Kroma Validator System v2 Security Audit

: Kroma Validator System v2 + GovernanceToken, MintManager, VestingWallet

---

Sep 19, 2024

Revision 1.0

ChainLight@Theori

Table of Contents

Executive Summary

Starting June 20, 2024, ChainLight of Theori audited Kroma's validator system v2 (including governance token, mint manager, and vesting wallet contracts) for 3 weeks. In the audit, we primarily considered the issues/impacts listed below.

- Theft of funds
- Permanent freeze of funds
- Insufficient access control
- Correctness of slashing amount/condition
- Discrepancy in the documented mint/unlock schedule and the implementation

As a result, we identified the issues listed below.

- Total: 7
- Critical: 2 (Multiple design flaws regarding the challenge process and VKRO, An exploitable reentrancy.)
- Medium: 3 (Share dilution, Insecure randomness, etc.)
- Low: 1 (Minor internal accounting error.)
- Informational: 1 (Minor suggestions.)

Audit Overview

Scope

Name	Kroma Validator System v2 Security Audit

Target / Version	• Git Repository (`kroma-network`)
	• commit `96aec7919d643b5ee485a8a49e99bf65dbf658ff`
	  ◦ (`feat/implement-validator-system-v2`)
	• commit `bf3eee263ec3a3867e7187960b2b6ddca4b282a5`
	  ◦ (`GovernanceToken.sol` and `MintManager.sol`)
	• commit `0e7487f3c7d24b4bfe7a5f526b3e3f624af61c40`
	  ◦ (`VestingWallet.sol`)

Application Type	• Smart contracts • Blockchain node (L2)
Lang. / Platforms	• Smart contracts [Solidity] • Blockchain node (L2) [Go]

Code Revision

N/A

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.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the client.
WIP	The client is working on the patch.
Patched	The client fully resolved the issue by patching the root cause.
Mitigated	The client resolved the issue by reducing the risk to an acceptable level by introducing mitigations.
Acknowledged	The client acknowledged the potential risk, but they will resolve it later.
Won't Fix	The client acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	**2**	• `KROMA-VALIDATOR-V2-002` • `KROMA-VALIDATOR-V2-004`
High	**0**	• N/A
Medium	**3**	• `KROMA-VALIDATOR-V2-003` • `KROMA-VALIDATOR-V2-005` • `KROMA-VALIDATOR-V2-006`
Low	**1**	• `KROMA-VALIDATOR-V2-001`
Informational	**1**	• `KROMA-VALIDATOR-V2-007`
Note	**0**	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	KROMA-VALIDATOR-V2-001	Changes to validatorKro of the AssetManager are n ot synced with totalKro	Low	Patched
2	KROMA-VALIDATOR-V2-002	Reentrancy attack via KGH N FT's safeTransferFrom() allows fund theft	Critical	Patched
3	KROMA-VALIDATOR-V2-003	Users may not be able to wit hdraw KGH NFTs under rare conditions	Medium	Patched
4	KROMA-VALIDATOR-V2-004	Design issues related to the challenge process	Critical	Patched
5	KROMA-VALIDATOR-V2-005	Incorrect KGH share calculati on causes new KGH delegat ors to receive less boosted Reward	Medium	Patched
6	KROMA-VALIDATOR-V2-006	Usage of insecure randomne ss in ValidatorManager._ updatePriorityValidato r()	Medium	Mitigated
7	KROMA-VALIDATOR-V2-007	Minor suggestions	Informational	Patched

#1 KROMA-VALIDATOR-V2-001 Changes to validatorKro of the AssetManager are not synced with totalKro

ID	Summary	Severity
KROMA-VALIDATOR-V2-001	The validatorKro value in the AssetManager does not increase or decrease proportionally to changes in totalKro.	Low

Description

The validatorKro value represents the amount of KRO deposited by a validator. When totalKro increases or decreases, the validatorKro should also appropriately increase or decrease. However, there are currently the following issues:

- _initUndelegate() : When a validator withdraws KRO, the validatorKro does not decrease by the correct amount corresponding to the withdrawal.
- modifyBalanceWithChallenge() : Changes to validatorKro do not properly reflect changes to totalKro . The validatorKro should increase or decrease proportionally to the amount of change in totalKro multiplied by validator's KRO share / total KRO share .
- increaseBalanceWithReward() : Even though totalKro increases, validatorKro does not increase accordingly.

Impact

Low

The quantity of validatorKro does not increase or decrease appropriately in proportion to changes in totalKro . However, even if validatorKro holds an inaccurate value, the amount of KRO calculated during a withdrawal via the _initUndelegate() call is based on the shares held by the validator, so it is not possible to withdraw more than the correct amount.

