# Cudos contest
# Findings & Analysis Report

2022-09-02

## Table of contents

# Overview

## About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Cudos smart contract system written in Solidity. The audit contest took place between May 3—May 9 2022.

## Wardens

64 Wardens contributed reports to the Cudos contest:

1. defsec
2. sorrynotsorry
3. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
4. p_crypt0

5. lllllll
6. dirk_y
7. 0xDjango
8. GermanKuber
9. WatchPug (jtp and ming)
10. 0x1337
11. dipp
12. jah
13. danb
14. cccz
15. GimelSec (rayn and sces60107)
16. Dravee
17. hubble (ksk2345 and shri4net)
18. kirk-baird
19. reassor
20. AmitN
21. csanuragjain
22. wuwe1
23. jayjonah8
24. 0xkatana
25. 0x1f8b
26. Funen
27. MaratCerby
28. gzeon
29. robee
30. oyc_109
31. ch13fd357r0y3r
32. ellahi
33. ilan

34. Waze
35. hake
36. simon135
37. delfin454000
38. JC
39. Hawkeye (0xwags and 0xmint)
40. orion
41. m9800
42. shenwilly
43. cryptphi
44. broccolirob
45. kebabsec (okkothejawa and FlameHorizon)
46. 0xNazgul
47. AlleyCat
48. slywaters
49. 0xf15ers (remora and twojoy)
50. rfa
51. peritoflores
52. 0v3rf10w
53. hansfriese
54. nahnah
55. jonatascm

This contest was judged by Albert Chon.

Final report assembled by liveactionllama.

## Summary

The C4 analysis yielded an aggregated total of 6 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 6 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 41 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 33 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

## Scope

The code under review can be found within the C4 Cudos contest repository, and is composed of 2 smart contracts written in the Solidity programming language and includes 615 lines of Solidity code.

## Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

## Medium Risk Findings (6)

### [M-01] Missing check in the `updateValset` function

*Submitted by CertoraInc, also found by 0x1337, cccz, danb, dipp, dirk_y, hubble, jah, and WatchPug*

[Gravity.sol#L276-L358]

The `updateValset` function don't check that the sum of the powers of the new validators in the new valset is greater than the threshold, which can lead to unwanted behavior.

There are 2 main problems that can occur in that situation:

1. The sum of the new validators' powers will be lower than the `state_powerThreshold`
2. The sum of the new validators' power will overflow and become lower than the `state_powerThreshold`

The second case is less dangerous, because it won't stuck the system in every case (only in specific cases where every sum of validators' power is less than the threshold). The first case is very dangerous though. It can lead to the system becoming stuck and to all of the tokens on the cudos chain to become locked for users, because the validators won't have enough power to approve any operation - whether it is transferring tokens or updating the valset.

## Proof of Concept

For the first case, consider the current validators set containing 100 validators with each ones power being equal to 10, and the threshold is 900 (91+ validators are needed for approvement). Now the `updateValset` function is being called with 100 validators with each ones power being equal to 1. This will lead to a state where no matter how much validators have signed a message, the sum of the powers won't pass the threshold and the action won't be able to be executed. This will cause all the tokens in the cudos blockchain become locked, and will DoS all the actions of the gravity contract - including updating the valset.

For the second case, consider the new validators set will have 128 validators, each validator's power is equal to `2**249` and `_powerThreshold = 2**256 - 1`. In this case the system will be stuck too, because every sum of validators' power won't pass the threshold.

### Tools Used

Remix and VS Code

### Recommended Mitigation Steps

Add a check in the `updateValset` to assure that the sum of the new powers is greater than the threshold.

**V-Staykov (Cudos) disputed and commented:**

> This check is done on the Gravity module side and since the message is also signed there by the validators, we can consider it to be always as per the module, unless there are malicious validators with more voting power than the threshold.

> If the message is considered correct this means that the values of the power are normalized which is in the core of the power threshold calculation. When they are normalized this means that the sum of the validator set will always equal 100% of the power which is more than the threshold.

