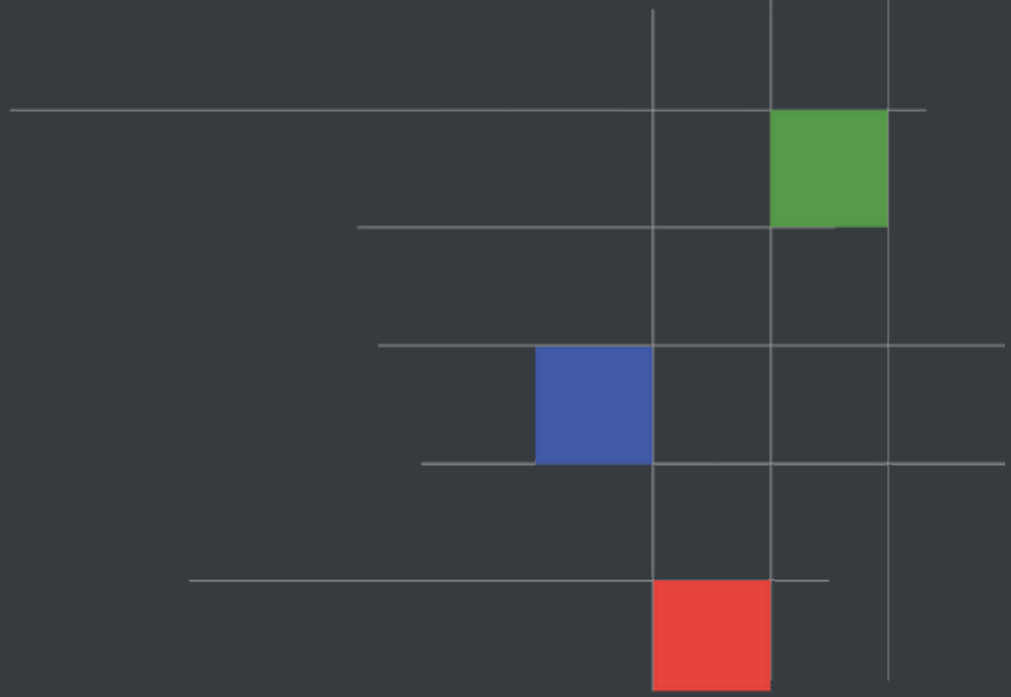
Xend.Finance: Esusu

Smart Contracts

Security Assessment

January 27th, 2021



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: Esusu
Description	The project comprise of Esusu service where user deposits are staked on yearn.finance to earn interest.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 2242488840d3a07a71ad2ae667425f8ae391a808 2. 8e97179757a2f97ce4db71b8ffe5f3f31a045b5a 3. 9d86eff2366ea6815ccc59f3b1c8553f39887d93

Audit Summary

Delivery Date	January 27th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	November 6th, 2020 - January 27th, 2021

Vulnerability Summary

Total Issues	68
Total Critical	4
Total Major	1
Total Medium	0
Total Minor	4
Total Informational	59

Executive Summary

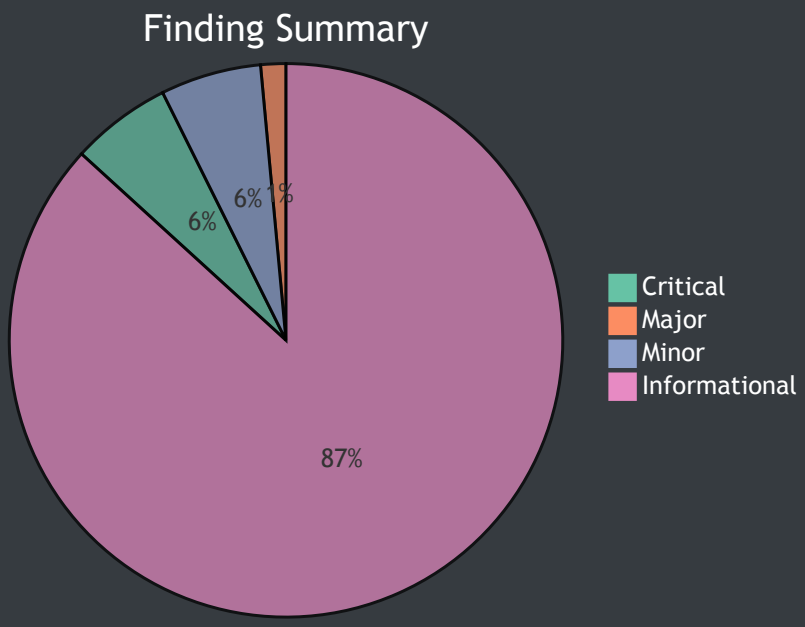
This report represents the results of CertiK's engagement with Xend on their implementation of the Esusu smart contracts.

