# XCAmpleforth

## Security Assessment

February 1st, 2021

For :
XCAmpleforth

# 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 | XCAmpleforth |
| --- | --- |
| Description | A cross-chain bridge and token implementation of the Ampleforth rebasing currency. |
| Platform | Ethereum; Solidity, Yul |
| Codebase | GitHub Repository |
| Commits | 1. 954d0d20de14a4a7641f1592a33410dd16059a2c<br>2. 9fd087667de9ae29db95945f7e42c749fbe75b9a |

## Audit Summary

| Delivery Date | February 1st, 2021 |
| --- | --- |
| Method of Audit | Static Analysis, Manual Review |
| Consultants Engaged | 2 |
| Timeline | January 25th, 2021 - February 1st, 2021 |

## Vulnerability Summary

| Total Issues | 14 |
| --- | --- |
| Total Critical | 0 |
| Total Major | 0 |
| Total Medium | 2 |
| Total Minor | 2 |
| Total Informational | 10 |

# Executive Summary

We were tasked with auditing the cross-chain Ampleforth bridge implementation that is meant to enable cross-chain transfers of the AMPL token from and to satellite chains that are different than the main Ethereum network.

The bridge operates by relaying the supply at the time of a cross-chain transfer in addition to the amount of tokens transferred, thus preventing any issues that would conventionally arise from the dynamic rebase mechanism of Ampleforth and the desynchronization of satellite chains in relation to the main chain.

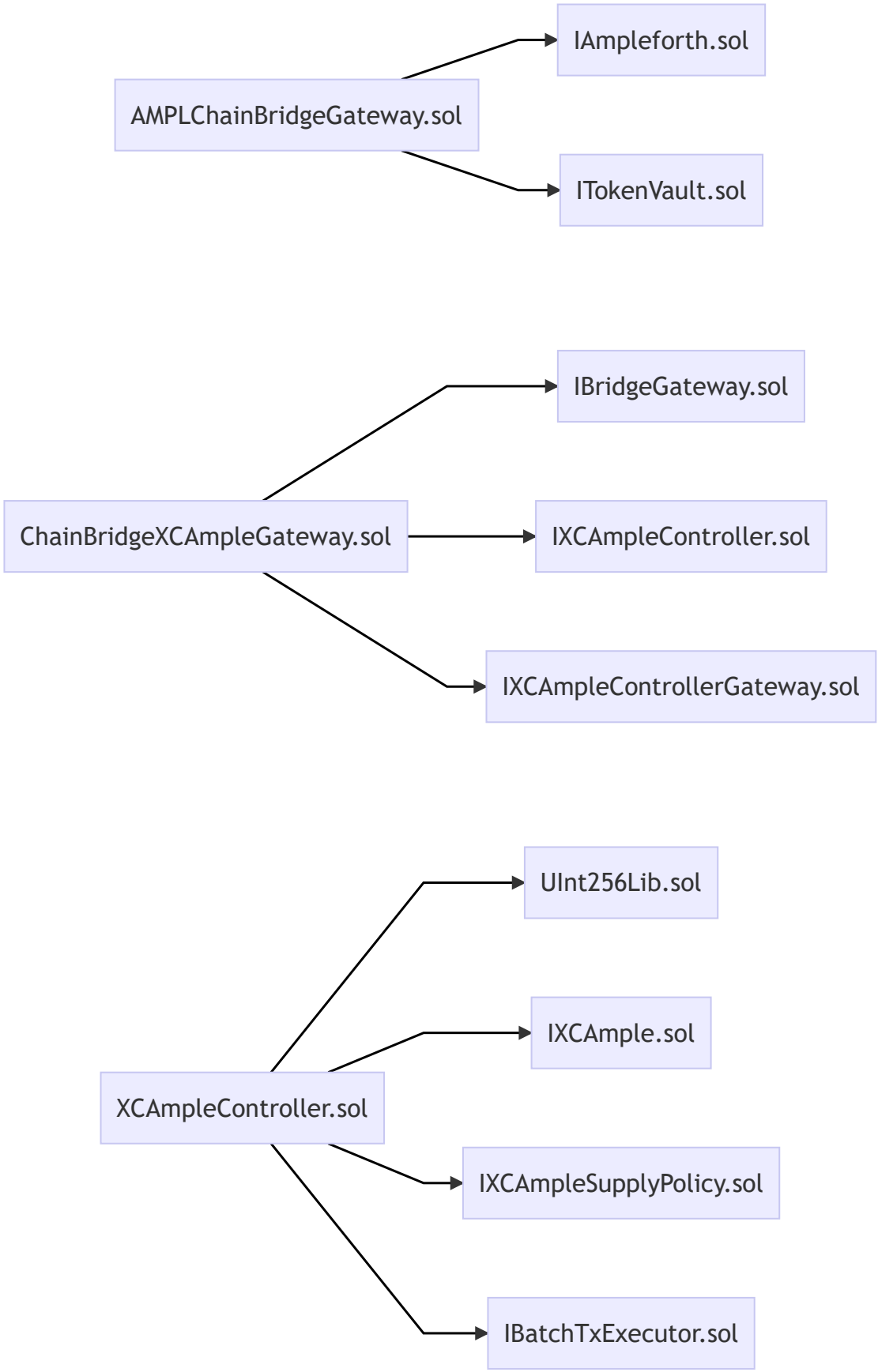
The codebase has been developed with the latest security standards in mind and as such, our findings mostly consisted of optimizations that could be applied to the codebase as well as an extended set of best practises that would aid in the maintainability of the codebase. We were able to pinpoint a minor issue in the way the `DOMAIN_SEPARATOR` is utilized in the EIP-2612 implementation of permits on the `XCAmple.sol` implementation, however the issue was promptly dealt with.

