Xend Finance: Yearn Savings

Security Assessment

December 6th, 2020

[Preliminary Report]

For :
Xend Finance: Yearn Savings

By :
Camden Smallwood @ CertiK
camden.smallwood@certik.io

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	Xend Finance: Yearn Savings
Description	Then codebase comprise of contracts which allow staking of DAI tokens individually and as group and allows yield farming on Yearn Dai contract.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 0b93bc9c84e53dffca0e0c292fb1d214104fa241

Audit Summary

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

Vulnerability Summary

Total Issues	74
Total Critical	5
Total Major	5
Total Medium	3
Total Minor	12
Total Informational	49

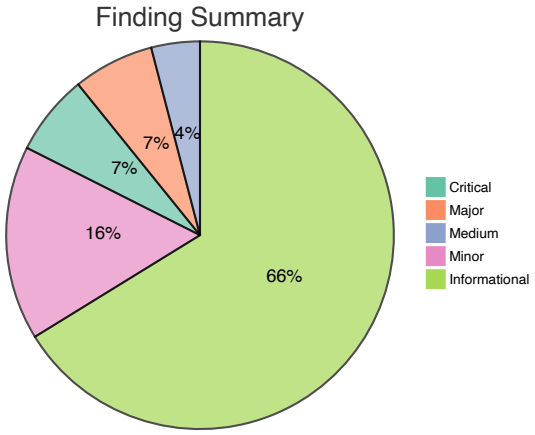
Executive Summary

This section will represent the summary of the whole audit process once it has concluded.

Files In Scope

ID	Contract	Location
CYC	Cycle.sol	Cycle.sol
CRD	ClientRecord.sol	ClientRecord.sol
GRO	Groups.sol	Groups.sol
ICE	ICycle.sol	ICycle.sol
IGS	IGroups.sol	IGroups.sol
IGR	IGroupSchema.sol	IGroupSchema.sol
ICR	IClientRecord.sol	IClientRecord.sol
ISC	ISavingsConfig.sol	ISavingsConfig.sol
ICS	IClientRecordShema.sol	IClientRecordShema.sol
ISS	ISavingsConfigSchema.sol	ISavingsConfigSchema.sol
SCG	SavingsConfig.sol	SavingsConfig.sol
SOS	StorageOwners.sol	StorageOwners.sol
TRE	Treasury.sol	Treasury.sol
XFG	XendFinanceGroup_Yearn_V1.sol	XendFinanceGroup_Yearn_V1.sol
XFI	XendFinanceIndividual_Yearn_V1.sol	XendFinanceIndividual_Yearn_V1.sol

Findings



ID	Title	Type	Severity	Resolved
IGR-01	Unlocked Compiler Version	Language Specific	Informational	!
IGR-02	Inefficient struct layout	Gas Optimization	Informational	!
IGR-03	Inefficient struct layout	Gas Optimization	Informational	!
IGR-04	Inefficient struct layout	Gas Optimization	Informational	!
SOS-01	Unlocked Compiler Version	Language Specific	Informational	!
SOS-02	Inefficient functions	Gas Optimization	Informational	!
SOS-03	`if` statement can subsituted with a `require` statement	Volatile Code	Minor	!
GRO-01	Unlocked Compiler Version	Language Specific	Informational	!
GRO-02	Unsafe addition	Mathematical Operations	Minor	!
GRO-03	Unsafe subtraction	Mathematical	Minor	

		Operations		
GRO-04	Unsafe addition	Mathematical Operations	Minor	
GRO-05	Unsafe subtraction	Mathematical Operations	Minor	
GRO-06	Inefficient function implementation	Gas Optimization	Informational	
GRO-07	Comparison with a literal boolean value	Gas Optimization	Informational	
GRO-08	Explicitly returning a local variable	Gas Optimization	Informational	
CYC-01	Unlocked Compiler Version	Language Specific	Informational	
CYC-02	Explicitly returning a local variable	Gas Optimization	Informational	
CYC-03	Comparison with a literal boolean value	Gas Optimization	Informational	
CYC-04	Inefficient function implementation	Gas Optimization	Informational	
CYC-05	`updateCycleMember` is not restricted by `onlyStorageOracle`	Volatile Code	Critical	
CYC-06	Incorrect code	Control Flow	Major	
CYC-07	Incorrect implementation of functions	Volatile Code	Major	
CRD-01	Unlocked Compiler Version	Language Specific	Informational	
CRD-02	Inefficient local variable	Gas Optimization	Informational	
CRD-03	Comparison with a	Gas Optimization	Informational	

