Security Assessment

CashCowFinance

Jun 17th, 2021

Table of Contents

Summary

This report has been prepared for CashCowFinance smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	CashCowFinance
Platform	BSC
Language	Solidity
Codebase	https://github.com/cashcowfinance/CashCowProtocolV2
Commit	5c75208f08e2977986fb8065e259594315fda3ba 161b35955be045a279bd2fac3247e9854b5d9042

Audit Summary

Delivery Date	Jun 17, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

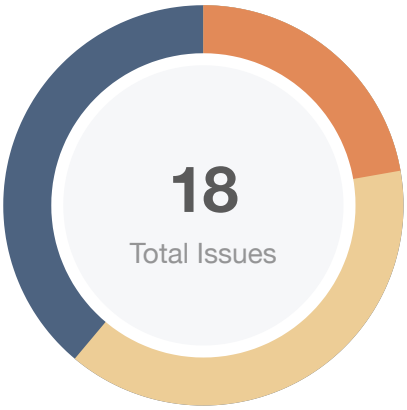
Total Issues	18
Critical	0
Major	4
Medium	0
Minor	7
Informational	7
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
GCK	contracts/Governable.sol	ebd9ab4e1da73fcdc05f4bd88ebde44fc09a174ea89a6c08cb6b964add6f17f9
HBC	contracts/HomoraBank.sol	91ae33ee2bbec25d3849b2db383112f9a53b8540a868c7f80775dd7218907d36
COC	contracts/oracle/CoreOracle.sol	97548ce4a80cb2d10296315d6f95b347fb3f267ea97cf6fef7a0ea18b4fd2621
POC	contracts/oracle/ProxyOracle.sol	9757e26faa26c2d4e18c379f2edd4b1fe5bf687281e3fd6b944000aeeb820089
UVO	contracts/oracle/UniswapV2Oracle.sol	26d9080fd5ea63cca7cfbda3345012a85156af3c7a544be96135f90dd6907659
UBO	contracts/oracle/UsingBaseOracle.sol	d9ca0fc593250f88501239d0913ba1292f899c652fb7f0b11caaf9ac8e9ee7a7
BSC	contracts/spell/BasicSpell.sol	a3305bc290f045a455bafdbe688067ab5b1476f8cce48e29c19b698c37419aa7
SSV	contracts/spell/SushiswapSpellV1.sol	204df1152ea65e50c2c4ed50b9ac783d97cba64e0bca3c8cc3c62bd4d801129e
WSC	contracts/spell/WhitelistSpell.sol	0420455b617f2c13d6e61aa41305e7da213e9103cb4078bc5174fd07d9a1f589
RDR	contracts/stake/RewardsDistributionRecipient.sol	0b81212c15664d487f9817f09c09568aaabeba453eab377960ba1101c87e7629
SRC	contracts/stake/StakingRewards.sol	89cc33b307ce57434db2a546aa504ad62bf3817b5b0689d714265e0e38c6bfbb
SRF	contracts/stake/StakingRewardsFactory.sol	16ece3db144cad09d5db9a822ba53cdceacd28eff7d497c3dab84523ebd2b4aa
BCC	contracts/utils/BConst.sol	5733e7df4c201968e58bfd25422c5f7193394208ce597e56dcdb4d629504d0d4
BNC	contracts/utils/BNum.sol	4a773c9a11c49331d08124e6a512b6a62b3dc9d9f99b25df645bb40257791934
ERC	contracts/utils/ERC1155NaiveReceiver.sol	1564f7b5b2d432c8274e0e535932051ab7401c21822954e2376f85298691aaa0
HMC	contracts/utils/HomoraMath.sol	34248d53236d4a64cfcc32be47564d657e678f1d5d8870bf0c3dca290648ef64
WER	contracts/wrapper/WERC20.sol	cc2b7592334bd2a83aa2fd167c9f98df37b01f312f5eb7154e38b28ca4ba79ea
WMC	contracts/wrapper/WMasterChef.sol	818fd3e95343600f528ad015df2133fa160db87e84b837db82fc68daddf310da
IBP	interfaces/IBalancerPool.sol	f6eb68c217ac6d54b70c277c6963fb8f036b40e4131f4d98ecc6e46a8dc055e2
IBC	interfaces/IBank.sol	f7f60add67a5dc1be8efbc74dfd3b05b54b98317865e5e9cb359ebec1aeccb87
IBO	interfaces/IBaseOracle.sol	e003782a46f3cbf713739bb7bc04a3ab1030fa00c86e19354bd5516b7500bc2c
ICE	interfaces/ICErc20.sol	aec14bcae287dcb97b33bf2e4cb160bd79a7f86be45beb444fee706cbeb3568d

