#HashEx
BLOCKCHAIN SECURITY

Fringe Finance

smart contracts
final audit report

June 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Fringe Finance team (previously Bonded Finance) to perform an audit of their smart contracts. The audit was conducted between 2021-12-16 and 2021-12-26.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available in the private GitHub repository @bonded-finance/primary-smart-contracts after the 2a2765c commit. The whitepaper is available on the team's website.

Update: Fringe Finance team responded to the audit. The recheck was performed after the 5041f67 commit.

Audited contracts are deployed to the Ethereum mainnet via TransparentUpgradeableProxy from OpenZeppelin:

PrimaryIndexToken proxy 0x46558DA82Be1ae1955DE6d6146F8D2c1FE2f9C5E (masterCopy),

fUSDC (BLendingToken) proxy 0x9fD0928A09E8661945767E75576C912023bA384D (masterCopy),

Bondtroller proxy 0x9a26929352095f6Cf7a941914669928eBBA8543D (masterCopy),

JumpRateModelV2Upgradeable proxy 0x31210fB9b20335416455636A78B4d65CF94d028C (masterCopy), PriceProviderAggregator proxy 0x6f1C5BDc02a3047cD6394787595a4eC1b3863540 (masterCopy),

BackendPriceProvider proxy 0x22d55e49234D0310f48d3300e43Eeb3aC93C0811 (masterCopy),

chainlinkPriceProvider proxy 0x963d80cA1a35F402bea95C3013e115C5c9a7BAe1 (masterCopy),

primaryLendingPlatformProxyAdmin 0xfB872a364E63950f9847a39202Bb4d1C07534466.

2.1 Summary

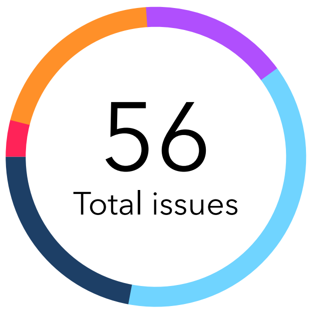
Project name	Fringe Finance
URL	https://fringe.fi
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
PrimaryIndexToken	`0x46558DA82Be1ae1955DE6d6146F8D2c1FE2f9C5E`
UniswapPathFinder	
BToken	
BEther	
Comp	

Bondtroller	0x9a26929352095f6Cf7a941914669928eBBA8543D
BPrimaryIndexToken	
PriceProviderAggregator	0x6f1C5BDc02a3047cD6394787595a4eC1b3863540
BLendingToken	0x9fD0928A09E8661945767E75576C912023bA384D
BackendPriceProvider	0x22d55e49234D0310f48d3300e43Eeb3aC93C0811
ChainlinkPriceProvider	0x963d80cA1a35F402bea95C3013e115C5c9a7BAe1
UniswapV2PriceProvider	
PrimaryLendingPlatformProxyAdmin	0xfB872a364E63950f9847a39202Bb4d1C07534466
ErrorReporter	
JumpRateModelV2	
JumpRateModelV2Upgradeable	0x22d55e49234D0310f48d3300e43Eeb3aC93C0811
Exponential	
ExponentialNoError	
BondtrollerStorage	
BErc20	
InterestRateModel	
UniswapV2Library	
PriceOracle	
CarefulMath	
SimplePriceOracle	

3. Found issues



●	Critical	2 (4%)
●	High	11 (20%)
●	Medium	9 (16%)
●	Low	21 (38%)
●	Info	13 (22%)

C1. PrimaryIndexToken

ID	Severity	Title	Status
C1-01	● Critical	borrow() logic	✓ Resolved
C1-02	● High	Removing project and lending tokens	✓ Partially fixed
C1-03	● High	liquidate() logic	✓ Resolved
C1-04	● High	Admin and Moderator privileges	✓ Partially fixed
C1-05	● High	Change bLendingToken	✓ Acknowledged
C1-06	● High	Wrong decimals in calculations	✓ Partially fixed
C1-07	● High	healthFactor checking	✓ Resolved
C1-08	● High	Locked project tokens	✓ Acknowledged
C1-09	● High	healthFactorForPosition/healthFactor calculation	✓ Partially fixed
C1-10	● Medium	Pausable functions	✓ Resolved

C1-11	● Medium	Redundant separation of functions	✓ Resolved
C1-12	● Medium	Flash-loan vulnerability	✓ Resolved
C1-13	● Low	Error messages	✓ Resolved
C1-14	● Low	Liquidation threshold error	✓ Acknowledged
C1-15	● Low	Gas optimizations	✓ Partially fixed
C1-16	● Low	Dubious calculations	✓ Resolved
C1-17	● Low	Excessive inheritance	✓ Resolved
C1-18	● Low	Gas limit issue	✓ Acknowledged
C1-19	● Info	Token support	✓ Acknowledged
C1-20	● Info	Lack of events	✓ Partially fixed
C1-21	● Info	Lending tokens	✓ Resolved
C1-22	● Info	Interest uniform distribution	✓ Resolved
C1-23	● Info	Admin role	✓ Acknowledged
C1-24	● Info	One token can be lending and project at the same time	✓ Acknowledged
C1-25	● Info	No check on user's input data	✓ Resolved

C2. UniswapPathFinder

ID	Severity	Title	Status
C2-01	● Low	Excessive reads from storage	✓ Resolved

C2-02	Low	Switched arguments in uniswapPositionCap()	Resolved
C2-03	Info	Redundant code	Resolved

C3. BToken

ID	Severity	Title	Status
C3-01	High	Allowed transfers	Resolved
C3-02	Low	Admin overpower	Acknowledged
C3-03	Low	Underflow in repayBorrowFresh()	Acknowledged
C3-04	Info	Redundant event	Resolved

C4. BEther

ID	Severity	Title	Status
C4-01	High	Allowed transfers	Resolved
C4-02	Medium	Possible ETH locking	Resolved
C4-03	Medium	Max borrow logic	Resolved
C4-04	Low	Native transfers	Resolved

