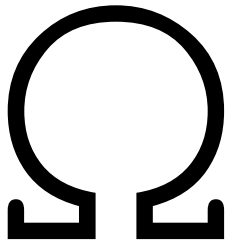
BlinDEX

Final Audit Report

December 13, 2021



Team Omega

Summary

BlinDEX has asked Team Omega to audit the contracts that define the behavior of the BlinDEX contracts.

We found 6 high severity issues - these are issues that can lead to a loss of funds, and are essential to fix.

We classified 5 issues as "medium" - these are issues we believe you should definitely address.

In addition, 34 issues were classified as "low", and 10 issues were classified as "info" - we believe the code would improve if these issues were addressed as well.

Scope of the Audit

The audit concerns the contracts committed here:
https://github.com/BlindexDAO/BlinDEX/commit/c91dca07105229d9cfb2b77e5a0cf8d173abbd8c

And specifically the following contracts and their related tests:

./BdStable/Pools/BdPoolLibrary.sol
./BdStable/Pools/BdStablePool.sol
./BdStable/BDStable.sol
./Bdx/BDXShares.sol
./ERC20/ERC20Custom.sol
./Oracle/MoneyOnChainPriceFeed.sol
./Oracle/WethToWethOracle.sol
./Oracle/FiatToFiatPseudoOracleFeed.sol
./Oracle/BtcToEthOracleMoneyOnChain.sol
./Oracle/IPriceFeed.sol
./Oracle/IMoCBaseOracle.sol

./Oracle/ICryptoPairOracle.sol
./Oracle/OracleBasedCryptoFiatFeed.sol
./Oracle/IOracleBasedCryptoFiatFeed.sol
./Oracle/AggregatorV3PriceFeed.sol
./Staking/StakingRewards.sol
./Staking/StakingRewardsDistribution.sol
./Staking/Vesting.sol

The files in the "uniswap" directory are derived from the Uniswap contracts]. The audit was limited to the changes with respect to the original files in the following files:

./Uniswap/UniswapV2Factory.sol
./Uniswap/UniswapV2ERC20.sol
./Uniswap/UniswapV2OracleLibrary.sol
./Uniswap/UniswapV2Library.sol
./Uniswap/UniswapV2Router02.sol
./Uniswap/UniswapV2Pair.sol
./Uniswap/Interfaces/IUniswapV2Router01.sol
./Uniswap/Interfaces/IUniswapV2ERC20.sol
./Uniswap/Interfaces/IUniswapV2Router02.sol
./Uniswap/Interfaces/IUniswapV2Factory.sol
./Uniswap/Interfaces/IUniswapV2PairOracle.sol
./Uniswap/Interfaces/IUniswapV2Pair.sol
./Uniswap/Interfaces/IUniswapV2Callee.sol

The audit is limited to the submitted solidity code - we did not include any observations about the protocol dynamics in this report.

Resolution

After delivering an intermediate report, the developers addressed most of the issues in https://github.com/BlindexDAO/BlinDEX/commit/1c9b9ac4a0a9f6f6837a442031dbafef9e8e68dc.

We audited the changes and have incorporated our observations in this report. Of the issues we found, all issues marked as “high severity” were resolved, and all those marked as “medium” are either resolved or partially resolved.

The new version added a new contract, SovrynSwapPriceFeed.sol, which defines an Oracle interface to Sovryn Swap. We also reviewed this new contract.

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used several automated analysis tools, including Slither and Remix to detect common potential vulnerabilities. One high severity issue was identified with the automated processes, which we discuss below as issue U1. Some low severity issues, concerning mostly the solidity version setting and functions visibility, were found and we have included them below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

There are a number of issues with the code base in general.

- There are no detailed specifications of what the software is meant to do, and there is almost no in-line documentation in the solidity code.
- Some of the code needs refactoring: there are cases of unused variables and duplicate code that we note below, and variable and function naming styles are not consistently used and make the code harder to read.
- There are no unit tests to quickly test a wide range of limit cases, and test coverage is incomplete as some functions and limit cases are not tested.
- There is no continuous integration - tests are not run automatically, and no coverage tool is configured to check if code is and remains working as expected

Although these issues do not concern the Solidity code directly (and as such are not in the scope of the audit) we believe it is important to address them: they make it difficult to properly audit the code and be confident it is free of bugs, and will make using, testing, and making future changes to the code harder and riskier.

G1. GPL code is published under MIT license [medium] [resolved]

The LICENSE file claims that “this software” is available under MIT license. However, the Uniswap contracts, which you included in a slightly changed form, are published under the GPL license. The source code WETH.sol currently contains references to both MIT as GPL licenses. The two licenses are not compatible, and so you are explicitly in breach of the license conditions of these files.
Severity: Medium
Recommendation: Either use the GPL license for the whole repository, or, if you prefer to maintain the MIT license, explicitly state that the MIT license does not apply to those contracts that are released under the GPL license. This may also be an occasion to acknowledge FRAX and Synthetix, from which code was used as well. (The latter code was published under MIT and so should be ok).
Resolution: This issue was resolved. The custom Uniswap contracts were removed from the code.

