Cook Finance

Distribution and Reward

Security Assessment

March 30rd, 2021

[Preliminary Report]

Audited By:
Sheraz Arshad @ CertiK
sheraz.arshad@certik.org
Reviewed By:
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Cook Finance - Distribution and Reward
Description	The audited codebase consists smart contracts implementing functionality of allocating Cook tokens, staking Cook tokens and its LP tokens, and rewarding Cook tokens.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 02ec24eae10774d5bca5587b01a7d05c5ae303b2 2. f871ecb58c585aa87a22df2aa609c4b8e3a898cf

Audit Summary

Delivery Date	March 30rd, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	March 1st, 2021 - March 30rd, 2021

Vulnerability Summary

Total Issues	64
Total Critical	0
Total Major	3
Total Medium	1
Total Minor	9
Total Informational	51

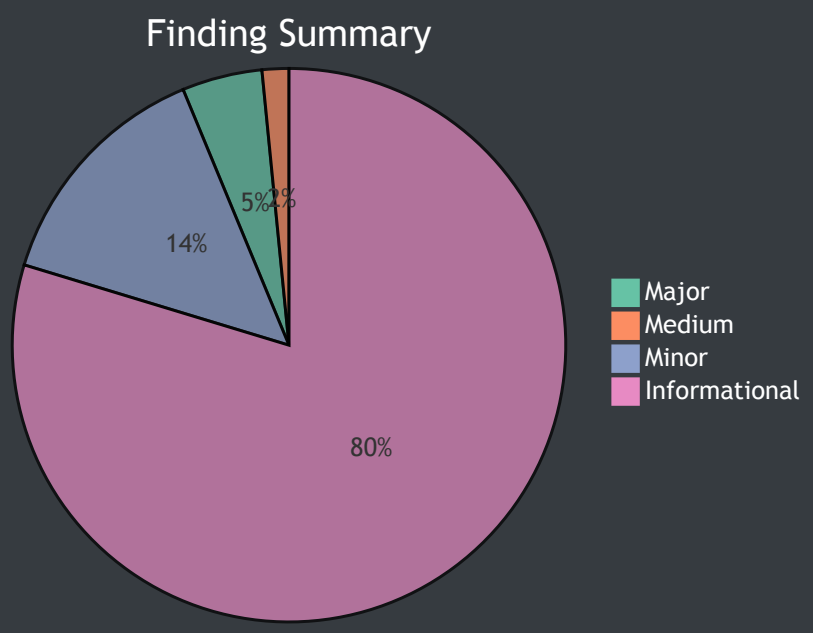
Executive Summary

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

Files In Scope

ID	Contract	Location
CON	Constants.sol	contracts/Constants.sol
COS	Constants.sol	contracts/core/Constants.sol
CDN	CookDistribution.sol	contracts/core/CookDistribution.sol
CPL	CookPool.sol	contracts/core/CookPool.sol
IPL	IPool.sol	contracts/core/IPool.sol
POO	Pool.sol	contracts/core/Pool.sol
PGS	PoolGetters.sol	contracts/core/PoolGetters.sol
PSS	PoolSetters.sol	contracts/core/PoolSetters.sol
PSE	PoolState.sol	contracts/core/PoolState.sol
UVL	UniswapV2Library.sol	contracts/external/UniswapV2Library.sol
UVO	UniswapV2OracleLibrary.sol	contracts/external/UniswapV2OracleLibrary.sol
IOE	IOracle.sol	contracts/oracle/IOracle.sol
IPC	IPriceConsumerV3.sol	contracts/oracle/IPriceConsumerV3.sol
IWE	IWETH.sol	contracts/oracle/IWETH.sol
ORA	Oracle.sol	contracts/oracle/Oracle.sol
PCV	PriceConsumerV3.sol	contracts/oracle/PriceConsumerV3.sol

