BLOCKAPEX

SMART CONTRACT SECURITY

v 1.0

Date: 03-06-2025

Prepared For: Light Link

About BlockApex

Founded in early 2021 BlockApex is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

Contents

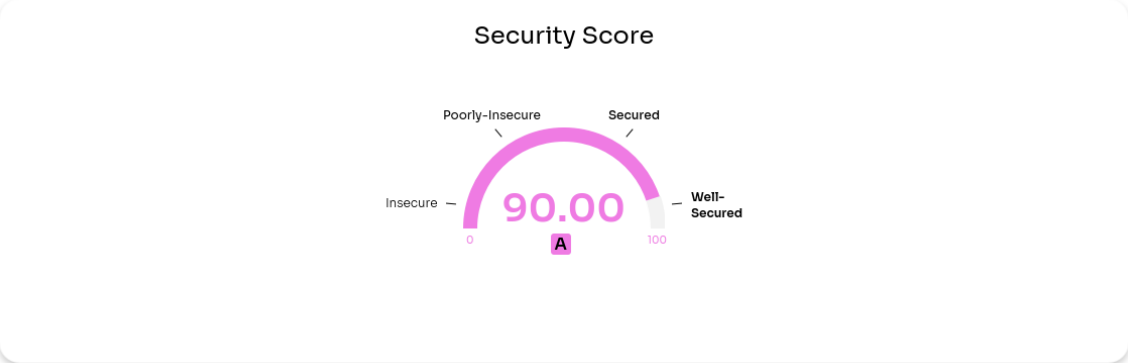
1 Executive Summary

Toucan is a cross-chain governance framework built on top of LayerZero messaging infrastructure and integrates tightly with the Aragon OSx plugin system for DAO management.

It enables decentralized autonomous organizations (DAOs) to bridge governance tokens and voting rights seamlessly across multiple blockchain networks, separating execution chains from voting chains.

Toucan leverages Aragon's modular DAO architecture to deploy and manage governance components, while LayerZero handles the secure cross-chain messaging.

The core idea is to allow a DAO to maintain governance on a cheaper, faster L2 (voting chain) while executing proposals on a more secure, canonical L1 (execution chain), thereby optimizing both cost efficiency and security.


Security Score
Poorly-Insecure
Secured
Insecure
90.00
Well-
Secured
0
100
A

1.1 Scope

1.1.1 In Scope

1. **Audit Repository:** 01b378592dd4c5e71b0d0d0aa490d61f8eebf85c
2. **Fixed Audit Repository:** https://github.com/aragon/toucan-voting-plugin/pull/2
3. **Review Type:** Comprehensive White-box Code Review
4. **Initiation Date:** Friday, 14th March, 2025
5. **Initial Report Delivery Date:** Monday, 7th April, 2025
6. **Final Report Delivery data:** Tuesday, 3rd june , 2025

1.1.2 Out of Scope

All features or functionalities not delineated within the “In Scope” section of this document shall be deemed outside the purview of this audit.

1.2 Methodology

The codebase was audited using a filtered audit technique. A band of three (3) auditors scanned the codebase in an iterative process for a time spanning 2.5 weeks. Starting with the recon phase, a basic understanding was developed, and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews to find logical flaws in the codebase complemented with code optimizations,software, and security design patterns, code styles and best practices

Project Overview

Taucan is a cross-chain governance framework built on top of LayerZero messaging infrastructure and integrates tightly with the Aragon OSx plugin system for DAO management.

It enables decentralized autonomous organizations (DAOs) to bridge governance tokens and voting rights seamlessly across multiple blockchain networks, separating execution chains from voting chains.

Taucan leverages Aragon's modular DAO architecture to deploy and manage governance components, while LayerZero handles the secure cross-chain messaging.

The core idea is to allow a DAO to maintain governance on a cheaper, faster L2 (voting chain) while executing proposals on a more secure, canonical L1 (execution chain), thereby optimizing both cost efficiency and security.

