# Gravity Bridge
# Findings & Analysis Report

2021-11-5

## Table of contents

- Overview
    - About C4
    - Wardens
- Summary
- Scope
- Severity Criteria
- High Risk Findings (4)
    - [H-01] Cannot actually submit evidence
    - [H-02] Freeze Bridge via Non-UTF8 Token Name/Symbol/Denom
    - [H-03] Freeze The Bridge Via Large ERC20 Names/Symbols/Denoms
    - [H-04] Large Validator Sets/Rapid Validator Set Updates May Freeze the Bridge or Relayers
- Medium Risk Findings (4)
    - [M-01] The function `updateValset` does not have enough sanity checks
    - [M-02] Crash Eth Oracle On Any `LogicCallEvent`
    - [M-03] Win all relayer rewards

    - [M-04] Incorrect accounting on transfer-on-fee/deflationary tokens in `Gravity`
- Low Risk Findings (10)
    - [L-01] Direct usage of `ecrecover` allows signature malleability
    - [L-02] Why nonces are not incrementing by 1 ?
    - [L-03] logic calls can steal tokens
    - [L-04] Large `ValSets` potentially freezes `Gravity.sol`
    - [L-05] ERC20s that block transfer to particular addresses enable DoS/Censorship
    - [L-06] Downcasting Can Freeze The Chain
    - [L-07] Validations of parameters
    - [L-08] SafeMath library is not always used in `Gravity`
    - [L-09] Possible miner incentive for chain reorgs if `ETHBlockDelay` is too small
    - [L-10] `cumulativePower` check should be inclusive
- Non-Critical Findings (16)
- Gas Optimizations (10)
- Disclosures

## Overview

### About C4

Code 432n4 (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code 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 code contest outlined in this document, C4 conducted an analysis of the Gravity Bridge smart contract system written in Solidity. The code contest took place between August 26—September 8 2021.

## Wardens

14 Wardens contributed reports to the Gravity Bridge code contest:

1. nascent
    - brock
    - 0xAndreas
    - chris_nascent
2. jmak
3. pauliax
4. shw
5. hrkrshnn
6. 0xito
7. ElliotFriedman
8. hack3r-0m
9. JMukesh
10. hickuphh3
11. patitonar
12. defsec

This contest was judged by Albert Chon.

Final report assembled by moneylegobatman and ninek.

## Summary

The C4 analysis yielded an aggregated total of 44 unique findings. All of the issues presented here are linked back to their original submission.

Of these vulnerabilities, 4 received a risk rating in the category of HIGH severity, 4 received a risk rating in the category of MEDIUM severity, and 10 received a risk rating in the category of LOW severity.

C4 analysis also identified 16 non-critical recommendations and 10 gas optimizations.

## Scope

The code under review can be found within the C4 Gravity Bridge code contest repository and is composed of 12 smart contracts written in the Solidity programming language and includes 748 lines of Solidity, 9,540 lines of Rust, and 36,283 lines of Go 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.

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.

## High Risk Findings (4)

### [H-01] Cannot actually submit evidence

*Submitted by jmak*

### Impact

The `SubmitBadSignatureEvidence` is not actually registered in the handler and hence no one can actually submit this message, rendering the message useless. This harms the security model of Gravity since validators have no disincentive to attempt to collude and take over the bridge.

### Proof of Concept

The `SubmitBadSignatureEvidence` handler is omitted from `module` / `x` / `gravity` / `handler.go`

### Tools Used

Visual inspection

### Recommended Mitigation Steps

Handle the `MsgSubmitBadSignatureEvidence` in `module` / `x` / `gravity` / `handler.go`

jkilpatr (Althea) confirmed and patched:

> This was resolved here

> https://github.com/althea-net/cosmos-gravity-bridge/commit/ad6bd78d4c968c3eef5a8ab7a38b42cd3269d186

> This is a valid bug considering this fix is not included in the code hash up for review.