File Dependency Graph


Finding Summary
5%
2%
14%
80%
Major
Medium
Minor
Informational

Manual Review Findings

ID	Title	Type	Severity	Resolved
COS-01	Unlocked Compiler Version	Language Specific	Informational	
CDN-01	Owner has centralized control over the contract's funds in Cook tokens	Volatile Code	Major	
CDN-02	Unclaimed allocation can be replaced by a new allocation	Logical Issue	Medium	
CDN-03	Lack of verification for the constructor parameters	Logical Issue	Minor	
CDN-04	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	
CDN-05	Event `AllocationRegistered` is not emitted	Logical Issue	Minor	
CDN-06	Lack of verification for the function's parameters	Logical Issue	Minor	
CDN-07	Unlocked Compiler Version	Language Specific	Informational	
CDN-08	mappings data can be packed in a struct	Gas Optimization	Informational	
CDN-09	Redundant Statements	Dead Code	Informational	
CDN-10	Missing reason string from `require` statement	Coding Style	Informational	
CDN-11	Redundant Variable Initialization	Coding Style	Informational	
CDN-12	Inefficient storage read	Gas Optimization	Informational	
CDN-13	Ineffectual declaration of `fallback` function	Coding Style	Informational	
CDN-14	Redundant Statements	Dead Code		

			Informational	
CDN-15	Return Variable Utilization	Gas Optimization	Informational	
CDN-16	Ineffectual code	Gas Optimization	Informational	
CDN-17	Inefficient storage read	Gas Optimization	Informational	
CDN-18	Redundant casting to `uint256`	Gas Optimization	Informational	
CDN-19	Comparison with boolean literal	Gas Optimization	Informational	
CDN-20	Inefficient `if-else` block	Gas Optimization	Informational	
CDN-21	Unnecessary `if-else` block	Gas Optimization	Informational	
CDN-22	Unnecessary `onlyOwner` restriction	Inconsistency	Informational	
CDN-23	Function Visibility Optimization	Gas Optimization	Informational	
CDN-24	Inefficient storage read	Gas Optimization	Informational	
CDN-25	Ineffectual tight packing of local variable	Gas Optimization	Informational	
CDN-26	Inefficient storage read	Gas Optimization	Informational	
CDN-27	Function's state mutability can be restricted to `view`	Language Specific	Informational	
CDN-28	Inefficient storage read	Gas Optimization	Informational	
CDN-29	Inefficient storage layout	Gas Optimization	Informational	
CDN-30	`require` statement can be subsituted with	Gas Optimization		

	modifier		Informational	
CPL-01	Owner has centralized control over the contract's funds in Cook tokens	Volatile Code	Major	
CPL-02	Lack of verification for the constructor parameter	Logical Issue	Minor	
CPL-03	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	
CPL-04	Unlocked Compiler Version	Language Specific	Informational	
CPL-05	Redundant Variable Initialization	Coding Style	Informational	
CPL-06	Ineffectual declaration of `fallback` function	Coding Style	Informational	
CPL-07	Redundant Statements	Dead Code	Informational	
IPL-01	Unlocked Compiler Version	Language Specific	Informational	
POO-01	Owner has centralized control over the contract's funds in Cook tokens	Volatile Code	Major	
POO-02	Lack of verification for the constructor parameters	Logical Issue	Minor	
POO-03	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	
POO-04	Unlocked Compiler Version	Language Specific	Informational	
POO-05	Redundant Variable Initialization	Coding Style	Informational	
POO-06	Ineffectual declaration of `fallback` function	Coding Style	Informational	
POO-07	Redundant Variable Initialization	Coding Style	Informational	
PGS-01	Unlocked Compiler Version	Language Specific		

