Audit Report

DoraHacks Quadratic Grant Injective

v1.0

March 4, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT IS ADDRESSED EXCLUSIVELY TO THE CLIENT. THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THE CLIENT OR THIRD PARTIES.

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Twenty Second Century Dora Technology Holdings Inc. to perform a security audit of the DoraHacks Quadratic Grant Injective smart contract.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/dorahacksglobal/quadratic-grant-injective
Commit	`6a2dbc7b6dcf7707f741df4c4bb3f79bdb78b12a`
Scope	All contracts were in scope.
Fixes verified at commit	`d1078015afca6b98bc8828d24f51b2e5f1962878` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The DoraHacks Quadratic Grant Injective is a smart contract, which supports multiple rounds of community funding, designed for Grant DAOs. Users can vote and donate to various projects with their voting weight adjusted by `vcDORA`, a non-transferrable governance credit issued to stakers. Upon the completion of a funding round, the admin is responsible for withdrawing and manually distributing the donations along with the matching prize pool to the participating projects.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	The protocol is written with the Sylvia framework, and a signature validation mechanism is implemented.
Code readability and clarity	Low-Medium	There are outstanding TODOs, commented code, magic numbers, and `println!` statements across the codebase.
Level of documentation	Low-Medium	Documentation is provided in the `README` file. However, it lacks documentation of entry points, execution flows, variables, and functions. Additionally, the documentation mentions that a quadratic progressive tax is implemented in the smart contract, which is misleading because it is computed off-chain.
Test coverage	Medium-High	`cargo tarpaulin` reports test coverage of `83.74%`.

Summary of Findings

No	Description	Severity	Status
1	Signatures can be replayed across different chains, networks, and contract addresses	Critical	Resolved
2	Signatures can be replayed within one hour before expiration	Major	Acknowledged
3	Quadratic funding taxation is not incentive-compatible	Major	Acknowledged
4	Incorrect voting calculation due to hardcoded decimals	Minor	Resolved
5	Rounds can be arbitrarily ended by the admins	Minor	Acknowledged
6	Lack of input validation	Minor	Resolved
7	Empty public keys configured allow users to inflate the `vcDORA` amount	Minor	Resolved
8	Updating the public key for completed voting rounds causes incorrect state	Minor	Resolved
9	Contracts should implement different admin roles and admin removals	Minor	Resolved
10	No entry point for banning projects	Minor	Acknowledged
11	Zero votes round cannot be marked as withdrawn	Informational	Resolved
12	Centralization risk when withdrawing project donations	Informational	Acknowledged
13	Usage of magic numbers decreases maintainability	Informational	Resolved
14	Overpaying funds will cause donations to fail	Informational	Acknowledged
15	Authorization check can be modularized into a function	Informational	Resolved
16	Limitations on base two logarithm implementation	Informational	Resolved
17	Small donations will not increase user votes due to rounding	Informational	Resolved

Detailed Findings

1. Signatures can be replayed across different chains, networks, and contract addresses

Severity: Critical

In `src/contract.rs:247-254`, the `weighted_batch_vote` function constructs the message for signature verification without including the chain ID and the contract address. This is problematic because signatures that are intended for different chains or contracts can be replayed, potentially bypassing the backend server's signature validation.

The impact of the issue depends on what signatures can be replayed. Fundamentally, the following bypasses can happen:

- A user can vote and donate to round IDs they are not allowed to.
- A user can vote and donate to projects they are not allowed to.
- A user can donate a different amount to projects they are not allowed to.
- A user can vote with a larger `vcDORA` amount they should not because they did not stake the required amount of DORA tokens.

To fix this issue, the chain ID and the contract address need to be included to ensure signatures cannot be replayed across different blockchains and deployments.

Adding the chain ID to the signed message prevents replaying across different blockchains, as well as across testnets and mainnet. During the development phase, test messages may be created for the testnet to verify the signature validation works as expected. Without a chain ID added to the message, these messages can be replayed on the mainnet. For example, a signed message with a large `vcDORA` value used on the testnet can be replayed to the mainnet to manipulate voting results.

