Audittens

ZKsync Gateway Audit

December 19, 2024

Contents

1 About Auditens

Auditens is an audit company with expertise across various sectors of Web3 security.

What truly sets us apart is that our team consists entirely of top participants in mathematics and competitive programming competitions, uniquely equipping us to rapidly analyze codebases, uncover explicit and implicit invariants, and identify unique attack vectors.

You can subscribe and learn more about us at https://x.com/Auditens.

2 About ZKsync

ZKsync is a Layer-2 protocol that scales Ethereum with cutting-edge ZK tech. Their mission is not only to merely increase Ethereum's throughput, but to fully preserve its foundational values — freedom, self-sovereignty, decentralization — at scale.

ZK-rollups like Era are the only scaling solution that can inherit 100% of Ethereum's security. But theory is not enough. ZKsync is committed to go above and beyond to make Era by far the most secure L2, in practice.

You can subscribe and learn more about the project at https://zksync.io.

3 Risk classification

The severities of the issues are determined based on the following properties:

- Critical — results in a significant loss of assets within the protocol, a major deviation from the expected behavior of protocol components, and/or a violation of key security invariants.
- High — causes loss of assets within the protocol and/or general violations of protocol security invariants, with limitations on the variety of potential attacks.
- Medium — enables barely profitable attacks on the protocol, violations of security invariants that pose minimal risk to protocol users, and/or functionality limitations affecting only a relatively small subset of users.
- Low — enables griefing attacks on the protocol, unexpected changes in the protocol's behavior that are imperceptible, and/or functionality limitations with negligible impact on the security of the protocol.
- Informational — issues that have no practical impact on the security of the current version of the protocol but could potentially lead to more serious consequences in future updates, create the possibility for incorrect use of the protocol by users, and/or result in a significant decline in code quality, making maintenance more challenging.

While determining the severity of issues, the constraints required for successful attacks and the potential actions to address their consequences are taken into account in the decision-making process.

4 Executive summary

4.1 Engagement overview

ZKsync Security Council Foundation engaged Audittens to review the security of ZKsync's bridging and chain migration protocol. From September 23, 2024, to November 29, 2024, a team of five security researchers audited the provided source code. After the completion of the fix period by ZKsync's team, the full audit was finalized by reviewing the corresponding commits.

4.2 Scope

The `matter-labs/era-contracts` repository at the 7198b54fbcba37aa7a1dd75fc3067391af33e03e commit was audited.

The following contracts were within the scope of the audit:

```
matter-labs
   era-contracts
   ├── da-contracts
   │   └── contracts
   │       ├── CalldataDA.sol
   │       ├── DAContractsErrors.sol
   │       └── RollupL1DAValidator.sol
   ├── l1-contracts
   │   └── contracts
   │       ├── bridge
   │       │   ├── BridgeHelper.sol
   │       │   ├── BridgedStandardERC20.sol
   │       │   ├── L1ERC20Bridge.sol
   │       │   ├── L1Nullifier.sol
   │       │   ├── L2SharedBridgeLegacy.sol
   │       │   ├── L2WrappedBaseToken.sol
   │       │   ├── asset-router
   │       │   │   ├── AssetRouterBase.sol
   │       │   │   ├── L1AssetRouter.sol
   │       │   │   └── L2AssetRouter.sol
   │       │   └── ntv
   │       │       ├── L1NativeTokenVault.sol
   │       │       ├── L2NativeTokenVault.sol
   │       │       └── NativeTokenVault.sol
   │       ├── bridgehub
   │       │   ├── Bridgehub.sol
   │       │   ├── CTMDeploymentTracker.sol
   │       │   └── MessageRoot.sol
   │       ├── common
   │       │   └── libraries
   │       │       └── DataEncoding.sol
   │       ├── governance
   │       │   ├── AccessControlRestriction.sol
   │       │   ├── ChainAdmin.sol
   │       │   ├── Common.sol
   │       │   ├── L2AdminFactory.sol
   │       │   ├── L2ProxyAdminDeployer.sol
   │       │   ├── PermanentRestriction.sol
   │       │   ├── TransitionaryOwner.sol
   │       │   └── restriction
   │       │       ├── Restriction.sol
   │       │       └── RestrictionValidator.sol
   │       └── state-transition
   │           ├── ChainTypeManager.sol
   │           ├── ValidatorTimelock.sol
   │           ├── chain-deps
   │           │   ├── DiamondInit.sol
   │           │   ├── ZKChainStorage.sol
   │           │   └── facets
```

```
    │   │   │   ├── Admin.sol
    │   │   │   ├── Executor.sol
    │   │   │   ├── Getters.sol
    │   │   │   ├── Mailbox.sol
    │   │   │   └── ZKChainBase.sol
    │   │   ├── data-availability
    │   │   │   ├── CalldataDA.sol
    │   │   │   ├── CalldataDAGateway.sol
    │   │   │   ├── RelayedSLDAValidator.sol
    │   │   │   └── ValidiumL1DAValidator.sol
    │   │   └── libraries
    │   │       ├── BatchDecoder.sol
    │   │       └── PriorityTree.sol
    │   └── l2-contracts
    │       └── contracts
    │           └── data-availability
    │               ├── DAErrors.sol
    │               ├── RollupL2DAValidator.sol
    │               ├── StateDiffL2DAValidator.sol
    │               └── ValidiumL2DAValidator.sol
```

Additionally, the differences of the files in the scope of the audit between the 7198b54f and a5754174 commits were separately briefly reviewed.

4.3 Summary of findings

Severity \ Status	Acknowledged	Partially fixed	Fixed	Total
Critical	0	0	1	1
High	0	0	7	7
Medium	0	0	3	3
Low	2	1	8	11
Informational	8	0	32	40
Total	10	1	51	62

Table 1. Distribution of found issues.

5 Assumptions and limitations

5.1 Explicit invariants of the codebase usage

The audit was conducted assuming that:

1. All new contracts on `L1` will be deployed and configured before the upgrade of the existing contracts. In particular, `L1NativeTokenVault.registerEthToken`, `L1AssetRouter.setNativeTokenVault` and `MessageRoot.initialize` will be called before the current `Bridgehub` will be upgraded.
2. All existing contracts on `L1` will be upgraded atomically in one multicall. In the same multicall all necessary initialization functions will be called (in particular, `Bridgehub.initializeV2` and `Bridgehub.setAddresses`).

5.2 Limitations

The codebase was audited as is. Any subsequent changes may introduce new vulnerabilities and require a separate audit. Fixes for all findings have been reviewed within the limited scope (in the context of only relevant protocol components).

6 Findings

6.1 Critical severity findings

6.1.1 C-01. When chain migrates to settlement layer, its priority queue there is empty but `priorityTree.startIndex` is non-zero

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Admin.sol:329
l1-contracts/contracts/state-transition/libraries/PriorityTree.sol:87
l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:573
l1-contracts/contracts/state-transition/chain-deps/facets/Executor.sol:444
l1-contracts/contracts/state-transition/chain-deps/facets/Admin.sol:369

Description: Suppose there is a chain which used priority queue, and then switched to priority tree. Its priority queue contains transactions with indices from `0` to `priorityTree.startIndex - 1` (or more), and priority tree contains transactions with indices from `priorityTree.startIndex` to `priorityTree.startIndex + priorityTree.tree._nextLeafIndex - 1`.

When this chain migrates to settlement layer, its state is copied. The priority tree is copied too, including `priorityTree.startIndex`, but priority queue on settlement layer is left empty.

Many checks in the codebase depend on `priorityQueue.getFirstUnprocessedPriorityTx` which will be equal to `0` on settlement layer. Until priority queue length reaches `priorityTree.startIndex`, the following will be happening on settlment layer:

- `Mailbox.sol:573` — mailbox will store transactions in both priority queue and priority tree (but these transactions have different indices);
- `Executor.sol:444` — executor will execute transactions from priority queue;
- `Admin.sol:369` — migration back will not be possible.

Therefore, transaction which is requested once on `L1` will be executed twice on `L2` (the first time when transaction is requested and the second time when priority queue length reaches `priorityTree.startIndex`). The attacker can abuse this by depositing assets: on `L1`, token will be transfered from the attacker to bridge only once but it will be minted on `L2` twice (once immediately when the transaction from `priorityQueue` is processed, and once later when the same transaction from `priorityTree` is processed). The attacker can immediately withdraw asset (not waiting for the second mint) and repeat this process many times to steal all chain's balance.

Another issue is that transactions which were already present in priority tree will have to wait for a long time until priority queue length reaches `priorityTree.startIndex`. Also, when priority queue length reaches `priorityTree.startIndex`, priority tree will contain many transactions, so future transactions will have to wait too.

Recommendation: After line `Admin.sol:329`, add the following statements:

```
s.priorityQueue.head = s.priorityTree.startIndex;
s.priorityQueue.tail = s.priorityTree.startIndex;
```

ZKsync: *Fixed* in the `4039eab5` commit.

Audittens: *Approved*.

6.2 High severity findings

6.2.1 H-01. Some earliest transactions from priority tree will be executed twice

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:572-582
l1-contracts/contracts/state-transition/chain-deps/facets/Executor.sol:443-453,387,349,419

Description: If chain used priority queue, and then introduces priority tree, it goes into an intermediate state until all transactions requested before priority tree introducing are processed.

When user requests new priority transaction in this state, this transaction is pushed to both queue and tree.