			Informational	
PGS-02	Explicitly returning local variable	Gas Optimization	Informational	
PGS-03	Inefficient storage read	Gas Optimization	Informational	
PGS-04	Unnecessary casting of unsigned integer literal to `uint256`	Gas Optimization	Informational	
PGS-05	Comparison with boolean literal	Gas Optimization	Informational	
PSS-01	Unlocked Compiler Version	Language Specific	Informational	
PSS-02	Inefficient storage read	Gas Optimization	Informational	
PSS-03	`require` statement can be subsituted with modifier	Gas Optimization	Informational	
PSE-01	Unlocked Compiler Version	Language Specific	Informational	
PSE-02	Inefficient storage layout	Gas Optimization	Informational	
IOE-01	Unlocked Compiler Version	Language Specific	Informational	
IPC-01	Unlocked Compiler Version	Language Specific	Informational	
IWE-01	Unlocked Compiler Version	Language Specific	Informational	
ORA-01	Lack of verification for the constructor parameters	Logical Issue	Minor	
ORA-02	Unlocked Compiler Version	Language Specific	Informational	
ORA-03	Unused constructor parameter	Inconsistency	Informational	
ORA-04	Explicitly returning local variable	Gas Optimization		

			Informational	
PCV-01	Unlocked Compiler Version	Language Specific	Informational	

COS-01: Unlocked Compiler Version

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

CDN-01: Owner has centralized control over the contract's funds in Cook tokens

Type	Severity	Location
Volatile Code	Major	CookDistribution.sol L562

Description:

The `emergencyWithdraw` on the aforementioned line allows for emergency withdrawal of Cook tokens by the owner of the contract. This gives centralized control of funds to the owner of the contract which can result in unwanted state of the contract if the owner withdraws more Cook tokens than the already allocated amount or if the owner's account is compromised.

Recommendation:

We recommend to place a check in the `emergencyWithdraw` function which allows only the withdrawal of Cook tokens that are not already allocated. This can be achieved by keeping tracking of total allocations and total released amounts across all beneficiaries and the difference of both amounts is kept in the contract for withdrawal by the beneficiaries.

CDN-02: Unclaimed allocation can be replaced by a new allocation

Type	Severity	Location
Logical Issue	Medium	CookDistribution.sol L445

Description:

The function on the aforementioned line assigns allocation to a beneficiary. If a new allocation is assigned in the presense of previous unclaimed allocation then new allocation will replace the previous allocation.

Recommendation:

We advise to introduce a check esuring that the previous allocation has been claimed completely. This can be achieved by asserting that the allocation's amount is equal to allocation's released amount.

CDN-03: Lack of verification for the constructor parameters

Type	Severity	Location
Logical Issue	Minor	CookDistribution.sol L93, L100-L101

Description:

The constructor parameters on the aforementioned lines are not validated against zero value and once set to zero, they cannot be changed and results in unwanted state of the contract.

Recommendation:

We advise to validate the aforementioned constructor parameters against zero address value.

CDN-04: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	CookDistribution.sol L295, L377, L384, L389, L405, L411, L415, L563

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.

CDN-05: Event `AllocationRegistered` is not emitted

Type	Severity	Location
Logical Issue	Minor	CookDistribution.sol L445

Description:

The function `addAddressWithAllocation` on the aforementioned line registers beneficiary and allots allocation to it, yet it does not emit the event `AllocationRegistered`.

Recommendation:

We recommend to emit the event `AllocationRegistered` at the end of function's body.

CDN-06: Lack of verification for the function's parameters

Type	Severity	Location
Logical Issue	Minor	CookDistribution.sol L454

Description:

The parameters of function `updatePricePercentage` on the aforementioned line are not validated to have equal lengths and greater than zero.

Recommendation:

We advise to verify the parameters of the function on the aforementioned lines such that their lengths are equal and the length must not be zero.

```
require(
   priceKey_.length == percentageValue_.length && priceKey_.length > 0,
   "incorrect values are provided for priceKey and percentagekey"
);
```

CDN-07: Unlocked Compiler Version

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

CDN-08: mappings data can be packed in a struct

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L36, L39

Description:

The mappings on the aforementioned lines have key of type address representing a user's `address`. These mappings can be combined into a single mapping having `address` as key type and the value type will be a struct having properties from all of the aforementioned mappings. This will reduce the lookup gas cost when reading data from these mappings.

Recommendation:

We advise to replace the aforementioned mappings with a single mapping by utilizing a struct for the value types across all the aforementioned mappings.

```
struct Beneficiary {
   Allocation allocation;
   bool isRegistered;
}
```

```
mapping(address => Beneficiary) private _beneficiaries;
```

CDN-09: Redundant Statements

Type	Severity	Location
Dead Code	Informational	CookDistribution.sol L77

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.