Additionally, the missing chain ID from signed messages could also cause issues when different chains use the same `Bech32` prefix, such as Terra and Terra Classic. Since both chains use the same prefix, a signature that is intended for the former chain can be replayed on the latter chain. This can also affect future chains that go through a hard fork and end up with the same `Bech32` prefix.

Including the contract address as part of the signed message prevents replaying across different contracts. If two or more versions of the contracts are deployed, a signature over a message without the contract address can be replayed on another contract.

Recommendation

We recommend adding the chain ID and contract address to the signed message to prevent replay attacks.

Status: Resolved

2. Signatures can be replayed within one hour before expiration

Severity: Major

In `src/contract.rs:257`, the `weighted_batch_vote` function validates the signature is not expired as long the timestamp has not exceeded an hour. This is problematic because the signature can be replayed multiple times within the one-hour period.

Consequently, voters can repeatedly donate the same amount of funds to their projects after the backend server signs the message, manipulating the voting results.

We classify this issue as major because replaying a signature requires voters to donate funds to the protocol.

Recommendation

We recommend implementing a state to record all signed messages in line `247` to prevent a signature from being replayed.

Status: Acknowledged

The client states that users can only actively increase their `vcDORA` within a one-hour period. If a replay attack is executed, it would not necessarily impact the outcome or might result in the user's increased `vcDORA` becoming ineffective, thereby reducing their voting weight. Therefore, users lack a strong motive to conduct a replay attack, as it would either have no effect or potentially downgrade their voting power.

3. Quadratic funding taxation is not incentive-compatible

Severity: Major

In mechanism design and game theory, the incentive-compatible (IC) mechanism is one that guarantees every participant can achieve the best outcome for themselves by acting according to their true preferences, such as voting for their preferred projects

However, the documented Grant Funding Distribution Algorithm (Progressive Tax V2) is not incentive-compatible. This becomes evident by studying two exemplary scenarios involving four projects with distinct values, all of which are far from the mean:

- Project #1: the top project.

- Project #2: above average.
- Project #3: below average.
- Project #4: least number of votes.

In the first scenario, no redistribution occurs because the gap is exactly the desired gap. Based on the documentation, if `s` is greater than 1, then `V_max` / `V_min` is already smaller than `R`, meaning the results will not change.

In the second scenario, let's assume it continues from the first scenario, and the top project receives a marginally small contribution. As documented, this can cause a significant shift of all projects, including the middle ones, towards the mean/average.

Consequently, voters who favor the third or fourth projects will be better off voting for the first project to trigger a redistribution, thereby manipulating the results instead of voting for their real preferences. Conversely, voters who favor the top and second projects can vote for the project with the least votes to prevent redistribution from taxation. Both cases demonstrate that the voters are not voting for their actual preferred projects.

Furthermore, the redistribution is computed off-chain after the admin withdraws the donations. This means voters are incentivized to wait until the last seconds before admin withdrawal because the optimal strategy depends on the votes of all other voters.

Recommendation

We recommend implementing an incentive-compatible redistribution mechanism.

Status: Acknowledged

The client states that the progressive tax is not implemented in this contract. At the same time, for projects with votes below average, the greatest benefit always comes from voting for oneself, and there will not be a strategy where voting for top projects yields maximum benefits for oneself. However, for top projects, if the bottom is very low and there are few projects, voting for the bottom rather than oneself can be a locally optimal strategy. But vote distribution is usually pyramid-shaped. When the top tries to lift the bottom, it might face the need to lift the entire lower part of the pyramid, which could significantly increase the cost. In reality, this might be less effective than voting for oneself.

4. Incorrect voting calculation due to hardcoded decimals

Severity: Minor

In `src/contract.rs:278`, the `weighted_batch_vote` function declares a hardcoded `DECIMALS` value of `18`, representing the decimals of INJ token and later divided in line `281` as part of the vote calculation. This is problematic because the decimal values will be incorrect for other tokens, such as IBC tokens and token factory denoms.

Consequently, if one of these tokens is selected for the round (see line `149`), the vote calculation will be incorrect. Moreover, due to the nature of integer division, the votes can round to zero.