	literal boolean value			
CRD-04	Redundant Variable Initialization	Coding Style	Informational	
CRD-05	Inefficient storage update	Gas Optimization	Informational	
ICS-01	Unlocked Compiler Version	Language Specific	Informational	
ISS-01	Unlocked Compiler Version	Language Specific	Informational	
ISS-02	Inefficient struct layout	Gas Optimization	Informational	
SCG-01	Unlocked Compiler Version	Language Specific	Informational	
SCG-02	Empty constructor declaration	Volatile Code	Informational	
SCG-03	Comparison with a literal boolean value	Gas Optimization	Informational	
SCG-04	Explicitly returning a local variable	Gas Optimization	Informational	
SCG-05	Redundant Statements	Dead Code	Informational	
SCG-06	`_validateRuleCreation` always reverts the transaction	Volatile Code	Major	
SCG-07	`modifyRule` does not save the Rule	Volatile Code	Major	
TRE-01	Unlocked Compiler Version	Language Specific	Informational	
TRE-02	Redundant `require` statement	Gas Optimization	Informational	
TRE-03	Redundant	Gas Optimization	Informational	

	`require` statement			
TRE-04	Comparison with a literal boolean value	Gas Optimization	Informational	
TRE-05	`enum` type is declared but never used	Dead Code	Informational	
XFI-01	Unlocked Compiler Version	Language Specific	Informational	
XFI-02	Redundant Variable Initialization	Coding Style	Informational	
XFI-03	Redundant storage variables	Gas Optimization	Informational	
XFI-04	Comparison with a literal boolean value	Gas Optimization	Informational	
XFI-05	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	
XFI-06	Inefficient local variable	Gas Optimization	Informational	
XFI-07	Function does not return a value	Logical Issue	Minor	
XFI-08	Unused local variables	Dead Code	Informational	
XFI-09	Incorrect code	Logical Issue	Critical	
XFI-10	Incorrect value provided for struct property	Logical Issue	Critical	
XFI-11	Incorrect code	Logical Issue	Critical	
XFI-12	Incorrect code	Logical Issue	Critical	

XFI-13	Possibility of re-entrancy attack	Control Flow	Medium	
XFG-01	Unlocked Compiler Version	Language Specific	Informational	
XFG-02	Redundant Variable Initialization	Coding Style	Informational	
XFG-03	Comparison with a literal boolean value	Gas Optimization	Informational	
XFG-04	Unnecessary local variables	Gas Optimization	Informational	
XFG-05	Explicitly returning a local variable	Gas Optimization	Informational	
XFG-06	Unnecessary local variables	Gas Optimization	Informational	
XFG-07	Function does return a value	Dead Code	Minor	
XFG-08	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	
XFG-09	Inefficient code	Gas Optimization	Informational	
XFG-10	Unsafe subtraction	Mathematical Operations	Minor	
XFG-11	Unnecessary parenthesis around expressions	Language Specific	Informational	
XFG-12	Confusing modifier name	Inconsistency	Informational	
XFG-13	Unused local variables	Gas Optimization	Informational	
XFG-14	Unsafe subtraction	Mathematical Operations	Minor	
XFG-15	Requisite Value of	Logical Issue	Minor	

		ERC-20 `transferFrom()` / `transfer()` Call			
	XFG-16	Ineffectual code	Gas Optimization	Informational	
	XFG-17	Anyone can make a particular depositor join cycle	Volatile Code	Major	
	XFG-18	Possibility of reentrancy attack	Control Flow	Medium	
	XFG-19	Possibility of reentrancy attack	Control Flow	Medium	

IGR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IGroupSchema.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

IGR-02: Inefficient struct layout

Type	Severity	Location
Gas Optimization	Informational	IGroupSchema.sol L5, L9

Description:

The struct properties of `bool` and `address` on the aforementioned lines can be placed together to tight pack the storage layout of the struct.

Recommendation:

We recommend to place the struct properties of `bool` and `address` types on the aforementioned lines together so they utilize only one slot instead of two slots.

IGR-03: Inefficient struct layout

Type	Severity	Location
Gas Optimization	Informational	IGroupSchema.sol L13, L20, L26

Description:

The struct properties of `bool` and `enum` on the aforementioned lines can be placed together to to tight pack the struct.

Recommendation:

We advise to place the struct properties of `bool` and `enum` on the aforementioned lines next to each other so they are stored in a single storage slot instead of three slots.