Our findings mainly refer to optimizations and a couple of major issues. All of the findings except a few informational were remediated. The overall security of the contracts can be deemed as high after the remediations were applied.

Files In Scope

ID	Contract	Location
EAR	EsusuAdapter.sol	EsusuAdapter.sol
ESE	EsusuService.sol	EsusuService.sol
ESG	EsusuStorage.sol	EsusuStorage.sol
EAW	EsusuAdapterWithdrawalDelegate.sol	EsusuAdapterWithdrawalDelegate.sol
IEA	IEsusuAdapter.sol	IEsusuAdapter.sol
IES	IEsusuService.sol	IEsusuService.sol
IES	IEsusuStorage.sol	IEsusuStorage.sol
IRC	IRewardConfig.sol	IRewardConfig.sol
OWN	Ownable.sol	Ownable.sol
OSE	OwnableService.sol	OwnableService.sol
RCG	RewardConfig.sol	RewardConfig.sol

Findings



ID	Title	Type	Severity	Resolved
ESG-01	Inefficient `storage` layout	Gas Optimization	Informational	✓
ESG-02	Incorrect Grammar	Language Specific	Informational	✓
ESG-03	Unused declared data-structure	Coding Style	Informational	✓
ESG-04	Redundant Variable Initialization	Coding Style	Informational	✓
ESG-05	Redundant Statements	Dead Code	Informational	✓

ESG-06	Inefficient `mapping` declerations	Gas Optimization	Informational	⟳
ESG-07	Inefficient `storage` data initialization	Gas Optimization	Informational	✓
ESG-08	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
ESG-09	`require` statement can be substituted with a `modifier`	Gas Optimization	Informational	✓
ESG-10	Inefficiant local variable decleration	Gas Optimization	Informational	✓
ESG-11	Inefficient code	Gas Optimization	Informational	✓
ESG-12	Inefficient code	Gas Optimization	Informational	✓
ESG-13	Inefficient `storage` access	Gas Optimization	Informational	✓
ESG-14	Ineffectual code	Dead Code	Informational	✓
ESG-15	Inefficient `storage` struct values assignment	Gas Optimization	Informational	✓
ESG-16	Explicitly returning local variable	Gas Optimization	Informational	✓
ESG-17	Unlocked Compiler Version	Language Specific	Informational	✓
ESG-18	Usage of alias `uint` instead of complete form `uint256`	Coding Style	Informational	✓
ESE-01	Usage of alias `uint` instead of complete form `uint256`	Language Specific	Informational	✓
ESE-02	Anyone can deposit funds on an address's behalf	Logical Issue	Major	✓
ESE-03	Incorrect code	Compiler Error	Critical	✓

ESE-04	Incorrect code	Compiler Error	Critical	✓
ESE-05	Unlocked Compiler Version	Language Specific	Informational	✓
ESE-06	Redundant Statements	Dead Code	Informational	✓
OSE-01	`if` block should be substitued with `require` statement	Coding Style	Minor	✓
OSE-02	`if` block can be substituted with a `require` statement	Coding Style	Minor	✓
OSE-03	Unlocked Compiler Version	Language Specific	Informational	✓
OSE-04	Incorrect contract name	Compiler Error	Critical	✓
EAR-01	Redundant Statements	Dead Code	Informational	✓
EAR-02	Non-idiomatic location `library` import in the contract	Language Specific	Informational	✓
EAR-03	Redundant `storage` variable	Gas Optimization	Informational	✓
EAR-04	Mutability Specifiers Missing	Gas Optimization	Informational	✓
EAR-05	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
EAR-06	Comparison with boolean literal	Gas Optimization	Informational	✓
EAR-07	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	✓
EAR-08	Incorrect spelling	Coding Style	Informational	✓
EAR-09	Inefficient code	Gas Optimization	Informational	✓