> Here is a link to the power normalization in the Gravity module side.

**Albert Chon (judge) decreased severity to Medium and commented:**

> Agreed with @V-Staykov - this would only fail if 2/3+ of the validator stake weight were controlled by malicious validators, at which point all bets are off.

## [M-02] Admin drains all ERC based user funds using `withdrawERC20()`

*Submitted by pcrypt0, also found by 0x1337, AmitN, csanuragjain, danb, dirky, GermanKuber, lllllll, kirk-baird, and WatchPug*

Gravity.sol#L632-L638
Gravity.sol#L595-L609

Ability for admin to drain all ERC20 funds stored in contract at will, meaning all ERC20 based Cudos tokens (and any other ERC20 tokens stored in the contract)

could be extracted by anyone with admin role and later sold, leaving users funds bridged on Cudos Cosmos chain with no ERC20 representation stored across the bridge - similar in impact as the wormhole hack.

This issue ought to fall within the limits the team allocated on assessing the governance role setups, since it describes a full-fledged security risk regarding users' funds. Crucially, this function is not in the original Gravity Bridge contract for Gravity.sol.

Furthermore, the function has not been commented and does not appear in the documentation, suggesting that it has perhaps not yet been reasoned through by the development team and it's critical this is flagged in the security audit.

## Proof of Concept

Firstly, User with admin role granted waits until CUDOS bridge has decent TVL from users bridging their CUDOS tokens from Ethereum to the CUDOS Cosmos chain,

Secondly, User manually calls withdrawERC20(address _tokenAddress) with the ERC token address of the CUDOS token

```
    function withdrawERC20(
                            address _tokenAddress)
                            external {
                            require(cudosAccessControls.hasAdminRole
                            uint256 totalBalance = IERC20(_tokenAddr
                            IERC20(_tokenAddress).safeTransfer(msg.s
    }
```

Thirdly, withdrawERC20() function checks if user has admin role and if so withdraws all the tokens of a given token address straight to the admin's personal wallet

```
                    require(cudosAccessControls.hasAdminRole(msg.senc
                     uint256 totalBalance = IERC20(_tokenAddress).bal
                     IERC20(_tokenAddress).safeTransfer(msg.sender ,
```

Fourth, user exchanges CUDOS on DEX and then sends funds to tornado cash, leaving all user funds at risk.

### Tools Used

My own logical reasoning and discussion with team on Discord for confirmation of admin roles and function's logic.

### Recommended Mitigation Steps

Delete the function or alternatively, send all funds to the '0' address to burn rather than give them to the admin.

Change withdrawERC20 to:

```
    function burnERC20(
            address _tokenAddress)
            external {
            require(cudosAccessControls.hasAdminRole(msg.sender), "F
            uint256 totalBalance = IERC20(_tokenAddress).balanceOf(a
            - IERC20(_tokenAddress).safeTransfer(msg.sender , totalF
         +   IERC20(_tokenAddress).safeTransfer(address(0) , totalBa
    }
```

**maptuhec (Cudos) acknowledged and commented:**

> The reason we have created this functions is that, if the bridge stop working, the funds for the users would be locked, and there is no chance to withdraw them. CUDOS have no intention and incentive to maliciously withdraw the ERC20 tokes, because that would lead to losing the trust in its clients and thus killing their own network. The best way for handling this is to communicate this with the community so they can be aware.

**Albert Chon (judge) decreased severity to Medium**

## [M-03] The `Gravity.sol` should have pause/unpause functionality

*Submitted by defsec*

In case a hack is occuring or an exploit is discovered, the team (or validators in this case) should be able to pause functionality until the necessary changes are made to the system. Additionally, the gravity.sol contract should be manged by proxy so that upgrades can be made by the validators.

Because an attack would probably span a number of blocks, a method for pausing the contract would be able to interrupt any such attack if discovered.

To use a thorchain example again, the team behind thorchain noticed an attack was going to occur well before the system transferred funds to the hacker. However, they were not able to shut the system down fast enough. (According to the incidence report here).

