Goldfinch

Goldfinch Protocol

Security Assessment

March 23rd, 2021

By:
Camden Smallwood @ CertiK
camden.smallwood@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 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	Goldfinch - Goldfinch Protocol
Description	A lending protocol focused on enabling crypto holders to earn yield from real-world, off-chain borrowers. This is achieved by allowing anyone to contribute USDC to the pool. Governance-approved underwriters can then extend credit linesto individual borrowers, who can draw down capital from the pool for use off-chain.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. f730015f0f92c32e8968cd2f60e6e2152575b850 2. 9c574d9715e924854d1c447b857c4afd53b88a78

Audit Summary

Delivery Date	March 23rd, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	February 23rd, 2021 - March 23rd, 2021

Vulnerability Summary

Total Issues	23
Total Critical	1
Total Major	1
Total Medium	2
Total Minor	1
Total Informational	18

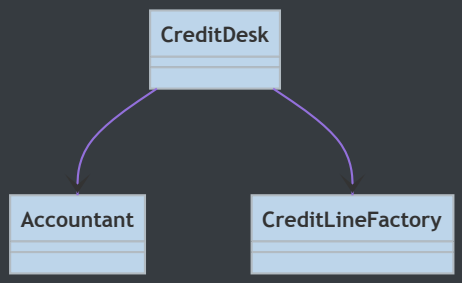
Executive Summary

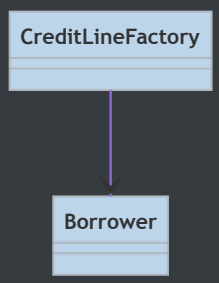
This report represents the results of CertiK's engagement with Goldfinch on their implementation of Goldfinch Protocol. The manual and static analysis were performed in the audit. Our findings mainly refer to optimizations issues, two medium, one minor, critical and major issues. Majority of the issues including critical, major and medium are remediated.

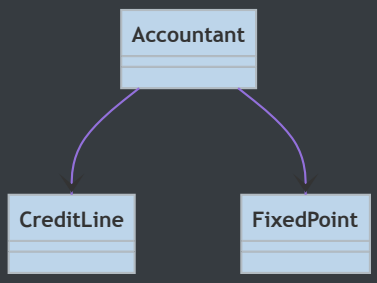
Files In Scope

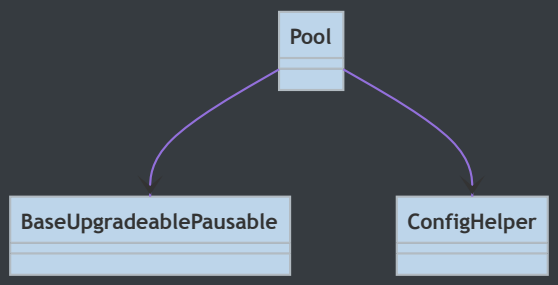
ID	Contract	Location
ACC	Accountant.sol	contracts/protocol/core/Accountant.sol
BOR	Borrower.sol	contracts/protocol/periphery/Borrower.sol
BUP	BaseUpgradeablePausable.sol	contracts/protocol/core/BaseUpgradeablePausable.sol
CDK	CreditDesk.sol	contracts/protocol/core/CreditDesk.sol
CLE	CreditLine.sol	contracts/protocol/core/CreditLine.sol
CHR	ConfigHelper.sol	contracts/protocol/core/ConfigHelper.sol
COS	ConfigOptions.sol	contracts/protocol/core/ConfigOptions.sol
CLF	CreditLineFactory.sol	contracts/protocol/core/CreditLineFactory.sol
FID	Token.sol	contracts/protocol/core/Token.sol
FPT	FixedPoint.sol	contracts/external/FixedPoint.sol
GCG	GoldfinchConfig.sol	contracts/protocol/core/GoldfinchConfig.sol
IFU	IToken.sol	contracts/interfaces/IToken.sol
IPL	IPool.sol	contracts/interfaces/IPool.sol
ICD	ICreditDesk.sol	contracts/interfaces/ICreditDesk.sol
IER	IERC20withDec.sol	contracts/interfaces/IERC20withDec.sol
ICU	ICUSDCContract.sol	contracts/interfaces/ICUSDCContract.sol
POO	Pool.sol	contracts/protocol/core/Pool.sol
PPE	PauserPausable.sol	contracts/protocol/core/PauserPausable.sol