CDN-10: Missing reason string from `require` statement

Type	Severity	Location
Coding Style	Informational	CookDistribution.sol L109-L110

Description:

The `require` statements on the aforementioned lines are missing reason strings.

Recommendation:

We recommend to add reason strings to the `require` statements on the aforementioned lines as they aid in debugging of the code and incease the legibility of the codebase.

CDN-11: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	CookDistribution.sol L149-L150, L152, L255, L338, L467, L478-L479, L487, L513

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.

CDN-12: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L158

Description:

The aforementioned line reads `_priceKey` array's length from the contract's storage upon each iteration of the `for` loop.

Recommendation:

We advise to store the array's length in a local variable which is gas efficient to read from compared to the storage of the contract.

CDN-13: Ineffectual declaration of `fallback` function

Type	Severity	Location
Coding Style	Informational	CookDistribution.sol L163

Description:

The fallback function on the aforementioned line is ineffectual as the same behaviour can be achieved by removing the `fallback` function from the code.

Recommendation:

We advise to remove the ineffectual declaration of `fallback` function to increase the legibility of the codebase.

CDN-14: Redundant Statements

Type	Severity	Location
Dead Code	Informational	CookDistribution.sol L72

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.

CDN-15: Return Variable Utilization

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L195, L200, L228, L216, L220, L224, L466, L476

Description:

The linked function declarations contain explicitly named `return` variables that are not utilized within the function's code block.

Recommendation:

We advise that the linked variables are either utilized or omitted from the declaration.

CDN-16: Ineffectual code

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L201, L207

Description:

The `effectiveDay` calculation on the aforementioned lines is redundant as it is subsequently calculated on `L229`.

Recommendation:

We advise to remove the redundant calculations on the aforementioned lines to save gas cost associated with its execution.

CDN-17: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L238

Description:

The body of `_getVestedAmount` function on the aforementioned line performs several inefficient storage reads. It performs storage read for `startDay()`, `_interval`, `_beneficiaryAllocations[userAddress]`, `_advancePercentage` and `_duration` multiple times which results in increased gas cost for function's execution.

Recommendation:

We recommend to store the aforementioned storage reads in a local variables and then utilize these variables in place of the storage reads which will cost significantly less gas compared to reading directly from the contract's storage.

CDN-18: Redundant casting to `uint256`

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L239, L481, L488

Description:

The aforementioned lines perform redundant castings of integer literals to `uint256`.

Recommendation:

We recommend to remove the reundant castings to `uint256` to save gas cost associated with it.

CDN-19: Comparison with boolean literal

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L277, L282, L287, L353, L358, L363

Description:

The aforementioned lines perform comparisons with boolean literal which results in increased gas cost for the operations.

Recommendation:

We advise to substitute the comparisons with boolean literal with the expression itself in case of comparsion with literal `true` and negation of expression in case of comparison with literal `false`.

CDN-20: Inefficient `if-else` block

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L380-L390

Description:

The `if-else` block on the aforementioned lines is not needed as the call to `_calWethAmountToPairCook` returns the address of asset which is paired with Cook token.

Recommendation:

We advise to remove the `if-else` block on the aforementioned lines and directly utilise the asset address contained as second value in the tuple returned from the call to `_calWethAmountToPairCook`.

```
(uint256 wethAmount, address assetAddress) = _calWethAmountToPairCook(cookAmount);
_token.transfer(_oracle.pairAddress(), cookAmount);

require(IERC20(assetAddress).balanceOf(msg.sender) >= wethAmount,"insufficient weth
balance");
require(IERC20(assetAddress).allowance(msg.sender,address(this)) >=
wethAmount,"insufficient weth allowance");
IERC20(assetAddress).transferFrom(msg.sender, _oracle.pairAddress(), wethAmount);

return (wethAmount, lpPair.mint(address(this)));
```

CDN-21: Unnecessary `if-else` block

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L407-L416

Description:

The `if-else` block on the aforementioned lines determines the asset paired with Cook token. The asset is already available in the scope of function as `wethAddress` rendering the `if-else` block redundant.

