OpenZeppelin | security

ZKsync Custom Asset Bridge Audit

August 9, 2024

Table of Contents

Summary

Type	ZK Rollup
Timeline	From 2024-06-17 To 2024-07-08
Languages	Solidity

Total Issues	25 (24 resolved, 1 partially resolved)
Critical Severity Issues	1 (1 resolved)
High Severity Issues	2 (2 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	7 (7 resolved)
Notes & Additional Information	14 (13 resolved, 1 partially resolved)

Scope

We diff-audited pull request #484 of the matter-labs/era-contracts repository at commit d397b7f. New contracts were fully audited. Other files that were fully audited despite only being partially changed are explicitly tagged as "full" in the scope definition given below.

The following files were in scope:

```
├── l1-contracts
│   └── contracts
│       ├── bridge
│       │   ├── L1NativeTokenVault.sol
│       │   ├── L1SharedBridge.sol (full)
│       │   └── interfaces
│       │       ├── IL1AssetHandler.sol
│       │       ├── IL1NativeTokenVault.sol
│       │       ├── IL1SharedBridge.sol
│       │       ├── IL2Bridge.sol
│       │       └── IL2BridgeLegacy.sol
│       ├── bridgehub
│       │   ├── Bridgehub.sol (full)
│       │   └── IBridgehub.sol
│       ├── common
│       │   ├── Config.sol
│       │   └── libraries
│       │       └── UnsafeBytes.sol
│       └── state-transition
│           ├── chain-deps
│           │   └── facets
│           │       ├── Getters.sol
│           │       └── Mailbox.sol
│           └── chain-interfaces
│               ├── IGetters.sol
│               └── IMailbox.sol
└── l2-contracts
    └── contracts
        ├── L2ContractErrors.sol
        ├── L2ContractHelper.sol
        └── bridge
            ├── L2NativeTokenVault.sol
            ├── L2SharedBridge.sol (full)
            └── interfaces
                ├── IL2AssetHandler.sol
                ├── IL2NativeTokenVault.sol
                ├── IL2SharedBridge.sol (full)
                └── ILegacyL2SharedBridge.sol
```

System Overview

The upgrade under review is driven by the goal of supporting bridging custom assets. This has necessitated a redesign of the existing bridge architecture to relieve token developers from the burden of creating their own bridges and repeatedly implementing the same bridge logic. Thus, the new concept of asset handlers, the new `L{1,2}NativeTokenVault` contracts, and changes to the `L{1,2}SharedBridge` have been introduced. Until now, the `L{1,2}SharedBridge` only supported ETH and ERC-20 tokens by deploying a standard ERC-20 implementation on L2. With this upgrade, assets are managed by handlers and identified by their ID, which is a hash of the origin chain ID, the initial asset handler registrant, and any `bytes32` asset data value (e.g., the padded token address). The shared bridge maintains a mapping of asset handlers for each asset ID bridged by users.

An asset handler is a smart contract on both layers that handles the burning, minting, locking, and releasing of bridged tokens. It must implement the `bridgeMint` and `bridgeBurn` functions on both layers, as well as the `bridgeRecoverFailedTransfer` function on L1. The shared bridge functionality of bridging standard ERC-20 tokens and ETH has been outsourced to the `L{1,2}NativeTokenVault` asset handler which will also hold all of the current shared bridge funds. For custom token needs, developers can deploy the token contract on L2 along with the asset handler on each layer, which can permissionlessly be registered in the shared bridge. Users can then refer to the associated asset ID when depositing/withdrawing this token, with the shared bridge partially delegating the flow to the asset handler.

The change from token address to asset ID required small adjustments to the `Bridgehub` and `Mailbox` facets to track each chain's base asset ID and to comply with the latest shared bridge interface. In addition, the legacy `L1ERC20Bridge` now forwards the tokens to the `L1NativeTokenVault`. Other than that, the main control flow and integrations remain unchanged while backwards compatibility, notably in the shared bridge, is maintained.