1.3 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.4 Summary of Findings Identified

S.NO	SEVERITY	FINDINGS	STATUS
#1	HIGH	Allowing Non-Changing Voting Modes Enables Denial of Service in Cross-Chain Voting	FIXED
#2	MEDIUM	Stale LayerZero Fee Quotes in Cross-Chain Proposal Execution	FIXED
#3	LOW	No veto period allows whales to create, vote and execute proposals instantly	ACKNOWLEDGED
#4	LOW	Extra and enforced options for lzReceive gas limit not handled properly	ACKNOWLEDGED
#5	LOW	User Excessive Fees Loss in ToucanRelay's dispatchVotes Function Due to Refund Flow Design	ACKNOWLEDGED
#6	LOW	Anyone can create proposals by bypassing hasEnoughVotingPower with flash loans	FIXED
#7	LOW	Enforce msg.value in _lzReceive	ACKNOWLEDGED
#8	INFO	If all tokens are bridged (locked in adapter), no proposals can get created	ACKNOWLEDGED
#9	INFO	Missing Action Count Validation in Proposal Creation	ACKNOWLEDGED
#10	INFO	Unrestricted Buffer Configuration in Relay Contract May Disrupt Voting Process	ACKNOWLEDGED
#11	INFO	minProposerVotingPower Can Be Set to Zero, Allow Proposal Spamming	ACKNOWLEDGED
#12	INFO	Voters Can Flip Results by Changing Their Vote at the Last Minute	ACKNOWLEDGED

2 Findings and Risk Analysis

2.1 Allowing Non-Changing Voting Modes Enables Denial of Service in Cross-Chain Voting

Severity: High

Status: Fixed

Location:

1. `src/execution-chain/voting/ToucanVoting.sol`
2. `src/execution-chain/crosschain/ToucanReceiver.sol`
3. `src/voting-chain/crosschain/ToucanRelay.sol`

Description: In the Taucan cross-chain governance system, the voting mode is initially enforced to be VoteReplacement, which allows updating votes across chains safely. However, after deployment, the updateVotingSettings function in ToucanVoting allows authorized entities (such as the DAO) to change the voting mode at any time, including switching it to Standard mode.

When the voting mode is set to Standard, users are only allowed to vote once per proposal. If a user's votes are submitted early through the cross-chain bridge and processed on the execution chain, it prevents all other users from voting on that proposal. This occurs because the vote() function in ToucanVoting checks if a user has already voted and blocks any further votes in Standard mode.

The problem arises when a malicious user votes and immediately dispatches their votes across the bridge. When a legit user dispatches vote , the ToucanReceiver tries to submitVotes() , it fails leading to SubmitVoteFailed event emission. Once processed, this prevents other legitimate voters from participating, leading to a denial-of-service (DoS) on the voting process.

Impact:

- Legitimate users may be permanently blocked from voting on proposals.
- Proposals could pass or fail based on the first dispatched votes without community participation.
- Undermines the reliability and fairness of the governance process.
- Creates a systemic risk if governance parameters are changed without strict controls.

Proof of Concept: The end-to-end test provided demonstrates the attack:

- Tokens are bridged.
- The voting mode is changed from VoteReplacement to Standard.
- A proposal is created.
- A malicious user votes and dispatches immediately.