## [H-02] Freeze Bridge via Non-UTF8 Token Name/Symbol/Denom

*Submitted by nascent*

Manual insertion of non-utf8 characters in a token name will break parsing of logs and will always result in the oracle getting in a loop of failing and early returning an error. The fix is non-trivial and likely requires significant redesign.

## Proof of Concept

Note the `c0` in the last argument of the call data (invalid UTF8).

It can be triggered with:

```
    data memory bytes = hex"f79556370000000000000000000000000000000(
    gravity.call(data);
```

The log output is as follows:

```
        ERC20DeployedEvent("atom", "name", ❮utf8 decode failed❯: 0x7
```

Which hits this code path:

```
        let symbol = String::from_utf8(input.data[index_start..index
        trace!("Symbol {:?}", symbol);
        if symbol.is_err() {
            return Err(GravityError::InvalidEventLogError(format!(
                "{:?} is not valid utf8, probably incorrect parsing'
                symbol
            )));
        }
```

And would cause an early return here:

```
    let erc20_deploys = Erc20DeployedEvent::from_logs(&deploys)?;
```

Never updating last checked block and therefore, this will freeze the bridge by disallowing any attestations to take place. This is an extremely low cost way to bring down the network.

## Recommendation

This is a hard one. Re-syncing is permanently borked because, on the Go side, there is seemingly no way to ever process the event nonce because protobufs do not handle non-utf8 strings. The validator would report they need event nonce `N` from the orchestrator, but they can never parse the event `N`. Seemingly, validators & orchestrators would have to know to ignore that specific event nonce. But it is a permissionless function, so it can be used to effectively permanently stop attestations & the bridge until a new `Gravity.sol` is deployed.

One potential fix is to check in the solidity contract if the name contains valid utf8 strings for denom, symbol and name. This likely will be expensive though. Alternatively, you could require that validators sign ERC20 creation requests and perform checks before the transaction is sent.

[jkilpatr (Althea) confirmed](#):

> This is a valid and well considered bug.

> I do disagree about the difficulty of the fix though, if we fail to parse the token name as utf8 we can just encode the bytes themselves in hex and pass that along. The result will be perfectly valid if a little unergonomic.

[albertchon (judge) commented](#):

> Clever, great catch

## [H-03] Freeze The Bridge Via Large ERC20 Names/Symbols/Denoms

*Submitted by nascent*

Ethereum Oracles watch for events on the `Gravity.sol` contract on the Ethereum blockchain. This is performed in the [`check_for_events`](#) function, and run in the [`eth_oracle_main_loop`](#).

In this function, there is [the following code snippet](#):

```
    let erc20_deployed = web3
        .check_for_events(
            starting_block.clone(),
            Some(latest_block.clone()),
            vec![gravity_contract_address],
            vec![ERC20_DEPLOYED_EVENT_SIG],
        )
        .await;
```

This snippet leverages the `web30` library to check for events from the `starting_block` to the `latest_block`. Inside the `web30` library this nets out to calling:

```
    pub async fn eth_get_logs(&self, new_filter: NewFilter) -> Resul
        self.jsonrpc_client
            .request_method(
                "eth_getLogs",
                vec![new_filter],
                self.timeout,
                Some(10_000_000),
            )
            .await
    }
```

The `10_000_000` specifies the maximum size of the return in bytes and returns an error if the return is larger:

```
    let res: Response<R> = match res.json().limit(limit).await {
        Ok(val) => val,
        Err(e) => return Err(Web3Error::BadResponse(format!("Web3 Er
    };
```

This can be triggered at will and keep the loop in a perpetual state of returning the `GravityError::EthereumRestError(Web3Error::BadResponse( "Failed to get logs!".to_string()))` error. To force the node into this state, you just have to deploy ERC20s generated by the public function in `Gravity.sol`:

```
    function deployERC20(
        string memory _cosmosDenom,
        string memory _name,
        string memory _symbol,
        uint8 _decimals
    ) public {
        // Deploy an ERC20 with entire supply granted to Gravity.sol
        CosmosERC20 erc20 = new CosmosERC20(address(this), _name, _s

        // Fire an event to let the Cosmos module know
        state_lastEventNonce = state_lastEventNonce.add(1);
        emit ERC20DeployedEvent(
            _cosmosDenom,
            address(erc20),
            _name,
            _symbol,
            _decimals,
            state_lastEventNonce
        );
    }
```

And specify a large string as the denom, name, or symbol.

