yAudit KEOM Upgrade Review

Review Resources:

- The "Toxic Liquidation Spirals" whitepaper explains the logic implemented in this code
- Custom summary of changes and background information related to the modified code

Auditors:

- engn33r
- spalen

Table of Contents

Review Summary

KEOM

KEOM is a liquidity market that generates yield by lending deposited tokens at interest rates determined by the utilization ratio of each asset. The protocol is based on Compound Finance, but has modification and improvements compared to the original code.

The contracts of the KEOM Repo were reviewed over 2 days. The code review was performed by 2 auditors between August 16 and August 17, 2023. The repository was under active development during the review, but the review was limited to the latest commit for two PRs at the start of the review. This was commit bc39bea2f2bf582668d5e5499a0748e7bd3ed623 for PR 60 and commit 11f360bbe8166d81cbb9545fcbd7e395248 7b80b for PR 62 of the KEOM repo.

Scope

The scope of the review consisted of reviewing the impact of two PRs that change crucial logic related to the liquidation mechanism of the protocol:

- https://github.com/0Vix/0vix-protocol/pull/60 (frozen at commit bc39bea2f2bf582668d5e5499a0748e7bd3ed623)
- https://github.com/0Vix/0vix-protocol/pull/62 (frozen at commit 11f360bbe8166d81cbb9545fcbd7e395248 7b80b)

After the findings were presented to the KEOM team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, KEOM and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Average	The pausing operations has the proper admin and pause guardian access checks. The pause guardian role was not set on-chain at the time of review, and the zkEVM on-chain deployment used an EOA for the admin role.
Mathematics	Good	No complex math was used beyond a standard Compound Finance math library.
Complexity	Good	The scope was very limited which reduced the overall complexity of the code examined
Libraries	Average	Only standard Compound Finance libraries were used.

Category	Mark	Description
Decentralization	Average	The protocol does not have decentralized governance yet like Compound Finance and the protocol team still controls many key protocol values without a Timelock.
Code stability	Low	The code was not fully finished and some tests were not passing at the start of the review.
Documentation	Good	The development team has performed extensive research on the tokenomics behind the code change and has done a good job of summarizing the intent of the changes that were in scope for the review.
Monitoring	Good	Events were emitted where applicable.
Testing and verification	Low	Some tests were not passing at the time of the review.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Critical Findings

None.

High Findings

None.

Medium Findings

None.

Low Findings

1. Low - `setProtocolPaused()` does not call `_setRedeemPaused()`

`setProtocolPaused()` is intended to be used by the admin to call all Comptroller pause functions. One function is missing from the list of internal calls. If `_setRedeemPaused()` is not called in the same block as `setProtocolPaused()`, it will be more difficult to determine an exact point in time at which the protocol was paused. This can add complications to scenarios where a checkpoint of user balances before pausing the protocol is needed. Users may also be able to observe the first transaction and frontrun the second transaction before the protocol is fully paused.

Technical Details

`_setRedeemPaused()` is newly created in the same PR 60 that introduces `setProtocolPaused()`. `_setRedeemPaused()` is not called in `setProtocolPaused()` but should be. A TODO indicates this should be done, but the code did not implement this call at the time of the review.

Impact

Low. Potential accounting errors may occur if the protocol is not fully paused in a single transaction.

Recommendation

Add the line `_setRedeemPaused(market, _paused);` inside the for loop of `setProtocolPaused()`.

Developer Response

Fixed in this commit.

2. Low - KEOM Comptroller admin is EOA

The Comptroller contract deployed on Polygon zkEVM has a EOA for the `admin` address.

Technical Details

The `admin` address for the KEOM Polygon zkEVM Comptroller is an EOA address. The `admin` has control over pausing the protocol, setting the price oracle for every token market, collateral factors, and more. Setting these functions in an arbitrary or malicious way can seriously impact user value by triggering liquidations or creating a denial of service (DoS) situation. Compound Finance has a Timelock contract for the admin address, and Compound Forks often choose the same approach or use a multisig for the Comptroller admin. Using an EOA for crucial protocol functionality reduces the decentralization of the protocol and can serve as a single point of failure. Losing private keys has been seen in many past protocol exploits, so reducing the risk of this attack vector by avoiding privileged EOAs improves the security of the protocol.

