OpenZeppelin | security

Taiko Ontake Fork Audit

taiko

November 13, 2024

Table of Contents

Summary

Type	Rollup
Timeline	From 2024-09-30 To 2024-10-18
Languages	Solidity

Total Issues	19 (19 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	3 (3 resolved)
Low Severity Issues	7 (7 resolved)
Notes & Additional Information	9 (9 resolved)

Scope

We audited the taikoxyz/taiko-mono-oz-audit repository at commit fd1c039.

In scope were the following files. For files marked with *(diff)*, we only reviewed the changes made since the previous audit commit b47fc34.

```
contracts
├── layer1
│   ├── based
│   │   ├── LibBonds.sol
│   │   ├── LibData.sol
│   │   ├── LibProposing.sol (diff)
│   │   ├── LibProving.sol (diff)
│   │   ├── LibUtils.sol (diff)
│   │   ├── LibVerifying.sol (diff)
│   │   ├── TaikoData.sol (diff)
│   │   └── TaikoL1.sol (diff)
│   └── verifiers
│       └── compose
│           ├── ComposeVerifier.sol
│           ├── TeeAnyVerifier.sol
│           ├── ZkAndTeeVerifier.sol
│           └── ZkAnyVerifier.sol
└── layer2
    └── based
        ├── Lib1559Math.sol (diff)
        └── TaikoL2.sol (diff)
```

System Overview

The Taiko protocol has already been described in our previous audit report. This audit focussed on the features introduced by the Ontake fork.

The main changes introduced are aimed at supporting preconfirmations, where L1 proposers make off-chain commitments to the transactions they will include in their block. This is achieved by requiring L1 proposers to register and lock the stake before they are allowed to propose blocks. Their stake will be slashed if they fail to honor the preconfirmations, and affected users may be compensated. Since not all L1 proposers will register, the L1 contracts cannot advance every block. Instead, the next proposer in the schedule will be responsible for issuing off-chain preconfirmations throughout all intermediate blocks. The proposers can also delegate their preconfirmation ability to a dedicated preconfer service, which helps limit the frequency of transitions between preconfers.

The registry, preconfer, and schedule logic features were not reviewed within this audit. However, the main protocol was updated to support multi-block proposals as well as an ability to prove multiple blocks, either with multiple proofs or a single batch proof. The codebase also introduces a `ComposeVerifier` framework, which provides a flexible way to describe a proof as a combination of subproofs. For example, the `TeeAnyVerifier` will accept one of two different TEE proofs, the `ZkAnyVerifier` will accept one of two different ZK proofs, and the `ZkAndTeeVerifier` requires both a TEE and a ZK proof. All verifiers also support a single batch proof for multiple blocks.

Lastly, the L2 gas mechanism is now described using issuance per second, rather than per block. This means that it is not affected by the frequency of blocks or variance in block time. It is also possible to change the gas target on any anchor transaction.

Code Complexity

The codebase has become more complex because it covers the transition to the Ontake fork. There are several updates to the internal data structures, and the codebase converts or chooses between them, depending on whether the relevant block should use the old or new structure. Moreover, it also handles proving blocks that were proposed using a previous version of the codebase, where some fields were assigned differently. Once the transition is complete, a future update to reduce the branching would be recommended.

Security Model and Trust Assumptions

The main change to the security model relates to the preconfirmation registry and look-ahead schedule, which are still being designed and were not reviewed in this audit.

One minor update is that when `ComposeVerifier` contracts are in use, the owner of the `AddressManager` contract will set the relevant sub-verifier contracts. As with all the address resolutions, users rely on them being set correctly and safely. This includes ensuring that only trusted verifiers are registered and that there are no duplicates across any of the sub-verifiers or that the same proof could be used repeatedly.

Integration With the Protocol

One potentially unexpected behavior is that for simplicity, validity bonds are either taken entirely from previous TKO deposits or are transferred entirely on demand. This means that even if a deposit partially covers the required bond, users must have and approve sufficient tokens for the whole bond. Of course, they can also top up the deposit beforehand, if desired.

