of inheritance by implementing a virtual function `getCurrentTime` which returns `block.timestamp` in the production versions of `ProposerV2` and `GovernorV2`, and deriving specialized testing contracts that override `getCurrentTime` to implement the time setting capabilities of `Testable`.

Update: *Fixed as of commit `90b6634f6daebf02100f4a1b8aa6bca22bc64cc8` in pull request #4095.*

Unlocked Solidity version pragma

Throughout the codebase, the Solidity version used is `^0.8.0`, which allows compilation with any version of Solidity from 0.8.0 up to the latest release. This may lead to unexpected behavior if the code is deployed with a different Solidity version than was used during testing. Further, allowing old versions of Solidity leaves the code potentially vulnerable to known security bugs which have already been patched. The official guidance is to always use the latest Solidity release when deploying contracts. When a bug is discovered that affects a range of Solidity versions, the general policy of the Solidity team is to only apply the fix to the latest release (i.e. no backporting of security updates).

Consider locking the version pragma to the same Solidity version used during development and testing. Also consider setting this version to be the latest release.

pragma has been updated to [illegible] in most contracts, however `Staker`, `SlashingLibrary` and `SpamGuardIdentifierLib` retain an unlocked pragma.

Withdraw role is not configured

The `DesignatedVotingV2Factory` contract inherits from the `Withdrawable` contract, a base contract which provides the ability to create a Withdraw role and assign it to one or more addresses to grant them the ability to withdraw ETH or ERC20 tokens from the contract by using the corresponding `withdraw` and `withdrawErc20` functions.

The `Withdrawable` contract provides two functions for configuring this role, `_createWithdrawRole` and `setWithdrawRole`, and the documentation for these functions indicates that one or the other must be called by the derived contract (`DesignatedVotingV2Factory` in this case) in order for withdraws to function properly. However, the `DesignatedVotingV2Factory` contract does not call either function, rendering the `withdraw` and `withdrawErc20` functions inoperable.

Given the intended purpose of the `DesignatedVotingV2Factory` contract, it does not appear that it requires the `Withdrawable` feature. Consider removing the inheritance from `Withdrawable` and the corresponding import of `Withdrawable.sol`.

Update: *Fixed as of commit `88c79f4403d6f61715fab044c8933e8d90ca4366` in pull request #4078.*

Notes & Additional Information

Inconsistent usage of reentrancy protection

In the `GovernorV2` contract, the function `executeProposal` is protected against reentrancy by the checks-effects-interactions pattern. In contrast, the `ProposerV2` contract provides an additional layer of safety against re-entrancy attacks by importing the `Lockable` contract and using its modifiers.

Update: *Fixed as of commit* `32e27d1ca004a33a8f0ac256ccfc4f526666eeb0` *in pull request #4080. The* `nonReentrant` *modifier was applied to the* `executeProposal` *function as recommended. Additionally, the* `nonReentrant` *modifier was also applied to the following functions:*

- *The* `propose` *function in* `GovernorV2.sol`
- *The* `stake`, `requestUnstake`, `executeUnstake`, `withdrawRewards`, *and* `withdrawAndRestake` *in functions in* `Staker.sol`
- *The* `requestPrice`, `commitVote`, `revealVote`, `setDelegate`, `setDelegator`, `signalRequestAsSpamForDeletion`, *and* `executeSpamDeletion` *functions in* `VotingV2`

Duplicate code

In the `VotingV2` contract, lines 1136-1140 in the `_priceRequestResolved` function perform the removal of an element from the `pendingPriceRequests` array.
The `executeSpamDeletion` function within the same contract re-implements this same code on lines 1010-1016, with the only difference being that `priceRequest` has been replaced with `priceRequests[requestId]`.

Consider moving this duplicated code into a new shared function.

Update: *Fixed as of commit* `ce97e912402996f77f2c8cf22c236f599de9456` *in pull request #4081 by moving the duplicated code into a new shared function named* `_removeRequestFromPendingPriceRequests`.