Recommendation:

We advise to substitute the `if-else` block with the transfer operation on the `wethAddress`.

```
_token.transfer(_oracle.pairAddress(), cookAmount);
require(IERC20(wethAddress).balanceOf(address(this)) >= wethAmount,"insufficient weth
balance");
IERC20(wethAddress).transferFrom(address(this), _oracle.pairAddress(), wethAmount);

return (wethAmount, lpPair.mint(address(this)));
```

CDN-22: Unnecessary `onlyOwner` restriction

Type	Severity	Location
Inconsistency	Informational	CookDistribution.sol L434, L438, L466, L476

Description:

The functions on the aforementioned lines do not modify the contract's state and are unnecessarily restricted by the `onlyOwner` modifier. As only data in the smart contract's storage is public, hence such restriction is ineffectual.

Recommendation:

We advise to remove the `onlyOwner` modifier from signatures of the aforementioned non-state modifying functions to increase the legibility of the codebase.

CDN-23: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L454

Description:

The linked function is declared as `public`, contains array function arguments and is not invoked in any of the contract's contained within the project's scope.

Recommendation:

We advise that the functions' visibility specifiers are set to `public` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

CDN-24: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L458-L459

Description:

The aforementioned lines perform inefficient storage read. It reads `_priceKey.length`, `_priceKey[i]` and `_percentageValue[i]` from the contract's storage upon each iteration of the `for` loop.

Recommendation:

We recommend utilize function parameters instead of reading from contract's storage as the same data is avaiable from function's parameters. It will significantly reduce the gas cost associated with function's execution.

```
for (uint256 i = 0; i < priceKey_.length; i++) {
   _pricePercentageMapping[priceKey_[i]] = percentageValue_[i];
}
```

CDN-25: Ineffectual tight packing of local variable

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L480, L515

Description:

The aforementioned lines utilize type `uint32` for the local variable used as counter of the `for` loop. As EVM operates on 32-byte values, the `uint32` type is padded to 32-byte for the EVM operation causing additional gas cost in the process.

Recommendation:

We advise to use the `uint256` type on the aforementioned lines as it will cost less gas during the function's execution.

CDN-26: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L481-L482

Description:

The aforementioned lines perform inefficient storage read. It reads `_oraclePriceFeed[today()-i]` from the contract storage twice.

Recommendation:

We recommend to store `_oraclePriceFeed[today()-i]` in a local variable and then utilize it in the code. It will be save gas cost associated with additional storage read.

CDN-27: Function's state mutability can be restricted to `view`

Type	Severity	Location
Language Specific	Informational	CookDistribution.sol L476

Description:

The function `getLatestSevenSMA` on the aforementioned line does not modify the contract's storage and hence its state mutability can be restricted to `view` to increase the legibility of the codebase.

Recommendation:

We advise to restricted the function's state mutability to `view`.

CDN-28: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L515-L517

Description:

The aforementioned lines perform inefficient storage reads. It reads `_priceKey.length` and `_priceKey[i]` from the contract's storage resulting in increased gas cost.

Recommendation:

We advise to store these values in local variable and then utilize them to reduce gas cost associated with multiple storage reads.

CDN-29: Inefficient storage layout

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L90, L53

Description:

The storage variables on the aforementioned lines occupy complete 32-byte slots, yet they can be tight-packed by placing them together on `L71`.

