OpenZeppelin

UMA Continuous Audit

OPENZEPPELIN SECURITY | APRIL 9, 2021

Security Audits

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 as well as the optimistic oracle and a new financial contract template. In this audit we are taking an iterative approach where we will review individual pull requests as they are developed by the UMA team. We will repeatedly update this report with any new findings for the duration of our engagement. Unless otherwise stated, the scope includes all solidity files affected by the specified pull requests.

Findings

To facilitate iterative reviews, these findings are listed in reverse chronological order.

Pull Request 3211

The PR addresses a dust-minting issue that was found by the UMA team, where the calculation that converts `LongShortPair` tokens to the equivalent amount of collateral rounds down in all cases. This commit updates the calculations to round down when the contract sends collateral and round up when the contract receives collateral, which ensures the contract remains fully collateralized. We don't have any comments.

Pull Request 3207

The PR addresses the comments from our review of PR 3184. We don't have any comments.

comments.

Pull Request 3188

This PR addresses the comments from our review of PR 3061 and our batch review of PRs 3054, 3082, and 3092. We have the following comment:

- The comment explaining the `constructPrefix` function says "appended as a prefix" instead of "prepended".

Update: *This was addressed in Pull Request 3208.*

Pull Request 3189

This PR addresses the comments from our review of PR 2969, including an explanation of why the `chainid` suggestion was not adopted. We don't have any comments.

Pull Request 3184

This PR introduces a new `pairName` property of `LongShortPair` financial contracts and allows the creator to set custom values for the optimistic oracle's liveness and bond. It also refactors the `LongShortPair` and `LongShortPairCreator` contracts to use an initialization struct instead of a large number of individual parameters.

Our comments are:

- The `createLongShortPair` `@param` comment does not list the `pairName` field.
- The `pairName` is not validated to have non-zero length.
- The `optimisticOracleLivenessTime` is described as a "timer" (instead of a "time"), in its definition, the `LongShortPair` constructor comments and the `createLongShortPair` function comments.
- Our assumption was that the `expirationTimestamp` is only 64 bits long so that it can share a storage location with the `contractState` variable. However, the new

Update: *All issues were addressed in Pull Request 3207.*

Pull Request 3167

This PR allows the creator of a `LongShortPair` financial contract to explicitly set the names and symbols used for the tokens, rather than conforming to a predetermined pattern. We don't have any comments.

Pull Request 3089

This PR modifies the chainbridge `SourceGovernor` and `SinkGovernor` contracts so the cross-chain messages cannot specify an ETH amount. We don't have any comments.

It also implements the Polygon Tunnel for governance actions. In particular, it introduces `GovernorRootTunnel` and `GovernorChildTunnel`. Our only comment is that line 19 of `GovernorChildTunnel.sol` references a non-existent `_publishPrice` function.

Update: *This was addressed in Pull Request 3208.*

Pull Requests 3054, 3082 and 3092

These PRs includes many changes that were not reviewed. In particular, they move Chainbridge files and introduce new Polygon files to an `external` directory. They also introduce test files that were not reviewed.

Along with PR 3061, PR 3054 introduces the `AncillaryData` library to help construct ancillary data strings in a standard format. The `FundingRateApplier` and `OptimisticOracle` contracts now use this library when constructing price requests. Our comments for these contracts are:

- In `AncillaryData.sol` on line 8 "libraries" should be "library".
- In the inline docs for `AncillaryData`, "LHS" should be written out as "left hand side".

The [illegible] function is confusingly named, because it doesn't append anything. The name `_constructPrefix` or similar would be clearer.
- In the `OptimisticOracle`, `public` `view` functions do not use reentrancy guards. We have not identified a scenario that requires reentrancy protection, but we thought it is worth mentioning because reentrancy guards are used on `public` `view` functions extensively throughout the code base, so this may be an oversight.

