# Grant Protocol

## Security Assessment

January 5th, 2021

For :

DoraHacks

By :

Owan Li @ CertiK

guilong.li@certik.org

Bryan Xu @ CertiK

buyun.xu@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** | Grant Protocol |
| **Description** | A quadratic voting smart contract |
| **Platform** | Ethereum; Solidity |
| **Codebase** | GitHub Repository |
| **Commit** | 34cfa26703e3d84c5ed4ba442fb9df206a8b7055 |

## Audit Summary

| | |
|---|---|
| **Delivery Date** | Jan. 5, 2021 |
| **Method of Audit** | Static Analysis, Manual Review |
| **Consultants Engaged** | 2 |
| **Timeline** | Dec. 29, 2020 - Dec. 30, 2020 |

## Vulnerability Summary

| | |
|---|---|
| **Total Issues** | 9 |
| **Total Critical** | 0 |
| **Total Major** | 0 |
| **Total Minor** | 2 |
| **Total Informational** | 7 |

# Executive Summary

This report has been prepared for **Grant** smart contract 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 Dynamic Analysis, 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 audited commit is [34cfa26703e3d84c5ed4ba442fb9df206a8b7055] and the files included in the scope were as below：

Source Code SHA-256 Checksum

- [grant.sol]

b116df437667f7e917ed3456144aca753ac4b769cee3b68534517a7315597bfa

---

## Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the **DoraHacks** team or reported an issue.

---

## Review Notes

Certain optimization steps that we pinpointed in the source code  mostly referred to coding standards and inefficiencies, however 2 minor vulnerabilities were identified during our audit that solely concerns the specification.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project does not have adequate doumentation and specification outside of the source files, and the code comment coverage was minimal.

---

## Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

---

# Findings

| ID | Title | Type | Severity |
|---|---|---|---|
| Exhibit-01 | Unlocked Compiler Version Declaration | Language Sepcific | Informational |
| Exhibit-02 | Proper Usage of “public” and “external” type | Gas Optimization | Informational |
| Exhibit-03 | Simplifying Existing Code | Optimization | Informational |
| Exhibit-04 | Lack of natspec comments | Optimization | Informational |
| Exhibit-05 | Use SafeMath | Mathematical Operations | Informational |
| Exhibit-06 | Missing Important Checks | Logical Issue | Minor |
| Exhibit-07 | Potentially Dangerous Operation | Mathematical Operations | Informational |
| Exhibit-08 | Missing Emit Events | Optimization | Minor |
| Exhibit-09 | Reducing Lines of Code | Optimization | Informational |

## Exhibit-01: Unlocked Compiler Version Declaration

| Type | Severity | Location |
|---|---|---|
| Language Sepcific | Informational | grant.sol |

### Description:

The compiler version utilized throughout the project uses the “^” prefix specifier, denoting that a compiler version which is greater than the version will be used to compile the contracts. Recommend the compiler version should be consistent throughout the codebase.

### Recommendation:

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

### Alleviation:

The team heeded our advice and locked the version of their contracts at version 0.7.4, ensuring that compiler-related bugs can easily be narrowed down should they occur.
The recommendations were applied in commit ecf112457ad5edc6dee76d2482fbc6e80f59d8df.

##  Exhibit-02: Proper Usage of "public" and "external" type

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | grant.sol |

### Description:

"public" functions that are never called by the contract could be declared "external" . When the inputs are arrays "external" functions are more efficient than "public" functions.
Examples
Functions like : `allProjects()`,`rankingList()`, `roundInfo()`, `votingCost()`, `projectOf()`, `roundOver()`, `setTaxPoint()`, `setInterval()`, `setVotingUnit()`, `roundStart()`, `uploadProject()`, `vote()`, `takeOutGrants()`, `withdraw()`

### Recommendation:

Consider using the "external" attribute for functions never called from the contract.

### Alleviation:

The team heeded our advice and opted to change from “public” to “external” functions.
The recommendations were applied in commit ecf112457ad5edc6dee76d2482fbc6e80f59d8df.

##  Exhibit-03: Simplifying Existing Code

| Type | Severity | Location |
| --- | --- | --- |
| Optimization | Informational | grant.sol L122, L128, L133, L138, L202 |

### Description:

Consider using a modifier to replace the below re-used codes existing in many functions:
`require(msg.sender == owner);`

Example:
Functions `setTaxPoint()`, `setInterval()`, `setVotingUnit()`, `roundStart()`, `withdraw()`

### Recommendation:

Require code that is re-used many times could be put in a modifier.
Consider changing it as following example:

```
modifier onlyOwner() {
       require(msg.sender == owner, "!owner");
       _;
   }
```

### Alleviation:

The team heeded our advice and use a modifier to replace some re-used codes existing in many functions.
The recommendations were applied in commit ecf112457ad5edc6dee76d2482fbc6e80f59d8df.