ID	file	SHA256 Checksum
ICC	interfaces/ICErc20_2.sol	b907a0421df9242b4e7979852a250c2406e4a000e5337fca752b007eb59d7953
ICP	interfaces/ICurvePool.sol	67a2b1cd32abb674020b93b6dac35624d91e61286ccdea11563838bca88fc757
ICR	interfaces/ICurveRegistry.sol	26cd380e75b7350721f9103811d75d63ea053418b1d02b57c883346653b320f2
IER	interfaces/IERC20Wrapper.sol	df5ac95d18b5094446bbf2da58e5d797f8a6559d330f5653bd3bf8813ba58724
IKV	interfaces/IKeep3rV1Oracle.sol	9fc2e3810573e8ddd5522498462552bfc601f0e69fc6295197a0e51dd011c2b5
ILG	interfaces/ILiquidityGauge.sol	daeff9a5612d320131e813f4ead6ffa5a56a5a610e8b98bac267f663f4d8d37c
IMC	interfaces/IMasterChef.sol	d315fc438dca637007d8b2786b0128ac0e1adfc100f33a50df74f3caca27938a
IMD	interfaces/IMerkleDistributor.sol	eb54b847e25f48cc8cb37b1453e65cb8f4ac2ab21ef7f4d1165060f0dc2d7e78
IOC	interfaces/IOracle.sol	fd19246d834ac50e0f5617f9427b3598ede53acb910d68a05f6be2c9d56f4da7
ISR	interfaces/IStakingRewards.sol	d3fac242d54d54859d3710e35321dc8c3055edcc2b470cce18e53467240e1fd1
IUV	interfaces/IUniswapV2Factory.sol	2563045828a0d76caf483f8eb294db84c32966249de42dc646c4a005b8b5d42d
IUP	interfaces/IUniswapV2Pair.sol	50bd727d6693f2ab767ef35952c93db09fc7f0fd9d0d3e5ef9bde2b7e8059be3
IUR	interfaces/IUniswapV2Router01.sol	674ed2a94ca71e01cf6cc2ca626c4b555f2c4a93c0911b42c14dedb4eaa59425
IUC	interfaces/IUniswapV2Router02.sol	1cd48434d1b275dcc16af1077dbb1f6f91f8ba118dd7a0830dd945024a9c5c7b
IWE	interfaces/IWERC20.sol	2ace24b0de84cd2fca1bc49709c129faddb3ba21a528c820f78e6360f44f516e
IWT	interfaces/IWETH.sol	3ab9213a6fce0a84a8228885e22e476631dc4a7227c5cce6f873700ed49ede29
IWL	interfaces/IWLiquidityGauge.sol	ad57a179ac6ed20f87f38c7608987b866374022f5e401ad86fed3a51132417fc
IWM	interfaces/IWMasterChef.sol	1ef90d3fc0a322cb9b9cdd7c8517de304f0423dde8417f5c372adb510502870a
IWS	interfaces/IWStakingRewards.sol	520115a772a930bacb0387d1b7ff915d268e4d828c9eace009d16702421c19f4

Findings



Critical	**0** (0.00%)
Major	**4** (22.22%)
Medium	**0** (0.00%)
Minor	**7** (38.89%)
Informational	**7** (38.89%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
BSC-01	Lack of Input Validation	Volatile Code	Informational	Resolved
BSC-02	Unknown Implementation Of `IERC20(token)`	**Centralization / Privilege**	**Minor**	**Resolved**
GCK-01	Lack of Input Validation	Volatile Code	Informational	Resolved
HBC-01	Lack of Input Validation	Volatile Code	Informational	Resolved
HBC-02	Unknown Implementation Of `IERC20(token)`	**Centralization / Privilege**	**Minor**	**Resolved**
HBC-03	Inconsistent `feeBps`	Logical Issue	Minor	Resolved
HBC-04	Centralized Risk	**Centralization / Privilege**	**Major**	**Resolved**
POC-01	Centralized Risk	**Centralization / Privilege**	**Major**	**Resolved**
SRC-01	Lack of Input Validation	Volatile Code	Informational	Resolved
SRC-02	Centralized Risk	**Centralization / Privilege**	**Minor**	**Resolved**
SRC-03	Redundant Function	Gas Optimization	Informational	Resolved
SRC-04	Centralized Risk	**Centralization / Privilege**	**Major**	**Resolved**
SRC-05	Assignment Simplification	Gas Optimization	Minor	Resolved