C5. Comp

ID	Severity	Title	Status
C5-01	● Info	Name and symbol variables	✓ Resolved

C6. Bondtroller

ID	Severity	Title	Status
C6-01	● Low	Admin overpower	✓ Resolved
C6-02	● Info	Typo	✓ Resolved
C6-03	● Info	Inconsistent comment	✓ Resolved

C7. BPrimaryIndexToken

ID	Severity	Title	Status
C7-01	● Medium	Contract is inoperable	✓ Resolved
C7-02	● Low	PrimaryIndexToken address update	✓ Resolved

C8. PriceProviderAggregator

ID	Severity	Title	Status
C8-01	● Low	AccessControl misuse	✓ Acknowledged

C9. BLendingToken

ID	Severity	Title	Status
C9-01	● Medium	setReserveFactor() function doesn't check input data	✓ Resolved
C9-02	● Low	PrimaryIndexToken address update	✓ Acknowledged
C9-03	● Low	Misuse of AccessControl	✓ Acknowledged

C10. BackendPriceProvider

ID	Severity	Title	Status
C10-01	● Medium	Oracle is inoperable	✓ Acknowledged
C10-02	● Low	AccessControl misuse	✓ Acknowledged

C11. ChainlinkPriceProvider

ID	Severity	Title	Status
C11-01	● Medium	Uint overflow chance	✓ Acknowledged
C11-02	● Low	AccessControl misuse	✓ Acknowledged
C11-03	● Info	Inconsistent comment	✓ Resolved

C12. UniswapV2PriceProvider

ID	Severity	Title	Status
C12-01	● Critical	Flash-loan vulnerability	✓ Acknowledged
C12-02	● Low	Division before multiplication	✓ Acknowledged
C12-03	● Low	Lack of safety checks	✓ Acknowledged
C12-04	● Low	AccessControl misuse	✓ Acknowledged

C13. PrimaryLendingPlatformProxyAdmin

ID	Severity	Title	Status
C13-01	● High	Planned upgrade can't be cancelled	✓ Resolved

4. Contracts

C1. PrimaryIndexToken

Overview

Main contract of the Fringe Finance system. Keeps records of the lending and project tokens, implements borrow, repay and liquidate functions.

Issues

C1-01 borrow() logic ● Critical ✓ Resolved

The `borrow()` function can be called multiple times for the same deposit of project tokens, i.e. 1 deposit of `X` value can be used to take up to `N*X` of the borrowed amount, where N is the number of lending tokens.

Recommendation

Consider refactoring the logic, e.g. using the `balanceOfPit()` instead of `balanceOfPitPosition()`. We strongly recommend testing the logic after refactoring.

C1-02 Removing project and lending tokens ● High ✓ Partially fixed

Admin can remove any project or lending token from the contract. If in the contract there are deposits for some project token and this project token is used as a collateral for some lending tokens and if admin removes this project token, this will disrupt the contract's logic. The same problem in case admin removes lending token

Recommendation

The code in `removeProjectToken()` function should look like this

```
function removeProjectToken(
    uint256 _projectTokenId
) public onlyAdmin isProjectTokenListed(projectTokens[_projectTokenId]) {
    address projectToken = projectTokens[_projectTokenId];
    delete projectTokenInfo[projectToken];

    require(totalDepositedProjectToken[projectToken] == 0);

    projectTokens[_projectTokenId] = projectTokens[projectTokens.length - 1];
    projectTokens.pop();
}
```

Also, the code in `removeLendingToken()` function should look like this

```
function removeLendingToken(
    uint256 _lendingTokenId
) public onlyAdmin isLendingTokenListed(lendingTokens[_lendingTokenId]) {
    address lendingToken = lendingTokens[_lendingTokenId];
    delete lendingTokenInfo[lendingToken];

    uint256 length = projectTokens.length;
    for(uint256 i = 0; i < length; ++i) {
        require(totalBorrow[projectTokens[i]][lendingToken] == 0);
    }
    require(IERC20Upgradeable(lendingToken).totalSupply() == 0);

    lendingTokens[_lendingTokenId] = lendingTokens[lendingTokens.length - 1];
    lendingTokens.pop();
}
```

Moreover, pause for lending and project tokens should be separated for deposit and withdraw. Because if there are only one pause for deposit and withdraw, users will always have possibility to deposit tokens in case the owner wants them to withdraw only.

Update

In function `removeLendingToken()` there is no check that total supply of lending token is zero.

Team response

Fringe Finance intends to proceed with this current state and address the issue as follows:

Collateral assets

- To (effectively) delist a collateral asset we will set Maximum Borrowing Capacity and LVR to ~0 (in this order: initially MBC, then LVR)
- Set MBC to 0 initially to wean borrowers off the asset.
- We will then later set LVR to 0 once borrow volume has attrited
- Of course, depending on the context, given sometimes waiting to set LVR to zero may have security solvency implications (potentially.)
- Setting LVR to zero will have the effect of making all open borrow positions secured by said collateral asset immediately liquidatable.

Capital assets

- We are not currently able to remove capital assets, but the impact of this is relatively minor as they do not pose systemic risks to the platform.
- As distinct from collateral assets.
- A temporary solution is to simply hide the frontend for making deposits in the event of an exploit/issue until it is disabled at a SC level.

Backlog items

- Support ability for withdraw and deposit functions to be paused separately rather than in a mutually-dependent manner. And provide a mechanism to formally delist assets (both collateral and capital assets.)

C1-03 liquidate() logic ● High ✓ Resolved

Liquidation calculations use `healthFactorForPosition()` and don't check the cumulative `healthFactor()`. For example, the user deposits $500 in project tokens and borrows $400 in two lending tokens each. In this case, this user can’t be liquidated because $400 is less than $500. Also, calculations do not take into account accrued interest that is not written in the `userBorrowPosition[account][lendingTokenId][prjId].amountBorrowed` variable.