G2. Npm audit reports vulnerabilities that should be addressed [medium] [resolved]

Running npm audit reports 122 vulnerabilities (13 low, 42 moderate, 56 high, 11 critical)

Some of these vulnerabilities concern the dependencies of solidity code:

```
@openzeppelin/contracts  >=3.3.0 <3.4.2
Severity: critical
TimelockController vulnerability in OpenZeppelin Contracts  -
https://github.com/advisories/GHSA-fg47-3c2x-m2wr
fix available via `npm audit fix --force`
Will install @openzeppelin/contracts

@openzeppelin/contracts-upgradeable  >=3.3.0 <3.4.2
Severity: critical
TimelockController vulnerability in OpenZeppelin Contracts -
https://github.com/advisories/GHSA-vrw4-w73r-6mm8
fix available via `npm audit fix --force`
Will install @openzeppelin/contracts-upgradeable@3.4.2, which is outside the stated dependency range
node_modules/@openzeppelin/contracts-upgradeablezeppelin/contracts@3.4.2, which is outside the
stated dependency range
node_modules/@open
```

Severity: Medium
Recommendation: Run npm audit --fix to upgrade these dependencies. In particular, upgrade the Open Zeppelin dependency to 3.4.2 - or even consider using the 4.* openzeppelin contracts.
Resolution: This issue was resolved. The Open Zeppelin contracts were upgraded to 3.4.2.

G3. Many roles control critical parameters [low] [partially resolved]

There is a relatively large number of roles in the system that, in malicious hands, can drain the contracts of deposited funds. Here are some examples:

- The bot that is responsible for updating the pseudo-Oracle can manipulate the prices and drain the contracts out of collateral
- The deployer of a BDStablePool and the BDStable contract can upgrade the contract and take of all the collateral it holds

- The owner of the BDStable contract can call transfer_bdx_force to drain the contract of all BDX, she can add a malicious pool (that can drain the system of all funds), or obtain the same result by changing the price oracles, allocating fees, etc.
- The owner of the MoneyOnChainPriceFeed can call setPrecision to manipulate the price and take all the funds
- The owner of the BDXShares contract can mint BDX up to the maximum supply

Each of these accounts have the power to influence or manipulate the system in critical ways. This means that a user interacting with the Blindex contracts must not just trust the code and the RSK blockchain, but also the owners of each of these addresses.
Severity: Low
Recommendation: It is best practice to limit this type of far-reaching powers to one or two accounts (say for an emergency pause and for upgrading the contracts) which require approval of multiple parties through multisig or a DAO, and often subject to time delays.
Specifically, reduce the amount and type of accounts that users need to trust in order to use the system. Consider carefully if these different roles are really needed, and remove them where that is reasonable. Do the same when choosing to deploy a contract as an upgradeable contract, or to leave it immutable. Make sure all "owners" and "admins" are held by multi-sigs that require at least 2 signatures and, where appropriate, pass the transactions through time locks. Migrate away from the "pseudo oracle" as soon as you can, but in any case implement safeguards as we recommend in issue F1.
Resolution: This issue was partially resolved. The README now includes a description of the different accounts responsible, and the bot account has been restricted in power as per the recommendation in F1.

G4. Do not include solidity code released in "alpha" [low] [partially resolved]

The uniswap-lib dependency that is included for the TransferHelper logic was released in "alpha", meaning the developers explicitly marked this code as not yet ready for production. No newer version is available.
Severity: Low
Recommendation: Remove the package dependency altogether and instead use the OpenZeppelin implementation - see issue G5.
Resolution: This issue was partially resolved. The use of the code released in "alpha" was minimized to the use of the FixedPoint library in the UniswapPairOracle contract.

G5. Choose between Open Zeppelin SafeERC20 and Uniswap TransferHelper [low] [resolved]

Currently, two implementations for safeTransfer are being used in the code: that of OpenZeppelin (used in StakingRewards.sol and Vesting.sol) and the TransferHelper contract that was developed by Uniswap (see issue G4).
Severity: Low
Recommendation: Choose a single implementation.
Resolution: This issue was resolved. The use of TransferHelper has been removed in favor of the use of Open Zeppelin SaafeERC20.

G6. Instructions in the README are not correct [low] [resolved]

Following the instructions in the README.md, running the “npm run node” command outputs the following error:

> Error HH8: There's one or more errors in your config file:
>
> * Invalid value {"url":"https://public-node.rsk.co","accounts":["472a082c0ea7300773c6fb27b3b3215807da7cb9ab4ca2ae0763eb5deb10725d","472a082c0ea7300773c6fb27b3b3215807da7cb9ab4ca2ae0763eb5deb10725d",null],"timeout":6000000,"gasPrice":79240000} for HardhatConfig.networks.rsk - Expected a value of type HttpNetworkConfig.

Severity: Low
Recommendation: Provide a correct hardhat config file in the repository, or fix the README.
Resolution: This issue was resolved.

G7. No sanity checks, value limits, and change delays when changing parameters [low] [partially resolved]

Functions for setting parameters of the system (for example, the BdStablePool.setPoolParameters function) contain no checks on the values that are passed to the function. This can be risky in case of mistakes or malicious behaviour on the side of someone with access to the owner address. It also has no delay for the parameters change before they take effect to ensure there is a reasonable response time left to react to unreasonable changes. For reference, here is the list of missing zero-checks from Slither:

- owner_address = _owner_address (contracts/BdStable/BDStable.sol#101)
- treasury_address = _treasury_address (contracts/BdStable/BDStable.sol#102)
- weth_address = _weth_address (contracts/BdStable/BDStable.sol#320)
- weth_address = _weth_address (contracts/BdStable/BDStable.sol#327)
- treasury_address = _treasury_address (contracts/BdStable/BDStable.sol#372)
- collateral_address = _collateral_address (contracts/BdStable/Pools/BdStablePool.sol#100)
- owner_address = _creator_address (contracts/BdStable/Pools/BdStablePool.sol#101)
- weth_address = _weth_address (contracts/BdStable/Pools/BdStablePool.sol#636)
- owner_address = _owner_address (contracts/Bdx/BDXShares.sol#45)
- updater = _updater (contracts/Oracle/FiatToFiatPseudoOracleFeed.sol#17)
- updater = newUpdater (contracts/Oracle/FiatToFiatPseudoOracleFeed.sol#30)
- wethAddress = _wethAddress (contracts/Oracle/WethToWethOracle.sol#19)
- vestingScheduler = _vestingScheduler (contracts/Staking/Vesting.sol#49)
- fundsProvider = _fundsProvider (contracts/Staking/Vesting.sol#50)
- vestingScheduler = _vestingScheduler (contracts/Staking/Vesting.sol#121)
- fundsProvider = _fundsProvider (contracts/Staking/Vesting.sol#133)

Severity: Low
Recommendation: Add checks to make sure the values passed to the setPoolParameters functions are within a reasonable range. We also suggest considering adding a delay to the changes to ensure users and the team have time to react in emergency cases where a mistake was made or the owner's private key was compromised.
Resolution: This issue was partially resolved. Some value checks were added, but not everywhere and with no delays.

G8. Use of precision for fractions is overly complex [low] [not resolved]

The code contains a lot of logic (and variable and function names) that is managing the precision of fractions (i.e. prices, percentages, ratios)

For example, the following constants are used to define various precisions:

BDPoolLibrary.PRICE_PRECISION (1e12)
BDPoolLibrary.COLLATERAL_RATIO_PRECISION (1e12)
BDPoolLibrary.COLLATERAL_RATIO_MAX (1e12)
StakingRewards.LOCK_MULTIPLIER_PRECISION (1e6)
StakingRewards.REWARD_PRECISION (1e18)
StakingRewardsDistribution.BDX_MINTING_SCHEDULE_PRECISON [1e3]
StakingRewardsDistribution.HUNDRED_PERCENT [1e2]

MoneyOnChainPriceFeed.precision [default 1e18]

In some places, a hardcoded value instead of the corresponding constant is used, such as for example in BDStable.sol:363

```
require(wantedCR_d12 >=0 && wantedCR_d12 <=1e12, "CR must be <0;1>");
```

Or in BtcToEthOracleMoneyOnChain.sol:35

```
return amountIn.mul(1e12).div(getPrice_1e12());
```

This is overly complex, and it is easy to make mistakes or lose track.
Severity: Low
Recommendation: Decide on a single value for precision (say 1e12, which is already used in most places) and use that for all fractions in the code (prices, ratios). A consistent usage of these precision values will also allow you to drop the suffixes like _1e12 and _d12 from function- and variable names, which will improve readability, and to simplify constructions like this one in BDStable.sol:182

```
(weth_fiat_pricer.getPrice_1e12()).mul(BdPoolLibrary.PRICE_PRECISION).div(1e12);
```

Resolution: This issue was not resolved. Some places where hardcoded value is used has been substituted for the use of a constant.

G9. Use more recent solidity versions [low] [partially resolved]

The code is currently written and compiled in Solidity 0.6.11. The hardhat config files also contain a reference to 0.8.0, but although some files are compiled using this version, they are not actually used in the deployment process.
Severity: Low
Recommendation: Upgrade in any case to 0.6.12. This release is over a year old - we recommend upgrading all or part of the code base to the latest release (currently 0.8.10).
Resolution: This issue was partially resolved. The contracts were upgraded to solidity version 0.6.12.

G10. Use Ownable contract instead of defining your own [low] [resolved]

Many contracts - BdStable.sol, BdStablePool.sol, BDXShares.sol, StakingRewards.sol, StakingRewardsDistribution.sol, Vesting.sol - each define a “onlyByOwner” modifier, where the latter three contracts use a different method to define ownership than the first three.
Severity: Low

Recommendation: The code can be simplified and it's API rendered more consistent by choosing a single pattern (and not duplicating the code). We recommend using Open Zeppelin's Ownable contract, and the onlyOwner modifier that is defined by OpenZeppelin.
Resolution: This issue was resolved.

G11. Use external instead of public where possible [low] [resolve]

In multiple places the public function visibility is used while the external visibility would be appropriate. Our automatic analysis tools report the following functions that are currently declared public and should be declared external:

- OwnableUpgradeable.renounceOwnership()
- BDStable.initialize(string,string,string,address,address,address,uint256)
- BDStable.getBdStablesPoolsLength()
- BDStable.updateOraclesIfNeeded()
- BDStable.shouldUpdateOracles()
- BDStable.BDX_price_d12()
- BDStable.get_effective_bdx_coverage_ratio()
- BDStable.pool_burn_from(address,uint256)
- BDStable.pool_mint(address,uint256)
- BDStable.pool_claim_bdx(uint256)
- BDStable.pool_transfer_claimed_bdx(address,uint256)
- BdPoolLibrary.calcMint1t1BD(uint256,uint256)
- BdPoolLibrary.calcMintAlgorithmicBD(uint256,uint256)
- BdPoolLibrary.calcRecollateralizeBdStableInner(uint256,uint256,uint256,uint256,uint256)
- BdStablePool.initialize(address,address,address,address,bool)
- BdStablePool.collatFiatBalance()
- BDXShares.mint(address,address,uint256)
- ERC20Custom.increaseAllowance(address,uint256)
- ERC20Custom.decreaseAllowance(address,uint256)
- ERC20Custom.burn(uint256)
- ERC20Custom.burnFrom(address,uint256)
- BtcToEthOracleMoneyOnChain.shouldUpdateOracle()
- WethToWethOracle.shouldUpdateOracle()
- BtcToEthOracleMoneyOnChain.when_should_update_oracle_in_seconds()
- WethToWethOracle.when_should_update_oracle_in_seconds()
- BtcToEthOracleMoneyOnChain.setPrecision(uint8)
- FiatToFiatPseudoOracleFeed.setUpdater(address)
- FiatToFiatPseudoOracleFeed.setPrice(uint256)
- MoneyOnChainPriceFeed.setPrecision(uint8)

