CERTIK

CertiK Audit Report for bZx

Contents

Disclaimer

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

About CertiK

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

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

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

Executive Summary

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

The auditing process pays special attention to the following considerations:

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

Testing Summary

SECURITY LEVEL

Smart Contract Audit

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

This audit was conducted to discover issues and vulnerabilities in the source code of bZx's Smart Contracts.

TYPE	Smart Contract
SOURCE CODE	https://github.com/bZxNetwork/contractsV2
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Jun 07, 2020
DELIVERY DATE	Sept 04, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by the bZx team to audit the design and implementation of their Smart Contracts and its compliance with the EIPs it is meant to implement.

The audited source code link is:

- https://github.com/bZxNetwork/contractsV2/tree/471dce9bbbf4f8204c0fb5a13a8f9c38babd502e

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

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

Documentation

The sources of truth regarding the operation of the contracts in scope were minimal although the token fulfilled a simple use case we were able to fully assimilate. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

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

Summary

The codebase of the project is a DeFi implementation, meaning that the audit had to focus on the inter-relationships of the various contracts as well as the proper functionality of each contract atomically.

As the contracts are meant to be constantly utilized and a raising trend can be seen in the Ethereum market with regards to transaction fees, we combined the security audit with a due diligence inspection on the codebase to ensure that it is as optimized as it can be in relation to the gas costs it incurs.

Although **certain optimization steps** that we pinpointed in the source code mostly referred to coding standards and inefficiencies, **the eight minor flaws** that were identified **should be remediated as soon as possible to ensure the security of the contracts.**

The codebase of the project strictly adheres to the standards and interfaces imposed by the OpenZeppelin open-source libraries and as such its typical ERC-20 functions **can be deemed to be secure**. **However the custom functionality built on top of it possessed flaws** we identified.

Conclusion

Overall, the codebase of the contracts has been refactored assimilating the findings of this report and **achieving a high standard of code quality and security**.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Redundant Utilization of `SafeMath`	Optimization	Informational	GasTokenUser.sol: L17-L61

[INFORMATIONAL] Description:

The Chi token integration implementation defined in their announcement page purposefully does not utilize the `SafeMath` methods as the calculations are guaranteed to be safe and they need to cost as little gas as possible.

Recommendations:

We advise the team to simply use basic mathematical operations instead of using `SafeMath` invocations.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Redundant `require` Check	Optimization	Informational	AdvancedToken.sol: L56

[INFORMATIONAL] Description:

The `sub` invocation of L60 already checks the conditional of the linked line and also exposes a separate method that accepts a second `string` argument that can be passed to it. which we advise.

Recommendations:

We advise the removal of the `require` statement and the use of the exposed second `string` argument of the `sub` invocation as the error message.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Comment	Coding Style	Informational	AdvancedToken.sol: L57

[INFORMATIONAL] Description:

The linked comments state that there is no need to require `value <= totalSupply` yet that is exactly what occurs on L67 due to the `sub` invocation.

Recommendations:

We advise the removal of the ambiguous comment.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Approval Race Condition	Language Specific Issue	Minor	AdvancedToken.sol: L14-L23

[MINOR] Description:

The ERC-20 `approve` function has a well-known race condition whereby an adjustment of an existing approval can be exploited to utilize the full amount of the previously set approval as well as the newly set approval by monitoring the transaction mempool.

Recommendations:

A workaround to this issue is usually `require`-ing that the `allowed` value between two addresses is first set to zero before being adjusted to another value.

Alleviations:

The team opted to implement two functions, `increaseApproval` and `decreaseApproval`, complying to the EIP20.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Unusual Logical Branch	Coding Style	Informational	AdvancedToken.sol: L61-L64

[INFORMATIONAL] Description:

The aforementioned branch indicates that a user should not burn an amount that would reduce his balance to less than `10`. The rationale behind this branch is unaccounted for.

Recommendations:

We advise its inclusion as a comment.

Alleviations:

The team opted to add a comment describing the procedure.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Potentially Misleading Comments	Coding Style	Informational	AdvancedTokenStorage.sol: L14, L21, L28, L36

[INFORMATIONAL] Description:

The comment lines included above refer to the topic of each `event` that subcedes them, however this hash will change should any of the variable types or the `event` name itself are changed. These types of values are better calculated by external code post-compilation rather tha included as comments since they may not be up to date.

Recommendations:

We advise the removal of ambiguous comments.

Alleviations:

The team opted to remove unnecessary comments.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Storage Layout Optimization	Optimization	Informational	LoanTokenBase.sol: L32

[INFORMATIONAL] Description:

The order of declaration within a contract matters in Solidity as state variables are tightly packed into 256-bit slots to the greatest extent possible. In the linked contract, it is possible to move the `lastSettleTime_` declaration of L32 before `loanTokenAddress`.

An `address` in Solidity is 160-bits, the preceding `decimals` variable occupies 8-bits and that leaves a total of 88-bits to be occupied by another variable. The current implementation utilizes a `uint256` for storing the `lastSettleTime_`, however unix timestamps, which `block.timestamp` is, do not need such a level of precision as even contemporary systems store timestamps in 64-bit at most representations.

Recommendations:

We advise the team to move the `lastSettleTime_` declaration before `loanTokenAddress` and to change the data type of `lastSettleTime_`.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Visibility Specifiers Missing	Coding Style	Informational	LoanTokenLogicDai.sol: L15

[INFORMATIONAL] Description:

The linked variables do not have a visibility specifier set.

Recommendations:

We advise that an explicit visibility specifier is defined to aid in the legibility of the codebase.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Magic Numbers	Coding Style	Informational	LoanTokenLogicDai.sol: L223, L224, L234, L284, L290, L321, L484, L485, L488

[INFORMATIONAL] Description:

The linked magic numbers should be set as `constant` and `internal` contract variables with a self-explanatory variable name as well as accompanying comments when necessary. This type of declaration is functionally equivalent to the current implementation as `constant` variables that are `internal` or `private` are simply replaced in the codebase with their literal value.

Additionally, some of these magic numbers are also highlighted in `LoanTokenLogicStandard` and could reside there as the `LoanTokenLogicDai` contract inherits from it.

Recommendations:

We advise the team adds proper documentation specifying the purpose of the linked numbers.

Alleviations:

The team opted to add descriptive `constant` variable names and in-line comments.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Function Inconsistency	Coding Style	Informational	LoanTokenLogicDai.sol: L162, L175, L192, L204

[INFORMATIONAL] Description:

The aforementioned functions both contain a `bytes memory` as the last argument, yet the former does not name it and instead passes the value literal `""` to the final `_borrowOrTrade` invocation and the latter explicitly names it and passes it directly.

Recommendations:

We advise that one of the two paradigms is utilized in both functions.

Alleviations:

No alleviations.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Potentially Dangerous Burn Implementation	Volatile Code	Minor	LoanTokenLogicDai.sol: L311-L313

[MINOR] Description:

The `_burnToken` function contains a conditional check that burns the full balance of an address in case the balance specified exceeds the user's balance. This is dangerous the value can be greater than the balance of a user for a multitude of reasons including mistypes (i.e. an extra zero appended), transaction race conditions (i.e. pre-approved tokens being consumed by another party while the user attempts to transact with the contract) etc.

Recommendations:

The conditional check should instead use the maximum representation of a `uint256` in an equality conditional as the "flag" that indicates a user's full balance should be burned.