In the updated code liquidation threshold is not used, the loan-to-value ratio is used instead (with a typo in `healty`):

```
(uint256 healthFactorNumerator, uint256 healthFactorDenominator) = healthFactor(account,
projectToken, lendingToken);
if(healthFactorNumerator >= healthFactorDenominator){
  revert("PIT: borrow position is healty");
}
```

Recommendation

Consider refactoring the liquidation calculations using the `healthFactor()` and accrued interest. We strongly recommend testing the refactored logic.

C1-04 Admin and Moderator privileges ● High Partially fixed

`lvr`, `ltf` and `prjSale` variables in the functions `addPrjToken()`, `setLvr()`, `setLtf()`, and `setPrjSale()` could be up to 1 by any account with the Moderator role. Significant changes of these parameters may cause sudden liquidations of some positions. Therefore these moderator accounts should be properly secured.

Admin can manipulate with `priceOracle` address. In one transaction he can change this variable to a malicious contract, process some action, and then change it back.

Moderator can pause the withdrawal of lending and project tokens. This may cause losing users' funds. Also, the `liquidate()` function won't work on pause.

Recommendation

We recommend securing the government accounts by putting them behind Timelock with multi-sig admin.

Update

Now the `liquidate()` function is callable if the project token is on pause.

Team response

Fringe Finance intends to proceed with this current state and mitigate the risks as follows:

- By implementing the Gnosis multisig Admin vault that will necessitate multiple Admins to authorize a governance change.
- We have enabled liquidations to still occur for paused collateral assets, which allows for us to not have to risk causing platform insolvency when pausing a token.
- Backlog future resolution is to modify the contract to build the parameter change delay into the Fringe Finance platform.

C1-05 Change bLendingToken ● High ✓ Acknowledged

There is a possibility for moderators to change the address of `bLendingToken` for some lending tokens on an invalid address or different `bLendingToken`. This may disrupt the contract's logic.

Recommendation

To the `setLendingTokenInfo()` function, add checks for the underlying token of `bLendingToken` to be equal to the `lending` token.

Update

Admin still can replace `bLendingToken` with a wrong address. Moreover, there is no check that the total supply of `bLendingToken` is zero. If changing takes place when there is a positive total supply, users will lose their funds in the former contract.

Team response

Fringe Finance intends to proceed with this current state by addressing the matter via a combination of a multisig Admin and to institute a manual policy that has each multisig Admin verify that the bToken contract is not configured maliciously. Backlog item will further mitigate this risk by introducing a governance parameter delay to enable users to react if the pending change is deleterious. Additionally, a backlog item will be considered to minimize governance in the future by disabling any changes to this bToken configuration.

C1-06 Wrong decimals in calculations ● High Partially fixed

In the function `withdraw()` there are errors in calculating the `collateralProjectTokenAmount` variable and in the `liquidate()` function in calculating the `projectTokenEvaluation` variable.

The first multiplier has decimals of a lending token and the last multiplier has decimals 6. In the general case, this produces errors in calculations because these decimals aren't equal.

Update

Only the `withdraw()` function is fixed. But in case lending token has decimals less than 6, calculation will be wrong too.

Team response

Fringe Finance intends to proceed with this current state because this matter only applies to lending tokens (e.g. USDC) and this matter will not manifest for the lending token USDC due to it having decimals = 6. Fringe Finance will not configure any other lending tokens until a later date, by which time we will have modified the smart contracts (to support lending tokens with decimals other than 6) which will undergo a further audit at that time to ensure Fringe Finance does not encounter this issue. Backlog future resolution is to modify the contract to not do subtraction with unsigned integers with the possibility of a negative result.

Update

In the function `pitRemaining()` there are calculations with the same problem. Varialbes `_pit` and `_totalOutstanding` have different decimals and because of that there will be wrong calculations in the `borrow()` function.

C1-07 healthFactor checking ● High ✓ Resolved

The function `withdrawTo()` checks the `healthFactorForPosition` only for the first lending token in the loop, see L240-249:

```
        for(uint i=0; i < lendingTokens.length; i++){
            UserBorrowPosition storage msgSenderBorrowPosition =
userBorrowPosition[_msgSender()][i][prjId];
            if(msgSenderBorrowPosition.amountBorrowed > 0){
                msgSenderPrjPosition.amountPrjDeposited -= amountPrj;
                (uint newNum, uint newDenom) = healthFactorForPosition(_msgSender(), i,
prjId);
                if(newNum >= newDenom){
                    totalStakedPrj[tokenPrj] -= amountPrj;
                    IERC20Upgradeable(tokenPrj).safeTransfer(beneficiar, amountPrj);
                    emit Withdraw(_msgSender(), prjId, tokenPrj, amountPrj, beneficiar);
                    return;
                }
                else{
                    revert("PIT: the new account health is less than 1 when withdrawing
this amount of PRJ");
                }
            }
```

Also, even if each of the individual health factors for all lending tokens passes the check, the cumulative health factor could reach the danger zone, i.e. deposit amount wouldn't be able to cover the total borrow.

Recommendation

Remove the return and revert actions from the loop. Iterating over the lending tokens should only set 1 red flag and break the loop as soon as it finds the low health factor position.

Consider checking the total `healthFactor()` instead of `healthFactorForPosition()`.

