CERTIK

SpiderDAO

Security Assessment

December 14th, 2020

For :
SpiderDAO

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	SpiderDAO
Description	The codebase contains the SpiderDAO smart contracts for a typical ERC20 implementation with burning, minting and pausing enhancements, batch token transfer capability, the vesting mechanisms as well as the liquidity farming.
Platform	Ethereum; Solidity, Yul
Codebase	1. Spider Liquidity Mining GitHub Repository 2. Spider Contracts GitHub Repository
Commits	pre-audit: 1. b1a2dc4cebfb132e5f195ca02836801dd9128293 2. 460caeac6dd24a128ce4184d572821814d6d50e1 post-audit: 1. 11cead52a13a0b662e9990c5ffa13b43d4539496 2. 69188f347c3f5ed92834d85645a934a3ae112bec

Audit Summary

Delivery Date	December 14th, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	3
Timeline	December 3rd, 2020 - December 14th, 2020

Vulnerability Summary

Total Issues	**28**
Total Critical	0
Total Major	0
Total Medium	4
Total Minor	5
Total Informational	19

Executive Summary

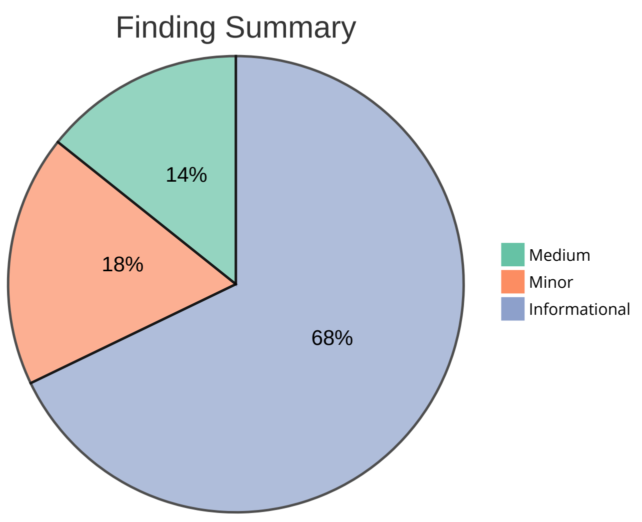
This report represents the results of CertiK's engagement with SpiderDAO on their implementation of the SpiderDAO smart contracts.

Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the safety of the contract deployment.

Files In Scope

ID	Contract	Location
BTR	BatchTransfer.sol	contracts/BatchTransfer.sol
PCR	PercentageCalculator.sol	contracts/PercentageCalculator.sol
STN	SpiderToken.sol	contracts/token/SpiderToken.sol
VES	Vesting.sol	contracts/Vesting.sol
VTM	VestingTeam.sol	contracts/VestingTeam.sol
VAS	VestingAdvisors.sol	contracts/VestingAdvisors.sol
LFG	LiquidityFarming.sol	contracts/LiquidityFarming.sol

Findings



ID	Title	Type	Severity	Resolved
STN-01	Unlocked Compiler Version	Language Specific	Informational	✓
STN-02	`external` Over `public` Function	Gas Optimization	Informational	✓
BTR-01	`external` Over `public` Function	Gas Optimization	Informational	✓
PCR-01	Unlocked Compiler Version	Language Specific	Informational	✓
VES-01	Redundant Named Variables	Gas Optimization	Informational	✓
VES-02	Add Existing Recipient as New	Volatile Code	Minor	✓
VES-03	Typo in an Error Message	Coding Style	Informational	!
VES-04	Use of `SafeERC20`	Volatile Code	Medium	✓
VES-05	Ambiguous Naming	Coding Style	Informational	!
VTM-01	Redundant Named Variables	Gas Optimization	Informational	✓
VTM-02	Add Existing Recipient as New	Volatile Code	Minor	✓
VTM-03	Typo in an Error Message	Coding Style	Informational	!
VTM-04	Use of `SafeERC20`	Volatile Code	Medium	✓
VTM-05	Ambiguous Naming	Coding Style	Informational	!
VAS-01	Redundant Named Variables	Gas Optimization	Informational	✓
VAS-02	Add Existing Recipient as New	Volatile Code	Minor	✓
VAS-03	Typo in an Error Message	Coding Style	Informational	!
VAS-04	Use of `SafeERC20`	Volatile Code	Medium	✓
VAS-05	Ambiguous Naming	Coding Style	Informational	!

ID	Title	Type	Severity	Resolved
LFG-01	Introduction of a `constant` Variable	Gas Optimization	Informational	✓
LFG-02	Introduction of a `immutable` Variable	Gas Optimization	Informational	ⓘ
LFG-03	Redundant `require` Statement	Gas Optimization	Informational	✓
LFG-04	Potential Re-entrancy	Language Specific	Minor	✓
LFG-05	Use of `SafeERC20`	Volatile Code	Minor	✓
LFG-06	`external` Over `public` Function	Gas Optimization	Informational	✓
LFG-07	`emergencyWithdraw` Functionality	Volatile Code	Medium	✓
LFG-08	Ambiguous Comments	Volatile Code	Informational	✓
LFG-09	Ambiguous Naming	Volatile Code	Informational	✓

STN-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	SpiderToken.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:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

STN-02: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	SpiderToken.sol L9

Description:

The linked `public` functions that are never called by a contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for functions never called from a contract.