The PRs also implement the Polygon Tunnel for Oracle requests. In particular, they introduce the `OracleBaseTunnel`, `OracleChildTunnel` and `OracleRootTunnel`. Our comments are:

- The `OracleBaseTunnel` comment claims that the contract ensures price request data is not duplicated, but the design explicitly allows duplicate price requests and responses to traverse the tunnel. Consider clarifying the comment.
- The `OracleBaseTunnel` events do not index the `requestHash` parameter, even though this parameter would naturally be used to track the request.
- The contracts do not use reentrancy guards. We have not identified a scenario that requires reentrancy protection, but we thought it is worth mentioning because reentrancy guards are used extensively throughout the code base, so this may be an oversight.
- The `OracleChildTunnel` contract does not validate that price requests have sufficiently short ancillary data (after the context is added). This could lead to price requests that cannot be resolved. Consider validating the length of the ancillary data before initiating a price request.
- In `OracleBaseTunnel` on line 72 “dentifier” should be “identifier”.
- In `OracleBaseTunnel` on line 8 “lifecyle” should be “lifecycle”.
- In `OracleChildTunnel` on line 117 and on line 118 “translateable” should be “translatable”.

The PR also introduces a README file with a brief overview of the Polygon Tunnel architecture. Our comments are:

- On line 42 “timeley” should be “timely”.
- On line 40 “beign” should be “being”.

Pull Request 3061

This PR introduces the `AncillaryData` library. It also modifies the `_getAncillaryData` function of the `FundingRateApplier` contract and the `_stampAncillaryData` function of the `OptimisticOracle` contract. Further, it removes some revert reasons in the code base to save bytecode. However, as the code is modified again in Pull Request 3054, we will defer the majority of our comments to our review of that pull request.

Our only immediate comments are as follows:

- The `requestPrice` function of the `OptimisticOracle` now calls `stampAncillaryData`, which is a pass-through function for the internal `_stampAncillaryData` function. Consider calling the internal function directly for simplicity and consistency with the rest of the contract.
- In the `OptimisticOracleInterface` there is a comment about how the DVM could refuse to accept a dispute "…with ancillary data length of a certain size.". Something along the lines of: "…with ancillary data length *over* a certain size" may be more closely aligned with the implementation.

Update: *All issues were addressed in Pull Request 3188.*

Pull Request 3016

This is one of the pull requests that was implicitly included when we reviewed the Long-Short-Pair template. However, it also introduced a new `burnFrom` function to the `ExpandedERC20` contract, which lets anyone with the `Burner` role burn tokens from any address. In particular, the `LongShortPair` contract uses this to easily dispose of user tokens, at the request of the user (when redeeming or settling them for collateral), without having to take possession of them first.

There are two additional consequences that should be noted:

- The `ExpandedERC20` tokens are also used in the `ExpiringMultiParty` and `Perpetual` financial templates. Since the `Owner` and `Burner` roles are limited to these templates, and they don't invoke the `burnFrom` function, there is no direct effect to the security model.

observation does not apply to the UMA Voting token that has already been deployed.

Pull Request 3152

This PR addressed the bulk of our comments from our review of the LSP template.

We have the following comments:

- The two `require` statements added to confirm the value of the `burnFrom` method, and the two `require` statements to confirm the return value of the `mint` method lack associated error messages. Consider adding an informative error message.
- The `LongShortPair` contract still switches between 1 and 1e18 to represent 100%.
- The libraries are still inconsistent about whether the `computeExpiryTokensForCollateral` (now `percentageLongCollateralAtExpiry`) function should ensure the parameters have been set – as described in the 3rd paragraph of our LSP template review.
- The new explanation for how `RangeBondLongShortPairFinancialProductLibrary` allocates collateral has some mistakes:
- It says “function’s” instead of “function”
- It says “discreet” instead of “discrete”
- There is an unnecessary period in “1)”
- The comment following “1)” is confusing, because it adds punctuation to our suggestion in unexpected places. In our suggestion, the phrase “(collateral tokens * lowPriceRange) is the notional value of the bond” was intended as a preamble before enumerating the cases. The second sentence was intended to imply that if the expiry price is below the `lowPriceRange`, then the collateral is worth less than the notional value of the bond. The current comment implies that the notional value of the bond depends on the expiry price.

Pull Request 3133

This PR partially addressed our comment from our review of the LSP template, wherein we noted that some allowances and permissions that were implicitly required were not documented.

Pull Request 3132

This PR implements an additional `require` in the `constructor` of the `LongShortPair` contract. `collateralPerPair` is now required to be greater than zero.

We don’t have any comments.

Pull Request 3130

This PR addressed our comment from our review of the LSP template, wherein we noted that `setLongShortPairParameters` would allow anyone to overwrite parameters at any time. The function now checks that the parameters have not already been set.

We don’t have any comments.

