Klaytn Square Governance Contracts Security Audit

: Klaytn Square Governance Contracts Security Security Audit 2023 1Q

Feb 21th, 2023

Revision 1.1

Theori

Table of Contents

Executive Summary

Starting on January 16, 2023, ChainLight of Theori assessed the smart contract for Klaytn Governance system, which is the solidity implementation of KIP-81 and will be introduced on the Kore (v1.10.0) hard fork.

The governance system is used to make proposals for the growth of the Klaytn network and to determine whether to implement the proposals based on GCs' opinion on the proposals. The entire process can also be viewed by all users.

We focused on identifying issues that can lead to the execution of a malicious proposal without the proper voting process, a denial of service, and whether a malicious attacker would be able to get more voting power than they have or seize others' voting power. Furthermore, we tried to standardize the governance systems used in the various decentralized protocols, identified the differences between them, and provided recommendations to make the voting process more secure and resilient.

Two high-severity issues allowed a malicious governance council member to steal other members' voting power. One low-severity issue allowed a malicious council member to unstake without penalties, like reducing the voting power when unstaking is requested and on the withdrawal locking period.

Audit Overview

Scope

Name	2023 Klaytn GC System
Target / Version	GitHub Repository: https://github.com/klaytn/governance-contracts-audit-theori
Application Type	Governance Smart Contract
Lang. / Platforms	Smart contracts [Solidity]

Code revision

Commit hash: 702920b

Before the revision, abstains were included only in the quorum, but now abstentions are treated as no. Although this change didn't introduce security issues, we advised that abstention should not belong to either yes or no. However, Klaytn team decided to distinguish between no and abstain on off-chain and kept the updated contract code.

Governance Table Mapping

Category	Item	Klaytn Governance	Compound V2 (Alpha)
Proposal creation requirement	Hold more than the configured amount of tokens or votes		
	Burn tokens		
	Any participant who meets other conditions		
Calculation of the voting power	Staked tokens		
	Tokens delegated to		
Voter	Token holders with a balance above the threshold		
	Users selected in the off-chain process		
Vote switch after casting a vote	Not allowed		
Voting period / delay	Fixed to hardcoded value		
	Fixed to configured value		
	Settable per proposal within hardcoded range		
	Settable per proposal within the configured range		
Voting power cap	None		
	Based on the number of total voters		
Place of voting	On-chain		
Cancelation of the proposal before voting period	By proposer		
	By admin		
Cancelation of the proposal after or during voting	By admin		
	By cancelation vote		
Proposal pass condition	Number of votes over the quorum		
	Number of yes votes over the required ratio		
	Number of voters over the quorum		
	Approval from the majority of the voters		
Limitation on the number of proposals	None		
Executor of a	Delegator or admin		

passed proposal	Everyone		
Safety mechanisms	The admin can cancel the queued proposal		
	Slashing or other penalties for malicious action		
Participation encouragement	Potential increase of the value of assets due to advancement of the protocol		

Klaytn Governance Characteristics	Note
GC and admin who have the right to vote can make propose	Permissions can be set by passing proposal
Staked tokens are calculated as voting power	Amount of KLAY staked on the Staking contract of GC
Only GC specified in off-chain can participate	Policy exists to hold participants accountable for malicious behavior
Unable to switch votes after casting	Unable to re-voting
Voting period can be set for each proposal	Existence of voting period limit
restrictions on voting rights proportional to the total number of voters	
Voting on-chain	
Proposer can cancel proposal before voting starts	
Unable to cancel proposal after voting starts	
Conditions for the proposal to pass are the number of votes over the quorum	More than one-third of all votes required
Conditions for the proposal to pass are the number of voters over the quorum	More than one-third of all voters required
Conditions for the proposal to pass are approval from the majority of the voters	Yes > No *(+ Abstain)
There is no limit on the number of proposals	

* Added after code revision. See the code revision section for details.