IGR-04: Inefficient struct layout

Type	Severity	Location
Gas Optimization	Informational	IGroupSchema.sol L49, L52, L56

Description:

The struct properties of `bool` and `address` types on the aforementioned lines can be placed together to tight pack the struct.

Recommendation:

We advise to place the struct properties of `bool` and `address` on the aforementioned lines next to each other so they are stored in a single storage slot instead of three slots.

SOS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	StorageOwners.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

SOS-02: Inefficient functions

Type	Severity	Location
Gas Optimization	Informational	StorageOwners.sol L11, L15

Description:

The functions on the aforementioned lines can be replaced with a single function accepting the activate or decactive state of the oracle as a `bool` parameter in the function signature. This will reduce the bytecode footprint of the contract resulting in reduced gas cost for the deployment of the contract.

Recommendation:

We recommend to replace functions on the aforementioned with a single function that accepts status of the oracle to change as `bool` parameter in the function signature.

```
function changeStorageOracleStatus(address oracle, bool status) external onlyOwner {
        storageOracles[oracle] = status;
}
```

SOS-03: `if` statement can subsituted with a `require` statement

Type	Severity	Location
Volatile Code	Minor	StorageOwners.sol L28-L30

Description:

The `if` statement on the aforementioned does not revert the transaction when the `newOwner` is set to `address(0)`.

Recommendation:

We advise to use a `require` statement in place of `if` statement that reverts when `newOwner` is set to `address(0)`.

GRO-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Groups.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

GRO-02: Unsafe addition

Type	Severity	Location
Mathematical Operations	Minor	Groups.sol L55

Description:

The aforementioned line performs unsafe addition which can result in overflow of integer value.

Recommendation:

We advise to use the `add` function from `SafeMath` library to perform safe addition which reverts the transaction if overflow happens.

```
totalTokensDeposited[tokenAddress] = totalTokensDeposited[tokenAddress].add(amount);
```

GRO-03: Unsafe subtraction

Type	Severity	Location
Mathematical Operations	Minor	Groups.sol L69

Description:

The aforementioned line performs unsafe subtraction which can be result in underflow of integer value.

Recommendation:

We advise to use `sub` function from `SafeMath` library to perform subtraction so that the transaction is reverted if underflow happens.

```
totalTokensDeposited[tokenAddress] = totalTokensDeposited[tokenAddress].sub(amount);
```

GRO-04: Unsafe addition

Type	Severity	Location
Mathematical Operations	Minor	Groups.sol L86

Description:

The aforementioned line performs unsafe addition which can result in overflow of integer value.

Recommendation:

We advise to use `add` function of `SafeMath` library to perform addition so that the transaction is reverted if overflow happens.

```
totalEthersDeposited = totalEthersDeposited.add(amount);
```

GRO-05: Unsafe subtraction

Type	Severity	Location
Mathematical Operations	Minor	Groups.sol L99

Description:

The aforementioned line performs unsafe subtraction which can result in underflow of integer value.

Recommendation:

We recommend to use `sub` function of `SafeMath` library to perform subtraction so that the transaction is reverted if underflow happens.

```
totalEthersDeposited = totalEthersDeposited.sub(amount);
```

GRO-06: Inefficient function implementation

Type	Severity	Location
Gas Optimization	Informational	Groups.sol L208

Description:

The implementation of the function on the aforementioned line is inefficient as it redundantly checks `bool` value with an `if` statement and then returns it as is.

Recommendation:

We advise to remove the `if-else` block and directly return the expression `MemberIndexer[depositor].exists` from the function to have efficient implementation.

GRO-07: Comparison with a literal boolean value

Type	Severity	Location
Gas Optimization	Informational	Groups.sol L1

Description:

The contract has several occurrences of comparison with a literal boolean values of `true` or `false` that can be replaced replacing with compared expression itself to increase the legibility of the code.

Recommendation:

We advise to use the compared expression itself in place of expression's comparison with a boolean literal. The expression can be replaced as is when the expression is expected to evaluate to `true` and negation of expression can be used when the expression is expected to have `false` value.

GRO-08: Explicitly returning a local variable

Type	Severity	Location
Gas Optimization	Informational	Groups.sol L123, L130, L194, L397, L180, L295

Description:

The functions on the aforementioned line explicitly return 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.

CYC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Cycle.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

CYC-02: Explicitly returning a local variable

Type	Severity	Location
Gas Optimization	Informational	Cycle.sol L370, L459, L453, L446, L428, L491, L465, L356, L361

Description:

The functions on the aforementioned line explicitly return 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.