Pull Request 3131

This PR partially addressed our comment from our review of the LSP template, wherein we noted a lack of non-reentrant modifiers.

We have the following comments:

- The `public` function `getPositionTokens` in `LongShortPair.sol` is still lacking a non-reentrant modifier.

Long-Short-Pair template

The Long-Short-Pair template was developed and modified over several pull requests. Instead of reviewing the PRs individually, we reviewed the completed files. In particular, we reviewed the files in the `long-short-pair` folder and the `long-short-pair-libraries` folder at commit `b508f536ddfd94a79f93f633142bdc868b73461c`.

Our most important observation is that the `setLongShortPairParameters` function of the `RangeBondLongShortPairFinancialProductLibrary` contract allows anyone to overwrite the parameters at any point in time to maliciously change the payout structure.

set. In particular, the

requires the strike to have been set, but the other libraries do not perform the equivalent check. Consider ensuring the configuration parameters have been set in all libraries before calculating the appropriate collateral distribution.

We understand that the UMA team intend to include `nonReentrant` and `nonReentrantView` modifiers on all external and public functions. However, the modifier is missing from the `create` and the `getPositionTokens` functions of the `LongShortPair` contract. Moreover, it's missing from the `view` functions on all the libraries. Consider including the modifiers where appropriate.

The `redeem` function is restricted by the `preExpiration` modifier. However, if a user settles an equal number of long and short tokens, it will have the same effect (with a different event emitted) as token redemption, but it requires the user to wait for the price to resolve. Consider allowing users to call the `redeem` function at all times.

The `_getSyntheticDecimals` function is described and named as a `private` function, but it is `public`. Consider making it private.

The `createLongShortPair` function of the `LongShortPairCreator` contract grants Minter and Burner roles for the synthetic tokens to the new `LongShortPair` contract. However, the creator contract still retains those roles. Consider renouncing them to reduce the attack surface.

The `create` function of the `LongShortPair` contract discards the return value when minting synthetic tokens. Similarly, the `settle` function discards the return value when burning the tokens. Consider requiring that the `mint` and `burnFrom` functions returns `true`.

The `computeExpiryTokensForCollateral` function of the `RangeBondLongShortPairFinancialProductLibrary` uses a complicated closed-form expression to determine how much collateral to assign to the long position. Instead, consider enumerating the cases explicitly so it is easier to reason about. This would reduce to something like:

`solidity`

```
// the long position is entitled to whatever is left
if(expiryPrice <= lowPriceRange) return 1;
// within the range, the long position is entitled to the notional
value,
// which is equal to the following fraction of collateral
if(expiryPrice <= highPriceRange) return lowPriceRange /
expiryPrice; // above the range, the long position is entitled to a
fixed number of tokens return lowPriceRange / highPriceRange;
```

