CERTIK

Security Assessment

Good Protocol Governance

Sept 13th, 2021

Table of Contents

Summary

Overview

- Project Summary
- Audit Summary
- Vulnerability Summary
- Audit Scope

Findings

- BGS-01 : Potential Reward Overminting
- BGS-02 : Inconsistent Reward Calculations
- CDG-01 : Lack of Event Emissions for Significant Transactions
- CVM-01 : Should Declare as `memory` for Read Only Object
- CVM-02 : Lack of Event Emissions for Significant Transactions
- MBG-01 : Potential Reward Overminting
- MBG-02 : Inconsistent Reward Calculations

Appendix

Disclaimer

About

Summary

This report has been prepared for GoodDollar to discover issues and vulnerabilities in the source code of the Good Protocol Governance project 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;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Good Protocol Governance
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/GoodDollar/GoodProtocol/tree/master/contracts/governance
Commit	7f743a54dae557d30351ee2fd4d2989864b04b99

Audit Summary

Delivery Date	Sept 13, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	2	0	0	0	0	2
Major	0	0	0	0	0	0
Medium	2	0	0	0	0	2
Minor	0	0	0	0	0	0
Informational	3	0	0	0	0	3
Discussion	0	0	0	0	0	0

Audit Scope

ID			File	SHA256 Checksum

Understandings

Overview

The GoodProtocol Governance contracts implement the governance reputation system and the voting system for the GoodProtocal project.

The governance reputation system is implemented by the contracts:

- ClaimersDistribution.sol
- GReputation.sol
- GovarnanceStaking.sol
- MultiBaseGovernanceShareField.sol
- Reputation.sol
- StakersDistribution.sol

The system mints reputation for users who staked their token or claimed the reputation in the Universal Basic Income (UBI) system. The reputation determines users' voting powers. Users can also delegate their voting powers to other accounts.

The voting system is implemented by

- CompoundVotingMachine.sol

It allows users to submit proposals, vote for proposals, cancel proposals, and execute succeeded proposals.

Dependencies

There are a few dependency injection contracts/addresses in the current project. We assume the contracts/addresses provided by `nameService` are valid and non-vulnerable actors, and they are implementing proper logic to collaborate with the current project.

Privilledged Functions

The restriction modifier `_onlyAvatar()` is applied in ClaimersDistribution.sol, GReputation.sol, GovernanceStaking.sol and StakersDistribution.sol, setting up the Avatar-Only writing access to modify system parameters or update crucial states of the contracts:

- `ClaimersDistribution.setMonthlyReputationDistribution()` to set monthly claimer reputation distributions;
- `GReputation. setBlockchainStateHash()` to set blockchain state hashes;

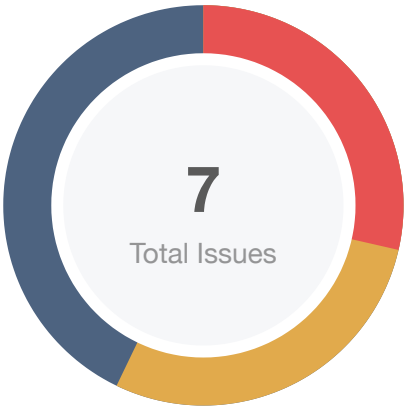
- `GovernanceStaking.setMonthlyRewards()` to set monthly rewards;
- `StakersDistribution.setMonthlyReputationDistribution ()` to set monthly staker reputation distributions.

The role `guardian` is introduced in the contract CompoundVotingMachine.sol to cancel proposals by calling `CompoundVotingMachine.cancel()`.

The restriction `_canMint()` is applied in the contract `GReputation` and `Reputation` to allow certain roles to mint reputation by calling `Reputation.mint()`.

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `CompoundVotingMachine` contract.

Findings



Critical	2 (28.57%)
Major	0 (0.00%)
Medium	2 (28.57%)
Minor	0 (0.00%)
Informational	3 (42.86%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
BGS-01	Potential Reward Overminting	Logical Issue	Critical	Resolved
BGS-02	Inconsistent Reward Calculations	Logical Issue	Medium	Resolved
CDG-01	Lack of Event Emissions for Significant Transactions	Logical Issue	Informational	Resolved
CVM-01	Should Declare as `memory` for Read Only Object	Gas Optimization	Informational	Resolved
CVM-02	Lack of Event Emissions for Significant Transactions	Logical Issue	Informational	Resolved
MBG-01	Potential Reward Overminting	Logical Issue	Critical	Resolved
MBG-02	Inconsistent Reward Calculations	Logical Issue	Medium	Resolved

