arbitrary./execution

AZTEC SECURITY ASSESSMENT

October 14, 2022

Prepared For:
Joe Andrews, Aztec
Prepared By:
John Bird, Jasper Clark

Changelog:
September 16, 2022 *Initial report delivered*
October 14, 2022 *Final report delivered*

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report contains the results of Arbitrary Execution's security assessment of the Aztec Connect smart contracts. The Aztec protocol uses PLONK technology to provide privacy for users and enable fast and inexpensive transactions on Ethereum. Aztec Connect enables Aztec users to interact with external DeFi protocols from within Aztec's layer 2 via smart contracts called bridges.

Two Arbitrary Execution (AE) engineers conducted this review over a 4-week period, from August 15, 2022 to September 12, 2022. The audited commit was `9558b62604c72e5d1ea70f330df057eaeae10bd1` in the `lh/compliance` branch of the `AztecProtocol/aztec2-internal/` repository. The complete list of files in scope is located in Appendix B. These repositories were private at the time of the engagement, so hyperlinks may not work for readers without access.

The team performed a detailed, manual review of the codebase with a focus on Aztec's `RollupProcessorV2`, `Decoder`, and `DefiBridgeProxy` contracts. In addition to manual review, the team used Slither for automated static analysis.

The assessment resulted in findings ranging in severity from low to note (informational). One low severity finding involves the absence of safety checks around the protocol's escape hatch parameters, a mechanism that allows ordinary users to send proofs directly to the rollup contract. Two other low findings focus on a time delay added to the escape hatch. The remaining low severity findings involve adding assets and bridges that do not function properly, and safety checks in the `TokenTransfers` contract. The note severity findings contain observations regarding code hygiene, documentation, and other best practices.

FIX REVIEW UPDATE

FIX REVIEW PROCESS

After receiving fixes for the findings shared with Aztec, the AE team performed a review of each fix. Each pull request was scrutinized to ensure that the core issue was addressed, and that no regressions were introduced with the fix. A summary of each fix review can be found in the *Update* section for a finding. For findings that the Aztec team chose not to address, the team's rationale is included in the update.

The Aztec team has fixed or acknowledged all major issues identified in the engagement. The full breakdown of fixes can be found in the Fixes Summary section. While the team acknowledged L04, they fixed an overflow in the same calculation in pull request #1479.

AUDIT OBJECTIVES

AE had the following high-level goals for the engagement:

- Ensure Aztec's contracts are implemented consistently with their documentation
- Identify smart contract vulnerabilities
- Evaluate adherence to development best practices

The Aztec team also identified specific questions to guide the engagement:

- Can an attacker manipulate the encoded proof data such that the proof still passes, but actions other than expected are performed? (e.g., insert extra withdraw or skip a user deposit)
- Can an attacker remove funds from the rollup, without a valid withdraw proof or valid deposit into a bridge? (Assuming the attacker doesn't hold `OWNER_ROLE` or control the `PROXY_ADMIN`)
- Can an attacker brick the contract or freeze funds indefinitely? (Assuming the attacker doesn't hold `OWNER_ROLE` or control the `PROXY_ADMIN`)
- Can an attacker escalate privileges?

OBSERVATIONS

The contracts in this repository make extensive use of inline assembly. Assembly is used in part to save on gas and reduce deployed bytecode size, but also to decode Aztec's custom proof data encoding scheme. Writing code in yul is more error prone than writing Solidity. Higher level languages place more burden on the compiler to choose code that will be executed, whereas assembly places that responsibility on the developer. It is the developer's responsibility to check every instruction written for issues like off-by one errors, mistyped bitmasks or literals, and incorrect argument ordering because code with these problems will often compile but not behave as intended.

Assembly code can also be more sensitive to Solidity compiler bugs, as some bugs are only reachable from inline assembly. Bugs are continuously being fixed in the Solidity compiler, and sometimes new bugs are introduced in the process. These bugfixes are **not** backported to older compiler versions. All projects, and projects that use assembly in particular, must take care in understanding what bugs are present in their current compiler version.

On top of the security implications of writing assembly code, there are tradeoffs to consider between readability/maintainability and performance. When using assembly, developers get additional control over code execution but the code can become more difficult to understand and maintain. It is critical to keep code comments up-to-date and accurate to aid developers and auditors. When in doubt, err on the side of being explicit over implicit.

