CERTIK

Coti's CVI Implementation

Security Assessment

January 6th, 2021

For :
Coti's CVI Implementation

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Georgios Delkos @ CertiK
georgios.delkos@certik.io

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	Coti's CVI Implementation
Description	The CVI (Cryptocurrency Volatility Index) platform is a decentralized solution to analyzing the market's expectation of future volatility and enabling users to open positions based on these predictions that can be subsequently liquidated or redeemed depending on whether the value remains above the liquidation threshold.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. da94e465d17296a15cf92813dab714e59c0105bf 2. 3405d157708672503cf0af7090e2e5c27cd527ac

Audit Summary

Delivery Date	January 6th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	December 28th, 2020 - January 6th, 2021

Vulnerability Summary

Total Issues	25
Total Critical	0
Total Major	1
Total Medium	3
Total Minor	7
Total Informational	14

Executive Summary

We were tasked with auditing the CVI implementation of Coti in Solidity, representing a full-scale decentralized ecosystem that computes the decentralized volatility index from cryptocurrency option prices and allows analysis of the market's expectation of future value.

The mechanism used by the CVI system is disimilar to traditional stock markets as it relies on a collateralization mechanism. The index's calculation occurs every minute whilst the averaged index value is used for any settlements in the ecosystem.

The system closely integrates with Chainlink to properly report the cryptocurrency volatility index reported by an open-source script provided by Coti. On-chain, a position opening mechanism exists for Ethereum as well as ERC20 tokens which allows anyone to report liquidate-able positions and acquire a reward in case the positions mentioned are indeed liquidate-able.

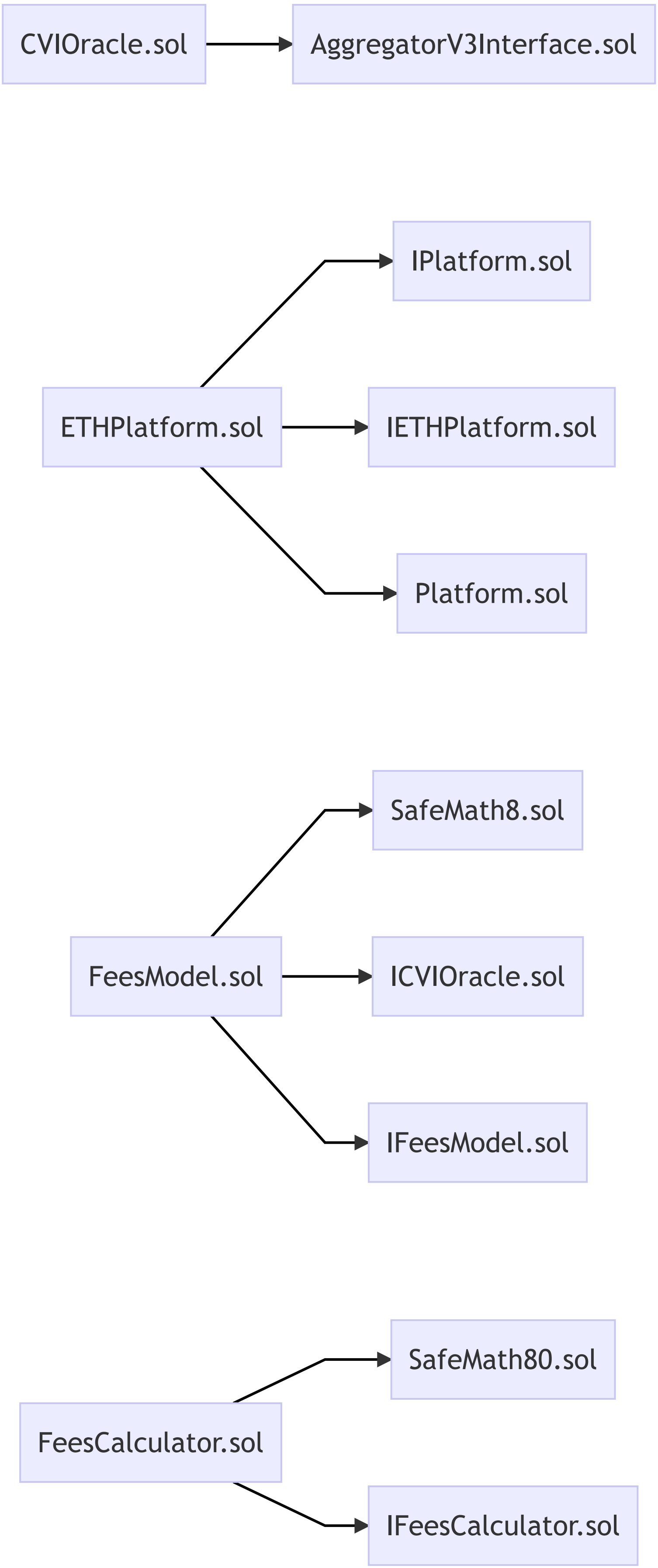
A staking mechanism is also introduced in one of the contracts as well as an intricate fee calculation system that disallows abuses of the system, such as unlimited positions being opened to maliciously prohibit withdrawals.