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the vendor, and they confirm that they received.
Reported	ChainLight reported the issue to the vendor.
Fixed	The vendor resolved the issue.
Acknowledged	The vendor acknowledged the potential risk, but they will resolve it later.
WIP	The vendor is working on the patch.
Won't Fix	The vendor acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	0	• N/A
High	2	• THE-KORE-001 • THE-KORE-002
Medium	1	• THE-KORE-005
Low	1	• THE-KORE-007
Informational	4	• THE-KORE-003 • THE-KORE-004 • THE-KORE-006 • THE-KORE-008

Findings

Summary

Last updated: Feb 7, 2023

#	ID	Title	Severity	Status
1	THE-KORE-001	populateFromAddressBook RewardAddress Spoofing	High	Fixed
2	THE-KORE-002	resolveStakingFromAddressBook RewardAddress Spoofing	High	Fixed
3	THE-KORE-003	Use Pull over Push pattern for KLAY transfer in CnStakingV2	Informational	Won't Fix
4	THE-KORE-004	TimingRule validation is insufficient	Informational	Fixed
5	THE-KORE-005	CnStakingV2 acceptRewardAddress can be front-run	Medium	Fixed
6	THE-KORE-006	Broken Backward Compatibility (KIP-81)	Informational	Acknowledged
7	THE-KORE-007	updateStakingTracker can accept a fake StakingTracker	Low	Fixed
8	THE-KORE-008	Emergency System Additional Suggestions	Informational	Acknowledged

#1 KORE-001 populateFromAddressBook RewardAddress Spoofing

ID	Summary	Severity
THE-KORE-001	In case a malicious GC sets its own `rewardAddress` equal to the `rewardAddress` of the victim's GC prior to the process in which the `populateFromAddressBook` function of `StakingTracker.sol` searches for the representative `NodeId`, the `balance` owned by the victim's `CnStakingV2` is accounted as the malicious GC's `balance`.	**High**

Description.

Klaytn network stores the information about GC in the `AddressBook` contract. GC-related information consists of `NodeId`, `StakingAddress`, and `rewardAddress`. The same multiple `NodeId` can't be registered in `AddressBook` at the same time. Each `NodeId` has its own `StakingAddress` and `rewardAddress`.

Because `AddressBook` does not provide information about the owner of `NodeId`, the function in `StakingTracker.populateFromAddressBook` determines the same GC if it has the same `rewardAddress` for finding the total `balance` of one GC. Here, the frontmost `NodeId` will be the representative `NodeId` of the GC and the `balances` of other `NodeIds` will be added to the representative `NodeId`. (GCs should set `rewardAddress` as their own address to receive the block creation reward. Thus, the code is assuming that the GCs will only register their own `rewardAddress`.)

```
function populateFromAddressBook(uint256 trackerId) private {
        Tracker storage tracker = trackers[trackerId];

        (address[] memory nodeIds,
         address[] memory stakingContracts,
         address[] memory rewardAddrs) = getAddressBookLists();

        // Consolidate staking contracts (grouped by reward address)
        for (uint256 i = 0; i < nodeIds.length; i++) {
            address n = nodeIds[i];
            address s = stakingContracts[i];
            address r = rewardAddrs[i];
            uint256 balance = getStakingBalance(s);

            if (tracker.rewardToNodeId[r] == address(0)) { // fresh rewardAddr
```

```
                tracker.nodeIds.push(n);
                tracker.rewardToNodeId[r] = n;

                tracker.stakingToNodeId[s] = n;
                tracker.stakingBalances[s] = balance;
                tracker.nodeBalances[n] = balance;
            } else { // previously appeared rewardAddr
                n = tracker.rewardToNodeId[r]; // use representative nodeId

                tracker.stakingToNodeId[s] = n;
                tracker.stakingBalances[s] = balance;
                tracker.nodeBalances[n] += balance;
            }
        }
    }
```

contracts/StakingTracker.sol #*populateFromAddressBook(uint256)*

Impact

High, **Vulnerable**

Attack scenario:

1. `CnStakingV1` of malicious GC is present in `AddressBook`. (not migrated.)
2. `rewardAddress` of CnStakingV1 of malicious GC is changed to `rewardAddress` of the victim GC before `propose` starts.
3. Once `propose` starts, the victim GC cannot recover the voting right in the corresponding `propose` whatever it tries. (`stakingToNodeId` is set when the first tracker is created and is not modifiable afterward.)
4. In the name of upgrade from V1 to V2 before the voting termination, the attacker can request for V1 `unregister` and V2 `register`. (`NodeId` for the `register` request must be the same as the old `NodeId`)
5. The attacker calls the `submitUpdateVoterAddress` function on the newly registered `CnStakingV2` contract.
6. The `NodeId` that stole the voting right is still stored in `stakngTracker`, so this can be used to vote.