CYC-03: Comparison with a literal boolean value

Type	Severity	Location
Gas Optimization	Informational	Cycle.sol L1

Description:

The contract has several occurrences of comparison with a literal boolean values of `true` or `false` that can be replaced replacing with compared expression itself to increase the legibility of the code.

Recommendation:

We advise to use the compared expression itself in place of expression's comparison with a boolean literal. The expression can be replaced as is when the expression is expected to evaluate to `true` and negation of expression can be used when the expression is expected to have `false` value.

CYC-04: Inefficient function implementation

Type	Severity	Location
Gas Optimization	Informational	Cycle.sol L517

Description:

The implementation of the function on the aforementioned line is inefficient as it redundantly checks `bool` value with an `if` statement and then returns it as is.

Recommendation:

We advise to remove the `if-else` block and directly return the expression `CycleMembersDeepIndexer[cycleId][depositor].exists;` from the function to have efficient implementation.

CYC-05: `updateCycleMember` is not restricted by `onlyStorageOracle`

Type	Severity	Location
Volatile Code	Critical	Cycle.sol L220

Description:

The function `updateCycleMember` on the aforementioned line updates a member within cycle and can be called by anyone while it should have been restricted to `onlyStorageOracle`.

Recommendation:

We advise to add `onlyStorageOracle` modifier in the function declaration so that only an allowed address could call this function keeping the integrity of the data.

```
function updateCycleMember(
        uint256 cycleId,
        address payable depositor,
        uint256 totalLiquidityAsPenalty,
        uint256 numberOfCycleStakes,
        uint256 stakesClaimed,
        bool hasWithdrawn
) external onlyStorageOracle  {...}
```

CYC-06: Incorrect code

Type	Severity	Location
Control Flow	Major	Cycle.sol L487

Description:

The aforementioned line calls the function `_getCycleIndex` to get the index of `CycleFinancial` but the call returns the index of `Cycle` corresponding to the `cycleId`.

Recommendation:

We advise to call the function `_getCycleFinancialIndex` to correctly get the index of `CycleFinancial`.

```
uint256 index = _getCycleFinancialIndex(cycleFinancial.cycleId);
```

CYC-07: Incorrect implementation of functions

Type	Severity	Location
Volatile Code	Major	Cycle.sol L378-L396

Description:

The functions `getRecordIndexForCycleMembersIndexerByDepositor` and `getRecordIndexForCycleMembersIndexer` on the aforementioned lines have incorrect implementation where `getRecordIndexForCycleMembersIndexerByDepositor` returns record index for cycle member while `getRecordIndexForCycleMembersIndexer` returns record index for cycle member by depositor.

Recommendation:

We advise to swap the implementations of both function so they returns record index from their corresponsing mappings.

```
function getRecordIndexForCycleMembersIndexerByDepositor(
        uint256 cycleId,
        uint256 recordIndexLocation
) external view returns (bool, uint256) {
  RecordIndex memory recordIndex
         = CycleMembersIndexerByDepositor[depositorAddress][recordIndexLocation];
        return (recordIndex.exists, recordIndex.index);
}
```

```
function getRecordIndexForCycleMembersIndexer(
        address depositorAddress,
        uint256 recordIndexLocation
) external view returns (bool, uint256) {
  RecordIndex memory recordIndex
         = CycleMembersIndexer[cycleId][recordIndexLocation];
        return (recordIndex.exists, recordIndex.index);
}
```

CRD-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ClientRecord.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

CRD-02: Inefficient local variable

Type	Severity	Location
Gas Optimization	Informational	ClientRecord.sol L18-L19

Description:

The local variable on the aforementioned line is inefficient as it copies the struct value into a memory variable and the it returns one property of the struct from the function.

Recommendation:

We recommend to directly return the value from the function instead of copying the struct into `memory` and then returning the property of it.

```
function doesClientRecordExist(address depositor)
        external
        view
        returns (bool)
{
  return ClientRecordIndexer[depositor].exists;
}
```

CRD-03: Comparison with a literal boolean value

Type	Severity	Location
Gas Optimization	Informational	ClientRecord.sol L1

Description:

The contract has several occurrences of comparison with a literal boolean values of `true` or `false` that can be replaced replacing with compared expression itself to increase the legibility of the code.

Recommendation:

We advise to use the compared expression itself in place of expression's comparison with a boolean literal. The expression can be replaced as is when the expression is expected to evaluate to `true` and negation of expression can be used when the expression is expected to have `false` value.