`ExecutorFacet.executeBatchesSharedBridge` executes each batch one by one. For each batch: while chain is in intermediate state, it calls one overload of `_executeOneBatch` (which works only with `priorityQueue`); as soon as chain comes out of the intermediate state, it calls another overload of `_executeOneBatch` (which works only with `priorityTree`).

Likely, there will be also "mixed" batch which contains both some latest transactions from priority queue and some earliest transactions from priority tree. When `ExecutorFacet.executeBatchesSharedBridge` processes this batch, it will call the former overload of `_executeOneBatch` (which works only with `priorityQueue`), so all these transactions will be processed from the queue's point of view, but none of them will be processed from tree's point of view. When the function will process the next batch, it will call the latter overload of `_executeOneBatch` (which works only with `priorityTree`) but since some earliest transactions from priority tree are not yet processed (from the priority tree's point of view), they'll have to be processed again.

Recommendation: After line `_executeOneBatch(batchesData[i], i);`, add the statement:

```
if(s.priorityTree.startIndex < s.priorityQueue.getFirstUnprocessedPriorityTx())
    s.priorityTree.unprocessedIndex =
        s.priorityQueue.getFirstUnprocessedPriorityTx() - s.priorityTree.startIndex;
```

ZKsync: *Fixed* in the 519729e6 commit.

Audittens: *Approved*.

6.2.2 H-02. Failed transfer recovering is possible through different asset ids

Context:
l1-contracts/contracts/bridge/L1Nullifier.sol:385,481
l1-contracts/contracts/common/libraries/DataEncoding.sol:108-111,134

Description: Function `_isLegacyTxDataHash` does not check that the provided `_assetId` was used for the deposit. Instead, it checks that the corresponding `tokenAddress` from `L1NativeTokenVault` was used for the deposit. It means, that different `_assetId` that corresponds to the same `tokenAddress` can be used for the successful verification of failed transfers. It can be exploited by a malicious attacker to manipulate chain balances and permanently freeze other users' funds.

Attack scenario: Imagine there is a native token on `L2A` (let `ID1` be the corresponding asset id for it) held by two users: first user holds `100 ETH` of `ID1` and the second — `10 ETH` of `ID1`. The first (regular) user does the following:

1. Withdraws `100 ETH` of `ID1` to `L1`. This creates a bridged token on `L1` (let `T1` be the address at which it's deployed). After that, `L1NTV.originChainId[ID1] = L2A`, `L1NTV.tokenAddress[ID1] = T1`, `L1NTV.assetId[T1] = ID1`.
2. Deposits `100 ETH` of `ID1` to some other chain `L2B`. Since the token is deposited to non-origin chain, `L1NTV.chainBalance[L2B][ID1]` is increased to `100 ETH` by NativeTokenVault.sol:209.

Now the second user (the attacker) does the following to permanently freeze the first user's funds:

1. Withdraws `10 ETH` of `ID1` to `L1`.
2. Registers `T1` by NativeTokenVault.sol:86. It registers a new asset id `ID2`, assigns `L1NTV.tokenAddress[ID2] = T1` and reassigns `L1NTV.assetId[T1] = ID2`.

3. Initiates deposit of `10 ETH` of `T1` to `L2B` with provided zero gas limit at Bridgehub.sol:557 and legacy-encoded data. Since `LEGACY_ENCODING_VERSION` is being used, `assetId = L1NTV.assetId[T1] = ID2` will be deposited (L1AssetRouter.sol:253,376,387). Since the token is deposited to non-origin chain, `L1NTV.chainBalance[L2B][ID2]` is increased to `10 ETH` by NativeTokenVault.sol:274. Legacy `txDataHash` is calculated at L1AssetRouter.sol:271 and later stored in L1Nullifier.sol:280.
4. Calls `bridgeRecoverFailedTransfer` with wrong `_assetId = ID1` (L1Nullifier.sol:310) after deposit fails on `L2B`. Since `L1NTV.tokenAddress[ID1] = L1NTV.tokenAddress[ID2]`, `_isLegacyTxDataHash` check at L1Nullifier.sol:385 returns true, and `L1NTV` is forced to mint `10 ETH` (NativeTokenVault.sol:132) of `T1` to the attacker. Besides minting `T1`, `L1NTV` also decreases `L1NTV.chainBalance[L2B][ID1]` by `10 ETH`.
5. Repeats steps 3-4 nine more times.

As a result of the operations above, the attacker manipulated chain balances almost at no cost (only paying for gas), making `L1NTV.chainBalance[L2B][ID1] = 0` and `L1NTV.chainBalance[L2B][ID2] = 100 ETH`, while in fact the only deposited asset to `L2B` is `ID1`. Since `ID1` is a non-native token on `L2B`, the first regular user can try to withdraw it only with new message format (L2AssetRouter.sol:174) which includes the correct `_assetId = ID1` into withdrawal message. Therefore when finalizing withdrawal on `L1`, the first user can use only `assetId = ID1` as well (L1Nullifier.sol:620), but since `L1NTV.chainBalance[L2B][ID1] = 0`, he will not be able to finalize his withdrawal, resulting in permanent loss of funds (both on `L2B` and on `L1`).

Nobody, including the attacker, will be able to manipulate chain balances back in a similar way (by depositinig `ID1` and recovering failed transfer of `ID2`). It's because at step 3, `LEGACY_ENCODING_VERSION` has to be used, which allows to deposit only `assetId = L1NTV.assetId[T1] = ID2`.

Recommendation: It's possible to fix the issue in two different ways depending on the project's needs:

1. If the functionality of depositing non-native assets from `L1` to `L2` through `LEGACY_ENCODING_VERSION` should exist, it's enough to forbid native registration of the bridged tokens (NativeTokenVault.sol:340) by adding the require statement `require(assetId[_nativeToken] == 0);`.
   However, such an approach limits the functionality of using bridged tokens as separate native assets, and what is more important creates an implicit invariant for the `_isLegacyTxDataHash` function that one `tokenAddress` can always correspond to at most one asset id. Such an invariant should always be taken into account during future upgrades.
2. Otherwise, such deposits can be forbidden by adding the following line after DataEncoding.sol:109: `require(_assetId == encodeNTVAssetId(block.chainid, tokenAddress));`.
   Such fix will automatically explicitly guarantee that only one exact `_assetId` corresponding to the `L1`-native token can succefully pass verification at `_isLegacyTxDataHash` function (L1Nullifier.sol:481).

ZKsync: *Fixed* in the 7673edf9 commit.

Audittens: *Approved*.

6.2.3 H-03. `Bridgehub.setLegacyChainAddress` doesn't register legacy chain completely

Context:
l1-contracts/contracts/bridgehub/Bridgehub.sol:229-255,384-391

Description: To create new chain, `Bridgehub.createNewChain` does the following steps:

- set `chainTypeManager[_chainId]`,
- set `baseTokenAssetId[_chainId]`,
- set `settlementLayer[_chainId]`,
- call `CTM.createNewChain(...)`,
- set `zkChainMap[_chainId]`,
- call `messageRoot.addNewChain(_chainId)`.

If the chain is already registered before the currently reviewed version of the code is introduced, its `chainTypeManager[_chainId]` will already be set, and `CTM.createNewChain(...)` will already have been

called. To set `baseTokenAssetId[_chainId]`, anyone can call `Bridgehub.setLegacyBaseTokenAssetId`. To set `zkChainMap[_chainId]`, anyone can call `Bridgehub.setLegacyChainAddress`. However, there is no way to set `settlementLayer[_chainId]` and call `messageRoot.addNewChain(_chainId)`. Without these steps, chain cannot function properly:

- without `settlementLayer[_chainId]`, it cannot be migrated (`Bridgehub.sol:700`);
- without `messageRoot.addNewChain(_chainId)`, new batches cannot be executed (`Executor.sol:397`).

Recommendation: Instead of separate functions `Bridgehub.setLegacyBaseTokenAssetId` and `Bridgehub.setLegacyChainAddress`, it would be easier to create a single function `Bridgehub.registerLegacyChain` which does all steps together:

- set `baseTokenAssetId[_chainId]`,
- set `settlementLayer[_chainId]`,
- set `zkChainMap[_chainId]`,
- call `messageRoot.addNewChain(_chainId)`.

ZKsync: *Fixed* in the `a11df6a1` commit.

Audittens: *Approved*.

6.2.4 H-04. Failed `ETH` deposits with legacy `_data` are not recoverable

Context:
`l1-contracts/contracts/bridge/asset-router/L1AssetRouter.sol:276`
`l1-contracts/contracts/bridge/ntv/NativeTokenVault.sol:259-263`

Description: Currently deployed `L1SharedBridge` requires a user to provide `_depositAmount == 0` to deposit `msg.value` into the non-`ETH` based chain. For backward compatibility, zero `_depositAmount` is also accepted by the new `NativeTokenVault.sol:262` and is substituted with the `msg.value`. However, when calculating `txDataHash`, `_depositAmount` is not being substituted with the correct value by `L1AssetRouter.sol:271-277`.

Therefore, in case such `ETH` deposit to the non-`ETH`-based chain fails on `L2`, it will be impossible to use the recovery mechanism to restore the funds on `L1`.

Recommendation: The easiest option would be to remove the reassignment of `_depositAmount` in the `NativeTokenVault.sol:259-263`. However, if backward compatibility has to be preserved, extra logic should be added to the `L1AssetRouter.sol` while calculating the `txDataHash`. For this `transferData` has to be additionally parsed and replaced with the correct data for the special case of depositing `ETH` through the `NativeTokenVault`.

ZKsync: *Fixed* in the `4f1dde90` commit.

Audittens: *Approved*.