Impact

Low. Using an EOA for a key protocol privileged address can create a single point of failure.

Recommendation

Set the Comptroller `admin` to a multisig or Timelock contract.

Developer Response

Fixed by setting admin to multisig

Gas Saving Findings

1. Gas - Unnecessary zero initializations

Setting a variable to its default value can waste gas.

Technical Details

Unnecessary zero initialization is here.

Impact

Gas Savings.

Recommendation

Remove the explicit variable initializations.

Developer Response

Fixed in this commit.

2. Gas - Calculate variable only if needed

Calculation of `vars.protocolSeizeTokens` can be skipped if some cases.

Technical Details

`vars.protocolSeizeTokens` variable is calculated every time but for cases when if statement above is false, end result will be 0. So there is no need to multiply with 0 to get 0.

Impact

Gas Savings.

Recommendation

Move `vars.protocolSeizeTokens` calculation inside above if statement:

```
        if (dynamicLiquidationIncentive > 1e18) {
            uint256 i;
            (vars.mathErr, i) = subUInt(dynamicLiquidationIncentive, 1e18); //no need to check mathErr because
```

```
dynamicLiquidationIncentive > 1e18
            protocolSeizeShare = mul_(
                i,
                Exp({mantissa: protocolSeizeShareMantissa})
            );
            vars.protocolSeizeTokens = mul_(
                seizeTokens,
                Exp({mantissa: protocolSeizeShare})
            );
        }
```

Developer Response

Fixed in this commit.

3. Gas - Remove `onlyAdminOrGuardian()` call from `setProtocolPaused()`

The `onlyAdminOrGuardian()` call is unnecessary in `setProtocolPaused()` of PR 60 because all other internal functions include this check.

Technical Details

The `onlyAdminOrGuardian()` line is repeated in the functions that are called by `setProtocolPaused()`, so `setProtocolPaused()` does not need to call it separately.

Impact

Gas Savings.

Recommendation

Remove the `onlyAdminOrGuardian()` line from `setProtocolPaused()`.

Developer Response

Fixed in this commit.

4. Gas - Use unchecked for gas savings

Unchecked math can be used when there is no risk of overflow or underflow due to existing logic preventing such conditions.

Technical Details

This line of OToken.sol can be simplified

```
- uint256 i;
- (vars.mathErr, i) = subUInt(dynamicLiquidationIncentive, 1e18); //no need to check mathErr because dynamicLiquidationIncentive > 1e18
+ unchecked { uint256 i = dynamicLiquidationIncentive - 1e18; } //no underflow risk because dynamicLiquidationIncentive > 1e18
```

This line in `Comptroller.getHypotheticalAccountLiquidityInternal()` can be unchecked

```
- vars.dynamicLiquidationIncentive -= 1;
+ unchecked { vars.dynamicLiquidationIncentive -= 1; }
```

This line in `OToken.seizeInternal()` can be unchecked

```
- vars.liquidatorSeizeTokens = seizeTokens - vars.protocolSeizeTokens;
+ unchecked { vars.liquidatorSeizeTokens = seizeTokens - vars.protocolSeizeTokens; }
```

Impact

Gas Savings.

Recommendation

Use unchecked when no risk of underflow exists.

Developer Response

Fixed in this commit. for `i` and `vars.dynamicLiquidationIncentive`, but not `vars.liquidatorSeizeTokens`, as this function cannot guarantee that `vars.protocolSeizeTokens` is less or equal to `seizeTokens`.

5. Gas - Use ++i for gas savings

++i can save gas compared to i++

Technical Details

i++ is used in this Comptroller for loop, but using ++i can reduce gas spent.

Impact

Gas Savings.

Recommendation

Use ++i instead of i++

Developer Response

Fixed in this commit.

Informational Findings

1. Informational - Newly added token markets should have code verified

There are 2 token markets added to the deployment script in PR 62. The tokens are KEOM tokens and should be verified on PolygonScan.