- Subsequent users attempting to vote and dispatch fail

```
function testE2E() public {
        // reset clocks
        vm.warp(1);
        vm.roll(1);

        // initialize the chains
        ExecutionChain memory e = setupExecutionChain();
        VotingChain memory v = setupVotingChain();

        // multisig requires that you have one block between
        // changing settings and proposal creation
        vm.roll(2);

        // deploy layerZero: this is cross chain
        _deployLayerZero(e.base, v.base);

        // setup the execution chain contracts
        _prepareSetupToucanVoting(e);
        _prepareSetupReceiver(e);

        // setup the voting chain contracts
        _prepareSetupRelay(v, e);
        _prepareSetupAdminXChain(v);

        // apply the installations and set the peers

        // exec chain
        _applyInstallationsSetPeersRevokeAdmin(e, v);

        // voting chain
        _applyInstallationsSetPeersRevokeAdmin(v, e);
Audit Draft [For internal use only]


        // give the agents cash money
        vm.deal(e.voter, initialDeal);
        vm.deal(e.voter2, initialDeal);

        vm.deal(v.voter, initialDeal);
        vm.deal(v.voter2, initialDeal);

        vm.deal(e.base.deployer, initialDeal);
        vm.deal(v.base.deployer, initialDeal);

        _addLabels(e, v);

         console2.log(&quot;e.voter2  governance  token&quot;,
e.token.balanceOf(e.voter2));
        // first we want a user to bridge
        _bridgeTokens(e, v);
        // then we want to create a proposal
        // _createProposal(e, v);
        // then do a vote
        // remote
        // bridge back

        console2.log(&quot;line 138&quot;);
        // _voteAndDispatch(e, v);

        // update the voting settings
        e.voting.updateVotingSettings(IToucanVoting.VotingSettings({
            minParticipation: 0,
```

```
            supportThreshold: 0,
            minProposerVotingPower: 0,
            minDuration: 360000,
            votingMode: IToucanVoting.VotingMode.Standard
        }));
         _createProposal(e, v);

        console2.log(&quot;Current Voting Mode&quot;, uint(e.voting.votingMode()));
       // _createProposal2(e, v);


        vm.roll(200);
Audit Draft [For internal use only]

        vm.warp(200);
        _voteAndDispatch(e, v, proposalRef);
        vm.roll(210);
        vm.warp(210);
        _voteAndDispatch(e, v, proposalRef);

        // vm.startPrank(e.voter2);
         //  e.voting.vote(proposalId,  Tally(0,  transferAmount  /  2,  0),
false);
        //  vm.stopPrank();

        // Vote on Execution Chain

        // _voteAndDispatch(e, v, proposalRef2);
        // _voteAndDispatch2(e, v, proposalRef2);
    }
```

Recommendation: :

- Restrict updateVotingSettings to disallow changing the voting mode away from VoteReplacement.
- Allow ToucanReciever to call the ToucanVoting.vote() repeatedly:

```
if  (!proposal_.voters[_account].isZero()  &amp;&amp;  msg.sender  ==  ToucanReciver
&amp;&amp;
    proposal_.parameters.votingMode != VotingMode.VoteReplacement) {
    return false;
}
```

2.2 Stale LayerZero Fee Quotes in Cross-Chain Proposal Execution

Severity: Medium

Status: Fixed

Location:

1. `src/execution-chain/voting/ToucanVoting.sol`
2. `src/execution-chain/crosschain/ActionRelay.sol`

Description: The current implementation of the cross-chain proposal system calculates and stores LayerZero transaction fees at proposal creation time. However, these fees may become invalid (stale) during the proposal's voting period, potentially causing proposal execution to fail or waste DAO funds when the proposal is finally executed.

When creating a proposal , a fee quote is fetched via ActionRelay.quote(). Which is then passed into the actions.

The issue arises from the fact that DAO proposals often have lengthy voting periods (days, weeks, or even months). During this period, various factors can cause the originally quoted fee to become invalid. When the proposal is finally executed after passing governance, the DAO.execute() function will attempt to execute the action with the outdated fee value.

Impact

If the actual required fee at execution time is higher than the quoted fee, the LayerZero message will fail to be delivered, causing the entire proposal execution to fail. This could block the desired actions. The impact is totally dependent on the type of actions being done when the proposal is finalized and executions need to be done

Recommendation: :

- **Dynamic Fee Quoting:** Modify the ActionRelay contract to retrieve a fresh quote at execution time rather than using the stored quote from proposal creation.

- **Fee Buffer System:** Implement a system that includes a substantial buffer in the fee calculation (e.g., 150-200% of the quoted fee) and returns any unused funds.
- **Fee Update Mechanism:** Add functionality that allows updating the fee for a pending proposal before execution.

2.3 No veto period allows whales to create, vote and execute proposals instantly

Severity: Low

Status: Acknowledged

Location:

1. `src/execution-chain/voting/ToucanVoting.sol`

Description: The ToucanVoting contract lacks a veto or review period, allowing large token holders (whales) to create, vote, and immediately execute proposals in a single transaction.

This behavior is enabled by the _tryEarlyExecution parameter present in both the createProposal and vote functions.

