CERTIK

Cudo Vesting

Security Assessment

November 9th, 2020

By :

Camden Smallwood @ CertiK

camden.smallwood@certik.org

Alex Papageorgiou @ CertiK

alex.papageorgiou@certik.org

Disclaimer

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

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

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

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

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

What is a CertiK report?

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

Overview

Project Summary

Project Name	Cudo Ventures
Description	Cudos Vesting Contracts; Single vesting schedule per address; Schedules are defined as address, total amount, total time vested; User can withdraw every second and will issue allowance since last draw down; Cudo is in control of setting up and cancelling any vesting schedules; Utility methods can be called to gain insight into a beneficiaries remaining allowance and draw down rates
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	1. 6d0b5a2efc7e62fd53126be60175b9312f79c19f 2. b398415f7a4746236cb0441110d5739fffb95446

Audit Summary

Delivery Date	Nov. 9, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Oct. 31, 2020 - Nov. 1 2020

Vulnerability Summary

Total Issues	5
Total Critical	-
Total Major	-
Total Minor	-
Total Informational	5

Executive Summary

The Cudos Vesting contract was initially found to be incomplete, but following a commit from the Cudo Ventures team, the re-addressed Vesting contract was found to be well-written, modeling a typical vesting scheme and incorporates proper access restriction through the use of administrative roles. No security concerns were encountered while testing the functionality of the contract and only 5 informational optimizations were found, which were all resolved by the Cudo Ventures team in commit b398415f7a4746236cb0441110d5739fffb95446.

Findings

Prefix	File
VSC	VestingContract.sol

ID	Title	Type	Severity	Resolved
VSC-01	Missing SPDX license identifier	Language Specific	Informational	✓
VSC-02	Unnecessary return value	Implementation	Informational	✓
VSC-03	Potential for events emitting out of order	Control Flow	Informational	✓
VSC-04	Unnecessary return value	Implementation	Informational	✓
VSC-05	Public function should be declared external	Implementation	Informational	✓

VSC-01: Missing SPDX license identifier

Type	Severity	Location	Resolved
Language Specific	Informational	VestingContract.sol	✓

Description:

SPDX license identifier not provided in source file `VestingContract.sol`.

Recommendation:

Before publishing, consider adding a comment containing "SPDX-License-Identifier: " to `VestingContract.sol`. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.

Alleviation:

The recommendation was applied in commit b398415f7a4746236cb0441110d5739fffb95446.

VSC-02: Unnecessary return value

Type	Severity	Location	Resolved
Implementation	Informational	VestingContract.sol L70	✓

Description:

The `createVestingSchedule` function in the `VestingContract` contract always returns `true`, which makes the return value unnecessary due to being external and not being an override function:

```
function createVestingSchedule(address _beneficiary, uint256 _amount, uint256 _start,
uint256 _durationInDays, uint256 _cliffDurationInDays) external returns (bool) {
  ...

  emit ScheduleCreated(_beneficiary, _amount, _start, _durationInDays);

  return true;
}
```

Recommendation:

Consider removing the return value from the function signature and removing the return statement on line 99:

```
function createVestingSchedule(address _beneficiary, uint256 _amount, uint256 _start,
uint256 _durationInDays, uint256 _cliffDurationInDays) external {
  ...

  emit ScheduleCreated(_beneficiary, _amount, _start, _durationInDays);
}
```

Alleviation:

The recommendation was applied in commit b398415f7a4746236cb0441110d5739fffb95446.

VSC-03: Potential for events emitting out of order

Type	Severity	Location	Resolved
Control Flow	Informational	VestingContract.sol L97	✓

Description:

The `createVestingSchedule` function in the `VestingContract` contract would have the potential for events to be emitted out of order in the case of transferring from a re-entrant contract on line 82. While this unlikely due to potentially draining the sender's funds, it should typically be avoided and as such was dropped to an informational finding:

```
  require(
    token.transferFrom(msg.sender, address(this), _amount),
    "VestingContract.createVestingSchedule: Unable to transfer tokens to vesting contract"
  );

  ...

  emit ScheduleCreated(_beneficiary, _amount, _start, _durationInDays);
}
```

Recommendation:

Consider deferring the transfer from the token on line 82 until after the state variables and events have been emitted in order to alleviate the issue. If the transfer from the token fails, all changes to state variables will be reverted:

```
  ...

  emit ScheduleCreated(_beneficiary, _amount, _start, _durationInDays);

  require(
    token.transferFrom(msg.sender, address(this), _amount),
    "VestingContract.createVestingSchedule: Unable to transfer tokens to vesting contract"
  );
}
```

Alleviation:

The recommendation was applied in commit b398415f7a4746236cb0441110d5739fffb95446.

VSC-04: Unnecessary return value

Type	Severity	Location	Resolved
Implementation	Informational	VestingContract.sol L102	✓

Description:

The `drawDown` function in the `VestingContract` contract always returns `true`, which makes the return value unnecessary due to being external and not being an override function:

```
function drawDown() whenNotPaused nonReentrant external returns (bool) {
  ...

  emit DrawDown(msg.sender, amount, _getNow());

  return true;
}
```

Recommendation:

Consider removing the return value from the function signature and removing the return statement on line 119:

```
function drawDown() whenNotPaused nonReentrant external {
  ...

  emit DrawDown(msg.sender, amount, _getNow());
}
```

Alleviation:

The recommendation was applied in commit b398415f7a4746236cb0441110d5739fffb95446.

VSC-05: Public function should be declared external

Type	Severity	Location	Resolved
Implementation	Informational	VestingContract.sol L142	✓

Description:

The `tokenBalance` in the `VestingContract` contract is declared public, which is unnecessary due to not being used within the contract directly:

```
function tokenBalance() public view returns (uint256)
```

Recommendation:

Consider refactoring the function's visibility to external in order to save on the overall cost of gas:

```
function tokenBalance() external view returns (uint256)
```

Alleviation:

The recommendation was applied in commit b398415f7a4746236cb0441110d5739fffb95446.

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.