6.2.5 H-05. `MailboxFacet._proveL2LeafInclusion` doesn't allow to prove log if the chain has exactly one batch on settlement layer

Context:
`l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:226-228,247`
`l1-contracts/contracts/common/libraries/Merkle.sol:52,121-123`

Description: In `MessageRoot`, each chain is represented as a root of a Merkle tree whose leaves are chain's batch hashes. If some chain on some settlement layer has exactly one batch, then its Merkle tree consists of exactly one node which is leaf and root at the same time. Merkle proof of this leaf is empty. However, function `MailboxFacet._proveL2LeafInclusion` forbids empty Merkle proof of batch leaf in two places: directly (`if (batchLeafProofLen == 0) { assume that root is stored here on L1 }`) and through `Merkle.calculateRootMemory` call.

Therefore, it will be impossible to prove logs sent in this batch until the second batch will be executed and sent to `L1`.

Also, malicious admin can execute one batch on settlement layer and then migrate chain back to L1. In such case, it will never be possible to prove logs sent in this batch.

Recommendation: In the first element of proof (where logLeafProofLen, batchLeafProofLen are encoded) encode also additional flag which distinguishes two scenarios — "Merkle proof is empty" or "root is stored here on L1". Also do not revert in Merkle._validatePathLengthForSingleProof in case _pathLength == 0.

ZKsync: *Fixed* in the dca7b558 commit.

Audittens: *Approved*.

6.2.6 H-06. NTV.assetId records can be overwritten

Context:
l1-contracts/contracts/bridge/ntv/NativeTokenVault.sol:56,340
l1-contracts/contracts/bridge/ntv/L2NativeTokenVault.sol:91
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:155

Description: Functions NTV.registerToken and L2NTV.setLegacyTokenAssetId allow to overwrite existing records of assetId mapping. It has the following impacts:

1. When some L2-native token (let ID1 be the corresponding asset id for it) is being withdrawn to L1 for the first time, bridged token T is being deployed on L1 and NativeTokenVault.sol:421 stores L1NTV.assetId[T] = ID1. It allows to use LEGACY_ENCODING_VERSION to deposit asset ID1 to other chains (both native L2 and not), and use legacy recovery mechanism claimFailedDeposit for failed deposits of asset ID1 (L1Nullifier.sol:651). However, at any time anyone can call NTV.registerToken, which overwrites L1NTV.assetId[T] with some other ID2 (which is native for L1). After that, using LEGACY_ENCODING_VERSION/claimFailedDeposit for depositing/recovering of token T will operate with new asset ID2. It leads to:
   - In short time after overwriting L1NTV.assetId[T], when regular users (A) try to call claimFailedDeposit, it'll revert due to zero L1NTV.chainBalance[L2][ID2].
   - In short time after overwriting L1NTV.assetId[T], when regular users (B) try to deposit token T, thinking that it still corresponds to asset ID1, it'll be deposited as ID2 and L1NTV.chainBalance[L2][ID2] will be increased.
   - After previous step, users (A) can successfully call claimFailedDeposit, since L1NTV.chainBalance[L2][ID2] is now increased by users (B).
   - After previous step, users (B) will realize that they deposited wrong asset ID2, but will not be able to withdraw it due to the decreased L1NTV.chainBalance[L2][ID2] by users (A). Therefore funds of users (B) remain permanently frozen, since all their withdrawals from L2 will be parsed on L1 (L1Nullifier.sol:607,620) as asset ID2 (not ID1, for which chainBalance exists).
2. On L2 when some native token T exists, it can be withdrawn using L2AssetRouter.withdrawToken function. However, at any time anyone can call L2NTV.setLegacyTokenAssetId(T), which overwrites L2NTV.assetId[T] with non-existing assetId and therefore blocks use of L2AssetRouter.withdrawToken until NTV.registerToken(T) is manually called. By constantly calling L2NTV.setLegacyTokenAssetId(T), griefers can prevent regular users from use of L2AssetRouter.withdrawToken function.

Recommendation: It's possible to fix the first impact of issue in two different ways depending on the project's needs:

1. If the functionality of depositing non-native assets from L1 to L2 through LEGACY_ENCODING_VERSION should exist, it's enough to forbid native registration of the bridged tokens (NativeTokenVault.sol:340) by adding the require statement require(assetId[_nativeToken] == 0);.
   However, such an approach limits the functionality of using bridged tokens as separate native assets.
2. Otherwise, it's better to remove assetId from the NTV's storage and instead compute it everywhere as a native asset id for the current chain. Note, that it'll not break the backward compatibility, since the on-chain codebase doesn't support non-native tokens at all, and the currently reviewed codebase doesn't support depositing non-native assets through LEGACY_ENCODING_VERSION as soon as anyone calls NTV.registerToken (which can be done immediately after token's contract deployment).

To fix the second impact of the issue, the following has to be done:

1. `L2NTV.setLegacyTokenAssetId` function should check that it's being called when `L2_LEGACY_SHARED_BRIDGE` exists and stores information about `_l2TokenAddress`: `require(L2_LEGACY_SHARED_BRIDGE) != address(0)); require(l1TokenAddress != address(0));`.
2. `L2NTV._registerToken` function should check that `_nativeToken` doesn't correspond to the legacy bridged token, not yet registered by the `setLegacyTokenAssetId`: `require(L2_LEGACY_SHARED_BRIDGE == address(0) || L2_LEGACY_SHARED_BRIDGE.l1TokenAddress(_nativeToken) == address(0))`.

ZKsync: *Fixed* in the 7673edf9 and 2bcd5de9 commits.

Auditttens: *Approved*.

6.2.7 H-07. Legacy withdrawals on L2 can be exploited to steal users' funds

Context:
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:284,297,307,316
l1-contracts/contracts/bridge/L2SharedBridgeLegacy.sol:103-108

Description: The current `L2SharedBridge` implements the `withdraw` function that successfully works only with `_l2Token` deployed by the bridge itself. However, the new implementation can successfully work with the malicious `_l2Token` as well, which can be exploited by the attacker.

Attack scenario: Imagine some passive vault on `L2` that allows bridged tokens to be deposited into it and withdrawn to `L1` (where withdrawals are being processed internally via `L2SharedBridge.withdraw`). In such a setup, the attacker can steal all the funds from the vault by taking the following steps:

1. Deploy a token `T` on `L2` that implements the `l1Address` function that returns an address `A`. Here `A` is the address of some real token on `L1` that was bridged to `L2` and deposited to the vault above.
2. Mint a necessary amount of `T` for himself.
3. Deposit all minted amount of `T` into the vault.
4. Withdraw all deposited amount of `T` from the vault. During this step, the vault checks that the attacker indeed has deposited a corresponding amount of `T`, and calls `L2SharedBridgeLegacy.withdraw` with `_l2Token = T`. Since `T.l1Address() = A`, asset to withdraw corresponds to the token `A` (L2AssetRouter.sol:307). Therefore the vault withdraws all its balance of corresponding to token `A` asset directly to the attacker on `L1`.
5. Repeat steps 1-4 with all other tokens held by the vault.

Recommendation: Replace the implementation of the `L2AssetRouter.l1TokenAddress` with the following:

```
function l1TokenAddress(address _l2Token) public view returns (address) {
    require(L2_LEGACY_SHARED_BRIDGE != address(0));
    return IL2SharedBridgeLegacy(L2_LEGACY_SHARED_BRIDGE).l1TokenAddress(_l2Token);
}
```

ZKsync: *Fixed* in the a9a28a2e commit.

Auditttens: *Approved*.

6.3 Medium severity findings

6.3.1 M-01. WETH withdrawing and recovering is impossible

Context:
l1-contracts/contracts/bridge/ntv/NativeTokenVault.sol:92

Description: Currently deployed L1SharedBridge forbids to deposit WETH token, but allows to withdraw and recover it for old deposits done through L1ERC20Bridge. When L1SharedBridge was introduced, ∼ 17.9 WETH was transferred to it from L1ERC20Bridge. Since then, 19 successful withdrawals happened, resulting in ∼ 8.8 WETH remaining on the L1SharedBridge balance.

Since NativeTokenVault.sol:92 forbids to register WETH token, it not only forbids to deposit WETH token, but also forbids to withdraw and recover WETH for already existing deposits. Transferring liquidity of WETH from L1SharedBridge to L1NativeTokenVault is also impossible due to the necessity of token registration at L1NativeTokenVault.sol:99.

Therefore introducing NativeTokenVault leads to the permanent freezing of all WETH liquidity in the L1SharedBridge.

Recommendation: Instead of forbidding WETH token registration at NativeTokenVault.sol:92, forbid only WETH deposits by adding the corresponding restriction to the _bridgeBurnNativeToken function at NativeTokenVault.sol:255.

ZKsync: *Fixed* in the 9db7c17a commit.

Audittens: *Approved*.

6.3.2 M-02. Depositing through L1ERC20Bridge can spend funds twice

Context: l1-contracts/contracts/bridge/L1ERC20Bridge.sol:206-210

Description: L1ERC20Bridge.sol:197 uses L1_ASSET_ROUTER.depositLegacyErc20Bridge to handle deposits. To calculate _assetId, L1AssetRouter.sol:514 uses _ensureTokenRegisteredWithNTV function, which returns existing assetId from NTV if it's not zero. It allows to use L1ERC20Bridge for depositing non-native tokens.

However, when doing so, a user will spend funds twice:

1. At L1ERC20Bridge.sol:229 when transferring funds to L1ERC20Bridge.
2. At NativeTokenVault.sol:208 when burning funds by L1NTV.

