coinspect

You build, we defend.

Liquity V2

Smart Contract Audit

Bold Governance

Jan, 2025

Governance V2

Smart Contract Audit

Version: v250120 | Prepared for: Liquity | January 2025

Security Assessment

1. Executive Summary

In **January 2025**, **Liquity** engaged Coinspect to perform a smart contract audit of Liquity's V2 Governance.

The objective of the project was to evaluate the security of the smart contracts that implement the second version of the protocol's governance system.

The code reviewed enables decentralized governance through a modular initiative-based system, allowing users to register initiatives, vote, and manage their `LQTY` token allocations. Liquity's V2 Governance leverages an epoch-based framework to balance broadness with resilience against manipulation. Incentives, such as bribe-driven initiatives, promote active participation while preserving the integrity of core processes. The design promotes inclusivity and transparency but requires diligent community oversight to address potential misuse of its permissionless features.

	✓ Solved	⚠ Caution Advised	✗ Resolution Pending
High	0	0	0
Medium	1	0	0
Low	0	0	0
No Risk	2	0	0
Total	**3**	**0**	**0**

During this security assessment, Coinspect identified one medium-risk issue related to how an arbitrary bribe token could be used to lock down all bribes sent in Bold. It is worth noting that, Liquity stated that they do not expect to deploy initiatives with arbitrary bribe tokens.

2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

2.3 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

Id	Title	Risk
LGOV-05	Attackers can lock users' Bold bribes for all future epochs	Medium
LGOV-06	Lack of validation during deployment allows spamming the governance	None
LGOV-07	Overflow when calculating absolute values	None

3. Scope

The scope was set to be the repository at https://github.com/liquity/V2-gov at commit `1c379b59f184f5805cf851d3969c6d0f800626b1`. The smart contract `UniV4Donations` is part of this commit but is out of this audit's scope.

On a previous engagement, Coinspect identified several issues on this smart contract. Liquity's team removed this smart contract from the project at commit `972e4e86619adba49a6a0e0b442e21d5b7aee7a1`.

4. Assessment

Liquity's V2 Governance system is designed to enable decentralized decision-making. Comprised by a core governance smart contract that enables the registration of initiatives and the allocation of voting power in a way designed to align with community consensus. Incentive mechanisms, such as those facilitated by the bribing initiatives, allow stakeholders to align rewards with participation, enhancing engagement without disrupting core governance.

4.1 Security assumptions

The security of the system relies on participants exercising due diligence in their interactions with governance mechanisms. This includes carefully reviewing initiatives, understanding their parameters, and assessing potential impacts before committing resources or votes. It is assumed that voters act with informed judgment, and the community collectively ensures accountability and transparency in decision-making. Seatbelts or a similar type of off-chain system could be deployed to help users identify potentially malicious initiatives by, for example, detecting forbidden or suspicious opcodes (e.g. selfdestruct) and upgradeable smart contracts.

4.2 Decentralization

Fixe The governance's design limits the influence of individual actors by enabling community-driven initiatives and voting processes. The staking of LQTY tokens determines the voting power of participants, and an epoch-based system is employed to manage governance stages. This design mitigates the risks of sudden parameter manipulation or attacks within a single block, enhancing the system's robustness.

A critical feature of the governance system is its openness; any user with sufficient voting power and the ability to pay the registration fee can propose and promote initiatives. This promotes transparency and inclusivity but introduces certain risks. For instance, malicious actors could exploit the system by deploying upgradeable initiatives and modifying their behavior post-deployment, potentially deceiving users and compromising rewards.

The governance framework lacks privileged roles or administrative functions to adjust parameters, underscoring the importance of carefully chosen initial configurations to ensure system stability.

4.3 Testing

The testing suite reflects Liquity's strong commitment to improving the security and robustness of its codebase. Its comprehensive design enabled Coinspect to efficiently explore a variety of adversarial scenarios and attack vectors. The suite includes other capabilities such as fork tests, which provide realistic simulations of blockchain conditions, and fuzz testing with frameworks like Echidna, allowing for the exploration of edge cases and unexpected behaviors. Moreover, the inclusion of proofs of concept from other reports highlights Liquity's proactive approach to incorporating external insights, further enhancing the testing suite's thoroughness and reliability.

4.4 Code quality

The codebase reflects a high level of quality, achieved through comprehensive testing, in-depth audits conducted by multiple firms, and well-prepared documentation. Detailed NatSpec on each relevant function and variable allows

5. Detailed Findings

LGOV-05

Attackers can lock users' Bold bribes for all future epochs

Status

Solved

Resolution

Fixed

Risk

Medium

Impact

High

Likelihood

Low

Location

```
V2-gov/src/BribeInitiative.sol:73"
```

Description

Anyone can deploy Bribe Initiatives with arbitrary tokens, which might lead to a scenario where all Bold bribes deposited into the smart contract are locked. This happens due to an overflow triggered when trying to claim bribes. As a consequence, unsuspecting bribers will lose all their Bold bribes since the claiming process reverts.