If an attacker uses the denom as the attack vector, they save significant gas costing just 256 per additional 32 bytes. For other cases, to avoid gas overhead, you can have the string be mostly 0s resulting in just 584 gas per additional 32 bytes. This leaves it feasible to surpass the 10mb response data in the 6 block buffer. This would throw every ethereum oracle into a state of perpetual errors and all would fall out of sync with the ethereum blockchain. This would result in the batches, logic calls, deposits, ERC20 creations, and `valset` updates to never receive attestations from other validators because their ethereum oracles would be down; the bridge would be frozen and remain frozen until the bug is fixed due to `get_last_checked_block`.

This will freeze the bridge by disallowing attestations to take place.

This requires a patch to reenable the bridge.

## Recommendation

Handle the error more concretely and check if you got a byte limit error. If you did, chunk the search size into 2 and try again. Repeat as necessary, and combine the results.

Additionally, you could require that validators sign ERC20 creation requests.

jkilpatr (Althea) confirmed:

> Excellent bug report.

> I just ran into the buffer limit issue this morning with an Ethereum block. I agree handling this error correctly is essential to long term reliability.

albertchon (judge) commented:

> Nice :)

## [H-04] Large Validator Sets/Rapid Validator Set Updates May Freeze the Bridge or Relayers

*Submitted by nascent*

In a similar vein to “Freeze The Bridge Via Large ERC20 Names/Symbols/Denoms”, a sufficiently large validator set or sufficiently rapid validator update, could cause both the `eth_oracle_main_loop` and `relayer_main_loop` to fall into a state of perpetual errors. In `find_latest_valset`, we call:

```
    let mut all_valset_events = web3
        .check_for_events(
            end_search.clone(),
            Some(current_block.clone()),
            vec![gravity_contract_address],
            vec![VALSET_UPDATED_EVENT_SIG],
        )
        .await?;
```

Which if the validator set is sufficiently large, or sufficiently rapidly updated, continuoussly return an error if the logs in a 5000 (see: `const BLOCKS_TO_SEARCH: u128 = 5_000u128;`) block range are in excess of 10mb. Cosmos hub says they will be pushing the number of validators up to 300 (currently 125). At 300, each log would produce 19328 bytes of data (4*32+64*300). Given this, there must be below 517 updates per 5000 block range otherwise the node will fall out of sync.

This will freeze the bridge by disallowing attestations to take place.

This requires a patch to reenable the bridge.

### Recommendation

Handle the error more concretely and check if you got a byte limit error. If you did, chunk the search size into 2 and try again. Repeat as necessary, and combine the results.

jkilpatr (Althea) confirmed:

> This is a solid report with detailed computations to back it up. I appreciate it and will take actions in our web3 library to prevent this exact scenario.

## Medium Risk Findings (4)

### [M-01] The function `updateValset` does not have enough sanity checks

*Submitted by hrkrshnn, also found by pauliax and shw*

In the updateValset function, the current set of validators adds a new set.

It is missing the check that the combined power of all new validators is above the `state_powerThreshold`. If this is false, then the contract is effectively stuck. Consider adding an on-chain check for this.

It is also worth adding a that the size of the new validator check is less than a certain number.

