OpenZeppelin | security

ZKsync: ZKChain and Gateway Upgrade Audit

October 24, 2024

Table of Contents

Summary

Type	Layer 2
Timeline	From 2024-09-16 To 2024-10-09
Languages	Solidity

Total Issues	26 (24 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	5 (5 resolved)
Medium Severity Issues	5 (5 resolved)
Low Severity Issues	9 (8 resolved)
Notes & Additional Information	7 (6 resolved)

Scope

We diff-audited the matter-labs/era-contracts repository with head commit ef318e21 against base commit 9615d90, which constitutes the changes, at the time of writing, introduced in pull request 793. A few files have been fully audited, either because of the significance of changes or because they were new files. The fully audited files will be noted in the following list with an asterisk (*).

In scope were the following files related to asset bridging, and migration of ZKChains and data availability modules:

```
da-contracts/contracts/
├── CalldataDA.sol
├── DAContractsErrors.sol *
├── IL1DAValidator.sol
├── RollupL1DAValidator.sol
└── ValidiumL1DAValidator.sol
l1-contracts/contracts
├── bridge
│   ├── BridgeHelper.sol
│   ├── BridgedStandardERC20.sol
│   ├── L1ERC20Bridge.sol
│   ├── L1Nullifier.sol *
│   ├── L2SharedBridgeLegacy.sol
│   ├── L2WrappedBaseToken.sol
│   ├── asset-router
│   │   ├── AssetRouterBase.sol *
│   │   ├── IAssetRouterBase.sol
│   │   ├── IL1AssetRouter.sol
│   │   ├── IL2AssetRouter.sol
│   │   ├── L1AssetRouter.sol *
│   │   └── L2AssetRouter.sol *
│   ├── interfaces
│   │   ├── IAssetHandler.sol *
│   │   ├── IBridgedStandardToken.sol
│   │   ├── IL1AssetDeploymentTracker.sol *
│   │   ├── IL1AssetHandler.sol
│   │   ├── IL1BaseTokenAssetHandler.sol *
│   │   ├── IL1ERC20Bridge.sol
│   │   ├── IL1Nullifier.sol *
│   │   ├── IL1SharedBridgeLegacy.sol *
│   │   ├── IL2SharedBridgeLegacy.sol
│   │   ├── IL2SharedBridgeLegacyFunctions.sol
│   │   └── IL2WrappedBaseToken.sol
│   └── ntv
│       ├── IL1NativeTokenVault.sol
│       ├── IL2NativeTokenVault.sol
```

```
│       ├── INativeTokenVault.sol *
│       ├── L1NativeTokenVault.sol *
│       ├── L2NativeTokenVault.sol *
│       └── NativeTokenVault.sol *
├── bridgehub
│   ├── Bridgehub.sol
│   ├── CTMDeploymentTracker.sol
│   ├── IBridgehub.sol
│   ├── ICTMDeploymentTracker.sol
│   ├── IMessageRoot.sol
│   └── MessageRoot.sol
└── state-transition
    ├── ChainTypeManager.sol
    ├── IChainTypeManager.sol
    ├── ValidatorTimelock.sol
    ├── chain-deps
    │   ├── DiamondInit.sol
    │   ├── ZKChainStorage.sol
    │   └── facets
    │       ├── Admin.sol
    │       ├── Executor.sol
    │       ├── Getters.sol
    │       ├── Mailbox.sol
    │       └── ZKChainBase.sol
    ├── chain-interfaces
    │   ├── IAdmin.sol
    │   ├── IDiamondInit.sol
    │   ├── IExecutor.sol
    │   ├── IGetters.sol
    │   ├── IL1DAValidator.sol
    │   ├── ILegacyGetters.sol
    │   ├── IMailbox.sol
    │   ├── IVerifier.sol
    │   ├── IZKChain.sol
    │   └── IZKChainBase.sol
    ├── data-availability
    │   ├── CalldataDA.sol
    │   ├── CalldataDAGateway.sol *
    │   └── RelayedSLDAValidator.sol
    ├── l2-deps
    │   ├── IL2GenesisUpgrade.sol
    │   └── ISystemContext.sol
    └── libraries
        ├── BatchDecoder.sol *
        └── PriorityTree.sol
l2-contracts/contracts
├── data-availability
│   ├── DAErrors.sol
│   ├── RollupL2DAValidator.sol
│   ├── StateDiffL2DAValidator.sol
│   └── ValidiumL2DAValidator.sol
├── interfaces
│   └── IL2DAValidator.sol
└── verifier
    └── chain-interfaces
        └── IVerifier.sol
```

```
system-contracts/contracts/interfaces
├── IBridgehub.sol *
├── IL2DAValidator.sol
└── IL2GenesisUpgrade.sol
```

Update: *We also audited pull request 939 and pull request 948 which contain the migration from `require` and `revert` statements to custom errors throughout the codebase.*

System Overview

The changeset introduces significant updates to ZKChain migration and settlement on non-L1 layers and the coordination infrastructure for message passing and token bridging. We provide a more detailed description below.

