CompliFi

Security Assessment

November 5th, 2020

For :

CompliFi

By :

Alex Papageorgiou @ CertiK

alex.papageorgiou@certik.org

Sheraz Arshad @ CertiK

sheraz.arshad@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	CompliFi
Description	A decentralized derivative issuance protocol with no counterparty risk and no default mechanism by design.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	Pre-audit: 04b9100db934d0706a917dd13f19e3fa3ddeabeb Post-audit: 96f5f7d9ad75f4de8e8d191d42a5d33d311dd4e9

Audit Summary

Delivery Date	October 26th, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	October 14th, 2020 - November 5th, 2020

Vulnerability Summary

Issue Type	Found	Alleviated	Outstanding
Critical	0	0	0
Major	0	0	0
Medium	3	3	0
Minor	8	8	0
Informational	84	58	26
Total	95	69	26

Executive Summary

The codebase comprise of contracts implementing logic for operations related to CompliFi Protocol. The contracts make use of `OpenZeppelin` contracts to implement `ERC20` token, pausable and ownable functionalities.
All of the contracts in the repository were reviewed and majority of the findings are `informational` for enhancing the optimization and code legibility of the contract..
The audit was perfomed on the commit hash `04b9100db934d0706a917dd13f19e3fa3ddeabeb` and the alleviations were added in the commit hash `96f5f7d9ad75f4de8e8d191d42a5d33d311dd4e9`.

Files In Scope

ID	Contract	Location
ARP	AddressRegistryParent.sol	contracts/registries/AddressRegistryParent.sol
COS	CallOptionSplit.sol	contracts/collateralSplits/CallOptionSplit.sol
CSP	CollateralSplitParent.sol	contracts/collateralSplits/CollateralSplitParent.sol
COI	ChainlinkOracleIterator.sol	contracts/oracleIterators/ChainlinkOracleIterator.sol
CSR	CollateralSplitRegistry.sol	contracts/registries/CollateralSplitRegistry.sol
CTR	CollateralTokenRegistry.sol	contracts/registries/CollateralTokenRegistry.sol
COV	ChainlinkOracleIteratorV1.sol	contracts/oracleIterators/ChainlinkOracleIteratorV1.sol
DSN	DerivativeSpecification.sol	contracts/DerivativeSpecification.sol
DSR	DerivativeSpecificationRegistry.sol	contracts/registries/DerivativeSpecificationRegistry.sol
ERC	ERC20PresetMinter.sol	contracts/tokens/ERC20PresetMinter.sol
EIP	EIP20NonStandardInterface.sol	contracts/tokens/EIP20NonStandardInterface.sol
FLR	FeeLogger.sol	contracts/FeeLogger.sol
FLP	FeeLoggerProxy.sol	contracts/FeeLoggerProxy.sol
IFL	IFeeLogger.sol	contracts/IFeeLogger.sol
IST	InsurSplit.sol	contracts/collateralSplits/InsurSplit.sol
ITB	ITokenBuilder.sol	contracts/tokens/ITokenBuilder.sol
IVB	IVaultBuilder.sol	contracts/IVaultBuilder.sol
IER	IERC20Metadata.sol	contracts/tokens/IERC20Metadata.sol
IPV	IPausableVault.sol	contracts/IPausableVault.sol
IOI	IOracleIterator.sol	contracts/oracleIterators/IOracleIterator.sol
IAR	IAddressRegistry.sol	contracts/registries/IAddressRegistry.sol
ICS	ICollateralSplit.sol	contracts/collateralSplits/ICollateralSplit.sol
IOV	IOracleIteratorV1.sol	contracts/oracleIterators/IOracleIteratorV1.sol
IEC	IERC20MintedBurnable.sol	contracts/tokens/IERC20MintedBurnable.sol

ICT	ICollateralSplitTemplate.sol	contracts/collateralSplits/ICollateralSplitTemplate.sol
IDS	IDerivativeSpecification.sol	contracts/IDerivativeSpecification.sol
ORY	OracleRegistry.sol	contracts/registries/OracleRegistry.sol
OIR	OracleIteratorRegistry.sol	contracts/registries/OracleIteratorRegistry.sol
SST	StableSplit.sol	contracts/collateralSplits/StableSplit.sol
TBR	TokenBuilder.sol	contracts/tokens/TokenBuilder.sol
VAU	Vault.sol	contracts/Vault.sol
VBR	VaultBuilder.sol	contracts/VaultBuilder.sol
VFY	VaultFactory.sol	contracts/VaultFactory.sol
VFP	VaultFactoryProxy.sol	contracts/VaultFactoryProxy.sol
X1S	x1Split.sol	contracts/collateralSplits/x1Split.sol
X5S	x5Split.sol	contracts/collateralSplits/x5Split.sol

Findings

ID	Title	Type	Severity	Resolved
DSN-01	Unlocked Compiler Version	Language Specific	Informational	ⓘ
DSN-02	User-Defined Getters	Gas Optimization	Informational	ⓘ
DSN-03	Mutability Specifiers Missing	Gas Optimization	Informational	ⓘ
FLR-01	Unlocked Compiler Version	Language Specific	Informational	ⓘ
FLR-02	Empty Function	Dead Code	Informational	ⓘ
FLP-01	Unlocked Compiler Version	Language Specific	Informational	ⓘ
FLP-02	Literal over `new` Instantiation	Coding Style	Informational	ⓘ
VBR-01	Unlocked Compiler Version	Language Specific	Informational	✓
VBR-02	Function Visibility Optimization	Gas Optimization	Informational	ⓘ
VBR-03	Proxy-over-Implementation Creation	Gas Optimization	Informational	ⓘ
VFY-01	Unlocked Compiler Version	Language Specific	Informational	✓
VFY-02	High-Level of Centralization	Logical Issue	Informational	✓
VFP-01	Unlocked Compiler Version	Language Specific	Informational	ⓘ
VFP-02	Literal over `new` Instantiation	Coding Style	Informational	ⓘ
CSP-01	Unlocked Compiler Version	Language Specific	Informational	✓
CSP-02	Function Visibility Optimization	Gas Optimization	Informational	ⓘ
CSP-03	Unsafe Multiplication	Arithmetic	Minor	✓
COS-01	Unlocked Compiler Version	Language Specific	Informational	✓
COS-02	Function Mutability Optimization	Gas Optimization	Informational	✓
COS-03	`if` Block Optimization	Gas Optimization	Informational	✓
COS-04	Redundant Statements	Dead Code	Informational	✓
COS-05	Invalid Formula in Comment	Arithmetic	Medium	✓
ICS-01	Function Mutability Specifier	Gas Optimization	Informational	✓
ICS-02	Function Visibility Optimization	Gas Optimization	Informational	ⓘ