Duplicate computations

In the `ProposerV2` contract, the function `resolveProposal` performs two subsequent calls to `AdminIdentifierLib._constructIdentifier(id)` in line 91 and line 99 without storing the return value.

Consider performing only one call and capturing the return value to enhance the gas efficiency of your code.

Erroneous imports

The `ProposalV2` contract erroneously imports `Voting.sol` instead of `VotingV2.sol`.

Currently, all function signatures used within `ProposalV2` are available in both `Voting.sol` and `VotingV2.sol`, thereby preventing any direct consequences.

Consider upgrading the import to `VotingV2.sol` for consistency.

Update: *Fixed as of commit* `0267ae4d344df061c9e392dc1cf862de82ff43d6` *in pull request #4083 by replacing the* `Voting.sol` *import with* `OracleAncillaryInterface.sol`*, which provides an interface for the* `hasPrice` *and* `getPrice` *voting functions used by* `ProposerV2.sol`.

WithdrawnRewards is emitted for zero rewards

The `withdrawRewards` function in the `Staker` contract emits a `WithdrawnRewards` event to log the minting of rewards tokens to the voter who called the function. However, the event is emitted outside the `if` block that checks for a non-zero value of `tokensToMint`. If no rewards are minted, the `WithdrawnRewards` event will be logged even though no state change occurred, wasting gas and possibly leading to confusion.

Consider only emitting the `WithdrawnRewards` event if `tokensToMint` is not zero.

Update: *Fixed as of commit* `703662231d4399a869e4ce990868cb2061f9e402` *in pull request #4084*.

Lack of immutable identifier

In the `ProposerV2` contract, the mutable state variables `token`, `governor`, and `finder` are only set once within the constructor.

In the `Staker` contract, the mutable state variable `votingToken` is only set once within the constructor. Consider declaring these state variables as `immutable` to increase gas efficiency and enhance the clarity of your code base.

Inconsistent declarations for `finder` variable

Many contracts rely on the `Finder` contract to lookup the addresses of other deployed UMA contracts. While most contracts in the DVM implementation store a `FinderInterface` type in their `finder` storage variable, the `Proposer` and `ProposerV2` contracts store a `Finder` type instead.

Additionally, some contracts declare their `finder` variable with `private` visibility, while others declare `finder` as a `public` variable.

Consider making these declarations consistent within the codebase by uniformly applying the same contract type and visibility to all instances of the `finder` storage variable.

Update: *Partially fixed as of commit `2f5f667994c96612e95417de24f526b94a37b6c0` in pull request #4086. In the `VotingV2` contract, the `finder` variable's visibility was changed from `private` to `public`, but there is still an inconsistent application of private and public visibility applied to this variable across `ProposerV2` (public), `GovernorV2` (private), `DesignatedVotingV2` (private), and `VotingV2` (public). Additionally, `Proposer` and `ProposerV2` still use the `Finder` type.*

Inconsistent function naming

Throughout the codebase, a leading underscore is used to denote private and internal functions. However, several functions that are marked `internal` do not follow this pattern. To avoid confusion, consider the following renaming suggestions:

- In `Staking.sol`:
  - `inActiveReveal` should be `_inActiveReveal`
  - `getStartingIndexForStaker` should be `_getStartingIndexForStaker`
- In `VotingV2.sol`:
  - `applySlashToVoter` should be `_applySlashToVoter`

Inconsistent event indexing

Within the `VotingV2` contract, the `PriceRequestAdded` event applies the `indexed` keyword to the `time` variable but not the `requester` variable. All other events within this contract that emit a `time` value do not apply indexing to it, and all other events apply indexing to any addresses they emit, except this one.

To use indexing consistently, in the `PriceRequestAdded` event consider removing the `indexed` keyword from `time` and adding it to `requester`.