In the createProposal function, if the creator supplies sufficient voting power and sets _tryEarlyExecution to true, the proposal can be executed instantly after creation, without waiting for the full voting period to pass.

Additionally, if the voting mode is set to allow early execution, a whale could create a proposal, vote for it, and execute it in the same transaction.

Even if early execution is not enabled, a whale could still create proposals with a very short minimum duration (e.g., 1 hour) and immediately satisfy the support and participation thresholds, allowing rapid execution before other participants have time to react.

This design can centralize decision-making power in the hands of large token holders and undermines the security and decentralization expected in a DAO governance process.

```
if (!_votes.isZero()) {
    vote(proposalId, _votes, _tryEarlyExecution);
}
```

Similarly, in the `vote` function, if a voter sets `_tryEarlyExecution` to true and the proposal meets the execution criteria, it will be executed immediately:

```
if (_tryEarlyExecution &amp;amp;&amp;amp; _canExecute(_proposalId)) {
    _execute(_proposalId);}
```

The `_canExecute` function allows early execution if:

1. The proposal is in `EarlyExecution` mode
2. The support threshold is reached early
3. The minimum participation is reached

```
if (_isProposalOpen(proposal_)) {
    // Early execution
    if (proposal_.parameters.votingMode != VotingMode.EarlyExecution) {
```

```
            return false;
        }
        if (!isSupportThresholdReachedEarly(_proposalId)) {
            return false;
        }
    } else {
        // Normal execution
        if (!isSupportThresholdReached(_proposalId)) {
            return false;
        }
    }
    if (!isMinParticipationReached(_proposalId)) {
        return false;
    }
```

Impact

- Whales can create, vote, and execute proposals in a single transaction, bypassing the intended governance process
- Other token holders have no opportunity to review or vote on proposals before they are executed
- This undermines the democratic nature of the governance system and could lead to malicious proposals being executed without community input

Recommendation: :

1. Implement a mandatory veto period between proposal creation and execution
2. Disable early execution for critical proposals that could significantly impact the protocol
3. Add a minimum voting duration that cannot be bypassed, even with early execution, Consider implementing a two-phase voting system where:

- Phase 1: Proposal creation and voting
- Phase 2: Execution after a mandatory waiting period

Developer Response

This is a design decision carried over from Aragon's existing token voting plugin. We opted not to change it.

2.4 Extra and enforced options for lzReceive gas limit not handled properly

Severity: Low

Status: Acknowledged

Location:

1. `src/voting-chain/crosschain/ToucanRelay.sol`
2. `src/execution-chain/crosschain/ActionRelay.sol`

Description: quote function in ToucanRelay and ActionRelay implements the OptionBuilder to add the gasLimit. This value represents the amount of gas to be used in the lzReceive call by the Executor on the destination chain. On the other hand, the gas limit for the executor in the destination chain can also be set by the owner as an enforce option, which is recommended in the integration checklist .

There are two possibilities:

1. The protocol sets the enforced options In this case, the caller of deposit can be charged twice for the gas limit. We can see the following warnings in the documentation:

CAUTION: When setting enforcedOptions, try not to unintentionally pass a duplicate _options argument to extraOptions. Passing identical _options in both enforcedOptions and extraOptions will cause the protocol to charge the caller twice on the source chain, because LayerZero interprets duplicate _options as two separate requests for gas.

CAUTION: As outlined above, decide on whether you need an application wide option via enforcedOptions or a call specific option using extraOptions. Be specific in what _options you use for both parameters, as your transactions will reflect the exact settings you implement.

2. The protocol does not set the enforced options:
   In this case the gas estimation is performed via the quote function.

In the current testcase , enforce option is not implemented but defined in the code.

Recommendation: It is recommended to follow the layerzero checklist implementation and implement the enforced options.