ZKChain Migration

Every new ZKChain is initiated on L1 after which it can be migrated to any whitelisted Settlement Layer (SL). Currently, the only whitelisted SLs are L1 and Gateway. Thus, ZKChains can essentially migrate from L1 to Gateway and back to L1.

The migration path follows the custom asset bridging framework via the `ChainTypeManager` (CTM). Similar to ZKChains, CTMs are initially registered on L1 and can only be registered to another SL via a cross-chain transaction afterwards. CTM registrations on L1 are initiated by the owner of the `Bridgehub` contract and only completed by the owner of the `CTMDeploymentTracker` contract. Each CTM is associated with an `assetId`, just like how the common tokens are tracked in the system.

CTMs handle the basic functionality for a ZKChain's migration. As such, in order to migrate a ZKChain, its associated SL CTM should be deployed and registered on the destination `Bridgehub`. In essence, `Bridgehub` is the asset handler of the CTM assets, just like `NativeTokenVault` is for common assets. For this reason, when a CTM is registered on an SL, it is necessary to also assign the local `bridgehub` contract as its asset handler. Both CTM and common assets share the same asset router contracts.

In light of the above, the complete flow to successfully migrate a ZKChain from L1 to Gateway would require the following cross-chain actions:

1. Deploy an L2 CTM and register it in the destination SL `BridgeHub`.
2. Assign that CTM's asset handler address to be `Bridgehub` on SL.
3. Migrate the ZKChain. The migration itself ensures that the ZKChain's L1 Diamond and helper contracts are updated properly and an L2 Diamond is deployed and properly configured as well.

Once the migration is complete, the ZKChain's batches are committed, proved, and executed on the Gateway. The Gateway itself is responsible for handling the published data of all of its settled ZKChains and settling them altogether on L1.

To migrate back from Gateway to L1, the ZKChain admin initiates a withdrawal request from the Gateway's `L2AssetRouter` contract, setting the `assetId` parameter equal to the `ctmAssetId` for the CTM associated with the migrated zkChain. The final cross-chain message can be used to call `finalizeDeposit` on L1's `AssetRouter` contract and complete the migration by updating the ZKChain's Diamond Proxy and the rest of the helper contracts.

A recovery mechanism has also been implemented in case a ZKChain migration fails.

Multichain Operation Access Control

Since a ZKChain is not restricted to being settled on Ethereum, the same state transition contracts, including the `ChainTypeManager` and the diamond proxy (with all its facets) can be deployed on any legitimate SL, including L1 and the Gateway. Furthermore, communication contracts such as the `Bridgehub` are deployed on all layers, including non-settlement L2s.

As such, some functions are restricted by the layer the contract is deployed on. Additional modifiers such as `onlyL1`, `chainOnCurrentBridgehub`, and `onlySettlementLayerRelayedSender` are used to reflect this. Moreover, further access control is added on key communication functions within a layer, for example, `onlyBridgehub`, `onlyChain`, `onlyAssetRouter`, `onlyNativeTokenVault` or between layers, such as `onlyAssetRouterCounterpart`.

ZKsync Era Legacy Support

To fit into the custom asset bridging (CAB) framework via the `bridgeBurn` and `bridgeMint` functions on the default asset handler, the legacy bridging for ZKsync Era on `L1ERC20Bridge` and `L2SharedBridgeLegacy` are re-routed to their respective asset router contracts.

Contract Renaming

Since the last audited version of the codebase, some contracts have been renamed and/or organized differently:

- The `StateTransitionManager` has been renamed to `ChainTypeManager`.
- The `L1SharedBridge` logic has been split into two separate contracts: `L1AssetRouter` and `L1Nullifier`.
- The `L2SharedBridge` has been renamed to `L2AssetRouter`.