The `RollupProcessorV2` contract is approaching the bytecode size limit introduced in the Spurious Dragon hard-fork. Hardhat's contract sizer reports a size of 24.438 KB, which leaves 138 bytes before `RollupProcessorV2` will exceed the size limit for mainnet deployment. It is feasible that a new feature could push the contract size over the limit of 24.576 KB. The team will have to be mindful of this constraint as they continue development.

SYSTEM OVERVIEW

USER CATEGORIES

USERS

Users can deposit funds into Aztec's Ethereum smart contracts and claim funds on Aztec's L2 to privately transact with one another and interact with external protocols through Aztec bridges. Users do not hold any special roles in the context of the smart contracts.

PRIVILEGED ROLES

There are 4 roles defined in Aztec's access control scheme in addition to `AccessControl`'s `DEFAULT_ADMIN_ROLE`.

OWNER_ROLE

The `OWNER_ROLE` has access to functions that modify the configuration of the `RollupProcessorV2` contract. This role can perform actions including:

- Adding and removing rollup providers
- Changing the addresses of the `DefiBridgeProxy` and PLONK verifier
- Changing the escape hatch delay
- Allowing third parties to add assets and bridges

EMERGENCY_ROLE

Holders of the `EMERGENCY_ROLE` can pause the `RollupProcessorV2` contract.

RESUME_ROLE

Holders of the `RESUME_ROLE` can unpause the `RollupProcessorV2` contract.

LISTER_ROLE

Holders of the `LISTER_ROLE` can add new supported assets and bridges to the `RollupProcessorV2` contract. They can also set the asset cap for a particular asset.

SYSTEM COMPONENTS

ROLLUPPROCESSORV2

The `RollupProcessorV2` is an updated version of `RollupProcessor.sol`. It is responsible for processing Aztec zk-rollup proofs, relaying the proofs to a verifier contract, and performing relevant ether and ERC-20 token transfers to users and DeFi bridges.

A `RollupState` structure is defined to track state information pertinent to the current rollup.

REAL AND VIRTUAL ASSETS

The rollup processor supports two types of assets:

- Real assets are either ether or ERC-20 tokens. Real assets on Aztec's L2 have a corresponding asset on L1.
- Virtual assets exist purely inside the Aztec network and do not have a corresponding asset on L1. These are used by bridges to track data such as loans or votes in a DAO.

Assets in the rollup processor are tracked by an `assetId`. Real and virtual assets can be distinguished by their `assetId` format.

ASSET CAP

Asset caps are restrictions placed on supported Aztec assets. Caps limit the daily amount of deposits for a particular asset. There is a `capped` flag inside the `rollupState` structure to enable and disable the enforcement of asset caps. This flag can be toggled by the `OWNER_ROLE` through calling the `setCapped` function.

ESCAPE HATCH

The escape hatch is a window of time (measured in blocks) in which anyone can submit rollup proofs to the rollup processor. It exists for the scenario where Aztec disappears or rollup providers are unavailable.

ROLLUPPROCESSORLIBRARY

The `RollupProcessorLibrary` is a helper contract that contains signature validation methods for the rollup processor.

ROLLUP PROVIDER

A rollup provider is a third party that constructs rollup proofs. Aztec currently acts as a rollup provider. Rollup providers are tracked in the rollup processor with the `rollupProviders` mapping. Rollup providers call `processRollup` to decode and verify rollup proofs.

PERMITHELPER

The `PermitHelper` is a helper contract for performing ERC-20 permit actions.

DECODER

The `Decoder` contract is responsible for decoding and extracting proof data. It receives encoded proof data in `calldata` when the rollup processor calls the `decodeProof` function. The decoder decodes the encoded calldata, and returns the full proof data back to the rollup processor.

DEFIBRIDGEPROXY

The `DefiBridgeProxy` calls bridge contracts to convert Aztec inputs into outputs based on an external protocol interaction.

BRIDGE

A bridge in the context of Aztec is an L1 smart contract that translates an external contract's interface into the Aztec Connect interface. For example, a Uniswap bridge contract would allow users to spend Aztec L2 funds to perform swaps on mainnet. Bridges are called through the `DefiBridgeProxy`.