CRD-04: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	ClientRecord.sol L66-L74

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

CRD-05: Inefficient storage update

Type	Severity	Location
Gas Optimization	Informational	ClientRecord.sol L78-L84

Description:

The aforementioned lines update a struct inside array and upon each update it computes the localtion of struct inside array which is an inefficient approach.

Recommendation:

We advise to store a refrence to struct inside a variable of type `ClientRecord` pointing to storage and then update the properties of struct using this storage variable which is gas efficient compared to the current implementation.

```
ClientRecord storage clientRecord = ClientRecords[index];
clientRecord.underlyingTotalDeposits = underlyingTotalDeposits;
clientRecord.underlyingTotalWithdrawn = underlyingTotalWithdrawn;
clientRecord.derivativeBalance = derivativeBalance;
clientRecord.derivativeTotalDeposits = derivativeTotalDeposits;
clientRecord.derivativeTotalWithdrawn = derivativeTotalWithdrawn;
```

ICS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IClientRecordShema.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

ISS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ISavingsConfigSchema.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

ISS-02: Inefficient struct layout

Type	Severity	Location
Gas Optimization	Informational	ISavingsConfigSchema.sol L5, L9, L10

Description:

The properties `bool` and `enum` of struct on the aforementioned lines can be placed together to tight pack the struct.

Recommendation:

We recommend to place the properties of `bool` and `enum` on the aforementioned lines next to each other so that they can be packed within a single storage slot.

```
struct RuleSet {
        uint256 minimum;
        uint256 maximum;
        uint256 exact;
        bool applies;
        RuleDefinition ruleDefinition;
  bool exists;
}
```

SCG-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	SavingsConfig.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

SCG-02: Empty constructor declaration

Type	Severity	Location
Volatile Code	Informational	SavingsConfig.sol L12

Description:

An empty constructor is declared on the aforementioned line which is unnecessary.

Recommendation:

We advise to remove the empty constructor declaration on the aforementioned line to increase the quality of the code.

SCG-03: Comparison with a literal boolean value

Type	Severity	Location
Gas Optimization	Informational	SavingsConfig.sol L1

Description:

The contract has several occurrences of comparison with a literal boolean values of `true` or `false` that can be replaced replacing with compared expression itself to increase the legibility of the code.

Recommendation:

We advise to use the compared expression itself in place of expression's comparison with a boolean literal. The expression can be replaced as is when the expression is expected to evaluate to `true` and negation of expression can be used when the expression is expected to have `false` value.

SCG-04: Explicitly returning a local variable

Type	Severity	Location
Gas Optimization	Informational	SavingsConfig.sol L40, L119

Description:

The functions on the aforementioned line explicitly return 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.

SCG-05: Redundant Statements

Type	Severity	Location
Dead Code	Informational	SavingsConfig.sol L150

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.

SCG-06: `_validateRuleCreation` always reverts the transaction

Type	Severity	Location
Volatile Code	Major	SavingsConfig.sol L96, L134

Description:

The call to function `_validateRuleCreation` on the aforementioned line always reverts the transaction because the `exists` property of an exisiting Rule is set to `true` and the `_validateRuleCreation` asserts that the `exists` property should be `false`.

Recommendation:

We advise to set the property of `exists` to `false` so that `_validateRuleCreation` call does not revert the transaction and it successfully modify the Rule.

```
ruleSet.exists = false;
```

SCG-07: `modifyRule` does not save the Rule

Type	Severity	Location
Volatile Code	Major	SavingsConfig.sol L96

Description:

The function `modifyRule` does not save the new state of Rule in the storage making the transaction ineffecutal.

Recommendation:

We advise to add the call to `_saveRule` so that the update RuleSet is saved to storage of the contract.

```
_saveRule(ruleKey, ruleSet);
```

TRE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Treasury.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

TRE-02: Redundant `require` statement

Type	Severity	Location
Gas Optimization	Informational	Treasury.sol L26-L29

Description:

The `require` statement on the aforementioned line is redundant as the same check is performed by the `require` statement on `L24`.

Recommendation:

We advise to remove the redundant `require` statement on the aforementioned line from the function.

TRE-03: Redundant `require` statement

Type	Severity	Location
Gas Optimization	Informational	Treasury.sol L58-L61

Description:

The `require` statement on the aforementioned line is redundant as the same check is performed by the `require` statement on `L56`.

Recommendation:

We advise to remove the redundant `require` statement on the aforementioned line from the function.