Technical Details

The two tokens added in the deployment data of PR 62, ostMATIC and owstETH, are unverified on PolygonScan. Users should be able to easily view the code they are interacting with, so these KEOM contracts should be verified.

Impact

Informational.

Recommendation

Verify all protocol code after deployment.

Developer Response

Two specified tokens have their implementation's verified which allows users to see functions they are interacting with. Proxy contracts are left unverified and we will add this finding to our backlog (it's not really trivial as Hardhat and Foundry cannot easily verify custom proxies).

2. Informational - Keep function `getAccountLiquidity()` clean

The additional return value `liquidationIncentive` is added to `getAccountLiquidity()`,

Technical Details

By adding an additional return value, the current function interface is broken. This call is really common in any application or protocol that uses Compound V2 forks. It will lead to broken integrations with all other applications and protocols. If added value is needed, it can be fetched using `getHypotheticalAccountLiquidity()`

Impact

Informational.

Recommendation

Remove added return value and keep the current interface.

Developer Response

Fixed in this commit.

3. Informational - Return 0 in error conditions

In some cases where an error is returned, it may be better to return zero instead of returning the `dynamicLiquidationIncentiveMantissa` value.

Technical Details

In this line and this line of Comptroller, it is possible that returning zero instead of dynamicLiquidationIncentiveMantissa may avoid confusion in these error edge cases. This is primarily a suggestion to consider, there was insufficient time to determine if all logic paths that reach these errors would be better served with a zero return value.

Impact

Informational.

Recommendation

Return a zero value in error return cases when it makes sense.

Developer Response

Fixed in this commit.

4. Informational - Combine lines to reduce complexity

Combine 2 lines of code to remove a variable and reduce code complexity.

Technical Details

Remove the `liquidationIncentive` variable:

```
(
    Error err,
    ,
    uint256 shortfall,
-    uint256 liquidationIncentive
+   dynamicLiquidationIncentiveMantissa
) = getHypotheticalAccountLiquidityInternal(
        borrower,
        IOToken(address(0)),
        0,
        0
    );

if (err != Error.NO_ERROR) {
    return (uint256(err), 0);
}

- dynamicLiquidationIncentiveMantissa = liquidationIncentive;
```

Impact

Informational.

Recommendation

Reduce the number of temporary variables for complexity reduction.

Developer Response

To the extent of my knowledge, when a function returns multiple values, all values need to be assigned to already declared variables or declaration of all variables should happen in this line. Your recommendation produced compilation error for me.

5. Informational - Unclear `closeFactor` calculation can be rewritten

The `closeFactor` math is unclear and can be improved. A comment in the PR points out the current formula is unclear.

Technical Details

The `closeFactor` formula is mentioned in a comment as:

```
closeFactor = 10 * (defaultCloseFactor * liquidationIncentive + defaultLiquidationIncentive - defaultCloseFactor - liquidationIncentive)
```

The variables can be grouped better to more easily understand the equation in this reorganized formula for `closeFactor`

`closeFactor = 10 * ( ( ( dynamicLiquidationIncentive - 1e18 ) * defaultCloseFactor ) + ( defaultLiquidationIncentive - dynamicLiquidationIncentive ) )`

By more clearly highlighting the relationship between dynamicLiquidationIncentive and 1e18, as well as the relationship between defaultLiquidationIncentive and dynamicLiquidationIncentive, the equation is easier to understand for any readers. The derivation of this equation from the Toxic Liquidation Spirals paper may also be useful to include in a comment.

Impact

Informational.

Recommendation

Use the more easily understood `closeFactor` equation above. Consider rewriting the solidity code to match the revised closeFactor formula.

```
- Exp memory downscaledCloseFactor = sub_(sub_(add_(mul_(defaultCloseFactor, dynamicLiquidationIncentive), Exp({mantissa: liquidationIncentiveMantissa})), defaultCloseFactor), dynamicLiquidationIncentive);
+ Exp memory downscaledCloseFactor = add_(mul_(defaultCloseFactor, sub_(dynamicLiquidationIncentive, Exp({mantissa: 1e18}))), sub_(Exp({mantissa: liquidationIncentiveMantissa}), dynamicLiquidationIncentive));
```

Developer Response

Your recommendation would produce an error when `dynamicLiquidationIncentive` is less than 1e18. As an alternative solution, I expanded the comment on the calculation of the closing factor in this commit.