Security Model and Trust Assumptions

Privileged Roles

In relation to the changeset, the following privileged roles can perform critical functionality:

- The owner of coordinating contracts such as `CTMDeploymentTracker`, `Bridgehub`, `L1AssetRouter`, `L1NativeTokenVault`, `L2AssetRouter`, and `L2NativeTokenVault`, can pause/unpause and upgrade the contract logic.
- The owners of the `BridgeHub` and `CTMDeploymentTracker` L1 contracts can register new CTM assets.
- The owner of the `Bridgehub` contract on L1 can add or remove an SL from the whitelist.
- The admin of each ZKChain can initiate and finalize chain migration.

We assume that the accounts in charge of the above actions always act in the intended way. Hence, any attacks or vulnerabilities targeting this part of the system were not considered throughout this audit.

High Severity

H-01 Inconsistent `assetId` Calculation Upon Bridging

In the `_assetIdCheck` function of the `NativeTokenVault` contract, the `_originChainId` used to compute the expected `assetId` is not referring to the chain where to token is native to. Instead, it refers to the chain where the token has been bridged from. In fact, the actual native chain of the token is retrieved later on in the execution flow, upon initializing the bridged token contract.

This breaks the assumption that each token has the same `assetId` across chains, potentially resulting in failed bridging transactions. This does not affect tokens that originate from an L2 and are then bridged to L1 as the `assetId` is coincidentally correctly calculated because the original `chainId` is the same as the chain where it is coming from. However, it impacts the bridging of bridged tokens, i.e., a bridged token on L1 cannot be bridged correctly to another L2, where the incoming `chainId` is different from the origin `chainId`. A use case of this would be bridging a token native to ZKsync Era first to L1 and then to the Gateway.

Consider retrieving the actual origin chain of the bridged token to compute its expected `assetId` upon bridging.

Update: *Resolved in pull request #894.*

H-02 Legacy ERC-20 Token Bridging Request Double Amount

When bridging ERC-20 tokens which are native to L1 via the legacy `L1ERC20Bridge.deposit`, a user is required to deposit twice the requested amount in the following order.

- First, one transfers the requested amount to `L1AssetRouter`.
- Then, in `L1AssetRouter.sol`, the tokens from the original caller are burned. Specifically, the original caller needs to transfer the same amount to

`L1NativeTokenVault` if there is enough allowance. If `L1AssetRouter` is not granted enough allowance, the transfer happens later.

To ensure that the correct amount is transferred for bridging, consider removing the first token transfer to `L1AssetRouter`.

Update: *Resolved in pull request #895.*

H-03 Incorrect `l2tol1Message` Decoding With New Message Format

When a user initiates a withdrawal on `L2AssetRouter.withdraw` using the new message format, the message is encoded with function signature `IAssetRouterBase.finalizeDeposit.selector`, `_assetId`, and `_l1bridgeMintData`. Then, on L1, the user finalizes the deposit on `L1Nullifier`, and the `l2toL1Message` is decoded to get the `_assetId` and `transferData` via _parseL2WithdrawalMessage.

However, the message decoding for the `finalizeDeposit` function selector accounts for an extra 32 bytes for `originChainId` which are not present when the message is packed on `L2AssetRouter`. Thus, the decoded `transferData` will not be correct, resulting in failed execution on L1. This impacts all token bridging from L2 to L1 using the new withdraw message format.

Consider correctly decoding the `12ToL1Message` to allow successful token withdrawal on L1.

Update: *Resolved in pull request #896.*

H-04 Incorrect Chain Balance Accounting For Bridged Token

When an amount of an L2-native token is bridged in from an L2 origin chain to L1, `L1NativeTokenVault` decreases the origin chain's balance through `_bridgeMintBridgedToken`. However, when the token is bridged out of L1, the receiving chain's balance is not accordingly increased in `_bridgeBurnBridgedToken`. This inconsistency can result in failed bridging transactions back to L1 due to underflow when attempting to decrease a chain's balance.

Consider increasing the receiving chain's balance within `_bridgeBurnBridgedToken` to ensure proper accounting consistency.

Update: *Resolved in pull request #897.*

H-05 Zero Chain Balance Increase for Bridged Native Tokens

In the `_bridgeBurnNativeToken` function of the `NativeTokenVault` contract, the token balance increase of the destination ZKChain that accepts the deposit is a no-op. More specifically, the balance increase action is performed before the `amount` assignment resulting in the balance not being updated. As a consequence, a subsequent transaction to bridge back the deposited token funds will fail due to an underflow error.

Consider changing the order of these two instructions so that the balances are properly updated.