TRE-04: Comparison with a literal boolean value

Type	Severity	Location
Gas Optimization	Informational	Treasury.sol L1

Description:

The contract has several occurrences of comparison with a literal boolean values of `true` or `false` that can be replaced replacing with compared expression itself to increase the legibility of the code.

Recommendation:

We advise to use the compared expression itself in place of expression's comparison with a boolean literal. The expression can be replaced as is when the expression is expected to evaluate to `true` and negation of expression can be used when the expression is expected to have `false` value.

TRE-05: `enum` type is declared but never used

Type	Severity	Location
Dead Code	Informational	Treasury.sol L16

Description:

The `enum` type `DepositType` is never used in the code and can be removed from the contract.

Recommendation:

We advise to remove the `enum` type declared on the aforementioned line to increase the quality of the code.

XFI-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	XendFinanceIndividual_Yearn_V1.sol L1

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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

XFI-02: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	XendFinanceIndividual_Yearn_V1.sol L47

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

XFI-03: Redundant storage variables

Type	Severity	Location
Gas Optimization	Informational	XendFinanceIndividual_Yearn_V1.sol L50-L51

Description:

The storage variables of `TreasuryAddress` and `TokenAddress` are redundant as its values are also stored in storage variables of `treasury` and `daiToken`.

Recommendation:

We advise to remove the redundant storage variables from the aforementioned lines and storage variables of `treasury` and `daiToken` be used in place of them.

XFI-04: Comparison with a literal boolean value

Type	Severity	Location
Gas Optimization	Informational	XendFinanceIndividual_Yearn_V1.sol L1

Description:

The contract has several occurrences of comparison with a literal boolean values of `true` or `false` that can be replaced replacing with compared expression itself to increase the legibility of the code.

Recommendation:

We advise to use the compared expression itself in place of expression's comparison with a boolean literal. The expression can be replaced as is when the expression is expected to evaluate to `true` and negation of expression can be used when the expression is expected to have `false` value.

XFI-05: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	XendFinanceIndividual_Yearn_V1.sol L86

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of SafeERC20 library.

XFI-06: Inefficient local variable

Type	Severity	Location
Gas Optimization	Informational	XendFinanceIndividual_Yearn_V1.sol L357

Description:

The aforementioned line declares a local variable which is used only once in the function and hence it is inefficient to declare and use it.

Recommendation:

We advise to use the initialization part of local variable declaration directly at the place where it is used.

```
_deposit(msg.sender);
```

XFI-07: Function does not return a value

Type	Severity	Location
Logical Issue	Minor	XendFinanceIndividual_Yearn_V1.sol L285

Description:

The function on the aforementioned lines specifies a `uint256` as a return type in its signature yet no value is returned from the body of the function.

Recommendation:

We advise to remove return type of `uint256` from the signature of the function as the function does not need to return a value.

XFI-08: Unused local variables

Type	Severity	Location
Dead Code	Informational	XendFinanceIndividual_Yearn_V1.sol L316, L317, L337, L338

Description:

The local variables on the aforementioned lines are declared to store the values from the returned tuple yet these local variables are never used within the code.

Recommendation:

We advise to remove the declaration of the local variables on the aforementioned lines as they are never used in the code.

XFI-09: Incorrect code

Type	Severity	Location
Logical Issue	Critical	XendFinanceIndividual_Yearn_V1.sol L446-L455

Description:

The aforementioned lines add the deposit amounts to `record` the second time as the amounts are already added when the struct is initialized on `L36`.

Recommendation:

We advise to remove the aforementioned lines so that the amounts are not added twice to the `record` variable of struct type.

XFI-10: Incorrect value provided for struct property

Type	Severity	Location
Logical Issue	Critical	XendFinanceIndividual_Yearn_V1.sol L440

Description:

The value of `underlyingAmountDeposited` is used for the initialization of struct property `underlyingTotalWithdrawn` which is incorrect as the property `underlyingTotalWithdrawn` should be initialized with `0` when the record is new.

Recommendation:

We advise to pass the literal `0` on the aforementioned line to correctly initialize the struct property of `underlyingTotalWithdrawn`.

```
ClientRecord memory record = ClientRecord(
                true,
                client,
                underlyingAmountDeposited,
                0,
                derivativeAmountDeposited,
                derivativeAmountDeposited,
                0
);
```

XFI-11: Incorrect code

Type	Severity	Location
Logical Issue	Critical	XendFinanceIndividual_Yearn_V1.sol L486

Description:

The aforementioned line adds `derivativeAmountWithdrawn` to `record.derivativeTotalDeposits` which is incorrect as the function is called after withdrawal and not after deposit.