BGS-01 | Potential Reward Overminting

Category	Severity	Location	Status
Logical Issue	Critical	BaseGovernanceShareField.sol: 157	Resolved

Description

The function `BaseGovernanceShareField._issueEarnedRewards()` in L157 returns the amount of a user's pending reward. However, in this function, `userInfo.rewardDebt` is not updated after the update of pending reward (i.e. `userInfo.rewardEarn`) in L159 (`_audit(user)`). Considering the pending reward is calculated logically as

```
    uint256 pending = (userInfo.amount * accAmountPerShare[_contract]) / 1e27 -
userInfo.rewardDebt;
```

If `userInfo.rewardDebt` is not updated, the pending reward will still be positive in the next call of `BaseGovernanceShareField._issueEarnedRewards()`. Although `userInfo.rewardEarn` is set as zero in L162, when the same user calls the function again right away, `userInfo.rewardEarn` could be updated as some non-zero value in L159 (`BaseGovernanceShareField._audit(user)`) and behave as the return value of the function.

Recommendation

We recommend adding proper `userInfo.rewardDebt` update logic in the function `BaseGovernanceShareField._issueEarnedRewards()` after executing `BaseGovernanceShareField._audit()`. Moreover, we advice encapsulating the logic of `userInfo.rewardDebt` update into the function `BaseGovernanceShareField._audit()` instead of doing it outside for better code readability and maintainability.

Alleviation

The development team removed this file in the commit 7c2ade00694466bb7e87e7e33cf689d4e79cd106.

BGS-02 | Inconsistent Reward Calculations

Category	Severity	Location	Status
Logical Issue	Medium	BaseGovernanceShareField.sol: 59, 141	Resolved

Description

When calculating the reward, the current code implementation applies `rounding` logic (via `DSMath.rdiv()`) instead of `flooring` logic, for instance,

```
57      accAmountPerShare =
58          accAmountPerShare +
59          rdiv(reward, totalProductivity * 1e16);
```

If a float number is rounded to a larger integer, it might lead to some unexpected behaviors. For example, assuming the contract sets 5 as monthly reward amount, meaning at most 5 rewards can be minted this month; and contains X (an even number) shares in total,

- Case 1: user A has X shares and could earn 5 rewards. In this case at most 5 rewards could be minted this month, which is as expected;
- Case 2: user A has X/2 shares and could earn 3 rewards (2.5, rounding to 3), and user B has X/2 shares and could earn 3 rewards (2.5, rounding to 3) as well. In this case, at most 6 rewards could be minted, which is larger than 5, the expected upper bound of rewards to be minted this month.

Ideally the expected behavior is, once the monthly reward amount is set as 5, the maximum amount of the reward that could be minted is always 5, regardless of the shares distribution. According to our observation, unless there is any specific design requirement, most projects will make use of `flooring` instead of `rounding` to avoid any potential project loss.

We hope to learn more about any specific reason to apply rounding instead of flooring. Is it an intended design?

Alleviation

The development team removed this file in the commit 7c2ade00694466bb7e87e7e33cf689d4e79cd106.

CDG-01 | Lack of Event Emissions for Significant Transactions

Category	Severity	Location	Status
Logical Issue	Informational	ClaimersDistribution.sol: 46	Resolved

Description

Event should be emitted for significant contract state(s) update. For example, in the function `ClaimerDistribution.setMonthlyReputationDistribution()`, it updates the an important state `monthlyReputationDistribution`.

Recommendation

We recommend emitting an event at the end of the aforementioned function.

Alleviation

The development team heeded our advice and fixed the issue in the commit bf5031f8a3372924227b9a84beec64446ea62772.

CVM-01 | Should Declare as `memory` for Read Only Object

Category	Severity	Location	Status
Gas Optimization	Informational	CompoundVotingMachine.sol: 327	Resolved

Description

In the function `CompoundVotingMachine.execute()`, `proposal` in L327 should be better declared as `memory` instead of `storage` for execution gas saving.
In general, when the variable is declared as `memory`, extra gas for reserving the memory would be paid in execution. From this perspective, using `storage` would be more efficient than using `memory`. However, reading from a `memory` variable is much more cost-efficient than reading from a `storage` variable. As a result, the more the reading happens later, the more the `EFFICIENCY` balance would be tilted towards `memory` variable.

In the loop in L329-L338, it iterates through a few array fields. The larger the array size is, the more gas would be saved if it reads from a `memory` variable.