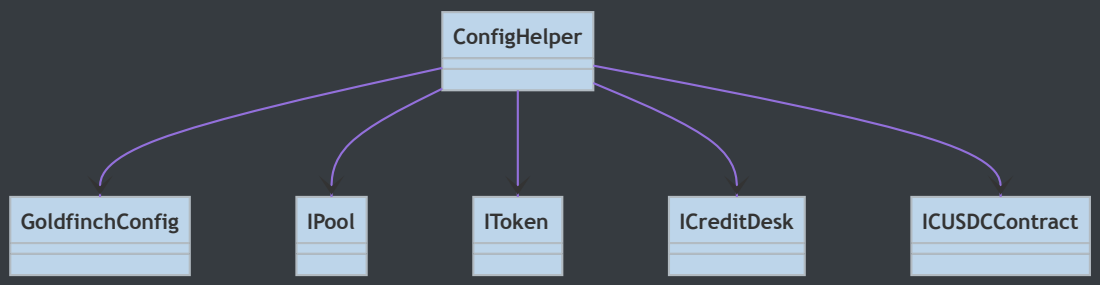
File Dependency Graph

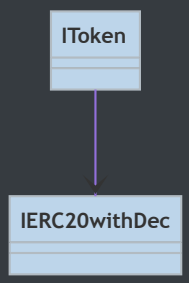


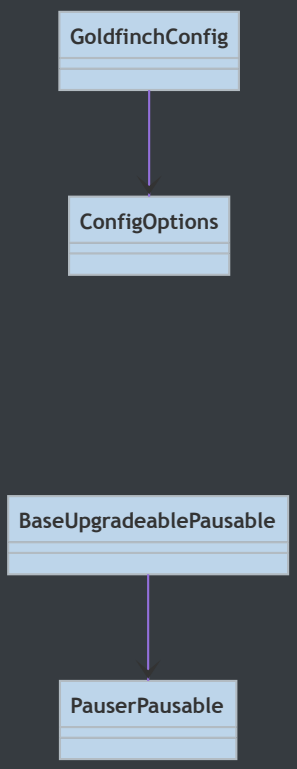




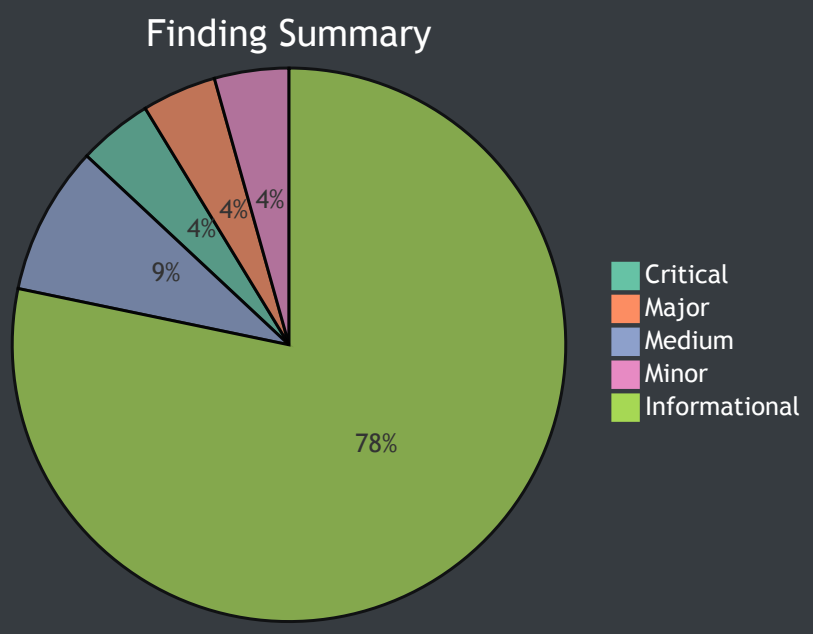

Pool
BaseUpgradeablePausable
ConfigHelper


ConfigHelper
GoldfinchConfig
IPool
IToken
ICreditDesk
ICUSDCContract


IToken
IERC20withDec


GoldfinchConfig
ConfigOptions
BaseUpgradeablePausable
PauserPausable

Findings



ID	Title	Type	Severity	Resolved
POO-01	Redundant Statements	Dead Code	Informational	✓
POO-02	Inefficient storage read	Gas Optimization	Informational	✓
POO-03	Illegible code	Coding Style	Informational	✓
POO-04	Explicitly returning local variable	Gas Optimization	Informational	✓
POO-05	Explicitly returning local variable	Gas Optimization	Informational	⏲

POO-06	Explicitly returning local variable	Gas Optimization	Informational	✓
POO-07	`if` statement can be substituted with a `require` statement	Coding Style	Informational	✓
POO-08	`if` statement can be substituted with a `require` statement	Coding Style	Informational	✓
POO-09	Ineffectual transfer of tokens	Volatile Code	Medium	✓
BOR-01	Redundant variable assignment	Gas Optimization	Informational	✓
BOR-02	Explicitly returning local variable	Gas Optimization	Informational	
BOR-03	Function Visibility Optimization	Gas Optimization	Informational	✓
BOR-04	Possibility of loss of token funds	Volatile Code	Major	✓
BOR-05	`drawdownWithSwapOnOneInch` is publicly callable	Logical Issue	Critical	✓
BOR-06	Variable data location can be changed from `memory` to `calldata`	Gas Optimization	Informational	✓
BOR-07	Lack of verification for array parameter's length	Control Flow	Minor	
BOR-08	Redundant casting to type `address`	Gas Optimization	Informational	✓
CLF-01	Redundant Statements	Dead Code	Informational	✓
CDK-01	Explicitly returning local variable	Gas Optimization	Informational	
CDK-02	Explicitly returning local variable	Gas Optimization	Informational	