Recommendation:

We recommend to use the struct property of `derivativeTotalWithdrawn` to correctly update the record with the amount of derivative that is withdrawn.

```
record.derivativeTotalWithdrawn = record.derivativeTotalWithdrawn.add(
            derivativeAmountWithdrawn
);
```

XFI-12: Incorrect code

Type	Severity	Location
Logical Issue	Critical	XendFinanceIndividual_Yearn_V1.sol L489

Description:

The aforementioned line adds `derivativeAmountWithdrawn` to `record.derivativeBalance` which is incorrect as the function is called after withdrawal and not after deposit.

Recommendation:

We recommend to subtract `derivativeAmountWithdrawn` from `record.derivativeBalance` as the derivative balance is decreased after withdrawal.

```
record.derivativeBalance = record.derivativeBalance.sub(
            derivativeAmountWithdrawn
);
```

XFI-13: Possibility of re-entrancy attack

Type	Severity	Location
Control Flow	Medium	XendFinanceIndividual_Yearn_V1.sol L258

Description:

The `transfer` function call on the aforementioned has possibility of re-entrancy if the `transfer` function of the called contract is compromised. The re-entrancy will allow the draining of funds from the contract as the record are updated only after the `transfer` call.

Recommendation:

We advise to either move the `transfer` call at the end of function or make the function non-reentrant by inheriting from Openzeppelin's `ReentrancyGuard` contract and using the modifier `nonReentrant`.

```
https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/utils/ReentrancyGuard.sol
```

XFG-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	XendFinanceGroup_Yearn_V1.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 `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

XFG-02: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	XendFinanceGroup_Yearn_V1.sol L96

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

XFG-03: Comparison with a literal boolean value

Type	Severity	Location
Gas Optimization	Informational	XendFinanceGroup_Yearn_V1.sol L1

Description:

The contract has several occurrences of comparison with a literal boolean values of `true` or `false` that can be replaced replacing with compared expression itself to increase the legibility of the code.

Recommendation:

We advise to use the compared expression itself in place of expression's comparison with a boolean literal. The expression can be replaced as is when the expression is expected to evaluate to `true` and negation of expression can be used when the expression is expected to have `false` value.

XFG-04: Unnecessary local variables

Type	Severity	Location
Gas Optimization	Informational	XendFinanceGroup_Yearn_V1.sol L108-L113

Description:

The local variables declarations on the aforementioned lines are unnecessary as the function call on `L115` can directly utilize the properties of struct variable `group` for arguments.

Recommendation:

We advise to remove the redundant local variable declarations on the aforementioned lines.

XFG-05: Explicitly returning a local variable

Type	Severity	Location
Gas Optimization	Informational	XendFinanceGroup_Yearn_V1.sol L118, L134, L154, L294, L304, L343, L382, L409, L448, L716, L643

Description:

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

Recommendation:

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

XFG-06: Unnecessary local variables

Type	Severity	Location
Gas Optimization	Informational	XendFinanceGroup_Yearn_V1.sol L255-L269

Description:

The local variables on the aforementioned lines are unnecessary as the function call on `L170` can directly utilize the properties of struct variable `cycleMember` as arguments.

Recommendation:

We advise to remove the local variables declarations on the aforementioned lines as they are redundant.

XFG-07: Function does return a value

Type	Severity	Location
Dead Code	Minor	XendFinanceGroup_Yearn_V1.sol L465

Description:

The function on the aforementioned line specifies `CycleMember` as a return type yet the body of the function does not return any value.

Recommendation:

We recommend to remove the `CycleMember` as return type from the signature of the function as the function does not need to return it.

XFG-08: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	XendFinanceGroup_Yearn_V1.sol L1189

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of SafeERC20 library.

XFG-09: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	XendFinanceGroup_Yearn_V1.sol L795-L796

Description:

The `if-else` block on the aforementioned lines is inefficient as it explicitly return boolean literal depending the evaluation of the predicate.

Recommendation:

We advise to directly return the predicate expression for the efficient implementation of the code.

```
return currentTimeStamp >= cycleEndTimeStamp;
```

XFG-10: Unsafe subtraction

Type	Severity	Location
Mathematical Operations	Minor	XendFinanceGroup_Yearn_V1.sol L763

Description:

The aforementioned line performs unsafe subtraction which can be result in underflow of integer value.