Recommendation

Instead of recalculating the validatorKro value every time the totalKro changes, it is recommended to modify the totalValidatorKro() function to calculate and return "_vaults[validator].asset.totalKro.mulDiv(_vaults[validator].kroDelegators[validator].shares, _vaults[validator].asset.totalKroShares)".

Remediation

Patched

This issue has been resolved by having the `validatorKro` increase and decrease independently from `totalKro`. (The validator's KRO share has been removed, and the `validatorKro` value is no longer included in the `totalKro`.)

#2 `KROMA-VALIDATOR-V2-002` Reentrancy attack via KGH NFT's `safeTransferFrom()` allows fund theft

ID	Summary	Severity
KROMA-VALIDATOR-V2-002	A reentrancy attack may occur when transferring the KGH NFT in `AssetManager.finalizeUndelegateKgh()`. This vulnerability could enable an attacker to steal significantly more KRO tokens than they are legitimately entitled to.	Critical

Description

The `finalizeUndelegateKgh()` completes the undelegate process by transferring a KGH NFT to the caller using `safeTransferFrom()`. If the `msg.sender` is a contract, the `safeTransferFrom()` calls the `onERC721Received()` of the contract. An attacker can use this callback function to re-enter the `finalizeUndelegateKgh()`, allowing repeated calls without updating the caller's `pendingShares` value. Each time reentrancy occurs, the attacker can receive an additional amount of KRO tokens equal to what they were originally entitled to.

Impact

Critical

An attacker can repeatedly collect KRO rewards through multiple reentrant calls, multiplying the reward by the number of successful re-entries, thereby stealing large quantities of KRO tokens.

Recommendation

It is recommended to follow the Checks-Effects-Interactions pattern. Therefore, the deletion of storage data related to `kghDelegator` must be performed before the `for` loop that executes `KGH.safeTransferFrom()`.

Remediation

Patched

The issue has been resolved as recommended.

#3 KROMA-VALIDATOR-V2-003 Users may not be able to withdraw KGH NFTs under rare conditions

ID	Summary	Severity
KROMA-VALIDATOR-V2-003	There are rare conditions in which users may be unable to successfully undelegate their KGH NFTs through `AssetManager.finalizeUndelegateKGH()`.	Medium

Description

A `mulDiv` overflow error occurs in `finalizeUndelegateKGH()`'s `kroSharesToUndelegate.mulDiv(pendingAsset.totalPendingAssets, pendingAsset.totalPendingKroShares)` under the following conditions:

- `pendingAsset.totalPendingKghShares` is greater than 0, so `rewardExists` is true.
- `pendingAsset.totalPendingKroShares` is 0.

This situation arises when `baseRewardsToReceive` is greater than 0, but `boostedRewardsToReceive` is 0 in `_initUndelegateKgh()`. Typically, this scenario does not occur because both base rewards and boosted rewards increase simultaneously when the output finalization rewards are accumulated. However, suppose slashing occurs after output finalization rewards are claimed, and the value of `kroShares` drops. In that case, there can be a rare case where `baseRewardsToReceive` becomes 0 while `boostedRewardsToReceive` remains positive in `_initUndelegateKgh()`. If the user calls `initUndelegateKgh()` under these conditions, a `mulDiv` overflow may occur when calling `finalizeUndelegateKGH()`, preventing the user from withdrawing their KGH NFTs.

Impact

Medium

In a rare case where `pendingAsset.totalPendingKghShares` is greater than 0 and `pendingAsset.totalPendingKroShares` is 0, the user cannot undelegate their KGH NFTs successfully.

Recommendation

In `AssetManager.finalizeUndelegateKgh()`, the calculation of `KroAssetsToUndelegate` should only occur when `totalPendingKroShares` is not equal to 0.

Remediation

Patched

The issue has been resolved with changes to the reward distribution logic during KGH undelegation.

#4 KROMA-VALIDATOR-V2-004 Design issues related to the challenge process

ID	Summary	Severity
KROMA-VALIDATOR-V2-004	Design flaws that may lead to the theft of validators' and delegators' funds exist in the challenge process.	Critical

Description

Several vulnerabilities have been identified in the challenge process:

1. **Challenge initiation without bond**: Validators can perform multiple `createChallenge()` calls simultaneously without needing a bond.

   a. Even if an attacker challenges all unfinalized `outputRoot`s over a seven-day period, the attack cost is only the minimum amount required to become a validator (`MIN_ACTIVATE_AMOUNT`).

   b. If an attacker loses multiple times while conducting challenges, victorious challengers may not receive their rewards due to the depletion of the attacker's validator funds.

   c. Defenders may incur losses even when they win, as generating a zkProof to `proveFault` is computationally intensive.

