Bancor Liquidity

Security Assessment

October 9th, 2020

For :
Yudi Levi @ Bancor

By :
Adrian Hetman @ CertiK
adrian.hetman@certik.org

By :
Abhishek Sharma @ CertiK
abhishek.sharma@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	Bancor Liquidity
Description	Bancor Liquidity Protection
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	1.0e094e3f9514f8dd81023960ddf49622b3508c7b 2. 9987315ce102f929b7959abc99366d15af2d90b5

Audit Summary

Delivery Date	Oct. 9, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Sep. 8, 2020 - Sep. 9 2020

Vulnerability Summary

Total Issues	47
Total Critical	0
Total Major	0
Total Minor	8
Total Informational	39

Executive Summary

The report represents the results of our engagement with Bancor on their Governance functionality. The initial review was conducted for seven days: Sep. 29, 2020 - Oct. 5, 2020, by Adrian Hetman and Abhishek Sharma.

During the initial audit, no significant issues were found, mostly different optimization and some small cases regarding good practices. The team decided to apply most of the optimizations but left other issues untouched due to time constraints and will be fixed in future revisions.

Findings

ID	Title	Type	Severity
BNC-01	Old implementation of Ownable functionality	Optimization	Informational
BNC-02	Unneficient ReentrancyGuard implementation	Optimization	Informational
BNC-03	Incorrect version of solidity	Implementation	Minor
BNC-04	Use of transfer() for sending ETH	Implementation	Minor
BNC-05	External calls inside a loop	Implementation	Informational
BNC-06	Use of an assert	Optimization	Informational
BNC-07	check-effect-patterns not followed when sending ether and tokens	Implementation	Informational
BNC-08	Usage of syncReserveBalances in the beginning of many functions	Implementation	Informational
BNC-09	Use of safemath is required in arithmetic operations	Implementation	Minor
BNC-10	Usage of literals instead of constant variables	Implementation	Informational
BNC-11	Variable from function param used on it's own.	Implementation	Informational
BNC-12	Usage of uint16 in for loops	Optimization	Informational
BNC-13	maxSystemNetworkTokenAmount limit is mutable.	Implementation	Informational
BNC-14	Local variable shadowing	Implementation	Informational
BNC-15	Inefficient greater-than comparison w/ zero	Optimization	Informational
BNC-16	Unneccessary if statement	Optimization	Informational
BNC-17	Unneccessary if statement	Optimization	Informational

BNC-01: Old implementation of Ownable functionality

Type	Severity	Location
Optimization	Informational	Owned.sol

Description:

Implementation of `Owned.sol` comes from OpenZeppelin version 1.12.0, which is from 2018 and is no longer valid.

Recommendation:

We recommend using the newest version of OpenZeppelin library and import `Ownable.sol`. For more complex role management, open zeppelin offers `AccessControl.sol` which is a contract module that allows children to implement role-based access control mechanisms, but in this case, we recommend using `Ownable.sol` from version 3.1.0

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-02: Unefficient ReentrancyGuard implementation.

Type	Severity	Location
Optimization	Informational	ReentrancyGuard.sol

Description:

`ReentrancyGuards.sol` implementation is un-optimal as booleans are more expensive than uint256 or any type that takes up a full word because each write operation emits an extra SLOAD to first read the slot's contents, replace the bits taken up by the boolean, and then write back.

Recommendation:

It's suggested to change use of `bool` to an `uint256` variable, similar to version 3.2.0 of OpenZeppelin's ReentrencyGuard.

Alleviation:

Issue was resolved.

BNC-03: Incorrect version of solidity

Type	Severity	Location
Implementation	Minor	General

Description:

The linked contracts necessitate a version too recent to be trusted. Consider deploying with 0.6.11. We do not recommend using any latest version for deployment, especially if any changes were made in the optimizer or the language semantic. Version 0.6.12 made changes to optimizer that's why we do not recommend using this version.