Here is a rough calculation explaining how 10000 validators (an extreme example) is too much:

1. Let us say that the new set of validators have the property that at least, say, `N` validators are needed to get the total threshold above `state_powerThreshold`.
2. Since each validating signature requires a call to `ecrecover`, costing at least `3000` gas, the minimum gas needed for getting a proposal over `state_powerThreshold` would be `N * 3000`

3. `N * 3000` cannot be more than the `block.gaslimit` Currently, this puts `N` to be less than `10000`

Another approach to solve the above potential problems is to do the updating as a two step process:

1. The current set of validators proposes a pending set of validators.
2. And the pending set of validators need to do the transition to become the new set of validators. Going through the same threshold checks.

This guarantees that the new set of validators has enough power to pass threshold and doesn't have gas limit issues in doing so.

jkilpatr (Althea) confirmed:

> Semi duplicate of #63, #37 which describes the power sum issue

> Also a semi duplicate of #9 which describes the block size issue

> these are both valid and should be assigned congruent severity with the duplicates.

## [M-02] Crash Eth Oracle On Any `LogicCallEvent`

*Submitted by nascent*

Likelihood: high

In `eth_oracle_main_loop`, `get_last_checked_block` is called. Followed by:

```
    let logic_call_executed_events = web3
        .check_for_events(
            end_search.clone(),
            Some(current_block.clone()),
            vec![gravity_contract_address],
            vec![LOGIC_CALL_EVENT_SIG],
        )
        .await;
```

and may hit the code path:

```
    for event in logic_call_executed_events {
        match LogicCallExecutedEvent::from_log(&event) {
            Ok(call) => {
                trace!(
                    "{} LogicCall event nonce {} last event nonce",
                    call.event_nonce,
                    last_event_nonce
                );
                if upcast(call.event_nonce) == last_event_nonce && e
                {
                    return event.block_number.unwrap();
                }
            }
            Err(e) => error!("Got ERC20Deployed event that we can't
        }
    }
```

But will panic at `from_log` here:

```
    impl LogicCallExecutedEvent {
        pub fn from_log(_input: &Log) -> Result<LogicCallExecutedEve
            unimplemented!()
        }
        // snip...
    }
```

It can/will also be triggered here in `check_for_events`:

```
    let logic_calls = LogicCallExecutedEvent::from_logs(&logic_calls
```

Attestations will be frozen until patched.

## Recommendation

Implement the method.

jkilpatr (Althea) confirmed:

> Valid issue, but with zero probability. Since there is nothing on the module side that currently triggers arbitrary logic.

> Despite the fact that it can't currently happen this is still a good report.

## [M-03] Win all relayer rewards

*Submitted by nascent*

"Large Validator Sets/Rapid Validator Set Updates May Freeze the Bridge or Relayer" can affect just the relayers & not affect the oracle in certain circumstances. This could result in valid attestations, but prevent any of the other relayers from being able to participate in the execution. While the other relayers are down from the other attack, the attacker can win all batch, logic, and valset rewards as their node is the only relayer running. This is possible because `find_latest_valset` is run in the main relayer loop and everytime tries for 5000 blocks of logs.

jkilpatr (Althea) confirmed:

> This is a reasonable consequence of #6

> I consider it medium risk because it reduces the number of relayers active, not because of the reward assignment

## [M-04] Incorrect accounting on transfer-on-fee/deflationary tokens in `Gravity`

*Submitted by shw*

### Impact

The `sendToCosmos` function of `Gravity` transfers `_amount` of `_tokenContract` from the sender using the function `transferFrom`. If the transferred token is a transfer-on-fee/deflationary token, the actually received amount could be less than `_amount`. However, since `_amount` is passed as a parameter of the

`SendToCosmosEvent` event, the Cosmos side will think more tokens are locked on the Ethereum side.

### Proof of Concept

Referenced code:

- [Gravity.sol#L535]
- [Gravity.sol#L541]

### Recommended Mitigation Steps

Consider getting the received amount by calculating the difference of token balance (using `balanceOf`) before and after the `transferFrom`.

jkilpatr (Althea) confirmed:

> This is a valid issue, it does present the ability to 'steal' tokens from the bridge, so I think that justifies the severity.