Security Model and Trust Assumptions

Several trust assumptions were made during our audit of the protocol:

- All of the addresses with the roles mentioned in the *Privileged Roles* section below can impact users in different ways and some, if compromised, could freeze or steal users' funds. These actors are assumed to behave honestly and in the best interests of the protocol users.
- Deposits and withdrawals from batches before the last bridge upgrade are considered to have been finalized prior to the upgrade and handled separately.
- The bridge address called as part of the `requestL2TransactionTwoBridges` function is expected to fully implement the `IL1SharedBridge` interface. For instance, this interface should allow users to reclaim their assets in case of a failed deposit.
- The upgrade is assumed to be well-coordinated, such that the contracts are upgraded and funds are transferred all at the same time. Due to interface changes, parts of the code will break if not upgraded simultaneously.
- ZKchains are assumed to have valid state transitions. For example, it is assumed that they can not force an invalid state transition in order to steal assets from the bridge.
- The `L2NativeTokenVault` contract is expected to be deployed at the current `L2SharedBridge` implementation, while the `L2SharedBridge` becomes a system contract. However, a compatible storage layout has not yet been finalized. As this is a known issue, we assume that the storage layout will still be adapted to not cause conflicts.

The above trust assumptions are considered inherent to the current design of the codebase.

Privileged Roles

There are multiple privileged roles in the system:

- **Admin**: Bridge contracts are deployed behind transparent proxies. The admin of these proxies on L1 and L2 can arbitrarily upgrade the contract implementations.
- **Owner**: The `Bridgehub`, `L1SharedBridge`, `L1NativeTokenVault`, and `L2NativeTokenVault` contracts have an owner. This owner can call privileged functions and pause some functionalities in the L1 contracts.

- **Asset Handler Registrant**: The initial registrant associated with an asset ID can edit the address of the asset handler on L1 and L2 at any time. Users should be aware of the risk of using custom asset IDs when bridging their assets.

Design Considerations

The following design considerations are motivated by the issues found during the audit and the perceived code clarity:

- The bridge contracts make use of multiple encodings to pass data and information from one layer to the other. Data encoding and decoding are always made in-place wherever they are needed, but mismatches between the two can result in issues. For concrete examples, please refer to H-01 and L-04. The approach taken appears to be prone to errors, difficult to maintain, and introduces code redundancy. Instead, a library containing the common encodings could be implemented and reused throughout the codebase.
- Backwards compatibility is a guarantee for developers, ensuring that their bridge integrations do not break with the upgrade. However, it also increases the surface for potential security issues. For instance, H-02 originates from the need to maintain backwards compatibility. Backwards compatibility makes the code more complex due to multiple control flow paths in parallel that want to achieve the same behavior, but are difficult to maintain, require extra code, and can introduce mismatching outcomes that lead to issues. When backwards compatibility is no longer necessary in the future, consider revisiting the bridge contracts and updating them to only support the latest version.

Critical Severity

C-01 All Assets Can Be Stolen From the Vault

Users can bridge ERC-20 tokens and ETH from L1 to L2 (and back) by using the native token vaults as asset handlers (L1, L2). A standard L2 token is deployed as a beacon proxy by the `L2NativeTokenVault` for each new token bridged from L1. Such tokens are minted when assets are bridged from L1 using the `L1NativeTokenVault` as the asset handler.

However, the `bridgeMint` function is not protected by the `onlyBridge` modifier and can be called by anyone on L2. This means that anyone can mint any amount of any token that is managed by the native token vault on L2, which could then be bridged back to drain all the assets in the `L1NativeTokenVault`, including those currently in the shared bridge.

Consider adding an `onlyBridge` modifier to the `bridgeMint` function.

Update: *Resolved in pull request #618 at commit c564076.*

High Severity

H-01 Mismatching Encoding Prevents Bridge Recovery

Funds can be recovered through the `L1SharedBridge` in case of a failed non-legacy deposit. A successful recovery requires that the hash of the sender, the token address, and the amount matches the stored deposit information which is stored through the `bridgehubConfirmL2Transaction` function that is called by the `Bridgehub`. This function stores the `txDataHash` returned by the `L1SharedBridge`.

However, this `txDataHash` returned by the bridge is encoded and hashed differently compared to during the recovery. This is because the recovery encodes `address l1Token` and `uint256 amount`, while the deposit function encodes a `bytes32 _assetId` and `bytes _transferData` for non-legacy deposits. Due to this mismatch, a recovery is impossible and user funds would be locked in the bridge.

Consider implementing an `internal` function to construct a consistent transaction data hash for both functions.

Update: *Resolved in pull request #619 at commit 16241e4. A function was implemented to handle the hash generation. During the recovery a try/catch mechanism determines whether the transaction data is hashed over the legacy or new encoding and properly checks it against the stored hash.*

H-02 User Funds Can Be Frozen in the `L1NativeTokenVault`

Users can bridge ERC-20 tokens from L1 to L2 in three different ways:

1. By calling `requestL2TransactionDirect` or `requestL2TransactionTwoBridges` on the `Bridgehub` contract.
2. By calling `requestL2Transaction` on the `MailboxFacet` contract (through the diamond proxy).
3. By calling `deposit` on the `L1ERC20Bridge` contract.