2.5 User Excessive Fees Loss in ToucanRelay's dispatchVotes Function Due to Refund Flow Design

Severity: Low

Status: Acknowledged

Location:

1. src/voting-chain/crosschain/ToucanRelay.sol

Description: The dispatchVotes function in the ToucanRelay contract allows any user to dispatch proposal votes across chains, but it has a design flaw in its refund handling. Excess funds from the transaction are sent to a hardcoded refundAddress (the peer contract on the destination chain) rather than back to the caller. These funds can only be recovered through the Sweeper contract's functions, which are restricted to the DAO. This design can lead to permanent loss of user excessive fees if a user calls with excessive funds. Since only Dao is able to get the funds back via sweeper.

```
function dispatchVotes(
       uint256 _proposalRef,
       LzSendParams memory _params
   ) external payable nonReentrant returns (MessagingReceipt memory receipt) {
       // check if we can dispatch the votes
       (bool success, ErrReason e) = canDispatch(_proposalRef);
       if (!success) revert CannotDispatch(_proposalRef, e);

       // get the votes and encode the message
       Tally memory proposalVotes = _proposals[dstEid][_proposalRef].tally;
       bytes memory message = abi.encode(
           ToucanVoteMessage({
               votingChainId: _chainId(),
               proposalRef: _proposalRef,
               votes: proposalVotes
           })
       );

       // refund should be somewhere safe on the dst chain
       address refund = refundAddress(dstEid);
]       // dispatch the votes via the lz endpoint
       receipt = _lzSend(dstEid, message, _params.options, _params.fee, refund);

       emit VotesDispatched(dstEid, _proposalRef, proposalVotes, receipt);
   }
```

Impact Regular User will lose its excessive fee amount, since only dao can recover it.

Recommendation: Amend the dispatchVotes as such that a check is performed that if the dao is not calling the dispatchVotes then the refundAddress is “msg.sender” and if not then hardcoded refund address is used.

Developer Response

CREATE2 contracts may not implement receive or fallback functions on the destination chain.DAO with sweeper guarantees funds can always be recovered and refunded.

2.6 Anyone can create proposals by bypassing hasEnoughVotingPower with flash loans

Severity: Low

Status: Fixed

Location:

1. `src/execution-chain/voting/ToucanVoting.sol`

Description: The `hasEnoughVotingPower` function in `ToucanVoting.sol` can be bypassed using flash loans, allowing anyone to create proposals regardless of the `minProposerVotingPower` setting. The function checks both the voting power and token balance of the caller:

```
function hasEnoughVotingPower(address _who) public view returns (bool) {
    uint256 minProposerVotingPower_ = minProposerVotingPower();

    if (minProposerVotingPower_ != 0) {
        if (
            votingToken.getVotes(_who) &lt; minProposerVotingPower_ &amp;amp;&amp;amp;
             IERC20Upgradeable(address(votingToken)).balanceOf(_who)  &lt;
minProposerVotingPower_
        ) {
            return false;
        }
    }
    return true;
}
```

This check is used in the `createProposal` function to determine if a user can create a proposal:

```
if (!hasEnoughVotingPower(_msgSender())) {
    revert ProposalCreationForbidden(_msgSender());
}
```

However, an attacker can bypass this check by:

1. Taking a flash loan to obtain the required amount of tokens
2. Creating a proposal while holding the tokens
3. Repaying the flash loan in the same transaction

Impact: Anyone can create proposals regardless of the `minProposerVotingPower` setting

Recommendation: Implement a time-weighted token balance check that requires tokens to be held for a minimum period before allowing proposal creation

2.7 Enforce msg.value in _lzReceive

Severity: Low

Status: Acknowledged

Location:

1. `src/execution-chain/crosschain/ToucanReceiver.sol`
2. `src/execution-chain/crosschain/ActionRelay.sol`

Description: The `_lzReceive` functions in the cross-chain voting implementation do not check the `msg.value` sent with the transaction, which could lead to unexpected behavior or potential security issues. As per the layer-zero integration checklist , enforcement of check on msg.value on sending and receiving side should be done

"If you specify in the executor _options a certain msg.value, it is not guaranteed that the message will be executed with these exact parameters because any caller can execute a verified message.

In certain scenarios depending on the encoded message data, this can result in a successful message being delivered, but with a state change different than intended. Encode the msg.value inside the message on the sending chain, and then decode it in the lzReceive or lzCompose and compare with the actual msg.value."