Update: *Fixed as of commit* `c6a8f7e088aefb5d5b5f77407e3ef400fe743d77` *in pull request #4088.*

Misleading variable name

In the `updateTrackersRange` function, the `indexTo` parameter is the "last price request index to update the trackers for". On line 810 there is a check that `indexTo <= priceRequestIds.length`; at first glance this appears to be an off-by-one error because array indexing starts at index 0 and the index is considered out of bounds if it is greater than or equal to the length of the array.

However, in the `_updateAccountSlashingTrackers` function, the for-loop test condition ensures that the maximum index value used to access the `priceRequestIds` array is `indexTo - 1`, thereby preventing any out-of-bounds access. Thus the logic appears correct, but the name `indexTo` is misleading because it specifies the maximum element number to include in the update, not the maximum index number.

To avoid confusion, consider renaming the `indexTo` variable, or update the documentation to clarify that this is not an array index value.

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

> *We chose to not implement any change for this note.*

Use of magic values

Using inline magic values hinders code legibility and increases the chance of introducing erroneous values.

Consider assigning all magic values to descriptive constants and replacing all semantically matching occurrences of the value with its respective constant.

Update: *Fixed as of commit* `baf99022bb69c0744a5b13b26610245c49c74b93` *in pull request #4089 by eliminating the magic value* `1e18` *and interpreting any non-zero price value as a "yes" vote. This behavior is consistent with the existing price validation that is performed in the* `executeProposal` *function.*

Misleading documentation

The code base contains multiple occurrences of misleading or incorrect documentation:

- In the contract `DesignatedVotingV2`, the comment on line 21 "Is also permanently permissioned as the minter role" appears to be a copy & paste error as a minting function is not part of the contract. Consider removing the comment.
- In the contract `SlashingLibrary`, the docstring for `calcSlashing` on line 55 contains the erroneous expression "cross-chain calls". Consider altering it to "cross-contract calls".
- In the contract `VotingV2`, the comment on line 161 describes the `spamDeletionProposals` variable as "Maps round numbers to the spam deletion request". However, mapping indices to values is a property every array possesses. Consider revising the comment to give a more contextual description of the identifier and its usage.
- In the contract `VotingV2`, the docstring for `getRoundEndTime` on lines 716-719 appears to be a copy & paste error that duplicates the docstring of the `getCurrentRoundId` function. Consider rewriting the docstring.
- In the contract `VotingV2`, the comment on line 767 "This method is public because calldata structs are not currently supported by solidity." appears to be a copy & paste error, because no such struct is present in the function's arguments. Consider removing the comment.

Missing license identifier

The following files do not contain an SPDX License identifier:

- `SpamGuardIdentifierLib.sol`
- `AdminIdentifierLib.sol`

Missing license agreements can lead to legal disputes and undesired forms of code usage.

Consider adding a license identifier to all mentioned files.

Update: *Fixed as of commit* `2da561bcee2c72ae36eb4257f04c68537c6e2cd5` *in pull request #4091*.

Missing Solidity version pragma

The `SpamGuardIdentifierLibs.sol` and `AdminIdentifierLib.sol` files do not define a Solidity version pragma.

It is considered good practice to always specify the desired compiler version by using the Solidity version pragma. If the contract contains non-trivial operations the resulting bytecode might differ between Solidity versions including security relevant breaking changes and non-trivial bug fixes.

Consider defining Solidity version pragmas for all Solidity source files in this project.

Update: *Fixed as of commit* `2da561bcee2c72ae36eb4257f04c68537c6e2cd5` *in pull request #4091*.

Function with return type does not return value

In the `DesignatedVotingV2` contract, both the signature of the `retrieveRewards` function and its docstring indicate that it returns a value, but the function has no `return` statement.

Consider adding the missing `rewardsMinted` return value in order to make the `retrieveRewards` function behavior match its public interface.

does not return a value.

Inconsistent use of named return variables

There is an inconsistent usage of named return variables across the codebase.