For such a scenario, first half of the funds transferred to the L1ERC20Bridge will remain unused and permanently frozen there, since the approval to L1_ASSET_ROUTER is nullified at L1ERC20Bridge.sol:206-210.

Additionally, if the transaction later fails on L2, claimFailedDepositLegacyErc20Bridge will not work as well due to incorrectly calculated assetId at L1Nullifier.sol:708, assuming native token was deposited.

Recommendation: Since L1ERC20Bridge is not supposed to handle deposit operations with non-native tokens, change nullifying approval at L1ERC20Bridge.sol:206-210 with the require statement that enforces all the allowance has been consumed: `require(token.allowance(address(this), L1_ASSET_ROUTER) == 0);`.

ZKsync: *Fixed* in the 59d11f7a commit.

Audittens: *Approved*.

6.3.3 M-03. `L2AssetRouter._ensureTokenRegisteredWithNTV` always returns `0`

Context:
`l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:135-139,160`

Description: Function `L2AssetRouter._ensureTokenRegisteredWithNTV` returns `bytes32 assetId`, but this function doesn't contain return statement nor assignment to `assetId`, so it always returns `0`.

This function is used by `L2AssetRouter.withdrawToken` where its return value is used as asset id to be withdrawn. However, because of the mentioned bug, it will be equal to `0`, so withdrawal will fail.

Recommendation: Add assignment `assetId = nativeTokenVault.assetId(_token)` at the end of the `L2AssetRouter._ensureTokenRegisteredWithNTV` function.

ZKsync: *Fixed* in the `7663cfc5` commit.

Audittens: *Approved*.

6.4 Low severity findings

6.4.1 L-01. Whitelisting of settlement layers affects `MailboxFacet._proveL2LeafInclusion` of all chains

Context:
`l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:276`
`l1-contracts/contracts/bridgehub/Bridgehub.sol:299-308`

Description: `Bridgehub.registerSettlementLayer` is used to allow or disallow chains to settle on certain settlement layers. Also, `MailboxFacet._proveL2LeafInclusion` allows to prove leaf inclusion through any settlement layer whitelisted in Bridgehub. This means that `Bridgehub.registerSettlementLayer` has non-obvious consequences:

- when settlement layer is whitelisted, each chain is forced to trust this settlement layer (even chains which never settled on this settlement layer);
- when settlement layer is un-whitelisted, no chain can prove leaf inclusion through this settlement layer (even chains which settled on this settlement layer).

Recommendation: For each chain, store a mapping of used settlement layers, and allow proving only through them.

ZKsync: *Acknowledged*. Expected, since governance is expected to whitelist only chains with correct `CTM`, which assures correct tree building and so it is assumed that gateway can never produce malicious outputs (even if gateway admin is malicious). The proposed feature may be added in future versions if needed.

6.4.2 L-02. `L1Nullifier._isPreSharedBridgeDepositOnEra` behaviour is changed from on-chain version

Context:
`l1-contracts/contracts/bridge/L1Nullifier.sol:487-505`
`on-chain L1SharedBridge.sol:550`

Description: During upgrade, `L1Nullifier` contract will replace currently deployed `L1SharedBridge` (`proxy`, `implementation`). Function `L1SharedBridge._isEraLegacyDeposit` is renamed to `L1Nullifier._isPreSharedBridgeDepositOnEra`, its behaviour is kept the same except one check: "`_l2TxNumberInBatch < eraLegacyBridgeLastDepositTxNumber`" is replaced with "`_l2TxNumberInBatch <= eraLegacyBridgeLastDepositTxNumber`". So there is one transaction which was treated as non-legacy but will be treated as legacy.

Recommendation: Depending on whether this transaction is legacy or not, either keep new implementation or restore the old one. In the latter case, it would be better to change variables' names and/or comments describing them accordingly.

ZKsync: *Fixed* in the `4c27dc67` commit.

Audittens: *Approved*.

6.4.3 L-03. Allow to call `AdminFacet.setPubdataPricingMode` even afer the first batch

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Admin.sol:129-132

Description: Function `AdminFacet.setPubdataPricingMode` has a restriction that it can be called only when no batches are committed. Comment "Validium mode can be set only before the first batch is processed" makes sense. However, this function affects only pricing while data availability can be changed using function `AdminFacet.setDAValidatorPair`. Therefore, this restriction doesn't protect chain from becoming validium. On the other hand, if changing data availability using `setDAValidatorPair` is allowed for some chains, `setPubdataPricingMode` should be also allowed to make pricing correct.

Recommendation: Remove this restriction.

ZKsync: *Fixed* in the `fd7e4cd6` commit.

Audittens: *Approved*.

6.4.4 L-04. In `RollupL1DAValidator`, blob can be published long before it is really used

Context:
da-contracts/contracts/RollupL1DAValidator.sol:22,148

Description: `RollupL1DAValidator` allows to publish data using either calldata or blobs. Blobs are much cheaper because they are not completely available to EVM and are stored only for 4096 epochs (approximately 18 days). Also `RollupL1DAValidator` allows to publish blob in a separate transaction — anytime before batch is executed. This allows malicious validator to publish blob long before batch is executed (or even before chain is created) — at the time when batch is executed, this data won't be available to users.

Recommendation: In `RollupL1DAValidator`, replace `mapping(bytes32 blobCommitment => bool isPublished) public publishedBlobCommitments` with `mapping(bytes32 blobCommitment => uint256 publishBlockNumber) public publishedBlobCommitments`.
In function `publishBlobs`, set `publishedBlobCommitments[blobCommitment] = block.number`.
In function `checkDA`, check `require(block.number - publishedBlobCommitments[prepublishedCommitment] <= 4096 * 32 / 2, "not published")` so that one has some time (half of whole blob living time) to retrieve the blob.

ZKsync: *Fixed* in the `ae91ae43` commit.

Audittens: *Approved*.

6.4.5 L-05. `AdminFacet.forwardedBridgeRecoverFailedTransfer` has redundant check

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Admin.sol:356

Description: `AdminFacet.forwardedBridgeRecoverFailedTransfer` contains `require(_depositSender == s.admin, "Af: not chainAdmin")` which effectively checks that admin isn't changed during failed migration. This check doesn't prevent any issues. Instead, it just creates ability to accidentally lose chain.

Recommendation: Remove this `require` statement.

ZKsync: *Fixed* in the `2c1aadc9` commit.

Audittens: *Approved*.

6.4.6 L-06. Bridging system only partially supports L2-value

Context:
l1-contracts/contracts/bridgehub/IBridgehub.sol:26
l1-contracts/contracts/bridgehub/Bridgehub.sol:505,540,555
l1-contracts/contracts/bridge/asset-router/L1AssetRouter.sol:225,264
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:122,126
l1-contracts/contracts/bridge/asset-router/AssetRouterBase.sol:109,112,131,145
l1-contracts/contracts/bridge/interfaces/IAssetHandler.sol:28

Description: Bridging system on L1 is designed to support asset handlers which use L2-value:

- `Bridgehub.requestL2TransactionTwoBridges` takes parameter `_request.l2Value`, passes it to `IL1AssetRouter(_request.secondBridgeAddress).bridgehubDeposit` and later creates transaction with this L2-value;
- `L1AssetRouter.bridgehubDeposit` takes parameter `_value` and passes it to `IAssetHandler(l1AssetHandler).bridgeBurn` for verification;
- on L2, this L2-value is expected to be sent to `IAssetHandler.bridgeMint` which is payable.

However, such transaction will fail on L2 because `L2AssetRouter.finalizeDeposit` is not payable and doesn't pass `msg.value` to `IAssetHandler.bridgeMint` function calls.

Recommendation: Make function `L2AssetRouter.finalizeDeposit` payable and pass `msg.value` to `IAssetHandler.bridgeMint` function calls in function `AssetRouterBase._finalizeDeposit`.

ZKsync: *Fixed* in the ed7da5dc commit.

Audittens: *Approved*.

6.4.7 L-07. `PermanentRestriction.allowL2Admin` is not resistant to `CREATE2` collision

Context:
l1-contracts/contracts/governance/PermanentRestriction.sol:100-120
l1-contracts/contracts/governance/L2AdminFactory.sol:48

Description: Function `PermanentRestriction.allowL2Admin` is used to permissionlessly register any `ChainAdmin` on L2 if its address is derived from approved L2AdminFactory.

However, this registration is not resistant to `CREATE2` collision. User of `ChainAdmin` can find `salt1` and `salt2` such that `L2_ADMIN_FACTORY` with `salt1` deploys `ChainAdmin` to the same address as some malicious contract with `salt2` deploys another malicious contract. This requires computing approximately 2^{81} hashes. In this case it's possible to register such an address in `PermanentRestriction.allowL2Admin` and do anything with the chain on L2, which should be restricted by `PermanentRestriction`.

Recommendation: In `L2AdminFactory.deployAdmin`, either:

- use `CREATE` instead of `CREATE2` and modify `PermanentRestriction.allowL2Admin` accordingly, or
- restrict `_salt` to be small (e.g. `bytes4`) and remove ability to use `_additionalRestrictions`.

ZKsync: *Fixed* in the 02d6fbb2 commit.

Audittens: *Approved*.