C1-08 Locked project tokens ● High ✓ Acknowledged

If users deposit some project tokens, for which there is no price provider or there is `BackendPriceProvider` as a price provider, that doesn't have a valid `getEvaluation()` function, then they won't be able to withdraw their funds through `withdraw()` function.

```
    function withdraw(...) {
        (...)
        (uint256 healthFactorNumerator, uint256 healthFactorDenominator) =
healthFactor(msg.sender, projectToken, lendingToken);
    }

    function healthFactor(...) {
        (...)
        numerator = pit(account, projectToken, lendingToken);
    }

    function pit(...) {
        (...)
        uint256 evaluation = getProjectTokenEvaluation(projectToken,
depositPosition[account][projectToken][lendingToken].depositedProjectTokenAmount);
    }

    function getProjectTokenEvaluation(...) {
        return priceOracle.getEvaluation(projectToken, projectTokenAmount);
    }
```

Recommendation

We recommend avoiding tokens without a Chainlink price feed.

C1-09 healthFactorForPosition/healthFactor calculation

● High ✓ Partially fixed

The `healthFactorForPosition()` function (`healthFactor()` in the updated code) returns 2 values evaluated in different currencies: the first one, nominated in USD stablecoin, is taken from `priceOracle.getEvaluation()`, the second one is the amount of borrowed lending token. Different tokens may have different decimals, causing a wrong health factor in general.

```
function healthFactorForPosition(address account,uint256 lendingTokenId,uint256 prjId)
public view returns(uint256 numerator, uint256 denominator){
  uint256 lt = liquidationThresholdForPosition(account, prjId);
  uint256 borrowedLendingToken = userBorrowPosition[account][lendingTokenId]
[prjId].amountBorrowed;
  return (lt,borrowedLendingToken);
}
```

Recommendation

Consider refactoring health factor calculations and/or oracle evaluation, e.g. the lending token address could be included in the oracle call.

Update

Calculations are fixed in case the lending token has `decimals >= 6`. Otherwise, this function reverts the transaction (this function is called in functions `withdraw()` and `liquidate()`).

Team response

Fringe Finance intends to proceed with this current state and will institute a manual policy that will only list collateral tokens with decimals >=6.

Backlog items

- Future resolution is to modify the contract to not do subtraction with unsigned integers with the possibility of a negative result. For example: this can be easily fixed by multiplying the lvr.denominator by 1e18 (1 ether expressed in wei) and dividing the end result by 1e18. This way, integer arithmetic doesn't get truncated to zero accidentally for assets with a low number of decimal places. This will support listing lending assets with >= 6 decimals.

C1-10 Pausable functions ● Medium ✓ Resolved

The only functions of the contract affected by `whenPaused` or `whenNotPaused` modifiers are `withdrawTo()` and `withdraw()`. We believe this may cause a problem then after discovering a contract bug only the withdrawals would be paused, but not the actual exploiting transactions.

C1-11 Redundant separation of functions ● Medium ✓ Resolved

The function `addBLendingToken()` should be a part of the `addLendingToken()` function. Adding a Lending token without a corresponding BLending one or messing with the existing Lending/BLending pairs may cause a partial malfunction of the lending subsystem.

Recommendation

We recommend merging `addBLendingToken()` function into the `addLendingToken()` and adding a check to prevent duplication of lending token addresses.

C1-12 Flash-loan vulnerability ● Medium ✓ Resolved

Because the contract gets token prices by calculating them from reserves of the pairs on Uniswap, the whole contract is vulnerable to flash-loan attacks. Price manipulation allows the attacker to borrow almost any amount of lending tokens from the system.

Recommendation

We recommend implementing Oracle price feeds of any off-chain form.

Update

3 different price providers were introduced with the code update. However, only the ChainlinkPriceProvider is operable and uses the off-chain data. BackendPriceProvider is unfinished, and UniswapV2PriceProvider still suffers from the described issue. We strongly recommend avoiding the latter 2 oracles in their current form. Otherwise, this issue must be considered a critical one.

Team response

No action required given the following:

- We already mitigate price manipulation attacks via collateral debt limits (max borrow limits).
- Uniswap price feeds are currently not going to be implemented with the Fringe Finance platform. If Uniswap/TWAP price feeds are eventually incorporated into the Fringe platform, it will first undergo a re-design and will be subject to further audit at that time.
- Price manipulation attacks intended to cause liquidations do not pose a systemic risk to the platform’s solvency. Borrowers can mitigate this risk with their choice of appropriate collateralisation ratios.

C1-13 Error messages ● Low ✓ Resolved

`require()` statements should contain error messages. Deployment gas could be saved with the library of errors.

C1-14 Liquidation threshold error ● Low ✓ Acknowledged

There's a wrong requirement in the `setProjectTokenInfo()` function. The argument `_liquidationTresholdFactorNumerator` should be not greater than the `_liquidationTresholdFactorDenominator` argument in order to confirm the whitepaper.

Audited code doesn't contain using the LTF threshold functions.

This issue was introduced with the code update.

Recommendation

Consider reverting the changes in the `ltf` ratio.

Team response

This issue is resolved and no further action is required.

Update

The `liquidationThreshold()` function is not used anywhere in the Fringe contracts after the code update. Its severity was mitigated, but we decide to not change its status.

C1-15 Gas optimizations ● Low ✓ Partially fixed

Excessive reads in the `for()` loop in `withdrawTo()`, `borrow()`, `repayBorrow()`, `balanceOfPit()` functions.

No need to calculate `2^256-1` multiple times. Precalculated constant or `type(uint256).max` should be used.

Functions that are not used inside the contract can be marked as `external` instead of `public` to save gas on transactions.

Global variable `prjSales` and functions `setPrjSale()` and `getPrjEvaluationInBasicTokenWithSale()` are redundant.

The following code in L631 is useless:

```
require(prjIndexesLength <= projectTokens.length);
```

The following code in L365 of the updated contract is useless:

```
currentBorrowBalanc = BLendingToken(bLendingToken).borrowBalanceCurrent(msg.sender);
```

C1-16 Dubious calculations ● Low ✓ Resolved

The `balanceOfPit()` function calculations rely on the `getPrjEvaluationInBasicToken()` function, i.e. price via reserves:

```
prjEvaluationInBasicToken = amountPrj * reserveBasicToken / reservePrj
```

Such calculations are better to be performed with the `getAmountOut()` function of UniswapV2Library.