Occurrences of named return values are:

- The `getResolvedPrice` function in the `ResultComputationV2` contract
- The `calcSlashing` function in the `SlashingLibrary` library
- The `_getIdentifierWhitelist` function in the `GovernorV2` contract
- The `_getIdentifierWhitelist` function in the `VotingV2` contract
- The `propose` function in the `ProposerV2` contract

Consider adopting a consistent approach to return values by removing all named return variables.

Update: *Fixed as of commit* `b62a8c90355b6e6c8643e65d354a9a253962073b` *in pull request #4092. For clarity, the UMA team has retained the use of named return values only for functions that return multiple values. As a result, the* `getResolvedPrice` *and* `calcSlashing` *functions were left unchanged.*

Naming issues

To favor explicitness and readability, several parts of the contracts may benefit from better naming. Our suggestions are:

- In the `VotingV2` contract, consider renaming the constant `ancillaryBytesLimit` to `ANCILLARY_BYTES_LIMIT`.
- In the `VotingV2` contract, the `deletedRequests` mapping is used to optimize the processing of the `priceRequestIds` array by skipping certain elements. The reason to skip an element could be deletion via the `executeSpamDeletion` function or the detection of a rolled vote within the `_updateAccountSlashingTrackers` function. The name `deletedRequests` is misleading as it only describes one of two origins for skips during request processing. Consider renaming the mapping to indicate the purpose of efficient request processing, e.g. `skippedRequestIndexes`.

Update: *Fixed as of commit* `bf04d1dce70eadbdeeff4760fca1cae2ecc3651c` *in pull request #4094. All naming suggestions were implemented on the contracts in scope for this audit.*

Public functions can be marked as external

The following functions are currently marked with `public` visibility, but can be declared `external` because they are never called internally by their own contracts or by child contracts:

- In `DesignatedVotingV2.sol`:
  - `retrieveRewards`
- In `GovernorV2.sol`:
  - `propose`
- In `Staker.sol`:
  - `requestUnstake`
  - `executeUnstake`
  - `withdrawAndRestake`
  - `setEmissionRate`
  - `setUnstakeCooldown`
- In `VotingV2.sol`:
  - `commitVote` (backwards compatible version only)
  - `commitAndEmitEncryptedVote` (backwards compatible version only)
  - `executeSpamDeletion`
  - `getCurrentRoundId`
  - `getNumberOfPriceRequests`
  - `getPendingRequests`
  - `getPrice` (backwards compatible version only)
  - `getPriceRequestStatuses` (backwards compatible version only)
  - `getRoundEndTime`
  - `getSpamDeletionRequest`
  - `requestGovernanceAction`
  - `requestPrice` (backwards compatible version only)
  - `revealVote` (backwards compatible version only)

- `setRewardsExpirationTimeout`
- `setSlashingLibrary`
- `signalRequestsAsSpamForDeletion`
- `updateTrackers`
- `updateTrackersRange`

- In `SlashingLibrary.sol`:
  - `calcSlashing`

Consider changing the visibility of these functions to `external` in order to reduce gas costs and clarify that these functions will only ever be called by external contracts.

Update: *Fixed as of commit `940ccda23259963687988c3cf7ae954d599980e0` in pull request #4102. Note that the `retrieveRewards` function has been renamed to `withdrawAndRestakeRewards`.*

Redundant code

Consider making the following changes to eliminate unnecessary code:

- In `Staker.sol`:
  - line 26: The `override` keyword can be removed
  - line 115: The `override` keyword can be removed
  - line 151: The `override` keyword can be removed
  - line 178: The `override` keyword can be removed
  - line 199: The `override` keyword can be removed
  - line 209: `(tokensToMint)` can be changed to `tokensToMint`
- In `VotingV2.sol`:
  - lines 1005-1006: The `uint256(i)` casts can be changed to `i`

Update: *Partially fixed as of commit `764d555eab7d58315561cd98904a06c24c980999` in pull request #4096. The changes in `Staker.sol` have been fully addressed, while the change to `VotingV2.sol` remains unaddressed. Also note that pull request #4096 inadvertently*

Amount to stake or unstake can be zero

Within the `Staking` contract, there are two functions that allow users to stake and unstake a specific amount of UMA tokens: `stake` and `requestUnstake`. Both functions have an `amount` parameter that specifies how many tokens to stake or unstake, but neither function checks whether this amount is zero. If the caller specifies an `amount` value of 0 for either function, all of the logic is still executed, wasting the user's gas.

