# UMA Audit – L2 Bridges

OPENZEPPELIN SECURITY | DECEMBER 1, 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, the Perpetual Multiparty template and various incremental pull requests over a longer engagement. In this audit we reviewed a new mechanism to quickly send tokens from a layer 2 chain to the Ethereum mainnet, and related changes to the Optimistic Oracle. The review was completed by 2 auditors over 3 weeks.

## Scope

The audited commit is `f24ad501c8e813cf685f72217e7f13c8f3c366df` and the scope includes the following contracts:

- contracts/insured-bridge/* (excluding test contracts)
- contracts-ovm/insured-bridge/implementation/*
- contracts/common/implementation/AncillaryData.sol
- contracts/oracle/implementation/SkinnyOptimisticOracle.sol

We also reviewed the changes to solidity files in Pull Request 3445.

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

## System overview

contract, the Deposit Box, knowing the tokens will eventually be transferred (as a batch) to an L1 UMA contract, the Bridge Pool. There is a separate Bridge Pool for each token to be transferred.

Following an L2 deposit, anyone can relay the details to the L1 Bridge Pool, which waits a short period in case someone wants to dispute the relayed information. All disputes are handled by the Skinny Optimistic Oracle (described below). Before accepting the relay, liquidity providers must pre-fund the Bridge Pool contract in exchange for LP tokens. Undisputed relays are assumed valid, and the Bridge Pool completes the transfer using its own reserves, where a fraction of the transfer is diverted to the relayer and a fraction is retained as liquidity fees. The funds will eventually be replenished when the L2 deposit is finalized, and the liquidity fees are assigned to the LP token holders.

The Bridge Pool also allows anyone to individually fund a transfer (without the Bridge Pool reserves) before the dispute period expires, in exchange for a fraction of the transfer amount. Since the relay can still be disputed, these funds would be lost if the relay was deemed to be incorrect. It is expected that in most cases, this mechanism will allow users to experience immediate L2-to-L1 token transfers.

The Skinny Optimistic Oracle is conceptually very similar to the existing Optimistic Oracle. It provides an incentive mechanism for users to simply assert the result of an oracle request, which is assumed to be accurate if it is not disputed. Disputes are relegated to the slower DVM mechanism described in our previous audit reports. The main difference is that the new version requires users to provide all relevant information when executing function calls, so the values do not need to be saved or retrieved from storage. It also removes the ability for requesters to change configuration parameters in active requests.

We have previously reviewed the Long-Short-Pair contract, which provides a generic mechanism to create various financial instruments. These contracts can be resolved when their expiration time is reached and the settlement price is known. The changes introduced by Pull Request 3445 introduce the possibility of resolving the contracts early if the settlement price is known before the expiration time.

## Privileged Roles

Bridge Pool. These parameters are set by an administrator contract on L1, which also parameterizes the dispute resolution process of the bridge pools. The contract is expected to be controlled by the UMA governance mechanism, so users must trust this process to manage the system sensibly and fairly.

## Ecosystem dependencies

All reviewed components use time-based logic, which means they depend on Ethereum availability. In particular, if dispute transactions are delayed significantly then invalid relays or price proposals may be incorrectly confirmed.

Additionally, the token bridge implicitly assumes that all funds sent to L2 deposit boxes will eventually be transferred to the corresponding L1 bridge pool. This relies on the correct and continued functioning of the Optimism and Arbitrum bridges, and their dispute resolution mechanisms.

Lastly, tokens sent to the L2 deposit box are assigned to the bridge pool in L1, not the intended recipient. To retrieve the funds from the pool, L1 token holders must first match them with additional tokens. Therefore, the mechanism relies on a sufficiently deep market of L1 tokens to ensure there is always liquidity.