Medium Severity

M-01 Inconsistent Use of State Roots

When proving a block, the state root is only saved on "sync blocks" (currently configured to every 16th block). All other blocks treat the state root as zero. However, the state root is emitted as part of the event. If another prover attempts to dispute the state root (with the same block hash), the code will interpret them as agreeing with the original transition. This also means that verifiers may reject transitions based on an unused state root. Moreover, all getters that return a state root may unexpectedly return zero for non-sync blocks.

In the interest of maximum safety, we believe that verifiers should still consider the state root for all blocks. However, for predictability and code clarity, consider treating state roots for non-sync blocks as zero consistently throughout the codebase. This would involve:

- setting them to zero in the emitted events.
- removing them from the `sameTransition` calculation.
- documenting that they are intentionally set to zero in all the getters.

Update: *Resolved in pull request #18303.*

M-02 Batch Proofs Cannot Be Composed

The `verifyProof` function of the `ComposeVerifier` contract uses the `onlyAuthorizedCaller` modifier, which allows descendant contracts to introduce additional authorized callers. However, the `verifyBatchProof` function must be called by the `TaikoL1` contract. This means that the `verifyBatchProof` function of the `ZkAndTeeVerifier` is not authorized to invoke the sub-verifiers and is therefore non-functional.

Consider using the `onlyAuthorizedCaller` modifier in both of the aforementioned cases.

Update: *Resolved in pull request #18302.*

M-03 Cannot Send Tip for Batch Proposals

When proposing a block, the caller can send a tip to the L1 `coinbase` address as an incentive to include the transaction. However, the `proposeBlocks` function executes the same line multiple times. This means that the contract attempts to tip `msg.value` for every proposal. Since the extra funds would be taken from the `TaikoL1` contract, this could be used to steal any available ETH. Fortunately, the `TaikoL1` contract is not expected to hold ETH. Nevertheless, this means that if a tip is provided, the `proposeBlocks` function will revert when processing the second block.

Consider ensuring that the tip is only sent when processing the first block in a batch.

Update: *Resolved at commit ce76e66. The Taiko team stated:*

> *In the `ontake_cleanup2` branch, the tip payment code has been removed and `proposeBlock` (v1), the only payable function, is also removed.*

Low Severity

L-01 Incorrect Error

The `verifyBatchProof` function reverts with `CV_DUPLICATE_SUBPROOF` when the subproof specifies a zero verifier. This is incorrect because duplicate subproofs are prevented by clearing valid verifiers once they are used, which would cause duplicates to trigger the `CV_SUB_VERIFIER_NOT_FOUND` error.

Consider using the `CV_INVALID_SUB_VERIFIER` error for consistency with the equivalent check in the `verifyProof` function.

Update: *Resolved in pull request #18305.*

L-02 Incorrect Fork Check

The `proposeBlock` function is expected to be disabled after the Ontake fork. This check does not work correctly because the `metaV1_` return parameter is zero after the fork, so the condition actually requires 0 to be less than the `ontakeForkHeight`. This means that `proposeBlock` will behave like the `proposeBlockV2` function, except it will return an

empty `meta_` parameter. In practice, if the `_params` buffer is encoded using the original layout, then it would fail to decode correctly.

Consider using the `BlockMetadataV2` return parameter to retrieve the actual block ID.

Update: *Resolved in pull request #18227*

L-03 Lack of Event Emission

Events are a crucial mechanism for tracking and monitoring contract activity. Throughout the codebase, instances of events not being emitted after sensitive actions were identified. For example, other than the standard token transfer events, the `TaikoL1` contract does not emit any specific events when depositing and withdrawing bonds.

Consider emitting events after sensitive changes occur. This would help improve contract monitoring and allow for early identification and resolution of potential issues.