EAR-10	Inefficient code	Gas Optimization	Informational	✓
EAR-11	Unused local variable	Gas Optimization	Informational	✓
EAR-12	Incorrect order of functions	Language Specific	Informational	✓
EAR-13	Unlocked Compiler Version	Language Specific	Informational	✓
EAR-14	Usage of alias `uint` instead of complete form `uint256`	Language Specific	Informational	✓
EAR-15	Function visibility can be changed to `external`	Language Specific	Informational	✓
EAW-01	Redundant Statements	Dead Code	Informational	⟳
EAW-02	The `require` call does not have a `reason` string	Language Specific	Informational	✓
EAW-03	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	✓
EAW-04	Incorrect code implementation	Volatile Code	Critical	✓
EAW-05	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
EAW-06	Comparison with boolean literal	Language Specific	Informational	✓
EAW-07	Inefficient code	Gas Optimization	Informational	✓
EAW-08	Inefficient code	Gas Optimization	Informational	✓
EAW-09	Inefficient code	Gas Optimization	Informational	✓
EAW-10	Usage of alias `uint` instead of complete form `uint256`	Language Specific	Informational	✓

EAW-11	Function visibility can be changed to `external`	Gas Optimization	Informational	✓
EAW-12	Unlocked Compiler Version	Language Specific	Informational	✓
RCG-01	Mutability Specifiers Missing	Gas Optimization	Informational	✓
RCG-02	Mutability Specifiers Missing	Gas Optimization	Informational	⟳
RCG-03	Redundant Statements	Dead Code	Informational	✓
RCG-04	Comparison with boolean literal	Language Specific	Informational	✓
RCG-05	Explicitly returning a local variable from function	Gas Optimization	Informational	✓
RCG-06	Incorrect Spelling	Coding Style	Informational	✓
RCG-07	Incorrect order of functions	Language Specific	Informational	✓
RCG-08	Usage of alias `uint` instead of complete form `uint256`	Language Specific	Informational	✓
RCG-09	Unlocked Compiler Version	Language Specific	Informational	✓
IEA-01	Unlocked Compiler Version	Language Specific	Informational	✓
IES-01	Unlocked Compiler Version	Language Specific	Informational	✓
IES-02	Unlocked Compiler Version	Language Specific	Informational	✓
IRC-01	Unlocked Compiler Version	Language Specific	Informational	✓

ESG-01: Inefficient `storage` layout

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

Description:

The `struct` `EsusuCycle` 's declaration is inefficient as the `struct's` properties on the aforementioned lines can be placed together to tight pack the data to use only one `storage` slot.

Recommendation:

We recommend to place the `struct` properties on the aforementioned lines next to each other to tight pack the `storage` slot.

```
struct EsusuCycle {
   address Owner;
   CurrencyEnum Currency;
   CycleStateEnum CycleState;
}
```

Alleviation:

Alleviation were applied as advised.

ESG-02: Incorrect Grammar

Type	Severity	Location
Language Specific	Informational	EsusuStorage.sol L37

Description:

The comment on the aforementioned line is grammatically incorrect.

Recommendation:

We recommend to recitify the comment on the aforementioned line.

```
//  Time, when the cycle starts has elapsed. Anyone can start cycle after this time has elapsed
```

Alleviation:

Alleviation were applied as advised.

ESG-03: Unused declared data-structure

Type	Severity	Location
Coding Style	Informational	EsusuStorage.sol L42-L46

Description:

The struct `Member` declared on the aforementioned lines is never used in the code.

Recommendation:

We recommend to remove the struct `Member` on the aforementioned lines to increase the legibility of the codebase as the struct is never used in the code.

Alleviation:

Alleviation were applied as advised.

ESG-04: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	EsusuStorage.sol L66

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.

Alleviation:

Alleviation were applied as advised.

ESG-05: Redundant Statements

Type	Severity	Location
Dead Code	Informational	EsusuStorage.sol L56, L64

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:

Alleviation were applied as advised.

ESG-06: Inefficient `mapping` declerations

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L69, L74-L78

Description:

The mapping declerations on the aforementioned lines is inefficient and can be converted into a single mapping that utilizes a struct to reperesnt data all across all mappings.