> If user (A) deposits a deflationary token and gets slightly more vouchers than where actually deposited into the bridge upon withdraw they could steal tokens from user (B) who had also deposited.

## Low Risk Findings (10)

### [L-01] Direct usage of `ecrecover` allows signature malleability

*Submitted by shw, also found by 0xito, JMukesh and pauliax*

### Impact

The `verifySig` function of `Gravity` calls the Solidity `ecrecover` function directly to verify the given signatures. However, the `ecrecover` EVM opcode allows malleable (non-unique) signatures and thus is susceptible to replay attacks.

Although a replay attack seems not possible here since the nonce is increased each time, ensuring the signatures are not malleable is considered a best practice (and so is checking `_signer != address(0)`, where `address(0)` means an invalid signature).

## Proof of Concept

Referenced code: Gravity.sol#L153

SWC-117: Signature Malleability SWC-121: Missing Protection against Signature Replay Attacks

## Recommended Mitigation Steps

Use the `recover` function from OpenZeppelin's ECDSA library for signature verification.

jkilpatr (Althea) confirmed:

> Best practicies advice may belong in category zero. But in general I agree with the advice here and that this is valid feedback despite lacking a specific attack vector.

> semi-duplicate of #43, #28 which mention the validation issue. #22 also mentions malleability.

albertchon (judge) commented:

> Marking https://github.com/code-423n4/2021-08-gravitybridge-findings/issues/61 as primary for the signature malleability issue.

> Duplicates:

- https://github.com/code-423n4/2021-08-gravitybridge-findings/issues/21
- https://github.com/code-423n4/2021-08-gravitybridge-findings/issues/22
- https://github.com/code-423n4/2021-08-gravitybridge-findings/issues/43
- https://github.com/code-423n4/2021-08-gravitybridge-findings/issues/28

## [L-02] Why nonces are not incrementing by 1 ?

*Submitted by pauliax, also found by 0xito*

### Impact

I am concerned why `invalidationId`, `invalidationNonce` or `valsetNonce` are only required to be greater than the previous value. Why did you choose this approach instead of just simply asking for an incremented value? While this may not be a problem if the validators are honest, but otherwise, they may submit a nonce of MAX UINT and thus block the whole system as it would be no longer possible to submit a greater value. Again, just wanted you to be aware of this issue, not sure how likely this to happen is in practice, it depends on the honesty of validators so you better know.

### Recommended Mitigation Steps

I didn't receive an answer on Discord so decided to submit this FYI to decide if that's a hazard or no.

jkilpatr (Althea) confirmed:

> This is frankly a good point. We need to be able to skip nonces because we may sometimes create batches that are not profitable or validator sets that we don't want to bother submitting.

> A reasonable mitigation for this issue would be to limit how far ahead we can skip at any one time. Preventing error or hostile action from locking funds in the bridge forever by setting the maximum nonce.

albertchon (judge) commented:

> Well this actually isn't an accurate attack since the nonce representation on the module side is uint64 whereas it's a uint256 on the cosmos side.

> Still, I think this attack is quite hard to trigger naturally since the nonce is incremented by 1 on the module side and as sending 2^64 -1 transactions on a Cosmos chain (to trigger this overflow) would be cost prohibitive/

> And I think attacks that assume validators collude aren't really attacks since that's the natural trust assumption already.

> Marking this one as the primary.

## [L-03] logic calls can steal tokens

*Submitted by Oxito, also found by ElliotFriedman*

### Impact

Attacker can send a logic call that performs a `token.approve(attackerAddress, type(uint256).max)` using the `submitLogicCall` function.

Afterwards, they can steal all tokens from the bridge using `token.safetransferfrom(bridge, attacker, amount)`.

### Proof of Concept

- `submitLogicCall` with `token.approve(attackerAddress, type(uint256).max)`
- call `token.safetransferfrom(bridge, attacker, amount)`

### Recommended Mitigation Steps

Disallow calls to the bridge contract, or to any token/NFT contracts that the bridge owns tokens of ( `token.balanceOf(address(this)) > 0` ).