IST-01	`if` Block Optimizations	Gas Optimization	Informational	✓
IST-02	Redundant Statements	Dead Code	Informational	✓
IST-03	Incorrect Formula	Arithmetic	Minor	✓
IST-04	Function Mutability Optimization	Gas Optimization	Informational	✓
SST-01	Unlocked Compiler Version	Language Specific	Informational	✓
SST-02	Function Mutability Optimization	Gas Optimization	Informational	✓
SST-03	`if` Block Optimization	Gas Optimization	Informational	✓
SST-04	Redundant Statements	Dead Code	Informational	✓
SST-05	Incorrect Formula	Arithmetic	Minor	✓
X1S-01	Unlocked Compiler Version	Language Specific	Informational	✓
X1S-02	Function Mutability Optimization	Gas Optimization	Informational	✓
X5S-01	Unlocked Compiler Version	Language Specific	Informational	✓
X5S-02	Function Mutability Optimization	Gas Optimization	Informational	✓
X5S-03	`if` Block Optimizations	Gas Optimization	Informational	✓
X5S-04	Redundant Statements	Dead Code	Informational	✓
X5S-05	Invalid Formula	Arithmetic	Minor	✓
COI-01	Unlocked Compiler Version	Language Specific	Informational	✓
COI-02	Function Mutability Optimization	Gas Optimization	Informational	✓
COI-03	Loop Exhaustion	Logical Issue	Medium	✓
COI-04	Conditional Optimization	Gas Optimization	Informational	✓
COI-05	`revert` to `require`	Coding Style	Informational	✓
COI-06	Conditional Optimization	Gas Optimization	Informational	✓
COI-07	Redundant `SafeMath` Utilization	Gas Optimization	Informational	✓
COI-08	Statement Relocation	Coding Style	Informational	✓
COV-01	Unlocked Compiler Version	Language Specific	Informational	ⓘ
COV-02	Function Mutability Optimization	Gas Optimization	Informational	ⓘ
COV-03	`revert` to `require`	Coding Style	Informational	ⓘ

COV-04	Conditional Optimization	Gas Optimization	Informational	ⓘ
IOI-01	Function Mutability Specifier	Gas Optimization	Informational	✓
IOV-01	Function Mutability Specifier	Gas Optimization	Informational	ⓘ
ARP-01	Unlocked Compiler Version	Language Specific	Informational	✓
ARP-02	Redundant Variable Declaration	Gas Optimization	Informational	✓
CSR-01	Unlocked Compiler Version	Language Specific	Informational	✓
CSR-02	Utilization of `keccak256` with `abi.encodePacked`	Logical Issue	Informational	✓
CTR-01	Unlocked Compiler Version	Language Specific	Informational	✓
CTR-02	Utilization of `keccak256` with `abi.encodePacked`	Logical Issue	Informational	✓
DSR-01	Unlocked Compiler Version	Language Specific	Informational	✓
DSR-02	Utilization of `keccak256` with `abi.encodePacked`	Logical Issue	Informational	✓
DSR-03	Loop Exhaustion	Logical Issue	Medium	✓
OIR-01	Unlocked Compiler Version	Language Specific	Informational	✓
OIR-02	Utilization of `keccak256` with `abi.encodePacked`	Logical Issue	Informational	✓
ORY-01	Unlocked Compiler Version	Language Specific	Informational	✓
ORY-02	Utilization of `keccak256` with `abi.encodePacked`	Logical Issue	Informational	✓
TBR-01	Unlocked Compiler Version	Language Specific	Informational	✓
TBR-02	Proxy-over-Implementation Creation	Gas Optimization	Informational	ⓘ
TBR-03	Mathematical Optimization	Arithmetic	Informational	✓
TBR-04	Optimization via `assembly`	Gas Optimization	Informational	ⓘ
TBR-05	Invalid Paradigm	Coding Style	Informational	ⓘ
VAU-01	Inconsistent Comment with Variable Name	Inconsistency	Minor	✓
VAU-02	Mutability Specifiers Missing	Gas Optimization	Informational	ⓘ
VAU-03	Inefficient `storage` layout	Gas Optimization	Informational	✓

VAU-04	Unnecessary explicit usage of `modifier`	Coding Style	Informational	✓
VAU-05	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	ⓘ
VAU-06	Ineffectual Function Call	Language Specific	Minor	✓
VAU-07	Ineffectual `require` Statement	Control Flow	Minor	✓
VAU-08	Substitution of `if` block with `require` Statement	Language Specific	Informational	✓
VAU-09	Event is Fired after State Changes	Language Specific	Informational	✓
VAU-10	Substitution of `if` Block with `require` Statement	Language Specific	Informational	✓
VAU-11	Ineffectual `require` Statement	Coding Style	Minor	✓
VAU-12	Incorrect Comparison of `uint256` with `less-than-zero`	Logical Issue	Informational	✓
VAU-13	Substitution of `if` Block with `require` Statement	Language Specific	Informational	✓
VAU-14	Inefficient Local Variable	Gas Optimization	Informational	✓
VAU-15	Inefficient Local Variable	Gas Optimization	Informational	✓
VAU-16	Duplicate Code can be Extracted to a `private` Function	Gas Optimization	Informational	ⓘ
VAU-17	Explicitly returning a local variable	Gas Optimization	Informational	✓
VAU-18	Incorrect Grammar	Language Specific	Informational	✓
VAU-19	Unnecessary usage of round brackets	Coding Style	Informational	✓
VAU-20	Incorrect order of functions	Language Specific	Informational	ⓘ
VAU-21	Usage of `uint` alias instead of `uint256`	Language Specific	Informational	✓
VAU-22	Generic `reason` Strings in `require` Statements	Language Specific	Informational	ⓘ
VAU-23	Variable type can be changed from `address payable` to `address`	Language Specific	Informational	✓

DSN-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	DerivativeSpecification.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

No alleviations.