## Client-reported issues

During the audit, the UMA team independently identified a number of issues and behaviors worth highlighting:

> If the Optimistic Oracle or Bridge Admin parameters change during a relay's challenge period, then disputing the relay deletes the relay with no additional recourse for either the proposer or the disputer. For example, imagine that the relay is sent for the collateral token `TOKEN_A`, but in the middle of the relay `TOKEN_A` is removed from the collateral whitelist. A dispute will now revert since you cannot submit any price requests to the OO or DVM for unwhitelisted collateral. Since we don't want to block valid dispute requests, the `BridgePool` will delete the pending relay for `TOKEN_A` in the case of a dispute. The consequences of this design decision are:
> 1. An increase in the final fee will cause: Any outstanding relay on that token to be "cancellable"

> griefer to spend gas to cancel relays and force them to be re-relayed.
> 2. A de-whitelisting of the identifier or token, which shouldn't happen unless something goes very wrong would cause:
> 3. An extended period of undisputable requests where any request can be canceled and there exist no economic incentives for disputing. This seems better than the alternative of blocking disputes completely, but it's admittedly quite bad, as any griefer can block relays indefinitely by paying gas or send bad relays without punishment (other than gas fees).
>
> Note: this is an alternative to having the OO that "freeze" parameters such as the final fee or collateral whitelist for some time, but this would require additional calls to the OO, which would be costly for the happy path.

> Relays can be sped up via `speedUpRelay()` after they pass liveness. While we don't see any risk with this, it does open up the possibility for flash loans + speed ups + settle after liveness, giving the flash borrower the instant relay fee for "free". We prevent this in this proposed PR.

> On `settle`, if the `BridgePool` is a `WETH` pool and the recipient is a contract that is not `payable` (cannot accept ETH), then `settle` will fail. We plan to fix this and fallback on sending `WETH`, but there is no outstanding work done yet on this.

> In `relayDeposit`, we check that the `BridgePool`'s balance is greater than the amount to relay PLUS the proposer bond. This is an outdated check and too conservative since the proposer bond is pulled from the user after the check. We address this in this proposed PR.

> I just caught a bug where `chainId` in `BridgePool`, included as part of the `Deposit` struct and as a function input to all of the relay-related functions (i.e. `relayDeposit`, `speedUpRelay`, `settle`) is type `uint8`. This is too small of a type to handle Arbitrum for example whose ID is 421611. We actually caught this bug and fixed it on the L2 side: `BridgeDeposit` has set its `chainId` type to `uint256`. This PR will make the `chainId` on `BridgePool` match the type on `BridgeDepositBox`: UMAprotocol/protocol#3463

> up the relay amount in the pool. It cannot be withdrawn by LPs or used by future relays. Fix is here: UMAprotocol/protocol#3473.

> We found a bug in `BridgeDepositBox` where `hasEnoughTimeElapsedToBridge` does not check if a `uint256` value is equal to `0` by default: Fixed in PR 3484

> The exchange rate method (which is state modifying) is called between tokens being transferred in and LP tokens being minted in the addLiquidity method of the bridge pool contract. This computation needs to be moved up to the top of the method. This causes very strange state values. See PR here to fix.

> The view method `liquidityUtilizationPostRelay` (which is only used offchain), reports an incorrect utilization number. The denominator on this line shouldn't be just `liquidReserves`, it should instead be a representation of unutilized and utilized reserves. Fixed here.

### Update

In addition to the issue fixes, we also reviewed the following incremental changes:

- PR3500 removes the redundant token parameter from `BridgePool` events.
- PR3478 adds the DVM final fee to the list of locally cached variables.
- PR3460 accounts for a possible negative liquidity utilization edge case (in addition to addressing N04)
- PR3482 removes redundant files and updates OVM constants in accordance with OVM 2.0 changes.
- PR3585 updates the `BridgeDepositBox` interface for consistency and uses the OpenZeppelin `SafeERC20` library.

While reviewing the fixes we identified another issue. When determining the value of `BridgePool` LP tokens, there is an intermediate calculation that could unexpectedly negative overflow, which would temporarily disable adding and removing liquidity. The calculation should be reordered to add the utilized reserves before subtracting the undistributed fees.