We classify this issue as minor because it can only be caused by the admins, which is a privileged role.

Recommendation

We recommend querying the decimal value for the denom instead of hardcoding it.

Status: Resolved

5. Rounds can be arbitrarily ended by the admins

Severity: Minor

The `end_round` function in `src/contract.rs:371-397` allows admins to update a round to the `RoundStatus::Finished` status. However, no conditions are being enforced before ending the round.

Consequently, admins can maliciously close a round when the votes are in favor of their preferred option or accidentally end a round earlier than intended.

We classify this issue as minor because it can only be caused by an admin, which is a privileged role.

Recommendation

We recommend enforcing a condition before rounds can be marked as finished, such as implementing a minimum voting time or a voting threshold.

Status: Acknowledged

The client states that they will reserve this flexibility for the admin.

6. Lack of input validation

Severity: Minor

In several instances of the codebase, validations are not performed sufficiently:

- In `src/contract.rs:45`, the `admins` vector is not validated to have at least one entry. If the vector is empty, there will be no admins controlling the contract, locking the main functionalities of the protocol and rendering it unusable. Consider returning an error if the vector is empty.
- In `src/contract.rs:149`, the `donation_denom` string is not validated as an existing native token. This would cause the `weighted_batch_vote` function to fail

in line `325`, preventing users from voting on projects. Consider querying the total supply of the denom to ensure it exists.
- In `src/contract.rs:150`, misconfiguring a `voting_unit` of zero causes all votes computed to become zero regardless of the donation amount, as seen in line `281`. Consider returning an error if the value is zero.
- In `src/contract.rs:156` and `415`, the public key is not validated to have a length of 65, which causes the public key comparison validation to fail in line `262`. Consider validating `pubkey.len()` to be 65.
- In `src/contract.rs:270`, no validation ensures the `project_ids` and `amounts` vectors have equal lengths and are not empty, which causes an out-of-bounds and `InvalidAmount` error, respectively. Consider returning an error early if this is the case.

Recommendation

We recommend applying the recommendations mentioned above.

Status: Resolved

7. Empty public keys configured allow users to inflate the `vcDORA` amount

Severity: Minor

In `src/contract.rs:243`, the `weighted_batch_vote` function validates the `vcDORA` amount signed by the backend server if the `round.pubkey` length is not zero. This validation is important because the `vcDORA` amount represents the governance credit issued to `DORA` token stakers, which is computed and verified as part of the message signature in line `261`.

The issue occurs when the `round.pubkey` is set to zero. In this case, the signature validation will not be performed, allowing voters to specify a large value of `vcDORA` to inflate the `area_diff` calculation in line `312`, ultimately affecting the quadratic funding result computed off-chain.

Consequently, voters can manipulate voting results.

This can happen due to a misconfiguration or ill-intended admin that sets the public key to an empty value because there is no proper validation in place, as detailed in the Lack of input validation issue.

We classify this issue as minor because it can only be caused by an admin, which is a privileged role.

Recommendation

We recommend removing the if statement in line `243` so signature validation is always performed.

Status: Resolved

8. Updating the public key for completed voting rounds causes incorrect state

Severity: Minor

In `src/contract.rs:415`, the `set_pubkey` function updates the public key for a voting round. However, no validation ensures the voting round is `RoundStatus::Voting`.

If the admin updates the public key to a different value for `RoundStatus::Finished` and `RoundStatus::Withdrawn`, users who query the voting rounds (see line `72`) will be misled that the new public key is used during the actual voting period, which is incorrect.

We classify this issue as minor because it can only be caused by an admin, which is a privileged role.

Recommendation

We recommend adding validation to ensure the voting round is `RoundStatus::Voting` before updating the public key.

Status: Resolved

9. Contracts should implement different admin roles and admin removals

Severity: Minor

The contracts within the scope of this audit allow multiple admins that can add more admins, and all admins have the same privileges. It is best practice to separate admin functionality into different roles with limited privileges.

Additionally, there is no entry point to remove admins. Consequently, there is a risk that old admin accounts get compromised when an admin is no longer on duty for the protocol, potentially leading to a loss of funds.