In practice, in most cases, the asset bridged will be safeguarded by the `L1NativeTokenVault` (as the canonical asset handler). When the `L1NativeTokenVault` handles an asset, `bridgeBurn` or `bridgeMint` is called. These two functions add or subtract the amount of tokens sent from the `chainBalance` mapping, respectively. This mapping is used as a security measure to segregate the funds sent by chain ID, ensuring that no more than the deposited amount of assets can be withdrawn from a certain chain.

However, the `deposit` function (see 3. above) does not increase the `chainBalance` mapping as it sends the funds directly to the `L1NativeTokenVault`. Still, withdrawals decrease the `chainBalance`, possibly reverting if it becomes negative. This asymmetry creates an imbalance, leading to a state where all funds cannot be bridged back as soon as `deposit` is called.

This could be used by a malicious user to actively freeze funds by calling `deposit` to bridge x amount of an ERC-20 token, and then immediately bridging back to L1. The net effect of this scheme would be to reduce `chainBalance` by x. By choosing x as the current `chainBalance`, the malicious user could cause all the future calls made to `bridgeMint` for a certain asset to revert, thereby causing users' funds to effectively be stuck in the `L1NativeTokenVault`.

Consider calling `bridgeBurn` on the `L1NativeTokenVault` when depositing through the `L1ERC20Bridge`. Note that this would require handling the allowance given by the user to the `L1NativeTokenVault`, similar to what is done for the `L1SharedBridge`.

Update: *Resolved in pull request #620 at commit 992c85a. The ERC-20 tokens are deposited from the legacy bridge to the shared bridge, which gives an allowance to the native token vault.*

Medium Severity

M-01 L2 Native Tokens Become Unmintable on Beacon Change

Any ERC-20 token can be bridged to L2 through the native token vault. On the first mint of a newly bridged token, a standard contract will be deployed using the beacon proxy pattern. The beacon proxy (L2 token) is deployed at a deterministic address using the `create2` function of the deployer system contract. As the L2 token address is deterministic given the L1 token address, the `bridgeMint` function is called on that expected L2 token address.

The problem is that the L2 token address is deterministic under three variables: The L1 token address, the beacon proxy bytecode hash, and the beacon address as a constructor argument. The latter two can be configured by the owner calling the `setL2TokenBeacon` function. Thus, if these beacon variables change, the L1 token address leads to a different expected address without any deployed code, making the `bridgeMint` call revert. This means that no more of this L2 token can be minted unless the beacon change is reverted.

Instead of calling the expected token address, consider calling the token address that is stored during deployment to guarantee that the same asset ID leads to the same token contract. Furthermore, consider moving the address computation logic from the `l2TokenAddress` function to an `internal` `_calculateCreate2TokenAddress` function. This function could be called within the `if`-block of the `bridgeMint` function instead of outside. Moreover, in the `l2TokenAddress` function, the asset ID could be computed to return the address stored in the `tokenAddress` mapping if it is non-zero. Otherwise, the address returned by `_calculateCreate2TokenAddress` could be returned.

Alternatively, consider if the beacon proxy bytecode hash and beacon address **ever** have to be changed. If this is not the case, the `setL2TokenBeacon` function can be removed, with the above implications clearly documented in the code.

Update: *Resolved in pull request #621 at commit 82ffa3a. The Matter Labs team applied the first suggestion, making use of the stored token address.*

Low Severity

L-01 Lack of Events

The `Bridgehub` contract has various owner-permitted functions to edit the set of active `StateTransitionManagers`, add a base token, and update the shared bridge in use, without emitting any events.

For improved transparency, consider emitting events when interacting with any of these functions.

Update: *Resolved in pull request #622 at commit ed9cc80.*

L-02 Misleading Value Is Forwarded During Deposit

The `L1SharedBridge` contract receives the `_l2Value` parameter in the `bridgehubDeposit` function and forwards this as `_mintValue` when calling the `bridgeBurn` function of the asset handler. However, `mintValue` and `l2Value` are conceptually different. The `mintValue` is what is minted on L2 as a native asset and what covers the transaction fee. On the other hand, the `l2Value` is the `msg.value` of the L2 transaction.

Consider clarifying the intention of the forwarded value.

Update: *Resolved in pull request #624 at commit 46b84aa.*

L-03 Imprecise Error

Users can deposit any `assetId` through the `Bridgehub` expecting that an asset handler is registered in the shared bridge that will handle the deposit. However, if no such asset handler is registered in the `assetHandlerAddress` mapping, the `bridgeBurn` call to `address(0)` will revert with a generic error.