2. **Intentional challenge losses to steal delegated funds as rewards**: Validators can cause slashing by intentionally losing challenges and, therefore, steal delegated funds as a `challengeReward`.

   a. Currently, validators can be activated by staking only the `MIN_REGISTER_AMOUNT` of KRO, supplemented by delegations from others. If other users delegate a large amount of KRO to the validator, the validator can profit by intentionally losing challenges to pay the `challengeReward` to an opponent. (Since the challenger can be specified, a malicious validator may intentionally lose to another validator they control.)

   b. The economic incentive for such attacks increases when the amount of delegated KRO is large, and the validator's own stake is small. Specifically, when the validator's own capital loss from losing a challenge is less than the `challengeReward` gained by the opposing validator, there is an economic motivation to lose intentionally.

3. **Integer underflow allowing to steal all KRO in `AssetManager`**: By deliberately losing challenges, underflows can be induced to steal all KRO deposited in the `AssetManager`.

   a. If an attacker intentionally causes multiple challenge losses simultaneously, there are cases where the `challengeReward` becomes `MIN_SLASHING_AMOUNT` in the `AssetManager.modifyBalanceWithChallenge()`. Additionally, if the attacker's validator funds decrease such that `challengeReward > totalAmount`, underflows can occur in calculations within unchecked blocks, such as `boostedReward -= arr[1].mulDiv(challengeReward, totalAmount);` and calculations related to variables like `totalPendingAssets`, `totalPendingBoostedRewards`, and `validatorRewardKro`.

   b. After triggering the underflow in `validatorRewardKro`, the attacker can call `initClaimValidatorReward()` to steal all KRO held in the `AssetManager` contract.

4. **Blocking KGH undelegation**: Validators can forcibly prevent KGH undelegation and hog `outputRoot` finalization rewards by setting the commission rate to 100%.

   a. By intentionally losing challenges and reducing the `totalKro` value, an underflow can be induced during the calculation of `baseRewardsToReceive` in the `_initUndelegateKgh()`. This causes a revert due to an overflow in `mulDiv()`, making KGH undelegation impossible.

   b. Until the `totalKro` amount increases and `kroAssetsToWithdraw` exceeds `kroInKgh` during `_initUndelegateKgh()` calls, KGH delegators are forced to continue delegating to the validator. The validator can set the `commissionRate` to 100% in this state, not sharing any `outputRoot` finalization rewards with delegators. Furthermore, since `totalKro` does not increase when the `commissionRate` is at 100%, users remain unable to undelegate their NFTs.

Impact

Critical

A malicious validator can steal all KRO from the `AssetManager`. Also, they can block KGH undelegation or steal funds delegated to them.

Recommendation

A comprehensive review of operational policies is necessary. Reimplementing the challenge and delegation mechanisms with a new design is recommended, as simply modifying problematic parts may not be sufficient in certain scenarios.

Remediation

Patched

Design changes and code rewrites have been implemented for the challenge and delegation processes. Validators are now required to provide collateral for each `outputRoot` submission and challenge creation. Additionally, only the validator's collateral is slashed when a challenge is lost, ensuring that users' delegated funds remain unaffected.

#5 KROMA-VALIDATOR-V2-005 Incorrect KGH share calculation causes new KGH delegators to receive less boostedReward

ID	Summary	Severity
KROMA-VALIDATOR-V2-005	As the boostedReward value accumulates, new KGH NFT delegators receive fewer shares than existing delegators, resulting in lower rewards for new delegators.	Medium

Description

When delegating KGH NFTs, the AssetManager._convertToKghShares() calculates KGH shares by converting them into a virtual KRO amount using the VKRO_PER_KGH ratio. The calculation is as follows:

```
function _convertToKghShares(address validator) internal view returns (uint128) {
  return
    VKRO_PER_KGH.mulDiv(
      _totalKghShares(validator) + DECIMAL_OFFSET,
      _totalKghAssets(validator) + 1
    );
}

function _totalKghAssets(address validator) internal view returns (uint128) {
  return
    _vaults[validator].asset.totalKgh *
    VKRO_PER_KGH +
    _vaults[validator].asset.boostedReward;
}
```

In this calculation, the denominator _totalKghAssets() increases as boostedReward grows due to accumulated rewards from outputRoot finalizations. As a result, new KGH NFT delegators receive fewer KGH shares because the increasing boostedReward dilutes their share calculation.

This leads to new delegators receiving a smaller portion of the `boostedReward` than existing delegators.

Impact

Medium

Boosted rewards for new KGH delegators are diluted.

Recommendation

Removing the `VKRO_PER_KGH` ratio and the KGH share calculations is recommended. Instead, `boostedReward` should be distributed based on the number of delegated NFTs.

Remediation

Patched

The issue has been resolved as recommended.