Update: *Resolved in pull request #898.*

Medium Severity

M-01 Inconsistent `ctmAssetId` Across Chains

The ctmAssetId function computes the `assetId` for a given `_ctmAddress` by encoding it with the `L1_CHAIN_ID` and `l1CtmDeployer` on the Bridgehub contract deployed on all chains. Thus, on an L2 chain, the returned `ctmAssetId` from `ctmAssetIdFromChainId` is computed from an `l2ctmAddress` because of the value of `chainTypeManager[_chainId]`. As a result, the returned `ctmAssetId` based on the `l2ctmAddress` will be different from the ctm asset ID that has been migrated over from an `l1ctmAddress`.

This has implications for the validation of the `ctmAssetId` during chain migration from the Gateway back to L1. When migrating from L1 to the Gateway, the chain type manager of the migrated chain will be an `l2ctmAddress` set via `bridgeMint` on the Gateway Bridgehub. However, when the chain intends to migrate back to L1, the Gateway `Bridgehub.bridgeBurn` is called with `_assetID` being the `l1ctmAssetId`, causing this check to fail because `ctmAssetIdFromChainId` computes from the `l2ctmAddress` on

Gateway and will not return the same asset ID as the `l1ctmAddress` asset ID. Note that a correctly returned `l2ctmAssetId` will not allow this check to pass either. The migrating `l1ctmAssetId` needs to be validated differently.

Consider correcting the returned CTM address to ensure the consistency of asset IDs across chains and validating them accordingly.

Update: *Resolved in pull request #899.*

M-02 Priority Tree Check Fails When Migrating Back to L1

When a ZKChain migrates back from the Gateway to L1, the Bridgehub forwards it to the ZKChain's existing `AdminFacet.forwardedBridgeMint` function. After this, the priority tree on L1 will be re-initiated by the Gateway priority tree commitment.

When an L2 transaction is requested, a `PriorityOp` is written in the L1 Diamond Proxy before the transaction is forwarded to the Gateway. However, it is never processed because `executeBatch` only happens on the settlement chain (i.e., the Gateway after migration). Only when processing batches can one increase the unprocessed index of the tree. Hence, the L1 `priorityTree.unprocessedIndex` will not increase while the chain is settled elsewhere. For this reason, when the chain migrates back to L1, the re-initiating check should be `_tree.unprocessedIndex <= _commitment.unprocessedIndex` instead of the other way around. Furthermore, the L1 priority `tree.unprocessedIndex` is not updated from the commitment, resulting in wrong values when calling functions such as `getFirstUnprocessedPriorityTx` and `getSize`.

Consider checking that the `unprocessedIndex` in the L1 priority tree is less than the commitment from the Gateway, while updating the `tree.unprocessedIndex` from the commitment when migrating back to L1.

Update: *Resolved in pull request #900.*

M-03 Chain Migration Cannot Be Reattempted After Failure

In the case of a failed chain migration, the `settlementLayer` is set to a new chain ID in the `bridgeBurn` function on the old chain, but the `bridgeMint` function is not successfully executed on the new chain. In this case, one can call the `bridgeRecoverFailedTransfer`

function of the `L1Nullifier` contract in order to reset the `settlementLayer` chainId on L1 `Bridgehub` back to its previous value.

However, within the `BridgeHub` contract, instead of being set back to `block.chainId`, the settlement layer is deleted. This prevents further migration attempts because of a requirement for the `settlementLayer` value in the `bridgeBurn` function.

Consider resetting the `settlementLayer` back to `block.chainid` rather than deleting it.

Update: *Resolved in pull request #918.*

M-04 Transition From `baseToken` to `baseTokenAssetId` Will Fail

The `setLegacyBaseTokenAssetId` function of the `BridgeHub` contract is supposed to set the base token's `assetId` for every ZKChain that is already registered, essentially facilitating the transition from the legacy `baseToken` mapping to the `baseTokenAssetId` mapping.

However, the function is mistakenly a no-op in case `baseTokenAssetId[_chainId]` is unassigned, whereas this should be required for the function to proceed. As a result, assigning an asset ID to the base tokens of already registered chains fails.

Consider fixing the `setLegacyBaseTokenAssetId` function so that, instead of performing a no-op, it requires the base token being set to not already be initialized.

Update: *Resolved in pull request #901.*

M-05 Missing Initialization

The L2AssetRouter does not have an `initialize` function despite its parent contract AssetRouterBase being initializable, ownable, and upgradeable. As such, the `owner` is not initialized resulting in the `onlyOnwner` pause/unpause functionality to not work. Similarly, the L2NativeTokenVault is missing an `initialize` function despite it inheriting from NativeTokenVault which is initializable, ownable, and upgradeable.