ID	Title	Category	Severity	Status
SRF-01	Lack of Input Validation	Volatile Code	Informational	Resolved
SRF-02	Centralized Risk	**Centralization / Privilege**	**Major**	**Resolved**
SSV-01	Lack of Input Validation	Volatile Code	Informational	Resolved
SSV-02	Division Before Multiplication	Mathematical Operations	Minor	Resolved
SSV-03	Unknown Implementation Of `IERC20(tokenA)`, `IERC20(tokenB)`, and `IERC20(lp)`	**Centralization / Privilege**	**Minor**	**Resolved**

BSC-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	contracts/spell/BasicSpell.sol: 25	Resolved

Description

The values of `_bank`, `_werc20`, and `_weth` in the constructor of the contract `BasicSpell` should be verified as non-zero values to prevent errors.

Recommendation

Check that the passed-in values are non-zero. Example:

```
require(_bank != address(0), "_bank is a zero address");
require(_werc20 != address(0), "_werc20 is a zero address");
require(_weth != address(0), "_weth is a zero address");
```

Alleviation

`[Cash Cow Finance]`: The client heeded the advice and added the checks in the commit 161b35955be045a279bd2fac3247e9854b5d9042.

BSC-02 | Unknown Implementation Of `IERC20(token)`

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	contracts/spell/BasicSpell.sol: 40, 65, 67	**Resolved**

Description

In L40, L65, and L67, `token` can be any contract address where the `ERC20` interface is implemented. As a result, the function invocations from `IERC20(token)` in function `ensureApprove` and function `doRefund` may bring dangerous effects as the implementation is unknown to the user.

Recommendation

We advise the client to restrict the group of users who can access to function `ensureApprove` and function `doRefund` and check and ensure the contract specified by `IERC20(token)` is a standard smart contract that follows the `ERC20` interface with correct implementation.

Alleviation

`[Cash Cow Finance]`: The whitelisted `Token` and `LPToken` are restricted in the project. Theoretically, only checked ERC20 token will be added in the project. `WhitelistSpell.setWhitelistLPTokens()` and `HomoraBank.setWhitelistTokens()` are the two functions that are controlled the whitelist mechanism.

GCK-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	contracts/Governable.sol: 31	Resolved

Description

The value of `_pendingGovernor` in function `setPendingGovernor` should be verified as non-zero value to prevent errors.

Recommendation

Check that the passed-in value is non-zero. Example:

```
require(_pendingGovernor != address(0), "_pendingGovernor is a zero address");
```

Alleviation

`[Cash Cow Finance]`: The client heeded the advice and added the check in the commit 161b35955be045a279bd2fac3247e9854b5d9042.

HBC-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	contracts/HomoraBank.sol: 31	Resolved

Description

The value of `target` in the function 'cast' should be verified as non-zero to prevent errors.

Recommendation

Check that the passed-in value is non-zero. Example:

```
require(target != address(0), "target is a zero address");
```

Alleviation

`[Cash Cow Finance]`: The `target` spell must be a legit address, which will not be `address(0)`. Its value is also restricted by the whitelist mechanism.

HBC-02 | Unknown Implementation Of `IERC20(token)`

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	contracts/HomoraBank.sol: 417~418, 587, 637, 639, 662~664	**Resolved**

Description

In the aforementioned lines, `token` can be any contract address where the `ERC20` interface is implemented. As a result, the function invocations from `IERC20(token)` in these lines may bring dangerous effects as the implementation is unknown to the user.

Recommendation

We advise the client to restrict the group of users who can access to functions enclosing the aforementioned lines and check and ensure the contracts specified by `IERC20(token)` are standard smart contracts that follow the `ERC20` interface with the correct implementation.