The `PositionSettled` event, contains a typographical error in the second argument's identifier. It is presently `colllateralReturned`, when it should be `collateralReturned`. The subtle error could lead to issues with off-chain systems parsing the event details. In `LongShortPairCreator.sol` there are no events emitted for the creation of the `longToken` or the `shortToken`, nor are those addresses part of the emitted `CreatedLongShortPair` event. Consider emitting events that contain the addresses of the new tokens to facilitate tracking. The following functions have incomplete or incorrect Natural Specification comments. Consider correcting them: – The `createLongShortPair` function of the `LongShortPairCreator` contract is missing its `@return` comment. – The `settle` function incorrectly lists `collateralReturned` as a parameter (in addition to the returned value). Some functions could benefit from renaming. These are our suggestions: – The `_getAddressWhitelist` function of the `LongShortPair` contract could be renamed to `_getCollateralWhitelist`. – The `computeExpiryTokensForCollateral` function of the `LongShortPairFinancialProductLibrary` contract could be renamed to `longPercentageCollateralAtExpiry` or similar. The `createLongShortPair` function of the `LongShortPairCreator` contract and the `create` function of the `LongShortPair` contract both transfer funds on behalf of the message sender. The contracts implicitly assume that they have been granted an appropriate allowance. Consider documenting this in the function comments. There are some misleading comments in the code base: – The parameter setters in the library functions, for example the `setLongShortPairParameters` function in `BinaryOptionLongShortPairFinancialProductLibrary.sol`, refer to the non-existent `financialProduct` parameter instead of `LongShortPair`. – The code base is inconsistent about using `1e18` to represent `100%`. For example, the `expiryPercentLong` comment switches between `1e18` and `1` in the same comment. This instead of `tokensToRedeem`. This comment would also be clearer if it started with "Redeem" instead of "Return". – In the `Testable` contract there is an inline comment that says "If the contract is being run on the test network, then `timerAddress` will be the 0x0 address." However, it appears that this comment is erroneous. The `onlyIfTest` modifier requires that `timerAddress != address(0x0)`. There are also other comments in the codebase that agree with the modifier: "_timerAddress … Set to 0x0 in production.". The comment in `Testable` should be modified to agree with the implementation to avoid potential confusion around such a critically important value. – In `Timer.sol` the `@notice` docstring for the `getCurrentTime` function is misleading. It states, "Otherwise, it will return the block timestamp." However, the function is only capable of returning the stored value of `currentTime` and has no logic to conditionally return the block timestamp. There are some unused imports in the code base: – `SafeMath.sol` in `LongShortPair.sol` – `IERC20Standard.sol` in `LongShortPair.sol` – `ContractCreator.sol` in `LongShortPairCreator.sol` – `AddressWhitelist.sol` in `LongShortPairCreator.sol` – `SyntheticToken.sol` in `LongShortPairCreator.sol` Lastly, there are some typographical errors in the code base: – Line 55 of `LongShortPairCreator.sol` says "appended" instead of "prepended". – Line 57 of `LongShortPairCreator.sol` repeats the word "as". – Line 106 of `LongShortPair.sol` says "Requires mint and burn needed by this contract", which is ungrammatical. – Line 19 of `LinearLongShortPairFinancialProductLibrary.sol` has the extra word "be". – Lines 35 and 41 of `LinearLongShortPairFinancialProductLibrary.sol` have the extra word "price". – Line 42 of `LinearLongShortPairFinancialProductLibrary.sol` has the extra word "a". – Line 63 of `LinearLongShortPairFinancialProductLibrary.sol` says "are" instead of "is". – Line 26 of `Lockable.sol` reads "… and make it call a" when it should read "… and **making** it call a". **Update:** *These issues were addressed in Pull Requests 3130, 3131, 3133 and 3152. Our review of those PRs include additional comments.* ### Pull Request 2949 This PR generalizes the `KpiOptionsFinancialProductLibrary` to allow multiple financial contracts to use the library with different individual transformed prices. Previously, the library targeted a particular use case where the pre-expiration price was set to 2. Since any address can call the new `setFinancialProductTransformedPrice` function, we'd like to re-iterate our recommendation from our review of 2926: >It should be noted that this introduces a front-contract address and invalidate the contract before deployment. Consider configuring the transformation, where applicable, in the `ExpiringMultiPartyLib` contract during deployment. Additionally, consider restricting the [`setFinancialProductTransformedPrice`] function's access control to the `ExpiringMultiPartyCreator` or the financial product itself to prevent someone pre-configuring the transformation at the deployment address.

In addition, there appears to be a missing validation in this new function. The function comments claim the price can't be set to blank, which presumably means it can't be set to zero. However, the function does not enforce this requirement.

Consider emitting an event when the transformed price is set to facilitate tracking.

There are a couple of misleading comments:

- the contract comment still claims that the price should always be set to 2, but that is no longer required.
- the `transformCollateralRequirement` function comment claims the requirement is equivalent to 1 token pre-expiry. This is accurate but confusing because the function does not consider the current time or the expiration time. It may be clearer if the comment explained why the function is only relevant pre-expiry.

Lastly, we'd like to note that the library does not necessarily ensure that the financial product will remain collateralized. In the original use case, this was implicitly guaranteed by a price identifier that would never return an oracle price higher than 2. Since the library will now be used more broadly, considering explicitly documenting the corresponding safety requirement in the contract comments.

Pull Request 2969

This PR extends the logic from PR 2903 to send governance actions from the main chain to another EVM-compatible "sink" chain.

`SinkGovernor`.
  - The `relayGovernance` function comment says `sinkChainID` instead of `destinationChainId`.

Additionally, as noted in the pull request, none of the new functions follow the Ethereum Natural Specification Format (NatSpec) for their arguments. Consider including them.

Lastly, instead of passing the chain ID to the `SourceGovernor` constructor, consider using the assembly `chainid` instruction, to avoid the possibility of mismatches. Note that this suggestion is also applicable to the `BeaconOracle` and its descendant contracts.