Overall, the security of the codebase can be deemed to be of a high standard.

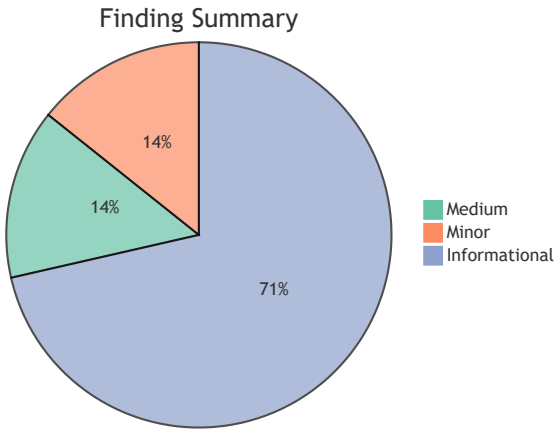
# Files In Scope

| ID | Contract | Location |
|---|---|---|
| AMP | AMPLChainBridgeGateway.sol | contracts/base-chain/bridge-gateways/AMPLChainBridgeGateway.sol |
| CBX | ChainBridgeXCAmpleGateway.sol | contracts/satellite-chain/bridge-gateways/ChainBridgeXCAmpleGateway.sol |
| TVT | TokenVault.sol | contracts/base-chain/TokenVault.sol |
| UIL | UInt256Lib.sol | contracts/satellite-chain/xc-ampleforth/UInt256Lib.sol |
| XCA | XCAmple.sol | contracts/satellite-chain/xc-ampleforth/XCAmple.sol |
| XCC | XCAmpleController.sol | contracts/satellite-chain/xc-ampleforth/XCAmpleController.sol |