DSN-02: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	DerivativeSpecification.sol L14-L31, L33-L82

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

No alleviations.

DSN-03: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	DerivativeSpecification.sol L18, L19, L21, L22, L24, L25, L27, L31, L99, L104, L105, L106, L107, L108, L109, L110

Description:

The linked variables are assigned to only once, either during their contract-level declaration or during the `constructor`'s execution.

Recommendation:

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation:

No alleviations.

FLR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	FeeLogger.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

No alleviations.

FLR-02: Empty Function

Type	Severity	Location
Dead Code	Informational	FeeLogger.sol L8-L10

Description:

The linked function is empty and contains a comment that alludes it is meant simply for retaining the `block.timestamp` of the transaction.

Recommendation:

We advise that it is instead replaced by an `event` which is easier to extract using off-chain processes.

Alleviation:

No alleviations.

FLP-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	FeeLoggerProxy.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

No alleviations.

FLP-02: Literal over `new` Instantiation

Type	Severity	Location
Coding Style	Informational	FeeLoggerProxy.sol L8

Description:

The `AdminUpgradeabilityProxy` constructor is called with the third argument being `new bytes(0)`.

Recommendation:

Instead of an instantiation, a literal can be passed via `bytes("")` or even `""`.

Alleviation:

No alleviations.

VBR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	VaultBuilder.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

VBR-02: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	VaultBuilder.sol L15, L16

Description:

The linked function is declared as `public`, contains array function arguments and is not invoked in any of the contract's contained within the project's scope.

Recommendation:

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation:

No alleviations.

VBR-03: Proxy-over-Implementation Creation

Type	Severity	Location
Gas Optimization	Informational	VaultBuilder.sol L23-L36

Description:

The linked code segment creates a new `Vault` contract on each execution, incurring a significant gas cost to the sender and resulting in duplicate code being deployed numerous times.

Recommendation:

Instead of a direct implementation deployment, the system could instead deploy new vaults using a proxy system that would reduce the gas cost significantly.

Alleviation:

No alleviations.

VFY-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	VaultFactory.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

VFY-02: High-Level of Centralization

Type	Severity	Location
Logical Issue	Informational	VaultFactory.sol L124-L182

Description:

The `owner` of the contract has administrative powers to freeze and unfreeze a vault at will as well as alter the parameters of the protocol for newly issued vaults.

Recommendation:

While this is an expected functionality of the protocol, we advise that this is properly relayed to the users i.e. the documentation explicitly mentions the capability of freezing vaults at will.

Alleviation:

Documentation was updated as alleviation at `https://docs.compli.fi/protocol/governance`.

VFP-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	VaultFactoryProxy.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

No alleviations.

VFP-02: Literal over `new` Instantiation

Type	Severity	Location
Coding Style	Informational	VaultFactoryProxy.sol L8

Description:

The `AdminUpgradeabilityProxy` constructor is called with the third argument being `new bytes(0)`.

Recommendation:

Instead of an instantiation, a literal can be passed via `bytes("")` or even `""`.

Alleviation:

No alleviations.

CSP-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	CollateralSplitParent.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

CSP-02: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	CollateralSplitParent.sol L17, L18, L21, L22

Description:

The linked function is declared as `public`, contains array function arguments and is not invoked in any of the contract's contained within the project's scope.

Recommendation:

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation:

No alleviations.

CSP-03: Unsafe Multiplication

Type	Severity	Location
Mathematical Operations	Minor	CollateralSplitParent.sol L52

Description:

The multiplication conducted between the price difference (`_u_T - _u_0`) and the `FRACTION_MULTIPLIER` can cause overflows / underflows.

Recommendation:

A `SafeMath` -like implementation should be used instead whereby the multiplication is conducted safely depending on the sign of the resulting price difference.

Alleviation:

Alleviations were applied as advised.

COS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	CallOptionSplit.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

COS-02: Function Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	CallOptionSplit.sol L9-L11

Description:

The linked function is defined as `view` in both its `interface` as well as its implementation.

Recommendation:

The state mutability can instead be changed to `pure` as it does not rely on any block or contract level variables.

Alleviation:

Alleviations were applied as advised.

COS-03: `if` Block Optimization

Type	Severity	Location
Gas Optimization	Informational	CallOptionSplit.sol L14-L20

Description:

The linked `if` blocks can be optimized as they cover opposite cases of the `_normalizedValue` variable.

Recommendation:

We advise that the second `if` block is instead moved as an `else` clause of the first `if` block.

Alleviation:

Alleviations were applied as advised.

COS-04: Redundant Statements

Type	Severity	Location
Dead Code	Informational	CallOptionSplit.sol L21

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Alleviations were applied as advised.

COS-05: Invalid Formula in Comment

Type	Severity	Location
Mathematical Operations	Medium	CallOptionSplit.sol L19

Description:

The comment states that the calculation should conduct `U / (X + U)`, where `X` is the value of 1 with decimals i.e. `X = 10^12`, however it conducts `U * X / ((X ^ 2) + U)` as the `U` of the denominator is not properly multiplied by `X` to offset the decimals.

Recommendation:

We advise that the `U` part of the denominator, here `_normalizedValue`, is properly offset by the decimals via multiplication with `FRACTION_MULTIPLIER` or that the comment formula is adjusted to reflect the intended functionality.

Alleviation:

As an alleviation, the confusing comments were removed.

ICS-01: Function Mutability Specifier

Type	Severity	Location
Gas Optimization	Informational	ICollateralSplit.sol L19

Description:

The function mutability specifier is set to `view`, however all implementations do not rely on state.

Recommendation:

We advise that it is instead changed to `pure` along with all of its implementations.

Alleviation:

Alleviations were applied as advised.

ICS-02: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	ICollateralSplit.sol L29-L30, L33-L34

Description:

The linked function is declared as `public`, contains array function arguments and is not invoked in any of the contract's contained within the project's scope.