Update: *Resolved in pull request #18305 and pull request #18443.*

L-04 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified:

- None of the parameters in the `verifyBlocks` function are documented.
- The `_user` parameter in the `bondBalanceOf` function is not documented.
- The `_config` parameter in the `init` function is not documented.
- The `verifiedAt_` return value in the `getBlockInfo` function is not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #18320.*

L-05 Misleading Inline Documentation

Throughout the codebase, multiple instances of potentially misleading inline documentation were identified:

- The `proveBlock` `_input` parameter comment describes the pre-fork structure. After the fork, it should start with a `BlockMetadataV2` object.
- The first `getTransition` function comment incorrectly mentions the `parentHash`.
- The `chain_pauser` comment should refer to the chain watchdog.
- An inline comment within the `_proveBlock` function claims that a new transition will have a transition ID of zero but it will be set to the block's `nextTransitionId`, which is at least 1.
- The `TransitionState` comment claims that slot 6 occupies 90 bits, but it uses 88.
- The `shouldVerifyBlocks` code comment states that verification will be performed on blocks 3 and 7 for a segment of size 8, but it will be on blocks 0 and 4.

Clear inline documentation is fundamental for outlining the intention behind a piece of code. Mismatches between the inline documentation and the implementation can lead to serious misconceptions about how the system is expected to behave. Consider clarifying the referenced inline documentation to avoid confusion for developers, users, and auditors alike.

Update: *Resolved in pull request #18320.*

L-06 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `Lib1559Math.sol`, the `MAX_EXP_INPUT` state variable
- In `TaikoL1.sol`, the `init2` function
- In `TaikoL1.sol`, the `proposeBlockV2` function
- In `TaikoL2.sol`, the `parentTimestamp` state variable
- In `TaikoL2.sol`, the `parentGasTarget` state variable
- In `TaikoL2.sol`, the `anchorV2` function
- In `TaikoL2.sol`, the `ontakeForkHeight` function

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #18320.*

L-07 Inconsistency Between `lastUnpausedAt` Variables

The `EssentialContract` contract, inherited by several of the contracts in the protocol such as the `TaikoL1` contract, declares a global `lastUnpausedAt` variable. This variable is used to update the timestamp when the contract has been unpaused. The `TaikoL1` contract implements a similar variable inside the `state.slotB` struct which is also updated with the same unpausing action.

However, this latest variable is also updated when unpausing the proving mechanism, while the one from the `EssentialContract` contract is not. This means that both variables will go out of sync under such an action. This could mislead anyone attempting to calculate when the proving window is, because they might get different outcomes depending on which `lastUnpausedAt` variable they checked. Moreover, it is worth noting that the `EssentialContract` version remains unused within the codebase.

In order to improve the readability of the code, remove inconsistencies, and prevent erroneous calculations, consider unifying them under a single variable, updating both in all corresponding cases or renaming one to make it clear they have different functions.

Update: *Resolved in pull request #18276.*

Notes & Additional Information

N-01 Leftover Code In Comments

Leftover code present in comments can negatively impact code clarity and maintainability. In line 443 of `LibProving.sol`, there is a comment containing unused code.

Consider removing the aforementioned instance of unused code or documenting its function.

Update: *Resolved in pull request #18305.*

N-02 Unused Errors

Throughout the codebase, multiple instances of unused errors were identified:

- Within the `LibProposing` library, the `L1_LIVENESS_BOND_NOT_RECEIVED` error
- Within the `LibVerifying` library, the `L1_INVALID_CONFIG` and `L1_TOO_LATE` errors
- Within the `LibUtils` library, the `L1_BLOCK_MISMATCH` error
- Within the `TaikoL1` contract, the `L1_INVALID_PARAMS` error

Consider either using or removing the aforementioned instances of unused errors.

Update: *Resolved in pull request #18305.*

N-03 Use of Magic Numbers

Throughout the codebase, multiple instances of unexplained magic numbers being used were identified.

- In the `LibProposing` library, the number 12
- In the `LibUtils` library, the factor 60

To improve code clarity and readability, consider using named constants instead of magic numbers.

Update: *Resolved in pull request #18305.*

N-04 Incomplete Genesis Block

The genesis block defaults to a zero `proposedIn` field. This makes sense after the fork, since there is no corresponding L1 anchor block. However, if it is set before the fork, it should be the L1 proposal block number.