6.4.8 L-08. `L2WrappedBaseToken.initializeV2` fails for already deployed contract

Context:
l1-contracts/contracts/bridge/L2WrappedBaseToken.sol:71
l2-contracts/contracts/bridge/L2WrappedBaseToken.sol:51 in old version

Description: Function `L2WrappedBaseToken.initializeV2` has to be called during upgrade process because new implementation introduces new variables `nativeTokenVault` and `baseTokenAssetId` and changes meaning of the old variable `l2Bridge`. However, this call will fail because of modifier `reinitializer(2)` since previous version was already initialized with `reinitializer(2)`.

Recommendation: Replace `reinitializer(2)` with `reinitializer(3)`.

ZKsync: *Fixed* in the aeade7e2 commit.

Audittens: *Approved*.

6.4.9 L-09. `L1AssetRouter` and `L1Nullifier` don't have some legacy functions for backward compatibility

Context:
l1-contracts/contracts/bridgehub/Bridgehub.sol:898-901

Description: During upgrade, `L1Nullifier` contract will replace currently deployed `L1SharedBridge`. On the other hand, `Bridgehub.sharedBridge` function (which returned address of `L1SharedBridge`) now returns address of `L1AssetRouter`. Therefore, contracts that used `L1SharedBridge` legacy interface may access either `L1Nullifier` (if they stored constant address) or `L1AssetRouter` (if they call `Bridgehub.sharedBridge` each time) and expect the same legacy interface. In fact, these new contracts implement almost all legacy functions, but:

- `L1AssetRouter` doesn't implement `depositHappened`;
- `L1Nullifier` doesn't implement `finalizeWithdrawal`, `L1_WETH_TOKEN` and bridging interface (`bridgehubDeposit`, `bridgehubConfirmL2Transaction`).

Recommendation: Implement all these functions.

ZKsync: *Partially fixed* in the 1982d826 commit. We decided to only add `finalizeWithdrawal` for now. Adding `bridgehubDeposit`-like methods would introduce too much complexity (e.g. in this case we would have to consider cases when users approve funds to `L1Nullifier` etc).

Audittens: *Approved*.

6.4.10 L-10. Settlement layer's admin can break system invariant

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:486-492,368-383

Description: Settlement layers system doesn't support situations when there is a chain `C4` which settles on settlement layer `SL3` which itself settles on settlement layer `SL2` which settles on `L1`. For example, in such scenario, requesting priority transactions from `L1` to `C4` doesn't work because `MailboxFacet.requestL2TransactionToGatewayMailbox` of `SL3` doesn't call `MailboxFacet.requestL2TransactionToGatewayMailbox` of `SL2`.

However, the code doesn't explicitly restrict this situation — malicious admin of `SL3` can migrate it to `SL2` to put the system into an unsupported state. There are two possible (independent) attack scenarios:

1. In such a case, if anyone sends priority transaction from `L1` to `C4`, it will be stored in priority queue of `C4`'s diamond proxy on `L1` but not in priority queue of `C4`'s diamond proxy on `SL3`. Then admin of `SL3` can migrate it back to `L1`. After that, new priority transactions will be stored in priority queues of both `C4`'s diamond proxies. However, the first transaction will forever stay in priority queue of `C4`'s diamond proxy on `L1` and will not allow `C4` to be migrated back to `L1` because of this check: `Admin.sol:315-320`.
2. If admin of `SL2` is malicious too, they can migrate `SL2` to `SL3` to create a loop where `SL3` settles on `SL2`, `SL2`

settles on SL3 and this structure is completely detached from L1. In such case, it will be impossible to migrate any SL to L1 to return back to the correct state because this would require sending message from other SL to L1 which is impossible since none of them settles on L1.

Recommendation: Forbid to migrate a chain to SL if it's whitelisted as settlement layer. Forbid to whitelist a chain as settlement layer if it's already on SL.

ZKsync: *Fixed* in the dc549e25 commit.

Audittens: *Approved*.

6.4.11 L-11. Operator can force user to execute the same transaction twice

Note: This finding is out of scope of this audit.

Description: Consider the following scenario:

1. User requests transaction directly to L2.
2. Operator ignores user.
3. User is forced to request the same transaction from L1.
4. Operator executes both transactions.

This way operator can force user to execute the same transaction twice (which means e.g. transfering twice more funds than expected and pay for transaction execution twice). User cannot avoid this because priority transaction doesn't increment user's nonce.

Recommendation: Add optional field "user's nonce" to priority transaction which has to be validated on L2 under usual rules of nonce management.

ZKsync: *Acknowledged*. It is a known issue with the current system.

6.5 Informational findings

6.5.1 I-01. `ChainTypeManager.revertBatches` fails unless `ChainTypeManager` is manually set as validator

Context:
l1-contracts/contracts/state-transition/ChainTypeManager.sol:283
l1-contracts/contracts/state-transition/chain-deps/facets/Executor.sol:540

Description: `ChainTypeManager.revertBatches` calls `ExecutorFacet.revertBatchesSharedBridge` which is callable only by validator because of modifier `onlyValidator`. So this call will fail unless `ChainTypeManager` is manually set as validator.

Recommendation: Apply modifier `onlyValidatorOrChainTypeManager` instead of `onlyValidator` to function `ExecutorFacet.revertBatchesSharedBridge`.

ZKsync: *Fixed* in the fcca8de8 commit.

Audittens: *Approved*.

6.5.2 I-02. `ChainTypeManager.registerSettlementLayer` always fails

Context:
l1-contracts/contracts/state-transition/ChainTypeManager.sol:443
l1-contracts/contracts/bridgehub/Bridgehub.sol:305

Description: `ChainTypeManager.registerSettlementLayer` calls `Bridgehub.registerSettlementLayer` which is callable only by the owner because of modifier `onlyOwner`. So this call will always fail.

Recommendation: Either remove the function `ChainTypeManager.registerSettlementLayer` (since `Bridgehub.registerSettlementLayer` already provides the same functionality), or update function `Bridgehub.registerSettlementLayer` to allow calls both from owner and registered `CTM`s.

ZKsync: *Fixed* in the `f54121e6` commit.

Audittens: *Approved*.

6.5.3 I-03. `MailboxFacet.proveL1ToL2TransactionStatusViaGateway` function is empty

Context:
`l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:122-130`

Description: Function `MailboxFacet.proveL1ToL2TransactionStatusViaGateway` is empty.

Recommendation: Remove this function entirely since `MailboxFacet.proveL1ToL2TransactionStatus` already works for chains that settle on gateway.

ZKsync: *Fixed* in the `9016bd39` commit.

Audittens: *Approved*.

6.5.4 I-04. Wrong expected message length in `L1Nullifier._parseL2WithdrawalMessage`

Context:
`l1-contracts/contracts/bridge/L1Nullifier.sol:617`

Description: In `L1Nullifier._parseL2WithdrawalMessage`, in case when `bytes4(functionSignature) == IAssetRouterBase.finalizeDeposit.selector`, it's checked that `_l2ToL1message.length >= 36`. However, actually message length has to be at least `bytes4 selector + uint256 originChainId + bytes32 assetId = 4 + 32 + 32 = 68`.

Recommendation: Replace `36` with `68`.

ZKsync: *Fixed* in the `6bd1fb65` commit.

Audittens: *Approved*.

6.5.5 I-05. `NativeTokenVault.bridgeMint` emits event `BridgeMint` twice

Context:
`l1-contracts/contracts/bridge/ntv/NativeTokenVault.sol:109-164`

Description: `NativeTokenVault.bridgeMint` always calls one of the internal functions: either `_bridgeMintNativeToken` or `_bridgeMintBridgedToken`. Both of these functions emit event `BridgeMint`. Then `NativeTokenVault.bridgeMint` itself emits exactly the same event again. Therefore, this event is anyway emitted twice.

Recommendation: Remove emit either in `NativeTokenVault.bridgeMint`, or in both `_bridgeMintNativeToken` and `_bridgeMintBridgedToken`.

ZKsync: *Fixed* in the `db410881` commit.

Audittens: *Approved*.

6.5.6 I-06. `ERC20` getters of `L2BaseToken` always return information about Ether

Context:
system-contracts/contracts/L2BaseToken.sol:131-147

Description: `ERC20` getters (`name`, `symbol` and `decimals`) of `L2BaseToken` always return information about Ether (`Ether`, `ETH` and `18`). However, on chains where base token is not Ether, correct information about base token should be returned.

Recommendation: Information about the base token may be sent from `L1` and set to the variables at genesis upgrade together with `setChainId`.

ZKsync: *Acknowledged*. It is a known issue and these methods have instability warning comment above them, so we may delete them in future releases.

6.5.7 I-07. `IERC20.approve` doesn't work with certain tokens

Context:
`l1-contracts/contracts/bridge/L1ERC20Bridge.sol:207,230`

Description: It's better to use `SafeERC20.forceApprove` instead of `IERC20.approve` because some tokens don't return `bool success`.

Recommendation: Replace `approve` with `forceApprove`.

ZKsync: *Fixed* in the `5c11f85a` commit.

Audittens: *Approved*.

6.5.8 I-08. `AdminFacet.setPriorityTxMaxGasLimit` and `AdminFacet.setTokenMultiplier` should be `onlyL1`

Context:
`l1-contracts/contracts/state-transition/chain-deps/facets/Admin.sol:83,114`

Description: Variables `s.priorityTxMaxGasLimit`, `s.baseTokenGasPriceMultiplierNominator` and `s.baseTokenGasPriceMultiplierDenominator` are used only on `L1` (when priority transaction is requested), so it makes sense to set them only on `L1`.

Recommendation: Apply modifier `onlyL1` to functions `AdminFacet.setPriorityTxMaxGasLimit` and `AdminFacet.setTokenMultiplier`.