The following conditions are present, and the `impact` may vary depending on the configuration or other modifications:

1. Due to the sequential storage in `AddressBook`, V1 is guaranteed to be registered ahead of V2 so that the attack can be executed against any GC.

2. If the sum of the voting right stolen by the malicious GC and its own voting right exceeds 50%, the voting results can be manipulated.
3. Once the voting is completed, a re-vote is not allowed.
4. Only the `secretary` can `execute` before the proposal of `voterExecute=true` is passed, which can serve as a kind of safety mechanism. However, after `voterExecute=true`, the impact of this issue will become greater.
5. `rewardAddress` should be changed before calling the `propose` function.
6. The attacker can execute the attack only once if they want to actually cast a vote with the stolen voting rights, but the steal of voting rights can be repeated.

Recommendations

Two remediation options are proposed.

1. In the initialization of the `CnStakingV2` contract, `GC_ID` should be stored and should be used to aggregate the contracts instead of the NodeId.
2. For future convenience and code simplification of `AddressBook`, modify `AddressBook` to fix the issue when the `Kore` hard fork is progressed.

Fix

Commit hash: 3853fac

GCs can specify their GC ID during the initialization of the `CnStakingV2`. And then, the Klaytn admin would validate the GC ID and register it to the `AddressBook`. This is similar to the first recommendation but not exactly the same since we expected the validation to be in the contract's code.
A failure to filter the malicious GC's request to register a `CnStakingV2` contract with another GC's ID would make this issue exploitable again. So Klaytn admins should be very careful when validating a register request for the `AddressBook`.

#2 KORE-002 resolveStakingFromAddressBook rewardAddress Spoofing

ID	Summary	Severity
THE-KORE-002	In case a malicious GC sets its own `rewardAddress` equal to the `rewardAddress` of the victim GC prior to the process in which the `resolveStakingFromAddressBook` function of `StakingTracker.sol` searches for the representative `NodeId`, the victim's `voter` can be overridden by the attacker.	**High**

Description

Klaytn network stores the information about GC in the `AddressBook` contract. GC-related information consists of `NodeId`, `StakingAddress`, and `rewardAddress`. The same multiple `NodeId` can't be registered in `AddressBook` at the same time. Each `NodeId` has its own `StakingAddress` and `rewardAddress`.

Because `AddressBook` does not provide information about the owner of `NodeId`, the function in `StakingTracker.resolveStakingFromAddressBook` determines the same GC if it has the same `rewardAddress` for finding the `NodeId` from the staking contract address. Here, the frontmost `NodeId` with the matching `rewardAddress` will be the representative `NodeId` of the provided staking contract.

```
function resolveStakingFromAddressBook(address staking) private view
returns(address) {
        (address[] memory nodeIds,
         address[] memory stakingContracts,
         address[] memory rewardAddrs) = getAddressBookLists();

        address rewardAddr;
        for (uint256 i = 0; i < nodeIds.length; i++) {
            if (stakingContracts[i] == staking) {
                rewardAddr = rewardAddrs[i];
                break;
            }
        }
        for (uint256 i = 0; i < nodeIds.length; i++) {
            if (rewardAddrs[i] == rewardAddr) {
                return nodeIds[i];
```

```
            }
        }
        return address(0);
    }
```

contracts/StakingTracker.sol #`resolveStakingFromAddressBook(address)`

Impact

High, **Vulnerable**

Attack scenario:

1. A malicious GC has `CnStakingV1`
2. A malicious GC will request the migration to `CnStakingV2` but the old version will remain in the AddressBook.
3. Call `submitUpdateRewardAddress` from `CnStakingV2` (for the first `voter` designation).
4. Change the `rewardAddress` of `CnStakingV1` to the `rewardAddress` of the victim GC.
5. Call `submitUpdateRewardAddress` from `CnStakingV2` (for the second `voter` designation).
6. For subsequent `propose` function calls, the attacker can vote on behalf of the victim GC.