Recommendation

We recommend implementing an owner and admin model:

1. The protocol should only have one owner who can add and remove admins.

2. Admins cannot add or remove other admins but have admin privileges to govern the contract.

Status: Resolved

10. No entry point for banning projects

Severity: Minor

In `src/state.rs:15`, the `ProjectStatus` enum implements the `OK` and `Banned` variants. However, there is no entry point for the admins to update the project status to `Banned`. The `weighted_batch_vote` function also does not ensure the project is not banned when voting.

Consequently, projects that are found malicious cannot be banned and can still be voted on as well as donated to.

Recommendation

We recommend implementing an entry point for admins to ban projects and modifying the `weighted_batch_vote` function so banned projects cannot be voted on and donated to.

Status: Acknowledged

The client states that since the projects are pre-selected, there is no need to ban any projects in this round.

11. Zero votes round cannot be marked as withdrawn

Severity: Informational

In `src/contract.rs:450`, the `withdraw` function tries to distribute all donation funds to the admin for a specific round. An edge case is that if there are no donations, the `round.total_amounts` variable will be zero, causing the transaction to fail because Cosmos SDK does not allow zero-amount native token transfers.

Consequently, the voting round cannot be updated to `RoundStatus::Withdrawn` status, causing it to always remain at `RoundStatus::Finished` status.

Recommendation

We recommend sending the funds if `round.total_amounts` is larger than zero so the round status can be updated without error.

Status: Resolved

12. Centralization risk when withdrawing project donations

Severity: Informational

In `src/contract.rs:449`, the `withdraw` function distributes the donations of funded projects to the admin. This is a centralization risk, particularly because there can be multiple admins that cannot be removed.

It is best practice to send funds either to a custodian contract or to the project owner's address directly.

Recommendation

We recommend sending funds to a separate custodian contract or the project owner's address.

Status: Acknowledged

13. Usage of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such “magic numbers” goes against best practices as they reduce code readability and maintenance as developers cannot understand their use easily and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `src/contract.rs:257, 266, 308-309, 312`
- `src/helper.rs:14, 20`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Resolved

14. Overpaying funds will cause donations to fail

Severity: Informational

In `src/contract.rs:325-333`, the `weighted_batch_vote` function ensures the voters supply the exact amount of funds according to their votes. Otherwise, the transaction will revert with an error. This may decrease the user experience.

Recommendation

We recommend allowing users to overpay and sending the overpaid funds back to the user.

Status: Acknowledged

The client states that all transactions are submitted by their front end, and they expect the data to be an exact match. If it does not match, the transaction should fail directly.

15. Authorization check can be modularized into a function

Severity: Informational

The contracts within scope implement custom access controls, such as `src/contract.rs:106-110`. Although no issues with access control have been found, using a single assert function to validate authorization reduces potential risks while improving the codebase's readability and maintainability.

Recommendation

We recommend using modular functions to implement access control checks.

Status: Resolved

16. Limitations on base two logarithm implementation

Severity: Informational

In `src/helper.rs:2-25`, the `log2_u64_with_decimal` function implements a custom version of a base two logarithm. However, there are issues with its current implementation:

- The function returns 0 for the base 0 logarithm, which is incorrect because the result should be an undefined value. To reproduce this, please run `log2_u64_with_decimal(0)`.
- The function does not correctly handle all values of `u64`. An overflow error will trigger during multiplication if a large value is provided. To reproduce this error, please run `log2_u64_with_decimal(7259549383510990226)`.

Recommendation

We recommend modifying the implementation to return an error if the input is zero and using a larger integer size during computation to prevent overflow errors.

Status: Resolved

17. Small donations will not increase user votes due to rounding

Severity: Informational

In `src/contract.rs:281`, the `weighted_batch_vote` function computes votes by multiplying the donation amount by the voting unit and dividing the result by the decimal values. The voting unit is used to limit the minimum unit allowed for voting.

However, if the user donates less than the minimum voting amount, the votes computed will be zero due to rounding, causing a zero increase in the project and user votes.

Recommendation

We recommend returning an error if the computed votes are zero.

Status: Resolved