Recommendation:

Deploy with any of the following Solidity versions:

- 0.5.11 - 0.5.13,
- 0.5.15 - 0.5.17,
- 0.6.8,
- 0.6.10 - 0.6.11. Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Alleviation:

The team decided to stay with current version of solidity i.e. `0.6.12`

BNC-04: Use of transfer() for sending ETH

Type	Severity	Location
Implementation	Minor	ConverterBase.sol L240, LiquidityPoolV1Converter.sol L179, LiquidityPoolV1Converter.sol 361,LiquidityPoolV1Converter.sol 608, LiquidityPoolV1Converter.sol 674

Description:

After EIP 1884 was included in the Istanbul hard fork, it's not recommended to use `.transfer()` and `.send()` as this function's a hard dependency on gas costs make them obsolete as they are forwarding a fixed amount of gas `2300`. This can cause some issues when sending ether to a contract. `send` and `transfer` do not longer protect from reentrancies in case of gas price changes.

Recommendation:

It's recommended to stop using `.transfer()` and `.send()` and instead either use one of the wrappers from openzeppelin like `sendValue()` from Address.sol or use `.call()`. Even if all transfers are done to a EOAs it is a good practice to switch to using `sendValue()` from OZ Address.sol or `call()` for having more future-proof systems.

Note that `.call()` does nothing to mitigate reentrancy attacks, so other precautions must be taken. To prevent reentrancy attacks, it is recommended to use the checks-effects-interactions pattern.

```
// This forwards all available gas. Be sure to check the return value!
(bool success, ) = msg.sender.call.value(amount)("");
require(success, "Transfer failed.");
```

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-05: External calls inside a loop

Type	Severity	Location
Optimization	Informational	ConverterUpgrader.sol L169-L184, ConverterUpgrader.sol L211-L229, ConverterUpgrader.sol L249-L254

Description:

External calls in the loop may lead to issues with GAS costs of a transaction.

Recommendation:

Revise the code and, if it's possible, favor pull over push strategy for external calls.

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-06: Use of an assert

Type	Severity	Location
Optimization	Informational	LiquidityPoolV1Converter.sol L159,LiquidityPoolV1Converter.sol L602, LiquidityProtectionStore.sol L184,LiquidityProtectionStore.sol L413, LiquidityProtectionStore.sol L522, LiquidityProtection.sol 722, ConverterBase.sol L569

Description:

`assert()` uses all gas available in the transaction. The `assert()` function should only be used to test for internal errors and check invariants. Within Solidity, unsigned integers are restricted to the non-negative range. As such, greater-than comparisons with the literal `0` are inefficient gas-wise.

Recommendation:

It is recommended to use `require()` with reason strings which reverts the transaction and sends the user rest of the gas and consider converting the linked comparisons to inequality ones in order to optimize their gas cost, for example:

```
uint256 length = poolWhitelist.length;
require(length != 0); // cannot be negative due to being unsigned
uint256 lastIndex = length - 1; // safe due to passing requirement above. No
need for SafeMath
```

Alleviation:

The team will be fixing the issues in their own timeframe.

 BNC-07: check-effect-patterns not followed when sending ether and tokens

Type	Severity	Location
Implementation	Informational	LiquidityPoolV1Converter.sol L586-L621,LiquidityPoolV1Converter.sol L338-L391

Description:

`transfer` or `safeTransferFrom` is done before storage variables are updated.

Recommendation:

It is recommended to follow checks-effects-interactions pattern for cases like this. Contract is protected from re-entrancies from using `protected` modifier but it's always a good practice to follow check-effects-interactions pattern.

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-08: Usage of `syncReserveBalances` in the beginning of many functions

Type	Severity	Location
Informational	Informational	LiquidityPoolV1Converter.sol L344,LiquidityPoolV1Converter.sol L590, LiquidityPoolV1Converter.sol L658

Description:

`syncReserveBalances` is called at the start of many functions.

Recommendation:

As this is only used at the start of functions it could be better to code it as a `modifier` rather than `function()`.

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-09: Use of safemath is required in arithmetic operations

Type	Severity	Location
Implementation	Minor	LiquidityPoolV1Converter.sol, LiquidityProtection.sol

Description:

SafeMath is not used on arithmetic operations. This can lead to potential overflow/underflow. `LiquidityPoolV1Converter.sol` and `LiquidityProtection.sol` does many math operations on uint256 but is not using SafeMath.

Recommendation:

It is recommended to use SafeMath.sol from openzeppelin for most mathematical operations on uint variables unless stated otherwise in other findings.