In `ToucanReceiver.sol`, the `_lzReceive` function does not check the `msg.value`:

```
function _lzReceive(
    Origin calldata _origin,
    bytes32 _guid,
    bytes calldata _message,
    address _executor,
    bytes calldata _extraData
) internal virtual override {
    // Process the message without checking msg.value
    // ...
}
```

Similarly, in `ActionReceiver.sol`, the same issue exists:

```
function _lzReceive(
    Origin calldata _origin,
Audit Draft [For internal use only]

    bytes32 _guid,
    bytes calldata _message,
    address _executor,
    bytes calldata _extraData
) internal virtual override {
    // Process the message without checking msg.value
    // ...
}
```

Recommendation: Implement `msg.value` validation in the `_lzReceive` as

2.8 If all tokens are bridged (locked in adapter), no proposals can get created

Severity: Info

Status: Acknowledged

Location:

1. `src/execution-chain/voting/ToucanVoting.sol`
2. `src/execution-chain/crosschain/GovernanceOFTAdapter.sol`

Description: The governance system has an extreme edge case scenario , where if all tokens are bridged (locked in the adapter), no proposals can be created. This occurs because the `hasEnoughVotingPower` function in `ToucanVoting.sol` checks both the voting power and token balance of the caller:

```
function hasEnoughVotingPower(address _who) public view returns (bool) {
    uint256 minProposerVotingPower_ = minProposerVotingPower();

    if (minProposerVotingPower_ != 0) {
        if (
            votingToken.getVotes(_who) &lt; minProposerVotingPower_ &amp;&amp;
             IERC20Upgradeable(address(votingToken)).balanceOf(_who)  &lt;
minProposerVotingPower_
        ) {
            return false;
        }
    }
    return true;
}
```

When tokens are bridged from the execution chain to the voting chain, they are locked in the adapter contract. If all tokens are bridged, there will be no tokens left on the execution chain to meet the `minProposerVotingPower` requirement for creating proposals

Impact: The governance system can be completely paralyzed if all tokens are bridged to the voting chain

Recommendation: :

- Implement a mechanism to ensure a minimum amount of tokens remain on the execution chain
- Allow proposal creation from voting chains

Developer Response

Noted, seems unlikely as the execution chain one would expect holds some significance for the DAO and team. People can always bridge back.

2.9 Missing Action Count Validation in Proposal Creation

Severity: Info

Status: Acknowledged

Location:

1. src/execution-chain/voting/ToucanVoting.sol

Description: The ToucanVoting.createProposal function does not validate that the number of actions in a proposal is less than or equal to the maximum number of actions supported by the DAO contract (256). This discrepancy allows users to create and vote on proposals that may be impossible to execute due to exceeding the action limit.

The DAO contract defines a constant maximum limit for the number of actions that can be included in a proposal:

```
/// @notice The internal constant storing the maximal action array length.
uint256 internal constant MAX_ACTIONS = 256;
```

However, when examining the ToucanVoting.createProposal function, there is no validation to ensure that the number of actions submitted adheres to this limit:

```
function createProposal(
    bytes calldata _metadata,
    IDAO.Action[] calldata _actions,
    uint256 _allowFailureMap,
    uint32 _startDate,
    uint32 _endDate,
    Tally memory _votes,
    bool _tryEarlyExecution
) external returns (uint256 proposalId) {
    // ... code ...

    for (uint256 i; i &lt; _actions.length; ) {
        proposal_.actions.push(_actions[i]);
        unchecked {
            ++i;
        }
    }
      // ... code ...
}
```

Recommendation: Add validation in the createProposal function to ensure that the number of actions does not exceed the DAO's limit. It ensures that a valid proposal is added and resources are not wasted on voting for a bad proposal as it will fail during execution

2.10 Unrestricted Buffer Configuration in Relay Contract May Disrupt Voting Process

Severity: Info

Status: Acknowledged

Location:

1. `src/voting-chain/crosschain/ToucanRelay.sol`

Description: In the ToucanRelay contract, the buffer parameter controls the minimum time required before a proposal closes to allow safe cross-chain vote dispatching. It is intended to account for LayerZero message delays and ensure that votes can still be processed before the proposal ends.