VULNERABILITY STATISTICS

Severity	Count
Critical	0
High	0
Medium	0
Low	5
Note	11

FIXES SUMMARY

Finding	Severity	Status
L01	Low	Fixed in pull request #1476
L02	Low	Fixed in pull request #1515
L03	Low	Fixed in pull request #1478
L04	Low	Acknowledged
L05	Low	Fixed in pull requests #1480 and #1517
N01	Note	Fixed in pull request #1487
N02	Note	Acknowledged
N03	Note	Fixed in pull request #1505
N04	Note	Fixed in pull request #1501
N05	Note	Acknowledged
N06	Note	Fixed in pull request #1502
N07	Note	Fixed in pull request #1513
N08	Note	Fixed in pull request #1503
N09	Note	Fixed in pull request #1506
N10	Note	Acknowledged
N11	Note	Fixed in pull request #1504

FINDINGS

LOW SEVERITY

[L01] LACK OF BOUNDS CHECKING ON ESCAPE HATCH VALUES

The upper and lower bounds for the escape hatch window are set in the `RollupProcessorV2` constructor without bounds checking:

```
constructor(uint256 _escapeBlockLowerBound, uint256 _escapeBlockUpperBound) {
    _disableInitializers();
    rollupState.paused = true;

    escapeBlockLowerBound = _escapeBlockLowerBound;
    escapeBlockUpperBound = _escapeBlockUpperBound;
}
```

If these values are inverted during deployment, the escape hatch will never open when the `RollupProcessor` calls `getEscapeHatchStatus`. Because `escapeBlockLowerBound` and `escapeBlockUpperBound` are declared `immutable`, a new `RollupProcessorV2` contract will have to be deployed to update the values.

RECOMMENDATION

Consider adding a check that ensures `escapeBlockUpperBound` is greater than `escapeBlockLowerBound`.

UPDATE

Fixed in pull request #1476 (commit hash `fba1e694bc1ed2ee0679072d9b08c6debc2b248b`), as recommended.

[L02] MISSING EXTCODESIZE CHECK WHEN USING LOW-LEVEL CALL

The `safeTransferTo` and `safeTransferFrom` functions in `TokenTransfers.sol` use the low-level `call` instruction when transferring tokens. The `call` instruction will return `true` when there is no code present at an address, which is known behavior. However, neither function checks to ensure code is present at the target address. While it is unlikely that an address with no code would be added to the `supportedAssets` whitelist, in the event that this did occur the `safeTransferTo` and `safeTransferFrom` functions would erroneously succeed.

RECOMMENDATION

Consider adding a check using the `extcodesize` instruction to ensure there is code present at a target address before using `call`.

UPDATE

Fixed in pull request #1515 (commit hash `75d31df88d3e796760e8ae6e581d9259b16a4c2b`), as recommended.

[L03] INCORRECTLY SET PROTOCOL GAS LIMIT CAN RENDER AN ASSET OR BRIDGE NON-FUNCTIONAL

In `RollupProcessorV2.sol`, the `setSupportedBridge` and `setSupportedAsset` functions call `sanitiseBridgeGasLimit` and `sanitiseAssetGasLimit` respectively to adjust contract gas limits within the bounds set by the protocol.

If a new asset or bridge specifies a gas limit that is above `MAX_BRIDGE_GAS_LIMIT` or `MAX_ERC20_GAS_LIMIT`, the sanitise function will cap the limit to the corresponding max. If `MAX_*_GAS_LIMIT` is lower than what the bridge or asset needs to function, the newly added contract will be unusable. Failing fast, rather than capping the limit is more clear to users and prevents the possibility of adding non-functional assets and bridges.

RECOMMENDATION

Consider reverting if a user specifies a gas limit that is greater than `MAX_BRIDGE_GAS_LIMIT` or `MAX_ERC20_GAS_LIMIT`.

UPDATE

Fixed in pull request #1478 (commit hash `8bfc8d1f034438b6aa9777dc7ad9b44824887148`), as recommended.

[L04] TRUNCATION OF `BLOCK.TIMESTAMP` CAN LEAVE ESCAPE HATCH OPEN INSIDE DELAY WINDOW

Timestamps for the latest rollup and asset cap updates are tracked in the `RollupProcessorV2` contract with `lastRollupTimeStamp` and `lastUpdatedTimestamp`. Both of these variables are `uint32` types.