The following conditions are present, and the `impact` may vary depending on the configuration or other modifications:

1. Due to the sequential storage in `AddressBook`, V1 is guaranteed to be registered ahead of V2 so that the attack can be executed against any GC.
2. If the sum of the voting right stolen by the malicious GC and its own voting right exceeds 50%, the voting results can be manipulated.
3. Once the voting is completed, a re-vote is not allowed.
4. Only the `secretary` can `execute` before the proposal of `voterExecute=true` is passed, which can serve as a kind of safety mechanism. However, after `voterExecute=true`, the impact of this issue will become greater.
5. The attack's impact is persistent unless the victim changes the voter back.

Recommendations

Same as `KORE-001`

Fix

Commit hash: 3853fac

Same as `KORE-001`

#3 KORE-003 Use Pull over Push pattern for KLAY transfer in CnStakingV2

ID	Summary	Severity
THE-KORE-003	Potential security issue due to reentrancy when transferring KLAY deposited in the `CnStakingV2` contract	Informational

Description

When withdrawing the KLAY deposited in the `CnStakingV2` contract, two methods can be used: `withdrawLockupStaking` that withdraws a `stake` that is `Lockup`, and `withdrawApprovedStaking` that withdraws a stake that is not Lockup. Both functions employ `.call{ value: _value }("")` to transfer the KLAY and call the `safeRefreshStake` function to reflect the balance change in `StakingTracker`.
The transmission of native tokens calls the contract at the destination. Since in the case of `CnStakingV2`, `safeRefreshStake` is not called yet at the time of `call` use, `CnStakingV2` and `StakingTracker` are not synchronized, and problems may occur if an attacker's contract calls another contract that references data from these contracts. (Read-Only Reentrancy)

```
function withdrawLockupStaking(address payable _to, uint256 _value) external
override
    onlyMultisigTx()
    notNull(_to) {
        ( , , , , uint256 withdrawableAmount) = getLockupStakingInfo();
        require(_value > 0 && _value <= withdrawableAmount, "Value is not
withdrawable.");

        remainingLockupStaking -= _value;

        (bool success, ) = _to.call{ value: _value }("");
        require(success, "Transfer failed.");

        safeRefreshStake();
        emit WithdrawLockupStaking(_to, _value);
    }
```

contracts/CnStakingV2.sol #*withdrawLockupStaking(address, uint256)*

```
function withdrawApprovedStaking(uint256 _id) external override
    onlyAdmin(msg.sender) {
        WithdrawalRequest storage request = withdrawalRequestMap[_id];
        require(request.to != address(0), "Withdrawal request does not exist.");
        require(request.state == WithdrawalStakingState.Unknown, "Invalid state.");
        require(request.value <= staking, "Value is not withdrawable.");
        require(request.withdrawableFrom <= block.timestamp, "Not withdrawable
yet.");

        uint256 withdrawableUntil = request.withdrawableFrom + STAKE_LOCKUP();
        if (withdrawableUntil <= block.timestamp) {
            request.state = WithdrawalStakingState.Canceled;
            unstaking -= request.value;

            safeRefreshStake();
            emit CancelApprovedStakingWithdrawal(_id, request.to, request.value);
        } else {
            request.state = WithdrawalStakingState.Transferred;
            staking -= request.value;
            unstaking -= request.value;

            (bool success, ) = request.to.call{ value: request.value }("");
            require(success, "Transfer failed.");

            safeRefreshStake();
            emit WithdrawApprovedStaking(_id, request.to, request.value);
        }
    }
```

contracts/CnStakingV2.sol #*withdrawApprovedStaking(uint256)*

Impact

Informational

In the current *CnStakingV2* code, there is no threat due to Reentrancy. However, As explained in the Description, however, Read-Only Reentrancy may occur in a new contract or 3rd party code.