### Proof of Concept

Gravity.sol#L175

### Recommended Mitigation Steps

Pause functionality on the contract would have helped secure the funds quickly.

mlukanova (Cudos) confirmed

V-Staykov (Cudos) resolved and commented:

> PR: CudoVentures/cosmos-gravity-bridge#18

## [M-04] Protocol doesn't handle fee on transfer tokens

*Submitted by wuwe1, also found by cccz, defsec, dipp, Dravee, GermanKuber, GimelSec, jah, reassor, and WatchPug*

Gravity.sol#L600

Since the `_tokenContract` can be any token, it is possible that loans will be created with tokens that support fee on transfer. If a fee on transfer asset token is chosen, other user's funds might be drained.

### Proof of Concept

1. Assume transfer fee to be 5% and `Gravity.sol` has 200 token.
2. Alice sendToCosmos 100 token. Now, `Gravity.sol` has 295 token.
3. Alice calls the send-to-eth method to withdraw 100 token.
4. `Gravity.sol` ends up having 195 token.

### Recommended Mitigation Steps

Change to

```
function sendToCosmos(
        address _tokenContract,
        bytes32 _destination,
        uint256 _amount
) public nonReentrant  {
        uint256 oldBalance = IERC20(_tokenContract).bala
        IERC20(_tokenContract).safeTransferFrom(msg.senc
        uint256 receivedAmout = IERC20(_tokenContract).b
        state_lastEventNonce = state_lastEventNonce.add(
        emit SendToCosmosEvent(
                _tokenContract,
                msg.sender,
                _destination,
                receivedAmout,
                state_lastEventNonce
        );
}
```

**mlukanova (Cudos) acknowledged and commented:**

> Token transfers are restricted to the Cudos token which doesn't support fee on transfer. Will be fixed with issue #58.

## [M-05] Calls inside loops that may address DoS

*Submitted by sorrynotsorry*

Calls to external contracts inside a loop are dangerous (especially if the loop index can be user-controlled) because it could lead to DoS if one of the calls reverts or execution runs out of gas. Reference

Gravity.sol#L453-L456

Gravity.sol#L568-L573

Gravity.sol#L579-L581

### Recommended Mitigation Steps

Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop.
Always assume that external calls can fail.
Implement the contract logic to handle failed calls.

mlukanova (Cudos) acknowledged

Albert Chon (judge) commented:

> Would really only happen for malicious/non-standard ERC-20 tokens which could then just be ignored by the orchestrator. No way of getting around doing the transfers for each token.

## [M-06] Non-Cudos Erc20 funds sent through `sendToCosmos()` will be lost.

*Submitted by p_crypt0, also found by CertoraInc*

No checks for non-Cudos tokens mean that non-Cudos ERC20 tokens will be lost to the contract, with the user not having any chance of retrieving them.

However, the admin can retrieve them through withdrawERC20.

Impact is that users lose their funds, but admins gain them.

The mistakes could be mitigated on the contract, by checking against a list of supported tokens, so that users don't get the bad experience of losing funds and CUDOS doesn't have to manually refund users

### Proof of Concept

User sends 100 ETH through sendToCosmos, hoping to retrieve 100 synthetic ETH on Cudos chain but finds that funds never appear.

```
function sendToCosmos(
        address _tokenContract,
        bytes32 _destination,
        uint256 _amount
) public nonReentrant  {
        IERC20(_tokenContract).safeTransferFrom(msg.senc
        state_lastEventNonce = state_lastEventNonce.add(
        emit SendToCosmosEvent(
                _tokenContract,
                msg.sender,
                _destination,
                _amount,
                state_lastEventNonce
        );
}
```

Gravity.sol#L595-L609

Admin can retrieve these funds should they wish, but user never gets them back because the contract does not check whether the token is supported.

```
function withdrawERC20(
        address _tokenAddress)
        external {
        require(cudosAccessControls.hasAdminRole(msg.ser
        uint256 totalBalance = IERC20(_tokenAddress).bal
        IERC20(_tokenAddress).safeTransfer(msg.sender ,
}
```

Gravity.sol#L632-L638

### Tools Used

Logic and discussion with @germanimp (Cudos)

### Recommended Mitigation Steps

Add checks in `sendToCosmos` to check the incoming tokenAddress against a supported token list, so that user funds don't get lost and admin don't need to bother refunding.