Recommendation:

We advise to use `sub` function from `SafeMath` library to perform subtraction so that the transaction is reverted if underflow happens.

```
uint256 derivativeBalanceToWithdraw = cycleFinancial.derivativeBalance.sub(
            cycleFinancial.derivativeBalanceClaimedBeforeMaturity
);
```

XFG-11: Unnecessary parenthesis around expressions

Type	Severity	Location
Language Specific	Informational	XendFinanceGroup_Yearn_V1.sol L819, L825

Description:

The expressions on the aforementioned lines have unnecessary parenthesis around them.

Recommendation:

We advise to remove the parenthesis around expressions on the aforementioned lines to increase the legibility of the codebase.

XFG-12: Confusing modifier name

Type	Severity	Location
Inconsistency	Informational	XendFinanceGroup_Yearn_V1.sol L816

Description:

The modifier `onlyCycleCreator` 's name suggests that it only allows the creator of cycle to execute function guarded by this modifier yet the implementation of the modifier suggests that it also allows the cycle member to execute function guarded by the modifier.

Recommendation:

We advise to change the name of modifier to `onlyCycleCreatorOrMember` to suggest that it also allows cycle member in addition to cycle creator to execute function guarded by the modifier.

XFG-13: Unused local variables

Type	Severity	Location
Gas Optimization	Informational	XendFinanceGroup_Yearn_V1.sol L1276, L1277, L1255, L1256, L1301, L1302

Description:

The local variables on the aforementioned lines are declared to store the values from the returned tuple yet these local variables are never used within the code.

Recommendation:

We advise to remove the declaration of the local variables on the aforementioned lines as they are never used in the code.

XFG-14: Unsafe subtraction

Type	Severity	Location
Mathematical Operations	Minor	XendFinanceGroup_Yearn_V1.sol L951

Description:

The aforementioned line performs unsafe subtraction which can be result in underflow of integer value.

Recommendation:

We advise to use `sub` function from `SafeMath` library to perform subtraction so that the transaction is reverted if underflow happens.

```
underlyingAmountThatMemberDepositIsWorth =
underlyingAmountThatMemberDepositIsWorth.sub(totalDeductible);
```

XFG-15: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	XendFinanceGroup_Yearn_V1.sol L977

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of SafeERC20 library.

XFG-16: Ineffectual code

Type	Severity	Location
Gas Optimization	Informational	XendFinanceGroup_Yearn_V1.sol L976

Description:

The `if` statement on the aforementioned line will never evaluate to `false` as the same condition is checked in a `require` statement on `L973` and the control flow reaches to `if` statement only after the condition in `require` statement evaluates to `true`.

Recommendation:

We advise to remove the `if` condition on the aforementioned line as it is redundant.

XFG-17: Anyone can make a particular depositor join cycle

Type	Severity	Location
Volatile Code	Major	XendFinanceGroup_Yearn_V1.sol L1713

Description:

The function `joinCycleDelegate` on the aforementioned line allows anyone to call it and make `depositorAddress` join the cycle if it has approved sufficient tokens amount to the contract.

Recommendation:

We advise to remove this function as only the depositing address should be allowed to call the function.

XFG-18: Possibility of reentrancy attack

Type	Severity	Location
Control Flow	Medium	XendFinanceGroup_Yearn_V1.sol L977

Description:

The `transfer` function call on the aforementioned line will allow reentrancy into the contract if the `transfer` function of the called contract is compromised leading draining of funds as the cycle, cycleMember and cycleFinancials are updated after the `transfer` call.

Recommendation:

We advise to move the `transfer` call at the end of function execution so reentrancy would not allow draining of funds or alternatively the function can be non-reentrant by inherting the contract from Openzeppelin's `ReentrancyGuard` contract and using the `nonReentrant` modifier on the function.

```
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol
```

XFG-19: Possibility of reentrancy attack

Type	Severity	Location
Control Flow	Medium	XendFinanceGroup_Yearn_V1.sol L1150

Description:

The `transfer` call on the aforementioned line will allow reentrancy into the contract if the `transfer` function of the called contract is compromised. This will lead draining of funds as the cycle related storage variables are updated after the call to `transfer`.

Recommendation:

We advise either to move `transfer` call at the end of function or make the function non-Reentrant by inherting the contract from Openzeppelin's `ReentrancyGuard` contract and use modfier `nonReentrant` with the function.

```
https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/utils/ReentrancyGuard.sol
```

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.