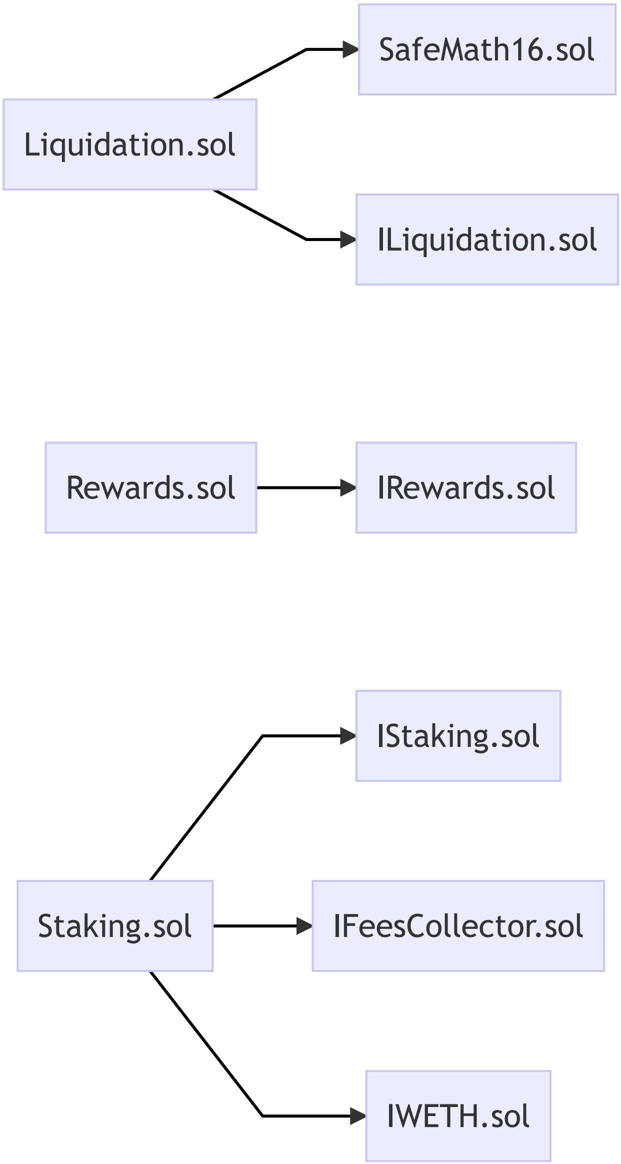
Overall, our analysis pointed some flaws that were promptly fixed by the development team behind CVI as well as certain optimizations most of which were applied on the codebase. We would like to note that we highly advise the test cases are expanded, as we found them to not cover as many edge cases as an extensive test suite should.