Alleviations:

The team opted to add a `require` statement, following our references.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Unsafe `SafeMath` Invocation	Coding Style	Informational	LoanTokenLogicDai.sol: L473

[INFORMATIONAL] Description:

The implementation of `_nextBorrowInterestRate2` in the parent `LoanTokenLogicStandard` contract indicates that the result of `_utilizationRate` can be greater than `100 ether` as it is handled gracefully whilst the invocation on the linked line would halt execution if the utilization rate is greater than `100 ether`.

Recommendations:

Whether a similar graceful handling should be imposed in the `_supplyInterestRate` of `LoanTokenLogicDai` should be evaluated to ensure consistency across the codebase.

Alleviations:

A similar graceful handling mechanism was introduced in LoanTokenLogicDai ensuring that the transaction will not fail if the `utilizationRate` is greater than `100 ether`.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Assembly Conditional	Coding Style & Optimization	Informational	LoanTokenLogicDai.sol: L543

[INFORMATIONAL] Description:

The assembly conditional in the linked line does not conform to the `rpow` implementation of DSMath where this code segment was converted to assembly from.

Recommendations:

The same `and` clause imposed on L549 should exist here as well, as the DSMath implementation uses `rmul` which in-turn uses a safe `mul` operation.

Alleviations:

The team opted to change the codebase.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Redundant Named Return Variables	Optimization	Informational	LoanTokenLogicStandard.sol: L44, L436, L450, L623, L1148

[INFORMATIONAL] Description:

The linked code segments contain named return variables for functions that do not utilize them.

Recommendations:

We advise the team to either remove or properly utilize the name variables.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Function to `modifier`	Optimization	Informational	LoanTokenLogicStandard.sol: L1162-L1173, L738-L748

[INFORMATIONAL] Description:

The first linked code segment can be converted to a `modifier` instead as it is executed at the start of a function's execution and imposes a `require` check on the function being called. Additionally, it could accept a `functionSignature` argument instead of directly utilizing `msg.sig`.

The second linked code segment can also be converted to a `modifier` as a modifier is able to alter the contract state and the `_settleInterest()` function always precedes any state-affecting statements.

Recommendations:

We advise the team to implement the two `modifiers` as described and properly introduce them to the codebase.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Redundant Assignments	Optimization	Informational	LoanTokenLogicStandard.sol: L165-L172, L226-L236

[INFORMATIONAL] Description:

On each linked segment, the value assignment to `sentAmounts[1]` is overridden in the line the previously-assigned value is utilized.

Recommendations:

We advise that the value is utilized directly instead.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Magic Numbers	Coding Style	Informational	LoanTokenLogicStandard.sol: L180, L259, L275, L288, L416, L422, L560, L561, L574, L637, L657, L688, L723, L788, L793, L794, L851, L984, L997, L999, L1014, L1015, L1114, L1115, L1129, L1136, L1139, L1185, L1186, L1189, L1190, L1203

[INFORMATIONAL] Description:

The linked magic numbers should be set as `constant` and `internal` contract variables with a self-explanatory variable name as well as accompanying comments when necessary. This type of declaration is functionally equivalent to the current implementation as `constant` variables that are `internal` or `private` are simply replaced in the codebase with their literal value.

Recommendations:

We advise the team adds proper documentation specifying the purpose of the linked numbers.

Alleviations:

The team opted to add descriptive `constant` variable names and in-line comments on the non-obvious magic numbers.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Function Inconsistency	Volatile Code	Informational	LoanTokenLogicStandard.sol: L136, L184, L197, L245

[INFORMATIONAL] Description:

The aforementioned functions both contain a `bytes memory` as the last argument, yet the former does not name it and instead passes the value literal `""` to the final `_borrowOrTrade` invocation and the latter explicitly names it and passes it directly.

Recommendations:

We advise that one of the two paradigms is utilized in both functions.

Alleviations:

No alleviations.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Arbitrary Transfer Control	Optimization	Informational	LoanTokenLogicStandard.sol: L274-L276

[INFORMATIONAL] Description:

Although intended as a consequence of the way the contract works, any loan pool is able to arbitrarily transfer user funds.

Recommendations:

The necessity of enabling unrestricted movement of funds in contrast to imposing a hard-limit should be evaluated.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 20

TITLE	TYPE	SEVERITY	LOCATION
Redundant `require` and `SafeMath` Utilizations	Optimization	Informational	LoanTokenLogicStandard.sol: L289, L290, L296, L301-L302

[INFORMATIONAL] Description:

The paired code segments (via an ampersand `&`) showcase `require` statements that compare two values that are subsequently utilized in a `SafeMath` operation where the exact same conditional is evaluated in.

Recommendations:

These conditionals can safely be replaced by a single `SafeMath` invocation that also passes in an error message in case the inner `require` statements of the invocation fail.

Alleviations:

The team opted to remove unnecessary `require` statements.

Exhibit 21

TITLE	TYPE	SEVERITY	LOCATION
Literal to `constant`	Optimization	Informational	LoanTokenLogicStandard.sol: L342-L344, L380-L383, L1166-L1167, L1220-L1225

[INFORMATIONAL] Description:

The linked literals can be safely set to `constant` contract-level variables as their assignment and subsequent utilization is replaced with their literal value within the codebase by the compiler, meaning that using a `constant` variable and a value literal are functionally equivalent with the former being far more readable.

Recommendations:

We advise the team to change the linked variables to `constant`.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 22

TITLE	TYPE	SEVERITY	LOCATION
`if` Block Optimization	Optimization	Informational	LoanTokenLogicStandard.sol: L347-L357

[INFORMATIONAL] Description:

The current `if` block can be optimized by breaking the first conditional into two `if` clauses. The outermost `if else if` clause would be set to an `if else` clause with the conditional being `_newBalance != 0` and an inner `if` block would be set that evaluates `_oldBalance != 0`. This would cause the `else` clause to not need an additional conditional check as it is already checked under all circumstances.

Recommendations:

We advise the team to change the code blocks to match the procedure described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 23

TITLE	TYPE	SEVERITY	LOCATION
Named Return Variable	Optimization	Informational	LoanTokenLogicStandard.sol: L393-L431

[INFORMATIONAL] Description:

The variable `profitSoFar` could be an explicitly named return variable to optimize the legibility of the codebase.

Recommendations:

We advise the team to change the variable `profitSoFar` to a named return variable.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 24

TITLE	TYPE	SEVERITY	LOCATION
`uint256` to `int256` Representation	Optimization	Informational	LoanTokenLogicStandard.sol: L393-L431

[INFORMATIONAL] Description:

The current implementation does not account for negative profit and instead zeroes out the profit in case it goes in the negative range. As profit appears to be a view-only indicator, its representation in the negative range should be evaluated as an `int256` representation may be more fitting.

This would also in-turn render the `if` statements of this block redundant.

Recommendations:

We advise the team to change the code block as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 25

TITLE	TYPE	SEVERITY	LOCATION
Redundant `SafeMath` Invocation	Optimization	Informational	LoanTokenLogicStandard.sol: L455-L465

[INFORMATIONAL] Description:

The `if` statement that precedes the `sub` invocation renders the computation safe under all circumstances and as such can be safely omitted.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 26

TITLE	TYPE	SEVERITY	LOCATION
Unsafe Ether Transfer	Volatile Code	Minor	LoanTokenLogicStandard.sol: L691-L695

[MINOR] Description:

The `_mintToken` function incorrectly evaluates the amount of ether sent to the contract and thus can lead to a user arbitrarily siphoning ether off the contract. The linked `if else` branch checks whether `msg.value` is zero and if not, it assumes that `msg.value` is equal to `depositAmount` whilst this is not guaranteed.