**mlukanova (Cudos) confirmed**

**V-Staykov (Cudos) resolved and commented**:

> PR: CudoVentures/cosmos-gravity-bridge#21

*Note: there were originally 7 items judged as Medium severity. After judging was finalized, further input from the sponsor was provided to the judge for reconsideration. Ultimately, the judge decreased issue #143 to non-critical.*

## Low Risk and Non-Critical Issues

For this contest, 41 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **IIIIIII** received the top score from the judge.

*The following wardens also submitted reports:* 0x1337, jayjonah8, GimelSec, dirk_y, GermanKuber, CertoraInc, ch13fd357r0y3r, kirk-baird, MaratCerby, gzeon, dipp, robee, 0xkatana, Hawkeye, sorrynotsorry, orion, hubble, jah, defsec, Waze, ilan, m9800, hake, shenwilly, AmitN, danb, Dravee, cccz, cryptphi, 0x1f8b, broccolirob, ellahi, Funen, 0xDjango, WatchPug, kebabsec, simon135, JC, oyc_109, *and* delfin454000.

### Low Risk Issues

| | Title | Instances |
|---|---|---|
| 1 | Validator signing address of zero not rejected, allowing anyone to sign | 1 |
| 2 | Unbounded loops may run out of gas | 1 |
| 3 | `deployERC20()` does not have a reentrancy guard | 1 |
| 4 | Comment does not match the behavior of the code | 2 |
| 5 | `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()` | 1 |

Total: 6 instances over 5 classes
(see lower down in this report for the summary table of the Non-critical findings)

## [1] Validator signing address of zero not rejected, allowing anyone to sign

`ecrecover()` returns `0` when the signature does not match. If the validators approve a valset including an address of `0`, then anyone will be able to sign messages for that signer, since invalid sigatures will return zero, and will match the zero address.

```
File: solidity/contracts/Gravity.sol   #1

185             return _signer == ecrecover(messageDigest, _v, _
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L185

## [2] Unbounded loops may run out of gas

The call to `ecrecover()` costs 3000 gas per call, and if there are too many validators, the update of the validator set may pass, but large batches will fail

```
File: solidity/contracts/Gravity.sol   #1

219     function checkValidatorSignatures(
```

```
220             // The current validator set and their powers
221             address[] memory _currentValidators,
222             uint256[] memory _currentPowers,
223             // The current validator's signatures
224             uint8[] memory _v,
225             bytes32[] memory _r,
226             bytes32[] memory _s,
227             // This is what we are checking they have signed
228             bytes32 _theHash,
229             uint256 _powerThreshold
230     ) private pure {
231             uint256 cumulativePower = 0;
232
233             for (uint256 i = 0; i < _currentValidators.lengt
234                     // If v is set to 0, this signifies that
235                     // (In a valid signature, it is either 2
236                     if (_v[i] != 0) {
237                             // Check that the current valida
238                             require(
239                                     verifySig(_currentValida
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L219-L239

## [3] `deployERC20()` does not have a reentrancy guard

`deployERC20()` increments the `state_lastEventNonce` so it's possible for the nonce to be incremented by a transfer hook. I don't see a way to exploit this given the code in scope, but perhaps some other area relies on event nonces happening in a specific order in relation to the other events.

```
File: solidity/contracts/Gravity.sol   #1

611     function deployERC20(
612             string memory _cosmosDenom,
613             string memory _name,
614             string memory _symbol,
615             uint8 _decimals
616     ) public {
617             // Deploy an ERC20 with entire supply granted to
618             CosmosERC20 erc20 = new CosmosERC20(address(this
```

```
619
620             // Fire an event to let the Cosmos module know
621             state_lastEventNonce = state_lastEventNonce.add(
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L611-L621

## [4] Comment does not match the behavior of the code

Both of the functions below have `require(isOrchestrator(msg.sender))`, and orchestrators are the first signer, so not just anyone can call these