Recommendation:

We recommend to use struct to represent data across mappings on the aforementioned lines as all of the mappings have `cycleId` as their key.

```
struct CycleData {
   EsusuCycle esusuCycle;
   position mapping(address=>uint);
   beneficiary mapping(address=> uint);
   withdrawnCapital mapping(address=> uint);
}

mapping(uint256 => CycleData) public cyclesData;
```

Alleviation:

No alleviations.

ESG-07: Inefficient `storage` data initialization

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L107-L120

Description:

The code on the aforementioned initialize `storage` data in an inefficient way by accessing struct through mapping each time when it updates the property on struct.

Recommendation:

We recommend to introduce a local `storage` variable of type `EsusuCycle` and then update properties through this local variable to save gas cost associated with redundant mapping lookup for struct access.

```
EsusuCycle storage cycle = EsusuCycleMapping[EsusuCycleId];
cycle.[property] = [value];
```

Alleviation:

Alleviation were applied as advised.

ESG-08: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L129, L144, L156, L165, L175, L184, L193, L202, L211, L219, L258, L267, L318, L248

Description:

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

Recommendation:

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

Alleviation:

Alleviation were applied as advised.

ESG-09: `require` statement can be substituted with a `modifier`

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L129, L144, L156, L165, L175, L184, L193, L202, L211, L219, L258, L267, L318

Description:

The `require` statements on the aforementioned line can substituted with a modifier to increase the legibility of the code.

Recommendation:

We recommend to introduce a modifier.

```
modifier isCycleIdValid(uint256 esusuCycleId) {
   require(esusuCycleId > 0 && esusuCycleId <= EsusuCycleId, "Cycle ID must be within
valid EsusuCycleId range");
}
```

The modfier can be used across all functions like so.

```
function funcName(uint256 esusuCycleId) isCycleIdValid(esusuCycleId) {...}
```

Alleviation:

Alleviation were applied as advised.

ESG-10: Inefficient local variable declaration

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L158, L167, L177, L186, L195, L204, L213, L221

Description:

The variable declerations on the aforementioned lines are inefficient as whole struct is copied to `memory` yet only single property is read from it.

Recommendation:

We recommend to directly return the struct properties from functions instead of storing them in a local variable.

```
    return EsusuCycleMapping[esusuCycleId].propertyName;
```

Alleviation:

Alleviation were applied as advised.

ESG-11: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L225, L234

Description:

The functions on the aforementioned lines make use of inefficient local variables which are only utilized once in the function's code.

Recommendation:

We recommend to directly utilize the mapping value instead of making use of local variables to save gas cost associated with extra local variables decelrations.

Alleviation:

Alleviation were applied as advised.

ESG-12: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L243

Description:

The function on the aforementioned line contains inefficient code which makes use of inefficient local variable and inefficient `if-else` block.

Recommendation:

We recommend to optimize the code in the aforementioned function.

```
function IsMemberInCycle(address memberAddress,uint esusuCycleId ) external view
returns(bool){
   return MemberAddressToMemberCycleMapping[memberAddress][esusuCycleId].CycleId > 0;
}
```

Alleviation:

Alleviation were applied as advised.

ESG-13: Inefficient `storage` access

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L260-L262

Description:

The aforementioned lines access same `storage` slot twice which is an inefficient implementation.

Recommendation:

We advise to use a local variable to return the value instead of directly reading from the `storage` .

```
uint256 amount = EsusuCycleMapping[esusuCycleId].TotalAmountDeposited.add(amount);
EsusuCycleMapping[esusuCycleId].TotalAmountDeposited = amount;
return amount;
```

Alleviation:

Alleviation were applied as advised.

ESG-14: Ineffectual code

Type	Severity	Location
Dead Code	Informational	EsusuStorage.sol L276

Description:

The aforementioned line add `0` to a `storage` variable which does not result in change of value of the variable and hence the line can be considered a dead code.

Recommendation:

We recommend to remove the aforementioned line from the code.

Alleviation:

Alleviation were applied as advised.

Alleviation:
Alleviation were applied as advised.

ESG-15: Inefficient `storage` struct values assignment

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L304-L307

Description:

The aforementioned lines update the properties of struct by accessing it through mapping. Each line perform mapping lookup operation to retrieve struct which inefficient.