Files In Scope

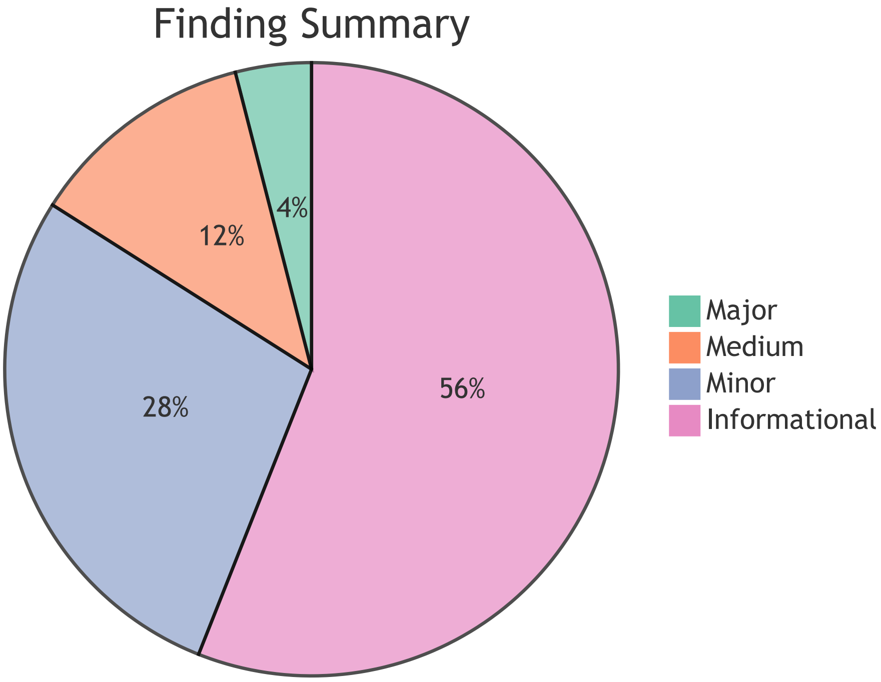
ID	Contract	Location
AVI	AggregatorV3Interface.sol	contracts/interfaces/AggregatorV3Interface.sol
CVI	CVIOracle.sol	contracts/CVIOracle.sol
ETH	ETHPlatform.sol	contracts/ETHPlatform.sol
FML	FeesModel.sol	contracts/FeesModel.sol
FCR	FeesCalculator.sol	contracts/FeesCalculator.sol
GOV	GOVI.sol	contracts/GOVI.sol
IWE	IWETH.sol	contracts/interfaces/IWETH.sol
IRS	IRewards.sol	contracts/interfaces/IRewards.sol
ISG	IStaking.sol	contracts/interfaces/IStaking.sol
IPM	IPlatform.sol	contracts/interfaces/IPlatform.sol
ICV	ICVIOracle.sol	contracts/interfaces/ICVIOracle.sol
IFM	IFeesModel.sol	contracts/interfaces/IFeesModel.sol
IET	IETHPlatform.sol	contracts/interfaces/IETHPlatform.sol
ILN	ILiquidation.sol	contracts/interfaces/ILiquidation.sol
IFC	IFeesCollector.sol	contracts/interfaces/IFeesCollector.sol
CON	IFeesCalculator.sol	contracts/interfaces/IFeesCalculator.sol
LIQ	Liquidation.sol	contracts/Liquidation.sol
PLA	Platform.sol	contracts/Platform.sol
REW	Rewards.sol	contracts/Rewards.sol
STA	Staking.sol	contracts/Staking.sol
SM8	SafeMath8.sol	contracts/utils/SafeMath8.sol
SM6	SafeMath16.sol	contracts/utils/SafeMath16.sol
SM0	SafeMath80.sol	contracts/utils/SafeMath80.sol
SRS	StakingRewards.sol	contracts/synthetix/StakingRewards.sol
WET	WETH9.sol	contracts/external/WETH9.sol

File Dependency Graph (BETA)




Liquidation.sol
SafeMath16.sol
ILiquidation.sol
Rewards.sol
IRewards.sol
Staking.sol
IStaking.sol
IFeesCollector.sol
IWETH.sol

Findings


Finding Summary
4%
12%
28%
56%
Major
Medium
Minor
Informational

ID	Title	Type	Severity	Resolved
CVI-01	Mutability Specifiers Missing	Gas Optimization	Informational	✓
CVI-02	Potentially Improper Chainlink Integration	Logical Issue	Minor	✓
CVI-03	`return` Instead of Assignment	Gas Optimization	Informational	(!)
FCR-01	Redundant `public` Specifier	Gas Optimization	Informational	✓
FCR-02	Confusing Naming Convention	Coding Style	Informational	✓
FCR-03	`TODO` Comments	Coding Style	Informational	✓
FCR-04	Function Visibility Optimization	Gas Optimization	Informational	✓
FCR-05	Data Type Optimization	Logical Issue	Minor	(!)
FCR-06	Codeblock Optimization	Gas Optimization	Informational	✓
FCR-07	Variable Tight-Packing	Gas Optimization	Informational	(!)
FCR-08	Minimum Oracle Heartbeat Period	Control Flow	Minor	(!)
FCR-09	Potential Function Lock	Mathematical Operations	Minor	✓
FCR-10	Function Visibility Optimization	Gas Optimization	Informational	✓
FCR-11	Convoluted Logic	Mathematical Operations	Informational	✓
FML-01	Mutability Specifiers Missing	Gas Optimization	Informational	(!)
FML-02	Invalid Opcode	Logical Issue	Major	✓
FML-03	Unnecesarily Complex Code Block	Gas Optimization	Informational	✓