Note that since the aforementioned contracts use the upgradeable version of the Ownable and Pausable OpenZeppelin libraries, initialization is only possible through a function decorated with the `initializer` modifier.

Consider adding an `initialize` function to initialize all state variables in the proxy. Alternatively, consider clearly documenting whether these contracts are meant for direct deployment without proxy.

Update: *Resolved in commit 7206350 and pull request #902. The contracts will be deployed directly and the owner will be set during construction.*

Low Severity

L-01 Naming Suggestions

Throughout the codebase, multiple instances of misleading naming were identified:

- The `setAssetHandlerAddress` function in `Bridgehub.sol` suggests that `Bridgehub` is the asset router for the CTM asset which serves chain migration purposes. However, `Bridgehub` itself is the asset handler, not the asset router. Since this function only updates the `ctmAssetIdToAddress` state variable, consider renaming it so that its name clearly reflects its purpose.
- In the setAssetHandlerAddress function in `BridgeHub.sol`, consider renaming the `assetInfo` variable to `ctmAssetId` to be consistent with the rest of the codebase.
- In the setAssetHandlerAddress function of the `L2AssetRouter` contract, the `_assetAddress` parameter name is misleading. Consider renaming it to `_assetHandlerAddress`.
- In the `calculateCreate2TokenAddress` function of the `L1NativeTokenVault` contract, the `l1Token` argument is actually an L2 token address. Consider renaming it to `_l2Token`.
- `BridgeMintData` is used to mint bridged tokens on both L1 and L2 chains. However, the names of its parameters do not reflect the bi-directional encoding. For example, _l2Receiver is in fact an L1 receiver when bridging is from L2. Consider differentiating the direction by non-specific references such as `_remoteReceiver` to represent the receiver address on another chain. Similarly, _l1Token can be simply called `_token` when referring to this chain and `_remoteToken` when referring to the receiving or incoming chain.

Consider adopting the above naming suggestions to improve the clarity and readability of the codebase.

Update: *Resolved in pull request #903.*

L-02 Redundant Code

Throughout the codebase, multiple instances of redundant code were identified:

- In the `Bridgehub` contract, `whitelistedSettlementLayers` is set to `true` for `L1_CHAIN_ID` during the construction of the implementation contract. However, it is only necessary to be set in the `initialize` function.
- In the `L1NativeTokenVault` and `ValidatorTimelock` contracts, the immutable `ERA_CHAIN_ID` state variable is defined and initialized but is never used.
- The `encodeNTVAssetId(uint256 _chainId, bytes32 _assetData)` function of the `EncodingData` library is not used within the codebase.
- In the `setAssetDeploymentTracker` function of the `L1AssetRouter` contract, `block.chainId` is redundantly cast to `uint256`.
- In the `IAssetRouterBase` interface, the `BridgehubWithdrawalInitiated` event is not used anywhere in the entire codebase.
- In `IAssetHandler.sol`, `IChainTypeManager.sol`, and `IAdmin.sol`, the `BridgeInitialize` event remains unused.
- In `L1ERC20Bridge`, the `isWithdrawalFinalized` mapping is never assigned and the code where it is used is essentially a no-op. Consider marking the mapping as deprecated and removing the unused code for improved code clarity.
- In `L1Nullifier.sol`, the `onlyBridgeHubOrEra` and `onlyAssetRouterOrErc20Bridge` modifiers are not used anywhere within the contract.
- The `onlyBaseTokenBridge` modifier defined in `ZKChainBase.sol` is never used in the codebase.
- The `onlyChainCTM` modifier in `Bridgehub.sol` is never used within the codebase.
- The `onlyValidatorOrChainTypeManager` modifier in `ZKChainBase.sol` is never used within the codebase.

Consider removing any redundant or unused code throughout the codebase for enhanced code clarity and readability.

Update: *Resolved in pull request #904.*

L-03 Unreliable L2 Token Address From Zero Address AssetId

In the `L2NativeTokenVault` contract, anyone can call `setLegacyTokenAssetId` with any `_l2TokenAddress`. For a non-existent `_l2TokenAddress`, the value returned by

`L2_LEGACY_SHARED_BRIDGE.l1TokenAddress(_l2TokenAddress)` would be the zero address. This can be used to form a non-zero asset ID and set its `tokenAddress` to be an arbitrary `l2TokenAddress` repeatedly. This results in an unreliable returned l2TokenAddress for that zero-address asset ID for third-party integrations.