Recommendations:

A `require` check should be introduced in the `else` clause that ensures these values are equal, otherwise it is possible to utilize an arbitrary amount of ether stored on the contract.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 27

TITLE	TYPE	SEVERITY	LOCATION
Non-Payable Function w/ `msg.value`	Volatile Code	Minor	LoanTokenLogicStandard.sol: L39-L50

[MINOR] Description:

The function `_mintToken` is an `internal` function that is solely utilized in the `mint` function of the contract. As `mint` is not declared as `payable`, `msg.value` within `_mintToken` will always be equal to zero yet its implementation indicates otherwise.

Recommendations:

We advise that the proper function attribute is set for the `mint` function.

Alleviations:

No alleviations.

Exhibit 28

TITLE	TYPE	SEVERITY	LOCATION
Potentially Dangerous Burn Implementation	Volatile Code	Minor	LoanTokenLogicStandard.sol: L706-L736

[MINOR] Description:

The `_burnToken` function contains a conditional check that burns the full balance of an address in case the balance specified exceeds the user's balance. This is dangerous the value can be greater than the balance of a user for a multitude of reasons including mistypes (i.e. an extra zero appended), transaction race conditions (i.e. pre-approved tokens being consumed by another party while the user attempts to transact with the contract) etc.

Recommendations:

The conditional check should instead use the maximum representation of a `uint256` in an equality conditional as the "flag" that indicates a user's full balance should be burned.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 29

TITLE	TYPE	SEVERITY	LOCATION
Illegible Conditional	Optimization	Informational	LoanTokenLogicStandard.sol: L764-L769

[INFORMATIONAL] Description:

The conditional check imposed should evaluate whether the `sourceToDestRate` is not equal to zero rather than the `sourceToDestPrecision` as it makes more sense legibility-wise.

Recommendations:

We advise the team to change the conditional check as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 30

TITLE	TYPE	SEVERITY	LOCATION
Comment Inconsistency	Coding Style	Informational	LoanTokenLogicStandard.sol: L786-L796

[INFORMATIONAL] Description:

The formula described by the comment is not what is implemented by the statements that subcede it:

*The formula of the comments is defined as: borrowAmount * 10^18 / (10^18 - interestRate * 7884000 * 10^18 / 31536000 / 10^20)*

*The formula of the implementation is defined as: borrowAmount * 10^18 / (10^18 - (interestRate * initialLoanDuration * 10^18 / (31536000 * 10^20)))*

Recommendations:

We advise that those two segments are kept in sync.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 31

TITLE	TYPE	SEVERITY	LOCATION
Unreachable Statements	Optimization	Informational	LoanTokenLogicStandard.sol: L821-L823

[INFORMATIONAL] Description:

The `internal` function `_borrowOrTrade` is solely invoked in two instances, both of which have a non-zero value set in the second position of the `sentAddresses` array. This means that the linked code segment should be omitted as it will never execute.

This appears to be leftover code before this assignment was introduced.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

No alleviations.

Exhibit 32

TITLE	TYPE	SEVERITY	LOCATION
Redundant Ternary Operator	Optimization	Informational	LoanTokenLogicStandard.sol: L845-L847, L856-L858

[INFORMATIONAL] Description:

The ternary operator in essence utilizes the result of its conditional evaluation and as such, the conditional evaluation can be utilized directly.

Additionally, as it is utilized twice its evaluation could be stored to an in-memory `bool` variable.

Recommendations:

We advise the team to store the value of the expression `withdrawAmount != 0` into a `bool` variable and use this variable instead of the ternary operators.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 33

TITLE	TYPE	SEVERITY	LOCATION
Inexistent Error Codes	Coding Style	Informational	LoanTokenLogicStandard.sol: L903-L905

[INFORMATIONAL] Description:

Error messages/codes are essential for debugging.

Recommendations:

As error codes are utilized throughout the codebase, we advise that error codes are also added to the linked code segments instead of empty strings.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 34

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `if` Block	Optimization & Volatile Code	Informational	LoanTokenLogicStandard.sol: L1061-L1064

[INFORMATIONAL] Description:

The linked `if` block should instead be moved inside the `else` clause of the subceding `if else` code block as the evaluation of `utilRate > 90 ether` to `true` is impossible if `utilRate < 80 ether` is `true`.

Recommendations:

We advise the team to change the code block as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 35

TITLE	TYPE	SEVERITY	LOCATION
Total `owner` Control	Optimization & Volatile Code	Informational	LoanTokenLogicStandard.sol: L1211-L1248

[INFORMATIONAL] Description:

The linked function implementation allows the `owner` of the contract to act on behalf of it arbitrarily, i.e. transfer its full token balance.

Recommendations:

We advise the team to carefully evaluate if necessity of such a centralized function is necessary.

Alleviations:

No alleviations.

Exhibit 36

TITLE	TYPE	SEVERITY	LOCATION
Unconventional Naming Convention	Coding Style	Informational	LoanTokenLogicStandard.sol: L1004

[INFORMATIONAL] Description:

The linked function is a `public` getter function yet it is prefixed with an underscore (`_`) indicating that it is an `internal` function. Its optimum visibility specifier should be assessed and defined.

Recommendations:

We advise the team to follow the standard Solidity naming conventions or revise the visibility specifier.

Alleviations:

The team opted to change the visibility specifier of the `_supplyInterestRate` function to `internal`.

Exhibit 37

TITLE	TYPE	SEVERITY	LOCATION
Inexistent Error Codes	Coding Style	Informational	LoanTokenLogicWeth.sol: L84

[INFORMATIONAL] Description:

Error messages/codes are essential for debugging.

Recommendations:

As error codes are utilized throughout the codebase, we advise that error codes are also added to the linked code segments instead of empty strings.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 38

TITLE	TYPE	SEVERITY	LOCATION
Literal to `constant`	Optimization	Informational	LoanTokenSettings.sol: L34-L38

[INFORMATIONAL] Description:

The linked literals can be safely set to `constant` contract-level variables as their assignment and subsequent utilization is replaced with their literal value within the codebase by the compiler, meaning that using a `constant` variable and a value literal are functionally equivalent with the former being far more readable.

Recommendations:

We advise the team to change the variables to `constant` ones.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 39

TITLE	TYPE	SEVERITY	LOCATION
Redundant Implementations	Optimization	Informational	LoanTokenSettings.sol: L51-L87

[INFORMATIONAL] Description:

The `recoverEther` and `recoverToken` implementations are redundant as the `updateSettings` function of `LoanTokenLogicStandard` allows the `owner` to carry out the exact same functionality by setting the appropriate parameters.

Recommendations:

We advise the team to remove redundant code.

Alleviations

No alleviations.

Exhibit 40

TITLE	TYPE	SEVERITY	LOCATION
Redundant `require` and `SafeMath` Utilizations	Optimization	Informational	LoanTokenSettings.sol: L95, L100

[INFORMATIONAL] Description:

The paired code segments (via an ampersand `&`) showcase `require` statements that compare two values that are subsequently utilized in a `SafeMath` operation where the exact same conditional is evaluated in.

Recommendations:

We advise the team to remove redundant `require` statements.

Alleviations:

The team opted to remove the redundant code of the `require` statement.