Consider checking beforehand that the asset ID has a defined handler and giving a specific revert reason if that is not the case.

Update: *Resolved in pull request #625 at commit 2d57678.*

L-04 Fragile Encodings

The bridge contracts make use of many encodings for the handling of assets. As seen in *Mismatching Encoding Prevents Bridge Recovery*, this led to a severe issue of locked funds. In addition, there are two more encodings which appear fragile, although a concrete issue could not be identified:

- The asset ID is a hash of the encoded chain ID, the initial asset handler registrant, and a `bytes32` asset data value. However, sometimes, this asset data is encoded as an `address` [1, 2] and sometimes as a `bytes32` value [3, 4, 5]. Due to the words being padded when using `abi.encode`, both variations lead to the same ID. Still, consider handling the ID generation with one function for standardization and ease of maintenance.
- When bridging tokens from L1 to L2 through the native token vault, `bridgeBurn` on L1 will provide the `bridgeMintData` passed to `bridgeMint` on L2. However, there is a dangerous mismatch between the encoding and decoding of that mint information, as the `amount` matches the `l1Sender` information and vice-versa. Luckily, this is not a problem because `bridgeMintData` is correctly decoded and encoded **for a second time**. The `L1SharedBridge` decodes the `bridgeMintData` to encode the call to the legacy `finalizeDeposit` function. That function in the `L2SharedBridge` then correctly encodes the data that is passed to the `L2NativeTokenVault`. Overall, the flow of data is not intuitive to follow and the double encoding is prone to errors. Consider simplifying the functions involved and using the same encoding of data in the native token vaults.

Consider creating a common library that will handle the encodings and decodings of bridge-related data in one place. Furthermore, consider simplifying the data flow to make code review and maintenance easier.

Update: *Resolved in pull request #627 at commit b70d2a7.*

L-05 Custom Assets May Not Be Withdrawable

When assets are withdrawn through the `L2SharedBridge`, the asset handler for this asset ID returns the bridge mint data that will be passed into the `bridgeMint` function of the L1 asset handler. In the case of a native token vault, for example, the bridge mint data encodes the amount and address of the L1 receiver. In an intermediate step, the shared bridge packs the `finalizeWithdrawal` selector, asset ID, and bridge mint data into an L1 withdraw message that is parsed in the `_parseL2WithdrawalMessage` function.

In this function, the L1 withdraw message is checked to be at least 56 bytes long. With the selector being 4 bytes and the asset ID 32 bytes, this means that the bridge mint data has to have 20 or more bytes. For the vast majority of custom asset handlers, this is expected to work out because the L1 receiver address will likely be encoded in it. However, there might be edge cases where less than 20 bytes are needed and, thus, the withdrawal will fail.

Consider checking the (minimum) expected length of the bridge mint data for each of the message formats.

Update: *Resolved in pull request #628 at commit de49b6e.*

L-06 ETH's `L2StandardERC20` Representation Does Not Have `name` or `symbol`

When bridging on L1 using the `L1NativeTokenVault` as asset handler, token metadata (`name`, `symbol`, `decimals`) is passed alongside the bridging information. This metadata is decoded on L2 when deploying a new `L2StandardERC20`.

However, ETH metadata contains the name and symbol as `bytes("Ether")` and `bytes("ETH")` respectively. These bytes arrays are not valid abi encoding and thus cannot be decoded in the `decodeString` function of the `L2StandardERC20` contract. This means the `L2StandardERC20` token representation of ETH on L2 would not have a `name` or `symbol` metadata, which is unintended.

Consider abi-encoding the names instead of casting them to `bytes` when sending the metadata for the ETH token.

Update: *Resolved in pull request #629 at commit c68390a.*

L-07 Inaccurate Legacy Deposits Identification

When a failed transfer is recovered, a check is performed to ensure that the transaction is not an Era legacy deposit. This is done by checking the following condition:

```
_l2BatchNumber < eraLegacyBridgeLastDepositBatch ||
(_l2BatchNumber == eraLegacyBridgeLastDepositBatch &&
_l2TxNumberInBatch < eraLegacyBridgeLastDepositTxNumber)
```

However, `eraLegacyBridgeLastDepositTxNumber` is defined as "The tx number in the `_eraLegacyBridgeLastDepositBatch` of the last deposit tx initiated by the legacy bridge". Thus, the very last transaction numbered `eraLegacyBridgeLastDepositTxNumber` in batch number `_eraLegacyBridgeLastDepositBatch` would be incorrectly identified as not being an Era legacy deposit.