- OracleBasedCryptoFiatFeed.getPrice_1e12()
- OracleBasedCryptoFiatFeed.getDecimals()
- StakingRewards.totalSupply()
- StakingRewards.balanceOf(address)
- StakingRewards.lockedBalanceOf(address)
- StakingRewards.withdraw(uint256)
- StakingRewards.withdrawLocked(bytes32)
- StakingRewards.getReward()

Severity: Low
Recommendation: Set function visibility to external whenever the function is not used internally by the contract to save gas and improve code readability.
Resolution: This issue was resolved.

G12. Emit events when parameters change [low] [resolved]

For each parameter that changes, the solidity code should emit an event, so that clients can update with the new information without need to poll the blockchain. Consider emitting an event in the following cases:

- minimumMintRedeemDelayInBlocks = _minimumMintRedeemDelayInBlocks (contracts/BdStable/BDStable.sol#376)
- precision = _precision (contracts/Oracle/BtcToEthOracleMoneyOnChain.sol#49)
- precision = _precision (contracts/Oracle/MoneyOnChainPriceFeed.sol#30)
- vestingRewardRatio_percent = _vestingRewardRatio (contracts/Staking/StakingRewardsDistribution.sol#131)
- vestingTimeInSeconds = _vestingTimeInSeconds (contracts/Staking/Vesting.sol#129)
- period = _period (contracts/Uniswap/UniswapV2Pair.sol#342)
- consultLatency = _consult_leniency (contracts/Uniswap/UniswapV2Pair.sol#346)

Severity: Low
Resolution: This issue was resolved. Appropriate events were added.

G13. Conditions that are always true can be removed [info] [resolved]

In multiple places there are require statements with conditions that are always true.
For example in line 397 of BDStable.sol, there is a check that wantedCR_d12 is >= 0, but since wantedCR_d12 is a uint, we know that it must be >= 0 already. A similar case can be found in StakingRewardsDistribution.sol line 130.
Severity: Info
Recommendation: Remove the tautological clauses of require statements.

Resolution: This issue was resolved.

BDStable/BDStable.sol

B1. Unused variables fiat and UNUSED_PLACEHOLDER_1 [low] [resolved]

On line 29, a variable “fiat” is defined, and on line 38, a variable called “UNUSED_PLACEHOLDER_1”. These variables are never used and should be removed.
Severity: Low
Recommendation: Remove these variables.
Resolution: This issue was resolved.

B2. Use constants for immutable values [low] [not resolved]

On line 109, refresh_cooldown is set in the constructor, but can not be changed.
Severity: Low
Recommendation: Define the value as a constant, for clarity and to save some gas.
Resolution: This issue was not resolved.

B3. Cache price info to save gas and oracle costs [low] [not resolved]

On line 128ff, in the globalCollateralValue function, the value of the total collateral is calculated by looping over all pools, and calculating the value of each pool. For each pool, this value is calculated by requesting the value of the collateral relative to wRBTC, and subsequently query-ing the Money-on-Chain oracle for the price of wRBTC relative to USD, and finally the pseudoOracle for the price of USD relative to the peg.
Severity: Low
Recommendation: Some gas and calls to the Oracle can be saved by caching the RBTC/USD price.
Resolution: This issue was not resolved.

BDStable/Pools/BdPoolLibrary.sol

L1. Duplicated calculation functions calcMint1t1BD and calcMintAlgorithmicBD [low] [not resolved]

The functions calcMint1t1BD and calcMintAlgorithmicBD are completely identical, with different variable names. You could generalize the variable naming and simply remove one of them.
Severity: Low
Recommendation: Remove the duplicated functions in favor of a more generically named function.
Resolution: This issue was not resolved.

L2. recollat_possible calculation is unnecessarily over-complicated [low] [resolved]

The calculation of recollat_possible in line 53 could be simplified by changing the current formula:

> (global_collateral_ratio * bdStable_total_supply - (bdStable_total_supply * effective_collateral_ratio)) / COLLATERAL_RATIO_PRECISION

To be the identical but simpler formula:

> ((global_collateral_ratio - effective_collateral_ratio) * bdStable_total_supply) / COLLATERAL_RATIO_PRECISION

Severity: Low
Recommendation: Use the simplified calculation.
Resolution: This issue was resolved.

BDStable/Pools/BdStablePool.sol

P1. Redemption functions do not honor the minimal output values that users provide [high] [resolved]

A user that calls redeemFractionalBdStable is not guaranteed to receive the amount of tokens as given by the BDX_out_min and COLLATERAL_out_min parameters. In the case where canLegallyRedeem returns false, 90% of the output funds are sent to the treasury.
A user may be unaware of the penalty and lose money on a transaction.
Severity: High

Recommendation: We strongly recommend removing the 90% penalty logic, also in the light of issue P2. If you decide to keep it, we recommend updating the redeem functions to ensure the 90% penalty is taken into account when comparing the “out_min”-parameters. If you need to keep the logic as is, at least change the names of the parameters, so that users are warned that they may receive only 1/10th of the minimal output they specified.
Resolution: This issue was resolved. The penalty logic was removed.