ZKsync: *Fixed* in the `f7364fb3` commit.

Audittens: *Approved*.

6.5.9 I-09. Unknown function selector is used in `isTransactionAllowed`

Context:
l1-contracts/contracts/transactionFilterer/GatewayTransactionFilterer.sol:94

Description: Function `GatewayTransactionFilterer.isTransactionAllowed` checks that transaction comes from `L1AssetRouter` and has correct function selector. Selectors `IL2AssetRouter.setAssetHandlerAddress` and `IAssetRouterBase.finalizeDeposit` are indeed functions of `L2AssetRouter`. However, the third checked selector `IL2Bridge.finalizeDeposit` is not implemented anywhere in the project. Therefore, it can be removed here.

Recommendation: Remove "`&& IL2Bridge.finalizeDeposit.selector != l2TxSelector`".

ZKsync: *Fixed* in the `19262a20` commit.

Audittens: *Approved*.

6.5.10 I-10. Approve in `L1AssetRouter.depositLegacyErc20Bridge` is redundant

Context:
l1-contracts/contracts/bridge/asset-router/L1AssetRouter.sol:515
l1-contracts/contracts/bridge/ntv/L1NativeTokenVault.sol:250-256

Description: Function `L1AssetRouter.depositLegacyErc20Bridge` calls `forceApprove` to grant allowance to the `nativeTokenVault`. Later, function `L1NativeTokenVault._depositFunds` checks if it has approval from `L1AssetRouter` — if so, it will transfer funds from `L1AssetRouter` instead of the original caller.

However, this process is redundant because `L1AssetRouter` should never have funds. Instead, during legacy deposit, `L1AssetRouter` transfers funds from legacy bridge directly to `L1NativeTokenVault`.

Recommendation: Remove line L1AssetRouter.sol:515 and function `L1NativeTokenVault._depositFunds`.

ZKsync: *Fixed* in the 3f2589c0 commit.

Audittens: *Approved*.

6.5.11 I-11. Several functions accept `msg.value` but don't use it

Context:
l1-contracts/contracts/bridge/asset-router/L1AssetRouter.sol:229,225,242-249
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:97
l1-contracts/contracts/bridge/ntv/NativeTokenVault.sol:181,183,177,118
l1-contracts/contracts/bridge/ntv/L1NativeTokenVault.sol:196
l1-contracts/contracts/bridgehub/Bridgehub.sol:695,736,773

Description:

1. `L1AssetRouter.sol:229,242-249` — `L1AssetRouter.bridgehubDeposit` is payable but "set asset handler counterpart" branch doesn't use `msg.value`;
2. `L1AssetRouter.sol:225,242-249` — in `L1AssetRouter.bridgehubDeposit` which accepts parameter `_value` (which is amount of base token to be sent to L2), "set asset handler counterpart" branch requests call to `L2AssetRouter.setAssetHandlerAddress` without checking that `_value` is non-zero but function `setAssetHandlerAddress` is non-payable (L2AssetRouter.sol:97);
3. `NativeTokenVault.sol:181,183` — `NativeTokenVault.bridgeBurn` is payable but "bridged token" branch doesn't use `msg.value`;
4. NativeTokenVault.sol:177, Bridgehub.sol:691 — `NativeTokenVault.bridgeBurn` and `Bridgehub.bridgeBurn` accept parameter `_msgValue` (which is amount of base token to be sent to `bridgeMint` function on the other chain) without checking that it's non-zero but corresponding `NativeTokenVault.bridgeMint` and `Bridgehub.bridgeMint` don't use `msg.value`;
5. NativeTokenVault.sol:118, L1NativeTokenVault.sol:196, Bridgehub.sol:695,736,773 — `NativeTokenVault.bridgeMint`, `L1NativeTokenVault.bridgeRecoverFailedTransfer`, `Bridgehub.bridgeBurn`, `Bridgehub.bridgeMint` and `Bridgehub.bridgeRecoverFailedTransfer` are payable but don't use `msg.value`.

In all these cases, if user accidentally sends `msg.value` / L2-value, they will lose it (with or without an ability to recover it).

Recommendation:
For items 1, 3 and 5, add statements `if (msg.value != 0) revert NonEmptyMsgValue()`.
For item 2, add statement `if (_value != 0) revert NonEmptyMsgValue()`.
For item 4, add statement `if (_msgValue != 0) revert NonEmptyMsgValue()`.

ZKsync: *Fixed* in the 2edde4c0 commit.

Audittens: *Approved*.

6.5.12 I-12. `L2AssetRouter.withdraw` and `L2AssetRouter.withdrawToken` are marked as legacy

Context:
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:141-162

Description: `L2AssetRouter.withdraw` and `L2AssetRouter.withdrawToken` are new functions that will be introduced in the upcoming upgrade. However, they are placed under the comment that describes the block of "*legacy functions*".

`L2AssetRouter.withdraw` is the only function in `L2AssetRouter` that can withdraw tokens. However, the comment above it says "*do not rely on this function, it will be deprecated in the future*".

According to these comments, currently contracts and applications cannot withdraw assets without risk of deprecation in future.

Recommendation: Move the comment that describes the block of "*legacy functions*" under the `L2AssetRouter.withdraw` and `L2AssetRouter.withdrawToken` functions. Remove the comment "*do not rely on this function, it will be deprecated in the future*" above the `L2AssetRouter.withdraw` function.

ZKsync: *Fixed* in the `4324fe65` commit.

Audittens: *Approved*.

6.5.13 I-13. `AccessControlRestriction.validateCall` may miss fallback role checks for invalid selectors

Context:
l1-contracts/contracts/governance/AccessControlRestriction.sol:73

Description: If the target contract doesn't have a function matching the given selector (for example, this function is removed in an upgrade) the `AccessControlRestriction.validateCall` function still checks `requiredRoles` instead of `requiredRolesForFallback`. This allows callers who have roles for the specific selector but lack the required fallback role to access the fallback function.

Recommendation: Ensure the consistency between roles granted for some selectors and actual selectors of the functions of the target contracts. Specifically, ensure that roles associated with outdated selectors are removed.

ZKsync: *Acknowledged*.

6.5.14 I-14. `L2AssetRouter.withdrawToken` supports both native tokens and tokens that came from another `L2`

Context:
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:157-159

Description: Function `L2AssetRouter.withdrawToken` reverts if token's origin chain is `L1`. This means that it supports both native tokens (origin chain is current `L2` or `0`) and tokens from other `L2`s. However, it seems that this function is supposed to support only native tokens.

Recommendation: Replace condition "`recordedOriginChainId == L1_CHAIN_ID`" with "`recordedOriginChainId != 0 && recordedOriginChainId != block.chainid`".

ZKsync: *Fixed* in the `201b7f05` commit.

Audittens: *Approved*.

6.5.15 I-15. `PermanentRestriction._validateRemoveRestriction` blocks self-calls with data shorter than 4 bytes

Context:
l1-contracts/contracts/governance/PermanentRestriction.sol:203

Description: Function `PermanentRestriction._validateRemoveRestriction` is designed to restrict calls to `removeRestriction` of the `ChainAdmin` who has this restriction. However, due to use of slice `_call.data[:4]`, this function will also revert when `_call.data` is shorter than 4 bytes. Currently this restricts only self-calls to `receive` but if `ChainAdmin` implementation will be extended in future, something else may become blocked too.

Recommendation: Before accessing slice, add one more check: "`if (_call.data.length < 4) return;`".

ZKsync: *Fixed* in the d335fe3b commit.

Audittens: *Approved*.

6.5.16 I-16. `BridgeHelper.getERC20Getters` doesn't check whether calls are successful

Context:
l1-contracts/contracts/bridge/BridgeHelper.sol:29-31
l1-contracts/contracts/bridge/BridgedStandardERC20.sol:117-127,135-140

Description: When token is bridged for the first time, function `BridgeHelper.getERC20Getters` calls `name`, `symbol` and `decimals` functions of this token to be later decoded on the other chain. `BridgeHelper.getERC20Getters` doesn't check success of these calls, so if some of them reverts, its revert data will be sent to the other chain. If this revert data will be accidentally parsed to the resulting type, bridged token's getter will return incorrect value instead of revert.

Recommendation: In `BridgeHelper.getERC20Getters`, check whether getter reverted — if so, replace its revert data with empty bytes.

ZKsync: *Fixed* in the 39ca05d6 commit.

Audittens: *Approved*.

6.5.17 I-17. `GettersFacet.isFunctionFreezable` and `GettersFacet.isFacetFreezable` are inconsistent

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Getters.sol:211-218,193-204

Description: Function `GettersFacet.isFunctionFreezable` reverts when called with non-registered selector. Function `GettersFacet.isFacetFreezable` returns `false` when called with non-registered facet. For consistency, either both of them should revert, or both of them should return `false`.

Recommendation: In `GettersFacet.isFacetFreezable`, revert if condition "`selectorsArrayLen != 0`" doesn't hold.

ZKsync: *Fixed* in the 4cab4c33 commit.

Audittens: *Approved*.

6.5.18 I-18. New `L3` → `L1` message proof is not completely backward-compatible

Context:
Nested L3 → L1 messages tree design for Gateway — Legacy support
system-contracts/contracts/L1Messenger.sol:309

Description: `L3` → `L1` messages were designed in a way to support legacy format of `L2` → `L1` logs proving: "*just provide a proof that assumes that stored* `settledMessageRoot` *is identical to local root, i.e. the hash of logs in the batch*".