Consider changing the `<` operator to `<=`.

Update: *Resolved in pull request #630 at commit c0749f4.*

Notes & Additional Information

N-01 Unused Code

Throughout the codebase, several instances of unused or obsolete code were identified:

- The following code is unused:
  - The `onlySelf` modifier of `L1NativeTokenVault`
  - The `IL2SharedBridgeLegacy` interface
- The following events are unused:
  - `AssetHandlerRegistered` of `IL1SharedBridge`
  - `AssetHandlerRegisteredInitial` of `IL2SharedBridge`
- The following imports are unused:
  - The `IL1ERC20Bridge` import in `L2SharedBridge.sol`
  - The `ILegacyL2SharedBridge` import in `L2SharedBridge.sol` (imported twice)

- The following variables are unused:
  - `ERA_CHAIN_ID` of `L1NativeTokenVault`
  - `hyperbridgingEnabled` of `L1SharedBridge`
  - `l1TokenAddress` of `L2SharedBridge`
  - `l1Bridge` of `L2SharedBridge` . It is obsolete to maintain and use in the `onlyL1Bridge` modifier because all calls to the `L2SharedBridge` are done through the `L1SharedBridge` [1, 2, 3]. This deprecation will also make `ERA_CHAIN_ID` of `L2SharedBridge` obsolete.

Consider removing any unused code. For state variables, consider marking them as deprecated. Note that removing a state variable would shift the storage layout leading to unexpected behavior with upgradeable contracts.

Update: *Resolved in pull request #617 at commit 1b0ce84. The Matter Labs team stated:*

> *`_hyperbridgingEnabled`, `l1TokenAddress`, and `l1Bridge` are variables that we have to keep for backwards compatibility, so marking it as Depracated would mean we need to introduce a function for it.*

N-02 Incorrect and Missing Documentation

Throughout the codebase, several instances of incorrect documentation were identified:

- The `onlyBridge` modifier of the `L1NativeTokenVault` contract permits the `L1_SHARED_BRIDGE` to be the message sender instead of the documented bridgehub.
- The `onlySelf` modifier is documented to be callable by the "shared bridge itself" whereas the contract is the `L1NativeTokenVault`.
- The `requestL2TransactionDirect` and `requestL2TransactionTwoBridges` functions are documented with "the msg.sender has approved mintValue allowance for the sharedBridge", whereas the allowance could also be given to the `L1NativeTokenVault`.
- The `createNewChain` function reverts with "Bridgehub: weth bridge not set" when the `sharedBridge` address is not set which does not allow for WETH to be deposited.
- The `_assetData` parameter of the `bridgeRecoverFailedTransfer` function is documented as "The amount of the deposit that failed", whereas `_assetData` is expected to encode `uint256 amount` and `address prevMsgSender`.
- The documentation of the `depositHappened` mapping says "Tracks deposit transactions from L2", whereas it should be "from L1" or "to L2". Furthermore, it

documents the `depositDataHash` in the legacy format with the token address and amount instead of asset ID and transfer data.

- The documentation of the `assetHandlerAddress` says "A mapping l2 token address => l1 token address", whereas it maps an asset ID to an L2 asset handler address.
- The `_getL1WithdrawMessage` function comment uses the "`IL1ERC20Bridge.finalizeWithdrawal` function selector", whereas the function is using the `IL1SharedBridge.finalizeWithdrawal` function selector.
- The code documentation goes by the old branding of "zkSync" instead of "ZKsync".
- The `param mintValue` documentation is missing `@` and `_`.
- The `_mintValue` variable is commented as being withdrawn by the "base token bridge", whereas the shared bridge is in charge of depositing base tokens.
- The documentation around the `setNativeTokenVault` was copied from the `setL1Erc20Bridge` function and does not correspond to what the function is actually doing.
- The documentation in the `_parseL2WithdrawalMessage` mentions that "there are two versions of the message". However, the `if/else` flow control statement below has three branches.
- In the first branch of the `_parseL2WithdrawalMessage` function, the message could be sent by calling `withdrawWithMessage` on the `L2BaseToken`. Consider documenting this extra possibility and the fact that in such a case, the extra arguments would be ignored.
- The documentation around the `bridgehubDeposit` function in the `IL1SharedBridge` interface only documents the legacy format for the `_data` argument. The data could be encoded differently.
- The NatSpec above the `sharedBridge` state variable references it as being the "weth bridge".
- A comment in the `createNewChain` function starts with "///" which is intended for documentation. Consider replacing it with a normal comment.
- The NatSpec above `setPendingAdmin` in the `IBridgeHub` interface says that "only the current admin can propose a new pending one". However, the owner can also propose a new admin.
- The NatSpec above the `withdraw` function in the `L2SharedBridge` contract documents the `_assetId` as "The L2 token address which is withdrawn".