# File Dependency Graph (BETA)



# Findings



| ID | Title | Type | Severity | Resolved |
| --- | --- | --- | --- | --- |
| TVT-01 | Function Simplification | Gas Optimization | Informational | ✓ |
| TVT-02 | Potential Incompatibility w/ Underlying Token | Language Specific | Minor | ✓ |
| AMP-01 | Mutability Specifiers Missing | Gas Optimization | Informational | ✓ |
| AMP-02 | Potential Loss of Precision | Logical Issue | Medium | ✓ |
| CBX-01 | Mutability Specifiers Missing | Gas Optimization | Informational | ✓ |
| CBX-02 | Potential Loss of Precision | Mathematical Operations | Medium | ✓ |
| UIL-01 | Usage of Deprecated Representation | Coding Style | Informational | ✓ |
| UIL-02 | Inexistent Error Message | Coding Style | Informational | ✓ |
| XCC-01 | Function Simplification | Gas Optimization | Informational | ✓ |
| XCA-01 | Usage of Deprecated Representation | Coding Style | Informational | ✓ |
| XCA-02 | Incorrect Utilization of `chainid` | Logical Issue | Minor | ✓ |
| XCA-03 | Usage of `memory` Variable Over `storage` | Gas Optimization | Informational | ✓ |
| XCA-04 | Approval Amount Desync | Mathematical Operations | Informational | ✓ |
| XCA-05 | Inexistent Error Message | Coding Style | Informational | ✓ |

# TVT-01: Function Simplification

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | TokenVault.sol L56-L82 |

## Description:

The linked functions toggle the `bool` state of the `whitelistedBridgeGateways` mapping to adjust whether a particular bridge is whitelisted to withdraw and deposit tokens to the vault.

## Recommendation:

As the toggle mechanism can only utilize two states, these two functions can be combined into a single one that accepts a `bool` variable as input, reducing the bytecode size of the contract and thus the overall gas footprint of its deployment.

## Alleviation:

The team stated that while a reduction in gas cost would be achievable, they opted to retain the structure as it currently is to minimize critical operational errors.

# TVT-02: Potential Incompatibility w/ Underlying Token

| Type | Severity | Location |
| --- | --- | --- |
| Language Specific | Minor | TokenVault.sol L95, L110 |

## Description:

The vault is meant to be utilized with the Ampleforth main-chain currency on Ethereum which currently conforms to the ERC-20 standard properly, however, this may not always be the case.

## Recommendation:

As the main chain Ampleforth implementation utilizes the proxy pattern, it is possible that an upgrade of the protocol will no longer be fully compliant with the ERC-20 standard causing the strict `require` checks utilized in the vault to fail and thus preventing any type of cross-chain transfer from occuring again. Although the likelihood of this scenario is low, it is still a plausible scenario as the same ERC-20 incompatibility is observed in the Tether stablecoin and has caused significant issues in the past.

It is more optimal to utilize the `SafeERC20` OpenZeppelin library implementation for conducting ERC-20 transfers as it is fully compatible with all types of ERC-20 tokens and will also allow the Ampleforth codebase to be utilized by other projects.

## Alleviation:

The issue was fully remediated by using the `SafeERC20` implementation by OpenZeppelin.

# AMP-01: Mutability Specifiers Missing

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | AMPLChainBridgeGateway.sol L41-L43 |

## 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 introduced the `immutable` keyword to the specified variables thus optimizing the gas cost involved in utilizing them.

# AMP-02: Potential Loss of Precision

| Type | Severity | Location |
| --- | --- | --- |
| Logical Issue | Medium | AMPLChainBridgeGateway.sol L111 |

## Description:

The Ampleforth protocol follows a strict rebase policy whereby consequent rebases will never incur loss of precision in the underlying values used to transact with the currency as denoted by their `uFragments.sol` supply adjustment analysis. Multiple epoch rebases that can accumulate, however, do not guarantee the same constraints in the rebase operation of the cross-chain Ampleforth in comparison to the main-chain Ampleforth.

## Recommendation:

We advise that a subsequent thorough analysis is performed on the impacts of accumulated rebases to the cross-chain transfers of AMPL to xcAMPL, as this can have a significant impact to the currency as a whole. Solutions to the introduction of accumulated rebases would be ensuring at the code level that cross-chain transactions fail if multiple epochs have passed on both chains and that `rebase` operations on satellite chains occur on each consequent epoch and fail if an attempt is made to 'skip' intermediate epoch adjustments.