CDK-03	Inefficient storage read	Gas Optimization	Informational	✓
CDK-04	Ineffectual code	Control Flow	Medium	✓
CDK-05	Unnecessary restriction of `view` function	Volatile Code		✓

			Informational	

POO-01: Redundant Statements

Type	Severity	Location
Dead Code	Informational	Pool.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 of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

POO-02: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	Pool.sol L244, L261

Description:

The aforementioned lines read `compoundBalance` from storage twice. As reading from storage is expensive, a local variable can be introduced to save `compoundBalance` from storage and then utilized on the aforementioned line.

Recommendation:

We advise to store the `compoundBalance` in a local variable and utilize it on the aforementioned lines to save gas cost associated with additional storage read.

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

POO-03: Illegible code

Type	Severity	Location
Coding Style	Informational	Pool.sol L311

Description:

The function on the aforementioned line converts cUSDC amount to the underlying USDC. The function makes use of confusing `cUSDCMantissa` to perform conversion of cUSDC amount to USDC amount.

Recommendation:

We advise to properly use the currency multipliers to aid in the readiablity of the code.

```
uint256 USDCDecimals = 6;
uint256 cUSDCDecimals = 8;
uint256 USDCDecimalsMultiplier = 10 ** 6;
uint256 cUSDCDecimalsMultiplier = 10 ** 8;
uint256 res = amount
    .mul(exchangeRate)
    .mul(USDCDecimalsMultiplier)
    .div(10 ** (18 + USDCDecimals - cUSDCDecimals))
    .div(cUSDCDecimalsMultiplier);
```

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

POO-04: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	Pool.sol L311

Description:

The function on the aforementioned line explicitly returns a local variable which increases the overall cost of gas.