Recommendation:

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation:

No alleviations.

IST-01: `if` Block Optimizations

Type	Severity	Location
Gas Optimization	Informational	InsurSplit.sol L13-L23

Description:

The linked `if` blocks cover all cases of the `_normalizedValue` variable independently.

Recommendation:

They can instead be joined to a single `if-else-if` chain that ends with an `else` clause as the last linked `if` block.

Alleviation:

Alleviations were applied as advised.

IST-02: Redundant Statements

Type	Severity	Location
Dead Code	Informational	InsurSplit.sol L24

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Alleviations were applied as advised.

IST-03: Incorrect Formula

Type	Severity	Location
Mathematical Operations	Minor	InsurSplit.sol L22

Description:

The formula in the comments does not properly represent the formula carried out in the linked statement, as the linked statement's numerator if we set `x = 1 = 10^12` is `x ^ 2` instead of just `x` which the equation implies. The denominator is incorrect as well.

Recommendation:

We advise that the formula is properly evaluated and depicted properly either in comments or in implementation.

Alleviation:

As an alleviation, the confusing comments were removed.

IST-04: Function Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	InsurSplit.sol L8-L10

Description:

The linked function is defined as `view` in both its `interface` as well as its implementation.

Recommendation:

The state mutability can instead be changed to `pure` as it does not rely on any block or contract level variables.

Alleviation:

Alleviations were applied as advised.

SST-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	StableSplit.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

SST-02: Function Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	StableSplit.sol L8-L10

Description:

The linked function is defined as `view` in both its `interface` as well as its implementation.

Recommendation:

The state mutability can instead be changed to `pure` as it does not rely on any block or contract level variables.

Alleviation:

Alleviations were applied as advised.

SST-03: `if` Block Optimization

Type	Severity	Location
Gas Optimization	Informational	StableSplit.sol L13-L19

Description:

The linked `if` blocks can be optimized as they cover opposite cases of the `_normalizedValue` variable.

Recommendation:

We advise that the second `if` block is instead moved as an `else` clause of the first `if` block.

Alleviation:

Alleviations were applied as advised.

SST-04: Redundant Statements

Type	Severity	Location
Dead Code	Informational	StableSplit.sol L20

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Alleviations were applied as advised.

SST-05: Incorrect Formula

Type	Severity	Location
Mathematical Operations	Minor	StableSplit.sol L18

Description:

The formula in the comments does not properly represent the formula carried out in the linked statement, as the linked statement's numerator if we set `x = 1 = 10^12` is `x ^ 2` instead of just `x` which the equation implies. The denominator is incorrect as well.

Recommendation:

We advise that the formula is properly evaluated and depicted properly either in comments or in implementation.

Alleviation:

As an alleviation the confusing comments were removed.

X1S-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	x1Split.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

X1S-02: Function Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	x1Split.sol L8-L10

Description:

The linked function is defined as `view` in both its `interface` as well as its implementation.

Recommendation:

The state mutability can instead be changed to `pure` as it does not rely on any block or contract level variables.

Alleviation:

Alleviations were applied as advised.

X5S-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	x5Split.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

X5S-02: Function Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	x5Split.sol L8-L10

Description:

The linked function is defined as `view` in both its `interface` as well as its implementation.

Recommendation:

The state mutability can instead be changed to `pure` as it does not rely on any block or contract level variables.

Alleviation:

Alleviations were applied as advised.

X5S-03: `if` Block Optimizations

Type	Severity	Location
Gas Optimization	Informational	x5Split.sol L13-L23

Description:

The linked `if` blocks cover all cases of the `_normalizedValue` variable independently.

Recommendation:

They can instead be joined to a single `if-else-if` chain that ends with an `else` clause as the last linked `if` block.

Alleviation:

Alleviations were applied as advised.

X5S-04: Redundant Statements

Type	Severity	Location
Dead Code	Informational	x5Split.sol L24

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Alleviations were applied as advised.

X5S-05: Invalid Formula

Type	Severity	Location
Mathematical Operations	Minor	x5Split.sol L22

Description:

The formula defined in the comments states `(X + 5 * U) / 2`, `X` representing the value of `1` with decimals i.e. `10^12`, whereas the calculation conducts `(X^2 + U * 5) / (2 * X)`.

Recommendation:

We advise that either the statements or the formula are updated accordingly.

Alleviation:

As an alleviation, the confusing comments were removed.

COI-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ChainlinkOracleIterator.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

COI-02: Function Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	ChainlinkOracleIterator.sol L23-L25

Description:

The linked function is defined as `view` in both its `interface` as well as its implementation.

Recommendation:

The state mutability can instead be changed to `pure` as it does not rely on any block or contract level variables.

Alleviation:

Alleviations were applied as advised.

COI-03: Loop Exhaustion

Type	Severity	Location
Logical Issue	Medium	ChainlinkOracleIterator.sol L42-L71

Description:

The linked `for` loop is quite expensive in terms of gas cost which can lead to the function being unable to execute in case `phaseId` is high enough.

Recommendation:

We advise that the methodology involved here is revised, i.e. loop should potentially `break` under certain circumstances or be considered completely unnecessary.

Alleviation:

As an alleviation, the code was revised and the `for-loop` was removed.

COI-04: Conditional Optimization

Type	Severity	Location
Gas Optimization	Informational	ChainlinkOracleIterator.sol L51

Description:

The former part of the conditional is redundant as `_timestamp` is guaranteed to be in the positive range due to L28.

Recommendation:

We advise that the former part of the `&&` conditional is omitted optimizing gas cost.

Alleviation:

The concerned code was removed and hence this exhibit is no longer applicable.

COI-05: `revert` to `require`

Type	Severity	Location
Coding Style	Informational	ChainlinkOracleIterator.sol L60-L62

Description:

The linked `if` block causes a `revert` operation in case its condition is evaluated to `true`.

Recommendation:

We advise that a `require` check is imposed instead which is equivalent and more legible as `revert` may be deprecated in the future.

Alleviation:

Alleviations were applied as advised.

COI-06: Conditional Optimization

Type	Severity	Location
Gas Optimization	Informational	ChainlinkOracleIterator.sol L63

Description:

The former part of the second conditional in the OR clause (`||`) can be optimized as `timestampNext > 0` will always evaluate to `true` .