Exhibit 41

TITLE	TYPE	SEVERITY	LOCATION
Literal to `constant`	Optimization	Informational	LoanTokenSettingsLowerAdmin.sol: L21-L24, L128-L129

[INFORMATIONAL] Description:

The linked literals can be safely set to `constant` contract-level variables as their assignment and subsequent utilization is replaced with their literal value within the codebase by the compiler, meaning that using a `constant` variable and a value literal are functionally equivalent with the former being far more readable.

Recommendations:

We advise the team to change the variables to `constant` ones.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 42

TITLE	TYPE	SEVERITY	LOCATION
Magic Numbers	Optimization	Informational	LoanTokenSettingsLowerAdmin.sol: L71, L113, L114

[INFORMATIONAL] Description:

The linked magic numbers should be set as `constant` and `internal` contract variables with a self-explanatory variable name as well as accompanying comments when necessary. This type of declaration is functionally equivalent to the current implementation as `constant` variables that are `internal` or `private` are simply replaced in the codebase with their literal value.

Recommendations:

We advise the team adds proper documentation specifying the purpose of the linked numbers.

Alleviations:

The team opted to add descriptive `constant` variable names and in-line comments.

Exhibit 43

TITLE	TYPE	SEVERITY	LOCATION
Identical Functions	Optimization	Informational	LoanTokenSettingsLowerAdmin.sol: L38-L80

[INFORMATIONAL] Description:

As the linked functions are identical, they could utilize a single `internal` function that accepts extra arguments passed in by the said functions.

Recommendations:

We advise the team to remove redundant code and implement the single function, as described.

Alleviations:

The team opted to remove redundant code and implement a single function, as described.

Exhibit 44

TITLE	TYPE	SEVERITY	LOCATION
Inexistent Error Codes	Coding Style	Informational	LoanTokenSettingsLowerAdmin.sol: L113-L114

[INFORMATIONAL] Description:

Error messages/codes are essential for debugging.

Recommendations:

As error codes are utilized throughout the codebase, we advise that error codes are also added to the linked code segments instead of empty strings.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 45

TITLE	TYPE	SEVERITY	LOCATION
Unsanitized Input	Volatile Code	Informational	LoanTokenSettingsLowerAdmin.sol: L103-L120

[INFORMATIONAL] Description:

The `_low` prefixed variables of each input should respectively be lower than its non-prefixed input, however this is not sanitized / checked within the function.

Recommendations:

We advise that the corresponding `require` checks are imposed.

Alleviations:

No alleviations.

Exhibit 46

TITLE	TYPE	SEVERITY	LOCATION
Magic Numbers	Coding Style	Informational	State.sol: L48, L52, L56, L63, L65, L73, L75

[INFORMATIONAL] Description:

The linked magic numbers can be optimized by setting the value of `10**18` itself to a literal representing "100%" and subsequently multiplying this value by fractionals. In Solidity, expressions like `0.2 * 10**18` are possible and properly evaluate as they are replaced with their value literal by the Solidity compiler, so `15 * 10**16` is much better legible as `0.15 * PERCENTAGE_MULTIPLIER` or something similar.

Recommendations:

We advise the team adds proper documentation specifying the purpose of the linked numbers.

Alleviations:

The team opted to add descriptive `constant` variable names and in-line comments.

Exhibit 47

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `struct` Layout	Optimization	Informational	LoanStruct.sol: L14, L17-18

[INFORMATIONAL] Description:

The first linked variable could be moved to the bottom of the `struct` declaration to be tightly-packed with another `address` variable, reducing the total space required by the `struct` by one full word (256-bit) slot.

The latter two linked variables can be reduced in precision as they are meant to represent a unix timestamp and even contemporary systems utilize 64-bit precision for representing them, meaning 128-bits of precision would be sufficient. This would once again reduce the total space required by the `struct` by one full word (256-bit) slot.

Recommendations:

We advise the team to change the code block, as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 48

TITLE	TYPE	SEVERITY	LOCATION
Potentially Misleading Comments	Coding Style	Informational	LoanOpeningsEvents.sol: L11, L26, L42

[INFORMATIONAL] Description:

The comment lines included above refer to the topic of each `event` that subcedes them, however this hash will change should any of the variable types or the `event` name itself are changed. These types of values are better calculated by external code post-compilation rather tha included as comments since they may not be up to date

Recommendations:

We advise the team to remove unnecessary comments.

Alleviations:

The team opted to remove unnecessary code.

Exhibit 49

TITLE	TYPE	SEVERITY	LOCATION
Redundant `indexed` Variable	Coding Style	Informational	PriceFeeds.sol: L23

[INFORMATIONAL] Description:

The variable `isPaused` is a `bool` variable with only two possible values. As such, it is redundant to declare it as an `indexed` event variable.

Recommendations:

We advise the team to remove the `indexed` specifier from the `event` declaration.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 50

TITLE	TYPE	SEVERITY	LOCATION
Magic Numbers	Coding Style	Informational	PriceFeeds.sol: L63, L110, L167, L209, L218, L223, L230, L356, L368, L372, L377, L378, L390, L413

[INFORMATIONAL] Description:

The linked magic numbers should be set as `constant` and `internal` contract-level variables with a self-explanatory variable name as well as accompanying comments when necessary. This type of declaration is functionally equivalent to the current implementation as `constant` variables that are `internal` or `private` are simply replaced in the codebase with their literal value.

Recommendations:

We advise the team adds proper documentation specifying the purpose of the linked numbers.

Alleviations:

The team opted to add descriptive `constant` variable names and in-line comments.

Exhibit 51

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Ether Checks	Optimization	Informational	PriceFeeds.sol: L130, L269

[INFORMATIONAL] Description:

The `amountInEth` function checks whether the `tokenAddress` represents Ether by comparing it with the `wethToken` address, yet the conditional on `getFastGasPrice` additionally checks the value against the zero address.

This is also evident in the `constructor` of the contract.

Recommendations:

Proper "Ether" address handling should be implemented across the board.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 52

TITLE	TYPE	SEVERITY	LOCATION
`if` Block Optimization	Optimization	Informational	PriceFeeds.sol: L226-L239

[INFORMATIONAL] Description:

The `if` block can be optimized whereby the `else` clause is dropped entirely and the statements within the `if` clause simply assign the result of `collateralToLoanAmount.sub(loanAmount).mul(10**20).div(loanAmount)` to `currentMargin`, as the function contains named return variables.

Recommendations:

We advise the team to change the code block, as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 53

TITLE	TYPE	SEVERITY	LOCATION
Identical Code Blocks	Optimization	Informational	PriceFeeds.sol: L347-L369

[INFORMATIONAL] Description:

The linked code blocks could instead utilize a single `internal` function for ease-of-maintainability as well as code readability.

Recommendations:

We advise the team to remove redundant code and implement the single function, as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 54

TITLE	TYPE	SEVERITY	LOCATION
Undocumented Literal	Coding Style	Informational	PriceFeeds.sol: L412

[INFORMATIONAL] Description:

The `priceFeeds` of `address(1)` is meant to represent `gasPrice`, yet this remains undocumented throughout the codebase.

Recommendations:

A relevant `constant` for the address should be set along with documentation surrounding it.

Alleviations:

The team opted to add descriptive `constant` variable names and in-line comments.