This statement is not a correct description of real proof format. In real proof format, the stored root is `fullRootHash = keccak256(localLogsRootHash || aggregatedRootHash)` — it may be seen as a root of non-standard tree where different leaves have different depths, but definitely not as "the hash of logs in the batch".

Recommendation: If legacy support is necessary, send from `L1Messenger` and store in `s.l2LogsRootHashes` both roots — `fullRootHash` and `localLogsRootHash`. Otherwise, mention in documentation that there is only partial support of the legacy format.

ZKsync: *Acknowledged*. We will update documentation.

6.5.19 I-19. Priority operation expiration timestamp is not stored in priority tree

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:572-583

Description: When a priority operation is written to the priority tree, its expiration timestamp is ignored and not stored.

Recommendation: Store expiration timestamps corresponding to priority operations for use in future versions of the protocol.

ZKsync: *Acknowledged*. Expected, we will fix the issue when we'll introduce full censorship resistance.

6.5.20 I-20. Priority operation expiration should be increased for chains on settlement layer

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:480

Description: `PRIORITY_EXPIRATION` is the amount of time the validator has to process the priority transaction. If the chain settles on some settlement layer, the amount of time needed to process the priority transaction depends not only on chain's validator, but also on settlement layer's validator:

1. settlement layer's validator has `PRIORITY_EXPIRATION` time to execute `forwardTransactionOnGateway` transaction;
2. chain's validator has `PRIORITY_EXPIRATION` time to execute the initial transaction on settlement layer;
3. if settlement layer's validator ignores chain's validator, then chain's validator is forced to request a new transaction from `L1` to settlement layer with execution of the batch which contains the initial transaction, so settlement layer's validator has another `PRIORITY_EXPIRATION` time to execute this new transaction.

Therefore, priority operation expiration should be multiplied by 3.

Recommendation: replace "`_params.expirationTimestamp = uint64(block.timestamp + PRIORITY_EXPIRATION)`" with "`_params.expirationTimestamp = uint64(block.timestamp + PRIORITY_EXPIRATION * (s.settlementLayer == address(0) ? 1 : 3))`".

ZKsync: *Acknowledged*. Expected, we will fix the issue when we'll introduce full censorship resistance.

6.5.21 I-21. In `ChainTypeManager._deployNewChain`, condition "`getZKChain(_chainId) != address(0)`" is never satisfied

Context:
l1-contracts/contracts/state-transition/ChainTypeManager.sol:355-358
l1-contracts/contracts/bridgehub/Bridgehub.sol:363,370,752-753

Description: `ChainTypeManager._deployNewChain` is called from two places:

- from `ChainTypeManager.createNewChain` which is called by `Bridgehub.createNewChain` where `_validateChainParams` checked that chain doesn't exist (`Bridgehub.sol:363,370`);

- from `ChainTypeManager.forwardedBridgeMint` which is called by `Bridgehub.bridgeMint` only if chain doesn't exist (`Bridgehub.sol:752-753`).

Therefore, condition "`getZKChain(_chainId) != address(0)`" in `ChainTypeManager._deployNewChain` cannot be satisfied.

Recommendation: Either completely remove this condition, or (to be safe) revert if it's satisfied.

ZKsync: *Fixed* in the `68d55a94` commit.

Audittens: *Approved*.

6.5.22 I-22. System design doesn't allow to register custom asset handlers for L2-native assets

Context:
`l1-contracts/contracts/bridge/asset-router/L1AssetRouter.sol:140,152,166,241`
`l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:93,103`

Description: In `L1AssetRouter`, there are several functions that allow to use custom asset deployment trackers and asset handlers:

- `setAssetDeploymentTracker` — register deployment tracker for some asset on current chain;
- `setAssetHandlerAddressThisChain` — register handler for some asset which has deployment tracker on current chain;
- `_setAssetHandlerAddressOnCounterpart` in `bridgehubDeposit` — register deployment tracker on other chain for some asset which has deployment tracker on current chain.

`L2AssetRouter` can only:

- accept `_setAssetHandlerAddressOnCounterpart` requests from `L1`;
- `setAssetHandlerAddressThisChain` — register handler for some asset which has deployment tracker on current chain.

Therefore, it's impossible to create native asset on `L2` (except `L2NativeTokenVault` which can be registered without `setAssetDeploymentTracker`).

Recommendation: Add missing functionality:

- `setAssetDeploymentTracker` on `L2`;
- `setAssetHandlerAddressOnCounterpart` on `L2` and ability to receive such message on `L1`;
- `setAssetDeploymentTrackerAddressOnCounterpart` on `L2` and ability to receive such message on `L1` — this is needed to be able to bridge custom `L2`-native asset from `L1` to another `L2`.

ZKsync: *Acknowledged*. Expected, it will be introduced in future releases.

6.5.23 I-23. Admin of a legacy chain can make `baseTokenAssetId` registered in `Bridgehub`

Context:
`l1-contracts/contracts/bridgehub/Bridgehub.sol:746-748`

Description: Function `Bridgehub.sol:732` assumes that `baseTokenAssetId` has been already registered in `Bridgehub` on `L1`. While this assumption is correct for new chains, for legacy chains it may not hold.

It allows the admins of legacy chains to register the corresponding `baseTokenAssetId` in `Bridgehub` on `L1` by migrating their chain to a new settlement layer and back. Therefore the functionality of asset ids registration (`Bridgehub.sol:290`) is available not only to the owner or admin of `Bridgehub`, but also partially available to the admins of legacy chains.

Recommendation: Since `assetIdIsRegistered` is used only on `L1`, remove redundant registration at `Bridgehub.sol:748`. Alternatively, if `assetIdIsRegistered` still has to be consistent, register the corresponding

`baseTokenAssetId` of legacy chains atomically inside `setLegacyBaseTokenAssetId` function. For this, the following line has to be added after `Bridgehub.sol:237`: `assetIdIsRegistered[baseTokenAssetId[_chainId]] = true`.

ZKsync: *Fixed* in the `a71a0491` commit.

Audittens: *Approved*.

6.5.24 I-24. Operator can force `Executor` to store `systemLogs` inconsistent with their hash

Context:
`l1-contracts/contracts/state-transition/chain-deps/facets/Executor.sol:91-100,156-157,615`

Description: `UnsafeBytes` library is being used for processing `systemLogs`. However, in case `emittedL2Logs.length % L2_TO_L1_LOG_SERIALIZE_SIZE != 0`, information about the last log will be partially read not from the `systemLogs`, but from the memory after it. In such a case, stored batch information at `Executor.sol:91-100` will not correspond to the hash at `Executor.sol:615` that is calculated for the batch commitment.

Recommendation: To guarantee consistency of system logs data with their hash and don't rely on the circuits' implementation, add the following line before processing `systemLogs`:
`require(emittedL2Logs.length % L2_TO_L1_LOG_SERIALIZE_SIZE == 0)`.

ZKsync: *Fixed* in the `924bf427` commit.

Audittens: *Approved*.

6.5.25 I-25. Factory deps are not checked during the creation of new chain

Context:
`l1-contracts/contracts/bridgehub/Bridgehub.sol:361,375`
`l1-contracts/contracts/state-transition/ChainTypeManager.sol:416-420,426`

Description: During the creation of a new chain, `CTM` checks that the provided `_forceDeploymentData` corresponds to the `initialForceDeploymentHash`. However, there's no similar check for the `_factoryDeps`. It allows the chain's creator to pass the wrong `_factoryDeps`, which will not allow execution of the genesis upgrade transaction. In such a case, the created chain will be unusable, and therefore other chain will have to be created, but with different not desired `chainId`.

Recommendation: Introduce new `initialFactoryDepsHash` variable and use it to validate the provided `_factoryDeps`: `require(keccak256(abi.encode(_factoryDeps)) == initialFactoryDepsHash);`.

ZKsync: *Acknowledged*, but we will not fix in this release.

6.5.26 I-26. `AssetRouterBase` uses an incorrect storage gap size

Context:
`l1-contracts/contracts/bridge/asset-router/AssetRouterBase.sol:53`

Description: According to the OpenZeppelin upgradeable contracts standard, the size of the `__gap` array is calculated so that the amount of storage slots used by a contract always adds up to the same number 50, but in the `AssetRouterBase` contract these add up to 49.

Recommendation: Change the `__gap` in the `AssetRouterBase` contract to `uint256[48]`.

ZKsync: *Fixed* in the `d947519c` commit.

Audittens: *Approved*.

6.5.27 I-27. Misleading comment in `ChainTypeManager.registerSettlementLayer` regarding settlement layer deployment

Context:
l1-contracts/contracts/state-transition/ChainTypeManager.sol:440-441

Description: The comment at ChainTypeManager.sol:440 states that it is required that the new settlement layer was deployed by the same `ChainTypeManager`, but the following check only ensures that it is registered in `Bridgehub`.

Recommendation: Ensure consistency between the mentioned comment and designed invariants of settlement layer registration.

ZKsync: *Fixed* in the f54121e6 commit.

Audittens: *Approved*.

6.5.28 I-28. Misleading use of `IBridgehub` in `abi.encodeCall`

Context:
l1-contracts/contracts/state-transition/chain-deps/facets/Mailbox.sol:401

Description: At Mailbox.sol:401, `IBridgehub(s.bridgehub).forwardTransactionOnGateway` is used as the selector for the call, while the actual target of the call is `L2_BRIDGEHUB_ADDR`, not the `s.bridgehub`.

Recommendation: Replace `IBridgehub(s.bridgehub).forwardTransactionOnGateway` with the `IBridgehub.forwardTransactionOnGateway`.