Recommendation:

We advise to place the storage variables on the aforementioned lines together on `L71` to tight pack them with the contract type variable `_priceConsumer`.

```
IPriceConsumerV3 private _priceConsumer;
uint32 private _interval;
bool private _pauseClaim;
bool private _revocable;
```

CDN-30: `require` statement can be subsituted with modifier

Type	Severity	Location
Gas Optimization	Informational	CookDistribution.sol L449, L454, L459, L467, L485, L503, L522, L556. L605, L611, L617, L623

Description:

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

Recommendation:

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

```
function isManager() {
   require(hasRole(MANAGER_ROLE, msg.sender), "Caller is not a manager");
}
```

```
modifier onlyManager() {
   isManager();
   _;
}
```

CPL-01: Owner has centralized control over the contract's funds in Cook tokens

Type	Severity	Location
Volatile Code	Major	CookPool.sol L219

Description:

The `emergencyWithdraw` on the aforementioned line allows for emergency withdrawal of Cook tokens by the owner of the contract. This gives centralized control of funds to the owner of the contract which can result in unwanted state of the contract if the owner withdraws more Cook tokens than the amount owed to users as their stakes or rewards, or if the owner's account is compromised.

Recommendation:

We recommend to place a check in the `emergencyWithdraw` function which allows only the withdrawal of Cook tokens that are not owed to the users as their stakes or rewards. Additionally, the emergency withdrawal must ensure that `REWARD_PER_BLOCK` is set to zero so no further Cook tokens are rewarded.

```
    require(
       cook().balanceOf(address(this)).sub(amount) >= totalStaked() + totalVesting() +
    totalRewarded() - totalClaimed() && REWARD_PER_BLOCK == 0.
       "cannot perform emergency withdrawal of cook tokens"
    );
```

CPL-02: Lack of verification for the constructor parameter

Type	Severity	Location
Logical Issue	Minor	CookPool.sol L17

Description:

The constructor parameter on the aforementioned line is not validated against zero value and once set to zero, it cannot be changed and results in unwanted state of the contract.

Recommendation:

We advise to validate the aforementioned constructor parameter against zero address value.

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

Type	Severity	Location
Logical Issue	Minor	CookPool.sol L39, L70, L114, L160

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.

CPL-04: Unlocked Compiler Version

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

CPL-05: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	CookPool.sol L18

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.

CPL-06: Ineffectual declaration of `fallback` function

Type	Severity	Location
Coding Style	Informational	CookPool.sol L32

Description:

The fallback function on the aforementioned line is ineffectual as the same behaviour can be achieved by removing the `fallback` function from the code.

Recommendation:

We advise to remove the ineffectual declaration of `fallback` function to increase the legibility of the codebase.

CPL-07: Redundant Statements

Type	Severity	Location
Dead Code	Informational	CookPool.sol L166

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.

IPL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IPool.sol L17

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;
```

POO-01: Owner has centralized control over the contract's funds in Cook tokens

Type	Severity	Location
Volatile Code	Major	Pool.sol L275

Description:

The `emergencyWithdraw` on the aforementioned line allows for emergency withdrawal of Cook tokens by the owner of the contract. This gives centralized control of funds to the owner of the contract which can result in unwanted state of the contract if the owner withdraws more Cook tokens than the already allocated amount or if the owner's account is compromised.

Recommendation:

We recommend to place a check in the `emergencyWithdraw` function which allows only the withdrawal of Cook tokens that are not already claimable or rewarded. Additionally, the emergency withdrawal must ensure that `REWARD_PER_BLOCK` is set to zero so no further Cook tokens are awarded.