Alleviation:

Issue is resolved

BNC-10: Usage of literals instead of constant variables

Type	Severity	Location
Implementation	Informational	LiquidityPoolV1Converter.sol L96-97

Description:

Literals with many digits are difficult to read and are hard to maintain in terms of code changes.

Recommendation:

It is recommended to convert literals to the constant values and use them instead. This way, code will look cleaner, and it will be easier to maintain.

Alleviation:

Issue is resolved.

BNC-11: Variable from function param used on it's own.

Type	Severity	Location
Implementation	Informational	ConverterUpgrader.sol L91

Description:

`_version` variable from param is used on its own.

Recommendation:

We suspect this is just a mistake, and removal of that line is advised.

Alleviation:

Explanation comment was added to the code. It was intentionally used to suppress compilation warning.

BNC-12: Usage of uint16 in for loops

Type	Severity	Location
Optimization	Informational	ConverterUpgrader.sol L169, ConverterUpgrader.sol L211, ConverterUpgrader.sol L249

Description:

`uint16` is not optimal use for `for loop`.

Recommendation:

The EVM works with 256bit/32byte words. Every operation is based on these base units. If data is smaller, further operations are needed to downscale from 256 bits to 16 bit. It is recommended to use `uint256` instead.

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-13: maxSystemNetworkTokenAmount limit is mutable.

Type	Severity	Location
Implementation	Informational	LiquidityProtection.sol L47, LiquidityProtection.sol L165-L170

Description:

According to the documentation, `The system holdings cannot exceed 500K BNT worth of pool tokens in a pool`. In contrast, `maxSystemNetworkTokenAmount` is not a constant value and can be changed using `setSystemNetworkTokenLimits()` function.

Recommendation:

Clarify this in the documentation and either allow this change or restrict changes to this variable by making it constant.

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-14: Local variable shadowing

Type	Severity	Location
Implementation	Informational	LiquidityProtectionStore.sol L374 LiquidityProtectionStore.sol L401

Description:

`address owner = liquidity.owner;` shadows `owner` variable from `Owned.sol` which is public one and contract inherits from it.

Recommendation:

Change of the local variable and use of OpenZeppelin's `Ownable.sol` contract which implements the same logic but with private `owner` variable.

Alleviation:

Issue is resolved.

BNC-15: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	LiquidityProtection.sol L510, LiquidityProtection.sol L545,LiquidityProtection.sol L651,LiquidityProtection.sol L736, LiquidityPoolV1Converter.sol L279, LiquidityPoolV1Converter.sol L408, LiquidityPoolV1Converter.sol L517, LiquidityPoolV1Converter.sol L521, LiquidityPoolV1Converter.sol L601,ConverterBase.sol L197, ConverterBase.sol L305,

Description:

Within Solidity, unsigned integers are restricted to the non-negative range. As such, greater-than comparisons with the literal `0` are inefficient gas-wise.

Recommendation:

Consider converting the linked comparisons to inequality ones in order to optimize their gas cost.

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-16: Unneccessary if statement

Type	Severity	Location
Optimization	Informational	ConverterUpgrader.sol L174-L184

Description:

If statements can be simplified as `_newConverter.addReserve(ETH_RESERVE_ADDRESS, weight);` is repeated in both cases.

Recommendation:

`if` and `else if` can be merged in one or use only one as the results are always the same i.e., usage of `_newConverter.addReserve(ETH_RESERVE_ADDRESS, weight);` in both cases.

Alleviation:

The team will be fixing the issues in their own timeframe.

BNC-17: Use of two arrays instead of a mapping

Type	Severity	Location
Optimization	Informational	LiquidityPoolV1Converter.sol L263-L297, LiquidityPoolV1Converter.sol L309-L326, LiquidityPoolV1Converter.sol L495-L522

Description:

As the `addLiquidity`, `removeLiquidity` and `verifyLiquidityInput` functions all accept an array of token addresses and an array of amounts that are associated with each other, it makes code harder to review and read.

Recommendation:

Using a mapping would prevent duplicate tokens and would allow for more clear logic to be expressed in the code.

Alleviation:

The team will be fixing the issues in their own timeframe.