Because `block.timestamp` is of type `uint` and `delayBeforeEscapeHatch` is of type `uint32`, the following condition will never be true once `block.timestamp` exceeds 2**32:

```
if (block.timestamp < lastRollupTimeStamp + delayBeforeEscapeHatch) {
    isOpen = false;
}
```

This will prevent the getEscapeHatchStatus function from closing the hatch until `delayBeforeEscapeHatch` has elapsed.

RECOMMENDATION

Consider storing `lastRollupTimeStamp` in a `uint256` type.

UPDATE

Acknowledged. Aztec's statement for this issue:

> *The time frame that the contract needs to stay active for this to become an issue, is longer than what we expect the specific instance of the project to survive.*

[L05] LASTROLLUPTIMESTAMP CAN BE INCORRECTLY SET

The `lastRollupTimeStamp` variable in the `RollupProcessorV2` contract is used to calculate the escape hatch delay and is updated when a new rollup is processed. It is also updated whenever `setCapped` is called. Repeated calls to `setCapped(true)` would increase the `lastRollupTimeStamp` at a quicker rate than expected and reset the delay window in `getEscapeHatchStatus`:

```
    function setCapped(bool _isCapped) external onlyRole(OWNER_ROLE)
noReenter {
        rollupState.capped = _isCapped;
        if (_isCapped) {
            lastRollupTimeStamp = uint32(block.timestamp);
        }
        emit CappedUpdated(_isCapped);
    }
```

This could only be performed by holders of the `OWNER_ROLE`, so the likelihood of this being abused is low.

RECOMMENDATION

Consider returning early from the `setCapped` function if `rollupState.capped` is equal to `_isCapped`.

UPDATE

Fixed in pull requests #1480 (commit hash `c93bf80a89d9b116224ed69f9a49cf0714231201`) and #1517 (commit hash `895ba7a0218041a3661466136b7ef1725b53c4ef`), as recommended.

NOTE SEVERITY

[N01] MISSING BRIDGE ZERO-ADDRESS CHECKS

In `RollupProcessorV2.sol`, the `setDefiBridgeProxy` function does not check that the `_defiBridgeProxy` address is nonzero.

If the `_defiBridgeProxy` address is set to zero, calls to the proxy will fail until an additional call to `setDefiBridgeProxy` is made to set the correct address.

RECOMMENDATION

Consider checking that the `_defiBridgeProxy` address supplied to `setDefiBridgeProxy` is nonzero.

UPDATE

Fixed in pull request #1487 (commit hash `a11d333fc4234ec787d658b778148e4d72047de6`), as recommended. A zero-address check was also added to the `setVerifier` function.

[N02] BUILD ISSUES

Build errors were encountered when following the steps outlined in `getting_started.md`

The following errors were encountered when building on MacOS Monterey 12.5.1:

```
/aztec2-
internal/barretenberg/src/aztec/ecc/curves/bn254/../../groups/./element_impl.
hpp:249:40: [ 27%] Building CXX object _deps/leveldb-
build/CMakeFiles/leveldb.dir/util/options.cc.o
fatal error: use of bitwise '|' with boolean operands [-Wbitwise-instead-of-
logical]
        const bool edge_case_trigger = x.is_msb_set() | other.x.is_msb_set();

gyp ERR! stack Error: Command failed: /opt/homebrew/bin/python3 -c import
sys; print "%s.%s.%s" % sys.version_info[:3];
gyp ERR! stack   File "<string>", line 1
gyp ERR! stack     import sys; print "%s.%s.%s" % sys.version_info[:3];
gyp ERR! stack                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
gyp ERR! stack SyntaxError: Missing parentheses in call to 'print'. Did you
mean print(...)?
```

The following errors were encountered when building on Ubuntu 20.04:

```
In file included from /aztec2-
internal/barretenberg/src/aztec/stdlib/hash/pedersen/../../primitives/compose
rs/composers.hpp:4:
/aztec2-
internal/barretenberg/src/aztec/plonk/composer/plookup_composer.hpp:126:14:
fatal error: no template named 'optional' in namespace 'std'
        std::optional<uint32_t> key_b_index = std::nullopt);
        ~~~~~^
```

The following additional steps were taken to successfully build on MacOS Monterey 12.5.1:

- Run `brew install llvm libomp clang-format` as per the instructions in Barretenberg's `bootstrap script`
- Squelch warnings that caused `fatal error: use of bitwise '|' with boolean operands [-Wbitwise-instead-of-logical]` in `barretenberg`
- Alias `nproc` to `sysctl -n hw.physicalcpu`
- Install Python 2.7 as it is required by `sqlite3`

Ensuring builds work on fresh systems will decrease developer and auditor spin-up time.