Update: *All issues were addressed in Pull Request 3189.*

Pull Request 3037

This PR addresses the comments in our review of PRs 2903 and 3013. However, instead of passing the `requester` and `pusher` values over the bridge, those parameters are removed from the `PriceRequestAdded` and `PushedPrice` events on both sides.

We still have some minor comments:
– `executePublishPrice` is missing the `@param` tag for the `price` parameter.
– the `_formatMetaData` function comments on the `SinkOracle` and the `SourceOracle` correctly state that the length field is 32 bytes, but they still incorrectly imply the data starts 64 bytes into the buffer. This mistake originates on Chainbridge's `GenericHandler` contract.

Update: *All issues were addressed in Pull Request 3064.*

Pull Requests 2903 and 3013

These PRs use Chainbridge to send messages between different EVM-compatible chains. In particular, DVM price requests are passed from another chain to the Ethereum mainnet, where the

On both chains, the `PriceRequestAdded` event includes the `msg.sender` as the `requester`. However, in the `SourceOracle`, this will be the `GenericHandler` contract, not the address that requested the price. Similarly, the `SinkOracle` will emit the `GenericHandler` as the `pusher` in the `PushedPrice` event. In both cases, consider passing the original sender over the bridge so it can be emitted on the destination chain.

Additionally, the following comments are misleading:

- The `_requestPrice` function comment says the function will revert if it's already been requested. In fact, it returns without doing anything.
- The `_formatMetadata` function comments on the `SinkOracle` and the `SourceOracle` claim the `uint256` length field takes up 64 bytes instead of 32.

The `publishPrice` function of the `SourceOracle` is missing its `@param` comments. Consider adding them.

Lastly, the `BeaconOracle` contract has the following typographical errors:

- Line 11 says "respectivly".
- Line 27 excludes `chainId` from the list of encoded values
- Line 68 also excludes `chainId` and refers to the whole collection of values as a "pair"

Update: *These issues were addressed in Pull Request 3037. Our review of the PR includes additional comments.*

Pull Request 2926

This PR modifies the `PreExpirationIdentifierTransformationFinancialProductLibrary` to remove the administrator. Instead, anyone can use this library to set a pre-expiration price identifier transformation for their financial product.

known in advance, it may be possible to predict the eventual contract address and invalidate the contract before deployment. Consider configuring the transformation, where applicable, in the `ExpiringMultiPartyLib` contract during deployment. Additionally, consider restricting the `setFinancialProductTransformedIdentifier` function's access control to the `ExpiringMultiPartyCreator` or the financial product itself to prevent someone pre-configuring the transformation at the deployment address.

The PR also updates the comments to account for the new behavior. While references to the contract owner have been removed, the comments still have two outdated references (here and here) to the library deployer.

Lastly, the PR introduces the `PostExpirationIdentifierTransformationFinancialProductLibrary`, which is equivalent to the `PreExpirationIdentifierTransformationFinancialProductLibrary` except the transformed price applies after expiration. We don't have any comments.

Update: *The UMA team have acknowledged the possibility of misconfigurations, noting that is a natural consequence of configurable contracts and user interfaces should validate the configuration.*

Pull Request 2828

This PR addressed the typographical errors and misleading comments that we noted in our review of 2668.

Pull Request 2768

The PR introduces the `KPIOptionsFinancialProductLibrary` contract, which allows `ExpiringMultiParty` contracts to enforce a 2:1 collateral requirement before expiration. The details of the payout structure at expiration are deferred to the price identifier, which should be specified to ensure the price is capped at two collateral tokens per synthetic token.

> We believe that invoking the full oracle mechanism to achieve consensus on an unused parameter is unnecessarily wasteful. If modifying the contracts to bypass the oracle is infeasible, consider transforming the price identifier instead, so the oracle would be expected to return [a constant]. At the very least, this simplifies the data retrieval and validation requirements. It also enables future upgrades where the oracle can reuse constant results across different timestamps or simply return the constant if it is known ahead of time.

Update: *The UMA team have acknowledged this and intend to address it in a pending upgrade to the `ExpiringMultiParty` contract.*

Pull Request 2744

This PR implements all of our recommendations from Pull Requests 2598 and 2672. We don't have any comments.