P2. Redeem penalty of 90% is unenforceable [medium] [partially resolved]

Users that call one of the various redeem functions (redeem1t1BD, redeemAlgorithmicBdStable, redeemFractionalBdStable) within minimumMintRedeemDelayInBlocks after calling one of the mint functions, will receive a 90% penalty.
The current implementation of the penalty is however not enforceable, since a user can redeem within the delay period by transferring the tokens to a different account they own, and redeem from that account.
Severity: Medium
Recommendation: We recommend removing the 90% penalty logic.
Resolution: This issue was partially resolved. The penalty logic has been removed - instead, now, the transaction will revert if the same account calls the function a second time in the given time period. However, the limitation is easily circumvented by transferring tokens to a different account.

P3. Unused duplicated variable collateral_address [low] [resolved]

The collateral_address is not used in the contract, and stores the same value as the collateral_token variable.
Severity: Low
Recommendation: Remove the collateral_address from the codebase and make the collateral_token visibility public, which will save some storage and gas costs.
Resolution: This issue was resolved.

P4. Unused variable collateral_decimals [low] [resolved]

The collateral_decimals variable is not used in the contract. It’s only used in line 104, but removing that variable and changing line 104 by getting the decimals directly there would allow to remove this unused variable and save the gas cost of writing to the permanent storage and reading from it.
Severity: Low

Recommendation: Remove the collateral_decimals state variable from the codebase and update line 104 to get the decimals from the collateral_token directly there.
Resolution: This issue was resolved. Collateral_decimals is now used only to calculate the missing_decimals variable and is not stored by itself.

P5. Unnecessary variable recollat_fee [low] [not resolved]

The contract defines two state variables, recollat_fee and bonus_rate, which on line 551 influence the amount of bdx a user receives when recollaterizing the protocol. One of these variables (and the corresponding setter logic) can be removed.
Severity: Low
Recommendation: Remove the recollat_fee and just adjust the bonus rate down when appropriate.
Resolution: This issue was not resolved.

P6. Contract will fail to initialize with ERC20 that do not define decimals [low] [resolved]

From the ERC20 specifications of the decimals getter:

> "OPTIONAL - This method can be used to improve usability, but interfaces and other contracts MUST NOT expect these values to be present."

However, in line 103 there is a call to this decimal function which does expect the existence of that method, so the contract will not be usable with ERC20 tokens that do not have the (optional) decimals defined.
Severity: Low
Recommendation: Do not depend on the "decimals()" function - either pass the decimals explicitly as a parameter to the initialize function, or (probably better) do not "correct" for the decimals value at all in the code.
Resolution: This issue was resolved. decimals() is no longer called.

P7. availableExcessCollatDV should be in the BDStable contract [low] [resolved]

The function availableExcessCollatDV() uses only state variables from the DBStable contract, while it uses no variables from the BDStablePool contract. The code could be simplified and become more gas efficient by moving the availableExcessCollatDV function to the BDStable contract so that the calls to external contracts in the BDStablePool will be reduced from 3 to 1 call.
Severity: Low

Recommendation: Move the availableExcessCollatDV function to the BDStable contract to save gas.
Resolution: This issue was resolved.

P8. Verify that _collateral_wrapping_native_token is correct before running the code and avoid using assert [low] [resolved]

In multiple places there is an:

```
assert(is_collateral_wrapping_native_token)
```

command which will fail if the user specified useNativeToken as true. Using assert should be used only for critical code mistakes, and not for user errors, as it spends all the gas of the transaction. Also, the check should be before the rest of the code, so that a user that provides an invalid value saves some gas by failing early instead of wasting gas on the function logic before failing.
Severity: Low
Recommendation: Use require instead of assert to save gas and follow general Solidity guidelines, and move the check to the top to ensure it fails early if the user specified wrong useNativeToken value. Also in the receive function we recommend to use require instead of assert.
Resolution: This issue was resolved.

P9. A user may forfeit funds when calling redeem1t1BD [high] [resolved]

If the effective collateral ratio is lower than 1, then a user that calls redeem1t1BD() to redeem her BDStable tokens will receive collateral based on the value of the effective collateral rate. If instead she calls redeemFractionalBdStable(), she will receive the same amount of collateral, but will receive BDX tokens as well. In this case, the user would lose money.
Severity: High
Recommendation: Remove the redeem1t1BD function (see also P10).
Resolution: This issue was resolved. The 3 mint and redeem functions are now combined as a single mint function and a single redeem function.

P10. Replace the 3 mint and redeem functions with a single function [low] [resolved]

The 3 minting and redeeming types could be replaced by a single mint and a single redeem functions, with the 1 to 1 and algorithmic minting and redeeming being edge cases of the fractional type. This could reduce a lot of code duplication that exists currently, therefore reduce the possibility of errors (P9, by reducing code complexity.

Severity: Low
Recommendation: We recommend to remove the functions mint1t1BD, mintAlgorithmicBdStable, redeem1t1BD and redeemAlgorithmicBdStable: these functions implement edge cases of mintFractionalBdStable and redeemFractionalBdStable but add no different functionality (except for the difference noted in P9, which we believe to be an error). Any difference in gas consumption can be mitigated by inserting appropriate checks for the limit cases in these functions.
Resolution: This issue was resolved. The 3 mint and redeem functions are now combined as a single mint function and a single redeem function.
Resolution: This issue was resolved. The 3 mint and redeem functions are now combined as a single mint function and a single redeem function.