```
File: solidity/contracts/Gravity.sol   #1

362     // Anyone can call this function, but they must supply v
363     // the batch.
364     function submitBatch (
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L362-L364

```
File: solidity/contracts/Gravity.sol   #2

274     // Anyone can call this function, but they must supply v
275     // the new valset.
276     function updateValset(
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L274-L276

## [5] `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`

Use `abi.encode()` instead which will pad items to 32 bytes, which will [prevent hash collisions](#) (e.g. `abi.encodePacked(0x123,0x456) => 0x123456 => abi.encodePacked(0x1,0x23456)`, but `abi.encode(0x123,0x456) => 0x0...1230...456`). "Unless there is a compelling reason, `abi.encode` should be preferred". If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` [instead](#).

```
File: solidity/contracts/Gravity.sol    #1

182             bytes32 messageDigest = keccak256(
183                     abi.encodePacked("\x19Ethereum Signed Me
184             );
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L182-L184

## Non-critical Issues

| | Title | Instances |
|---|---|---|
| 1 | Best practice is to prevent signature malleability | 1 |
| 2 | Inconsistent variable naming convention | 2 |
| 3 | Inconsistent tabs vs spaces | 3 |
| 4 | `if(` should be `if (` to match other lines in the file | 1 |
| 5 | Misleading function name | 1 |
| 6 | Avoid the use of sensitive terms in favor of neutral ones | 4 |
| 7 | `public` functions not called by the contract should be declared `external` instead | 10 |
| 8 | `2**<n> - 1` should be re-written as `type(uint<n>).max` | 1 |
| 9 | `constant`s should be defined rather than using magic numbers | 3 |
| 10 | Use a more recent version of solidity | 1 |
| 11 | Variable names that consist of all capital letters should be reserved for | 1 |

| | Title | Instances |
|---|---|---|
| | `const` / `immutable` variables | |
| 12 | Non-library/interface files should use fixed compiler versions, not floating ones | 2 |
| 13 | Typos | 1 |
| 14 | File does not contain an SPDX Identifier | 2 |
| 15 | File is missing NatSpec | 2 |
| 16 | Event is missing `indexed` fields | 5 |
| 17 | Consider making the bridge ‘pausable’ | 1 |

Total: 41 instances over 17 classes

## [1] Best practice is to prevent signature malleability

Use OpenZeppelin’s `ECDSA` contract rather than calling `ecrecover()` directly

```
File: solidity/contracts/Gravity.sol   #1

182             bytes32 messageDigest = keccak256(
183                     abi.encodePacked("\x19Ethereum Signed Me
32", _theHash)
184             );
185             return _signer == ecrecover(messageDigest, _v, _
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L182-L185

## [2] Inconsistent variable naming convention

Most state variables use the `state_` prefix in their variable name. There are some that don’t. Use the prefix everywhere, and manually add public getters where necessary

```
File: solidity/contracts/Gravity.sol   #1
```

```
63      CudosAccessControls public cudosAccessControls;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L63

```
File: solidity/contracts/Gravity.sol   #2

65      mapping(address => bool) public whitelisted;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L65

## [3] Inconsistent tabs vs spaces

Most lines use tabs, but some use spaces, which leads to alignment issues

```
File: solidity/contracts/Gravity.sol   #1

128             for (uint256 i = 0; i < _users.length; i++) {
129             require(
130                 _users[i] != address(0),
131                 "User is the zero address"
132             );
133             whitelisted[_users[i]] = _isWhitelisted;
134         }
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L128-L134

```
File: solidity/contracts/Gravity.sol   #2

117             require(
118             whitelisted[msg.sender] || cudosAccessControls.
119             "The caller is not whitelisted for this operati
```

```
120           );
121               _;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L117-L121

