CERTIK

Audit Report

Produced by CertiK

for bZx

Dynamic Analysis, Static Analysis, and Manual Review
of BZRX Token and vBZRX vesting mechanism

Contents

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and bZx (the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the “Agreement”). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK’s prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK’s mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance’s BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for **bZx** to discover issues and vulnerabilities in the source code of their **BZRX Token and vBZRX vesting mechanism** as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Testing Summary

SECURITY LEVEL

TIER-ONE
VERY HIGH CONFIDENCE
VERIFIED BY CERTIK

Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by bZx.

This audit was conducted to discover issues and vulnerabilities in the source code of bZx's BZRX ERC token and vesting mechanism.

TYPE	Smart Contract & Vesting Mechanism
SOURCE CODE	https://github.com/bZxNetwork/contractsV2/tree/bzrx_v2
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	July 07, 2020
DELIVERY DATE	July 10, 2020 / Revised July 11, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by the bZx team to audit the design and implementation of their BZRX ERC token smart contract with the enhanced functionality of time stamped balances based on block numbers as well as the vesting mechanism of the said token.

The audited source code link is:

- BZRX & Vesting Source Code: https://github.com/bZxNetwork/contractsV2/tree/6f5386fecd7367b3eafcc48219e76ac913638334

The goal of this audit was to review the Solidity implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

The findings of the initial audit have been conveyed to the team behind the contract implementations and the source code is expected to be re-evaluated before another round of auditing has been carried out.

The new audited source code link is:

- BZRX & Vesting Source Code: https://github.com/bZxNetwork/contractsV2/tree/33308ab93ecc096d62925270cef6d83d726569c6

The bZx team quickly applied the optimization exhibits of the report where sensical to reduce the gas costs of the contract's functions across the board and render it ready for a second round. The second round revealed no types of exploits in the codebase, confirming that the bZx team has once again applied the highest level of care when developing their codebase.

The final audited source code link is:

- BZRX & Vesting Source Code: https://github.com/bZxNetwork/contractsV2/tree/cc39fc93d05bbeade8bea1d191870c3a4f879ea2

Documentation

The sources of truth regarding the operation of the contracts in scope were comprehensive and **aided in our efforts to audit the project**. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the bZx team or reported an issue.

Summary

The codebase of the project is relatively straightforward and the additional functionality that was introduced with regards to a checkpointed balance sheet was implemented based on source code from Aragon One, an established entity of the Ethereum developer community.

Some optimization steps were also **pinpointed but were of negligible importance** and mostly referred to coding standards and inefficiencies. Overall, the code has been developed to the highest standards as evident by the lack of any minor or higher vulnerabilities being identified.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code **to achieve a high standard of code quality and security**.

Conclusion

The bZx team applied our recommendations across the codebase in a very small time frame, highlighting the team's dedication to a secure codebase. We can safely say that there are no further recommendations we could suggest for the state of the codebase at the final iteration of our audit.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Impossible Edge Case	Optimization	Informational	Checkpointing.sol: L56 - L62

[INFORMATIONAL] Description:

The "if-else-if" chain of the aforementioned slides checks for the cases whereby "currentCheckpointTime" is less than "_time", equal to "_time" or larger than "_time". The bZx system utilizes the Solidity variable "block.number" as the checkpoint "time" variable.

Recommendations:

As the "block.number" of Solidity is a sequentially incremental number, the edge case of L60 to L62 that ensures ordering of the list is redundant as it is guaranteed by the EVM. As such, the logical comparison of L58 is also redundant and the code block can be refactored to a single "if-else" clause with the "if" conditional directly checking the "currentCheckpoint.time" instead of creating an in-memory variable.

Alleviation:

After consulting with bZx, it became apparent that the Checkpointing library will not be solely utilized for block.number and as such, it made sense to retain the "if-else-if" chain as is.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Unnamed Return Variables	Code Legibility	Informational	Checkpointing.sol: L76 - L88 (“lastUpdated”) L90 - L102 (“latestValue”)

[INFORMATIONAL] Description:

The functions “lastUpdated” and “latestValue” contain a “return” statement of the value literal zero at the end of their code blocks with a single “if” statement returning a different value.