ID	Title	Type	Severity	Resolved
ETH-01	Codebase Consistency	Inconsistency	Informational	✓
PLA-01	Inconsistent Usage of Naming Convention	Inconsistency	Minor	(!)
PLA-02	Liquidation Award Discrepancy	Logical Issue	Medium	✓
PLA-03	Strict `require` Check	Logical Issue	Minor	✓
PLA-04	Inexistent Access Control of Position Opening	Language Specific	Medium	✓
LIQ-01	Code Duplication	Gas Optimization	Informational	✓
LIQ-02	Inexistent Input Sanitization	Control Flow	Minor	✓
LIQ-03	Reward Regardless of Liquidation	Logical Issue	Medium	✓

CVI-01: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	CVIOracle.sol L17

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:

The team properly introduced the `immutable` specifier to the linked variable.

CVI-02: Potentially Improper Chainlink Integration

Type	Severity	Location
Logical Issue	Minor	CVIOracle.sol L24

Description:

The linked line retrieves the `cviOracleTimestamp` from the Chainlink oracle using the returned `startedAt` variable instead of the `updatedAt` variable.

Recommendation:

We advise that, depending on whether the value of a particular round could potentially be updated, that the `updatedAt` variable is used instead as a more accurate representation of the round data's timestamp.

Alleviation:

A change was introduced that utilizes the latest update timestamp rather than creation timestamp ensuring the system is compatible with oracle values that have been updated.

CVI-03: `return` Instead of Assignment

Type	Severity	Location
Gas Optimization	Informational	CVIOracle.sol L39

Description:

The linked statement is executed within the edge-case `if` clause that surrounds it, resulting in the overall function returning a static value.

Recommendation:

The value can be directly returned here either by being pre-calculated or represented by its `constant` variables being divided in raw format. On an additional note, the utilization of `div` is unnecessary as well as the divisor is always positive due to it being a `constant`.

Alleviation:

The Coti's CVI Implementation development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

FCR-01: Redundant `public` Specifier

Type	Severity	Location
Gas Optimization	Informational	FeesCalculator.sol L22, L23, L25, L26, L27, L29, L31, L32

Description:

The linked variable declarations are `constant` variables that are made publicly available via auto-generated getter functions, significantly increasing the gas cost of deployment as well as the bytecode of the contract.

Recommendation:

We advise that these `public` specifiers are instead omitted as they are non-essential and can easily be retrieved by the confirmed source code on the Ethereum blockchain.

Alleviation:

The redundant specifiers were properly set to `private` within the codebase.

FCR-02: Confusing Naming Convention

Type	Severity	Location
Coding Style	Informational	FeesCalculator.sol L25, L26

Description:

The linked variables are confusingly named so.

Recommendation:

We advise that they are instead renamed to a more sensible name that immediately informs the reader of what they are meant to represent.

Alleviation:

The variables were renamed to properly reflect what they are meant to represent.

FCR-03: `TODO` Comments

Type	Severity	Location
Coding Style	Informational	FeesCalculator.sol L48

Description:

The linked line contains `TODO` comments that request a test of several functions within the contract.

Recommendation:

We advise that the `TODO` comments are either fulfilled or simply removed as production-grade code should not possess such comments.

Alleviation:

The `TODO` comments were removed from the codebase.

FCR-04: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	FeesCalculator.sol L55

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 `public` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation:

The variables and function visibility specifiers were properly matched to ensure `calldata` is utilized wherever sensible.

FCR-05: Data Type Optimization

Type	Severity	Location
Logical Issue	Minor	FeesCalculator.sol L56

Description:

The linked `for` loop conducts an iteration using a `uint8` variable without sanitizing the loop limit.

Recommendation:

Apart from potentially failing execution due to the input variable's `_periods` length being unconstrained, it is actually more expensive to use a `uint8` iterator for the loop rather than a `uint256` iterator due to the EVM operating in 256-bit format.

Alleviation:

The Coti's CVI Implementation development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

FCR-06: Codeblock Optimization

Type	Severity	Location
Gas Optimization	Informational	FeesCalculator.sol L55-L69

Description:

The linked code block can be optimized significantly reducing its gas footprint on both deployment as well as execution.