Exhibit 55

TITLE	TYPE	SEVERITY	LOCATION
Loop Optimization	Optimization	Informational	EnumerableBytes32Set.sol: L141-L143

[INFORMATIONAL] Description:

Instead of starting the loop at `0` and performing an addition on each iteration w/ `start`, it is possible to simply start the loop from `start`, change the conditional to less-than `end` and utilize `i` directly.

Recommendations:

We advise the team to implement the loop as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 56

TITLE	TYPE	SEVERITY	LOCATION
Magic Numbers	Coding Style	Informational	FeesHelper.sol: L27, L39, L132, L134

[INFORMATIONAL] Description:

The linked magic numbers should be set as `constant` and `internal` contract-level variables with a self-explanatory variable name as well as accompanying comments when necessary. This type of declaration is functionally equivalent to the current implementation as `constant` variables that are `internal` or `private` are simply replaced in the codebase with their literal value.

Recommendations:

We advise the team adds proper documentation specifying the purpose of the linked numbers.

Alleviations:

The team opted to add descriptive `constant` variable names and in-line comments.

Exhibit 57

TITLE	TYPE	SEVERITY	LOCATION
Unredeemed Fees	Coding Style	Informational	FeesHelper.sol: L27, L39, L132, L134

[INFORMATIONAL] Description:

Should the price feed stop functioning correctly, the usage of the low-level `staticcall` and consequent `assembly` block would render fees unredeemable during this period.

Recommendations:

As this appears to be intended, the consequences of this should be carefully evaluated and explained in the surrounding documentation.

Alleviations:

No alleviations.

Exhibit 58

TITLE	TYPE	SEVERITY	LOCATION
Magic Numbers	Coding Style	Informational	InterestUser.sol: L29, L59

[INFORMATIONAL] Description:

The linked magic numbers should be set as `constant` and `internal` contract-level variables with a self-explanatory variable name as well as accompanying comments when necessary. This type of declaration is functionally equivalent to the current implementation as `constant` variables that are `internal` or `private` are simply replaced in the codebase with their literal value.

Recommendations:

We advise the team adds proper documentation specifying the purpose of the linked numbers.

Alleviations:

The team opted to add descriptive `constant` variable names and in-line comments.

Exhibit 59

TITLE	TYPE	SEVERITY	LOCATION
Formula Inconsistency	Volatile Code	Minor	LiquidationHelper.sol: L33-L49, L54-L62

[MINOR] Description:

The linked code segments contain mathematical formula implementations in pseudo-code in comments as well as in actual Solidity statements. There appear to be discrepancies between the pseudo-code implementations and the actual implementations.

Recommendations:

We advise that these discrepancies are properly evaluated i.e. `(desiredMargin - 0.05)` is actually conducted by subtracting `incentivePercent` instead of a value literal.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 60

TITLE	TYPE	SEVERITY	LOCATION
`revert` to `require`	Optimization	Informational	SwapsImplKyber.sol: L41-L70

[INFORMATIONAL] Description:

The linked `if` block contains an `else` statement that simply `revert`s.

Recommendations:

We advise that the `if` conditional is instead executed as a `require` statement thus removing the necessity of an `if` block.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 61

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Expressions	Optimization	Informational	SwapsImplKyber.sol: L49, L51, L57, L66, L75

[INFORMATIONAL] Description:

The results of `IERC20(sourceTokenAddress)` and `address(this)` could instead be stored to in-memory variables to aid in the legibility of the codebase and render duplicate reads from memory redundant.

Recommendations:

We advise the team to change the code block as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 62

TITLE	TYPE	SEVERITY	LOCATION
Potentially Dangerous Allowance	Volatile Code	Informational	SwapsImplKyber.sol: L50-L55

[INFORMATIONAL] Description:

An unlimited allowance is generally considered ill-practice when dealing with user funds.

Recommendations:

We advise that this block is instead revamped to set a precise allowance to the Kyber contract.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 63

TITLE	TYPE	SEVERITY	LOCATION
Nested to Single Conditional	Optimization	Informational	SwapsImplKyber.sol: L72-L80

[INFORMATIONAL] Description:

The nested `if` blocks can instead be converted to a single `if` block with a joint (`&&`) conditional.

Recommendations:

We advise the team to change the code block as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 64

TITLE	TYPE	SEVERITY	LOCATION
Function Optimization	Optimization	Informational	SwapsImplKyber.sol: L83-L120

[INFORMATIONAL] Description:

The function `internalExpectedRate` can be optimized by explicitly naming the return variable to `expectedRate` causing its default value to be `0`. This would, in turn, require a single `if else` conditional with the first condition remaining as is (`sourceTokenAddress == destTokenAddress`) and the `else-if` condition becoming `sourceTokenAmount != 0`.

Lastly, the `assembly` block can be optimized by simply introducing an `if` conditional that makes sure `result` is `true` and subsequently executes the `assembly` statement `expectedRate := mload(add(data, 32))`.

Recommendations:

We advise the team to change the code block as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 65

TITLE	TYPE	SEVERITY	LOCATION
Conditional Order	Optimization	Informational	SwapsImplKyber.sol: L157-L159

[INFORMATIONAL] Description:

The linked conditional check can be moved before the multiplication and division that precedes it, as the multiplication is done with the value of `bufferMultiplier` which is guaranteed to yield a value different than one when divided by `10**20` and multiplied by any value different than `0`.

Recommendations:

We advise the team to change the code block as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 66

TITLE	TYPE	SEVERITY	LOCATION
Function Visibilities	Coding Style	Informational	SwapsImplKyber.sol: L24-L33, L83-L89

[INFORMATIONAL] Description:

The functions `internalSwap` and `internalExpectedRate` are declared as `public` despite what their naming convention implies.

Recommendations:

We advise that either their visibility or naming convention is amended properly.

Alleviations:

The team opted to change the names of the functions from `internalSwap` and `internalExpectedRate` to `dexSwap` and `dexExpectedRate` respectively.

Exhibit 67

TITLE	TYPE	SEVERITY	LOCATION
Conditional Optimization	Optimization	Informational	SwapsUser.sol: L90-L93

[INFORMATIONAL] Description:

The linked `if` and consequent `require` statements can be optimized into an `else` branch that contains the `require` statement as should the `if` conditional evaluate to `true` the `require` statement will pass under all circumstances.

Recommendations:

We advise the team to change the code block, as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 68

TITLE	TYPE	SEVERITY	LOCATION
`revert` to `require`	Optimization	Informational	SwapsUser.sol: L127-L157

[INFORMATIONAL] Description:

The linked `if` block contains an `else` statement that simply `revert`s.

Recommendations:

We advise that the `if` conditional is instead executed as a `require` statement thus removing the necessity of an `if` block.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 69

TITLE	TYPE	SEVERITY	LOCATION
Commented Out Code	Coding Style	Informational	SwapsUser.sol: L134-L156

[INFORMATIONAL] Description:

The usefulness of the commented out code should be evaluated and, should the code block be considered no longer necessary.

Recommendations:

We advise the team to remove the comments to avoid any type of confusion from the readers of the codebase.

Alleviations:

The team opted to remove unnecessary comments.

Exhibit 70

TITLE	TYPE	SEVERITY	LOCATION
Strict Conditional	Optimization	Informational	SwapsUser.sol: L161

[INFORMATIONAL] Description:

The linked conditional is accompanied by a comment that states `"all of vals[0] (minSourceTokenAmount) must be spent"` whilst the `require` conditional conducts a strict equality.