Recommendations

This issue has no impact as of now, but we recommend eliminating the reentrancy to mitigate the potential threat. When transmitting a native token, using a call in the middle of the function should be avoided, and the Pull over Push pattern should be used instead. This can be implemented by recording only the target and quantity and making users call the exclusive withdrawal function later. Because the `StakingTracker.getStakingBalance` function uses the KLAY balance of the `CnStakingV2` function, when the Pull over Push pattern is used, the unstaking variable should be adjusted appropriately.

Fix

We recommended eliminating the reentrancy to mitigate the potential threat. However, Klaytn team answered that it seems an attacker can't profit from exploiting this, so they decided not to fix it. However, they should re-evaluate this issue when they add other contracts that interact with `CnStakingV2` in the future.

#4 KORE-004 TimingRule validation is insufficient

ID	Summary	Severity
THE-KORE-004	When changing the `TimingRule`, the minimum period can be set to a very short time, leading to insufficient voting time for new proposals.	Informational

Description

When creating the proposal in the Voting contract, each procedure's minimum and maximum periods are limited according to the `TimingRule`. Those limits can be updated via governance or Klaytn admin, and the sanity check is insufficient. For example, the minimum voting period can be set to one second. And a newly created proposal after this setting may be passed without giving sufficient voting time to GCs.

```
function updateTimingRule(
        uint256 minVotingDelay, uint256 maxVotingDelay,
        uint256 minVotingPeriod, uint256 maxVotingPeriod)
    public override onlyGovernanceOrSecretary {
        TimingRule storage tr = timingRule;
        tr.minVotingDelay = minVotingDelay;
        tr.maxVotingDelay = maxVotingDelay;
        tr.minVotingPeriod = minVotingPeriod;
        tr.maxVotingPeriod = maxVotingPeriod;

        validateTimingRule();

        emit UpdateTimingRule(tr.minVotingDelay, tr.maxVotingDelay,
                              tr.minVotingPeriod, tr.maxVotingPeriod);
    }
```

contracts/Voting.sol #updateTimingRule(uint256, uint256, uint256, uint256)

Impact

Informational

Recommendations

It is better to set sufficient minimum limits that are not zero.

Fix

Commit hash: 56bb6ee

Klaytn team decided that one day was an appropriate time for the minimum voting period and delay, and adjusted checks accordingly.

#5 KORE-005 CnStakingV2 acceptRewardAddress can be front-run

ID	Summary	Severity
THE-KORE-005	`rewardAddress` of `CnStakingV2` can be changed to the arbitrary address only if Klaytn admin approves. However, the approval TX can be front-run to change the pending address to the different one with the approved one.	**Medium**

Description

To change the `rewardAddress`, `CnStakingV2` `admin` must call the `updateRewardAddress` function first to set the `pendingRewardAddress` and wait for the `acceptRewardAddress` function call. And the `acceptRewardAddress` function can only be called by the Klaytn admin or the `pendingRewardAddress` itself. But the `acceptRewardAddress` function doesn't have arguments and will just put `pendingRewardAddress` to the `rewardAddress` if the caller is valid.

```
function acceptRewardAddress() external override {
        require(canAcceptRewardAddress(), "Unauthorized to accept reward address");


IAddressBook(ADDRESS_BOOK_ADDRESS()).reviseRewardAddress(pendingRewardAddress);
        rewardAddress = pendingRewardAddress;
        pendingRewardAddress = address(0);

        emit UpdateRewardAddress(rewardAddress);
    }
```

contracts/CnStakingV2.sol #`acceptRewardAddress()`

Impact

Medium, **Vulnerable**

To do TX front-running in Klaytn, the attacker should be either the `operator` of `EN` used by the victim or the block proposer (one of the GCs) of the next block. Although it may not be a serious problem with only this attack, if `rewardAddress` can be arbitrarily changed, a strong attack vector that uses the `rewardAddress` of other GCs can be created. (Like `KORE-001` and `KORE-002`.)

Recommendations

It should be modified such that the change only occurs when the value recognized by the manager and the value in the storage is the same. This can be achieved by receiving `pendingRewardAddress` as the augment of the `acceptRewardAddress` function and comparing it with the value in the storage.

Fix

Commit hash: c88232e

Klaytn team patched as we recommended.