In both `stake` and `requestUnstake`, consider adding a check that `amount` is a non-zero value before proceeding.

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

> *We don't have any issue with a user unstaking 0, which is a no op.*

Too many digits in numeric literals

Within the `SlashingLibrary` contract, the functions `calcWrongVoteSlashPerToken` and `calcNoVoteSlashPerToken` both return a hard-coded value of `1600000000000000`. Numeric literals with a large number of digits are more difficult to interpret, requiring the reader or reviewer to count the number of digits to ensure correctness.

Solidity supports scientific notation representation for large values. Consider replacing these integer values with their more compact equivalent representation `1.6e15`.

Update: *Fixed as of commit* `1d11705161a9d2d49943eace26ab6239886c21e8` *in pull request #4097.*

Typographical errors

Consider addressing the following typographical errors:

- In `DesignatedVotingV2.sol`:
  - line 54: "EG" should be "E.g."
  - line 71: "Eg:" should be "E.g."

- In `GovernorV2.sol`:
  - line 98: “an an array” should be “an array”
- In `ProposerV2.sol`:
  - line 115: “system, itself” should be “system itself”
- In `SlashingLibrary.sol`:
  - line 59: Remove space before “wrongVoteSlashPerToken”
- In `Staker.sol`:
  - line 13: “prorate” should be “pro rata”
  - line 53: “voterPendingUnStake” should be “voterPendingUnstake”
  - line 91: “prorate” should be “pro rata”
  - line 93: Remove erroneous comment line
  - line 111: “a active reveal” should be “an active reveal”
  - line 175-176: Remove the incomplete sentence that begins with “Note that this…”
  - line 230: “prorate” should be “pro rata”
  - line 231: “prorate” should be “pro rata”
- `In VoteTimingV2.sol`:
  - line 31: “roundID” should be “round ID”
  - line 44: “round Id” should be “round ID” (or `roundId`)
- In `VotingV2.sol`:
  - line 26: “UMA’s DVM mechanism” should be “UMA’s DVM” (mechanism is redundant)
  - line 52: “UINT_MAX” should be “UINT64_MAX”
  - line 59: “uint56” should be “uint64”
  - line 295: “eg” should be “E.g.”
  - line 311: “eg” should be “E.g.”
  - line 327: “eg” should be “E.g.”
  - line 391: “eg” should be “E.g.”
  - line 413: “eg” should be “E.g.”
  - line 485: “EG” should be “E.g.”
  - line 528: “EG” should be “E.g.”
  - line 585: “commits” should be “Commits”
  - line 588: “Eg:” should be “E.g.”
  - line 589: “of for the price request” should be “of the price request”

- line 631: “two way” should be “two-way”
- line 632: “of the delegate” should be “of the delegator”
- line 656: “of type `PendingRequest`” should be “of type `PendingRequestAncillary`“
- line 866: “cant” should be “can’t”
- line 943: “it’s” should be “its”
- line 907: Remove blank line
- line 946: “diregarded” should be “disregarded”
- line 959: “sequently” should be “sequential”
- line 1134: “pending request” should be “pending requests”

Additionally, some errors were found in out-of-scope files:

- In `packages/core/README.md`:
  - line 92: “optimsim-up” should be “optimism-up”
- In `packages/core/contracts/common/implementation/MultiCaller.sol`:
  - line 1: “Uniswaps’s” should be “Uniswap’s”
- In `packages/core/test/oracle/VotingV2.js`:
  - line 2300: “i.e” should be “i.e.”
  - line 2481: “cant” should be “can’t”
  - line 2482: “cant” should be “can’t”
  - line 2630: “Dont” should be “Don’t”
  - line 2708: “cant” should be “cant”
  - line 2815: “dont” should be “don’t”
  - line 2857: “i.e” should be “i.e.”