```
File: solidity/contracts/Gravity.sol   #3

647               address[] memory _validators,
648       uint256[] memory _powers,
649               CudosAccessControls _cudosAccessControls
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L647-L649

## [4] `if(` should be `if (` to match other lines in the file

```
File: solidity/contracts/Gravity.sol   #1

264                   if(_newValset.validators[i] == _sender)
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L264

## [5] Misleading function name

`onlyWhitelisted()` should be `onlyWhitelistedOrAdmin()`

```
File: solidity/contracts/Gravity.sol   #1

116     modifier onlyWhitelisted() {
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L116

## [6] Avoid the use of sensitive terms in favor of neutral ones

Use allowlist rather than whitelist

```
File: solidity/contracts/Gravity.sol   #1

116     modifier onlyWhitelisted() {
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L116

```
File: solidity/contracts/Gravity.sol   #2

65      mapping(address => bool) public whitelisted;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L65

```
File: solidity/contracts/Gravity.sol   #3

109     event WhitelistedStatusModified(
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L109

```
File: solidity/contracts/Gravity.sol   #4

124     function manageWhitelist(
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L124

## [7] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

```
File: solidity/contracts/Gravity.sol   #1

124     function manageWhitelist(
125             address[] memory _users,
126             bool _isWhitelisted
127             ) public onlyWhitelisted {
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L124-L127

```
File: solidity/contracts/Gravity.sol   #2

140     function testMakeCheckpoint(ValsetArgs memory _valsetArg
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L140

```
File: solidity/contracts/Gravity.sol   #3

144     function testCheckValidatorSignatures(
145             address[] memory _currentValidators,
146             uint256[] memory _currentPowers,
147             uint8[] memory _v,
148             bytes32[] memory _r,
149             bytes32[] memory _s,
150             bytes32 _theHash,
```

```
151             uint256 _powerThreshold
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L144-L151

```
File: solidity/contracts/Gravity.sol   #4

166     function lastBatchNonce(address _erc20Address) public vi
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L166

```
File: solidity/contracts/Gravity.sol   #5

170     function lastLogicCallNonce(bytes32 _invalidation_id) pu
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L170

```
File: solidity/contracts/Gravity.sol   #6

276     function updateValset(
277             // The new version of the validator set
278             ValsetArgs memory _newValset,
279             // The current validators that approve the chanc
280             ValsetArgs memory _currentValset,
281             // These are arrays of the parts of the current
282             uint8[] memory _v,
283             bytes32[] memory _r,
284             bytes32[] memory _s
285     ) public nonReentrant {
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gra

vity.sol#L276-L285

File: solidity/contracts/Gravity.sol #7

```
364     function submitBatch (
365             // The validators that approve the batch
366             ValsetArgs memory _currentValset,
367             // These are arrays of the parts of the validatc
368             uint8[] memory _v,
369             bytes32[] memory _r,
370             bytes32[] memory _s,
371             // The batch of transactions
372             uint256[] memory _amounts,
373             address[] memory _destinations,
374             uint256[] memory _fees,
375             uint256 _batchNonce,
376             address _tokenContract,
377             // a block height beyond which this batch is not
378             // used to provide a fee-free timeout
379             uint256 _batchTimeout
380     ) public nonReentrant {
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L364-L380

File: solidity/contracts/Gravity.sol #8

```
479     function submitLogicCall(
480             // The validators that approve the call
481             ValsetArgs memory _currentValset,
482             // These are arrays of the parts of the validatc
483             uint8[] memory _v,
484             bytes32[] memory _r,
485             bytes32[] memory _s,
486             LogicCallArgs memory _args
487     ) public nonReentrant {
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L479-L487

```
File: solidity/contracts/Gravity.sol   #9

595     function sendToCosmos(
596             address _tokenContract,
597             bytes32 _destination,
598             uint256 _amount
599     ) public nonReentrant  {
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L595-L599