Recommendation

We recommend updating L327 as

```
327     Proposal memory proposal = proposals[proposalId];
```

Alleviation

The development team heeded our advice and fixed the issue by loading the status into memory in the commit 97836ab191f57f7809f802a966350252bd49d8ac.

CVM-02 | Lack of Event Emissions for Significant Transactions

Category	Severity	Location	Status
Logical Issue	Informational	CompoundVotingMachine.sol: 655, 664	Resolved

Description

Event should be emitted for significant contract state(s) update. For example,
in the function `CompoundVotingMachine.renounceGuardian()`, it updates the important states: `guardian` and `foundationGuardianRelease`;
in the function `CompoundVotingMachine.setGuardian()`, it updates the important state `guardian`.

Recommendation

We recommend emitting corresponding events at the end of the aforementioned functions to log the state(s) update.

Alleviation

The development team heeded our advice and fixed the issue in the commit 97836ab191f57f7809f802a966350252bd49d8ac.

MBG-01 | Potential Reward Overminting

Category	Severity	Location	Status
Logical Issue	Critical	MultiBaseGovernanceShareField.sol: 199	Resolved

Description

The function `MultiBaseGovernanceShareField._issueEarnedRewards()` returns the amount of a user's pending reward which will be used in `GovernanceStaking._mintRewards()` and `StakersDistribution._claimReputation()` to mint reputation for the user.

However, in this function, `userInfo.rewardDebt` is not updated after the update of pending reward (`userInfo.rewardEarn`) in L206 (`_audit(_contract, _user)`). Considering the pending reward is calculated by

```
    uint256 pending = (userInfo.amount * accAmountPerShare[_contract]) / 1e27 -
userInfo.rewardDebt;
```

If `userInfo.rewardDebt` is not updated, the pending reward will still be positive in the next call of `MultiBaseGovernanceShareField._issueEarnedRewards()`, which means users can infinitely gain reward by calling the function `GovernanceStaking.withdrawRewards()` repeatedly (the calling routine is `GovernanceStaking.withdrawRewards()` => `GovernanceStaking._mintRewards()` => `MultiBaseGovernanceShareField._issueEarnedRewards()`). Although `userInfo.rewardEarn` is set as zero in L209, when the same user calls the function again right away, `userInfo.rewardEarn` could be updated as some non-zero value in L206 (`MultiBaseGovernanceShareField._audit(_contract, _user)`) and behave as the return value of the function. As a result, the amount of tokens would be minted for the user for another time.

Recommendation

We recommend adding proper `userInfo.rewardDebt` update logic in the function `MultiBaseGovernanceShareField._issueEarnedRewards()` after executing `MultiBaseGovernanceShareField._audit()`. Moreover, we advice encapsulating the logic of `userInfo.rewardDebt` update into the function `MultiBaseGovernanceShareField._audit()` instead of doing it outside for better code readability and maintainability.

Alleviation

The development team heeded our advice and fixed the issue in the commit 7792b1fedb72ad7b5ed1d7c8b5f46b2d04cfd147.

MBG-02 | Inconsistent Reward Calculations

Category	Severity	Location	Status
Logical Issue	Medium	MultiBaseGovernanceShareField.sol: 90	Resolved

Description

When calculating the reward, the current code implementation applies `rounding` logic (via `DSMath.rdiv()`) instead of `flooring` logic, for instance,

```
90      _accAmountPerShare += rdiv(reward, totalProductivity[_contract]);
```

If a float number is rounded to a larger integer, it might lead to some unexpected behaviors. For example, assuming the contract sets 5 as monthly reward amount, meaning at most 5 rewards can be minted this month; and contains X (an even number) shares in total,

- Case 1: user A has X shares and could earn 5 rewards. In this case at most 5 rewards could be minted this month, which is as expected;
- Case 2: user A has X/2 shares and could earn 3 rewards (2.5, rounding to 3), and user B has X/2 shares and could earn 3 rewards (2.5, rounding to 3) as well. In this case, at most 6 rewards could be minted, which is larger than 5, the expected upper bound of rewards to be minted this month.

Ideally the expected behavior is, once the monthly reward amount is set as 5, the maximum amount of the reward that could be minted is always 5, regardless of the shares distribution. According to our observation, unless there is any specific design requirement, most projects will make use of `flooring` instead of `rounding` to avoid any potential project loss.

We hope to learn more about any specific reason to apply rounding instead of flooring. Is it an intended design?

Alleviation

The development team heeded our advice and fixed the issue in the commit 3d59716b27dd5d55292792fbf41cd123c7eb6262.

Appendix

Finding Categories

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.

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.

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 you ("Customer" or 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, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

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.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

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.