Alleviation:

The development team opted to consider our references and changed the attribute of the linked function to an `external`.

BTR-01: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	BatchTransfer.sol L8

Description:

The linked `public` function that is never called by a contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for functions never called from a contract and change to `calldata` arguments.

Alleviation:

The development team opted to consider our references and optimized the linked function as proposed.

PCR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	PercentageCalculator.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:

The development team opted to consider our references and locked the compiler to version `0.6.2`.

VES-01: Redundant Named Variables

Type	Severity	Location
Gas Optimization	Informational	Vesting.sol L131, L143

Description:

The linked function use named variables while also having a `return` statement, which leads to unnecessary increase in gas.

Recommendation:

We advise the team to remove the named variables from the linked functions.

Alleviation:

The development team opted to consider our references and optimized one of the linked function as proposed.

VES-02: Add Existing Recipient as New

Type	Severity	Location
Volatile Code	Minor	Vesting.sol L70-L89

Description:

If a user tries adding an already existing recipient as a new recipient with the `addRecipient()` function, then the existing `withdrawnAmount` and `_withdrawPercentage` variables for that recipient will forever be lost.

Recommendation:

We advise the team to add a `require` statement to check whether the `recipients` mapping already contains data for the specific address.

Alleviation:

The development team opted to consider our references and added a `require` statement to ensure that there is no record for the specific address.

VES-03: Typo in an Error Message

Type	Severity	Location
Coding Style	Informational	Vesting.sol L106

Description:

The linked error message contains a typo.

Recommendation:

We advise the team to update the linked error message.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

VES-04: Use of `SafeERC20`

Type	Severity	Location
Volatile Code	Medium	Vesting.sol L123

Description:

The linked `require` statement causes incompatibility major token like `USDT`.

Recommendation:

We advise the team to remove the linked `require` statement and change from a `transfer` to a `safeTransfer`.

Alleviation:

The development team opted to consider our references and used the `safeTransfer` function of the `SafeERC20` contract.

VES-05: Ambiguous Naming

Type	Severity	Location
Coding Style	Informational	Vesting.sol L131-L138

Description:

The `hasClaim` function does not return a boolean variable.

Recommendation:

We advise the team to re-name the linked function.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

VTM-01: Redundant Named Variables

Type	Severity	Location
Gas Optimization	Informational	VestingTeam.sol L131, L143

Description:

The linked function use named variables while also having a `return` statement, which leads to unnecessary increase in gas.

Recommendation:

We advise the team to remove the named variables from the linked functions.

Alleviation:

The development team opted to consider our references and optimized one of the linked function as proposed.

VTM-02: Add Existing Recipient as New

Type	Severity	Location
Volatile Code	Minor	VestingTeam.sol L70-L89

Description:

If a user tries adding an already existing recipient as a new recipient with the `addRecipient()` function, then the existing `withdrawnAmount` and `_withdrawPercentage` variables for that recipient will forever be lost.

Recommendation:

We advise the team to add a `require` statement to check whether the `recipients` mapping already contains data for the specific address.

Alleviation:

The development team opted to consider our references and added a `require` statement to ensure that there is no record for the specific address.

VTM-03: Typo in an Error Message

Type	Severity	Location
Coding Style	Informational	VestingTeam.sol L106

Description:

The linked error message contains a typo.

Recommendation:

We advise the team to update the linked error message.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

VTM-04: Use of `SafeERC20`

Type	Severity	Location
Volatile Code	Medium	VestingTeam.sol L123

Description:

The linked `require` statement causes incompatibility major token like `USDT`.

Recommendation:

We advise the team to remove the linked `require` statement and change from a `transfer` to a `safeTransfer`.

Alleviation:

The development team opted to consider our references and used the `safeTransfer` function of the `SafeERC20` contract.

VTM-05: Ambiguous Naming

Type	Severity	Location
Coding Style	Informational	VestingTeam.sol L131-L138

Description:

The `hasClaim` function does not return a boolean variable.

Recommendation:

We advise the team to re-name the linked function.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

VAS-01: Redundant Named Variables

Type	Severity	Location
Gas Optimization	Informational	VestingAdvisors.sol L131, L143

Description:

The linked function use named variables while also having a `return` statement, which leads to unnecessary increase in gas.

Recommendation:

We advise the team to remove the named variables from the linked functions.

Alleviation:

The development team opted to consider our references and optimized one of the linked function as proposed.

