CERTIK

Audit Report

Produced by CertiK

for

March 9^{th}, 2020

CertiK Review Report for bZx

CertiK Eng Team
Mar 9, 2020
Version 2.0.0

Executive Summary

While this Review does not constitute a full audit, in the time allocated for this Review, we have not found any vulnerabilities arising from the changes in the pull request.

Dislaimer

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 Network (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.

As there have been numerous interactions with the Company throughout the entire duration of this audit, and as the codebase target for the audit has evolved over said duration, not all of CertiK's opinions or comments have necessarily made it into this final culmination.

Background

The bZx protocol recently suffered two losses. This first one ocurred at early morning UTC, Feb 15, 2020, and the second early morning UTC, Feb 18.

The first attack was caused by a bug in the function `takeOrderFromiToken` in the base bZx protocol. Since `withdrawalAmount = newLoanAmount` and the loan had an empty data field, the oracle was never consulted, and as a result, the protocol ended up with an under-collaterized loan (with respect to non-manipulated prices), which is undesired behavior.

This logical flaw was fixed in the commit `7cfebd9e289d1f7ee541d5a7556e3f679fa216af` on Feb 17 and is not within the scope of this audit.

The second attack on the other hand did not abuse any logical bugs. Rather, it abused a weak oracle. The bZx protocol used to use Kyber as an on-chain price oracle. Using a very large bZx flash loan, the attacker drove the price of the `sUSD` token up on Kyber using their automated market maker (AMM). She

then proceeded to take a loan, denominated in `ETH` and collateralized by `sUSD`. Since bZx uses Kyber as an on-chain price oracle, the protocol over-valued the collateral. Consequently, it again ended up with an under-collateralized loan, with respect to non-manipulated prices. It is this attack that is meant to be fixed by the following pull request.

Scope of work

The scope of the review was PR #30 in the bZx monorepo, from branch `rs_update` to `development`.

- Link to PR: https://github.com/bZxNetwork/bZx-monorepo/pull/30/files
- `development` commit hash: `c5fdab1eb7e0f158841671c78d324045cb438f3c`
- `rs_update` commit hash: `9a17e7349624e4f298b4d20ad12142c70dff7b2b`

Note: The monorepo constists of many packages, all path references are w.r.t. `/packages/contracts`.

The PR consists of 14 commits with dates ranging from Feb 27 to Mar 5. The PR changes 17 files in the base protocol (`/contracts`) and 5 files in the extension (`/extensions`). It is those files that are are part of the scope of the review.

Assumptions

In this Review, we assumed the operators and admins to be honest. Hence while some changes might lead to a lack of correctness (e.g. some users might not be able to withdraw), such effects are not within the scope. This is because in this emergency case, admins can potentially fix this by e.g. submitting an `onlyWhitelist` transaction themselves *for* the user, or potentially by relaxing some limitations, and redeploying the contracts. What we are interested in are *security* bugs, i.e. those that lead to unexpected, unrecoverable situations such as the ones that unfolded earlier.

Changes

We will now proceed to summarize the changes made in the PR.

Note: For the following, `P._` represents functions in all three touched pToken contracts, namely:

- `PositionTokenLogic` ,
- `PositionTokenLogic_WBTCShort` , and
- `PositionTokenLogicV2` .

Whitelist

- `contracts/modules/LoanHealth_MiscFunctions4.sol` adds a new function `setCallerWhitelist(address,bool)` .
- `contracts/modifiers/Whitelistable.sol` is a new file that adds the `onlyWhitelist` function modifier.

Functions that are now whitelisted:

- `LoanHealth_MiscFunctions4.liquidatePosition`
- `LoanHealth_MiscFunctions4.liquidateWithCollateral`

Owner

Functions that have now been set to `onlyOwner` :

- `LoanTokenLogicV4.borrowTokenFromDeposit`
- `LoanTokenLogicV4.marginTradeFromDeposit`
- `LoanTokenLogicV4_Chai.borrowTokenFromDeposit`
- `LoanTokenLogicV4_Chai.marginTradeFromDeposit`
- `P.donateAsset`
- `P.transferFrom`
- `P.transfer`

Removed functions

The PR removes a lot of both external, public, and internal functions. There are three types of removals:

1. *removing source code*
   Functions can still be called on the proxy (bZx system uses a `targets` router), but will fallback to the default function of the enclosing contract due to the Solidity compiler. We have checked that all of the enclosing contracts have a default function that reverts, rendering these functions uncallable.
2. *nulled*
   Turned into (mathematically) pure functions of the form `(...) -> 0x0`.
3. *disabled*
   Their signature is nulled in `targets` in the proxy, and the default function on the proxy reverts in that case, again meaning they are uncallable.