#6 KORE-006 Broken Backward Compatibility (KIP-81)

ID	Summary	Severity
THE-KORE-006	KIP-81 states that the `refreshStake` function is present in `StakingTracker` and can be called manually to include the balance of V1 in the total staked balance for the backward compatibility of `CnStakingV1` and `V2`, but the real code does not support this.	Informational

Description

The `refreshStake` function in `StakingTracker` plays the role of checking the balance of the staking contracts to track the staked balance. The `getStakingBalance` function that fetches the balance checks the version. If the version is not V2, 0 is returned even if the balance is present. As such, the staked balance in the V1 contract is not reflected, differently from the content in the document.

```
function getStakingBalance(address staking) public view virtual returns(uint256) {
        if (isCnStakingV2(staking)) {
            uint256 accountBalance = staking.balance;
            uint256 unstaking = ICnStakingV2(staking).unstaking();
            return (accountBalance - unstaking);
        }
        return 0;
    }
```

contracts/StakingTracker.sol #`getStakingBalance(address)`

Impact

Informational

Recommendations

It may confuse GCs because KIP-81 and the code are different. Thus, we propose that KIP-81 or the code be modified to unify the content.

Fix

Klaytn team answered that they would edit the KIP-81 to match the code.

#7 KORE-007 updateStakingTracker can accept a fake StakingTracker

ID	Summary	Severity
THE-KORE-007	`CnStakingV2` `admin` can freely change the `StakingTracker` address and thus bypass the voting rights tracking.	Low

Description

`CnStakingV2` synchronizes by calling `StakingTracker` when Balance or Voter is changed. However, this synchronization can be avoided since the `CnStakingV2` `admin` can freely change the `StakingTracker` address to the address of the fake `StakingTracker` they created.

```
function setStakingTracker(address _tracker) external override
    beforeInit()
    onlyAdmin(msg.sender)
    notNull(_tracker) {
        require(validStakingTracker(_tracker), "Invalid contract");

        stakingTracker = _tracker;
        emit UpdateStakingTracker(_tracker);
    }
```

contracts/CnStakingV2.sol #`setStakingTracker(address)`

Impact

Low, **Vulnerable**

When unstaking from `CnStakingV2`, the voting right is reduced from the time of obtaining the withdrawal right, even though the actual withdrawal is only possible after a certain period. However, a malicious GC can bypass this reduction while `unstaking` by making their `CnStakingV2` use a fake `StakingTracker`.

Attack scenario:

- A new proposal is created. `StakingTracker` creates a Tracker and stores the current state.
- The attacker calls `updateStakingTracker` to change the address of `StakingTracker`, which is used by its own `CnStakingV2`.
- The attacker calls `submitApproveStakingWithdrawal` to acquire the withdrawal right. (At this time, `refreshStake` should be called, but it will not have an effect since it's the attacker-controlled code.)
- The vote progresses.
- After a certain period, `withdrawApprovedStaking` is called to withdraw the staked balance.

The voting right is not limited even with withdrawal using the method above. However, additional voting rights will not be granted because the withdrawal would be received after the vote.

Recommendations

A fix for this issue would be simple. However, we propose two options to mitigate the potential issues that can be introduced later.

1. Use a Factory contract to ensure the code and the initial state of the new `StakingTracker` contract instance, and then only use contracts created by the Factory wherever the `StakingTracker` contract is used.
2. Deprecate the `StakingTracker` contract and calculate every one's voting rights when the first vote is cast for a proposal. It would lead to slightly inaccurate calculations since the first vote may not happen at the exact beginning of the voting period. However, the error would not be too big, and the benefit of simplifying the code is greater than the error because we expect the proposer to vote quickly.

Fix

Commit hash: e2185e8

The first option we gave was not selected because it would make upgrading the `StakingTracker` contract harder, and the second one was not selected because even if the error is small, it may confuse users, and also, it's harder to communicate the vote start time if they don't have an exact timestamp.
Alternatively, the Klaytn team decided to allow a change of `stakingTracker` only when `livetracker` doesn't exist. Although their solution is not as robust as we recommended, it fixes the issue.