For consistency with the `proposedAt` field, consider changing the value of the `proposedIn` field to be the current L1 block.

Update: *Resolved in pull request #18312.*

N-05 Unusable ETH Withdrawal Functionality

The `TaikoL2` contract contains a mechanism for a privileged address to withdraw ETH. However, there is no straightforward mechanism to send ETH to this contract.

Consider documenting the aforementioned fact in the function comments.

Update: *Resolved in pull request #18305. The Taiko team stated:*

> *We added a comment in this PR. The TaikoL2 address is also used as L2 base fee recipient, that's why it will receive ETH.*

N-06 Incomplete Resizing

The `transitionId` indices have been resized to `uint24`. However, the `transitions` mapping still uses `uint32`.

Consider resizing the `transitions` mapping as well.

Update: *Resolved in pull request #18305.*

N-07 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified:

- In two instances (1 and 2), "an" should be "will".
- This comment misspells "executed", "deterministically", and "referring".
- In this comment, "fist" should be "first"

To improve code readability, consider correcting any typographical errors in the codebase.

Update: *Resolved in pull request #18305.*

N-08 Naming Suggestions

Inconsistent naming can lead to confusion in understanding the code.

For consistency with the `blockParamsV1ToV2` function and standard camel-case, considering capitalizing the word "to" in the other conversion functions.

Consider applying the above renaming suggestion to improve code clarity and readability.

Update: *Resolved in pull request #18305.*

N-09 Code Simplifications

Throughout the codebase, multiple opportunities for code simplification were identified:

- In the `_proposeBlock` function, the `local.params` object has a zero `anchorBlockId` and `timestamp` for pre-fork blocks. This means that the first part of the condition, the `postFork` flag, is redundant when setting default values.
- This `unchecked` block does not wrap any math operations and can be removed.
- Since `local.params.timestamp` is set to the L1 block timestamp before the fork, it covers both conditions when setting the block's `proposedAt` field.
- This conditional statement is unnecessary because the computation returns zero when `_amount` is zero.
- This `else` block is redundant because the `if` case breaks out of the loop.

To improve code maintainability and clarity, consider implementing the aforementioned simplification suggestions.

Update: *Resolved in pull request #18308.*

Recommendations

Test Suite Could Be Improved

Some opportunities for improving the test suite were identified during the audit. In particular, while the test suite has both positive and negative cases for testing code behavior along with the restrictions that the codebase imposes, there are no fuzzing scenarios in assets such as the `Lib1559Math` library to validate and find edge cases in the implementation. Nor are there invariant tests to find possible inconsistencies by randomizing both inputs and execution paths.

Consider adding fuzzing scenarios in places where data is being calculated/wrapped/encoded/decoded to validate the limitations and find edge cases. In addition, consider implementing an invariant suite focused primarily on non-admin functionalities to find complex exploit paths with randomized data, along with creating a multi-level suite that validates the integration of the different layers during this fork. Such a test suite should consist of contract-level tests with

95%-100% coverage, per chain / layer deployment and integration tests that test the deployment scripts as well as the system as a whole, along with per chain / layer fork tests for planned upgrades. Crucially, the test suite should be documented in such a way that a reviewer can set up and run all these test layers independently of the development team.

Update: *The Taiko team added some additional test in the pull request #18338 and pull request #18475. In addition, the Taiko team noted that additional tests will be added later.*

Conclusion

The audited codebase contains changes made to the Taiko rollup protocol, aiming at adding support for a new mechanism called preconfirmations. Overall, we found the codebase to be well-written. However, it was relatively complex due to the fork event and there still being two different ways of handling the flow of the blocks. Docstrings are mostly present in the codebase, but we recommend having additional external documentation about any design considerations being taken for the dynamic of the preconfers, the registry, and fork events, along with any implications for the codebase. We would also recommend adding more tests to ensure a smooth transition over the fork. Throughout our audit, the Taiko team was responsive and receptive to feedback, providing up-to-date information and specifications of other modules that could contribute to the audit context.