The implementation of Bribe Initiative allows its deployment with any bribe token:

```
    constructor(address _governance, address _bold, address
_bribeToken) {
        require(_bribeToken != _bold, "BribeInitiative: bribe-token-
cannot-be-bold");

        governance = IGovernance(_governance);
        bold = IERC20(_bold);
        bribeToken = IERC20(_bribeToken);

        EPOCH_START = governance.EPOCH_START();
        EPOCH_DURATION = governance.EPOCH_DURATION();
    }
```

Then, any actor is able to supply bribes in both Bold and Bribe tokens.

When depositing, the smart contract increases internal trackers with each supplied amount for future epochs:

```
    function depositBribe(uint256 _boldAmount, uint256
_bribeTokenAmount, uint256 _epoch) external {
        uint256 epoch = governance.epoch();
        require(_epoch >= epoch, "BribeInitiative: now-or-future-
epochs");

        bribeByEpoch[_epoch].remainingBoldAmount += _boldAmount;
        bribeByEpoch[_epoch].remainingBribeTokenAmount +=
_bribeTokenAmount;

        emit DepositBribe(msg.sender, _boldAmount, _bribeTokenAmount,
_epoch);

        bold.safeTransferFrom(msg.sender, address(this), _boldAmount);
        bribeToken.safeTransferFrom(msg.sender, address(this),
_bribeTokenAmount);
    }
```

Afterwards, once the epoch starts, allocators are able to claim the bribes according to a share-based calculation that depends on their votes, in `_claimBribes()`:

```
        uint256 epochEnd = EPOCH_START + _epoch * EPOCH_DURATION;
        uint256 totalVotes = _lqtyToVotes(totalLQTYAllocation.lqty,
epochEnd, totalLQTYAllocation.offset);
        uint256 votes = _lqtyToVotes(lqtyAllocation.lqty, epochEnd,
lqtyAllocation.offset);
        uint256 remainingVotes = totalVotes - bribe.claimedVotes;

        boldAmount = bribe.remainingBoldAmount * votes /
remainingVotes;
        bribeTokenAmount = bribe.remainingBribeTokenAmount * votes /
remainingVotes;
```

```
        bribe.remainingBoldAmount -= boldAmount;
        bribe.remainingBribeTokenAmount -= bribeTokenAmount;
        bribe.claimedVotes += votes;
```

Consider a scenario where an upgradeable Bribe token is used. It can be upgraded to a malicious implementation that assigns `remainingBribeTokenAmount` to `type(uint256).max` upon bribe deposit, without the need of transferring any token. This could be done after many Bold bribes were sent for a given epoch, just before the epoch ends. As a consequence, `_claimBribes()` overflows when trying to calculate the claimed `bribeTokenAmount` since the remaining bribes were inflated:

```
        bribeTokenAmount = bribe.remainingBribeTokenAmount * votes /
remainingVotes;
```

Since the calculations to claim both Bold and the bribe tokens are made on the same context, all bribes in Bold for that epoch will remain locked because the execution reverts.

This scenario can be exploited by directly deploying the Bribe Initiative with the malicious token or compromising an upgradeable bribe token, triggering an upgrade to a malicious implementation.

Coinspect considers the likelihood to be low, since users still have to allocate funds to this initiative or requires attackers compromising an upgradeable token. The impact is high as valuable Bold tokens will remain locked inside the smart contract.

Recommendation

Allow users to continue claiming Bold bribes if the calculation for the bribe token amount overflows.

Alternatively, document this scenario related to upgradeable or potentially malicious bribe tokens.

Status

Fixed by adding documentation.

The Liquity Team added this concern into the repository's readme as a known issue.

