Audit Report for Aztec -April 10, 2022

Summary

Audit Report prepared by Solidified covering the Aztec protocol Ethereum smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief on 15 March 2022.

Audited Files

The source code has been supplied in the form of a Github repository:

https://github.com/AztecProtocol/aztec2-internal

Commit number: `25c02619428f03d91358c0685dffcecf6f640d3c`

The scope of the audit was limited to the following files:

```
contracts
|-- AztecTypes.sol
|-- bridges
|   `-- UniswapBridge.sol
|-- Decoder.sol
|-- DefiBridgeProxy.sol
|-- interfaces
|   |-- IDefiBridge.sol
|   |-- IERC20Permit.sol
|   |-- IFeeDistributor.sol
|   |-- IRollupProcessor.sol
|   `-- IVerifier.sol
|-- libraries
|   |-- RollupProcessorLibrary.sol
|   `-- TokenTransfers.sol
`-- RollupProcessor.sol
```

Cryptographic libraries, proof verifications, and the fee distribution contract have been explicitly excluded from scope

Intended Behavior

The smart contracts implement the Ethereum base contracts for the Aztec L2 solution, providing functionality to process rollups

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than in a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	High	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Solidified found that the Aztec contracts contain 3 critical issues, 2 major issues, 2 minor issues 2, in addition to 3 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	ETH deposit attack can revert bridge transactions	Critical	Resolved
2	ETH deposit attack for async bridge interactions	Critical	Resolved
3	Entire rollup transaction can be reverted by a malicious ERC20 implementation	Critical	Resolved
4	Anyone can deposit the assets of users after ERC20 approval	Major	Resolved
5	Missing access control for processing DeFi interaction	Minor	Resolved
6	Contract deployer address always added as rollupProvider	Minor	Resolved
7	Missing bridge canFinalise check in the RollupProcessor	Minor	Resolved
8	Outdated Comments	Note	-
9	Lack of event usage	Note	-
10	Unnecessary use SafeMath	Note	-

Critical Issues

1. ETH deposit attack can revert bridge transactions

Each DeFi bridge has a unique `interactionNonce` which is used by the bridge to deposit ETH into the Rollup contract by calling the public function `receiveEthFromBridge`.

The defi bridge returns the deposited ETH amount as return value which is compared with the actual transferred ETH amount in function `DefiBridgeProxy.recoverTokens`:

```
if (outputValue != ethPayment) {
   revert INSUFFICIENT_ETH_PAYMENT();
}
```

An attacker could frontrun each rollup transaction to calculate the interactionNonce and deposit 1 wei for all `interactionNonce` values which involves an ETH transaction. This would force a revert of all bridge interactions with ETH as an output asset.

The attack would be rather cheap because it would only involve calling the `receiveETHFromBridge` function multiple times for different bridge `interanctionNonce` values.

This attack could block all defi bridges which include ETH as an output asset which can lead to financial losses because users can't swap their assets to ETH as expected.

Recommendation:

Consider replacing the above code with the following:

```
if (outputValue > ethPayment) {
   revert INSUFFICIENT_ETH_PAYMENT();
}
```

2. ETH deposit attack for async bridge interactions

The ETH deposit attack described in issue 1 can also be applied for async bridge interactions.

The `transferTokenAsync` method in the `RollupProcessor` contract includes the same check:

```
require(outputValue == ethPayments[interactionNonce], 'argh insufficient eth payment');
```

This method is called as part of the `processAsyncDefiInteraction` function in the `RollupProcessor` contract.

Recommendation:

Consider replacing the above `require` statement with the following:

```
require(outputValue <= ethPayments[interactionNonce], 'argh insufficient eth payment');
```

3. Entire rollup transaction can be reverted by a malicious ERC20 implementation

Anyone can add a new ERC20 as a supported asset to the Aztec protocol by calling the public `setSupportedAsset` function in the `RollupProcessor`.

One parameter defines the `uint256 gasLimit` for the ERC20 transferFrom calls.