**jkilpatr (Althea) disputed:**

> I would classify this as low risk at most. Arbitrary logic calls can only be triggered by the Cosmos module itself with full consensus, the ability of arbitrary logic to do unknown dangerous things is the design intent and any call actually deployed would have to have the upmost inspection before being used.

> duplicate of #1

**albertchon (judge) commented:**

> Agreed on the low risk classification. Perhaps the trust assumptions of the model should've been made more clear.

> Duplicate of https://github.com/code-423n4/2021-08-gravitybridge-findings/issues/1

jkilpatr (Althea) commented:

> I agree, arbitrary logic could be better documented. But it's also very clear in the existing code that there's no way to create arbitrary logic transactions as a user.

## [L-04] Large `ValSets` potentially freezes `Gravity.sol`

*Submitted by nascent, also found by hack3r-0m, and pauliax*

Gas requirements of `makeCheckpoint`: If the size of the validator set grows large enough during a time of block-size expansion, it may be possible to make the validator set large enough that, when the block size shrinks, the gas required to perform `makeCheckpoint` may be larger than the amount of gas available in the block. In that case, the validator set could not be updated until the block size increased. If a reduction in upper gas limit for blocks occurs at the miner layer, it may be bricked permanently.

jkilpatr (Althea) acknowledged

## [L-05] ERC20s that block transfer to particular addresses enable DoS/Censorship

*Submitted by nascent*

Tokens that prevent transfers to particular addresses (most commonly `address(0)` as is the OpenZeppelin standard) enables DoS against a batch. If the attacker submits the bad transaction, the relayer wont submit the batch. The attacker never has to worry about the transaction being submitted and paying the fee because the transaction will fail, leaving the relayer stuck with the bill. This can enable MEV between chains by disabling others' ability to close arbitrage between chains by denying them their transfers off the chain.

jkilpatr (Althea) acknowledged:

> The relayer will not actually pay the bill, since we simulate the tx before submission. That being said this is a valid way to block a batch for long enough that it times out.

> I would describe this as low risk. Since it doesn't compromise the bridge or lose tokens, just potential value from arbitrage.

> The correct solution here is to block invalid transactions from being added to batches on the Cosmos side. (which I just checked we do not block the zero address in MsgSendToEth)

## [L-06] Downcasting Can Freeze The Chain

*Submitted by nascent*

The function `utils::downcast_uint256() -> Option<u64>` returns `None` if the input value is greater than `U64MAX`. If the value being downcast is read from a contract (e.g. a nonce), and the contract could be put into such a state where a `Uint256` is set to higher value, this will cause all nodes to halt execution upon reading this value, requiring a patch to reenable the bridge.

### Recommendation

Change the signature of `downcast_uint256()` to return a `Result<>`, and/or remove any `unwrap()` s of the result.

jkilpatr (Althea) acknowledged:

> This is valid, dealing with nonces as big-ints is something of a pain and it's reasonable to not expect these values to go over u64 max. I believe with nonce increase limitations as described in #32 this can be mitigated.

albertchon (judge) commented:

> Low risk since this is very costly/impractical to make happen

## [L-07] Validations of parameters

*Submitted by pauliax*

### Impact

There are a few validations that could be added to the system: the constructor could check that `_gravityId` is not empty. `state_powerThreshold` should always be greater than 0, otherwise, anyone will be available to execute actions.

### Recommended Mitigation Steps

Consider implementing suggested validations.

[jkilpatr (Althea) confirmed](#):

> These are good suggestions. In my opinion powerThreshold should probably just be hard coded at this point. GravityID being empty is not a vulnerability I had considered before.

## [L-08] SafeMath library is not always used in `Gravity`

*Submitted by shw, also found by pauliax*

### Impact

SafeMath library functions are not always used in the `Gravity` contract's arithmetic operations, which could cause integer underflow/overflows. Using SafeMath is considered a best practice that could completely prevent underflow/overflows and increase code consistency.