Proof of Concept

The following test shows how a malicious bribe token allows locking all Bold bribes deposited for an epoch, thanks to an overflow. The example uses a simplified malicious implementation, but it can ultimately be any replaced by any upgradeable token. An upgradeable token allows hiding the malicious intention by making the sudden code update followed with the malicious bribe deposit.

```
    ├─ [9383] BribeInitiative::claimBribes([ClaimData({ epoch: 3,
prevLQTYAllocationEpoch: 3, prevTotalLQTYAllocationEpoch: 3 })])
    │   ├─ [478] Governance::epoch() [staticcall]
    │   │   └─ ← [Return] 5
    │   └─ ← [Revert] panic: arithmetic underflow or overflow (0x11)
    └─ ← [Revert] panic: arithmetic underflow or overflow (0x11)
```

```
contract ArbBribeToken {
    function transferFrom(address from, address to, uint256 amount)
external pure returns (bool) {
        return true;
    }
}
```

```
function test_RevertClaimBribesWithArbBribeToken() public {
    // NOTE: This test requires deploying the ArbBribeToken in setUp()
    // and deploy the bribeInitiative with that token

    // =========== epoch 1 ==================
    // user stakes in epoch 1
    _stakeLQTY(user1, 1e18);

    // =========== epoch 2 ==================
    vm.warp(block.timestamp + EPOCH_DURATION);
    assertEq(2, governance.epoch(), "not in epoch 2");

    // lusdHolder deposits lqty claimable in epoch 3,
    vm.startPrank(lusdHolder);
    lusd.approve(address(bribeInitiative), 1e18);
    bribeInitiative.depositBribe(1e18, 0, governance.epoch() + 1);
    vm.stopPrank();

    // Right before the epoch ends, a malicious actor inflates
remainingBribeTokenAmount
    vm.warp(block.timestamp + EPOCH_DURATION - 12);
    bribeInitiative.depositBribe(0, type(uint256).max,
governance.epoch() + 1);

    uint256 depositedBribe = governance.epoch() + 1;

    // =========== epoch 3 ==================
    vm.warp(block.timestamp + 13);
    assertEq(3, governance.epoch(), "not in epoch 3");
    // user votes on bribeInitiative
    _allocateLQTY(user1, 1e18, 0);

    // =========== epoch 5 ==================
    vm.warp(block.timestamp + (EPOCH_DURATION * 2));
    assertEq(5, governance.epoch(), "not in epoch 5");

    // user won't be able to get the Bold bribes
    // all bold bribes for that epoch will remain locked
    (uint256 boldAmount, uint256 bribeTokenAmount) =
        _claimBribe(user1, depositedBribe, depositedBribe,
depositedBribe);
}
```

LGOV-06

Lack of validation during deployment allows spamming the governance

Status

Solved

✓

Resolution

Acknowledged

Risk

None

Impact

Recommendation

Likelihood

-

Location

`V2-gov/src/Governance.sol:77`

Description

The smart contract's deployment does not check that the registration fee and threshold are not extremely low values. As a consequence, adversaries could spam the governance by registering many malicious initiatives trying to disrupt the functioning of the system.

The deployment includes multiple range checks for critical operational variables:

```
require(_config.minClaim <= _config.minAccrual, "Gov: min-claim-gt-min-
accrual");
REGISTRATION_FEE = _config.registrationFee;

// Registration threshold must be below 100% of votes
require(_config.registrationThresholdFactor < WAD, "Gov: registration-
config");
REGISTRATION_THRESHOLD_FACTOR = _config.registrationThresholdFactor;

// Unregistration must be X times above the `votingThreshold`
require(_config.unregistrationThresholdFactor > WAD, "Gov:
unregistration-config");
UNREGISTRATION_THRESHOLD_FACTOR =
_config.unregistrationThresholdFactor;
UNREGISTRATION_AFTER_EPOCHS = _config.unregistrationAfterEpochs;
```

However, these checks allow dangerous configurations such as setting:

- `REGISTRATION_FEE == 0`
- `REGISTRATION_THRESHOLD_FACTOR == 0`

In such case, anyone would be able to register initiatives for free.

Recommendation

Revert the deployment when setting parameters to overly extreme values.

Status

Acknowledged.

The Liquity Team stated that they will pay attention and validate these values when deploying to prevent abuse and spam.

LGOV-07

Overflow when calculating absolute values

Status

Solved

✓

Resolution

Acknowledged

Risk

None

Impact

Recommendation

Likelihood

-

Location

`V2-gov/src/utils/Math.sol:22`

Description

The calculation of absolute values for integer variables reverts due to an overflow when the input is `INT256_MIN`.

The `int256` type can represent values from `-2^255` to `2^255 - 1`. The minimum value, `INT256_MIN`, is `-2^255`. When the function attempts to compute the absolute value of `INT256_MIN`, it performs the operation `-int256(a)`, which translates to `-(-2^255)`. This results in `2^255`, a value that exceeds the maximum representable value for `int256 (2^255 - 1)`, causing an integer overflow.

```
function abs(int256 a) pure returns (uint256) {
    return a < 0 ? uint256(-int256(a)) : uint256(a);
}
```

The add and sub functions rely on the abs function to handle negative values of b:

```
function add(uint256 a, int256 b) pure returns (uint256) {
    if (b < 0) {
        return a - abs(b);
    }
    return a + uint256(b);
}
```

```
function sub(uint256 a, int256 b) pure returns (uint256) {
    if (b < 0) {
        return a + abs(b);
    }
    return a - uint256(b);
}
```

Recommendation

Check that a is not the minimum int256 before performing the negation.

Status

Acknowledged.

Liquity acknowledges this issue and considers that the overflow is not reachable with the circulating supply of LQTY.

Proof of Concept

```
function test_CoinspectAbsMinReverts() external {
    int256 absMin = type(int256).min;

    uint256 absMinVal = abs(absMin); // overflows
}
```

6. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.