Recommendations:

We advise this comparison to instead become a greater-than-or-equal (`>=`) comparison.

Alleviations:

No changes were made to the codebase, but the team opted to add descriptive inline-comments as to why this procedure remained unchanged.

Exhibit 71

TITLE	TYPE	SEVERITY	LOCATION
Potentially Incorrect Error Message	Volatile Code	Informational	SwapsUser.sol: L93

[INFORMATIONAL] Description:

The error message states that `"sourceAmount larger than max"` whilst the values that are checked are the minimum and maximum `SourceTokenAmount`s.

Recommendations:

We advise the team to add more descriptive error messages.

Alleviations:

The team opted to change the error message of the `require` statement.

Exhibit 72

TITLE	TYPE	SEVERITY	LOCATION
Duplicate Conditionals	Optimization	Informational	SwapsUser.sol: L100, L104, L120, L158, L162, L171

[INFORMATIONAL] Description:

The linked conditional statements are evaluated multiple times when they can be evaluated once and accessed from memory variables.

Recommendations:

We advise the team to change the code, as described.

Alleviations:

No alleviations.

Exhibit 73

TITLE	TYPE	SEVERITY	LOCATION
Redundant `SafeMath` Utilizations	Optimization	Informational	SwapsUser.sol: L163-L164, L179-L180

[INFORMATIONAL] Description:

The linked mathematical statements can be safely conducted using their "raw" counterparts as they basically offset the already-safely-conducted mathematical operations in the preceding `if` block.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 74

TITLE	TYPE	SEVERITY	LOCATION
`assembly` to Solidity	Optimization & Language Specific Issue	Informational	SwapsUser.sol: L208-L211

[INFORMATIONAL] Description:

The linked `assembly` block reads the values from the returned `bytes memory` and assigns them to the return variables.

Recommendations:

We advise the team to use the function `decode` of the globally available `abi` which can be utilized instead which is more verbose and overall safer.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 75

TITLE	TYPE	SEVERITY	LOCATION
General Codebase Comment	Optimization	Informational	LoanClosings.sol

[INFORMATIONAL] Description:

The codebase of the LoanClosings.sol contract is inefficient in its utilization of `storage` and `memory` pointers. When a `storage` pointer is passed as a `memory` pointer, the full struct is copied to `memory` redundantly. The codebase goes to great lengths to optimize certain statements, i.e. not perform additions with the variable `0` at all, yet conducts high-gas inefficiencies such as utilizing the `storage` pointers of loan structs as `memory` in the `_checkAuthorized` and `_doCollateralSwap` functions.

Additionally, duplicate conditionals are re-evaluated numerous times throughout certain functions such as `_closeWithSwap`, where this conditional could be changed to a greater-than-or-equal (`>=`) conditional that is subsequently stored in memory and utilized in L489 and L490 as well.

Storage read duplications also occur in the codebase, whereby a variable is read from the `storage` pointer multiple times in the codebase whilst it could have been read once and stored to an in-memory variable such as the IDs of loans. All the aforementioned points significantly affect the overall gas cost of the functions.

Recommendations:

We advise the team to revise and restructure the codebase for `LoanClosings.sol`.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 76

TITLE	TYPE	SEVERITY	LOCATION
Fallback Paradigm	Language Specific Issue & Optimization	Informational	LoanClosings.sol: L29-L33

[INFORMATIONAL] Description:

The current fallback function implementation `revert`s to prevent any type of Ethereum transfer to the contract. This paradigm was deployed pre-v0.4.0 because a contract would be able to receive and essentially lock Ethereum as part of regular transfers.

After v0.4.0, a contract automatically throws if no fallback function is provided and as such the linked code can simply be omitted if the desire is to prevent Ether transfers to the contract.

The warning of the linked Solidity docs should be kept in mind whereby a contract is still able to receive Ether either from a `selfdestruct` operation or a block reward.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 77

TITLE	TYPE	SEVERITY	LOCATION
Potential Incompatibility w/ Proxy Model	Coding Style	Informational	LoanClosings.sol: L75

[INFORMATIONAL] Description:

The conditional check on the linked line ensures that only externally owned accounts invoke the particular function, however judging from the codebase the function may be invokable by a proxy and as such, the `msg.sender` may mismatch the `tx.origin`.

Recommendations:

Whether this conditional functions as intended should be evaluated.

Alleviations:

The team opted to remove unnecessary code from the codebase.

Exhibit 78

TITLE	TYPE	SEVERITY	LOCATION
Commented Out Variable	Coding Style	Informational	LoanClosings.sol: L68, L80, L110, L125

[INFORMATIONAL] Description:

As with the other functions we have identified in the codebase to follow a similar convention, we advise that the variable is uncommented and passed directly to the function that follows instead of the literal `""`.

Alleviations:

The team opted to remove unnecessary code from the codebase.

Exhibit 79

TITLE	TYPE	SEVERITY	LOCATION
`if` Optimization	Optimization	Informational	LoanClosings.sol: L172-L178

[INFORMATIONAL] Description:

The linked code block can be optimized whereby the `if-else-if-else` clause is converted to an `if-else` clause with the previously top-level `else-if` clause being nested within as the statement of L175 and L177 is executed in both circumstances.

Recommendations:

We advise the team to change the code block, as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 80

TITLE	TYPE	SEVERITY	LOCATION
Race Condition	Language Specific Issue	Minor	LoanClosings.sol: L46-L64

[MINOR] Description:

The linked function is meant to liquidate a particular loan and transmit any seized funds to the specified `receiver` address. As no form of access control is imposed at this point nor within the `_liquidate` function, any party is able to liquidate loans at will with themselves designated as seized fund recipients as long as the loan is active.

This introduces a race condition whereby an attacker is able to scan the transaction mempool of Ethereum, identify a transaction that is meant to liquidate a particular asset with a high amount of seized funds and transmit an identical transaction themselves with a higher transaction fee to make sure it is executed first.

Recommendations:

We advise that the ACL logic of this function is re-evaluated.

Alleviations:

The team opted to remove unnecessary code from the codebase.

Exhibit 81

TITLE	TYPE	SEVERITY	LOCATION
Usage of Magic Number	Coding Style	Informational	LoanClosings.sol: L258

[INFORMATIONAL] Description:

The `endTimestamp` of a `Loan` is compared against the current time after being subtracted the value of `3600`. The rationale behind this subtraction should be explained in a comment that precedes the line. Additionally, different code paths execute in case the current time is within the range of the subtracted `3600`.

This range should instead become a variable as Ethereum block times vary greatly and in case of a substantial increase in difficulty, a change in the way blocks are produced or any number of codebase changes that are expected to be made in the Ethereum blockchain could affect the window this function is executable in, if at all.

Recommendations:

We advise the team adds proper documentation specifying the purpose of the linked number.

Alleviations:

The team opted to add a descriptive `constant` variable name.

Exhibit 82

TITLE	TYPE	SEVERITY	LOCATION
Ungrouped Calculations	Coding Style	Informational	LoanClosings.sol: L303-L306, L323-L328, L943-L949

[INFORMATIONAL] Description:

As `SafeMath` invocations can be chained, the linked statements can occur in a single assignment by chaining the various operations.

Recommendations:

We advise the team to change the code block, as described above.

Alleviations:

No alleviations.