VAS-02: Add Existing Recipient as New

Type	Severity	Location
Volatile Code	Minor	VestingAdvisors.sol L70-L89

Description:

If a user tries adding an already existing recipient as a new recipient with the `addRecipient()` function, then the existing `withdrawnAmount` and `_withdrawPercentage` variables for that recipient will forever be lost.

Recommendation:

We advise the team to add a `require` statement to check whether the `recipients` mapping already contains data for the specific address.

Alleviation:

The development team opted to consider our references and added a `require` statement to ensure that there is no record for the specific address.

VAS-03: Typo in an Error Message

Type	Severity	Location
Coding Style	Informational	VestingAdvisors.sol L106

Description:

The linked error message contains a typo.

Recommendation:

We advise the team to update the linked error message.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

VAS-04: Use of `SafeERC20`

Type	Severity	Location
Volatile Code	Medium	VestingAdvisors.sol L123

Description:

The linked `require` statement causes incompatibility major token like `USDT`.

Recommendation:

We advise the team to remove the linked `require` statement and change from a `transfer` to a `safeTransfer`.

Alleviation:

The development team opted to consider our references and used the `safeTransfer` function of the `SafeERC20` contract.

VAS-05: Ambiguous Naming

Type	Severity	Location
Coding Style	Informational	VestingAdvisors.sol L131-L138

Description:

The `hasClaim` function does not return a boolean variable.

Recommendation:

We advise the team to re-name the linked function.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

LFG-01: Introduction of a `constant` Variable

Type	Severity	Location
Gas Optimization	Informational	LiquidityFarming.sol L16-L17

Description:

The linked statements contain contract-level variable declarations and assignments, the variables of which are never assigned to elsewhere in the codebase.

Recommendation:

We advise that the mutability specifier `constant` is imposed on those variables to greatly reduce the gas cost incurred by utilizing them.

Alleviation:

The development team opted to consider our references and changed the mutability of the linked variables to `constant`.

LFG-02: Introduction of a `immutable` Variable

Type	Severity	Location
Gas Optimization	Informational	LiquidityFarming.sol L95-L97

Description:

A value is arbitrarily assigned to the linked variables in the constructor of the contract and are never updated afterwards.

Recommendation:

We advise the team to change the mutability of the linked variables to `immutable`.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

LFG-03: Redundant `require` Statement

Type	Severity	Location
Gas Optimization	Informational	LiquidityFarming.sol L235

Description:

The linked `require` statement is unnecessary, as the `sub` invocation in L240 will check the came conditional.

Recommendation:

We advise the team to remove the redundant code to save gas.

Alleviation:

The development team opted to consider our references and removed the redundant code.

LFG-04: Potential Re-entrancy

Type	Severity	Location
Language Specific	Minor	LiquidityFarming.sol L313

Description:

The linked statements make external calls before updating the state variables or emitting an event.

Recommendation:

We advise the team to make the external call last, hence following the Solidity coding standards.

Alleviation:

The development team opted to consider our references and made the external call at the end of the function.

LFG-05: Use of `SafeERC20`

Type	Severity	Location
Volatile Code	Minor	LiquidityFarming.sol L313

Description:

The linked statement invokes the `transferFrom` function.

Recommendation:

We advise the team to use `SafeERC20`'s `safeTransferFrom` function in the linked statement.

Alleviation:

The development team opted to consider our references and used the `safeTransferFrom` function of the `SafeERC20` contract.

LFG-06: `external` Over `public` Function

Type	Severity	Location
Gas Optimization	Informational	LiquidityFarming.sol L104, L125, L201, L207, L230, L250

Description:

The linked `public` functions that are never called by a contract should be declared `external` to save gas.

Recommendation:

We advise that the team uses the `external` attribute for functions never called from a contract.

Alleviation:

The development team opted to consider our references and changed the attribute of the linked function to an `external`.

LFG-07: `emergencyWithdraw` Functionality

Type	Severity	Location
Volatile Code	Medium	LiquidityFarming.sol L250-L261

Description:

The `emergencyWithdraw()` function only resets the `amount` and `rewardDebt` values for a particular user.

Recommendation:

We advise the team to reset the value of `rewardDebtAtBlock` for a particular user as well.

Alleviation:

The development team opted to consider our references and added the proposed functionality to the linked address.

LFG-08: Ambiguous Comments

Type	Severity	Location
Volatile Code	Informational	LiquidityFarming.sol L34-L46

Description:

The linked comment segment references the SushiSwap.

Recommendation:

We advise the team to either update or remove the linked comment segment.

Alleviation:

The development team opted to consider our references and removed the linked code segment.

LFG-09: Ambiguous Naming

Type	Severity	Location
Volatile Code	Informational	LiquidityFarming.sol L59

Description:

The `poolId1` mapping naming does not follow the standard coding conventions.

Recommendation:

We advise the team to change to a more descriptive name for the linked `mapping`, like `tokenToPoolId`.

Alleviation:

The development team opted to consider our references and re-named the linked variable.

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.