```
File: solidity/contracts/Gravity.sol   #10

611     function deployERC20(
612             string memory _cosmosDenom,
613             string memory _name,
614             string memory _symbol,
615             uint8 _decimals
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L611-L615

## [8] `2**<n> - 1` should be re-written as `type(uint<n>).max`

Earlier versions of solidity can use `uint<n>(-1)` instead. Expressions not including the `- 1` can often be re-written to accomodate the change (e.g. by using a `>` rather than a `>=`, which will also save some gas)

```
File: solidity/contracts/CosmosToken.sol   #1

5       uint256 MAX_UINT = 2**256 - 1;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/CosmosToken.sol#L5

## [9] `constant`s should be defined rather than using magic numbers

```
File: solidity/contracts/Gravity.sol   #1

202             bytes32 methodName = 0x636865636b706f696e740000(
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L202

```
File: solidity/contracts/Gravity.sol   #2

433                                             0x7472616e736163
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L433

```
File: solidity/contracts/Gravity.sol   #3

535                             0x6c6f67696343616c6c000000000000(
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L535

## [10] Use a more recent version of solidity

Use a solidity version of at least 0.8.4 to get `bytes.concat()` instead of `abi.encodePacked(<bytes>,<bytes>)` Use a solidity version of at least 0.8.12 to get `string.concat()` instead of `abi.encodePacked(<str>,<str>)`

```
File: solidity/contracts/Gravity.sol   #1
```

```
1   pragma solidity ^0.6.6;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L1

## [11] Variable names that consist of all capital letters should be reserved for `const` / `immutable` variables

If the variable needs to be different based on which class it comes from, a `view` / `pure` *function* should be used instead (e.g. like this).

```
File: solidity/contracts/CosmosToken.sol   #1

5       uint256 MAX_UINT = 2**256 - 1;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/CosmosToken.sol#L5

## [12] Non-library/interface files should use fixed compiler versions, not floating ones

```
File: solidity/contracts/CosmosToken.sol   #1

1   pragma solidity ^0.6.6;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/CosmosToken.sol#L1

```
File: solidity/contracts/Gravity.sol   #2
```

```
1   pragma solidity ^0.6.6;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L1

## [13] Typos

```
File: solidity/contracts/Gravity.sol   #1

564             // Update invaldiation nonce
```

invaldiation https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L564

## [14] File does not contain an SPDX Identifier

```
File: solidity/contracts/CosmosToken.sol   #1

0   pragma solidity ^0.6.6;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/CosmosToken.sol#L0

```
File: solidity/contracts/Gravity.sol   #2

0   pragma solidity ^0.6.6;
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L0

## [15] File is missing NatSpec

```
File: solidity/contracts/CosmosToken.sol (various lines)    #1
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/CosmosToken.sol

```
File: solidity/contracts/Gravity.sol (various lines)    #2
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol

## [16] Event is missing `indexed` fields

Each `event` should use three `indexed` fields if there are three or more fields

```
File: solidity/contracts/Gravity.sol    #1

73      event TransactionBatchExecutedEvent(
74              uint256 indexed _batchNonce,
75              address indexed _token,
76              uint256 _eventNonce
77      );
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L73-L77

```
File: solidity/contracts/Gravity.sol    #2

85      event ERC20DeployedEvent(
86              // FYI: Can't index on a string without doing a
87              string _cosmosDenom,
```

```
88              address indexed _tokenContract,
89              string _name,
90              string _symbol,
91              uint8 _decimals,
92              uint256 _eventNonce
93      );
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L85-L93

```
File: solidity/contracts/Gravity.sol   #3

94      event ValsetUpdatedEvent(
95              uint256 indexed _newValsetNonce,
96              uint256 _eventNonce,
97              uint256 _rewardAmount,
98              address _rewardToken,
99              address[] _validators,
100             uint256[] _powers
101     );
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L94-L101