Recommendation:

We recommend to use a local `storage` variable pointing to struct and update properties directly on it to save gas cost associated with redundant mapping lookups.

```
EsusuCycle storage cycle = EsusuCycleMapping[esusuCycleIdi];
cycle.property = value;
```

Alleviation:

Alleviation were applied as advised.

ESG-16: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	EsusuStorage.sol L356, L362-L364

Description:

The function `CalculateMemberWithdrawalTime` 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 localvariable declaration and explicit return statement in order to reduce the overall cost of gas.

```
function CalculateMemberWithdrawalTime(uint cycleId, address member) external view
returns(uint withdrawalTime){

        mapping(address=>uint) storage memberPositionMapping =
CycleToMemberPositionMapping[cycleId];

        uint memberPosition = memberPositionMapping[member];

        withdrawalTime =
(EsusuCycleMapping[cycleId].CycleStartTime.add(memberPosition.mul(EsusuCycleMapping[cyc
leId].PayoutIntervalSeconds)));
}
```

Alleviation:

Alleviation were applied as advised.

ESG-17: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EsusuStorage.sol L2

Description:

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

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Alleviation were applied as advised.

ESG-18: Usage of alias `uint` instead of complete form `uint256`

Type	Severity	Location
Coding Style	Informational	EsusuStorage.sol L1

Description:

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

Recommendation:

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

Alleviation:

Alleviation were applied as advised.

ESE-01: Usage of alias `uint` instead of complete form `uint256`

Type	Severity	Location
Language Specific	Informational	EsusuService.sol L1

Description:

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

Recommendation:

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

Alleviation:

Alleviation were applied as advised.

ESE-02: Anyone can deposit funds on an address's behalf

Type	Severity	Location
Logical Issue	Major	EsusuService.sol L57

Description:

The call into Esusu Adapter on the aforementioned line passes `member` as second argument to the function `JoinEsusu` which later utilizes funds from `member` using `transferFrom` enabling a random address moving funds for the `member` if there is sufficient amount is approved to the Esusu Adapter contract.

Recommendation:

We advise to use `msg.sender` instead of `member` parameter so a user can only deposit funds of his own and not of others to safeguard against unintended deposits from users.

```
    _esusuAdapter.JoinEsusu(esusuCycleId, msg.sender);
```

Alleviation:

Alleviation were applied as advised.

ESE-03: Incorrect code

Type	Severity	Location
Compiler Error	Critical	EsusuService.sol L92-L94

Description:

The function on the aforementioned lines expects a return value of `bool` type but no value is returned from the body of the function.

Recommendation:

We recommend to correct the code and prepend the keyword `return` on `L93`.

```
function IsMemberEligibleToWithdrawROI(uint esusuCycleId, address member) external view
returns(bool){
   return
_esusuAdapterWithdrawalDelegate.IsMemberEligibleToWithdrawROI(esusuCycleId,member);
}
```

Alleviation:

Alleviation were applied as advised.

ESE-04: Incorrect code

Type	Severity	Location
Compiler Error	Critical	EsusuService.sol L96-L98

Description:

The function on the aforementioned lines expects a return value of type `bool` yet no value is returned from the body of the function.

Recommendation:

We advise to prepend the keywork `return` on `L97` to rectify the code.

```
function IsMemberEligibleToWithdrawCapital(uint esusuCycleId, address member) external
view returns(bool){
   return
_esusuAdapterWithdrawalDelegate.IsMemberEligibleToWithdrawCapital(esusuCycleId,member);
}
```

Alleviation:

Alleviation were applied as advised.

ESE-05: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EsusuService.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;
```

Alleviation:

Alleviation were applied as advised.

ESE-06: Redundant Statements

Type	Severity	Location
Dead Code	Informational	EsusuService.sol L13

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:

Alleviation were applied as advised.

OSE-01: `if` block should be substitued with `require` statement

Type	Severity	Location
Coding Style	Minor	OwnableService.sol L32-L34

Description:

The `if` block on the aforementioned evaluates to `false` when an zero address is provided yet the transaction executes successfully without setting a new owner.

Recommendation:

We advise to use a `require` statement such that the transaction reverts when an zero address is provided to increase the legibility of the code.