## Exhibit-04: Lack of natspec comments

| Type | Severity | Location |
| --- | --- | --- |
| Optimization | Informational | grant.sol |

### Description:

Contract code is missing natspec comments, which helps understand the code and all the functions' parameters.

### Recommendation:

Please follow these style guides for adding natspec comments.
https://docs.soliditylang.org/en/v0.6.11/style-guide.html?highlight=natspec%23natspec

### Alleviation:

Currently no alleviation. The team confirmed to add the natspec comments later.

## Exhibit-5: Use SafeMath

| Type | Severity | Location |
| --- | --- | --- |
| Mathematical Operations | Informational | grant.sol L43,L76,L86,L116,L137,L145,L163,L192 |

### Description:

Many functions in the `grant` contract did not use SafeMath.

Example:
Functions like : `rankingList()`, `votingCost`, `grantsOf()`, `roundOver()`, `roundStart()`, `donate()`, `vote()`, `takeOutGrants()`

### Recommendation:

We recommend to use SafeMath for calculations.

### Alleviation:

The team made great use of the SafeMath library and the recommendations were applied in commit ecf112457ad5edc6dee76d2482fbc6e80f59d8df.

## Exhibit-6: Missing Important Checks

| Type | Severity | Location |
| --- | --- | --- |
| Logical Issue | Minor | grant.sol L116 |

### Description:

Function `roundOver` is missing some important checks.
This function can be called multiple times hence the `currentRound` could be changed incorrectly.

### Recommendation:

We recommend to add checks as below:

```
    function roundOver() public {
        require(block.timestamp > endTime[currentRound] &&
endTime[currentRound] > 0);
        currentRound++;
    }
```

### Alleviation:

The team heeded our advice and added the important check.
The recommendations were applied in commit ecf112457ad5edc6dee76d2482fbc6e80f59d8df.

## Exhibit-7: Potentially Dangerous Operation (Divide before Multiply)

| Type | Severity | Location |
| --- | --- | --- |
| Mathematical Operations | Informational | grant.sol L146 |

### Description:

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

For example, with msg.value = 9999, fee will be zero.

```
    function donate() public payable {
        uint256 fee = msg.value / 10000 * taxPoint;
        uint256 support = msg.value - fee;
        _tax += fee;
        supportPool[currentRound] += support;
        preTaxSupportPool[currentRound] += msg.value;
    }
```

### Recommendation:

Be overly cautious when performing multiplication on the result of a division.

### Alleviation:

The team made great use of the SafeMath library while also avoiding multiplication on the result of a division as much as possible. As a result, the possibility of a bad result, due to truncated digits, is significantly lowered.
The recommendations were applied in commit ecf112457ad5edc6dee76d2482fbc6e80f59d8df.

## Exhibit-08: Missing Emit Events

| Type | Severity | Location |
| --- | --- | --- |
| Optimization | Minor | grant.sol L121,L127,L132 |

### Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions like : `setTaxPoint()` , `setInterval` , `setVotingUnit()`

### Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below.

```
    event SetTaxPoint(uint256 taxPoint);
    function setTaxPoint(uint256 _taxPoint) public {
        require(msg.sender == owner);
        require(_taxPoint <= 5000);
        taxPoint = _taxPoint;
        emit SetTaxPoint(taxPoint);
    }
```

### Alleviation:

The team heeded our advice and added events for sensitive actions, and emit it in the function. The recommendations were applied in commit ecf112457ad5edc6dee76d2482fbc6e80f59d8df.

##  Exhibit-09: Reducing Lines of Code

| Type | Severity | Location |
| --- | --- | --- |
| Optimization | Informational | grant.sol L121,L127,L132 |

### Description:

The similar codes like below are re-used in function `votingCost()` and `vote()` which caused code redundancy.

```
    Project storage project = _projects[_projectID];
    votable = project.round == currentRound && block.timestamp <
endTime[currentRound];

    uint256 voted = project.votes[_from];
    uint256 votingPoints = (1 + _votes) * _votes / 2;
    votingPoints += voted * _votes;
    cost = votingPoints * votingUnit[project.round];
```

## Recommendation:

Consider to call the function `votingCost()` in function `vote()` to avoid redundant codes.

```
    function vote(uint256 _projectID, uint256 _votes) public payable {
    Project storage project = _projects[_projectID];
    (uint256 cost,bool votable,uint256 voted) = votingCost(msg.sender,
_projectID,         _votes);
    require(votable == true);
    require(msg.value >= cost);

    ...
        }
    }
```

## Alleviation:

The team heavily investigated this exhibit and concluded that the context of the method `vote()` needs more than the returns values of `votingCost()`, hence the existing code will not be changed for the lower gas to call.

# 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 instorage 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.