```
File: solidity/contracts/Gravity.sol   #4

102     event LogicCallEvent(
103             bytes32 _invalidationId,
104             uint256 _invalidationNonce,
105             bytes _returnData,
106             uint256 _eventNonce
107     );
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L102-L107

```
File: solidity/contracts/Gravity.sol   #5

109     event WhitelistedStatusModified(
110             address _sender,
111             address[] _users,
112             bool _isWhitelisted
113     );
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L109-L113

## [17] Consider making the bridge 'pausable'

Having this ability would help to mitigate attacks and would ameleorate the need for this `withdrawERC20()` to be all-or-nothing

```
File: solidity/contracts/Gravity.sol   #1

632     function withdrawERC20(
633             address _tokenAddress)
634             external {
635             require(cudosAccessControls.hasAdminRole(msg.ser
636             uint256 totalBalance = IERC20(_tokenAddress).bal
637             IERC20(_tokenAddress).safeTransfer(msg.sender ,
638     }
```

https://github.com/code-423n4/2022-05-cudos/blob/de39cf3cd1f1e1cf211819b06d4acf6a043acda0/solidity/contracts/Gravity.sol#L632-L638

V-Staykov (Cudos) commented:

> This is particularly high quality.

## Gas Optimizations

For this contest, 33 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **GermanKuber** received the top score from the judge.

*The following wardens also submitted reports:* lllllll, defsec, 0xkatana, Dravee, 0x1f8b, Funen, 0xNazgul, CertoraInc, AlleyCat, slywaters, 0xf15ers, oyc_109, robee, 0xDjango, rfa, peritoflores, 0v3rf10w, WatchPug, ellahi, MaratCerby, simon135, GimelSec, hake, gzeon, delfin454000, ilan, JC, sorrynotsorry, hansfriese, Waze, nahnah, *and* jonatascm.

## [G-01]

In the `sendToCosmos()` function it is not validated that _amount != 0, therefore the state_lastEventNonce could be made to grow only by spending gas. If they go up to type(uint256).max could it cause an overflow and DoS system wide?

## [G-02]

An if could be added inside the for loop to transfer if only the following condition is met if(_destinations[i]!= address(0) && _amounts[i] != 0).

## [G-03]

An if could be added before transferring the fees with if(totalFee != 0).

## [G-04]

An if could be added before transferring the totalBalance with if(totalBalance!= 0).

## [G-05]

Gas is saved if the variable in storage: state_lastValsetNonce is not set to zero, since it is its default value (the tests in remix said a difference of 2246).

## [G-06]

It would save 20,000 gas if instead of using a modifier a view function was used.

## [G-07]

L118/L233/L263/L453/L568/L579/L660 - Instead of using i++, you could use ++i unchecked and save 20,000 gas in 10 iterations.

## [G-08]

L118/233/L263/L453/L568/L579/L660 - It would save 2,000 gas in the for if instead of “uint256 i = 0;” were “uint256 i ;”

## [G-09]

L231 - It would save 2,000 gas in the for if instead of “uint256 cumulativePower = 0;;” were “uint256 cumulativePower;”

## [G-10]

L659 - Gas is saved if the variable in storage: state_lastValsetNonce is not set to zero, since it is its default value (the tests in remix said a difference of 2246).

V-Staykov (Cudos) commented:

> **[G-01]:** Marked it with “disagree with severity” because this is not a gas optimization issue. It seems to be low/mid finding. It is indeed a valid issue, but mitigating it with just checking if the amount is not zero doesn’t seem good, since an attack can then be made with _amount= 1e-18 lets say and still be cheap enough.

> **[G-04]:** Disputed. This seems totally not worth it, since this function is to be used in very rare cases, i.e. changing the contract, and only by admin, who would not do it if he is not sure there are funds worth withdrawing from the contract. That said, adding a check would only cause more gas consumed.

> **[G-06]:** Disputed. This does not describe what it refers to and I personally don’t understand it. It seems not helpful at all.

## Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top