Update: *Fixed as of commit `69e9574f3ffb3b9085e58b731862c92728134710` in pull request #4098. All typos were corrected except for those identified in the `VotingV2.js` file, which was out of scope.*

Unnecessary use of SafeMath library

the SafeMath library is redundant in this case.

Consider removing the `SafeMath.sol` import from `GovernorV2` and replacing the `sub` function with the `-` operator.

Update: *Fixed as of commit `9714b2df60965edd99bec6cb674f01f6d5fb141b` in pull request #4099.*

Unused function parameters

The `SlashingLibrary` contract defines a standard interface for the `calcWrongVoteSlashPerToken`, `calcWrongVoteSlashPerTokenGovernance`, and `calcNoVoteSlashPerToken` functions. Each function accepts the following input parameters: `totalStaked`, `totalVotes`, and `totalCorrectVotes`. These variables are not used at present but are intended for future use. However, because they are not used, the Solidity compiler will raise an "Unused function parameter" warning for each instance.

Although the unused parameters are harmless in this case, the goal should be to have zero compiler warnings, in order to prevent other warnings mixed in with these from going unnoticed during compilation. Consider adding the following block of no-op code to the body of each of the aforementioned functions in order to suppress the compiler warnings:

```
totalStaked;  // currently unused
totalVotes;  // currently unused
totalCorrectVotes;  // currently unused
```

This code would then be removed at a later date when these variables are used to perform slashing calculations.

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

> *We are ok with the compiler warnings for this and are not too worried about this.*

Unused imports

- In the `VotingV2` contract, the `AncillaryData.sol` library
- In the `VotingV2Interface` contract, the `FixedPoint.sol` library

To improve readability and avoid confusion, consider removing the unused imports.

Update: *Fixed as of commit* `160c985d9560b78d41b2998d2ca40f4c1ef297b4` *in pull request #4100 and commit* `0f9bdee0d0584254a93c48fddf3987e2d8559a6e` *in pull request #4095*.

State variable visibility not explicitly declared

Within `VotingV2.sol` the state variable `spamDeletionProposalBond` lacks an explicitly declared visibility.

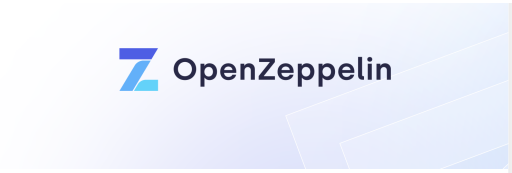
For clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

Update: *Fixed as of commit* `e3ae35386ddaa674145816c80a11b2117fff44ad` *in pull request #4101*.

Conclusions

One critical and one high severity issue were found and subsequently resolved. Some changes were proposed to follow best practices and reduce potential attack surface. The supporting usage of a monitoring system and the addition of emergency administration functionality was recommended.

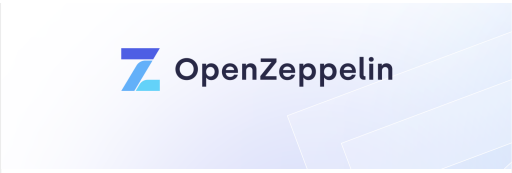
Related Posts



Beefy Zap Audit

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

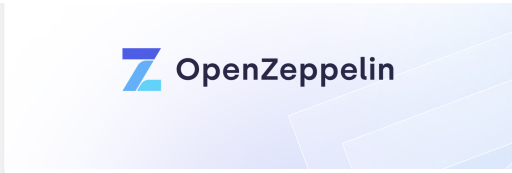
Security Audits



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