RECOMMENDATION

Consider re-testing builds on the supported platforms, and updating documentation accordingly.

UPDATE

Acknowledged. Aztec's statement on the issue:

> *We are actively working on translating our tests to use Foundry and repackaging the blockchain sub-dir, such that it can be run as a standalone for developers and auditors to reduce spin-up time.*

[N03] REDUNDANT INPUT VALIDATION IN `WITHDRAW`

The `withdraw` function in the `RollupProcessorV2` contract contains redundant input validation. Both the `validateAssetIdIsNotVirtual` modifier and the `getSupportedAsset` function call ensure that the `_assetId` input is non-virtual.

RECOMMENDATION

Consider removing the extraneous validation check.

UPDATE

Fixed in pull request #1505 (commit hash `b7805e2ebf81e1fe9752e0ff28f09543f8fcb04f`), as recommended.

[N04] HARDCODED GAS VALUES

The `RollupProcessorV2` contract uses assembly `call` opcodes to perform transfers in the following locations:

- `RollupProcessorV2.sol`, line 1304
- `RollupProcessorV2.sol`, line 1341

The rationale for ignoring `call` return values is explained in comments, but the rationale for using hard-coded gas values (50000 and 30000) is not.

RECOMMENDATION

Consider adding comments to justify the use of fixed gas parameters over the `gas()` opcode.

UPDATE

Fixed in pull request #1501 (commit hash `516d12fc369dce7e99ac5c32e286ec3cf5f7eaa6`), as recommended.

[N05] USE OF FLOATING COMPILER VERSION PRAGMA

All contracts in this audit float their Solidity compiler versions (e.g. `pragma solidity >=0.8.4`).

Locking the compiler version prevents accidentally deploying the contracts with a different version than what was used for testing. The current pragma prevents contracts from being deployed with an outdated compiler version, but still allows contracts to be deployed with newer compiler versions that may have higher risks of undiscovered bugs.

It is best practice to deploy contracts with the same compiler version that is used during testing and development (in this case `0.8.10`).

RECOMMENDATION

Consider locking the compiler pragma to the specific version of the Solidity compiler used during testing and development.

UPDATE

Acknowledged. Aztec's statement on the issue:

> *The pragma is locked through the configuration of deployment. We let the pragma float in the code to allow easy patching if an issue should be found in the used version (0.8.10).*

[N06] INCORRECT DOCUMENTATION

Throughout the codebase, there are comments that do not match the referenced code.

The following comments mention a "92-byte length parameter in the `signature` byte array" when the code uses a 96-byte length:

- `RollupProcessorV2.sol` line 1224
- `RollupProcessorV2.sol` line 1234
- `RollupProcessorLibrary.sol` line 104

The following comments also do not match the implementation:

- `DefiBridgeProxy.sol`, Line 70: `receiveEthPayment` should be `receiveEthFromBridge`

Inaccurate comments impact code readability, and can cause developers to make errors in the future.

RECOMMENDATION

Consider updating code comments to match the implementation.

UPDATE

Fixed in pull request #1502 (commit hash `659f896717e4ff8332ee6308d33de89a520dec60`), as recommended.

[N07] SHADOWING WITH FUNCTION PARAMETER

In `PermitHelper.sol`, the `depositPendingFundsPermit` and `depositPendingFundsPermitNonStandard` functions shadow the `owner` getter function defined in OpenZeppelin's `Ownable.sol`:

```
    function depositPendingFundsPermit(
        uint256 assetId,
        uint256 amount,
        address owner, <--- also defined in Ownable.sol
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    )
```

The parameter is used in the following locations:

- `PermitHelper.sol`, line 62
- `PermitHelper.sol`, line 90

There is no security impact in this particular case, as the functions in `PermitHelper.sol` behave correctly. Regardless, name collisions and variable shadowing can lead to confusion when reading or writing code.

RECOMMENDATION

Consider renaming the `owner` parameter in `PermitHelper.sol` or prefixing the name with an underscore to avoid shadowing.