Consider reverting a call to `setLegacyTokenAssetId` that attempts to set a non-existent `_l2TokenAddress`. This will ensure consistent return value for all asset IDs.

Update: *Resolved in pull request #905.*

L-04 Pausable Methods Are Not Exposed

If a contract inherits pausable contracts, the `_pause` and `_unpause` internal functions should be exposed using public/external functions. However, `CTMDeploymentTracker` is a pausable contract that does not expose the internal pausable functions. The `whenNotPaused` modifier is not used either.

To ensure the intended pausable functionality of the contracts, consider exposing `_pause` and `_unpause` functions through public or external functions. Alternatively, consider removing the pausable functionality if not intended.

Update: *Resolved in pull request #906. The `CTMDeployerTracker` does not inherit the `PausableUpgradeable` library anymore.*

L-05 Outdated References

Throughout the codebase, multiple instances of docstrings and variable/function names referring to legacy identifiers were identified:

References to "state transition manager":

- `Bridgehub.sol`, line 246: "State transition" should be `chainTypeManager`.
- `Bridgehub.sol`, line 260: "State transition" should be `chainTypeManager`.

References to "shared bridge":

- `Bridgehub.sol`, line 49: "shared Bridge" should be "asset router".
- `Bridgehub.sol`, line 440: "Shared Bridge" should be "asset router".
- `CTMDeploymentTracker.sol`, line 50: `_sharedBridge` should be `l1AssetRouter` (also in the function's parameter).

- `Bridgehub.sol`, line 888: The `sharedBridge` function is defined in a section denoting legacy functions. However, it is used by the `ChainTypeManager` contract to retrieve the asset router contract address. Therefore, neither its call site nor the returned data are related to legacy functionality.

Consider updating all references to legacy names when the corresponding functionality is non-legacy. This will help enhance the clarity and readability of the codebase.

Update: *Resolved in pull request #907.*

L-06 Misleading Documentation

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

`ChainTypeManager.sol`:

- Line 445: does not describe the code below it.
- Line 362: does not describe the code below it.

`DataEncoding.sol`:

- Line 86: does not match the implementation. "assetData" should be "tokenAddress".

`L1AssetRouter.sol`:

- Line 160: the comment states that the caller is not subject to access control because `msg.sender` is encoded in the `assetId`. However, `assetId` remains unused within the call to the `bridgeCheckCounterpartAddress` function, while the original caller is actually access-controlled.

`NativeTokenVault.sol`:

- Line 49: `tokenAddress` should be `originChainId`.
- Line 163: `bridgeBurn` returns calldata that is used not just for L2->L1 messages but also for L1->L2 messages.

`L2AssetRouter.sol`:

- Line 128: the `LEGACY FUNCTIONS` comment is placed earlier than the actual starting point of the legacy functions in the contract.
- Line 133: the comment mistakenly states that this `withdraw` function is legacy.

`L1Nullifier.sol`:

- Line 270: the comment states that the internal `_encodeTxDataHash` function is called but such a function does not exist. Instead, the `DataEncoding` library is called.
- Line 411: `_verifyWithdrawal` handles all withdrawal cases instead of only the "special case from ZKsync Era that were initiated before the Shared Bridge".

`BridgeHub.sol`:

- Line 30: `Bridgehub` is currently the primary entry point only for L1->L2 communication, not L2->L1 communication.

`IAssetHandler.sol`:

- Line 30: `bridgeBurn` returns calldata that is used not only for L2->L1 messages but also for L1->L2 messages.

`CTMDeploymentTracker.sol`:

- Line 25: "Bridgehub" should be "L1AssetRouter".

`Admin.sol`:

- Line 344: the caller is checked to be the chain admin.

`L2ContractAddresses.sol`:

- Line 41: "L2<>L2" should be L1->L2 since L2<>L2 communication is not yet supported.

`MessageRoot.sol`:

- Line 24: "chain tree" should be "shared tree".

`FullMerkle.sol`:

- Line 23: "Bytes32Pushtree" should be "FullTree".
- Line 26: the function does not "reset the tree from blank state" as it merely pushes another zero to the existing tree.

`BridgeHelper.sol`:

- Line 14: the library not only works with L2 contracts on L1 but also with native tokens on both L1 and L2.

Consider providing clear documentation within the codebase for enhanced readability and maintainability.