```
    require(
       cook().balanceOf(address(this)).sub(amount) >= totalVesting() + totalRewarded() -
    totalClaimed() && REWARD_PER_BLOCK == 0,
       "cannot perform emergency withdrawal"
    );
```

POO-02: Lack of verification for the constructor parameters

Type	Severity	Location
Logical Issue	Minor	Pool.sol L16

Description:

The constructor parameters on the aforementioned lines are not validated against zero value and once set to zero, they cannot be changed and results in unwanted state of the contract.

Recommendation:

We advise to validate the aforementioned constructor parameters against zero address value.

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

Type	Severity	Location
Logical Issue	Minor	Pool.sol L44, L75, L119, L165, L196-L197, L212, L214

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.

POO-04: Unlocked Compiler Version

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

POO-05: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	Pool.sol L19

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.

POO-06: Ineffectual declaration of `fallback` function

Type	Severity	Location
Coding Style	Informational	Pool.sol L32

Description:

The fallback function on the aforementioned line is ineffectual as the same behaviour can be achieved by removing the `fallback` function from the code.

Recommendation:

We advise to remove the ineffectual declaration of `fallback` function to increase the legibility of the codebase.

POO-07: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	Pool.sol L233-L234

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.

PGS-01: Unlocked Compiler Version

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

PGS-02: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	PoolGetters.sol L76, L85, L94, L139, L147

Description:

The functions on the aforementioned lines explicitly return 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.

PGS-03: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	PoolGetters.sol L97, L141, L150

Description:

The aforementioned lines perform inefficient storage read where the array lengths from the contracts' storage are read upon each iterations of the `for` loops.

Recommendation:

We recommend to make use of local variables to store array lengths, so repeated reads from storage could be avoided resulting in reduced gas cost.

PGS-04: Unnecessary casting of unsigned integer literal to `uint256`

Type	Severity	Location
Gas Optimization	Informational	PoolGetters.sol L108, L162

Description:

The aforementioned lines perform unnecessary casting of unsigned integer literal to `uint256`.

Recommendation:

We advise to remove unnecessary casting to `uint256` on the aforementioned lines to save gas cost associated with it.

PGS-05: Comparison with boolean literal

Type	Severity	Location
Gas Optimization	Informational	PoolGetters.sol L194, L201, L209, L217

Description:

The aforementioned lines perform comparisons with boolean literal which results in increased gas cost for the operations.

Recommendation:

We advise to substitute the comparisons with boolean literal with the expression itself in case of comparsion with literal `true` and negation of expression in case of comparison with literal `false`.

PSS-01: Unlocked Compiler Version

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

PSS-02: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	PoolSetters.sol L44

Description:

The aforementioned line reads `_state.accounts[account].stakings` array's length from the contract's storage upon each iteration of the `for` loop.

Recommendation:

We recommend to store the array's length in a local variable so repeated reads from the contract's storage could be avoided saving gas cost associated with it.

PSS-03: `require` statement can be subsituted with modifier

Type	Severity	Location
Gas Optimization	Informational	PoolSetters.sol L112, L118, L124, L131, L138, L143, L148

Description:

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

Recommendation:

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

```
function isManager() {
   require(hasRole(MANAGER_ROLE, msg.sender), "Caller is not a manager");
}
```

```
modifier onlyManager() {
   isManager();
   _;
}
```

PSE-01: Unlocked Compiler Version

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

PSE-02: Inefficient storage layout

Type	Severity	Location
Gas Optimization	Informational	PoolState.sol L50

Description:

The struct member on the aforementioned line is inefficiently laid out in the storage where it is occupying a complete 32-byte slot. Additionally, the variable declaration has incorrect spellings for the word `Mining`

Recommendation:

We recommend to move the aforementioned struct member on `L39` such that `provider` occupying 20 bytes and `pauseMining` occupying 1 byte could tight packed in a single storage slot.

```
    Provider provider;
    bool pauseMining;
```

IOE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IOracle.sol L17

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;
```

IPC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IPriceConsumerV3.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;
```

IWE-01: Unlocked Compiler Version

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

ORA-01: Lack of verification for the constructor parameters

Type	Severity	Location
Logical Issue	Minor	Oracle.sol L34

Description:

The constructor parameters on the aforementioned lines are not validated against zero value and once set to zero, they cannot be changed and results in unwanted state of the contract.

Recommendation:

We advise to validate the aforementioned constructor parameters against zero address value.

ORA-02: Unlocked Compiler Version

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

ORA-03: Unused constructor parameter

Type	Severity	Location
Inconsistency	Informational	Oracle.sol L34

Description:

The constructor parameter `tokenB` on the aforementioned line is never used in the body of the constructor.

Recommendation:

We advise to remove the constructor parameter `tokenB` to increase the legibility of the codebase.

ORA-04: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	Oracle.sol L48

Description:

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

Recommendation:

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

PCV-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	PriceConsumerV3.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;
```

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.

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.

Volatile Code

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

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.

Dead Code

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