UPDATE

Fixed in pull request #1513 (commit hash `a8dffd1d7ecf84c7df2bdfd6d98b374e21b91272`), as recommended. Function parameters in `PermitHelper.sol` are now prefixed with underscores.

[N08] TYPOGRAPHICAL ERRORS

The following lines contain typographical errors:

- `RollupProcessorV2.sol`, line 1113: `If does not return` should be `It does not return`
- `RollupProcessorV2.sol`, line 1239: `sheild` should be `shield`.
- `RollupProcessorLibrary.sol`, line 110: `sheild` should be `shield`.

RECOMMENDATION

Consider making the suggested changes to fix the typographical errors.

UPDATE

Fixed in pull request #1503 (commit hash `e3c062fc5e46cbdb2d52b4515a6377863c0d60bc`), as recommended.

[N09] USE OF LONG NUMERICAL LITERALS

Long numerical literals are used in the `RollupProcessorV2` and `Decoder` contracts.

Underscores can separate digits of numeric literals to aid in readability. The following bitmasks defined in `RollupProcessorV2.sol` `Decoder.sol` will be easier to read and verify with underscores separating words (e.g. `0xffff_ffff` instead of `0xffffffff`):

- `RollupProcessorV2.sol`, lines 256-259
- `RollupProcessorV2.sol`, line 266
- `Decoder.sol`, line 114

These changes will increase code readability for developers and auditors.

RECOMMENDATION

Consider adding underscores to separate digits for long numerical literals.

UPDATE

Fixed in pull request #1506 (commit hash `6ab1ff65be153b970589ac88cc08311de0a244ab`), as recommended.

[N10] INCOMPLETE INITIALIZATION

The `RollupProcessorV2` contract adds two new access roles (`LISTER_ROLE` and `RESUME_ROLE`) but does not call `_grantRole` in its `initialize` function. This behavior is different than the original `RollupProcessor` contract. The `LISTER_ROLE` and `RESUME_ROLE` must be configured after deployment.

If the `RollupProcessorV2` is paused before the `RESUME_ROLE` is configured the normal unpause workflow will not work as intended, requiring the `OWNER_ROLE` to unpause the contract.

RECOMMENDATION

Add the configuration of the `LISTER_ROLE` and the `RESUME_ROLE` to the `initialize` function in `RollupProcessorV2.sol`.

UPDATE

Acknowledged. Aztec's statement for the issue:

> *We acknowledge this, but are not configuring the roles in the initializer as the addresses of the holders are not "stable" and we are very close to code size limits for the current optimizer config.*

[N11] NONSTANDARD USE OF UNNAMED FUNCTION PARAMETER

The `processRollup` function in the `RollupProcessorV2` contract contains an unnamed parameter that is used in the `decodeProof` function by accessing calldata:

```
function processRollup(
    bytes calldata, /* encodedProofData */
    bytes calldata _signatures
) external override(IRollupProcessor) whenNotPaused allowAsyncReenter {
```

This technique works and the reasoning is documented in the decoder, but according to the Solidity documentation it is not a normal use case:

> The names of unused parameters (especially return parameters) can be omitted. Those parameters will still be present on the stack, but they are inaccessible.

An additional `@dev` comment in the `processRollup` docstring will make it clear to readers why `encodedProofData` is ignored, and how it will be used later.

RECOMMENDATION

Consider documenting why the `encodedProofData` parameter is unnamed, and how it is being accessed in `decodeProof`.

UPDATE

Fixed in pull request #1504 (commit hash `5165e20a685e6a8ad3f37d20beff64bf201d7db2`), as recommended.

APPENDIX

APPENDIX A: SEVERITY DEFINITIONS

Severity	Definition
Critical	This issue is straightforward to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users.
High	This issue is difficult to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users.
Medium	This issue is important to fix and puts a subset of users' data at risk and is possible to lead to moderate financial impact.
Low	This issue is not exploitable in a recurring basis and cannot have a significant impact on execution.
Note	This issue does not pose an immediate risk but is relevant to security best practices.

APPENDIX B: FILES IN SCOPE

```
Decoder.sol
DefiBridgeProxy.sol
libraries/RollupProcessorLibrary.sol
libraries/TokenTransfers.sol
periphery/PermitHelper.sol
processors/RollupProcessorV2.sol
```