```
    require(newOwner != address(0), "address cannot be zero");
    owner = newOwner;
```

Alleviation:

Alleviation were applied as advised.

OSE-02: `if` block can be substituted with a `require` statement

Type	Severity	Location
Coding Style	Minor	OwnableService.sol L41-L43

Description:

The `if` block on the aforementioned line evaluates to `false` when an zero address is provided yet the transaction executes successfully with setting a new contract owner.

Recommendation:

We recommend to use a `require` function such that the transaction reverts when an zero address is provided to increase the legibility of the code.

```
    require(newServiceContract != address(0), "address cannot be zero");
```

Alleviation:

Alleviation were applied as advised.

OSE-03: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	[OwnableService.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;
```

Alleviation:

Alleviation were applied as advised.

OSE-04: Incorrect contract name

Type	Severity	Location
Compiler Error	Critical	OwnableService.sol L9

Description:

The contract name on the aforementioned line is specified as `Ownable` while the filename and as well the usage of this contract at several places in the codebase expects its name to be `OwnableService`.

Recommendation:

We advise to change the name of the contract from `Ownable` to `OwnableService` to correctly compile the contracts codebase.

Alleviation:

Alleviation were applied as advised.

EAR-01: Redundant Statements

Type	Severity	Location
Dead Code	Informational	EsusuAdapter.sol L79, L86

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:

Alleviation were applied as advised partly.

EAR-02: Non-idiomatic location `library` import in the contract

Type	Severity	Location
Language Specific	Informational	EsusuAdapter.sol L96

Description:

The statement for linking of `SafeMath` library to `uint256` on the aforementioned line has non-idiomatic location in the contract.

Recommendation:

We adivse to move the aforementioned line at the start of contract's body to increase the quality of the codebase.

```
contract EsusuAdapter is OwnableService, ISavingsConfigSchema {
   using SafeMath for uint256;
   ...
}
```

Alleviation:

Alleviation were applied as advised.

EAR-03: Redundant `storage` variable

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapter.sol L85

Description:

The purpose of the storage variable `_owner` on the aformentioned line can be served by the storage `owner` inherited from the contract `OwnableService`.

Recommendation:

We recommend to remove the `_owner` from the aforementioned line and its usage in the code be replaced by the storage variable `owner`.

Alleviation:

Alleviation were applied as advised.

EAR-04: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapter.sol L87, L90, L91, L92

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:

Alleviation were applied as advised.

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

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapter.sol L154, L225, L325

Description:

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

Recommendation:

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

Alleviation:

Alleviation were applied as advised partly.

EAR-06: Comparison with boolean literal

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapter.sol L168, L409

Description:

The aforementioned lines perform comparison with boolean literal as predicate of `if` statements.

Recommendation:

We recommend to use the boolean expressions directly as predicate of `if` statements instead of comparing them with boolean literals.

Alleviation:

Alleviation were applied as advised.

EAR-07: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	EsusuAdapter.sol L178, L382, L394

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.

Alleviation:

Alleviation were applied as advised.

EAR-08: Incorrect spelling

Type	Severity	Location
Coding Style	Informational	EsusuAdapter.sol L200

Description:

The comment on the aforementioned line has incorrect spelling for the word `multiply`.

Recommendation:

We recommend to correct the word spellings on the aforementioned line.

Alleviation:

Alleviation were applied as advised.

EAR-09: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapter.sol L319

Description:

The function on the aforementioned line has inefficient code in its body which can be optimized to save gas cost associated with its execution.

Recommendation:

We recommend to use the following code in the function's body to have optimized code.

```
function _isMemberABeneficiaryInCycle(address memberAddress,uint esusuCycleId )
internal view returns(bool){
   return _esusuStorage.GetMemberCycleToBeneficiaryMapping(esusuCycleId, memberAddress)
> 0;
}
```

Alleviation:

Alleviation were applied as advised.

EAR-10: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapter.sol L333

Description:

The function on the aforementioned line has inefficient code in its body which can be optimized to save gas cost associated with its execution.

Recommendation:

We advise to use the following code in the function body to optimize it.

```
function _isMemberInWithdrawnCapitalMapping(address memberAddress,uint esusuCycleId )
internal view returns(bool){
   return _esusuStorage.GetMemberWithdrawnCapitalInEsusuCycle(esusuCycleId,
memberAddress) > 0;
}
```

Alleviation:

Alleviation were applied as advised.