In addition, the following instances of missing documentation were identified:

- The `getAssetId` function is not documented.
- The `setAssetHandlerAddressInitial` function documentation could be more complete to guide custom asset handler developers.

- The `setAssetHandlerAddressOnCounterPart` function is not documented.
- The legacy functions in `L2SharedBridge` are not documented.
- The `setL2TokenBeacon`, `bridgeMint`, and `bridgeBurn` functions are not documented.

To improve code clarity and readability, consider addressing the aforementioned instances of incorrect and missing documentation.

Update: *Resolved in pull request #623 at commit d85474c.*

N-03 Code Redundancy

Throughout the codebase, a few instances of redundant code were found:

- The `claimFailedDepositLegacyErc20Bridge` function is redundant to the `claimFailedDeposit` function except for the `_chainId` parameter. Consider calling the `claimFailedDeposit` function from the legacy bridge while specifying the `ERA_CHAIN_ID` as `_chainId`.
- The `bridgeRecoverFailedTransfer` and `bridgeMint` function are identical except for the address returned and `BridgeMint` event being emitted in the `bridgeMint` function. Consider reusing the same logic to ease maintenance.
- The `getERC20Getters` function in the `L1SharedBridge` is equally implemented in the `L1NativeTokenVault`. Consider reusing the logic from a common library implementation.

Consider applying the above suggestions to reduce the footprint of the codebase and make its maintenance less error-prone.

Update: *Resolved in pull request #634 at commit f43178f. The second bullet point is no longer applicable with the H-01 fix.*

N-04 Naming Suggestions

Throughout the codebase, a few instances of contract members that could be better named were found:

- The `transferBalancesFromSharedBridge` and `transferBalanceToNTV` functions do not transfer any tokens but instead update a mapping used for internal accounting. Consider renaming them to `updateChainBalancesFromSharedBridge` and `nullifyChainBalanceByNTV` or something similar.

- The name of the `_assetAddressOnCounterPart` parameter of the `setAssetHandlerAddressOnCounterPart` function suggests that it refers to the token address, whereas the parameter is actually used to set the asset handler address on the L2 side. Consider renaming it to `_assetHandlerAddressOnCounterPart`.
- The `BridgehubDepositFinalized` event is emitted when the `Bridgehub` flow is completed on the L1 side, without guaranteeing its success on L2. However, "Finalized" is usually used to refer to the successful completion of bridge operations on the counterpart (e.g., the `FinalizeDepositSharedBridge` or `WithdrawalFinalizedSharedBridge` event). To prevent confusion, consider renaming the event to `CompletedL1BridgehubDeposit` or something similar.

For improved code clarity, consider applying the above renaming suggestions.

Update: *Partially resolved in pull request #635 at commit 8a65222. The Matter Labs team stated:*

> *`BridgehubDepositFinalized` renaming makes sense, but that event is already in production so someone might rely on it, so I don't think we should change that. Fixed the others.*

N-05 Gas Optimizations

Throughout the codebase, several opportunities for gas optimizations were identified:

- In the `Bridgehub`, the `secondBridgeAddress` check could be placed before the `bridgehubDeposit` call to fail early and save some gas.
- The usage of the `MessageParams` struct solely for the internal call between `_finalizeWithdrawal` and `_checkWithdrawal` requires memory space to be allocated, without making the code cleaner. Consider propagating the message data directly through function parameters.
- The same `baseTokenAssetId` storage variable is read twice in the `requestL2TransactionTwoBridges` function.
- The `ntvAddress` stack variable can be reused instead of re-reading `nativeTokenVault` from storage.
- The `_l2ToL1message` parameter can be handled in `calldata`.
- The `ReentrancyGuard` could use transient storage introduced in EIP-1153. An inspiration can be found in OpenZeppelin's contracts library.

Consider applying the above suggestions to make the code more gas efficient.