Recommendation:

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

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`. The return of expression is utilized in the function.

POO-05: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	Pool.sol L135

Description:

The function `transferFrom` on the aforementioned line explicitly returns a local variable which increases the overall cost of gas.

Recommendation:

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

Alleviation:

No alleviations.

POO-06: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	Pool.sol L154

Description:

The function `drawdown` on the aforementioned line explicitly returns a local variable which increases the overall cost of gas.

Recommendation:

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

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

POO-07: `if` statement can be substituted with a `require` statement

Type	Severity	Location
Coding Style	Informational	Pool.sol L254-L259

Description:

The `if` statements on the aforementioned lines can be subsituted with the `require` statements to increase the legibility of the codebase.

Recommendation:

We advise to substitute the `if` statements on the aforementioned lines with `require` statements.

```
require(
   error == 0,
   "Sweep from compound failed"
);

require(
   postRedeemUSDCBalance.sub(preRedeemUSDCBalance) == redeemedUSDC,
   "Unexpected redeem amount"
);
```

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

POO-08: `if` statement can be substituted with a `require` statement

Type	Severity	Location
Coding Style	Informational	Pool.sol L237-L239

Description:

The `if` statement on the aforementioned line can be subsituted with a `require` statement to increase the legibility of the codebase.

Recommendation:

We advise to subsitute the `if` statement on the aforementioned line with a `require` statement.

```
require(
   error == 0,
   Sweep to compound failed
);
```

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

POO-09: Ineffectual transfer of tokens

Type	Severity	Location
Volatile Code	Medium	Pool.sol L285

Description:

The aforementioned line transfers token to the contract itself. The sender of the token is derived from the function parameter `from`. This function parameter can be the contract itself when the function is called on `L262` in `sweepFromCompound` function rendering transfer of token at that instance redundant. It can result in reverting of transaction if the token contract does not allow the sender and recipient to be the same addresses.

Recommendation:

We advise to make changes to the code such that `L285-L286` are only executed when `from` is not the current contract's address.

```
if (from != address(this)) {
   bool success = doUSDCTransfer(from, address(this), principal.add(poolAmount));
      require(success, "Failed to collect principal repayment");
}
```

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

BOR-01: Redundant variable assignment

Type	Severity	Location
Gas Optimization	Informational	Borrower.sol L44

Description:

The aforementioned line performs redundant assignment of signature to the variable `data`. The same signature is already assigned to the variable on `L42`.

Recommendation:

We advise to remove the redundant assignment of signature to the variable `data` on the aforementioned line.

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

BOR-02: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	Borrower.sol L175

Description:

The function `invoke` on the aforementioned line explicitly returns a local variable which increases the overall cost of gas.

Recommendation:

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

Alleviation:

No alleviations.

BOR-03: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	Borrower.sol L67

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:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

BOR-04: Possibility of loss of token funds

Type	Severity	Location
Volatile Code	Major	Borrower.sol L82

Description:

When withdraw is performed through `drawdownWithSwapOnOneInch` function and `addressToSendTo` parameter is either provided as zero address or provided as the contract's address itself then any `toToken` swapped from USDC seems to be stuck in the borrower contract without any functionality to withdraw them resulting in loss of token funds.

Recommendation:

We advise to set a default value for the recipient e.g. `msg.sender`, so that the token funds are not lost in the event when `toAddress` parameter is either zero adress or it is the contract's address.

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

BOR-05: `drawdownWithSwapOnOneInch` is publicly callable

Type	Severity	Location
Logical Issue	Critical	Borrower.sol L67

Description:

The function `drawdownWithSwapOnOneInch` on the aforementioned line is publicly callable and anyone can call this function and steal funds from the contract's creditline.