EAR-11: Unused local variable

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapter.sol L353

Description:

The local variable `exists` is assigned the first value from the tuple returned from the function. This local variable is never used in the code and hence its decleration can be omitted.

Recommendation:

We recommend to skip the local variable `exists` declaration on the aforementioned line to save gas cost associated with it.

```
    (, uint index ) = _groupsContract.getGroupIndexerByName(name);
```

Alleviation:

Alleviation were applied as advised.

EAR-12: Incorrect order of functions

Type	Severity	Location
Language Specific	Informational	EsusuAdapter.sol

Description:

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

Recommendation:

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

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

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

Alleviation:

Alleviation were applied as advised.

EAR-13: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EsusuAdapter.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;
```

Alleviation:

Alleviation were applied as advised.

EAR-14: Usage of alias `uint` instead of complete form `uint256`

Type	Severity	Location
Language Specific	Informational	EsusuAdapter.sol L1

Description:

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

Recommendation:

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

Alleviation:

Alleviation were applied as advised.

EAR-15: Function visibility can be changed to `external`

Type	Severity	Location
Language Specific	Informational	EsusuAdapter.sol L122, L149, L208, L274, L279, L302, L309, L350, L374

Description:

The functions on the aforementioned lines can have their visibilities changed from `public` to `external` as they are never called from within the contract.

Recommendation:

We recommend to changed the visibilites of the functions on the aforementioned lines from `public` to `external` .

Alleviation:

Alleviation were applied as advised.

EAW-01: Missing mutability specifier

Type	Severity	Location
Dead Code	Informational	EsusuAdapterWithdrawalDelegate.sol L54-L63

Description:

The linked lines do not have mutability specificier for the variables declared..

Recommendation:

We advise that the mutability specifier is added for the linked variable declarations.

Alleviation:

No alleviations.

EAW-02: The `require` call does not have a `reason` string

Type	Severity	Location
Language Specific	Informational	EsusuAdapterWithdrawalDelegate.sol L105

Description:

The `require` call on the aforementioned line does not have a `reason` string.

Recommendation:

We advise to add the `reason` string in the `require` call on the aforementioned line to increase the legibility of the code.

```
require(_isMemberEligibleToWithdrawCapital(esusuCycleId,member), "member is not eligible to withdraw");
```

Alleviation:

Alleviation were applied as advised.

EAW-03: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	EsusuAdapterWithdrawalDelegate.sol L127, L171, L290, L321

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.

Alleviation:

Alleviation were applied as advised.

EAW-04: Incorrect code implementation

Type	Severity	Location
Volatile Code	Critical	EsusuAdapterWithdrawalDelegate.sol L235-L240

Description:

The code block on the aforementioned lines calculate the `ROI` in `DAI` for transfer to the user. It is using incorrect equations which results in incorrect amount beint sent to user as `ROI` .

Recommendation:

We recommend to rectify the equations to correctly send the `ROI` amount to the user. Aftert having discussion with the team, the correct payout calculation code was identified as followed:

```
uint Bt = _esusuStorage.GetEsusuCycleTotalBeneficiaries(esusuCycleId);
uint Ta = TotalMembers.sub(Bt);
uint Troi =
overallGrossDaiBalance.sub(_esusuStorage.GetEsusuCycleTotalAmountDeposited(esusuCycleId
).sub(_esusuStorage.GetEsusuCycleTotalCapitalWithdrawn(esusuCycleId)));
uint Mroi = Troi.div(Ta);
```

Alleviation:

Alleviation were applied as advised.

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

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapterWithdrawalDelegate.sol L362, L389, L416, L429

Description:

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

Recommendation:

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

Alleviation:

Alleviation were applied as advised partly.

EAW-06: Comparison with boolean literal

Type	Severity	Location
Language Specific	Informational	EsusuAdapterWithdrawalDelegate.sol L371, L397, L399

Description:

The aforementioned line perform comparison with boolean literals.

Recommendation:

Boolean expressions can be used directly instead of comparing them with boolean literals to increase the legibility of the codebase.

Alleviation:

Alleviation were applied as advised.

EAW-07: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapterWithdrawalDelegate.sol L375-L380

Description:

The `if-else` block on the aforementioned lines is unnecessary and can be replaced with a `return` statement.

Recommendation:

We recommend to remove the `if-else` block and replace it with a `return` statement.