The `LongShortPair` contract retrieves a proposer reward from whichever address triggers the expiration, which is used to incentivize price proposals in the Optimistic Oracle. However, the `LongShortPairCreator` contract also retrieves and forwards the funds from the deployer address. These additional funds are not passed to the Optimistic Oracle, and instead remain trapped within the `LongShortPair` contract.

Consider removing the duplicate transfer.

**Update:** *Fixed as of commit `9bab1ff353a417952ba8c96a098773f340d9da17` in PR3523.*

## High severity

### [H01] Concurrent relays deplete reserves

The `relayDeposit` function of the `BridgePool` contract ensures the contract has sufficient funds to execute the transfer. However, it does not account for the pending reserves, which tracks funds that are earmarked for active relays. Therefore, multiple simultaneous relays may rely on the same funds, and they may not all be settleable immediately. In particular, with a steady stream of transfers, instant relayer returns may be delayed indefinitely.

Consider preventing relays that would cause the pending reserves to exceed the liquid reserves.

**Update:** *Fixed as of commit `6290f3facbca8d878605a1d390ed59d4b6b6db02` in PR3501.*

### [H02] Bridging parameter bounds don’t match

The `deposit` function of the `BridgeDepositBox` contract, deployed on layer 2 chains, is used to bridge funds between the L2 and L1. In particular, relayers are incentivized to relay the transaction details on the associated L1 `BridgePool`. However, the deposit box uses inclusive bounds to restrict the relay fees, while the bridge pool uses exclusive bounds. This means that some deposits (with 25% relay fees) cannot be relayed, and the funds will be inaccessible on both layers.

*This was originally classified as Critical severity but was downgraded when the UMA team pointed out the funds would not be strictly trapped and could be released if the DVM voters agreed to accept a modified relay description for affected deposits.*

## Medium severity

### [M01] Callbacks to wrong address

The `SkinnyOptimisticOracle` invokes callback functions on the price requester, if they exist, so the requester can respond to significant state changes. However, the callback is incorrectly invoked on the price proposer instead of the price requester in the `proposePriceFor` function. This means the price requester is unable to respond to price proposals.

Fortunately, this feature is not used in the current code base. Nevertheless, consider invoking the `priceProposed` callback on the requester.

**Update:** *Fixed at commit `7bd3faeb6f3706132f77b9ba2dce192d1a151e74` in PR3531.*

### [M02] Faulty append function

The `appendKeyValueBytes32` function should combine its inputs into a formatted `bytes` array. However, the `currentAncillaryData` is incorrectly discarded.

Since the ancillary data influences the oracle resolution process, an incorrect value could undermine the oracle results. Fortunately, there is only one call to `appendKeyValueBytes32` in the codebase, and it uses an empty `currentAncillaryData` buffer, so the bug does not affect this case.

Consider updating the `appendKeyValueBytes32` function so that the `currentAncillaryData` is included in the returned bytes array.

**Update:** *Fixed in commit `5609433c154f47e8ee9c52f9b6d7c787fbe3e455` of PR3532.*

### [M03] Incomplete ancillary data validation

may unexpectedly reject early expiration price requests, which would disable this feature.

Consider updating the validation to account for the additional field.

**Update:** *Fixed as of commit `4a56e66492f40e20254cebb145c2d91304f7cb43` in PR3524.*

## [M04] Mishandled zero timestamp

In the `LongShortPair` contract, a zero early expiration timestamp is used as a flag to indicate that nobody has triggered the early expiration mechanism. However, it is possible to trigger that mechanism with a zero timestamp. In this scenario, the Optimistic Oracle will be be invoked but the protection against subsequent price requests will not be effective. Fortunately, once the settlement price is chosen, it won't be overriden, so this won't lead to inconsistent settlements. Nevertheless, a subsequent price request could change the recorded early expiration timestamp, even if the zero timestamp is used to determine the settlement price. It could also emit a misleading event.