Recommendation:

We advise to introduce modifier `onlyAdmin` in the signature of the function, so only admin of the contract could call this function.

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

BOR-06: Variable data location can be changed from `memory` to `calldata`

Type	Severity	Location
Gas Optimization	Informational	Borrower.sol L106

Description:

The function `payMultiple` on the aforementioned line has visibility `external` yet specifies data location for its array type parameters to be `memory`. It is an inefficient implementation as variables' data location can be changed to `calldata` to avoid copying of the parameters to `memory` and saving gas cost associated with it.

Recommendation:

We advise to change the data location of the function parameters on the aforementioned lines from `memory` to `calldata`.

```
function payMultiple(address[] calldata creditLines, uint256[] calldata amounts)
external onlyAdmin {...}
```

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

BOR-07: Lack of verification for array parameter's length

Type	Severity	Location
Control Flow	Minor	Borrower.sol L106

Description:

The function `payMultiple` receives array type parameters of `creditLines` and `amounts`. If both of these parameters are passed without any elements then the function will still successfully execute without having any effects.

Recommendation:

We advise to introduce a check asserting that any either of the parameter has length greater than zero.

Alleviation:

No alleviations.

BOR-08: Redundant casting to type `address`

Type	Severity	Location
Gas Optimization	Informational	Borrower.sol L125, L142

Description:

The aforementioned lines perform redundant castings to type `address` as the variables being casted are already of type `address`.

Recommendation:

We advise to remove the redundant castings on the aforementioned lines to save gas cost associated with the castings.

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

CLF-01: Redundant Statements

Type	Severity	Location
Dead Code	Informational	CreditLineFactory.sol L42-L63

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 of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

CDK-01: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	CreditDesk.sol L97

Description:

The function `createCreditLine` on the aforementioned line explicitly returns a local variable which increases overall cost of gas.

Recommendation:

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

Alleviation:

No alleviations.

CDK-02: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	CreditDesk.sol L557

Description:

The function `getCreditCurrentlyExtended` on the aforementioned line explicitly returns a local variable which increases overall cost of gasg.

Recommendation:

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

Alleviation:

No alleviations.

CDK-03: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	CreditDesk.sol L547, L550

Description:

The aforementioned lines read `underwriter.governanceLimit` from the storage of the contract. The code can be optimized by storing the result of storage read in a local variable and then utilizing it on the aforementioned lines to reduce gas cost associated with an additional storage read.

Recommendation:

We advsie to make use of local variable to store the storage read's result and then utilize it on the aforementioned lines.

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

CDK-04: Ineffectual code

Type	Severity	Location
Control Flow	Medium	CreditDesk.sol L193-L194

Description:

The aforementioned lines performs withdraw of USDC from the `Pool` contract. The argument `amount` passed to the `drawdown` function call can be zero when all the withdraw amount is avaibale from the CreditLine contract and the `amount` variable is set to `0` on `L176`. Passing of `0` amount to `drawdown` call can result in failing transaction if the `transferFrom` call from USDC reverts the `0` transfer or the future changes in the `Pool` contract disallows drawdown of `0` amount.

Recommendation:

We advise to only execute the aforementioned lines when the amount is not zero to avoid the ineffectual function call and unexpected behaviour.

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

CDK-05: Unnecessary restriction of `view` function

Type	Severity	Location
Volatile Code	Informational	CreditDesk.sol L307, L315

Description:

The functions `getUnderwriterCreditLines` and `getBorrowerCreditLines` on the aforementioned lines are intended as public getters yet unnecessarily restricted with `whenNotPaused` modifier.

Recommendation:

We advise to remove the `whenNotPaused` modifier from the signatures of the functions on the aforementioned lines.

Alleviation:

Alleviations were applied as of commit hash `9c574d9715e924854d1c447b857c4afd53b88a78`.

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 invokeable 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.