Recommendation:

We advise that `timestampNext > 0` is removed from the AND clause (`&&`) of the OR clause (`||`).

Alleviation:

Alleviations were applied as advised.

COI-07: Redundant `SafeMath` Utilization

Type	Severity	Location
Gas Optimization	Informational	ChainlinkOracleIterator.sol L64

Description:

The linked subtraction will never underflow due to the conditional of L59.

Recommendation:

As such, we advise that its raw form is utilized instead.

Alleviation:

The concerned code was removed and hence this exhibit is no longer applicable.

COI-08: Statement Relocation

Type	Severity	Location
Coding Style	Informational	ChainlinkOracleIterator.sol L90

Description:

The linked `return` statement will only be executed in the `catch` clause of the `try-catch` structure.

Recommendation:

As such, we advise that it is moved there to increase code legibility.

Alleviation:

The concerned code was removed and hence this exhibit is no longer applicable.

COV-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ChainlinkOracleIteratorV1.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Client response was that this contract is subject to removal and hence the exhibit is no longer applicable.

COV-02: Function Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	ChainlinkOracleIteratorV1.sol L15-L17

Description:

The linked function is defined as `view` in both its `interface` as well as its implementation.

Recommendation:

The state mutability can instead be changed to `pure` as it does not rely on any block or contract level variables.

Alleviation:

Client response was that this contract is subject to removal and hence the exhibit is no longer applicable.

COV-03: `revert` to `require`

Type	Severity	Location
Coding Style	Informational	ChainlinkOracleIteratorV1.sol L28, L32

Description:

The linked condition, if evaluating to `false`, will cause a `revert` statement to execute.

Recommendation:

We advise that the structure is instead changed to a `require` as it is more legible and `revert` may be deprecated in the future.

Alleviation:

Client response was that this contract is subject to removal and hence the exhibit is no longer applicable.

COV-04: Conditional Optimization

Type	Severity	Location
Gas Optimization	Informational	ChainlinkOracleIteratorV1.sol L44

Description:

The latter part of the linked conditional evaluates `oracle.getTimestamp(roundId)` which is already evaluated in L41 and stored to the `roundTimestamp` variable.

Recommendation:

The `roundTimestamp` variable can be utilized instead substituting the external call.

Alleviation:

Client response was that this contract is subject to removal and hence the exhibit is no longer applicable.

IOI-01: Function Mutability Specifier

Type	Severity	Location
Gas Optimization	Informational	IOracleIterator.sol L14

Description:

The function mutability specifier is set to `view`, however all implementations do not rely on state.

Recommendation:

We advise that it is instead changed to `pure` along with all of its implementations.

Alleviation:

Alleviations were applied as advised.

IOV-01: Function Mutability Specifier

Type	Severity	Location
Gas Optimization	Informational	IOracleIteratorV1.sol L14

Description:

The function mutability specifier is set to `view`, however all implementations do not rely on state.

Recommendation:

We advise that it is instead changed to `pure` along with all of its implementations.

Alleviation:

Client response was that this contract is subject to removal and hence the exhibit is no longer applicable.

ARP-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	AddressRegistryParent.sol L2

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

ARP-02: Redundant Variable Declaration

Type	Severity	Location
Gas Optimization	Informational	AddressRegistryParent.sol L8

Description:

The variable `_keys` is utilized in the `set` function of the `AddressRegistryParent` contract, however it is only used by the `DerivativeSpecificationRegistry` contract.

Recommendation:

As a result, it can be omitted from the parent contract and moved to the child contract as it is only used there and causes redundant gas costs for the other contracts.

Alleviation:

Alleviations were applied as advised.

CSR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	CollateralSplitRegistry.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

CSR-02: Utilization of `keccak256` with `abi.encodePacked`

Type	Severity	Location
Logical Issue	Informational	CollateralSplitRegistry.sol L12

Description:

The function of `abi.encodePacked` is only sensible when the desire is to tight-pack the input variables to the closest space possible. This tight-packing mechanism does not affect full-slot variables (256-bit / 32-byte) or single variables and additionally can lead to hash-collissions in case the tight-packing mechanism packs variables in such a way that their binary representations overlap.

Recommendation:

We advise that `abi.encode` is utilized instead to prevent redundant gas costs and hash collissions.

Alleviation:

Client decided not to apply this alleviation as only a single string or address is packed for hashing which is collision-safe.

CTR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	CollateralTokenRegistry.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

CTR-02: Utilization of `keccak256` with `abi.encodePacked`

Type	Severity	Location
Logical Issue	Informational	CollateralTokenRegistry.sol L11

Description:

The function of `abi.encodePacked` is only sensible when the desire is to tight-pack the input variables to the closest space possible. This tight-packing mechanism does not affect full-slot variables (256-bit / 32-byte) or single variables and additionally can lead to hash-collissions in case the tight-packing mechanism packs variables in such a way that their binary representations overlap.

Recommendation:

We advise that `abi.encode` is utilized instead to prevent redundant gas costs and hash collissions.

Alleviation:

Client decided not to apply this alleviation as only a single string or address is packed for hashing which is collision-safe.

DSR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	DerivativeSpecificationRegistry.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

DSR-02: Utilization of `keccak256` with `abi.encodePacked`

Type	Severity	Location
Logical Issue	Informational	DerivativeSpecificationRegistry.sol L14, L19-L20

Description:

The function of `abi.encodePacked` is only sensible when the desire is to tight-pack the input variables to the closest space possible. This tight-packing mechanism does not affect full-slot variables (256-bit / 32-byte) or single variables and additionally can lead to hash-collissions in case the tight-packing mechanism packs variables in such a way that their binary representations overlap.

Recommendation:

We advise that `abi.encode` is utilized instead to prevent redundant gas costs and hash collissions.

Alleviation:

Alleviations were applied as advised.

DSR-03: Loop Exhaustion

Type	Severity	Location
Logical Issue	Medium	DerivativeSpecificationRegistry.sol L16-L31

Description:

The linked `for` loop can lead to an exhaustion of the available `gas`, forever locking the function's execution as the `_keys` array is incremental.