Update: *Resolved in pull request #636 at commit 8ec22b4. The Matter Labs team stated:*

> *`MessageParams` were needed because of stackTooDeep issues. `_l2ToL1message` needs to be in memory as it is handled by `readRemainingBytes`. TransientStorage for `ReentrancyGuard` is true, however optimising that is out of scope for now, we did not write that contract for this upgrade.*

N-06 Misleading Error

The `AssetIdMismatch` error reverts with two `bytes32` values, which are the expected and supplied asset IDs. However, in the `bridgeMint` function where the error is used, the parameters are given in reverse order, with the expected asset ID coming second.

Consider swapping the variable order in the emitted event to prevent confusion when interpreting the output.

Update: *Resolved in pull request #637 at commit e482840.*

N-07 Unreachable Code

Before the changes being audited were made, the bridge contracts referred to assets by their token address. This was changed by referring to the asset ID. To maintain backwards compatibility when depositing a base token, the `_getAssetProperties` function takes the input in either format (padded token address or asset ID) and returns the asset handler and ID.

However, the `bridgehubDepositBaseToken` function is always called with a valid asset ID [1, 2, 3], while also conforming to the right interface. Thus, since the asset ID cannot be a padded token address, the deposit call would always revert on the address check if a base token is not yet registered. In other words, the `registerToken` call cannot be reached.

Consider simplifying the `bridgehubDepositBaseToken` function by checking that the asset ID has a registered handler and reverting otherwise. It is a good practice to keep the code self-contained (e.g., to expect that a token was previously registered instead of invoking the registration within the same call).

Update: *Resolved in pull request #638 at commit 23fcec7. The code was simplified by removing the `_getAssetProperties` function.*

N-08 Typographical Errors

There is a typographical error in the `IL1NativeTokenVault` interface, where "Used to get the the ERC20 data for a token" has one extra "the".

To improve code readability, consider correcting any typographical errors.

Update: *Resolved in pull request #631 at commit 8e114ec.*

N-09 Misleading Function Name

The `l2TokenAddress` function of the `L2SharedBridge` contract returns the calculated L2 counterpart address of an L1 token address, assuming it was deployed through the `L2NativeTokenVault`. However, as the shared bridge can use custom asset handlers, or an L1 token may not have an L2 counterpart, this function's name can be misleading.

Consider checking that the L1 token is indeed part of the L2 native token vault, and clarifying the behavior of the function in the documentation.

Update: *Resolved in pull request #642 at commit 0d0e218. Documentation was added to raise awareness about the potentially misleading return value.*

N-10 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are several instances of contracts not having a security contract:

- The `IBridgehub` interface
- The `IL1ERC20Bridge` interface
- The `IL2AssetHandler` interface
- The `IL2Bridge` interface

- The `IL2BridgeLegacy` interface
- The `IL2NativeTokenVault` interface
- The `IL2SharedBridge` interface
- The `IL2SharedBridgeLegacy` interface
- The `ILegacyL2SharedBridge` interface

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #641 at commit 040026b.*

N-11 Incomplete and Mismatching Interfaces

Throughout the codebase, several instances of interface mismatches were identified:

- The `_msgSender` parameter is implemented as `_prevMsgSender`.
- The `_tokenData` parameter is implemented as `_assetData`.
- The `_baseToken` parameter is implemented as `_token`.
- The `l1TokenAddress` parameter is implemented as `_l1TokenAddress`.
- The `l1Token` return parameter is implemented as `l1Asset`.
- The `txHash` named return parameters [1, 2] are implemented as `l2TxHash` [3, 4, 5].
- The `_l2Token` parameter is implemented as `l2TokenAddress`.
- The `finalizeDeposit` function `_data` parameter [1, 2] is implemented as `_transferData` [3, 4] and the `withdraw` function `_data` parameter is implemented as `_assetData`.
- Leading underscores on mapping variables are inconsistent. The `Bridgehub` and interface getters have leading underscores, while most contracts do not. This makes such contracts inconsistent with other contracts as well as their own interfaces (1.1, 1.2), (2), (3.1, 3.2), (4.1, 4.2).
- The legacy `withdraw`, `l2TokenAddress`, and `l1TokenAddress` functions should be defined in the `IL2BridgeLegacy` interface. However, instead of the new `withdraw` function, the legacy `withdraw` function has been defined in the `IL2Bridge` interface.
- The legacy `getL1TokenAddress` function is not defined in `ILegacyL2SharedBridge`.
- The `baseTokenAssetId` Diamond Proxy storage variable has no `Getters` facet function.

- The `chainBalance` public variable is missing a function in the `IL1NativeTokenVault` interface.
- The `depositAmount` function should be defined as a `view` function.

Consider correcting the above interface mismatches to improve the code clarity.