Recommendation:

Firstly, the function conducts multiple storage writes and reads to the `turbulenceIndicatorPercent` variable instead of reading it off storage once, assigning it to `memory` and utilizing it until the end of the function where it is finalized and assigned to `storage` once again.

Secondly, the `else` clause can be combined with the inner `if` clause to form an `else-if` clause.

Additionally, the utilization of SafeMath's `div` is redundant as a positive literal is used as the divisor.

Lastly, the `if` clause of L57 should also check that the `turbulenceIndicatorPercent` variable is less than `buyingPremiumFeeMaxPercent` to avoid superfluous calculations.

Alleviation:

The codeblock was heavily optimized and a zeroing operation was introduced in the new conditional within the `else` block of the `for` loop that sets the variable `updatedTurbulenceIndicatorPercent` to zero if it is below `turbulenceFeeMinPercentThreshold`.

FCR-07: Variable Tight-Packing

Type	Severity	Location
Gas Optimization	Informational	FeesCalculator.sol L44

Description:

The linked variable declaration is a `uint16` declaration that can be tightly packed with other variables it is simultaneously read with.

Recommendation:

We advise that it is relocated within the declarations of L34-L40 as these variables are read alongside the linked one.

Alleviation:

The Coti's CVI Implementation development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

FCR-08: Minimum Oracle Heartbeat Period

Type	Severity	Location
Control Flow	Minor	FeesCalculator.sol L95-L97

Description:

A minimum oracle heartbeat period is not guaranteed by the contract's code.

Recommendation:

We advise that a `require` check is imposed to ensure that the heartbeat period is within certain sensible bounds and cannot be maliciously altered so.

Alleviation:

The Coti's CVI Implementation development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

FCR-09: Potential Function Lock

Type	Severity	Location
Mathematical Operations	Minor	FeesCalculator.sol L120

Description:

The linked line conducts a SafeMath `sub` operation between `PRECISION_DECIMALS` and `_collateralRatio` whilst the `if` clause that precedes it alllows `PRECISION_DECIMALS` to be lower than `_collateralRatio`.

Recommendation:

We advise that this scenario is evaluated as it would lead to the function being un-invokable and if impossible, that it is sufficiently described by comments or guaranteed by the `if` clause.

Alleviation:

A comment was introduced that explains the `revert` functionality at this point is desired as a sanity check as the subtraction should under all circumstances not underflow.

FCR-10: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	FeesCalculator.sol L136

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 `public` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation:

The data location specifier of the input was properly set to be `calldata` optimizing the function block.

FCR-11: Convoluted Logic

Type	Severity	Location
Mathematical Operations	Informational	FeesCalculator.sol L142-L170

Description:

The linked code segment over-utilizes the SafeMath library, potentially causing it to be non-executable in trivial underflow / overflow scenarios that should otherwise be accounted for via `if` clauses i.e. the multiple `mul` conducted on L168.

Recommendation:

We advise that the complexity is evaluated and potentially simplified to ensure that the function is executable under all edge cases.

Alleviation:

The calculations were broken down into components and SafeMath utilization was toned down, optimizing the code segment and increasing the code's legibility by a substantial amount.

FML-01: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	FeesModel.sol L23, L24

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:

The Coti's CVI Implementation development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

FML-02: Invalid Opcode

Type	Severity	Location
Logical Issue	Major	FeesModel.sol L85

Description:

The linked statement within the `if` clause that guarantees the length of the `cviPeriods` array is `0` will execute an invalid OPCODE, causing the function to remain non-executable and wasting all gas supplied to it.

Recommendation:

We advise that the logic path of this particular `if` clause is evaluated as it can cause significant misbehaviour on the part of the contract.

Alleviation:

The invalid opcode was fixed and is no longer executed in the linked `if` block.

FML-03: Unnecesarily Complex Code Block

Type	Severity	Location
Gas Optimization	Informational	FeesModel.sol L38-L117

Description:

The linked code block retrieves a set of values from the CVI oracle and aggregates them using a jump mechanic to ensure the ensuing `while` loop does not iterate infinitely and thus causing the function impossible to be invoked. The logic utilized to achieve this, however, seems unnecessarily convoluted.

Recommendation:

The `if` clauses, `while` loop and general logic of the function can be significantly optimized if abstractions are applied and the utilization of SafeMath is toned down as it is redundant in many cases.

Alleviation:

The code block was optimized by toning the utilization of `SafeMath` down and increasing the legibility of the codebase.