Recommendation:

We advise that a different mechanism is utilized for checking whether the specification already exists, such as a `mapping` for the resulting hash of `abi.encode`-ing the variables checked in the `if` block of L19-L27.

Alleviation:

Alleviations were applied as advised.

OIR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	OracleIteratorRegistry.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

OIR-02: Utilization of `keccak256` with `abi.encodePacked`

Type	Severity	Location
Logical Issue	Informational	OracleIteratorRegistry.sol L12

Description:

The function of `abi.encodePacked` is only sensible when the desire is to tight-pack the input variables to the closest space possible. This tight-packing mechanism does not affect full-slot variables (256-bit / 32-byte) or single variables and additionally can lead to hash-collissions in case the tight-packing mechanism packs variables in such a way that their binary representations overlap.

Recommendation:

We advise that `abi.encode` is utilized instead to prevent redundant gas costs and hash collissions.

Alleviation:

Client decided not to apply this alleviation as only a single string or address is packed for hashing which is collision-safe.

ORY-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	OracleRegistry.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

ORY-02: Utilization of `keccak256` with `abi.encodePacked`

Type	Severity	Location
Logical Issue	Informational	OracleRegistry.sol L11

Description:

The function of `abi.encodePacked` is only sensible when the desire is to tight-pack the input variables to the closest space possible. This tight-packing mechanism does not affect full-slot variables (256-bit / 32-byte) or single variables and additionally can lead to hash-collissions in case the tight-packing mechanism packs variables in such a way that their binary representations overlap.

Recommendation:

We advise that `abi.encode` is utilized instead to prevent redundant gas costs and hash collissions.

Alleviation:

Client decided not to apply this alleviation as only a single string or address is packed for hashing which is collision-safe.

TBR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	TokenBuilder.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

Alleviations were applied as advised.

TBR-02: Proxy-over-Implementation Creation

Type	Severity	Location
Gas Optimization	Informational	TokenBuilder.sol L38-L48

Description:

The linked code segment creates a new `ERC20PresetMinter` contract on each execution, incurring a significant gas cost to the sender and resulting in duplicate code being deployed numerous times.

Recommendation:

Instead of a direct implementation deployment, the system could instead deploy new vaults using a proxy system that would reduce the gas cost significantly.

Alleviation:

No alleviations.

TBR-03: Mathematical Optimization

Type	Severity	Location
Mathematical Operations	Informational	TokenBuilder.sol L56

Description:

The linked code block conducts a modulo (`%`) operation on the input in an excessive way via divisions and multiplications.

Recommendation:

A modulo operation could be utilized directly which is more optimal gas-wise.

Alleviation:

Alleviations were applied as advised.

TBR-04: Optimization via `assembly`

Type	Severity	Location
Gas Optimization	Informational	TokenBuilder.sol L67-L79

Description:

The linked code block iterates through all digits of the input number twice, once to identify the length and secondly to set each `byte` of the `bytes` array to its corresponding value.

Recommendation:

We advise that an `assembly` block is introduced that affects the `length` of the `bytes` array manually, thus requiring only one iteration of all digits of the input number.

Alleviation:

No alleviations.

TBR-05: Invalid Paradigm

Type	Severity	Location
Coding Style	Informational	TokenBuilder.sol L119

Description:

In Solidity, the `NaN` type does not exist and as such the return of this function is relatively ambiguous.

Recommendation:

We advise that a `require` check is imposed instead that ensures `_month` is within the `[1,12]` range.

Alleviation:

No alleviations.

VAU-01: Inconsistent Comment with Variable Name

Type	Severity	Location
Inconsistency	Minor	Vault.sol L56

Description:

The comment on the aforementioned line incorrectly refers to `complement conversion` as `primary conversion`.

```
/// @notice primary token conversion rate multiplied by 10 ^ 12
```

Recommendation:

We recommend to rectify the comment correctly identify `complement conversion`.

```
/// @notice complement token conversion rate multiplied by 10 ^ 12
```

Alleviation:

Alleviations were applied as advised.

VAU-02: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	Vault.sol L41, L43, L45, L47, L60, L62, L68, L70, L75, L77, L78, L81

Description:

The linked variables are assigned to only once, either during their contract-level declaration or during the `constructor`'s execution.

Recommendation:

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation:

No alleviations.

VAU-03: Inefficient `storage` layout

Type	Severity	Location
Gas Optimization	Informational	Vault.sol L65, L81

Description:

The `storage` variables `state` and `feeWallet` on the aforementioned lines occupy two `storage` slots. The variables can be tight packed to utilize only a single `storage` slot by placing them next to each other which save gas cost associated with an additional `storage` slot.

Recommendation:

We recommend that the `storage` variables on the aforementioned lines be placed next to each other to tight pack them and save gas cost associated with a `storage` slot.

```
// @notice protocol's fee receiving wallet
address public feeWallet;
// @notice current state of the vault
State public state;
```

Alleviation:

Alleviations were applied as advised.

VAU-04: Unnecessary explicit usage of `modifier`

Type	Severity	Location
Coding Style	Informational	Vault.sol L101

Description:

The explicit usage of modifier `Ownable()` to call constructor of the `Ownable` contract is unnecessary as the constructor does not expect arguments is called implicitly nevertheless.

```
constructor() Ownable() public {...}
```

Recommendation:

We recommend to remove the explicit usage of `Ownable()` modifier as it is redundant and is already implicitly called.

```
constructor() public {...}
```

Alleviation:

Alleviations were applied as advised.

VAU-05: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	Vault.sol L102, L116, L120, L207, L237, L327, L254, L273, L292, L297

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

No alleviations.

VAU-06: Ineffectual Function Call

Type	Severity	Location
Language Specific	Minor	Vault.sol L133

Description:

The function call `changeState(State.Created)` on the aforementioned line is ineffectual as the value of state variable `state` is already set to default value of `State.Created` and hence function call is redundant.

Recommendation:

We recommend to remove the function call on the aforementioned line.

Alleviation:

Alleviations were applied as advised.