## Alleviation:

The team responded that in the case that a rebase epoch on the main chain is not propagated to the satellite chain in due time, the satellite chain balances will be temporarily out of sync. However, this mechanism is not exploitable as cross-chain AMPL transfers are accompanied by the current total supply at the time of the transfer, meaning that no arbitrage can occur in this scenario. As such, all types of cross-chain transfers will be pristine.

# CBX-01: Mutability Specifiers Missing

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | ChainBridgeXCAmpleGateway.sol L38-L39 |

## 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 introduced the `immutable` keyword to the specified variables thus optimizing the gas cost involved in utilizing them.

# CBX-02: Potential Loss of Precision

| Type | Severity | Location |
| --- | --- | --- |
| Mathematical Operations | Medium | ChainBridgeXCAmpleGateway.sol L84 |

## Description:

The Ampleforth protocol follows a strict rebase policy whereby consequent rebases will never incur loss of precision in the underlying values used to transact with the currency as denoted by their `uFragments.sol` supply adjustment analysis. Multiple epoch rebases that can accumulate, however, do not guarantee the same constraints in the rebase operation of the cross-chain Ampleforth in comparison to the main-chain Ampleforth.

## Recommendation:

We advise that a subsequent thorough analysis is performed on the impacts of accumulated rebases to the cross-chain transfers of AMPL to xcAMPL, as this can have a significant impact to the currency as a whole. Solutions to the introduction of accumulated rebases would be ensuring at the code level that cross-chain transactions fail if multiple epochs have passed on both chains and that `rebase` operations on satellite chains occur on each consequent epoch and fail if an attempt is made to 'skip' intermediate epoch adjustments.

## Alleviation:

The team responded that in the case that a rebase epoch on the main chain is not propagated to the satellite chain in due time, the satellite chain balances will be temporarily out of sync. However, this mechanism is not exploitable as cross-chain AMPL transfers are accompanied by the current total supply at the time of the transfer, meaning that no arbitrage can occur in this scenario. As such, all types of cross-chain transfers will be pristine.

# UIL-01: Usage of Deprecated Representation

| Type | Severity | Location |
| --- | --- | --- |
| Coding Style | Informational | UInt256Lib.sol L9 |

## Description:

The maximum value of `int256` is currently represented by the contract using bitwise shifts and negations whereas the current standard utilizes the special `type` keyword to wrap the type and access the `max` member i.e. `type(int256).max`, as introduced in Solidity 0.6.8.

## Recommendation:

We advise that the representation style is changed to the one mentioned in this exhibit's description.

## Alleviation:

The team has replaced the deprecated representation with `type(uint256).max`.

## UIL-02: Inexistent Error Message

| Type | Severity | Location |
| --- | --- | --- |
| Coding Style | Informational | UInt256Lib.sol L15 |

### Description:

Error messages should always accompany a `require` invocation to aid in both explaining what the imposed check is meant to achieve as well as aid in debugging processes.

### Recommendation:

We advise that an error message is provided for the linked `require` check.

### Alleviation:

The team introduced an error message to the linked `require` check.

# XCC-01: Function Simplification

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | XCAmpleController.sol L83-L99 |

## Description:

The linked functions toggle the `bool` state of the `whitelistedBridgeGateways` mapping to adjust whether a particular bridge is whitelisted to burn and mint tokens in the satellite chain.

## Recommendation:

As the toggle mechanism can only utilize two states, these two functions can be combined into a single one that accepts a `bool` variable as input, reducing the bytecode size of the contract and thus the overall gas footprint of its deployment.

## Alleviation:

The team stated that while a reduction in gas cost would be achievable, they opted to retain the structure as it currently is to minimize critical operational errors.