Consider preventing early expiration using the zero timestamp.

**Update:** *Fixed as of commit `11d287c07c93c04f534b2ef3c869966d9f18ac60` in PR3526.*

## [M05] Possible zero bond

The `requestPrice` function of the `SkinnyOptimisticOracle` contract uses the final fee as the bond if the bond is not specified. However, the `requestAndProposePriceFor` function may use a zero bond, which contradicts its `@notice` and `@param` comments. A zero bond weakens the incentive against invalid proposal or disputes.

Fortunately, the only call to this function in the code base sets a proposer bond. Nevertheless, consider using the final fee if the bond is not specified.

**Update:** *Fixed as of commit `daaabfc342ba1395a577159b6eb26adb20fcd232` in PR3534.*

## [M06] Unnecessary administrator privileges

provides administrator privileges to the contract deployer, which allows them to mint and burn L tokens. However, this power is not required and, if exercised, could unfairly penalize liquidity providers.

Consider modifying `BridgePool` to inherit directly from `ERC20` instead of `ExpandedERC20`.

**Update:** *Fixed in commit `370e8b21b660543eadbd764fed984a5bdeddce24` in PR3492.*

## Low severity

### [L01] Cannot settle at expiration

The `settle` function of the `LongShortPair` contract considers settlement conditions when the current time is strictly before or after the expiration timestamp. However, it incorrectly reverts when the current time matches the expiration timestamp.

Consider using an inclusive bound to match the `postExpiration` modifier.

**Update:** *Fixed in commit `f03cdaa50b16d29e8f42f000bf7cd50a042cf616` in PR3527.*

### [L02] Missing error message in require statement

There is a require statement in the `BridgePool` contract without an error message.

Consider including specific and informative error messages in all require statements.

**Update:** *Fixed as of commit `67e60faa3a44c842c37211d2e903a983ff192e57` in PR3536.*

### [L03] Missing docstrings

There are some instances throughout the codebase where the Ethereum Natural Specification is missing or incomplete. Examples include:

- the `BridgeDepositBox`, `AVM_BridgeDepositBox` and `OVM_BridgeDepositBox` constructors are missing a `@param` comment for the

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API.

**Update:** *The highlighted comments were fixed in commit `e943e85a7dae60acd17a6d6aa027fbb1017c95ee` of PR3533. We did not validate NatSpec completeness in the rest of the code base.*

## Notes & Additional Information

### [N01] Call return value not checked

In the `deposit` function of the L2 `BridgeDepositBox` contract there is a low-level call to the `l2Token` when the `l1Token` is `l1Weth`. This low-level call is to the `deposit()` function, which belongs to the WETH interface. If this `l2Token` behaves exactly like WETH it should never fail. But in the case the `l2Token` behaves differently and does fail, there would be no revert since the success flag of this low-level call is never checked.

Consider checking and reacting appropriately to the return values of all low-level calls.

### [N02] Lack of indexed parameters in events

Many of the events defined in this codebase have parameters that should be indexed:

- `newAdmin` in `BridgePoolsAdminTransferred`
- `chainId` in `WhitelistToken`
- `l2Token` in `TokensBridged`
- `newAdmin` in `SetXDomainAdmin`

Consider indexing event parameters to avoid hindering the task of off-chain services searching and filtering for specific events.

**Update:** *Partially fixed in commit `d156b40b2ddb109806336c4d169dbdea91ed1c3e` of PR3535. The `chainId` parameter of `WhitelistToken` was not updated.*

### [N03] Implicit casting inconsistency

`[illegible]` parameter of the `[illegible]` function is prematurely cast to a `uint256`. This has no functional consequences.

Nevertheless, in the interest of consistency, consider using a `uint64` for this parameter and allowing it to be implicitly cast to a `uint256` when passed to the Optimistic Oracle.