P11. Move require in line 340 to line 321 to save gas [low] [resolved]

On line 340 there's a check to make sure the minimum amount of tokens to send from the redeem operation has been reached. It is better to do such checks as early as possible to save gas in case the check reverts.
Severity: Low
Recommendation: Do this check right after computing the relevant variables needed for it to save gas and fail early if the conditions do not meet.
Resolution: This issue was resolved.

P12. Use SafeMath [info] [resolved]

In the subtraction on line 567, it is better to use SafeMath to avoid potential overflow errors

```
msg.value - collateral_units_precision
```

Severity: Info - the current calculations actually guarantee overflow will not happen
Recommendation: Use SafeMath consistently where appropriate.
Resolution: This issue was resolved.

P13. Misleading require check at line 195 [info] [resolved]

Since globalCR is capped at BdPoolLibrary.COLLATERAL_RATIO_MAX, it can never be higher. Thus the require should check equality (==) not equal or greater than (>=) and the error message should be updated accordingly.
Severity: Info
Recommendation: Change the check to == and update the error message.
Resolution: This issue was resolved.

Bdx/BDXShares.sol

S1. Remove bdstables variable [low] [resolved]

The bdstables variable that is defined on line 24 is private and never actually used, so it can be removed.
Severity: Low
Recommendation: Remove the variable.
Resolution: This issue was resolved. The bdstables variable is now public as future contract integrations might make use of it.

Erc20/ERC20Custom.sol

E1. __gap variables is not of the right size [low] [resolved]

On line 286, a __gap variable is defined:

```
uint256[45] private __gap;
```

This variable is part of Open Zeppelin's contract upgrade mechanism (see https://docs.openzeppelin.com/contracts/4.x/upgradeable) - it reserves space for storing values of state variables in a future upgraded contract. The idea is to reserve a total of 50 slots. This is why the size of the __gap variable depends on how many other state variables are defined in the contract - and so it should be redefined to the correct value when it is used in the BDXShares and BDStable contracts.
Severity: Low
Recommendation: Either remove the __gap value altogether, or set it to a better value in BDShares and BDStable
Resolution: This issue was resolved. The __gap variable was removed.

E2. burnFrom and _burnFrom implement the same logic [info] [resolved]

The function _burnFrom seems to serve the same purpose as burnFrom, with the only difference that the function is marked internal.
Severity: Info
Recommendation: Remove _burnFrom function definition.
Resolution: This issue was resolved.

E3. Fix compiler warnings [info] [resolved]

The compiler outputs a number of warnings that would be elegant to fix:

```
contracts/ERC20/ERC20Custom.sol:393:35: Warning: Unused function parameter. Remove or
comment out the variable name to silence this warning.
   function _beforeTokenTransfer(address from, address to, uint256 amount) inter ...
                                 ^----------^

contracts/ERC20/ERC20Custom.sol:393:49: Warning: Unused function parameter. Remove or
comment out the variable name to silence this warning.
 ... _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual  ...
                                    ^--------^

contracts/ERC20/ERC20Custom.sol:393:61: Warning: Unused function parameter. Remove or
comment out the variable name to silence this warning.
 ... Transfer(address from, address to, uint256 amount) internal virtual {c_0xc3c24563(0x ...
                                    ^------------^

contracts/ERC20/ERC20Custom.sol:393:5: Warning: Function state mutability can be restricted
to pure
   function _beforeTokenTransfer(addre ... 65d94bce6e9658ebf); /* function */
   ^ (Relevant source part starts here and spans across multiple lines).
```

Severity: Info
Resolution: This issue was resolved. _beforeTokenTransfer was removed, which removed the related compiler errors.

Oracle/ICryptoPairOracle.sol

O1. Function when_should_update_oracle_in_seconds is not used [info] [resolved]

The when_should_update_oracle_in_seconds() function is not used anywhere in the solidity code, and in some some of its implementations (for example in the WethToWethOracle) returns 1e12, which does not seem to be a reasonable return value.
Severity: Info
Recommendation: Consider removing this function.
Resolution: This issue was resolved. The function was kept as it is used in the front end, but was changed to external and, where irrelevant, the function now returns type(uint256).max.

Oracle/FiatToFiatPseudoOracleFeed.sol

F1. Implement safeguards on Oracle updates [medium] [resolved]

The FiatToFiatPseudoOracleFeed contract provides essential information for the system. The intention is that this oracle will be updated by an external bot, which makes it one of the weakest links in the system (see also issue G2). Currently, the update function has no safeguards beyond the check of the address calling the update function, which gives that address significant power to manipulate the system.
Severity: Medium - we believe this oracle is one of the weakest points of the system, and safeguards are needed to ensure there would be time to intervene if the bot is somehow compromised.
Recommendation: In the current situation, no safeguards are in place - we recommend implementing some simple checks on the way that the oracle is updated - for example, by not allowing the price to change with more than 1% each 24 hours (which seems a reasonable value given that the oracle reports the relative price of two stable coins).
Resolution: Now there is both an owner and an updater role, where the updater is limited to a maximum change per day (by default 1%). However, the owner still has the ability to make any changes to the oracle, which is a potential security concern to the system.