VAU-07: Ineffectual `require` Statement

Type	Severity	Location
Control Flow	Minor	Vault.sol L143

Description:

The `require` statement on the aforementioned line is ineffectual. The `L142` ensures through `require` statement that `liveTime > block.timestamp` and as `settleTime` is either greater or equal to `liveTime`, so the predicate `settleTime > block.timestamp` will never be `false`.

```
require(liveTime > block.timestamp, "Live time");
require(settleTime > block.timestamp, "Settle time");
```

Recommendation:

We recommend to remove the `require` statement on the aforementioned line as it is ineffectual.

Alleviation:

Alleviations were applied as advised.

VAU-08: Substitution of `if` block with `require` Statement

Type	Severity	Location
Language Specific	Informational	Vault.sol L170-L172

Description:

The `if` block on the aforementioned lines can be replaced with a `require` statement as it is an idiomatic way of reverting transactions in Solidity.

```
if(state != State.Minting) {
  revert('Incorrect state');
}
```

Recommendation:

We recommend to use a `require` statement instead of `if` block on the aforementioned lines to increase the legibility of the code.

```
require(
  state == State.Minting,
  "Incorrect state"
);
```

Alleviation:

Alleviations were applied as advised.

VAU-09: Event is Fired after State Changes

Type	Severity	Location
Language Specific	Informational	Vault.sol L180-L181

Description:

The event `StateChanged` is fired on `L180` before the state change on `L181`. An idiomatic way in Solidity is to fire events after the state changes.

```
function changeState(State _newState) internal {
  emit StateChanged(state, _newState);
  state = _newState;
}
```

Recommendation:

We advise to fire the event `StateChanged` after the state changes are applied.

```
function changeState(State _newState) internal {
        state = _newState;
        emit StateChanged(state, _newState);
}
```

Alleviation:

Alleviations were applied as advised.

VAU-10: Substitution of `if` Block with `require` Statement

Type	Severity	Location
Language Specific	Informational	Vault.sol L191-L193

Description:

The `if` block on the aforementioned lines can be replaced with a `require` statement as it is an idiomatic way of reverting transaction in Solidity.

```
if(state != State.Live) {
  revert('Incorrect state');
}
```

Recommendation:

We recommend to replace the `if` block with a `require` statement to increase the legibility of the code.

```
require(
  state == State.Live,
  'Incorrect state'
);
```

Alleviation:

Alleviations were applied as advised.

VAU-11: Ineffectual `require` Statement

Type	Severity	Location
Coding Style	Minor	Vault.sol L194-L195

Description:

The `require` statement on `L195` will never evaluate to `false` as the `require` statement on `L194` ensures that `block.timestamp >= settleTime` and the subsequent `require` statement will never evaluate to `false` when that is the case.

```
require(block.timestamp >= (settleTime), "Incorrect time");
require(block.timestamp >= (settleTime + settlementDelay), "Delayed settlement");
```

Recommendation:

We recommend to remove the `require` statement on `L194` as the `require` statement on `L195` covers the case of `require` statement on `L194` when `settlementDelay` is `0`.

```
require(block.timestamp >= (settleTime + settlementDelay), "Delayed settlement");
```

Alleviation:

Alleviations were applied as advised.

VAU-12: Incorrect Comparison of `uint256` with `less-than-zero`

Type	Severity	Location
Logical Issue	Informational	Vault.sol L216-L224

Description:

The predicate of `if` block on `L220-L222` compares a `uint256` value `_split` with `less-than-zero`, which will always evaluate to `false` as a `uint256`, being an unsigned integer, can never have a value less than zero.

```
if(_split < 0) {
    return 0;
}
```

Additionally, the explicit `0` literal on `L221` does not need to be returned as the `uint256` return type will have a default value of `0`.

Recommendation:

We advise to correct the implementation of function `range`. If the parameter `_split` is incorrectly typed as `uint256` instead of `int256` then it be rectified. The default value `0` of `uint256/int256` does not need to be returned as literal on `L221`.

The function `range` can be rectified in a following way consdering that `_split` was intended to be type `int256` instead of `uint256`.

```
function range(int256 _split) public pure returns(int256) {
    if(_split > FRACTION_MULTIPLIER) {
        return FRACTION_MULTIPLIER;
    }
    if(_split < 0) {
        return;
    }
    return _split;
}
```

Alleviation:

As alleviation, the parameter `_split` 's type was changed to `uint256` .

VAU-13: Substitution of `if` Block with `require` Statement

Type	Severity	Location
Language Specific	Informational	Vault.sol L233-L235

Description:

The `if` block on the aforementioned lines can be replaced with a `require` statement as it is an idiomatic way of reverting transaction in Solidity.

```
if(state != State.Minting){
    revert('Minting period is over');
}
```

Recommendation:

We recommend to replace the `if` block with `require` statement to increase the legibility of the code.

```
require(
    state == State.Minting,
    'Minting period is over'
);
```

Alleviation:

Alleviations were applied as advised.

VAU-14: Inefficient Local Variable

Type	Severity	Location
Gas Optimization	Informational	Vault.sol L341, L343

Description:

The local variable declaration `token` on `L341` is inefficient as the local variable is only used once in the code.

```
EIP20NonStandardInterface token = EIP20NonStandardInterface(address(collateralToken));
```

```
token.transferFrom(from, address(this), amount);
```

Recommendation:

We recommend to directly utilize the intialization part of the variable declaration on `L343` to save gas cost associated with additional local variable.

```
EIP20NonStandardInterface(address(collateralToken)).transferFrom(from, address(this),
amount);
```

Alleviation:

Alleviations were applied as advised.

VAU-15: Inefficient Local Variable

Type	Severity	Location
Gas Optimization	Informational	Vault.sol L374-L375

Description:

The local variable declaration on `L374` is inefficient as the local variable is only utilized once in the code.

```
EIP20NonStandardInterface token = EIP20NonStandardInterface(address(collateralToken));
token.transfer(to, amount);
```

Recommendation:

We recommend to remove the local variable declaration and instead directly utilize the intialization part in place of local variable where it is used to save gas associated with an additional local variable.

```
EIP20NonStandardInterface(address(collateralToken)).transfer(to, amount);
```

Alleviation:

Alleviations were applied as advised.