6. Informational - NatSpec

Use NatSpec for all function arguments and return values to properly explain the purpose of different variable values.

Technical Details

- Function `liquidateBorrowAllowed()` has 2 return values which are not explained in the function description.
- Function `getHypotheticalAccountLiquidityInternal()` has multiple return values but one is not in return tag.
- Function `getHypotheticalAccountLiquidity()` has multiple return values but one is not in return tag.
- Function `getAccountLiquidity()` as multiple return values but one is not in return tag.
- Function `seize()` has the new function arg `dynamicLiquidationIncentive` but no new NatSpec.
- Function `seizeInternal()` has the new function arg `dynamicLiquidationIncentive` but no new NatSpec.

Impact

Informational.

Recommendation

Add `@return` comment for returned values: https://docs.soliditylang.org/en/develop/natspec-format.html#tags. If your function returns multiple values use multiple `@return` statements in the same format as the `@param` statements.

Developer Response

Fixed in this commit for all functions lacking documentation of an input arg.

7. Informational - Possible denial of service in `setProtocolPaused()`

`setProtocolPaused()` has a for loop that loops through `allMarkets`. If an extremely large number of markets is added to the protocol, it may become impossible to call with function without hitting the maximum gas limit for the chain on which the contract is deployed.

Technical Details

`setProtocolPaused()` loops through `allMarkets` without allowing the user to specify which markets to pause with a function argument. This can cause a problem is a large number of markets exist in the protocol, because the function may revert before reaching the end of the loop. This is likely a theoretical issue given the gas limit of Polygon PoS is currently 30 million while the gas limit of zkEVM is currently 20 million.

Impact

Informational.

Recommendation

If the protocol plans to support adding hundreds of markets in the future, add gas tests to make sure the protocol never reaches a point where certain function revert while looping through all markets.

Developer Response

In case that we add hundreds of markets, we will probably change the implementation of this function and hire yAudit for another audit :)

8. Informational - Typos

Some typos exist in comments.

Technical Details

- involed -> invoked
- move parenthesis from old comment line to the following line that reads “dynamic liquidation incentive”
- `autoCollaterize` -> `autoCollateralize`

Impact

Informational.

Recommendation

Fix typos.

Developer Response

`involed` is fixed in this commit, but renaming `autoCollaterize` is left out as it is used on multiple places, including contracts and scripts interacting with the Comptroller and changing its name might break uncertain number of contracts and scripts.

Final remarks

While the scope of the review was limited, substantial thought and analysis went into the liquidation incentive changes. The implementation logic matches that logic from the Toxic Liquidation Spiral whitepaper and may provide the first improvement on the Compound Finance liquidation logic since the issue of toxic liquidation spirals were first mentioned in an audit report in 2019. There are obviously some side effects of introducing a dynamic liquidation incentive, such as reduced profit incentive for liquidators as the LTV increases. In the unlikely case that the LTV exceeds 100% and there is no incentive for liquidators to liquidate underwater positions, the governance team has stated they will step in to clear bad debt in the protocol. While the completeness level of some of the code and tests were not ideal, especially in PR 60, the remainder of the review went smoothly.

The main two suggestions that aren’t in specific findings in the report are to:

1. Update tests so that all tests are passing with the changes from the reviewed PRs. Tests are one of the first lines of defense from bugs, and the tests were not passing at the time of the review. Consider adding some fuzz tests such as these.
2. Create a comprehensive list of design differences between KEOM and Compound Finance. During this review, it was observed that KEOM has customized the reward distribution functions in Comptroller and added the `autoCollaterize` bool. Other custom changes are also likely and a list of such changes would be useful for any future review of the protocol.