C1-17 Excessive inheritance ● Low ✓ Resolved

ERC20Upgradeable and PausableUpgradeable contracts from inheritance are not used at all.

Update

Pausable functionality was implemented with the code update, ERC20 inherited contract is still not in use.

C1-18 Gas limit issue ● Low ✓ Acknowledged

If the admin adds too many lending or project tokens, calling almost every write function will cost too much gas. In extreme cases, the amount of gas for executing these functions will exceed the gas limit of the block and the users won't be able to call them. Some users may even lose their funds.

Recommendation

We recommend optimizing functions with loops over the token arrays. The admin (or DAO governance) should monitor the average gas costs and stop proposing project extensions to prevent gas problems.

C1-19 Token support ● Info Acknowledged

Project tokens cannot be RFI or tokens with commissions. An admin (or DAO governance) should never propose adding such tokens to the system.

C1-20 Lack of events ● Info Partially fixed

Governance functions `addProjectToken()`, `addLendingToken()`, `setPriceOracle()` don't emit corresponding events.

Update

Most of the governance functions don't emit any events.

C1-21 Lending tokens ● Info ✓ Resolved

Calls of `repayBorrowTo()` in `repayBorrow()` and `repayBorowAll()` functions are possible only to BUSDC token as if it's meant to be the only lending token in the system.

Update

BUSDC contract was transformed into the universal BLendingToken contract.

C1-22 Interest uniform distribution ● Info ✓ Resolved

Interest is accrued on all project tokens that were used as collateral for the lending token in equal proportions despite the proportional amount of the collaterals:

```
estimateInterest = (currentBorrowBalance - cumulativeBorrowBalance) / borrowedPositions;
```

C1-23 Admin role

● Info ✓ Acknowledged

To use `addProjectToken()` and `addLendingToken()` functions, admin has to have moderator role too.

Team response

Fringe Finance intends to proceed with this current state, given it only represents a minor inconvenience and this does not result in any additional security risks.

C1-24 One token can be lending and project at the same time

● Info ✓ Acknowledged

Lending/project/basic tokens could be added to each other's lists. In this case logic and math of the contract will be crashed.

Recommendation

Adding safety checks on input data in `addPrjToken()` and `addLendingToken()` functions.

Team response

We have assessed this matter with our development team and an external auditing team and the outcome is that this issue does not manifest. We have tested the configuration where an asset is configured as both a lending and borrowing asset and the platform remains well-behaved.

Update

We've mitigated issue's severity after the code update.

C1-25 No check on user's input data ● Info ✓ Resolved

The function `borrow()` works with an arbitrary `prj` address as it's a `projectTokens[0]` address. It should revert in case the user passes the wrong address.

C2. UniswapPathFinder

Overview

Helper contract for finding the effective swap path. In this version of the code, UniswapPathFinder is used only for retrieving information about reserves of pairs on Uniswap.

Update: the contract was removed from the scope and from the repository.

Issues

C2-01 Excessive reads from storage ● Low ✓ Resolved

Excessive reads in the `for()` loops may cause a significant gas consumption and should be avoided using the local variables.

C2-02 Switched arguments in uniswapPositionCap() ● Low ✓ Resolved

The `uniswapPositionCap()` function has its internal reserves variables switched in L49.

C2-03 Redundant code ● Info ✓ Resolved

The whole contract may be declared redundant as only one function is used in the Fringe project.

C3. BToken

Overview

Modification of the CToken contract by Compound Finance. Contains an additional `view` version of `balanceOfUnderlying()` function. Possible transfer restrictions from Comptroller are lifted.

Issues

C3-01 Allowed transfers ● High ✓ Resolved

The removed `comptroller.transferAllowed()` call in the `transferTokens()` function leads to anyone being able to withdraw their funds, even if they borrowed the maximum amount for their deposit.

Moreover, some functions that are called in the `comptroller.transferAllowed()` function are not called. Because of that, a new vector of attack is possible. The hacker needs to make these steps:

1. Mint a big amount of BToken tokens to some account.
2. Transfer all BTokens to another account.
3. Call `claimComp()` on a new account and this BToken.
4. Repeat steps 2 and 3 enough times to withdraw all the Governance tokens from the Comptroller account.

Recommendation

We recommend updating the documentation, explicitly mentioning that all the tokens inherited from BToken must implement borrow/redeem functions via the PrimaryIndexToken contract.

Also, call the `transferAllowed()` function on the Comptroller to make sure that the `updateCompSupplyIndex()` and `distributeSupplierComp()` functions are called.

Update

All rewards calculations, including the `claimComp()` function, were removed from the Bondtroller (previously Comptroller) contract.

C3-02 Admin overpower

● Low ✓ Acknowledged

Using the `_reduceReserves()` function the admin can receive all reserves to his account.

The `_setInterestRateModel()` function could be used to set a malicious rate model contract with broken `getBorrowRate()` parameters causing a permanent failure of the `accrueInterest()` function of the BToken contract. Because of that, the users won't be able to enter or extract the token.

Recommendation

We recommend securing the admin account with Timelock and multi-sig governance until the future DAO implementation.

C3-03 Underflow in repayBorrowFresh()

● Low ✓ Acknowledged

Undeflow checks have been disabled in the updated code:

```
(vars.mathErr, vars.accountBorrowsNew) = subUInt(vars.accountBorrows,
vars.actualRepayAmount);
//require(vars.mathErr == MathError.NO_ERROR,
"REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED");

(vars.mathErr, vars.totalBorrowsNew) = subUInt(totalBorrows, vars.actualRepayAmount);
//require(vars.mathErr == MathError.NO_ERROR,
"REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED"); // not sufficient
```

The `PrimaryIndexToken.repay()` function is able to cause careful underflow (resulting in zero returning value) in BToken if `borrowPositionsAmount > 1`:

```
if (borrowPositionsAmount == 1) {
  ...
} else {
  ...
  (, amountRepaid) = info.bLendingToken.repayTo(repayer, borrower,
lendingTokenAmountToRepay);
  ...
}
```

C3-04 Redundant event ● Info ✓ Resolved

In the code, there is a redundant event `Test3` that is not used anywhere.