# XCA-01: Usage of Deprecated Representation

| Type | Severity | Location |
| --- | --- | --- |
| Coding Style | Informational | XCAmple.sol L46, L54 |

## Description:

The maximum value of `uint256` and `uint128` is currently represented by the contract using bitwise negations whereas the current standard utilizes the special `type` keyword to wrap the type and access the `max` member i.e. `type(uint256).max`, as introduced in Solidity 0.6.8.

## Recommendation:

We advise that the representation style is changed to the one mentioned in this exhibit's description.

## Alleviation:

The team has replaced the deprecated representation with `type(uint256).max`.

## XCA-02: Incorrect Utilization of `chainid`

| Type | Severity | Location |
| --- | --- | --- |
| Logical Issue | Minor | XCAmple.sol L88, L148-L160 |

### Description:

The linked code relates to the utilization of the `chainid` variable which is meant to represent the current chain's ID for usage in the EIP-712 and EIP-2612 standards. As noted in the EIPs, by computing the `chainid` once the standards' functions are susceptible to cross-chain attacks in case of a fork.

### Recommendation:

When an Ethereum-based chain is forked, its chain ID changes whilst its state remains the same at the point of forking. This means that the forked chain's XCAmple implementation will be utilizing an incorrect `chainid` to validate signatures with. This can lead to replay attacks whereby a single EIP-712 signature is valid for both the forked chain and the base chain.

To alleviate this, the `chainid` and consequent hash of the `DOMAIN_SEPARATOR` need to be computed on a need-to-use basis. Otherwise, if compatibility in the forked chain is of no concern, the `chainid` computed during the `initialize` function can be stored at a contract-level variable and consequently compared on each EIP-712 bearing function to a dynamically evaluated `chainid`, throwing in case the chain IDs do not match. This will prevent replay attacks in forks, however, it will also render the EIP-712 scheme unusable in the forked chain implementation.

### Alleviation:

The team refactored the way the `DOMAIN_SEPARATOR` is computed by replacing it with a function call that dynamically computes the `chainid` of the current chain and prevents cross-chain replay attacks due to a misassigned `chainid`. A further optimization that could be done at this point is to cache the `keccak256` result of the "main-chain" and, should the `chainid` be differt, dynamically compute it. This should optimize the gas cost of the function significantly.

## XCA-03: Usage of `memory` Variable Over `storage`

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | XCAmple.sol L114 |

### Description:

The linked line performs a `return` statement on the `globalAMPLSupply` stored in `storage` after performing a successful equality check of this value with the in-memory `newGlobalAMPLSupply` variable.

### Recommendation:

As the current `return` statement performs a redundant `storage` read operation, we advise that the `newGlobalAMPLSupply` variable is instead returned here optimizing the gas cost of the function.

### Alleviation:

The linked segment was adjusted to prioritize utilization of in-memory variables where possible instead of in-storage ones.

# XCA-04: Approval Amount Desync

| Type | Severity | Location |
| --- | --- | --- |
| Mathematical Operations | Informational | XCAmple.sol L326-L331 |

## Description:

As the Ampleforth is unique in the sense that it is a rebasing token, a set amount of permitted tokens to be transmitted via an approval can have a different underlying gon value due to supply rebases.

## Recommendation:

While this is a well known trait of the protocol, it should still be mentioned in the accompanying comments of the `approve` function to ensure users of the codebase are fully aware of this functionality. Additionally, an optional deviation threshold can be introduced here whereby a user permits their allowances to deviate in the underlying gon value by a set amount to prevent sharp differences in supply being taken advantage of by approved addresses.

## Alleviation:

A comment was introduced properly defining this behaviour.

# XCA-05: Inexistent Error Message

| Type | Severity | Location |
| --- | --- | --- |
| Coding Style | Informational | XCAmple.sol L419, L429 |

## Description:

Error messages should always accompany a `require` invocation to aid in both explaining what the imposed check is meant to achieve as well as aid in debugging processes.

## Recommendation:

We advise that an error message is provided for the linked `require` check.

## Alleviation:

The team introduced error messages to the linked `require` checks.

# 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.