Update: *Resolved in pull request #908.*

L-07 The `onlyAssetRouterCounterpartOrSelf` Modifier Allows Chain IDs Other Than `L1_CHAIN_ID`

The `onlyAssetRouterCounterpartOrSelf` modifier checks whether the caller is the counterpart (i.e. `l1AssetRouter` address) or self only when the chain ID is `L1_CHAIN_ID`. Since only L2 to L1 communication is supported at the moment, this modifier functions correctly. However, it will not perform the check correctly if other non-L1 chain IDs are activated in the future.

To prevent potential future misuse, consider explicitly handling other chain IDs in this modifier.

Update: *Resolved in pull request #909.*

L-08 Indirect `BridgeHub` Calls Via `ChainTypeManager`

The `chainTypeManager` variable in `ValidatorTimelock` can be replaced by `Bridgehub` as it is only used to get the chain admin or chain address indirectly from the `Bridgehub` since these calls route to `Bridgehub` via the `getZKChain` function.

To avoid unnecessary complexity, consider using `Bridgehub` directly.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *Due to the upgrade process, we need the `ValidatorTimelock` contract to work even before `Bridgehub` is upgraded. This means that we should call the `getZKChain` function on the CTM, as it works before and after the upgrade.*

L-09 Unconventional Proof Length

To prove that a leaf belongs to a Merkle Tree, usually, a proof of length equal to the height of the tree is needed to reconstruct the tree root. This is noted in the `Merkle.sol` library.

However, in order to prove the leaf inclusion of an L2 message, the `logLeafProofLen` is not the tree height, but is the tree height + 1. This is because the aggregated root hash needs to be provided additionally at the end in order to hash to the stored `s.l2LogsRootHashes` value. Similarly, the `batchLeafProofLen` needs to be the chainId's tree height + 1 for settlements done on the Gateway. This discrepancy on the proof aggregation mechanism can be remarked upon for clarity.

Consider adding more docstrings pertaining to the `_proof` content and explicitly note the proof length and the discrency from the `Merkle.sol` library's advice note.

Update: *Resolved in pull request #910.*

Notes & Additional Information

N-01 Todo Comments in the Code

During development, having well-described TODO/Fixme comments makes the process of tracking and solving them easier. However, if left unattended, these comments might age and important information for the security of the system might be forgotten by the time it is released to production. As such, all TODO/Fixme comments should be tracked in the project's issue backlog and resolved before the system is deployed.

Throughout the codebase, multiple instances of unaddressed TODO/Fixme comments were identified:

- The `Todo` comment in line 39 of `IAssetRouterBase.sol`
- The `ToDo` comment in line 112 of `AssetRouterBase.sol`

Consider removing all instances of TODO/Fixme comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO/Fixme to the corresponding issues backlog entry.

Update: *Resolved in pull request #911.*

N-02 Typographical Errors

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

- `DataEncoding.sol`, lines 74-75: `_tokenAaddress` should be `_tokenAddress`.
- `L1AssetRouter.sol`, line 113: the comment has been mistakenly copied from line 124.
- `IL2AssetRouter.sol`, line 21: "assedAddress" should be "assetHandlerAddress".
- `IL1DAValidator.sol`, line 26: `_chainId` should be `_batchNumber`.
- `CalldataDAGateway.sol`, line 9: "process" should be "processing".
- `PriorityTree.sol`, lines 30 and 41: "queue" should be "tree".

Consider fixing the aforementioned typographical errors to improve the clarity and readability of the codebase.

Update: *Resolved in pull request #912. All instances of typographical errors have been fixed.*

N-03 Multiple Contracts With the Same Name

Throughout the codebase, multiple instances of contracts and interfaces with identical names were identified:

- The `CalldataDA` contract in `da-contracts/contracts`
- The `CalldataDA` contract in `l1-contracts/contracts/state-transition/data-availability`
- The `IBridgehub` interface in `system-contracts/contracts/interfaces`
- The `IBridgehub` interface in `l1-contracts/contracts/bridgehub`
- The `IL1DAValidator` interface in `da-contracts/contracts`
- The `IL1DAValidator` interface in `l1-contracts/contracts/state-transition/chain-interfaces`
- The `IL2DAValidator` interface in `l2-contracts/contracts/interfaces`
- The `IL2DAValidator` interface in `system-contracts/contracts/interfaces`
- The `IL2GenesisUpgrade` interface in `system-contracts/contracts/interfaces`
- The `IL2GenesisUpgrade` interface in `l1-contracts/contracts/state-transition/l2-deps`
- The `IVerifier` interface in `l1-contracts/contracts/state-transition/chain-interfaces`
- The `IVerifier` interface in `l2-contracts/contracts/verifier/chain-interfaces`