```
    return now > memberWithdrawalTime;
```

Alleviation:

Alleviation were applied as advised.

EAW-08: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapterWithdrawalDelegate.sol L416-L421

Description:

The `if-else` block on the aforementioned lines is inefficient can be replaced with a `return` statement.

Recommendation:

We advise to remove `if-else` block and use `return` statement.

```
    return amount > 0;
```

Alleviation:

Alleviation were applied as advised.

EAW-09: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapterWithdrawalDelegate.sol L429-L434

Description:

The `if-else` block on the aforementioned lines is unnecessary and can be replaced with a `return` statement.

Recommendation:

We recommend to remove the `if-else` block and replace with a `return` statement.

```
    return amount > 0;
```

Alleviation:

Alleviation were applied as advised.

EAW-10: Usage of alias `uint` instead of complete form `uint256`

Type	Severity	Location
Language Specific	Informational	EsusuAdapterWithdrawalDelegate.sol L1

Description:

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

Recommendation:

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

Alleviation:

Alleviation were applied as advised.

EAW-11: Function visibility can be changed to `external`

Type	Severity	Location
Gas Optimization	Informational	EsusuAdapterWithdrawalDelegate.sol L100, L219, L340, L351

Description:

The functions on the aforementioned lines can have their visibilities changed from `public` to `external` as they are never called from within the contract.

Recommendation:

We recommend to change the visibilites of the functions on the aforementioned lines from `public` to `external`.

Alleviation:

Alleviation were applied as advised.

EAW-12: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EsusuAdapterWithdrawalDelegate.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;
```

Alleviation:

Alleviation were applied as advised.

RCG-01: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	RewardConfig.sol L79, L80

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:

Alleviation were applied as advised.

RCG-02: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	RewardConfig.sol L81

Description:

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

Recommendation:

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

Alleviation:

No alleviations.

RCG-03: Redundant Statements

Type	Severity	Location
Dead Code	Informational	RewardConfig.sol L84, L91

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:

Alleviation were applied as advised partly.

RCG-04: Comparison with boolean literal

Type	Severity	Location
Language Specific	Informational	RewardConfig.sol L136, L157, L177

Description:

The aforementioned lines perform comparison with boolean literal.

Recommendation:

We advise to directly use the boolean expression on the aforementioned lines instead of performing comparison with boolean literal.

Alleviation:

Alleviation were applied as advised.

RCG-05: Explicitly returning a local variable from function

Type	Severity	Location
Gas Optimization	Informational	RewardConfig.sol L133, L154, L174, L197, L229, L240, L254, L270, L286

Description:

The functions on the aforementioned lines explicitly return 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 from all of the function on the aforementioned lines in order to reduce the overall cost of gas.

Alleviation:

Alleviation were applied as advised.

RCG-06: Incorrect Spelling

Type	Severity	Location
Coding Style	Informational	RewardConfig.sol L194

Description:

The comment on the aforementioned line has incorrect spelling for the word `multiplied`.

Recommendation:

We recommend to correct the spellings of the word on the aforementioned line.

Alleviation:

Alleviation were applied as advised.

RCG-07: Incorrect order of functions

Type	Severity	Location
Language Specific	Informational	RewardConfig.sol

Description:

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

Recommendation:

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

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

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

Alleviation:

Alleviation were applied as advised.

RCG-08: Usage of alias `uint` instead of complete form `uint256`

Type	Severity	Location
Language Specific	Informational	RewardConfig.sol L1

Description:

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

Recommendation:

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

Alleviation:

Alleviation were applied as advised.

RCG-09: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	RewardConfig.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;
```

Alleviation:

Alleviation were applied as advised.

IEA-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IEsusuAdapter.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;
```

Alleviation:

Alleviation were applied as advised.

IES-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IEsusuService.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;
```

Alleviation:

Alleviation were applied as advised.

IES-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IEsusuStorage.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;
```

Alleviation:

Alleviation were applied as advised.

IRC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IRewardConfig.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;
```

Alleviation:

Alleviation were applied as advised.

Appendix

Finding Categories

Gas Optimization

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

Mathematical Operations

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

Logical Issue

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

Control Flow

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

Volatile Code

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

Data Flow

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

Language Specific

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

Coding Style

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

Inconsistency

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

Magic Numbers

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

Compiler Error

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

Dead Code

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