ZKsync: *Fixed* in the 96e3a9b8 commit.

Audittens: *Approved*.

6.5.29 I-29. Misleading variable names corresponding to `bridgehubData` in `Bridgehub` contract

Context:
l1-contracts/contracts/bridgehub/Bridgehub.sol:709,753,782,716,759,789

Description: It is confusing that `bridgehubData.ctmData` has different values in `bridgeBurn`, `bridgeRecoverFailedTransfer` and `bridgeMint` functions of the `Bridgehub` contract. The same happens with `bridgehubData.chainData`.

Recommendation: Rename the `bridgehubData` to `bridgehubBurnData` in `bridgeBurn` and `bridgeRecoverFailedTransfer` functions, and to `bridgehubMintData` in `bridgeMint` function.

ZKsync: *Fixed* in the 51363b9c commit.

Audittens: *Approved*.

6.5.30 I-30. Redundant check for `ASSET_ROUTER` in `L1NativeTokenVault.receive`

Context:
l1-contracts/contracts/bridge/ntv/L1NativeTokenVault.sol:72

Description: At L1NativeTokenVault.sol:72, the `L1NativeTokenVault.receive` function ensures that `msg.sender` is either `L1_NULLIFIER` or `ASSET_ROUTER`. But there is no way for `msg.sender` to be equal to `ASSET_ROUTER` in this context, making the check for `ASSET_ROUTER` redundant.

Recommendation: Remove the mentioned check.

ZKsync: *Fixed* in the 77ea5daa commit.

Audittens: *Approved*.

6.5.31 I-31. Usage of raw `keccak256` computation instead of `DataEncoding.encodeAssetId`

Context:
l1-contracts/contracts/bridge/asset-router/L1AssetRouter.sol:144-146
l1-contracts/contracts/bridgehub/Bridgehub.sol:334
l1-contracts/contracts/common/libraries/DataEncoding.sol:65-67

Description: At L1AssetRouter.sol:144-146 and Bridgehub.sol:334, the `assetId` is computed directly using `keccak256`. However, the function `DataEncoding.encodeAssetId` (DataEncoding.sol:65-67) is designed to provide such functionality.

Recommendation: Replace the direct computation with a call to `DataEncoding.encodeAssetId`.

ZKsync: *Fixed* in the 8409737f commit. It was already partially fixed in a separate branch in the 7fb81ec2 commit.

Audittens: *Approved*.

6.5.32 I-32. Incorrect comment in `NativeTokenVault._bridgeBurnNativeToken` function regarding `msg.value` check

Context:
l1-contracts/contracts/bridge/ntv/NativeTokenVault.sol:269-272

Description: At NativeTokenVault.sol:269-272, there is an enforcement that `msg.value` is not zero. However, the comment above it states "*The Bridgehub also checks this, but we want to be sure*", while `Bridgehub` does not perform this check.

Recommendation: Update the comment to reflect the actual behavior.

ZKsync: *Fixed* in the d994088c commit.

Audittens: *Approved*.

6.5.33 I-33. Inconsistency between `MessageRoot` implementation and documentation

Context:
Nested L3 → L1 messages tree design for Gateway
l1-contracts/contracts/bridgehub/MessageRoot.sol:116-118,141

Description: The documentation states: "*In reality, AggregatedRoot will have a meaningful value only on SyncLayer and L1. On other chains it will be a root of an empty tree*". In the `MessageRoot.getAggregatedRoot` function it is checked if `chainCount` is equal to zero, to return a root of an empty tree. However, at initialization of `MessageRoot`, `_addNewChain(block.chainid)` is called, therefore `chainCount` is greater than zero after initialization.

Recommendation: Replace the condition `chainCount == 0` with `chainCount == 1`.

ZKsync: *Acknowledged*. We will update documentation.

6.5.34 I-34. Incorrect implementation in the `L2AssetRouter`'s `onlyAssetRouterCounterpartOrSelf` modifier

Context:
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:51-60

Description: The implementation of the `L2AssetRouter`'s `onlyAssetRouterCounterpartOrSelf` modifier allows all calls where `_originChainId` is not equal to `L1_CHAIN_ID`.

Recommendation: Add an enforcement that the `_originChainId` is equal to `L1_CHAIN_ID`.

ZKsync: *Fixed* in a separate branch in the 21ac0837 commit.

Audittens: *Approved*.

6.5.35 I-35. Inconsistent parameter naming of `_l1Asset` and `_l1Token`

Context:
l1-contracts/contracts/bridge/L1Nullifier.sol:698
l1-contracts/contracts/bridge/L1ERC20Bridge.sol:265
l1-contracts/contracts/bridge/interfaces/IL1Nullifier.sol:44
l1-contracts/contracts/bridge/asset-router/IL1AssetRouter.sol:36

Description: There is inconsistency in the naming of the parameter representing the `L1` token address across different contracts: in some places, it is referred to as `_l1Asset`, while in others it is called `_l1Token`.

Recommendation: Replace `_l1Asset` with `_l1Token` where `L1` token is implied.

ZKsync: *Fixed* in the b8970e5b commit.

Audittens: *Approved*.

6.5.36 I-36. Inconsistent use of `SUPPORTED_ENCODING_VERSION` constant in `BatchDecoder`

Context:
l1-contracts/contracts/state-transition/libraries/BatchDecoder.sol:170

Description: In BatchDecoder.sol:170, `encodingVersion` is checked for equality with `0`, while elsewhere (`BatchDecoder.sol`:37,103), the `SUPPORTED_ENCODING_VERSION` constant is used for the same check.

Recommendation: Replace the hardcoded `0` with `SUPPORTED_ENCODING_VERSION`.

ZKsync: *Fixed* in the 0be70a20 commit.

Audittens: *Approved*.

6.5.37 I-37. Inconsistent variable naming for chain identifiers

Context:
l1-contracts/contracts/bridge/ntv/NativeTokenVault.sol:133,153
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:39,52,94

Description: The parameter `_originChainId` is used inconsistently in the `_bridgeMintBridgedToken`, `_bridgeMintNativeToken` functions of the `NativeTokenVault` contract and in the `onlyAssetRouterCounterpart`, `onlyAssetRouterCounterpartOrSelf` modifiers and `setAssetHandlerAddress` function of the `L2AssetRouter` contract. In these functions and modifiers, it denotes the `chainId` from which the message originates, while in the rest of the codebase, `originChainId` is used to indicate the chain from which the token originated.

Recommendation: Replace `_originChainId` with `_chainId` in mentioned places.

ZKsync: *Fixed* in the 067b33ca and fe16f546 commits.

Audittens: *Approved*.

6.5.38 I-38. Inconsistent naming of `assetAddress` and `assetHandlerAddress` in functions and events

Context:
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:96
l1-contracts/contracts/bridge/asset-router/IAssetRouterBase.sol:49

Description: In most of the codebase, the naming `_assetHandlerAddress` is used to denote the address of an asset handler. However, there are two places in the codebase, where `_assetAddress` is used but the use of `_assetHandlerAddress` is appropriate:

- Function `setAssetHandlerAddress` takes `_assetAddress` as a parameter (L2AssetRouter.sol:93-100).
- The `AssetHandlerRegistered` event (IAssetRouterBase.sol:49) uses `_assetAddress` to denote the address of asset handler.

Recommendation: Rename `_assetAddress` to `_assetHandlerAddress` in the mentioned code.

ZKsync: *Fixed* in the 2a7723f1 commit. It was already partially fixed in a separate branch in the 21ac0837 commit.

Audittens: *Approved*.

6.5.39 I-39. Inconsistent behaviour of the legacy `L1AssetRouter.finalizeWithdrawal` function between legacy and new chains

Context:
l1-contracts/contracts/bridge/asset-router/L1AssetRouter.sol:582

Description: The function `L1AssetRouter.finalizeWithdrawal` is designed to support legacy withdrawals initiated by the legacy bridge. However, for new withdrawals that use new message format (`L2AssetRouter.sol:151,161,186-190`) and therefore corresponding withdrawal messages are sent by L2AssetRouter.sol:189, the behaviour of this function with correctly provided parameters is different for different chains:

- For legacy chains, it will always revert, since `l2Sender` will be wrongly set as `legacyL2Bridge` (L1AssetRouter.sol:582).
- For new chains, it will succeed, since `l2Sender` will be correctly set as `L2_ASSET_ROUTER_ADDR`.

Recommendation: To have the same behaviour for all chains, allow to finalize only withdrawals sent by the legacy bridge: `require(legacyL2Bridge != address(0))`, `{..., l2Sender: legacyL2Bridge, ...}`.

ZKsync: *Fixed* in the 235b42ea commit.

Audittens: *Approved*.

6.5.40 I-40. Lack of reentrancy control in `L2AssetRouter` and `L2SharedBridgeLegacy`

Context:
l1-contracts/contracts/bridge/asset-router/L2AssetRouter.sol:117,150,154,284,297
l1-contracts/contracts/bridge/L2SharedBridgeLegacy.sol:103,116

Description: Functions of the `L2AssetRouter` and `L2SharedBridgeLegacy` contracts, that serve as entry points for deposit and withdrawal flows, are missing the `nonReentrant` modifier. This is inconsistent with the corresponding functionality on `L1`.

Recommendation: Add `nonReentrant` modifier to all of the entry points for finalizing deposits from `L1` to `L2` and withdrawing funds from `L2` to `L1`.

ZKsync: *Fixed* in the 35b17442 commit.

Audittens: *Approved*.