Consider naming all contracts uniquely to avoid unexpected behavior and improve the overall clarity and readability of the codebase.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *This is unfortunately the case due to compilation issues. We use two separate compilers, one for EVM the other for zkEVM i.e. EraVM. Some contracts can only be compiled with one of the compilers. Some other contracts are deployed on L1, some on L2, some on both (i.e. Bridgehub). These interfaces/contracts share names because they are the same, they are just in different locations due to these complications.*

N-04 Duplicated Code

In the `L1Nullifier` contract, the `nullifyChainBalanceByNTV` function only allows calls made by the `L1NativeTokenVault` contract. However, the `onlyL1NTV` modifier is already supposed to handle this kind of access control.

Consider decorating the `nullifyChainBalanceByNTV` function with the `onlyL1NTV` modifier to avoid code duplication.

Update: *Resolved in pull request #913.*

N-05 Variables Could Be `immutable`

If a variable is only ever assigned a value from within the `constructor` of a contract or during compile time, then it could be declared as `immutable`.

Throughout the codebase, multiple instances of variables that could be `immutable` were identified:

- The `l1AssetRouter` state variable in `L2AssetRouter.sol`
- The `l2TokenProxyBytecodeHash` state variable in `L2NativeTokenVault.sol`

To better convey the intended use of variables and to potentially save gas, consider adding the `immutable` keyword to variables that are only set in the constructor.

Update: *Resolved in pull request #914. The Matter Labs team stated:*

> *On L2, immutables are more expensive at the moment, but were added for readability.*

N-06 Unused Error

Unused code can reduce code clarity and make it difficult to reason about the implemented logic. In `DAContractsErrors.sol`, the `PubdataCommitmentsTooBig` error is unused.

To improve the overall clarity and readability of the codebase, consider either using or removing any currently unused errors.

Update: *Resolved in pull request #915. All unused errors have been removed.*

N-07 Duplicate Imports

Throughout the codebase, multiple instances of duplicate imports were identified:

- `import {MigrationPaused, AssetIdAlreadyRegistered, ChainAlreadyLive, ChainNotLegacy, CTMNotRegistered, ChainIdNotRegistered, AssetHandlerNotRegistered, ZKChainLimitReached, CTMAlreadyRegistered, CTMNotRegistered, ZeroChainId, ChainIdTooBig, BridgeHubAlreadyRegistered, AddressTooLow, MsgValueMismatch, ZeroAddress, Unauthorized, SharedBridgeNotSet, WrongMagicValue, ChainIdAlreadyExists, ChainIdMismatch, ChainIdCantBeCurrentChain, EmptyAssetId, AssetIdNotSupported, IncorrectBridgeHubAddress} from "../common/L1ContractErrors.sol";` imports duplicated alias `CTMNotRegistered` in `Bridgehub.sol`.
- `import {DataEncoding} from "../common/libraries/DataEncoding.sol";` is duplicated in `L1Nullifier.sol`.
- `import {Unauthorized, SharedBridgeKey, DepositExists, AddressAlreadySet, InvalidProof, DepositDoesNotExist, SharedBridgeValueNotSet, WithdrawalAlreadyFinalized, L2WithdrawalMessageWrongLength, InvalidSelector, SharedBridgeValueNotSet, ZeroAddress} from "../common/L1ContractErrors.sol";` imports duplicated alias `SharedBridgeValueNotSet` in `L1Nullifier.sol`.
- `import {IChainTypeManager} from "../../IChainTypeManager.sol";` is duplicated in `Mailbox.sol`.
- `import {IBridgehub} from "../../../bridgehub/IBridgehub.sol";` is duplicated in `Mailbox.sol`.

Consider removing duplicate imports to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #916. All duplicate imports have been removed.*

Conclusion

The audited codebase updates key bridging functionalities with added access control for multi-layer deployment. Within this asset bridging framework, ZKChain migration between L1 and Gateway is supported and settlement on Gateway is considered. Given the complexity and the backward compatibility requirement, we recommend more integration tests covering chain migration from L1 to Gateway and back to L1 to further improve the robustness of the mechanism. We deeply appreciate the Matter Labs team for their generous support throughout the audit which greatly aided our understanding of the system.