Alleviation

`[Cash Cow Finance]`: The whitelisted `Token` and `LPToken` are restricted in the project. Theoretically, only checked ERC20 token will be added in the project. `WhitelistSpell.setWhitelistLPTokens()` and `HomoraBank.setWhitelistTokens()` are the two functions that are controlled the whitelist mechanism.

HBC-03 | Inconsistent `feeBps`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/HomoraBank.sol: 135, 441	Resolved

Description

In function `setFeeBps` on L441, the new value for `feeBps` is required to be no more than 10000. But for function `initialize` on L135, there is no such requirement.

Recommendation

We advise the client to add a requirement for function `initialize`:

```
require(_feeBps <= 10000, 'fee too high');
```

Alleviation

`[Cash Cow Finance]`: The client heeded the advice and added the requirement check in the commit 161b35955be045a279bd2fac3247e9854b5d9042.

HBC-04 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/HomoraBank.sol: 431~437, 508~515	Resolved

Description

In function `setStakes`, the governor of the contract `governor` could set the addresses for `stakes`. And on L511, an address from `stakes` is written to `positions[positionId].stake`. The value of `positions[positionId].stake` is used to determine whether to stake or withdraw on L514.

Recommendation

We advise the client to carefully manage the `governor` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Cash Cow Finance]`: Gnosis multi-sig wallet has been deployed to governance the project at address 0xF528DeD0B462Ad2d116FAe5104Ef0F9bD3c431C1

POC-01 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	contracts/oracle/ProxyOracle.sol: 42, 139	**Resolved**

Description

In function `setTokenFactors`, the governor of the contract `governor` could set the `borrowFactor` for each token by passing in the value `_tokenFactors`. And in function `asETHBorrow`, `borrowFactor` is used to determine the output value.

Recommendation

We advise the client to carefully manage the `governor` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Cash Cow Finance]`: Gnosis multi-sig wallet has been deployed to governance the project at address 0xF528DeD0B462Ad2d116FAe5104Ef0F9bD3c431C1

SRC-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	contracts/stake/StakingRewards.sol: 36~39	Resolved

Description

The values of `_rewardsDistribution`, `_bank`, `_rewardsToken`, `_stakingToken` in the constructor of the contract `StakingRewards` should be verified as non-zero values to prevent errors.

Recommendation

Check that the passed-in values are non-zero. Example:

```
require(_rewardsDistribution != address(0), "_rewardsDistribution is a zero address");
require(_bank != address(0), "_bank is a zero address");
require(_rewardsToken != address(0), "_rewardsToken is a zero address");
require(_stakingToken != address(0), "_stakingToken is a zero address");
```

Alleviation

`[Cash Cow Finance]`: The client heeded the advice and added the checks in the commit 161b35955be045a279bd2fac3247e9854b5d9042.

SRC-02 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	Minor	contracts/stake/StakingRewards.sol: 123, 159	Resolved

Description

`referral` is an address that can be updated by calling function `setReferral()` by `owner`. It can decide the implementation of function `payReferral()`, where presumably pay a certain fee from user to `referral` address. As the unknown of `referral` value, the implementation of `payReferral()` in L159 is decided by `owner` in fact. Any compromise to the account `owner` may allow the hacker to change the `setReferral()` address to receive the referral fees.

Recommendation

We advise the client to carefully manage the `owner` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Cash Cow Finance]`: Gnosis multi-sig wallet has been deployed to governance the project at address 0xF528DeD0B462Ad2d116FAe5104Ef0F9bD3c431C1

SRC-03 | Redundant Function

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/stake/StakingRewards.sol: 118~121	Resolved

Description

Function `exit()` is an internal function which cannot be called by external users, and also is not called by any functions in the project.

Recommendation

We advise the client to remove `exit()` function from the contract `StakingRewards.sol` to save gas.

Alleviation

`[Cash Cow Finance]`: The client heeded the advice and removed the redundant code in the latest commit.

SRC-04 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/stake/StakingRewards.sol: 133~134	Resolved

Description

The account that is granted with `governor` or `bank` role can stake to or withdraw from a specific user's balance by changing the user's value in `_balances` array. Any compromise to such accounts may allow the hacker to manipulate any specific user's ERC20 balance.