Recommendations:

If the return variable is named, the last return statement can be safely omitted and a simple assignment would be necessary within the “if” clauses.

Alleviation:

Instead of naming the return variable the return statement of the value literal “0” was completely omitted. This falls in line with our recommendation and results in the same OPCODEs, thus considering this Exhibit dealt with.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	Checkpointing.sol: L83, L97 CheckpointingToken.sol: L191

[INFORMATIONAL] Description:

The lines above conduct a greater-than “>” comparison between unsigned integers and the value literal “0”.

Recommendations:

As unsigned integers are restricted to the positive range, it is possible to convert this check to an inequality “!=” reducing the gas cost of the function.

Alleviation:

Our recommendation was applied to the letter, converting these comparisons to inequality comparisons.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
“while” Conditional	Optimization	Informational	Checkpointing.sol: L137

[INFORMATIONAL] Description:

The conditional currently being imposed by the “while” loop is a greater-than “>” comparison between the variables “high” and “low”. Internally, a “mid” variable is declared which will be at most equal to “high” and never equal to “low” and the only branches that alter the variables “high” and “low” are setting them to a value at or within range of one another.

Recommendations:

It is possible to change the conditional to an inequality as the code within guarantees that a change from the conditional “high > low” evaluating true to the conditional “high < low” evaluating true in a single step cannot occur.

Alleviation:

The conditional was correctly changed to an inequality thus optimizing the loop condition and reducing the gas cost of the Checkpointing library.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Redundant Overlay Functions	Optimization	Informational	CheckpointingToken.sol: L46 - L54 (“balanceOfAt”)

[INFORMATIONAL] Description:

The public function “balanceOfAt” calls the internal function “_balanceOfAt” passing the arguments as is and containing no specialized instructions.

Recommendations:

The function “_balanceOfAt” should be renamed to “balanceOfAt” and be converted to public in place of the existing “balanceOfAt” implementation as it is redundant.

Alleviation:

Our recommendation was followed and the “balanceOfAt” function was replaced by the “_balanceOfAt” function renamed and set to public.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Duplicate “require” Statements	Optimization	Informational	CheckpointingToken.sol: L98, L101

[INFORMATIONAL] Description:

Both of the “require” statements contained above conduct a sanity check against the variables that are meant to be added to and subtracted from whilst the contract itself utilizes safe math operation functions.

Recommendations:

These cases are covered by the statements of L102 and L107 - L110 respectively as the “sub” function internally verifies the subtraction. If the error message is meant to be unique for these cases for an off-chain reason, it would be more sensical to instead adjust the “sub” function to accept a “string” argument that is passed in to the “require” statement instead of a generic “addition-overflow” error message.

Alleviation:

The bZx team decided instead to not use the safe math functions they have defined for the statements L102 and L107 - L110 respectively as these subtractions are guaranteed to never underflow thanks to the “require” checks, nullifying this Exhibit.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Internal Exposed Variable	Optimization	Informational	BZRXToken.sol: L17, L27 - L33

[INFORMATIONAL] Description:

The internal variable “totalSupply_” is exposed via a getter function titled “totalSupply” that contains no special instructions apart from retrieving the variable from storage.

Recommendations:

We advise that the trailing underscore is dropped from the variable name and that it is converted to “public” to have the compiler generate a getter for it automatically instead of creating a user-defined one.

Alleviation:

As this change would require changing the IERC interface library, we concluded with the bZx team that the function should remain as is to avoid unnecessary library changes.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Superfluous “require” Check	Optimization	Informational	BZRXVestingToken.sol: L55

[INFORMATIONAL] Description:

The “require” statement contained on L55 ensures that the constant variable “cliffDuration” is less than the constant variable “vestingDuration”.