Pull Requests 2598 and 2672

These PRs introduce and modify the `MerkleDistributor` contract. For simplicity, we reviewed the final result.

The contract has an owner that can easily distribute tokens to a large number of addresses by combining all transfer descriptions in a Merkle tree, publishing the root, and transferring the total number of funds to the contract. Subsequently, the tokens can be retrieved by providing a Merkle proof of an unclaimed transfer. The Merkle trees are not validated in any way, so the system assumes the contract owner behaves honestly.

The distributor allows anyone to prove and execute transfers on anyone elses behalf. A `claimMulti` function is provided to allow for multiple transfers to be claimed in a single transaction.

Since anyone can execute transfers on behalf of anyone else, it is possible for an attacker to "grief" a claimer by front-running their claim transaction with a transaction that performs the same transfer first. This will cause the honest user's transaction to revert in the `_verifyAndMarkClaimed` function when it checks whether the claim has already occurred. This is not a major concern for single claims via the `claim` function, because the honest user ends up spending less gas than

However, this attack can be more problematic for the `claimMulti` function. The attacker can front run the call to `claimMulti` with a call to `claim` that performs only the *last* claim that the `claimMulti` call will attempt. This results in the entire `claimMulti` transaction reverting. In this way, an attacker can prevent an entire block of claims by making only a single claim. They may be motivated to do this, for example, to prevent large batches of new claims from being sold on the open market shortly after a new claim window opens.

If this is a concern, consider submitting all `claimMulti` transactions via a private transactions channel, such as SparkPool's Taichi private endpoint.

Moreover, we have several recommendations to improve the quality of the code base:

- The `lastCreatedIndex` variable name and comments suggest it indicates an existing window. In fact, it is the index that will be assigned to the next window. Consider updating the name and comments accordingly.
- The `WithdrawRewards` event does not specify the reward currency. Consider including it as well.
- The `setWindow` function performs the token transfer before emitting the event. If the reward token follows the ERC777 specification, it is possible that the caller has a `tokensToSend` hook that re-enters the `setWindow` function, which would cause the `CreatedWindow` events to be emitted in the wrong order. This may fall outside the threat model, since the function can only be called by the owner, who is assumed to behave honestly. Nevertheless, as a matter of good practice, consider emitting the event before performing the token transfer or introducing a reentrancy guard.
- The code does not use Ethereum Natural Specification Format (NatSpec) comments. Consider following this specification on everything that is part of the contract's public API.
- The `setWindow` function implicitly assumes the caller has granted an appropriate allowance to the contract. Consider stating this explicitly in the function comments.
- The `setWindow` function comment has an extra closing bracket.
- The `withdrawRewards` function comment is missing the space in the phrase "in case".

Update: *All recommendations were implemented in Pull Request 2744.*

`ExpiringMultiParty` contracts to mimic the payout structure of covered call options.

Before the option expires, the `transformPrice` function discards the oracle price and returns 1. We believe that invoking the full oracle mechanism to achieve consensus on an unused parameter is unnecessarily wasteful. If modifying the contracts to bypass the oracle is infeasible, consider transforming the price identifier instead, so the oracle would be expected to return 1. At the very least, this simplifies the data retrieval and validation requirements. It also enables future upgrades where the oracle can reuse constant results across different timestamps or simply return the constant if it is known ahead of time.

We would also like to note some typographical errors:

- Line 77 says "an collateral token" instead of "a collateral token"
- Line 86 says "cover call" instead of "covered call"

Lastly, while investigating this pull request, we identified a misleading comment in Line 454 of the `Liquidatable` contract that states the sponsor and disputer reward percentages cannot (cumulatively) exceed zero but the actual restriction is that they cannot exceed 1.

Update: *The UMA team have acknowledged the suboptimal* `transformPrice` *function and intend to address it in a pending upgrade to the* `ExpiringMultiParty` *contract. The typographical errors and misleading comment are fixed in Pull Request 2828.*

Pull Request 2600

This PR removes restrictions in the `DesignatedVotingFactory` contract. It is now possible for voters to change the `DesignatedVoting` contract that they are associated with. We don't have any comments.

Pull Requests 2395 and 2546

These PRs add additional parameters to the `ProposePrice` and `DisputePrice` events in the `OptimisticOracle`. They also remove unnecessary "//Event" comments preceding some event emissions. We don't have any comments.

OpenZeppelin

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

OpenZeppelin