C4. BEther

Overview

Modification of the CEther contract by Compound Finance. Added an empty `receive()` function, all other changes are related to the updated pragma version. BEther can't be used with the PrimaryIndexToken contract, thus should be marked as deprecated.

Update: Fringe Finance team assures us this contract is deprecated by removing it from the repository.

Issues

C4-01 Allowed transfers ● High ✓ Resolved

Anyone can borrow any amount of ETH and redeem or transfer tokens without restrictions to the deposit/borrow amount due to a disabled Comptroller checking in the BToken contract. We specifically address this issue to the BEther as it's the only contract in the Fringe repo that has public redeem/borrow functions with original Compound logic.

Recommendation

BEther should be marked as deprecated in its current state. We recommend full refactoring if it's meant to be deployed.

Team response

No action required given the bEther contract is not going to be deployed to mainnet. The code has now been deleted from the code repository.

C4-02 Possible ETH locking ● Medium ✓ Resolved

The empty `receive()` function may lock the ETH sent without `msg.data`.

Recommendation

Move logic from the `fallback()` function to the `receive()` function and then delete the `fallback()` function.

Team response

No action required given the bEther contract is not going to be deployed to mainnet. bEther code has now been removed from the code repository.

C4-03 Max borrow logic ● Medium ✓ Resolved

Max borrow amount is different compared to other tokens. Because the max borrow amount for BEther is calculated through Comptroller and Oracles, but for the BToken it is calculated through PrimaryIndexToken and Uniswap's reserves.

Recommendation

BEther should be marked as deprecated in its current state. We recommend full refactoring if it's meant to be deployed.

Team response

No action required given the bEther contract is not going to be deployed to mainnet. bEther code has now been removed from the code repository.

C4-04 Native transfers ● Low ✓ Resolved

The `doTransferOut()` function uses the `transfer()` method of sending native currency that is now discouraged due to non-flexible gas management. The recommended function is `call()` with an additional reentrancy guard.

C5. Comp

Overview

Modification of the Comp contract by Compound Finance. ERC20 governance token with voting support.

Update: the contract was removed from the scope and from the repository.

Issues

C5-01 Name and symbol variables ● Info ✓ Resolved

The name and symbol of the COMP token aren't changed from Compound Finance.

C6. Bondtroller

Overview

Modification of the Comptroller contract by Compound Finance. Entering the markets is restricted only to the PrimaryIndexToken. The `getHypotheticalAccountLiquidity()` function is effectively disabled. Using this contract by the BToken contract should be reviewed because in most cases all actions with Comptroller are a waste of gas. It should be used only for distribution of the Governance token.

Update: the contract was renamed from Comptroller to Bondtroller, original Comptroller logic and Comp rewards distribution were disabled.

Issues

C6-01 Admin overpower ● Low ✓ Resolved

The `_setCompSpeed()` function has no checks on input `compSpeed` value causing the possible minting of the arbitrary amount of rewards in a single block and an attack on the liquidity pools by a malicious admin of the Comptroller.

Recommendation

We recommend securing the admin account with Timelock and multi-sig governance until the future DAO implementation.

C6-02 Typo ● Info ✓ Resolved

All cTokens were renamed to bTokens with the code update, but not in the `enterMarkets()` and `exitMarket()` functions.

C6-03 Inconsistent comment ● Info ✓ Resolved

The comment in L898 contains the wrong constant:

```
// Check collateral factor <= 0.9
```

The actual value of `collateralFactorMaxMantissa` is set in L82:

```
uint internal constant collateralFactorMaxMantissa = 1e18; // 1
```

C7. BPrimaryIndexToken

Overview

BToken-based token for the project tokens of the PrimaryIndexToken contract. BPrimaryIndexTokens are not in use in the PrimaryIndexToken, thus it should be marked as deprecated.

Update: the contract was removed from the scope and from the repository.

Issues

C7-01 Contract is inoperable ● Medium ✓ Resolved

In the `mintTo()`, `redeemUnderlyingTo()` and `redeemTo()` functions there is a check that

```
msg.sender == underlying
```

Because `primaryIndexToken` doesn't equal `underlying`, the contract won't work and users won't be able to enter the token.

Recommendation

BPrimaryIndexToken should be marked as deprecated in its current state. We recommend refactoring if it's meant to be deployed.

C7-02 PrimaryIndexToken address update ● Low ✓ Resolved

Setting the wrong address with the `setPrimaryIndexToken()` function may cause locked lending funds. Moreover, there is no point in changing PrimaryIndexToken to a new PrimaryIndexToken, because the math of BPrimaryIndexToken and the new PrimaryIndexToken will crash.

Recommendation

We recommend removing the ability to change the `primaryIndexToken` address. If it's not possible, the `setPrimaryIndexToken()` function should perform a check of input data, and/or the admin account should be secured with a Timelock.

C8. PriceProviderAggregator

Overview

Oracle aggregator. The contract implements the AccessControl authorization model from the OpenZeppelin library. The contract was added to the scope and to the repository with the code update.

Issues

C8-01 AccessControl misuse ● Low ✓ Acknowledged

`grandModerator()` and `revokeModerator()` checks the role of `msg.sender` twice:

```
modifier onlyAdmin() {
  require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "Caller is not the Admin");
  _;
}

function grantRole(bytes32 role, address account) public virtual override
onlyRole(getRoleAdmin(role)) {
  _grantRole(role, account);
}
```

C9. BLendingToken

Overview

BErc20 implementation token to be used as a lending token in the Fringe system. Major public functions could be called only by the PrimaryIndexToken contract.