### Proof of Concept

Referenced code:

- [Gravity.sol#L202](#)
- [Gravity.sol#L586](#)

### Recommended Mitigation Steps

Consider using the SafeMath library functions in the referenced lines of code.

jkilpatr (Althea) confirmed:

> An overflow in the powers would be a significant bug, while it would require some pretty dramatic issues no the go module side there is value in checking in. I agree with the severity

> duplicate of #38

## [L-09] Possible miner incentive for chain reorgs if `ETHBlockDelay` is too small

### Impact

If `ETHBlockDelay` is too small and the incentive for miners is large enough, it would be profitable for miners to attempt to double spend by depositing assets, waiting for confirmation on the cosmos-SDK and then reorging the blockchain.

Although an attack like this has never been done, it could potentially cost hundreds of millions of dollars in damages. With MEV at all time highs and miners regularly using custom geth implementations its not totally out of the question to see an attack similar to this happening soon.

### Recommended Mitigation Steps

The best way to avoid something like this is to make sure to wait for a large number of blocks until a transaction is confirmed by the cosmos system.

jkilpatr (Althea) acknowledged:

> We currently wait 6 blocks, as noted here I've done some math on the subject. A 7 block deep reorg would actually halt the bridge so they could only pull this off once.

> I do agree it's a moderate risk, but computing how probable (and therefore risky) this sort of attack is requires info that's pretty hard to get.

albertchon (judge) commented:

> We did some investigation into this and concluded that 6 blocks was safe enough

## [L-10] `cumulativePower` check should be inclusive

*Submitted by pauliax*

### Impact

Based on my understanding `cumulativePower` checks should be inclusive to indicate when the threshold is met. Otherwise, there might be impossible to reach it in certain cases (e.g. when 100% power is required). Replace `>` with `>=` in constructor and function `checkValidatorSignatures`:

```
    if (cumulativePower > \_powerThreshold) {
        break;
    }
    require(
        cumulativePower > \_powerThreshold,
        "Submitted validator set signatures do not have enough powe
    );
```

### Recommended Mitigation Steps

`cumulativePower >= \_powerThreshold`

jkilpatr (Althea) acknowledged:

> I would classify this as low risk since the bridge would never in any sane situation be configured to require 100% of the power. It's a valid report in the context that a slightly more permissive check could save the day in very specific situations.

## Non-Critical Findings (16)

- [N-01] validator set can be updated with same set
- [N-02] Unhandled reverts from Cosmos to Eth batches can cause *Denial Of Service*
- [N-03] Filter Logic calls to gravity cosmos at client level to avoid reverts
- [N-04] Gravity: Consider enforcing validation expiry on-chain

- [N-05] Actions can be frontrunned
- [N-06] Anyone can deploy ERC20 tokens
- [N-07] use of floating pragma
- [N-08] Passing by ownership instead of borrowing
- [N-09] `Vec::new()` instead of `Iterator::collect()`
- [N-10] Anti-pattern `is_err()`, `return`, then `.unwrap()`
- [N-11] Panics as error-handling
- [N-12] Style issues
- [N-13] Does the cosmos-sdk listen to only 1 gravity.sol contract address?
- [N-14] Lack of Validation Check
- [N-15] Consider adding a token whitelist in `sendToCosmos` function
- [N-16] The gravity.sol router should have pause/unpause functionality.

## Gas Optimizations (10)

- [G-01] Smart Contract Gas Optimization
- [G-02] Avoid long revert strings.
- [G-03] Upgrade to at least Solidity 0.8.4
- [G-04] Caching the length in for loops
- [G-05] Use `calldata` instead of `memory` for function parameters
- [G-06] State Variables that can be changed to `immutable`
- [G-07] powers in a decreasing order
- [G-08] Skip functionCall when the payload is empty
- [G-09] Cache values
- [G-10] Pack structs tightly

## Disclosures

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