ETH-01: Codebase Consistency

Type	Severity	Location
Inconsistency	Informational	ETHPlatform.sol L32-L33,

Description:

The linked statements are replicated within the `sendETH` function of the same contract.

Recommendation:

We advise that the `sendETH` function replaces them to optimize the gas cost of the contract's deployment and ensure consistency in the codebase.

Alleviation:

The code was adjusted to utilize the already built-in `sendETH` function.

PLA-01: Inconsistent Usage of Naming Convention

Type	Severity	Location
Inconsistency	Minor	Platform.sol L51, L133, L348

Description:

The linked lines represent the usage of the `revertLockedTransfered` mapping whose naming convention indicates that if it yields `true`, a function should `revert`. This is not the case however as per L348.

Recommendation:

We advise that the variable is either renamed or the linked `require` clause is set to not use the negation of the mapping as it is currently misleading to users and integrators of the application.

Alleviation:

The Coti's CVI Implementation development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PLA-02: Liquidation Award Discrepancy

Type	Severity	Location
Logical Issue	Medium	Platform.sol L93

Description:

The `getLiquidationReward` function within the `liquidation` contract returns a value to be rewarded regardless of whether the position was actually liquidated as an `else` clause exists within its implementation.

Recommendation:

We advise that either an `if` clause is introduced that precedes the addition of the `finderFeeAmount` or that the implementation within `liquidation` is adjusted to properly reflect a liquidation reward.

Alleviation:

The liquidation award mechanism was adjusted to properly ensure an account is liquidated before compounding the reward for liquidation.

PLA-03: Strict `require` Check

Type	Severity	Location
Logical Issue	Minor	Platform.sol L238

Description:

The `require` check ensures that the collateral ratio is broken on a designated withdrawal, however it can firstly lead to complete lockup of withdrawals if the ratio is broken and can also malfunction in case of a token that imposes fees on outgoing transactions, thus breaking the ratio even if the `require` check passes.

Recommendation:

The `require` check should be relocated after the `safeTransfer`'s execution and an optional emergency system should be introduced that permits withdrawals even if the collateral ratio is broken to ensure that funds will be withdrawable under all circumstances.

Alleviation:

An emergency mechanism was introduced that bypasses these checks in a failsafe scenario, ensuring funds can never be locked within the protocol.

PLA-04: Inexistent Access Control of Position Opening

Type	Severity	Location
Language Specific	Medium	Platform.sol L81-L83

Description:

The function `openPosition` does not appear to impose any ACL on the user that invokes the position's opening, meaning it is possible for a malicious user to block the withdrawal of another via race-condition by introducing their position opening transaction first within a block.

Recommendation:

The game theory behind this function should be re-evaluated and a form of access control should potentially be imposed as the current system can be gamed to prevent withdrawals for other users.

Alleviation:

A premium-imposing mechanism was introduced to make such an attack prohibitively costly, nullifying this exhibit.

LIQ-01: Code Duplication

Type	Severity	Location
Gas Optimization	Informational	Liquidation.sol L38

Description:

The linked `if` conditional is replicated as the `return` of the `isLiquidationCandidate` function.

Recommendation:

We advise that the `isLiquidationCandidate` function is directly utilized here to optimize the gas cost of the contract.

Alleviation:

The code was adapted to use the negation of the linked optimized function and adjust the scenarios tested by the `if` clauses, nullifying this exhibit.

LIQ-02: Inexistent Input Sanitization

Type	Severity	Location
Control Flow	Minor	Liquidation.sol L21-L31

Description:

The linked functions adjust the liquidation thresholds and reward amounts, however there is no sanitization conducted to ensure that the `liquidationMaxRewardAmount` will be greater-than the `liquidationMinRewardAmount`, potentially breaking the contract's calculations.

Recommendation:

We advise that corresponding `require` checks are imposed that prohibit this kind of descrepancy at the code-level rather than at the user-level.

Alleviation:

Input thresholds were introduced to properly sanitize the input variables of these functions.

LIQ-03: Reward Regardless of Liquidation

Type	Severity	Location
Logical Issue	Medium	Liquidation.sol L37-L47

Description:

As noted within the `Platform` contract, this function returns a reward regardless of whether the address was liquidated, enabling a user to invoke the `Platform` function multiple times to acquire a reward without liquidating positions.

Recommendation:

We advise that the final `else` clause of the function is re-evaluated and potentially adjusted to also check certain conditions that affect the reward returned.

Alleviation:

Fixed as a side-effect of PLA-02.

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.