Update: the contract was renamed from BUSDC to BLendingToken.

Issues

C9-01 setReserveFactor() function doesn't check input data ● Medium ✓ Resolved

There is no need for this function because the BToken contract already has this functoinality in the function `_setReserveFactor()`. Moreover, inside the `setReserveFactor()` function, there is a call of the `_setReserveFactorFresh()` function without calling `accrueInterest()`. Because of that, through this function, it won't be possible to make any changes to the `reserveFactorMantissa` variable. Also, a transaction in case of failing won't be reverted and this may confuse the admin.

Recommendation

We recommend adding the input data filtering as well as `accrueInterest()` call before the `_setReserveFactorFresh()`.

C9-02 PrimaryIndexToken address update ● Low ✓ Acknowledged

Setting the wrong address with the `setPrimaryIndexToken()` function may cause locked lending funds. Moreover, there is no point in changing PrimaryIndexToken to the new PrimaryIndexToken, because the math of BUSDC and the new PrimaryIndexToken will crash.

Recommendation

We recommend removing the ability to change the primaryIndexToken address. If it's not possible, the `setPrimaryIndexToken()` function should perform the check of input data, and/or the admin account should be secured with a Timelock.

C9-03 Misuse of AccessControl ● Low ✓ Acknowledged

`grandModerator()` and `revokeModerator()` functions perform different checks: one with the `onlyAdmin` modifier and second inside the `grantRole()` internal check:

```
modifier onlyAdmin(){
  require(msg.sender == admin,"msg.sender not admin!");
  _;
}

function grantRole(bytes32 role, address account) public virtual override
onlyRole(getRoleAdmin(role)) {
  _grantRole(role, account);
}
```

`admin` account must have the `DEFAULT_ADMIN_ROLE` to call these functions.

However, `init()` function does not set up the `DEFAULT_ADMIN_ROLE` role at all. Only the initial deployer address would be able to use `onlyModerator` functions of the contract.

Recommendation

Remove the `admin` variable, fix AccessControl initialization.

Update

AccessControl initialization was fixed.

`onlyAdmin()` modifier was changed to

```
    modifier onlyAdmin(){
        require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender),"msg.sender not admin!");
        _;
    }
```

duplicating the `grantRole()` requirements.

Team response

Fringe Finance intends to proceed with this current state. `grantRole` and `revokeRole` are public functions in `AccessControlUpgradeable` which PIT inherits, so any admins can always be added as moderators too before calling the `grandModerator` and `revokeModerator` functions.

C10. BackendPriceProvider

Overview

Price oracle operated by the owner. The contract implements the AccessControl authorization model from OpenZeppelin library.

Issues

C10-01 Oracle is inoperable ● Medium ✓ Acknowledged

The `getEvaluation()` and `getPrice()` functions revert all the calls. Signature verification could be replicated within the `validTo` period of time.

Recommendation

We recommend finishing the contract and properly testing it before introducing it into the project.

Team response

Our findings are that this is a non-issue and no action is required due to these functions not being called. If they're not implemented, the contract wouldn't be deployable. The two functions just revert and are never used. The reason why they need to be there (and just revert if they're called; analogous to a "default" implementation) is because they're a part of the `PriceProvider` interface.

Update

Current version of PrimaryIndexToken calls only the `priceOracle.getEvaluation()` function but not the `getEvaluationSigned()` one.

C10-02 AccessControl misuse ● Low ✓ Acknowledged

`grandModerator()` and `revokeModerator()` checks the role of `msg.sender` twice:

```
modifier onlyAdmin() {
  require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "Caller is not the Admin");
  _;
}

function grantRole(bytes32 role, address account) public virtual override
onlyRole(getRoleAdmin(role)) {
  _grantRole(role, account);
}
```

C11. ChainlinkPriceProvider

Overview

Oracle contract using the Chainlink price feed. The contract implements the AccessControl authorization model from the OpenZeppelin library. The contract was added to the scope and to the repository with the code update.

Issues

C11-01 Uint overflow chance ● Medium Acknowledged

In the function `getPrice()` on line 109, uint256 overflow can be reached, causing the revert in multiple functions of the `PrimaryIndexToken` contract.

Recommendation

This problem may be resolved in two ways:

1. By including an additional check on the sum of oracles' decimals in the `setTokenAndAggregator()` function.
2. By performing the L112-117 calculations inside the `for()` cycle.

Team response

These issues will not manifest due to the way Fringe Finance uses data feeds and therefore Fringe Finance intends to proceed with this current state.

The first issue will only manifest if `AggregatorV3Interface#latestAnswer` returns numbers with more than 18 decimal places of precision. (USD answers should have 8, answers priced in terms of ETH should have 18). Fringe Finance will not encounter this condition.

The second issue will only manifest when the 'price path' employs a large number of elements. A 'price path' refers to the sequence of price feeds that must be used to arrive at the collateral asset price denominated in USD, i.e. CollateralTokenPrice/USD.

In many circumstances, the price feeds already denominate token prices in USD and therefore 'price path' is just 1. But sometimes the price feeds denominate token prices in ETH and requires two price paths, as follows:

CollateralTokenPrice/ETH & ETH/USD ==> CollateralTokenPrice/USD.

Two price paths are within the limit where this issue could manifest and therefore this audit item requires no action.

C11-02 AccessControl misuse ● Low ✓ Acknowledged

`grandModerator()` and `revokeModerator()` checks the role of `msg.sender` twice:

```
modifier onlyAdmin() {
  require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "Caller is not the Admin");
  _;
}

function grantRole(bytes32 role, address account) public virtual override
onlyRole(getRoleAdmin(role)) {
  _grantRole(role, account);
}
```

C11-03 Inconsistent comment ● Info ✓ Resolved

Comment at L11-13 is pasted from the UniswapV2PriceProvider oracle contract.