If the gas limit is zero, a default value is assigned:

```
assetGasLimits[assetId] = gasLimit == 0 ? DEFAULT_ERC20_GAS_LIMIT : gasLimit;
```

However, it is possible to add a very high number as a gas limit.

An attacker could generate a malicious ERC20 implementation which can activate a very gas intensive `transferFrom` method call to use up the entire gas of a block. The malicious token can be added as a supported asset with a very high gas limit.

If a rollup block includes a `withdraw` transaction of the malicious token, the entire rollup would revert with an out of gas error.

This attack would be expensive to perform but it could block all rollup transactions which can lead to financial losses if users can't withdraw their assets.

Recommendation:
We recommend reconsidering the security assumptions that anyone can add the support for an ERC20. As a concrete fix for the gas attack, a `MAX_ERC20_GAS_LIMIT` constant could be introduced.

Major Issues

4. Anyone can deposit the assets of users after ERC20 approval

The function `depositPendingFunds` allows depositing `ERC20` tokens on behalf of any account as long as `ERC20` approval has been issued for the `RollupProcessor`. It is quite common for users to approve a bigger `ERC20` amount for a contract to save the gas costs. The usual expectation is that the contract will not use such approval until the user himself signs an `ERC20` deposit transaction. While the deposited `ERC20` tokens are still recoverable in case of EOA account, such tokens might be lost in case of a smart contract, if a smart contract is not programmed to handle such a case.

Recommendation:
Consider enforcing that `msg.sender` equals `depositorAddress` when depositing `ERC20` tokens.

Minor Issues

5. Missing access control for processing DeFi interaction

The `processAsyncDefiInteraction` in the `RollupProcessor` contract should only be callable by the `bridgeContract`, as stated in the comments.

However, the actual check for this is missing, leading to the function being callable by anyone, even before the bridge is ready to finalize the bridge interaction.

Depending on the bridge implementation, this may result in calling `bridge.finalise` call before the async interaction is finished.

In the worst case, this could lead to locked assets in the bridge contract.

Recommendation:

Add the following guard:

```
require(msg.sender == bridgeContract)
```

Resolved:
This is by design - the `processAsyncDefiInteraction` should be publicly callable. The finalise method in the bridge needs to handle the case, in which the `interactionNonce` can't be finalised at the current point in time.

An incorrect comment in the source code misguided the auditor team, that the method should only be callable by the bridge itself.

6. Contract deployer address always added as rollupProvider

In the `initialize` method in the `RollupProcessor` contract a `contractOwner` parameter is supplied.

However, the `rollupProvider` is always set to `msg.sender`. in the case where `contractOwner` is not the deployer's address this may be incorrect.

Recommendation

Consider using the following assignment:

```
rollupProviders[_contractOwner] = true;
```

In addition, an event should be emitted.

7. Missing bridge canFinalise check in the RollupProcessor

The bridge interface defines a method `canFinalise` which indicates if the bridge is async and has a `finalise` method.

The `processAsyncDefiInteraction` method in the RollupProcessor doesn't check if the bridge implements this method.

Recommendation

Consider removing the interface method if not needed, or checking if the call returns true before calling the finalise method

Informational Notes

8. Outdated Comments

The comments are outdated or incorrect in multiple places. Examples of this are:

Incorrect `receiveEthFromBridge` function name in comments

The receiveEthFromBridge is referred as receiveEthPayment in the comments

Debug Helper import still in comments:

```
// import 'hardhat/console.sol';
```

`RollupProcessor.setSupportedAsset` is still described as callable only by the owner

9. Lack of event usage

The codebase lacks events in multiple places.

- no event for a new adding a DeFi bridge in `setDefiBridgeProxy`
- no `RollupProviderUpdated` event in the initialize method
- no event in the `setAssetPermitSupport` method

10. Unnecessary use SafeMath

The `UniswapBridge` and the `AztecFeeDistributor` contract use the `SafeMath` library. However, this is not required in Solidity versions above 0.8.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Aztec Protocol or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors from legal and financial liability.

Oak Security GmbH