VAU-16: Duplicate Code can be Extracted to a `private` Function

Type	Severity	Location
Gas Optimization	Informational	Vault.sol L346-L358, L378-L390

Description:

The code block on the aforementioned lines is duplicate accross `doTransferIn` and `doTransferOut` functions. The code block can be extracted to `private` function utilized instead. This will reduce the `bytecode` footprint of the contract resulting in reduced gas cost when deploying the contract and furthermore, it increases the legibility of the code by avoiding the code duplication.

Recommendation:

We recommend to extract the duplicate code to a `private` and that be utilized instead.

```
function successfullyTransferred() private returns(bool success) {
  assembly {
            switch returndatasize()
            case 0 {                      // This is a non-standard ERC-20
                success := not(0)          // set success to true
            }
            case 32 {                     // This is a complaint ERC-20
                returndatacopy(0, 0, 32)
                success := mload(0)        // Set `success = returndata` of external call
            }
            default {                     // This is an excessively non-compliant ERC-20,
revert.
                revert(0, 0)
            }
        }
}
```

```
function doTransferIn(address from, uint amount) internal returns (uint) {
        EIP20NonStandardInterface token =
EIP20NonStandardInterface(address(collateralToken));
        uint balanceBefore = collateralToken.balanceOf(address(this));
        token.transferFrom(from, address(this), amount);

        bool success = successfullyTransferred();
        require(success, "TOKEN_TRANSFER_IN_FAILED");

        // Calculate the amount that was *actually* transferred
        uint balanceAfter = collateralToken.balanceOf(address(this));
        require(balanceAfter >= balanceBefore, "TOKEN_TRANSFER_IN_OVERFLOW");
        return balanceAfter - balanceBefore;   // underflow already checked above, just
subtract
}
```

```
function doTransferOut(address payable to, uint amount) internal {
        EIP20NonStandardInterface token =
EIP20NonStandardInterface(address(collateralToken));
        token.transfer(to, amount);

        bool success = successfullyTransferred();
        require(success, "TOKEN_TRANSFER_OUT_FAILED");
}
```

Alleviation:

No alleviations.

VAU-17: Explicitly returning a local variable

Type	Severity	Location
Gas Optimization	Informational	Vault.sol L325, L326, L330

Description:

The function `calcAndTransferFee` returns a local variable from its body which increases the overall gas cost.

```
uint feeAmount = _amount.mul(_fee).div(FRACTION_MULTIPLIER);
```

```
return feeAmount;
```

Recommendation:

Since named return variables can be declared in the signature of the function, consider refactoring to remove the local variable declaration and explicit return statement to reduce the overall cost of gas.

```
function calcAndTransferFee(uint _amount, address payable _beneficiary, uint _fee) internal
returns(uint feeAmount) {
    uint feeAmount = _amount.mul(_fee).div(FRACTION_MULTIPLIER);
    if(feeAmount > 0) {
        doTransferOut(_beneficiary, feeAmount);
    }
}
```

Alleviation:

Alleviations were applied as advised.

VAU-18: Incorrect Grammar

Type	Severity	Location
Language Specific	Informational	Vault.sol L266

Description:

The aforementioned line has incorrect grammar.

```
/// @notice Redeems unequal amounts previously calculated conversions if the vault in
Settled state
```

Recommendation:

We recommend to correct the grammar of the aforementioned comment.

```
/// @notice Redeems unequal amounts previously calculated conversions if the vault is in
Settled state
```

Alleviation:

Alleviations were applied as advised.

VAU-19: Unnecessary usage of round brackets

Type	Severity	Location
Coding Style	Informational	Vault.sol L194, L195

Description:

The aforementioned lines use unnecessary brackets around the `uint256` variable and an expression.

```
require(block.timestamp >= (settleTime), "Incorrect time");
require(block.timestamp >= (settleTime + settlementDelay), "Delayed settlement");
```

Recommendation:

We recommend to remove the unnecessary parenthesis on the aforementioned lines to increase the quality and legibility of the code.

```
require(block.timestamp >= settleTime, "Incorrect time");
require(block.timestamp >= settleTime + settlementDelay, "Delayed settlement");
```

Alleviation:

Alleviations were applied as advised.

VAU-20: Incorrect order of functions

Type	Severity	Location
Language Specific	Informational	Vault.sol L1

Description:

The structure of the codebase does not conform to the official Solidity style guide of `v0.6.x`.

Recommendation:

An indicative excerpt of the style guide is that functions should be grouped according to their visibility and ordered:

```
constructor
receive function (if exists)
fallback function (if exists)
external
public
internal
private
```

Within a grouping, place the `view` and `pure` functions last.

Alleviation:

No alleviations.

VAU-21: Usage of `uint` alias instead of `uint256`

Type	Severity	Location
Language Specific	Informational	Vault.sol L1

Description:

The library is using `uint` to declare 256-bit unsigned integers. Although, `uint` is an alias for `uint256` and both represent the same underlying integer allocation. It is advisable that for clean coding practices the complete form `uint256` should be used instead of the alias `uint`.

Recommendation:

We advise to use `uint256` instead of alias `uint` in all of the occurrences in the contract.

Alleviation:

Alleviations were applied as advised.

VAU-22: Generic `reason` Strings in `require` Statements

Type	Severity	Location
Language Specific	Informational	Vault.sol L102, L107, L110, L114, L116, L120, L124, L127, L130, L142, L143, L173

Description:

The `require` statements on the aforementioned lines have generic `reason` string. Expressive `reason` strings can greatly aid in readability, debugging of the code and additionaly increases the quality of the code.

Recommendation:

We recommend to use more comprehensive `reason` strings in the `require` statements to aid readability and debugging of the code-base.

Alleviation:

No alleviations.

VAU-23: Variable type can be changed from `address payable` to `address`

Type	Severity	Location
Language Specific	Informational	Vault.sol L373

Description:

The function parameter `to` has a type `address payable` yet no `ether` is transferred to the address in the `doTransferOut` function.

```
function doTransferOut(address payable to, uint amount) internal {...}
```

Recommendation:

We recommend to change the variable `to` type from `address payable` to `address` to increase the legibility of the code.

```
function doTransferOut(address to, uint amount) internal {...}
```

Alleviation:

Alleviations were applied as advised.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.