#8 KORE-008 Additional Emergency System Suggestions

ID	Summary	Severity
THE-KORE-008	There should be safety mechanisms to cancel the passed proposal.	Informational

Description

Klaytn governance system puts the passed `proposal` into the queued state and delays the execution of it for at least 48 hours to let ecosystem participants act against the side effects of the `proposal` if needed. But the proposal can only be canceled by the proposer before the start of the voting, and it can't be canceled in other states.
If participants detect unexpected side effects during this delay, they should submit a `proposal` to negate this as soon as possible. However, since this new `proposal` will also be subject to the `voting delay`, `voting period`, and queuing delay, it would be impossible to cancel out the effect of the already queued proposal perfectly. So, if things go wrong, halting the network and/or performing a hard fork might be the only solution.

If we compare Klaytn with Compound:

- A execute function can only be called by the `secretary`(Klaytn admin) or a GC for Klaytn, but Compound lets anyone call it.
- Compound had a multisig `guardian` account to cancel the dangerous `proposal` in their early version of the governance, but Klaytn doesn't have such a feature.

Compound protocol's governance was open to participation from the beginning, so they made a tradeoff of decentralization and security by adding the guardian feature.

On the other hand, Klaytn is not really decentralized due to the use of the PBFT consensus algorithm, only pre-approved GCs can produce blocks, and the governance system only allows the participation of GCs. These characteristics make it unlikely for an outright malicious `proposal` to be submitted. But still, if such a `proposal` is submitted and passed, it could be problematic.

So, we recommend having safety mechanisms to let Klaytn admin cancel the queued `proposal`. Since Klaytn governance is already somewhat centralized, it would add security without much loss of decentralization.

Impact

Informational

Recommendations

A guardian account should be able to cancel the queued proposal. And to deter abuse of the safety feature, this `guardian` account should be a multisig wallet. Even with the multisig, censoring a `proposal` can be a problem, but it's a minor risk compared to executing a bad proposal because of the characteristics of the Klaytn mentioned in the description section.

Fix

We recommended adding a `guardian` account under the assumption that the aftermath of side effects would be more than the problem of censoring the `proposal`. However, the Klaytn team did not accept our recommendation due to the following reasons. 1) At least one of GC can figure out the side effects before the voting starts. And currently, a proposer can cancel their proposal before the voting starts. 2) A `secretary` (Klaytn admin) may passively cancel a `proposal` by refusing to execute it since only the `secretary` can execute the `proposal` as of now.

Revisions

Revision	Date	History
1.0	2023.02.06	Original writing
1.1	2023.02.21	Code revision `Commit hash: 702920b`

Appendix

Appendix #1. Governance Characteristics Classification

Category	Item
Proposal creation requirement	Hold more than the configured amount of tokens or votes
	Burn tokens
	Only admin
	Any participant who meets other conditions
Calculation of the voting power	Liquid tokens
	Staked tokens
	Vested staking rewards
	Tokens delegated to
Voter	Token holders
	Token holders with a balance above the threshold
	Token holders with a balance (including the amounts delegated to) above the threshold
	Users selected in the off-chain process
	Everyone
Vote switch after casting a vote	Not allowed
	Allowed
Voting period / delay	Fixed to hardcoded value
	Fixed to configured value
	Settable per proposal within hardcoded range
	Settable per proposal within the configured range
Voting power cap	None
	Based on the number of total voters
Place of voting	Off-chain

	On-chain
Cancelation of the proposal before voting period	By proposer
	By admin
	Not allowed
Cancelation of the proposal after or during voting	By proposer
	By admin
	By cancelation vote
	Not allowed
Proposal pass condition	Number of votes over the quorum
	Number of yes votes over the required ratio
	Number of voters over the quorum
	Approval from the majority of the voters
Limitation on the number of proposals	None
	Exists
Executor of a passed proposal	Only admin
	Delegator or admin
	Everyone
Safety mechanisms	The admin can cancel the queued proposal
	Cancelation vote is possible for the queued proposal
	Slashing or other penalties for malicious action
	None
Participation encouragement	Potential increase of the value of assets due to advancement of the protocol
	A reward for vote participation
	A reward for submitting a good proposal

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.

ChainLight