Recommendations:

As these variables are defined on the codebase on compile time rather than on the contract’s deployment, we advise this “require” check is omitted. If these variables are instead meant to be set up during the initialization of the contract, we advise that the function signature of “initialize” is altered to accept those two variables as arguments.

Alleviation:

The “require” check was correctly omitted from the “initialize” function body, nullifying this Exhibit.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Redundant Overlay Functions	Optimization	Informational	BZRXVestingToken.sol: L126 - L133 (“totalSupplyAt”)

[INFORMATIONAL] Description:

The public function “totalSupplyAt” calls the internal function “_totalSupplyAt” passing the arguments as is and containing no specialized instructions.

Recommendations:

The function “_totalSupplyAt” should be renamed to “totalSupplyAt” and be converted to public in place of the existing “totalSupplyAt” implementation as it is redundant.

Alleviation:

Our recommendation was followed and the “totalSupplyAt” function was replaced by the “_totalSupplyAt” function renamed and set to public.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Unnamed Return Variable	Code Legibility	Informational	BZRXVestingToken.sol: L152 - L172 (“vestingBalanceOf”)

[INFORMATIONAL] Description:

The function “vestingBalanceOf” internally declares an in-memory variable called “balance” which is subsequently returned at the end of the code block’s execution.

Recommendations:

The return variable can be named in the function signature to omit the last return statement of the function.

Alleviation:

Our recommendation was followed and a return variable named “balance” was implemented reducing the number of LoC in the function.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Internal Exposed Variable	Optimization	Informational	BZRXVestingToken.sol: L37, L194 - L200

[INFORMATIONAL] Description:

The internal variable “totalClaimed_” is exposed via a getter function titled “totalClaimed” that contains no special instructions apart from retrieving the variable from storage.

Recommendations:

We advise that the trailing underscore is dropped from the variable name and that it is converted to “public” to have the compiler generate a getter for it automatically instead of creating a user-defined one.

Alleviation:

The “totalClaimed_” variable was properly renamed to “totalClaimed” and converted to a public function according to our recommendation.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Redundant Ternary Operator	Optimization	Informational	BZRXVestingToken.sol: L237 - L239

[INFORMATIONAL] Description:

The assignment of those lines ensures that the variable “_timestamp” that is assigned to “lastClaimTime_[_owner]” is less than or equal to “_vestingEndTimestamp” in case it exceeds it.

Recommendations:

This type of ternary operator is redundant in the case of the contract as “lastClaimTime_” is accessed only when passed in to the context of the “_totalVested” function which in turn checks that “_lastClaimTime” is higher than or equal to “_vestingEndTimestamp”. As such, it is possible to omit this operator and directly assign the value of “_timestamp” to “lastClaimTime_[_owner]”.

Alleviation:

The conditional ternary operator was omitted according to our recommendation.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Numerical Division Truncation	Mathematical	Informational	BZRXVestingToken.sol: L268

[INFORMATIONAL] Description:

The vesting reward calculation conducted on L268 essentially calculates how much time has elapsed since the vesting period has begun proportionally to the total vesting period. As a multiplication followed by a division is conducted, loss of precision can occur which would result in units of the reward token being locked in the contract.

Recommendations:

These units even if stacked many times across multiple users would amount to a relatively small fiscal impact on the ecosystem of the BZRX token and thus could be considered negligible as any type of accurate truncation tracking mechanism would significantly increase the gas cost of the function and offset the units of BZRX correctly tracked.

Regardless, we would advise that a safety mechanism for retrieving locked up vested rewards from the vBZRX token is created that would be invokable after a significant amount of time has passed beyond the vesting period's end. This would cover both truncated units of BZRX as well as unclaimed vested BZRX due to loss of address ownership.

Alleviation:

The bZx team decided to create such a safety mechanism that would be invokable 1 year beyond the vesting period’s end and would be callable by the owner of the vBZRX contract, completely addressing this Exhibit.