Oracle/MoneyOnChainPriceFeed.sol

M1. Remove setPrecision function [low] [resolved]

There does not seem to be much point to changing the precision of the oracle after it has been deployed in any case, and removing the function would reduce the attack surface of the system and allow you to simplify the code by not inheriting from Ownable.
This issue also applies to BtcToEthOracleMoneyOnChain.sol
Severity: Low .
Recommendation: Remove the setPrecision function and do not inherit from Ownable.
Resolution: This issue was resolved. The MoneyOnChainPriceFeed contract was removed.

Oracle/SovrynSwapPriceFeed.sol

Y1. setTimeBeforeMustUpdate should have reasonable minimum limit [low] [resolved]

If the value of the setTimeBeforeMustUpdate variable is too low the price function will always revert which could get the system temporarily stuck.

Severity: Low
Recommendation: Add a reasonable minimum value limit for the setTimeBeforeMustUpdate variable in the constructor and its setter.
Resolution: The issue was resolved

Staking/StakingRewards.sol

R1. getReward will fail after being called 128 times [high] [resolved]

The function getReward calls stakingRewardsDistribution.releaseReward, which in turn calls vesting.schedule. The latter function will revert after it is called more than 128 times by the same account.
Note that 128 is an absolute, lifetime, limit. A staking contract will run an updateReward-getReward cycle daily, or weekly (similar to what yearn.finance does) will quickly hit this limit. In that case, this account will not be able to retrieve her rewards, now or in the future.
Severity: High - a user will not be able to claim her rewards, which will remain locked in the contract forever.
Recommendation: The issue can be somewhat mitigated by freeing memory by deleting vesting schedules that are fully vested, as we describe in issue V1. Implementing the change we recommend in that issue will allow the beneficiary to simply wait until one of their vesting schedules is fully vested to free a slot and claim her rewards.
To fully solve the issue, remove the limit of 128 vesting schedules, but instead add a range argument to the Vesting.claim(...) function that specifies which schedules should be claimed, so the user can claim her rewards in separate transactions should the function run out of gas.
Resolution: The logic for claiming rewards has changed, and the “getReward” function was removed.

R2. withdrawLocked can run out of gas [high] [resolved]

On line 275 in the withdrawLocked function, there is a loop over the items in the array lockedStakes[msg.sender] to search for a particular entry. If the array is large and the wanted entry is towards the end of the array, this loop may run out of gas, causing withdrawLocked to fail.
Severity: High - a user that creates such a stake will not be able to withdraw her funds.
Recommendation: The loop is not needed. Instead, do not store the stakes in an array at all, but define lockedStakes as a mapping(bytes 32 => LockedStake) that maps the kek_id to a LockedStake instance, and use the kek_id to identify which lockedStake to withdraw.
Resolution: This issue was resolved. A from and to range parameters were added to ensure too large array will not cause the function to fail if proper range is specified.

R3. setRewardsDuration requirements are hard to fulfill [low] [resolved]

Setting the rewardsDurationSeconds variable with the function setRewardsDuration is quite hard to fulfill, as it requires that block.timestamp > periodFinish. However, periodFinish is reset to a time in the future each time updateReward is called, which is on almost every interaction. This means that the reward can only be set in a very short window of time - between the time the previous period ended but before anyone else interacted with the contract.
Severity: Low
Recommendation: Consider making the rewardsDurationSeconds not settable at all.
Resolution: This issue was resolved. The requirement has been removed and the owner can now change the rewardsDurationSeconds variable at any time.

R4. Remove unused DeploymentTimestamp variable [low] [resolved]

The DeploymentTimestamp variable is private and unused, remove it from the code.
Severity: Low
Recommendation: Remove DeploymentTimestamp from the code.
Resolution: This issue was resolved.

R5. Use safeTransfer instead of transfer [low] [resolved]

In line 403 transfer is used to send tokens, but using safeTransfer is recommended when possible.
Severity: Low
Recommendation: Use safeTransfer.
Resolution: This issue was resolved.

R6. Duplicate code in stakeLocked and lockExistingStake [info] [not resolved]

The functions stakeLocked() and lockExistingStake() share most of their code.
Severity: Info
Recommendation: Refactor and remove duplicate code.
Resolution: This issue was not resolved.

Staking/StakingRewardsDistribution.sol

D1. Declare unchangeable variables as constants [low] [not resolved]

In the initialize() function, a number of variables are set that later are not changeable, such as BDX_MINTING_SCHEDULE_YEAR_NN and EndOfYear_NN.
Severity: Low
Recommendation: Define these variables as constants, to save some gas and to have more explicit code.
Resolution: This issue was not resolved.

D2. resetRewardsWeights can run out of gas [low] [resolved]

The contract keeps an array of stakingRewardsAddresses, which can be extended (by the owner) using registerPools, which will add new elements to the array. The array can grow indefinitely, and if it is large enough, the loop in line 108 in resetRewardsWeights will run out of gas and revert.
The function resetRewardsWeights is presumably only there for convenience: the distribution can be reset to 0 for each user by using the registerPools function.
Severity: Low
Recommendation: Remove the resetRewardsWeights functions from the code. This will also allow you to remove the stakingRewardsAddresses array, which will save some gas.
Resolution: This issue was resolved. The resetRewardsWeights function was removed.

D3. Do division last to avoid rounding errors [low] [resolved]

On line 86:

```
uint256 bdxPerSecond = yearSchedule
        .div(365*24*60*60)
        .mul(stakingRewardsWeights[_stakingRewardsAddress])
        .div(stakingRewardsWeightsTotal);
```

It is better to do the division by the number of seconds in the year as the last step in the calculation, which will diminish the rounding error
Severity: Low
Resolution: This issue was resolved.