Update: *Resolved in pull request #633 at commit f80c0fb.*

N-12 Code Quality and Readability Suggestions

The following opportunities to improve the codebase were identified:

- The `bridgehubDepositBaseToken` is marked as `virtual`. If there is no intention of overriding it, consider removing the `virtual` attribute.
- The `_assetId` argument of the `bridgehubDepositBaseToken` function could, in theory, be an address padded to 32 bytes and handled by `_getAssetProperties` which returns the computed `assetId`. However, the original padded `_assetId` argument is emitted. Consider replacing the argument to emit `assetId`. In addition, if supporting `bytes32` arguments with padded addresses is desired and to avoid such issues, consider renaming the argument to clarify this possibility (e.g., to `_assetIdOrLegacyAddr`).
- A `try/catch` around `this.handleLegacyData` is used to check if the data could be decoded in the legacy format. However, calls to `this.handleLegacyData` can revert for reasons other than unsuccessful decoding (e.g., if the token address is WETH, or does not have code). If support for a legacy data format is desired, consider replacing the `handleLegacyData` function with a `decodeLegacyData` `external` function that only handles the decoding of the parameters. The rest of the function handling the data could be done in an `internal` function called in the success branch of the `try/catch` to bubble up errors.
- The `finalizeDeposit` has a commented-out `onlyBridge` modifier which could be removed.
- The order of functions within the contracts does not adhere to the Solidity Style Guide.
- The `amount` and `l1Receiver` stack variables could be only declared in the scope they are used in, while the commented-out variables `l1ReceiverBytes` and `parsedL1Receiver` can be removed.
- There is an unspecified `//todo` in the `_parseL2WithdrawalMessage` function.
- The `FinalizeDepositSharedBridge` and `WithdrawalInitiatedSharedBridge` events emit the hash of asset data as one of their parameters. This does not appear to be very useful as the pre-image is rather

interesting as information compared to its hash. Consider emitting the pre-image `bytes` data.
- The parameters of the `L2TokenBeaconUpdated` event could be indexed as they are used for address derivation in new token deployments.

Consider applying the aforementioned suggestions to improve the clarity of the codebase.

Update: *Resolved in pull request #640 at commit 8cec7bb. The second bullet point was addressed with N-07. Regarding the `_handleLegacyData` function, an encoding version byte is now prefixing the `_data` parameter as differentiation for non-legacy deposits. The `_handleLegacyData` function now has `internal` visibility and is regularly invoked.*

N-13 Lack of Input Validation on Emitted Data

The `_depositSender` argument of the `bridgeRecoverFailedTransfer` function is user-controlled and not validated by the function. The function decodes and validates `prevMsgSender` which plays a similar role but does not validate `_depositSender`. This could cause the wrong information to be emitted in the `ClaimedFailedDepositSharedBridge` event.

Consider removing `_depositSender` from the interface and replacing it with `prevMsgSender` in the event parameters.

Update: *Resolved in pull request #619 at commit 16241e4. The `_depositSender` argument is now used to check the transaction data hash and is forwarded to the `bridgeRecoverFailedTransfer` function of the asset handler to receive the funds.*

N-14 Padded Token Address as Asset ID Is Registered Redundantly on Base Token Deposits

The `_getAssetProperties` function can be provided an `address` padded to 32 bytes as its `_assetId` argument for backwards compatibility. In this case, `assetId` is computed as `keccak256(abi.encode(block.chainid, NATIVE_TOKEN_VAULT_VIRTUAL_ADDRESS, _assetId))`.

However, when querying the asset handler address for such assets, the padded address is used in place of the `assetId`. This means that the asset handler would always be 0 and the token would be re-registered on each bridging transaction.

Consider replacing `_assetId` by `assetId` when fetching the asset handler address. This will help avoid re-registering the asset and save gas.

Update: *Resolved in pull request #639 at commit d41438d. The code was simplified by removing the `_getAssetProperties` function.*

Conclusion

The audited code updates the ZKsync bridge contracts by adding support for custom asset handlers. Such handlers can be used to support more token types with the addition of custom logic on mints and burns. A canonical asset handler supporting ETH and simple ERC-20 tokens, called the native token vault, was also added.

We commend the Matter Labs team on their effort to ensure backwards compatibility with previous interfaces and integrations, but note that it adds significant complexity to the codebase. The high-level design of the code seems well thought out, though we found issues which could have been caught by additional tests. We also made multiple recommendations to enhance the quality of the codebase.

We thank the Matter Labs team for their responsiveness and for providing us with documentation explaining the changes introduced by the update.