C12. UniswapV2PriceProvider

Overview

Oracle contract using the direct price calculation with the Uniswap pair reserves, marked 'for development purposes only'. The contract implements the AccessControl authorization model from the OpenZeppelin library. The contract was added to the scope and to the repository with the code update.

Issues

C12-01 Flash-loan vulnerability ● Critical ✓ Acknowledged

Because the contract gets token prices by calculating them from reserves of the pairs on Uniswap, the whole contract is vulnerable to flash-loan attacks. Price manipulation allows the attacker to borrow almost any amount of lending tokens from the system.

Update: Fringe Finance team assures that UniswapV2 won't be used (at least initially) as a price oracle in the mainnet.

Recommendation

We recommend avoiding price calculations based on pair reserves. For decentralized oracle, a TWAP oracle model could be used.

Team response

No action required given the following:

- We already mitigate price manipulation attacks via collateral debt limits (max borrow limits).
- Uniswap price feeds are currently not going to be implemented with the Fringe Finance platform. If Uniswap/TWAP price feeds are eventually incorporated into the Fringe platform, it will first undergo a re-design and will be subject to further audit at that time.

- Price manipulation attacks intended to cause liquidations do not pose a systemic risk to the platform's solvency. Borrowers can mitigate this risk with their choice of appropriate collateralisation levels.

C12-02 Division before multiplication   ● Low   Acknowledged

Calculations at L115 contain the division operation before the multiplication. This should be avoided in order to reduce the division error.

C12-03 Lack of safety checks   ● Low   Acknowledged

`setTokenAndPair()` function checks if `pair.token0() == token` at L77, but doesn't check the `token1`.

C12-04 AccessControl misuse   ● Low   Acknowledged

`grandModerator()` and `revokeModerator()` checks the role of `msg.sender` twice:

```
modifier onlyAdmin() {
  require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "Caller is not the Admin");
  _;
}

function grantRole(bytes32 role, address account) public virtual override
onlyRole(getRoleAdmin(role)) {
  _grantRole(role, account);
}
```

C13. PrimaryLendingPlatformProxyAdmin

Overview

Admin contract for the PrimaryLendingToken proxy built on ProxyAdmin from OpenZeppelin library.

Issues

C13-01 Planned upgrade can't be cancelled ● High ✓ Resolved

The `appendUpgrade()` function requires zero `appendTimestamp` value, and the `upgrade()` function requires the opposite:

```
function appendUpgrade(TransparentUpgradeableProxy proxy, address newImplementation)
public onlyOwner {
  UpgradeData storage _upgrade = upgradeData[address(proxy)];
  if (_upgrade.appendTimestamp != 0) {
    revert("PrimaryLendingPlatformProxyAdmin: wait for next upgrade");
  }
  ...
}

function upgrade(TransparentUpgradeableProxy proxy, address implementation) public
override onlyOwner {
  UpgradeData storage _upgrade = upgradeData[address(proxy)];
  if(_upgrade.appendTimestamp == 0) {
    revert("PrimaryLendingPlatformProxyAdmin: Call first appendUpgrade(...)");
  }
  ...
  delete upgradeData[address(proxy)];
}
```

Planned upgrade with a bug can't be cancelled or overrides. The only option is to change implementation and fix bugs with a delay.

Recommendation

We recommend adding the cancelling function or remove the `appendTimestamp == 0` requirement from `appendUpgrade()`.

C14. ErrorReporter

Overview

Copy of the ErrorReporter contract by Compound Finance. No issues were found.

C15. JumpRateModelV2

Overview

Copy of the JumpRateModelV2 contract by Compound Finance. No issues were found.

C16. JumpRateModelV2Upgradeable

Overview

Initializable version of the BaseJumpRateModelV2 contract by Compound Finance. No issues were found.

Update: the contract was renamed from BaseJumpRateModelV2 to JumpRateModelV2Upgradeable.

C17. Exponential

Overview

Copy of the Exponential contract by Compound Finance. No issues were found.

C18. ExponentialNoError

Overview

Copy of the ExponentialNoError contract by Compound Finance. No issues were found.

C19. BondtrollerStorage

Overview

A modification of the ComtrollerStorage contract by Compound Finance with removed Unitroller storage variables for pending admin and Comptroller implementation. Changed mapping for market memberships. No issues were found.

Update: the contract was renamed from ComptrollerStorage to BondtrollerStorage.

C20. BErc20

Overview

Severely modified version of the CErc20 contract by Compound Finance. Most of the logic is removed and needs to be implemented in the top-level token contracts. No issues were found.

C21. InterestRateModel

Overview

Copy of the InterestRateModel contract by Compound Finance. No issues were found.

C22. UniswapV2Library

Overview

Copy of the UniswapV2Library contract by Uniswap. No issues were found.

C23. PriceOracle

Overview

Copy of the PriceOracle contract by Compound Finance. No issues were found.

Update: the contract was removed from the scope and from the repository.

C24. CarefulMath

Overview

Copy of the CarefulMath contract by Compound Finance. No issues were found.

C25. SimplePriceOracle

Overview

Copy of the SimplePriceOracle contract by Compound Finance. No issues were found.

Update: the contract was removed from the scope and from the repository.

5. Conclusion

2 critical and 11 high severity issues were found, 6 of them were fixed and 4 other were fixed partially with the update. Some contracts with acknowledged issues aren't going to be used in mainnet.

The contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured. Until the DAO governance is implemented, we recommend securing the admin account with a Timelock and multi-sig. The contracts are well-tested. We recommend adding the NatSpec descriptions to the contracts with public accessible functions.

This audit includes recommendations on the code improving and preventing potential attacks.

Appendix A. Issues’ severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

contact@hashex.org

@hashexbot

blog.hashex.org

linkedin

github

twitter

#HashEx

BLOCKCHAIN SECURITY