Exhibit 83

TITLE	TYPE	SEVERITY	LOCATION
Time Units & Magic Numbers	Coding Style	Informational	LoanClosings.sol: L257, L291, L300, L310, L315, L327

[INFORMATIONAL] Description:

Overall, magic numbers should be stored in the form of `constant` contract variables as the generated bytecode would be equivalent to its current state and simply aid in the legibility of the codebase. As the loans rely on multiple time-based concepts, it would be wise to instead use the [time units](https://solidity.readthedocs.io/en/v0.5.17/units-and-global-variables.html#time-units) readily available by Solidity to greatly aid in the legibility of the codebase.

We should note that decimal denominations are also acceptable i.e. `30.45 days` as the compiler takes care of converting them to the corresponding integer representations.

On a final note, as this exhibit is slightly uniform across the codebase a singular contract could be created that simply declares the time units as `constant` variables and is subsequently inherited wherever they are necessary. This would pose no bytecode overhead and would greatly optimize the codebase as for example `UNIX_DAY` is much more readable than `86400`.

Recommendations:

We will not replay this exhibit in other contracts, however it does apply and a common inheritance chain should be defined.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 84

TITLE	TYPE	SEVERITY	LOCATION
Potentially Unwanted `throw`	Volatile Code	Informational	LoanClosings.sol: L322-L323

[INFORMATIONAL] Description:

If a loan's `maxDuration` is smaller than the delta that has passed since `loanLocal.endTimestamp` and `block.timestamp` while `block.timestamp` is less than `loanLocal.endTimestamp + 3600` the linked lines will throw due to the `SafeMath` subtraction invocation.

Recommendations:

We advise the team to evaluate whether or not this is intended.

Alleviations:

No alleviations.

Exhibit 85

TITLE	TYPE	SEVERITY	LOCATION
Duplicate Subtraction	Optimization	Informational	LoanClosings.sol: L340-L350

[INFORMATIONAL] Description:

In the linked lines the function `_doCollateralSwap` is called by passing a pointer to the `storage` of `loanLocal`. The `sourceTokenAmountUsed` is subtracted in both `_doCollateralSwap` here as well as on the linked lines in the caller function, meaning that the value will incorrectly be subtracted twice.

Recommendations:

We advise one of the subtractions is removed.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 86

TITLE	TYPE	SEVERITY	LOCATION
Gas Rebate Lock	Coding Style	Minor	LoanClosings.sol: L362-L377

[MINOR] Description:

The linked lines are meant to rebate the high gas cost of the function by transmitting some tokens to the caller, the transaction itself costing some gas as well. The core issue with this function is that it would perform correctly under optimal circumstances, however a malfunction in the price feed of the gas price or an exceptionally high gas price would render this function non-invocable due to a very high rebate fee.

Recommendations:

We advise that either the rebate system is revamped or that the if conditional also ensures enough collateral exists to rebate instead of conducting a `SafeMath` subtraction invocation that may halt execution.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 87

TITLE	TYPE	SEVERITY	LOCATION
Redundant `SafeMath` Invocation	Optimization	Informational	LoanClosings.sol: L442-L443

[INFORMATIONAL] Description:

The statements that precede the code block ensure that the subtraction will never underflow and thus the `SafeMath` invocation is redundant.

Recommendations:

We advise the `SafeMath` invocation is removed.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 88

TITLE	TYPE	SEVERITY	LOCATION
Redundant Assignment	Optimization	Informational	LoanClosings.sol: L502-L506, L763

[INFORMATIONAL] Description:

All values in Solidity obtain a default value on instantiation usually equal to zero. Thus, a consequent assignment of zero to an already initialized variable is unnecessary and the linked code block can be omitted.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 89

TITLE	TYPE	SEVERITY	LOCATION
Redundant Conditional	Optimization	Informational	LoanClosings.sol: L638-L645

[INFORMATIONAL] Description:

The conditional ensures that the subtraction of L633 yields a non-zero value. This means that it essentially guarantees that `interestRefundToBorrower` is greater-than (`>`) the value of `loanCloseAmountLessInterest`.

Recommendations:

As this is always true due to the conditional of the outer `if` clause on L619, the `if` clause can be safely removed as `interestRefundToBorrower` will always be non-zero.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 90

TITLE	TYPE	SEVERITY	LOCATION
Potentially Redundant Utilization of Assembly	Optimization	Informational	LoanClosings.sol: L841-L856

[INFORMATIONAL] Description:

The rationale behind utilizing a `staticcall` instead of wrapping the `address` to the provided `IPriceFeeds` interface and invoking the function directly should be relayed to us.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

No alleviations.

Exhibit 91

TITLE	TYPE	SEVERITY	LOCATION
Fallback Paradigm	Optimization	Informational	LoanMaintenance.sol: L39-L43

[INFORMATIONAL] Description:

The current fallback function implementation `revert`s to prevent any type of Ethereum transfer to the contract. This paradigm was deployed pre-v0.4.0 because a contract would be able to receive and essentially lock Ethereum as part of regular transfers.

After v0.4.0, a contract automatically throws if no fallback function is provided and as such the linked code can simply be omitted if the desire is to prevent Ether transfers to the contract.

The warning of the linked Solidity docs should be kept in mind whereby a contract is still able to receive Ether either from a `selfdestruct` operation or a block reward.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 92

TITLE	TYPE	SEVERITY	LOCATION
Struct Optimization	Optimization	Informational	LoanMaintenance.sol: L19-L35

[INFORMATIONAL] Description:

The variable `endTimestamp` can safely be grouped with an `address` declaration as a unix timestamp, even on contemporary systems, has at most 64-bits of precision and an `address` leaves up 96 bits that could be utilized by the unix timestamp to store it in a single `storage` slot reducing the number of words the `struct` occupies by one.

Recommendations:

We advise the team to change the data type of variable `endTimestamp` to `uint64`.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 93

TITLE	TYPE	SEVERITY	LOCATION
Provide Error Message	Optimization	Informational	LoanMaintenance.sol: L127-L128

[INFORMATIONAL] Description:

This `SafeMath` subtraction will fail in case the withdrawal amount requested is greater than the amount of available collateral.

Recommendations:

We advise a proper error message is provided as this is a likely scenario.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 94

TITLE	TYPE	SEVERITY	LOCATION
Ungrouped Conditional	Optimization	Informational	LoanMaintenance.sol: L250-L256, L331-L337

[INFORMATIONAL] Description:

Instead of conducting assignments past the first `require` clause, a `&&` conditional pair could be used whereby the second conditional simply subtracts `block.timestamp` and evaluates the second `require` conditional. This is possible because if the first conditional of the pair evaluates to `true` any `SafeMath` utilization will be redundant as the subtraction will never underflow.

Recommendations:

We advise the team to change the `require` statement, as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 95

TITLE	TYPE	SEVERITY	LOCATION
Code Block Optimization	Optimization	Informational	LoanMaintenance.sol: L441-L463, L492-L514

[INFORMATIONAL] Description:

The linked code block can be greatly optimized. First, the `LoanReturnData[]` array instantiation should have a length of `count.min256(set.values.length)` rather than simply `count` as an `assembly` "hack" is utilized to manually reduce the size here.

Secondly, instead of retaining a separate `itemCount` variable it would be possible to instead decrement the `count` variable until it reaches zero and utilize `loansData.length - count` as the index in the assignment instead of `itemCount`.

Finally, the loop conditionals as well as the location the `i` variable is utilized can be adjusted to not require an addition and subtraction on each iteration by adjusting the ends of the `for` loop, i.e. have `i` start at `end - 1` instead of `end - start` and convert the conditional from `i > 0` to `i > start - 1`.