Recommendation

We advise the client to carefully manage the `governor` account's and the `bank` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Cash Cow Finance]`: Gnosis multi-sig wallet has been deployed to governance the project at address 0xF528DeD0B462Ad2d116FAe5104Ef0F9bD3c431C1

SRC-05 | Assignment Simplification

Category	Severity	Location	Status
Gas Optimization	Minor	contracts/stake/StakingRewards.sol: 143, 148	Resolved

Description

The expression `_balances[user].add(dev)` on L143 always has the same value as `amount`. And the expression `_balances[user].sub(dev)` on L148 always has the same value as `amount`.

Recommendation

Consider to simplify L143 as:

```
_balances[user] = amount;
```

And simplify L148 as:

```
_balances[user] = amount;
```

Alleviation

`[Cash Cow Finance]`: The client heeded the advice and simplified the code in the latest commit.

SRF-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	contracts/stake/StakingRewardsFactory.sol: 27~28	Resolved

Description

The values of `_bank`,`_rewardsToken` in the constructor of the contract `StakingRewardsFactory` should be verified as non-zero values to prevent errors.

Recommendation

Check that the passed-in values are non-zero. Example:

```
require(_bank != address(0), "_bank is a zero address");
require(_rewardsToken != address(0), "_rewardsToken is a zero address");
```

Alleviation

`[Cash Cow Finance]`: The client heeded the advice and added the check in the commit 161b35955be045a279bd2fac3247e9854b5d9042.

SRF-02 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/stake/StakingRewardsFactory.sol: 81, 91	Resolved

Description

In function `retrieveReward(address stakingToken)` the owner of the contract `owner` could send rewards to the contract. And in function `retrieveReward()`, the owner of the contract could send rewards from the contract to itself.

Recommendation

We advise the client to carefully manage the `owner` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Cash Cow Finance]`: Gnosis multi-sig wallet has been deployed to governance the project at address 0xF528DeD0B462Ad2d116FAe5104Ef0F9bD3c431C1

SSV-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	contracts/spell/SushiswapSpellV1.sol: 30~33	Resolved

Description

The values of `_bank`, `_werc20`, `_router`, `_wmasterchef` in the constructor of the contract `SushiswapSpellV1` should be verified as non-zero values to prevent errors.

Recommendation

Before line 35, check that the passed-in values are non-zero. Example:

```
require(address(_bank) != address(0), "_bank is a zero address");
require(_werc20 != address(0), "_werc20 is a zero address");
require(address(_router) != address(0), "_router is a zero address");
require(_wmasterchef != address(0), "_wmasterchef is a zero address");
```

Alleviation

`[Cash Cow Finance]`: The client heeded the advice and added the check in the commit 161b35955be045a279bd2fac3247e9854b5d9042.

SSV-02 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	Minor	contracts/spell/SushiswapSpellV1.sol: 95	Resolved

Description

Mathematical operations in the aforementioned line perform divisions before multiplications. Performing multiplication before division can sometimes avoid loss of precision.

Recommendation

We advise the client to consider ordering multiplication before division:

```
uint c = _c.mul(1000).mul(resA).div(amtB.add(resB));
```

Alleviation

`[Cash Cow Finance]`: Current implementation of mathematical operation follows the intended design.

SSV-03 | Unknown Implementation Of `IERC20(tokenA)`, `IERC20(tokenB)`, and `IERC20(lp)`

Category	Severity	Location	Status
Centralization / Privilege	Minor	contracts/spell/SushiswapSpellV1.sol: 141~142, 161~162, 281, 330~332	Resolved

Description

In the aforementioned lines, `tokenA`, `tokenB`, and `lp` can be any contract addresses where the `ERC20` interface is implemented. As a result, the function invocations from `IERC20(tokenA)`, `IERC20(tokenB)`, and `IERC20(lp)` in these lines may bring dangerous effects as the implementation is unknown to the user.

Recommendation

We advise the client to restrict the group of users who can access to functions enclosing the aforementioned lines and check and ensure the contracts specified by `IERC20(tokenA)`, `IERC20(tokenB)`, and `IERC20(lp)` are standard smart contracts that follow the `ERC20` interface with correct implementation.

Alleviation

`[Cash Cow Finance]`: The whitelisted `Token` and `LPToken` are restricted in the project. Theoretically, only checked ERC20 token will be added in the project. `WhitelistSpell.setWhitelistLPTokens()` and `HomoraBank.setWhitelistTokens()` are the two functions that are controlled the whitelist mechanism.

`[]`

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings 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 findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings 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.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.