**Update:** *Fixed in commit `1c3c5c000ef450f5e2da056e41caff468c3fcdcb` of PR3528. The timestamp is now explicitly cast.*

## [N04] Incorrect type

The `sendMessage` function of the `iOptimism_CrossDomainMessenger` interface uses a `uint256` gas limit while Optimism's `OVM_CrossDomainEnabled` uses a `uint32` gas limit.

For consistency and predictability, consider updating the `iOptimisim_CrossDomainMessenger` `sendMessage` function to use a `uint32` gas limit.

**Update:** *Fixed as of commit `381951aad988bbba6b2ef1b136ed5c48df50aa88` in PR3460.*

## [N05] Lack of validation

All functions in `BridgeAdmin` that call `_relayMessage` assume the transaction value matches the `l1CallValue` parameter, but this is not enforced.

Consider ensuring the correct `msg.value` is set.

**Update:** *Fixed as of commit `f19b8d04c2343051ff2a8145abd41c39bd025063` in PR3537.*

## [N06] Readability

The `_getDepositHash` function of the `BridgePool` contract unrolls the `depositData` struct to interstice the `l1Token` as an argument in the composition of `keccak256` with the

Consider simplifying the arguments to simply be the ordered pair `depositData` and `l1Token`.

**Update:** *Fixed as of commit `31754be4a818109fa12131f854c3f70d6c72dba7` in PR3538.*

## [N07] Reentrant function

The `requestAndProposePriceFor` function of the `SkinnyOptimisticOracle` contract makes a call to an untrusted `msg.sender` but is not guarded by a `nonReentrant` modifier. While, in this instance, this does not seem to be a security concern, this can introduce unexpected behavior.

Consider adding the `nonReentrant` modifier to all functions which make calls to possibly untrusted contracts.

**Update:** *Fixed in commit `b744d24e7579b7afa2c778f4dd680f26117b3990` of PR3539.*

## [N08] `seqNum` not logged

The `relayMessage` function of the `Arbitrum_Messenger` contract does not emit a relevant event after executing a sensitive action. The `relayMessage` function calls as a subroutine `sentTxToL2NoAliasing` which itself returns the `uint256` value `seqNum`, but this return value is not logged in the `relayMessage` function.

Consider emitting events after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract’s activity.

**Update:** *Fixed as of commit `30343f33532a6c255dc4cc18c3b497d9b2767a7c` in PR3541.*

## [N09] Typographical errors

The codebase contains the following typos:

- “int” should be “uint”.

- “method” should be removed.
- “its” should be “it’s”.
- “Can not” should be “Cannot”.
- “expire” should be “expired”.
- “Disiputer” should be “Disputer”.
- “second” should be “seconds”.
- “a instant” should be “an instant”.
- “OptimismMessenger” should be “Optimism_Messenger”.
- “tokens” should be “token”.
- “callers” should be “caller’s”.
- “to” should be “from”.
- “Can not” should be “Cannot”.

Consider correcting these typos to improve code readability.

**Update:** *Fixed as of commit `2dccbe1c2c82fe2a21c179ac06c2d4f0d911a2ca` in PR3540.*

## [N10] Undocumented ERC20 approval requirement

The `requestEarlyExpiration` and `expire` functions of the `LongShortPair` contract each assume that the caller has granted the contract an allowance to pull the proposer reward.

For the sake of predictability, consider documenting this requirement in the function comments.

**Update:** *Fixed in commit `da3754f50284480df57b90b80002da06a1ce0d02` in PR3529.*

## [N11] Unused modifier

In the `BridgePool` contract, the `onlyFromOptimisticOracle` modifier is defined but is never used in the codebase and should therefore be removed.

**Update:** *Fixed in commit `7abece6377637e8c4cd3bd07ab9adcfa051d4e94` in PR3542.*

## Conclusions

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

OpenZeppelin