1. Removing source code

- `LoanMaintenance_MiscFunctions._depositPosition`
- `LoanTokenLogicV4.flashBorrowToken`
- `LoanTokenLogicV4_Chai.flashBorrowToken`
- `P.mintWithEther` (all)
- `P.mintWithToken` (all)
- `P.triggerPosition`
- `P._mintWithToken`
- `P._triggerPosition`
- `PositionTokenLogic.marketLiquidityFor *`
- `PositionTokenLogic_WBTCShort.marketLiquidityFor *`
- `PositionTokenLogicV2._checkTradeSize`

2. Nulled

- `LoanHealth_MiscFunctions.closeLoan`
- `LoanHealth_MiscFunctions.closeLoanForBorrower`

- LoanHealth_MiscFunctions3.closeLoanPartially
- LoanMaintenance_MiscFunctions.changeCollateral
- LoanMaintenance_MiscFunctions.depositPosition
- LoanMaintenance_MiscFunctions.depositPositionForBorrower
- LoanMaintenance_MiscFunctions.withdrawPosition
- LoanMaintenance_MiscFunctions._depositPosition
- LoanMaintenance_MiscFunctions2.changeTraderOwnership
- LoanMaintenance_MiscFunctions2.changeLenderOwnership
- LoanMaintenance_MiscFunctions2.updateLoanAsLender

3. Disabled

All functions from the following contracts have been disabled.

- OrderTaking_MiscFunctions
- OrderTaking_takeLoanOrderAsLender
- OrderTaking_takeLoanOrderAsTrader
- OrderTaking_takeLoanOrderOnChainAsLender
- OrderTaking_takeLoanOrderOnChainAsTrader
- OrderTaking_takeLoanOrderOnChainAsTraderByDelegate
- TradePlacing_ZeroEx

Restricted to EOA

The following functions have been restricted to externally-owned accounts:

- iTokens_loanManagementFunctions.paybackLoanAndClose
- P.burnToEther
- P.burnToToken
- P.depositCollateralToLoan

Custom changes

- LoanMaintenance_MiscFunctions._depositCollateral
  - change in require logic: now it is mandatory that depositToken == collateralToken of loan position

- `LoanHealth_MiscFunctions4._handleRollOver`
  - `owedPerDay` is not recalculated and is set to former amount
- `iTokens_loanManagementFunctions2.extendLoanByInterest`
  - `useCollateral` assumed to be false
- `P.depositCollateralToLoan`
  - cannot use custom depositToken (always uses tradeToken)
- `PositionTokenLogicV2.burnToEther` (with four args)
  - `loanDataBytes` cannot be used
- `PositionTokenLogicV2.burnToToken`
  - `loanDataBytes` cannot be used
- `PositionTokenLogicV2._burnToken`
  - no longer calls `_checkTradeSize`
- `BZxOracle`
  - default `function()` - empty return used to be on `msg.value > 0`, now instead on `gasleft() <= 2300`
  - add new storage `mapping (address => AggregatorInterface) public linkPricesFeeds` that stores Chainlink price feeds
  - new function `setLinkPriceFeedsBatch` that allows to set Chainlink price feed for tokens
  - `_querySaneRate` used to query Kyber, now queries Chainlink
  - `getCurrentMarginAndCollateralSize` used to call `_getExpectedRateCall`, now calls `_querySaneRate`
  - `_getExpectedRateCall` accepts one fewer parameter (`saneRate` assumed to be false)
  - `_trade` - if `sourceToken != destToken && txnData.length > 0`, now calls `_verifyPriceAgreement`
  - new function `_verifyPriceAgreement` that bounds the percentage difference between the local exchange rate and the corresponding Chainlink one.
  - `_checkTradeSize` used to switch based on tokenAddress, now it queries `_querySaneRate` to convert to ETH, and imposes a 1500 eth cap for all tokens

Analysis

Items are labeled CRITICAL, MAJOR, MINOR, INFO, DISCUSSION (in decreasing significance).

We see three types of changes:

1. Limiting the behavior of the protocol. As outlined in `Scope of work / Assumptions`, we are interested in security implications only. We do not see any vulnerabilities arising from these sorts of changes.
2. Eliminating most likely attack vectors / reducing complexity. While these changes make the system less flexible and customizable, we don't see any attack vectors arising.
3. New oracle integration. We have checked the correct implementation of the Chainlink API. In particular, the client code makes two checks after getting the price:

- `price != 0`
- `price >> 128 == 0`
  The first check guarantees that the feed has address is correct and the feed is live.
  The second check is to ensure that the price is bounded by 2^{128}, as well as checking that it's not negative (`latestAnswer()` returns an `int`). We see these checks as reasonable.
  The code now calls Chainlink feed in the following functions that it didn't before:
- `getCurrentMarginAndCollateralSize`
- `getTradeData`
- `getPositionOffset`
- `checkTradeSize`
- as well as all functions that used to call `_querySaneRate` directly before.
  INFO While the working of these functions in the broader scope of the system falls beyond the scope of this Review, the client views this as desired behavior. We don't see any bugs arising out of the integration of the new oracle.

In summary, no vulnerabilities have been found due to the new changes.