#6 KROMA-VALIDATOR-V2-006 Usage of insecure randomness in ValidatorManager._updatePriorityValidator()

ID	Summary	Severity
KROMA-VALIDATOR-V2-006	ValidatorManager._updatePriorityValidator() uses predictable on-chain data for random number generation, enabling a malicious priority validator to manipulate the selection process to increase their chances of re-selection and unfairly accumulate rewards.	Medium

Description

ValidatorManager._updatePriorityValidator() function relies on publicly accessible on-chain data—such as block.number, block.coinbase, block.difficulty, block.prevrandao, and blockhash(block.number - 1)—to generate random values that determine the next priority validator. As the function is executed when the current priority validator submits an outputRoot and the result can be predicted, the malicious validator can use services like Flashbots to execute the transaction at a specific moment, manipulating the random value generation.

Moreover, the malicious priority validator has a 30-minute window (approximately 120 blocks) to monitor the chain and time their transaction to increase the likelihood of being re-selected. By repeatedly exploiting this vulnerability, the malicious actor can frequently become the priority validator and accumulate more rewards through successive outputRoot submissions.

Impact

Medium

A malicious priority validator can exploit this vulnerability to manipulate the random number generation and significantly increase their chances of being re-selected, leading to an unfair accumulation of rewards.

Recommendation

Implementing off-chain monitoring of validator selection patterns is recommended to detect any anomalies that deviate significantly from expected probabilities based on _validatorTree weights.

Additionally, adopting Verifiable Random Functions (VRFs) should be considered to ensure that the randomness used in validator selection is secure and cannot be manipulated.

Remediation

Mitigated

The team plans to monitor off-chain for any instances of abuse where a priority validator exhibits abnormal selection probabilities.

#7 KROMA-VALIDATOR-V2-007 Minor suggestions

ID	Summary	Severity
KROMA-VALIDATOR-V2-007	The description outlines multiple suggestions to prevent incorrect settings due to operational mistakes, mitigate potential issues, enhance code maturity and readability, and address other minor concerns.	Informational

Description

Operational Risk Mitigation / Sanity Check

1. **BalancedWeightTree.insert():** It is recommended to add a `require(_tree.nodeMap[_addr] == 0)` check to ensure that an address is not added to the tree more than once.
2. **BalancedWeightTree._pullUp():** If an incorrect index is assigned to an empty node, `node.parent` could be 0, leading to `_tree.root` being initialized to 0. While the current implementation prevents this by calling `_pullUp()` within `remove()`, future changes could reintroduce this risk. It is advised to set `_tree.root = 0` only when `tree.root == _index` is true.
3. **Ownership Management:** Contracts inheriting from `Ownable` should consider using `Ownable2Step` to avoid transferring ownership to the wrong address, as it also allows for `renounceOwnership()`.
4. **Vesting Wallet:** To ensure proper duration settings, a check `_durationSeconds % VESTING_CYCLE == 0` is recommended in `VestingWallet.initialize()`.
5. **Mint Manager:** The function `renounceOwnershipOfToken()` in the `MintManager` contract should be restricted to execution only after the distribution is complete to prevent accidental freezing of funds.

Code Maturity

- The use of `unchecked { _updatePriorityValidator(); }` in `ValidatorManager.afterSubmitL2Output()` does not impact the internal logic of `_updatePriorityValidator()`. It is recommended that `unchecked` be removed or integrated within `_updatePriorityValidator()` for clarity and optimization.

Gas Optimization

- **Vesting Wallet:**
  In `KromaVestingWallet._vestingSchedule()`, changing the condition from `timestamp > start() + duration()` to `timestamp >= start() + duration()` could yield gas savings.

Missing / Confusing Events

- **Governance Token:** The `GovernanceToken` contract does not emit events for `mint()` and `burn()` as its parent, `KromaMintableERC20`, does. It is recommended that consistent events be emitted for clarity.

Other Recommendations

1. **Governance Token:** Since the `GovernanceToken` inherits from `ERC20Burnable`, it allows addresses other than the bridge to burn tokens, which could unintentionally reduce `totalSupply`. If not needed, this functionality should be removed.
2. **Mint Manager:** The `balance == 0` condition in `MintManager.distribute()` should be removed to prevent potential exploits where attackers freeze funds by sending tokens before distribution completes.
3. **Mint Manager:** When deploying a `MintManager`, if the same recipient is specified multiple times as a constructor argument, previous values may be overwritten (e.g., assigning shares of 1 and 2 could lead to 4 instead of the intended 3). A check should be added to verify whether `shareOf[recipient]` has already been set to avoid unintended distributions.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

Patched

The issues have been resolved as recommended.

Revision History

Version	Date	Description
1.0	Sep 19, 2024	Initial version

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.