Staking/Vesting.sol

V1. Repack array after deleting an element [low] [resolved]

On line 81, when a schedule is fully vested, it is deleted from the userVestingSchedules array:

```
delete userVestingSchedules[i];
```

This however will free some memory, but it will not change the length of the array - specifically, no new slot will be freed when the limit of 128 vesting schedules is met. See also issue R128.
Recommendation: Re-pack the array, for example using the same pattern as in BdStable.removePool().
Severity: Low
Resolution: This issue was resolved.

Uniswap/UniswapV2Pair.sol

U1. DOMAIN_SEPARATOR isn't properly set [high] [resolved]

The DOMAIN_SEPARATOR state variable is not set to an appropriate value on initialization. This breaks the EIP-2612 specifications, that say:

> "The DOMAIN_SEPARATOR should be unique to the contract and chain to prevent replay attacks from other domains, and satisfy the requirements of EIP-712, but is otherwise unconstrained."

This can lead to several different problems. First, as mentioned in the spec, the signature could potentially be replayed between different pairs and used to steal funds. Second, since the domain separator is empty, it doesn't comply with the ERC-721 and the EIP-2612 standards, which could result in problems with MetaMask and other wallets for signing the transactions.
Severity: High
Recommendation: Refer the constructor of the UniswapV2ERC20 contract for the correct value. Please also consider issue U4, which we think is at the root of this issue.
Resolution: This issue was resolved. The custom UniswapV2Pair contract was removed from the code.

U2. Users could lose 90% of the tokens to the treasury in a griefing attack [high] [resolved]

The swap function transfers 90% of the swap value to a treasury contract if the swap is called before the minimum delay has passed.
An attacker can front-run a swap transaction from a user with a tiny swap to the “to” address that the user has specified. This will cause the counter of the minimum delay to be set for the victim, and if her transaction is mined before the minimum delay passed, it will send 90% of the swap value to the treasury address.
Severity: High
Recommendation: We recommend in U3 to remove the penalty logic altogether. If you must have it, set the counter based on the message sender instead of the receiving address.
Resolution: This issue was resolved. The custom UniswapV2Pair contract was removed from the code.

U3. Users could lose 90% of the tokens calling swap before minimum delay has passed [medium] [resolved]

The swap function has been modified to transfer 90% of the swap to a treasury contract if the swap is called before the minimum delay has passed. The swap function does not respect its “promise” to return at least “amount0Out” and “amount1Out” of tokens to the “to” address. This constitutes a significant divergence from what a user or a developer would normally expect from a contract that implements the uniswap interface, and may cause loss of funds with users or other contracts unintentionally swapping before the minimum delay has passed.
In any case, an end-user will typically not call the swap() function directly, but rather as part of a chain of swap calls triggered by the UniswapV2Router, and these functions expect the amountOut values to be respected, and will fail if they do not.
Severity: Medium
Recommendation: We recommend removing the 90% penalty logic: the penalty does not seem to preclude any potentially malicious behavior (as one can avoid the penalty and call the swap function from a second address within the same period). If the penalty is kept, then we believe the swap function should be modified to respect its promise of returning “amount0Out” and “amount1Out” tokens to the “to” address.
Resolution: This issue was resolved. The custom UniswapV2Pair contract was removed from the code.

U4. UniswapV2Pair.sol code is heavily and unnecessarily modified [low] [resolved]

The UniswapV2Pair code contains code from several different files from the uniswap code base. Specifically, the inheritance from UniswapV2ERC20 was removed, and its code was duplicated. Also an oracle functionality has been added into the contract, while the oracle could, and likely should be, implemented as a separate contract.

Severity: Low

Recommendation: We recommend to inherit code from the uniswap contracts and to override those functions that need to be changed, rather than copying the code and changing it. This will make it explicit where your code deviates from the uniswap code, and lowers the possibility of accidental mistakes. We also recommend you move the Oracle code into a separate contract, rather than integrating it in the uniswap contract itself.

Resolution: This issue was resolved. The custom UniswapV2Pair contract was removed from the code.

U5. Duplicated import of the IUniswapV2Factory [info] [resolved]

The import of the IUniswapV2Factory is duplicated in lines 14 and 16.

Severity: Info

Recommendation: Remove one of the imports to avoid duplication.

Resolution: This issue was resolved. The custom UniswapV2Pair contract was removed from the code.

U6. Avoid rounding errors when calculating penalties [info] [resolved]

On lines 219ff and 230ff, the payment is split into two parts. A similar pattern is used in BdStablePool.sol, one lines 270ff and 438ff:

```
first_amount = original_amount.div(10);
second_amount = original_amount.mul(9).div(10);
```

A better pattern to use, that avoids rounding errors and guarantees that first_amount + second_amount = original_amount is:

```
first_amount = original_amount.div(10);
second_amount = original_amount - first_amount;
```

Severity: Info

Resolution: This issue was resolved. The custom UniswapV2Pair contract was removed from the code.

Uniswap/UniswapV2Library.sol

N1. Wrong reference to original code [info] [resolved]

The comment references the UniswapV2 Core (https://github.com/Uniswap/v2-core) repository as the source of the file, but the correct repo is the UniswapV2 Periphery (https://github.com/Uniswap/v2-periphery/).
Severity: Info
Recommendation: Update the comment to the correct base repo. Note that this is true also for the IUniswapV2Router01, IUniswapV2Router02, and UniswapV2OracleLibrary.sol files.
Resolution: This issue was resolved. The custom UniswapV2Library contract was removed from the code.