The logic would be equivalent whilst the gas cost will be reduced.

Although the linked blocks concern exhibit A, the same optimizations can be applied in exhibit B.

Recommendations:

We advise the team to change the code block as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 96

TITLE	TYPE	SEVERITY	LOCATION
`if` Block Optimization	Optimization	Informational	LoanMaintenance.sol: L528-L535

[INFORMATIONAL] Description:

The convoluted `if` block can be greatly optimized into two conditionals. As `loanType` can only be `0`, `1` or `2` the conditional can be simplified to a single `if` clause with the expression `(loanType == 1 && loanParamsLocal.maxLoanTerm == 0) || (loanType == 2 && loanParamsLocal.maxLoanTerm != 0)`.

Additionally, as `loanType` can only hold 3 values it would be wise to use an `enum` instead.

Recommendations:

We advise the team to change the code block as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 97

TITLE	TYPE	SEVERITY	LOCATION
Fallback Paradigm	Optimization	Informational	LoanOpenings.sol: L20-L24

[INFORMATIONAL] Description:

The current fallback function implementation `revert`s to prevent any type of Ethereum transfer to the contract. This paradigm was deployed pre-v0.4.0 because a contract would be able to receive and essentially lock Ethereum as part of regular transfers.

After v0.4.0, a contract automatically throws if no fallback function is provided and as such the linked code can simply be omitted if the desire is to prevent Ether transfers to the contract.

The warning of the linked Solidity docs should be kept in mind whereby a contract is still able to receive Ether either from a `selfdestruct` operation or a block reward.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 98

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `return` Variable	Optimization	Informational	LoanOpenings.sol: L468-L536

[INFORMATIONAL] Description:

As the function `_initializeLoan` is called once here, it would be possible to return a `storage` pointer directly instead of an ID that is subsequently looked up again in the `mapping`.

Recommendations:

We advise the team to change the code statement, as described.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 99

TITLE	TYPE	SEVERITY	LOCATION
Redundant `abi.encodePacked` Utilization	Optimization	Informational	LoanOpenings.sol: L487-L492

[INFORMATIONAL] Description:

All variables included in the `abi.encodePacked` invocation cannot be packed under a single 256-bit slot and as such, the invocation is equivalent to `abi.encode` which is more gas efficient. Additionally, when calculating hashes as identifiers it is wise to utilize `abi.encode` instead of `abi.encodePacked` as unaccounted-for tight packs can lead to the same ID being generated with different input variables.

Recommendations:

We advise the team to favor utilizing `abi.encode` over `abi.encodePacked`.

Alleviations:

No alleviations.

Exhibit 100

TITLE	TYPE	SEVERITY	LOCATION
Fallback Paradigm	Optimization	Informational	LoanSettings.sol: L17-L21

[INFORMATIONAL] Description:

The current fallback function implementation `revert`s to prevent any type of Ethereum transfer to the contract. This paradigm was deployed pre-v0.4.0 because a contract would be able to receive and essentially lock Ethereum as part of regular transfers.

After v0.4.0, a contract automatically throws if no fallback function is provided and as such the linked code can simply be omitted if the desire is to prevent Ether transfers to the contract.

The warning of the linked Solidity docs should be kept in mind whereby a contract is still able to receive Ether either from a `selfdestruct` operation or a block reward.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 101

TITLE	TYPE	SEVERITY	LOCATION
Code Block Optimization	Optimization	Informational	LoanSettings.sol: L112-L126

[INFORMATIONAL] Description:

The linked code block can be greatly optimized. First, the `LoanReturnData[]` array instantiation should have a length of `count.min256(set.values.length)` rather than simply `count` as an `assembly` "hack" is utilized to manually reduce the size here.

Secondly, instead of retaining a separate `itemCount` variable it would be possible to instead decrement the `count` variable until it reaches zero and utilize `loansData.length - count` as the index in the assignment instead of `itemCount`.

Finally, the loop conditionals as well as the location the `i` variable is utilized can be adjusted to not require an addition and subtraction on each iteration by adjusting the ends of the `for` loop, i.e. have `i` start at `end - 1` instead of `end - start` and convert the conditional from `i > 0` to `i > start - 1`.

The logic would be equivalent whilst the gas cost will be reduced.

Recommendations:

We advise the team to change the code block as described above.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 102

TITLE	TYPE	SEVERITY	LOCATION
Redundant `abi.encodePacked` Utilization	Optimization	Informational	LoanSettings.sol: L144-L151

[INFORMATIONAL] Description:

All variables included in the `abi.encodePacked` invocation cannot be packed under a single 256-bit slot and as such, the invocation is equivalent to `abi.encode` which is more gas efficient. Additionally, when calculating hashes as identifiers it is wise to utilize `abi.encode` instead of `abi.encodePacked` as unaccounted-for tight packs can lead to the same ID being generated with different input variables.

Recommendations:

We advise the team to favor utilizing `abi.encode` over `abi.encodePacked`.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 103

TITLE	TYPE	SEVERITY	LOCATION
Fallback Paradigm	Optimization	Informational	ProtocolSettings.sol: L20-L24

[INFORMATIONAL] Description:

The current fallback function implementation `revert`s to prevent any type of Ethereum transfer to the contract. This paradigm was deployed pre-v0.4.0 because a contract would be able to receive and essentially lock Ethereum as part of regular transfers.

After v0.4.0, a contract automatically throws if no fallback function is provided and as such the linked code can simply be omitted if the desire is to prevent Ether transfers to the contract.

The warning of the linked Solidity docs should be kept in mind whereby a contract is still able to receive Ether either from a `selfdestruct` operation or a block reward.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 104

TITLE	TYPE	SEVERITY	LOCATION
Invalid `require` Message	Coding Style	Informational	ProtocolSettings.sol: L94

[INFORMATIONAL] Description:

The `require` conditional ensures that either an addition or removal of a pool `address` is being done in the particular loop iteration yet the error message implies that the pool simply does not exist which is not the case for the first part of the "OR" conditional.

Recommendations:

We advise the message be adapted to better represent the check imposed here.

Additionally, we would advise an additional check whereby, in case of an addition, it is ensured that the pool hasn't been added in the past as it would be overwriting storage.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 105

TITLE	TYPE	SEVERITY	LOCATION
Fallback Paradigm	Optimization	Informational	SwapsExternal.sol: L20-L24

[INFORMATIONAL] Description:

The current fallback function implementation `revert`s to prevent any type of Ethereum transfer to the contract. This paradigm was deployed pre-v0.4.0 because a contract would be able to receive and essentially lock Ethereum as part of regular transfers.

After v0.4.0, a contract automatically throws if no fallback function is provided and as such the linked code can simply be omitted if the desire is to prevent Ether transfers to the contract.

The warning of the linked Solidity docs should be kept in mind whereby a contract is still able to receive Ether either from a `selfdestruct` operation or a block reward.

Recommendations:

We advise the team to remove redundant code.

Alleviations:

The team opted to change the codebase according to our references.

Exhibit 106

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `require` Location	Optimization	Informational	SwapsExternal.sol: L55

[INFORMATIONAL] Description:

The `require` statement could instead be relocated to an `else` branch of the preceding `if` clause as the logical check is redundant if the `if` clause evaluates to `true`.

Recommendations:

We advise the team to change the relevant code, as described above.

Alleviations:

The team opted to change the codebase according to our references.