However, the _setBridgeDelayBuffer function does not enforce any validation on the buffer value. This means the DAO can unintentionally (or maliciously) set the buffer to a value that is either too small (e.g., 0) or excessively large (e.g., several days).

- If the buffer is set too small, there may not be enough time to bridge and submit votes before a proposal closes, especially in high-latency cross-chain environments, leading to votes being lost.
- If the buffer is set too large, users could be prevented from voting even though the proposal remains open, effectively shortening the usable voting window and disrupting normal governance participation.

Because setting the buffer is controlled via DAO permissions, a misconfiguration or a malicious DAO action could impact the availability and fairness of the voting process

Recommendation: Enforce validation on the buffer value during setBridgeDelayBuffer() by introducing reasonable bounds

Developer Response

The buffer is kept unopinionated simply because the bridging infrastructure is evolving. At this time we rely on operators or the DAO itself to set sensible buffers.

2.11 minProposerVotingPower Can Be Set to Zero, Allow Proposal Spamming

Severity: Info

Status: Acknowledged

Location:

1. src/execution-chain/voting/ToucanVoting.sol

Description: The ToucanVoting contract allows the DAO to set minProposerVotingPower (the minimum token threshold required to create a proposal) through updateVotingSettings(). However, there's no lower bound enforced — it can be set to zero. This means anyone, even with zero voting power, can create proposals if this setting is zero.

The hasEnoughVotingPower() function explicitly skips all checks when minProposerVotingPower is zero, making proposal creation permissionless, regardless of token ownership or delegation.

```
function hasEnoughVotingPower(address _who) public view returns (bool) {
    uint256 minProposerVotingPower_ = minProposerVotingPower();

    if (minProposerVotingPower_ != 0) {
        // checks if user meets voting power requirement
        ...
    }
    return true; // &lt;-- if it&#x27;s 0, any user is allowed
}
```

Impact

If minProposerVotingPower is set to zero, accidentally or intentionally, this means any user could:

- Spam the DAO with endless proposals.
- Submit fake or invalid proposals to clog the system or confuse voters.
- Poison the proposal pool, overwhelming voters or disrupting governance UX.

This weakens governance integrity and can lead to degraded participation and trust

Recommendation: Enforce a non-zero minimum proposer voting power in updateVotingSettings() or restrict proposal creation to users with at least 1 token

2.12 Voters Can Flip Results by Changing Their Vote at the Last Minute

Severity: Info

Status: Acknowledged

Location:

1. src/voting-chain/crosschain/ToucanRelay.sol

Description: The vote() function in the ToucanRelay contract allows users to change their vote at any time before the proposal ends, even in the very last block. This opens the door for large token holders to first vote in favor of a proposal to influence community sentiment, then quietly switch their vote to "no" or "abstain" just before the deadline. This kind of behavior can flip the outcome or cancel quorum, especially in low-participation votes.

```
function vote(uint256 _proposalRef, Tally calldata _voteOptions) external nonReentrant {
       // check that the user can actually vote given their voting power and the proposal
       (bool success, ErrReason e) = canVote(_proposalRef, msg.sender, _voteOptions);
       if (!success) revert CannotVote(_proposalRef, msg.sender, _voteOptions, e);

       // get the proposal data
       Proposal storage proposal = _proposals[dstEid][_proposalRef];
       Tally storage lastVote = proposal.voters[msg.sender];

       // revert the last vote, doesn&#x27;t matter if user hasn&#x27;t voted before
       proposal.tally = proposal.tally.sub(lastVote);

       // update the total vote
       proposal.tally = proposal.tally.add(_voteOptions);

       // update the last vote
       // we have to set by item due to no implicit storage casting
       lastVote.abstain = _voteOptions.abstain;
       lastVote.yes = _voteOptions.yes;
       lastVote.no = _voteOptions.no;

       emit VoteCast(dstEid, _proposalRef, msg.sender, _voteOptions);
   }
```

Recommendation: Introduce a vote freeze buffer, a short period before the proposal ends (e.g. last 10 minutes) where vote changes are no longer allowed. This discourages last-minute manipulation while still giving users flexibility during most of the voting window

Reference: Cyfrin Governance Attacks

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts