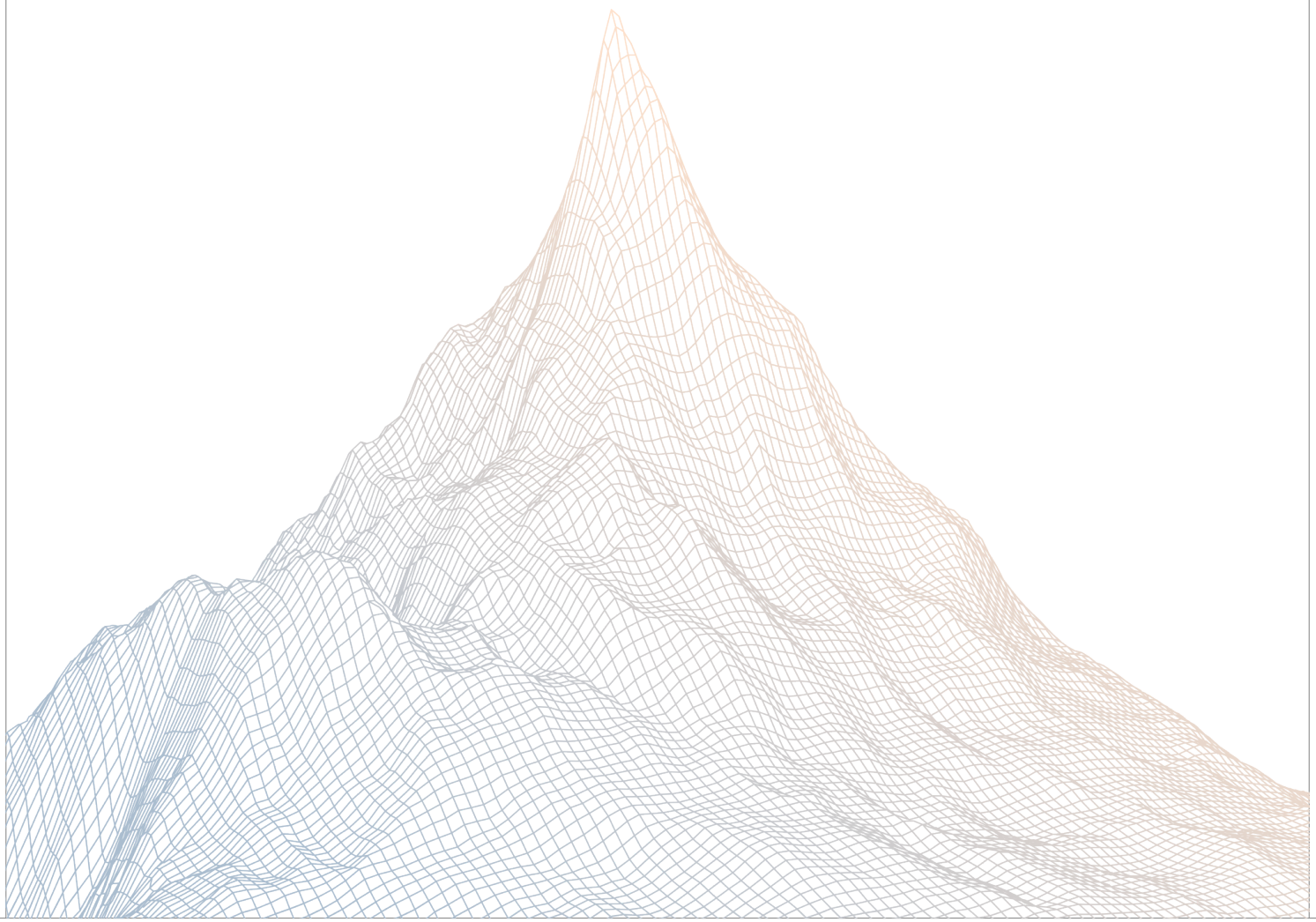
Zenith

Kinetiq

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: February 26th to March 6th, 2025

AUDITED BY: ether_sky
ladboy

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Kinetiq

Kinetiq is a protocol designed to provide liquid staking services on the Hyperliquid blockchain. Liquid staking allows users to stake their tokens to support network security and operations while maintaining liquidity of their assets. This means that users can earn staking rewards without locking up their tokens, enabling them to utilize their assets in other decentralized finance (DeFi) activities simultaneously

2.2 Scope

The engagement involved a review of the following targets:

Target	kinetiq-protocol
Repository	https://github.com/kinetiq-research/kinetiq-protocol
Commit Hash	`d35151c204bea4221ffe2dc1cbb98512707bb210`

Files	`DefaultAdapter.sol`
	`KHYPE.sol`
	`OracleManager.sol`
	`PauserRegistry.sol`
	`StakingManager.sol`
	`ValidatorManager.sol`

2.3 Audit Timeline

February 26, 2025	Audit start
March 6, 2025	Audit end
March 17, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	2
Medium Risk	6
Low Risk	3
Informational	4
Total Issues	**15**

3

Findings Summary

ID	Description	Status
H-1	The targetBuffer amount of HYPE tokens will be locked in the StakingManager	Resolved
H-2	The HYPE tokens are not withdrawn from validator when the validator is deactivated	Resolved
M-1	Abnormal validator is closed directly without applying slashing penalty.	Resolved
M-2	Hyper to KHyper exchange ratio is ignored and the ratio is always one to one	Resolved
M-3	The generatePerformance function can revert when the active validator's balance becomes 0	Resolved
M-4	The getExchangeRatio function is incorrect	Resolved
M-5	The generatePerformance function can unexpectedly revert	Resolved
M-6	The staking limit check is not functioning correctly	Resolved
L-1	The withdraw amount is sliently downcasted from uint256 to uint64	Resolved
L-2	The RebalanceRequest struct should specify the StakingManager	Resolved
L-3	It's impossible to set a non-zero maxStakeAmount when the stakingLimit is 0	Resolved
I-1	The check for stakingLimit is insufficient when maxStakeAmount is 0	Resolved
I-2	GeneratePerformance consider large loop in OracleManager.sol	Resolved
I-3	Unused code in StakingManager.sol	Resolved

ID	Description	Status
I-4	There is no need to use both validatorExists and validatorActive modifiers simultaneously	Resolved

4

Findings

4.1 High Risk

A total of 2 high risk findings were identified.

[H-1] The targetBuffer amount of HYPE tokens will be locked in the StakingManager

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- StakingManager.sol

Description:

Users stake `HYPE` tokens through `StakingManager`. However, not all deposited `HYPE` tokens are directly `delegated` to `validators`; some are kept as a `buffer`.

- StakingManager.sol#L328

```
function _distributeStake(uint256 amount) internal {
    // Handle buffer first
    uint256 currentBuffer = hypeBuffer;
    uint256 target = targetBuffer;
    if (amount > 0 && currentBuffer < target) {
        uint256 bufferSpace = target - currentBuffer;
        uint256 amountToBuffer = Math.min(amount, bufferSpace);
@->         hypeBuffer = currentBuffer + amountToBuffer;
        amount -= amountToBuffer;
    }
}
```

The intention behind this `buffer` is to use it instead of `undelegating` from `validators` when `stakers` queue a `withdrawal`. Unfortunately, the current implementation does not utilize the `buffer` when `stakers` queue a `withdrawal`. The assets are withdrawn directly from the `validators`, regardless of the `buffers` in the `StakingManager`.

- StakingManager.sol#L212

```
function queueWithdrawal(uint256 amount)
    external nonReentrant whenNotPaused whenWithdrawalNotPaused {
    address currentDelegation
    = validatorManager.getDelegation(address(this));
    require(currentDelegation ≠ address(0), "No delegation set");
@->     _withdrawFromValidator(currentDelegation, amount);
}
```

As a result, the `targetBuffer` amount of `HYPE` tokens remains locked and cannot be used.

Recommendations:

```
    function queueWithdrawal(uint256 amount)
    external nonReentrant whenNotPaused whenWithdrawalNotPaused {
    +    uint256 currentBuffer = hypeBuffer;
    + ^^I uint256 amountFromBuffer = Math.min(amount, currentBuffer);
    +^^I hypeBuffer = currentBuffer - amountFromBuffer;

    +^^I emit WithdrawalQueued(msg.sender, withdrawalId, amount);

    +^^I amount -= amountFromBuffer;

    +^^Iif (amount > 0) {
        // Withdraw from current delegation
        address currentDelegation
    = validatorManager.getDelegation(address(this));
        require(currentDelegation ≠ address(0), "No delegation set");
        _withdrawFromValidator(currentDelegation, amount);
    +^^I}

    emit WithdrawalQueued(msg.sender, withdrawalId, amount);
}
```

Consider add a function to update the hypeBuffer and reduce the buffer during withdraw.

Kinetiq: Fixed in commit 2f4b5316eb

Zenith: Verified.

[H-2] The HYPE tokens are not withdrawn from validator when the validator is deactivated

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- ValidatorManager.sol

Description:

In the `ValidationManager`, there is an `_addRebalanceRequest` function where `rebalance requests` are created.

- ValidatorManager.sol#L263-L264

```
function _addRebalanceRequest(address validator, uint256 withdrawalAmount)
    internal {
    (bool exists, uint256 index) = _validatorIndexes.tryGet(validator);
    require(exists, "Validator does not exist");
    require(_validators[index].balance >= withdrawalAmount, "Insufficient
    balance");

    validatorRebalanceRequests[validator] = RebalanceRequest({validator:
    validator, amount: withdrawalAmount});
    _validatorsWithPendingRebalance.add(validator);
}
```

These `requests` will be closed in the future by `rebalancing` the `HYPE` to the current delegate using the `closeRebalanceRequests` function.

- ValidatorManager.sol#L295

```
function closeRebalanceRequests(address stakingManager, address[]
    calldata validators)
    external
    whenNotPaused
    nonReentrant
    onlyRole(MANAGER_ROLE)
```

```
{
    for (uint256 i = 0; i < validators.length;) {
        RebalanceRequest memory request
    = validatorRebalanceRequests[validator];
        totalAmount += request.amount;

        delete validatorRebalanceRequests[validator];
        _validatorsWithPendingRebalance.remove(validator);

        unchecked {
            ++i;
        }
    }

    if (totalAmount > 0) {

    IStakingManager(stakingManager).processValidatorRedelegation(totalAmount);
    }
}
```

The normal flow for `rebalancing` is as follows:

- The necessary `rebalance` amount is withdrawn from the `validator`, and a `rebalance request` is created.
- The `manager` closes the `request` and `redelegates` the `withdrawn HYPE` to the current delegate.

For example, when the `manager` calls the `rebalanceWithdrawal` function, the `rebalance request` is made at `line 239`, and the necessary `HYPE` is withdrawn from the `validators` at `line 247`.

- ValidatorManager.sol#L239

```
function rebalanceWithdrawal(
    address stakingManager,
    address[] calldata validators,
    uint256[] calldata withdrawalAmounts
) external whenNotPaused nonReentrant onlyRole(MANAGER_ROLE) {
    require(validators.length == withdrawalAmounts.length, "Length
    mismatch");
    require(validators.length > 0, "Empty arrays");

    for (uint256 i = 0; i < validators.length;) {
        require(validators[i] ≠ address(0), "Invalid validator address");

        // Add rebalance request (this will check for duplicates)
```

```
239:         _addRebalanceRequest(validators[i], withdrawalAmounts[i]);

            unchecked {
                ++i;
            }
        }

        // Trigger withdrawals through StakingManager
247:     IStakingMan-
    ager(stakingManager).processValidatorWithdrawals(validators,
    withdrawalAmounts);
}
```

However, if the `validator` is `deactivated` by the `manager` or `oracle manager`, only the `rebalance request` is created, and the actual `withdrawal` is not performed.

- ValidatorManager.sol#L186

```
function deactivateValidator(address validator)
    external whenNotPaused nonReentrant validatorExists(validator) {
    Validator storage validatorData = _validators[index];
    require(validatorData.active, "Validator already inactive");

    // Create withdrawal request before state changes
    if (validatorData.balance > 0) {
        _addRebalanceRequest(validator, validatorData.balance);
    }

    // Update state after withdrawal request
    validatorData.active = false;
}
```

This `rebalance request` will still be closed by the `manager` in the future, like other requests. Also these `rebalance requests` need to be closed before `activation` can occur again. However, the `HYPE` for this `request` is not `withdrawn` and it is also impossible to `withdraw` these `HYPE` using other methods. As a result, there will not be enough `HYPE` to close this `rebalance request`, and the tracking the `HYPE` amounts in the specific `StakingManager` will be broken. Furthermore, there is no guarantee that this `validator`'s `balance` belongs to a single `StakingManager`.

Recommendations:

There is no need to create a `rebalance request` in the `deactivateValidator` function. The `manager` can still create a `rebalance request` using the `rebalanceWithdrawal` function for these `deactivated validators`.

```
function deactivateValidator(address validator)
    external whenNotPaused nonReentrant validatorExists(validator) {
    Validator storage validatorData = _validators[index];
    require(validatorData.active, "Validator already inactive");

    // Create withdrawal request before state changes
-    if (validatorData.balance > 0) {
-        _addRebalanceRequest(validator, validatorData.balance);
-    }

    // Update state after withdrawal request
    validatorData.active = false;
}
```

This fix can also prevent unexpected transaction reverts in the `generatePerformance` function of the `OracleManager`. The `generatePerformance` function is important and calls the `deactivateValidator` function of the `ValidationManager`. Additionally, the `_addRebalanceRequest` function can revert in some cases. By removing this, the transaction revert can be avoidable.

Kinetiq: Fixed in commit @231453f9ee...

Zenith: Verified.

4.2 Medium Risk

A total of 6 medium risk findings were identified.

[M-1] Abnormal validator is closed directly without applying slashing penalty.

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- OracleManager.sol

Description:

- OracleManager.sol#L271

```
if (
    !_checkValidatorBehavior(
        validator, previousSlashing, previousRewards, avgSlashAmount,
    avgRewardAmount, avgBalance
    )
) {
    // Deactivate validator if behavior is unexpected
    validatorManager.deactivateValidator(validator);
    continue;
}
```

When generate the performance report and update the validator balance, if the validator behave is deemed as "abnormal" => while report change or slash change exceed certain threshold, the validator is deactivated.

However, even the validator mishaves and subject to slash, the code does not apply the slash.

The code just deactivate the validator and process validator withdraw.

Abnormal validator is closed directly without applying slashing penalty and the malicious / misbehaved validator can be reckless and does not worry about slashing.

Recommendations:

```
// Check for anomalies
  if (
      !_checkValidatorBehavior(
          validator, previousSlashing, previousRewards, avgSlashAmount,
    avgRewardAmount, avgBalance
      )
  ) {

      if (avgSlashAmount > previousSlashing) {
          uint256 newSlashAmount = avgSlashAmount - previousSlashing;
          validatorManager.reportSlashingEvent(validator, newSlashAmount);
      }

      validatorManager.deactivateValidator(validator);
      continue;
  }
```

Slash the validator. before deactivation.

Kinetiq: Fixed in commit @b19d899149ae...

Zenith: Verified. The fix ensures if the validator's behavior is abnormal, the code return false to not update any validator's performance.

[M-2] Hyper to KHyper exchange ratio is ignored and the ratio is always one to one

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- StakingManager.sol

Description:

The Hyper to KHype exchange ratio is not always one to one, the exchange ratio can be retrieved by calling the function getExchangeRatio

```
// Create withdrawal request
    _withdrawalRequests[msg.sender][withdrawalId]
    = WithdrawalRequest({amount: amount, timestamp: block.timestamp});
```

However, when a user queues a withdrawal and confirms it, the number of KHYper tokens burned is exactly the number the user receives, and the exchange ratio is not applied.

```
function confirmWithdrawal(uint256 withdrawalId)
    external nonReentrant whenNotPaused {
    uint256 amount = _processConfirmation(msg.sender, withdrawalId);
    require(amount > 0, "No valid withdrawal request");
    require(address(this).balance >= amount, "Insufficient contract
    balance");

    // Burn kHYPE and process withdrawal
    kHYPE.burn(address(this), amount);
    payable(msg.sender).transfer(amount);
}
```

The _processConfirmation returns the queued amount and does not apply the exchange ratio.

Recommendations:

There can be case when a portion of staked HYPE is slashed and the Hyper to KHyper exchange ratio is no longer one to one, then consider query the exchange ratio when the withdraw is confirmed.

Kinetiq: Fix in commit 2f4b5316eb85

Zenith: Verified.

[M-3] The generatePerformance function can revert when the active validator's balance becomes 0

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- OracleManager.sol

Description:

The `generatePerformance` function is crucial in the `OracleManager` as it periodically updates the metrics of all `validators`. In this function, the `validator`'s behavior is checked. If the check fails, the `validator` is `deactivated`.

- OracleManager.sol#L272

```
function generatePerformance()
    external whenNotPaused onlyRole(OPERATOR_ROLE) returns (bool) {
    for (uint256 i = 0; i < validatorCount; i++) {
        if (
            !_checkValidatorBehavior(
                validator, previousSlashing, previousRewards,
    avgSlashAmount, avgRewardAmount, avgBalance
            )
        ) {
            // Deactivate validator if behavior is unexpected
            validatorManager.deactivateValidator(validator);
            continue;
        }
    }

    return true;
}
```

In the `_checkValidatorBehavior` function, if the `validator`'s `balance` is 0, the function reverts instead of returning `false`.

- OracleManager.sol#L139

```
function _checkValidatorBehavior(
    address validator,
    uint256 previousSlashing,
    uint256 previousRewards,
    uint256 avgSlashAmount,
    uint256 avgRewardAmount,
    uint256 avgBalance
) internal returns (bool isValid) {
    // Ensure we have a balance to compare against
    require(avgBalance > 0, "Zero balance");

    return true;
}
```

This means that the `generatePerformance` function can revert in this case. There is a scenario where an `active validator`'s `balance` becomes 0. The `activation` and `deactivation` of `validators` in the `ValidationManager` are asynchronous and can be performed by external calls from the `manager` or `oracle manager`. Therefore, there is no guarantee that `validators` with a `0 balance` are `deactivated` immediately. The current implementation prevents this `deactivation` by reverting the entire transaction.

Recommendations:

```
function _checkValidatorBehavior(
    address validator,
    uint256 previousSlashing,
    uint256 previousRewards,
    uint256 avgSlashAmount,
    uint256 avgRewardAmount,
    uint256 avgBalance
) internal returns (bool isValid) {
    // Ensure we have a balance to compare against
    require(avgBalance > 0, "Zero balance");
    if (avgBalance == 0) return false;

    return true;
}
```

Kinetiq: Fixed in commit @b19d899149...

Zenith: Verified.

[M-4] The getExchangeRatio function is incorrect

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

- StakingManager.sol

Description:

As the sponsor confirmed, there are several `StakingManagers` and one shared `ValidationManager`. Each `StakingManager` has a `getExchangeRatio` function that returns the ratio between `HYPE` and `KHYPE` tokens.

- StakingManager.sol#L371

```
function getExchangeRatio() public view returns (uint256) {
    uint256 totalSlashing = validatorManager.totalSlashing();
    uint256 totalRewards = validatorManager.totalRewards();
    uint256 kHYPESupply = kHYPE.totalSupply();

    require(kHYPESupply > 0, "No kHYPE supply");

    // Calculate total HYPE: totalStaked + totalRewards - totalClaimed -
    totalSlashing
371:    uint256 totalHYPE = totalStaked + totalRewards - totalClaimed
    - totalSlashing;

    // Return ratio = totalHYPE / kHYPE.totalSupply (scaled by 1e18)
    return (totalHYPE * 1e18) / kHYPESupply;
}
```

Currently, the use case of this function is ambiguous and unclear.

Regardless, the `getExchangeRatio` function should return the correct ratio. However, while `totalRewards` and `totalSlashing` values are obtained from the `ValidationManager` and used in every `StakingManager`, `totalStaked` and `totalClaimed` values are specific to each `StakingManager`. They do not represent the total `staked` and `claimed` amounts across the entire protocol. As a result, the total `HYPE` amount at `line 371` is incorrect and will vary between `StakingManagers`.

Recommendations:

Move the `getExchangeRatio` function to the `ValidationManager`. Also, track the total `staked` and `claimed` amounts within the `-ValidationManager`.

Kinetiq: Fix in commit @2f4b5316eb8...

Zenith: Verified.

[M-5] The generatePerformance function can unexpectedly revert

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- ValidatorManager.sol

Description:

The `getPerformance` function returns various metrics for the `validator`. The `slashAmount` always increases because it accumulates from the beginning, as confirmed by the comments.

- IOracleAdapter.sol#L20

```
/**
    * @notice Get performance metrics for a validator
    * @param validator Address of the validator
    * @return balance Current balance of the validator
    * @return uptimeScore Uptime performance score (0-10000)
    * @return speedScore Speed performance score (0-10000)
    * @return integrityScore Integrity performance score (0-10000)
    * @return selfStakeScore Self-stake performance score (0-10000)
    * @return rewardAmount Cumulative rewards earned by validator

@→    * @return slashAmount Cumulative amount slashed from validator
    */
function getPerformance(address validator)
    external
    view
    returns (
        uint256 balance,
        uint256 uptimeScore,
        uint256 speedScore,
        uint256 integrityScore,
        uint256 selfStakeScore,
        uint256 rewardAmount,
        uint256 slashAmount
```

```
    );
```

The `generatePerformance` function is crucial in the `OracleManager` as it periodically updates the metrics of all `validators`. Within this function, three other functions are called:

- At `line 284`, the `reportSlashingEvent` function is called.
- At `line 290`, the `reportRewardEvent` function is called.
- Finally, at `line 294`, the `updateValidatorPerformance` function is called.

- OracleManager.sol#L294

```
function generatePerformance()
    external whenNotPaused onlyRole(OPERATOR_ROLE) returns (bool) {
    // Update validators with averaged values
    for (uint256 i = 0; i < validatorCount; i++) {
        if (avgSlashAmount > previousSlashing) {
            uint256 newSlashAmount = avgSlashAmount - previousSlashing;
284:            validatorManager.reportSlashingEvent(validator,
    newSlashAmount);
        }

        // Handle rewards
        if (avgRewardAmount > previousRewards) {
            uint256 newRewardAmount = avgRewardAmount - previousRewards;
290:            validatorManager.reportRewardEvent(validator,
    newRewardAmount);
        }

        // Update validator performance
294:        validatorManager.updateValidatorPerformance(
            validator, avgBalance, avgUptimeScore, avgSpeedScore,
    avgIntegrityScore, avgSelfStakeScore
        );
    }

    // Update lastUpdateTime at the end of successful execution
    lastUpdateTime = block.timestamp;
    emit PerformanceUpdated(block.timestamp);

    return true;
}
```

The issue arises in the `reportSlashingEvent` function, where the transaction can revert at `line 510`.

- ValidatorManager.sol#L510

```
function reportSlashingEvent(address validator, uint256 amount)
    external
    onlyRole(ORACLE_ROLE)
    validatorExists(validator)
    validatorActive(validator)
{
    require(amount > 0, "Invalid slash amount");

    Validator storage val = _validators[_validatorIndexes.get(validator)];
510:    require(val.balance >= amount, "Insufficient stake for slashing");

    // Update balances
    unchecked {
        // These operations cannot overflow:
        // - val.balance ⩾ amount (checked above)
        // - totalBalance ⩾ val.balance (invariant maintained by the
    contract)
517:        val.balance -= amount;
518:        totalBalance -= amount;
    }

    // Update slashing amounts
    totalSlashing += amount;
    validatorSlashing[validator] += amount;

    emit SlashingEventReported(validator, amount);
}
```

Keep in mind that the `balance` is the `old balance`, and the `slash amount` is the `new` amount. Consider the following scenario:

- The `old balance` is `500`.
- Before the new update, some `stakers` delegate to this `validator`, increasing the `balance` to `3000`.
- The `validator` experiences some losses, causing the accumulated `slash amount` to increase by `600`, which is possible.

In such a scenario, the transaction will revert.

```
// val.balance = 500, amount = 600
require(val.balance >= amount, "Insufficient stake for slashing");
```

Moreover, this check for updating the `balance` of this `validator` and the `total balance` at `lines 517-518` is not necessary, as these `balances` are correctly updated in the last function, `updateValidatorPerformance`.

- ValidatorManager.sol#L406

```
function updateValidatorPerformance(
    address validator,
    uint256 balance,
    uint256 uptimeScore,
    uint256 speedScore,
    uint256 integrityScore,
    uint256 selfStakeScore
) external whenNotPaused onlyRole(ORACLE_ROLE) validatorExists(validator)
    validatorActive(validator) {
    uint256 oldBalance = val.balance;

    val.balance = balance;

    // Update total balance
    totalBalance = totalBalance - oldBalance + balance;
}
```

Therefore, we can remove these lines from the `reportSlashingEvent` function.

Recommendations:

```
function reportSlashingEvent(address validator, uint256 amount)
    external
    onlyRole(ORACLE_ROLE)
    validatorExists(validator)
    validatorActive(validator)
{
    require(amount > 0, "Invalid slash amount");

-    Validator storage val = _validators[_validatorIndexes.get(validator)];
-    require(val.balance >= amount, "Insufficient stake for slashing");
-    // Update balances
-    unchecked {
-        // These operations cannot overflow:
-        // - val.balance ⩾ amount (checked above)
-        // - totalBalance ⩾ val.balance (invariant maintained by the
    contract)
-        val.balance -= amount;
-        totalBalance -= amount;
-    }

    // Update slashing amounts
```

```
        totalSlashing += amount;
        validatorSlashing[validator] += amount;

        emit SlashingEventReported(validator, amount);
    }
```

Kinetiq: Fixed in commit @e0eecc28a...

Zenith: Verified

[M-6] The staking limit check is not functioning correctly

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- StakingManager.sol

Description:

There is a `staking limit check` that caps the `total staking` amount, which is normal in many protocols.

- StakingManager.sol#L181

```
function stake()
    external payable nonReentrant whenNotPaused whenStakingNotPaused {
    require(msg.value >= minStakeAmount, "Stake amount below minimum");
    if (maxStakeAmount > 0) {
        require(msg.value <= maxStakeAmount, "Stake amount above maximum");
    }
    if (stakingLimit > 0) {
        require(totalStaked + msg.value <= stakingLimit, "Staking limit
    reached");
    }

    totalStaked += msg.value;
    kHYPE.mint(msg.sender, msg.value);

    _distributeStake(msg.value);
}
```

However, the issue is that the `totalStaked` value only increases when users stake `HYPE` tokens. The `staking limit check` should verify whether the sum of the `current total staked` amount and the new `staking` amount is less than the defined limit. However, `totalStaked` does not account for the `withdrawn` amounts, which are represented in `totalClaimed`. This oversight allows users to prevent `staking` by repeatedly `staking` and `withdrawing`, thereby inflating the `totalStaked` value.

Recommendations:

```
function stake()
    external payable nonReentrant whenNotPaused whenStakingNotPaused {
    require(msg.value >= minStakeAmount, "Stake amount below minimum");
    if (maxStakeAmount > 0) {
        require(msg.value <= maxStakeAmount, "Stake amount above maximum");
    }
    if (stakingLimit > 0) {
-         require(totalStaked + msg.value <= stakingLimit, "Staking limit
    reached");
+^^I^^I require(totalStaked - totalClaimed + msg.value <= stakingLimit,
    "Staking limit reached");
    }

    totalStaked += msg.value;
    kHYPE.mint(msg.sender, msg.value);

    _distributeStake(msg.value);
}
```

Kinetiq: Fixed in @38e05e074d...

Zenith: Verified

4.3 Low Risk

A total of 3 low risk findings were identified.

[L-1] The withdraw amount is sliently downcasted from uint256 to uint64

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- StakingManager.sol

Description:

```
function _withdrawFromValidator(address validator, uint256 amount)
    internal {
    require(validator != address(0), "Invalid validator address");
    require(amount > 0, "Invalid withdrawal amount");

    l1Write.sendTokenDelegate(validator, uint64(amount), true);

    emit ValidatorWithdrawal(validator, amount);
}
```

When withdrawing from validator, the passed in amount is uint256, and then the amount is silently truncated to uint64.

If the requested amount exceed the uint64, the value get truncated and lost

Recommendations:

Use openzeppelin safeCast or

```
    function _withdrawFromValidator(address validator, uint256 amount) internal {
```

```
function _withdrawFromValidator(address validator, uint64 amount) internal {
```

Kinetiq: Fix in commit 4d96561bab

Zenith: Verified

[L-2] The RebalanceRequest struct should specify the StakingManager

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ValidatorManager.sol

Description:

The current `RebalanceRequest` struct is as follows:

- IValidatorManager.sol#L22-L25

```
struct RebalanceRequest {
    address validator; // Validator being rebalanced
    uint256 amount; // Amount to move (negative = withdraw, positive =
    deposit)
}
```

There are several `StakingManagers`, and users can `delegate` to the same `validator` through different `StakingManagers`. This means there would be a need to `withdraw HYPE` tokens from this `validator` through multiple `StakingManagers`. When the `rebalance request` is made in the `rebalanceWithdrawal` function, only the `amount` is recorded.

- ValidatorManager.sol#L239

```
function rebalanceWithdrawal(
    address stakingManager,
    address[] calldata validators,
    uint256[] calldata withdrawalAmounts
) external whenNotPaused nonReentrant onlyRole(MANAGER_ROLE) {
    require(validators.length == withdrawalAmounts.length, "Length
    mismatch");
    require(validators.length > 0, "Empty arrays");

    for (uint256 i = 0; i < validators.length;) {
        require(validators[i] != address(0), "Invalid validator address");
```

```
        // Add rebalance request (this will check for duplicates)
        _addRebalanceRequest(validators[i], withdrawalAmounts[i]);

        unchecked {
            ++i;
        }
    }

    // Trigger withdrawals through StakingManager
    IStakingManager(stakingManager).processValidatorWithdrawals(validators,
    withdrawalAmounts);
}
```

In the `closeRebalanceRequests` function, it is unclear from which `StakingManager` these `HYPE` tokens can be withdrawn.

- ValidatorManager.sol#L307

```
function closeRebalanceRequests(address stakingManager, address[]
    calldata validators)
    external
    whenNotPaused
    nonReentrant
    onlyRole(MANAGER_ROLE)
{

    // Trigger redelegation through StakingManager if there's an amount to
    delegate
    if (totalAmount > 0) {

    IStakingManager(stakingManager).processValidatorRedelegation(totalAmount);
    }
}
```

`Managers` need to be careful, but there is always a risk regarding this.

Therefore, the `RebalanceRequest` struct should include an additional property to specify the `StakingManager`.

Recommendations:

```
struct RebalanceRequest {
    address validator; // Validator being rebalanced
     address stakingManager;
    uint256 amount; // Amount to move (negative = withdraw, positive =
    deposit)
}


    function _addRebalanceRequest(address validator, uint256 withdrawalAmount)
     internal {
function _addRebalanceRequest(address validator, address stakingManager,
    uint256 withdrawalAmount) internal {
    validatorRebalanceRequests[validator] = RebalanceRequest({validator:
        validator, amount: withdrawalAmount});
- validatorRebalanceRequests[validator] = RebalanceRequest({validator:
    validator, stakingManager: stakingManager, amount: withdrawalAmount});
}

function rebalanceWithdrawal(
    address stakingManager,
    address[] calldata validators,
    uint256[] calldata withdrawalAmounts
) external whenNotPaused nonReentrant onlyRole(MANAGER_ROLE) {
    for (uint256 i = 0; i < validators.length;) {
        require(validators[i] != address(0), "Invalid validator address");

        // Add rebalance request (this will check for duplicates)
        _addRebalanceRequest(validators[i], withdrawalAmounts[i]);

    _addRebalanceRequest(validators[i], stakingManager, withdrawalAmounts[i]);

        unchecked {
            ++i;
        }
    }
}

function requestEmergencyWithdrawal(address stakingManager,
    address validator, uint256 amount)
    external
    onlyRole(SENTINEL_ROLE)
```

```
    whenNotPaused
{

    // Create rebalance request
    _addRebalanceRequest(validator, amount);
     _addRebalanceRequest(validator, stakingManager, amount);
}


function closeRebalanceRequests(address stakingManager, address[]
    calldata validators)
    external
    whenNotPaused
    nonReentrant
    onlyRole(MANAGER_ROLE)
{
    for (uint256 i = 0; i < validators.length;) {
        address validator = validators[i];
        require(_validatorsWithPendingRebalance.contains(validator), "No
    pending request");

        // Add amount to total for redelegation
        RebalanceRequest memory request
    = validatorRebalanceRequests[validator];

         require(request.stakingManager == stakingManager, "");

        totalAmount += request.amount;

        // Clear the rebalance request
        delete validatorRebalanceRequests[validator];
        _validatorsWithPendingRebalance.remove(validator);

        emit RebalanceRequestClosed(validator, request.amount);

        unchecked {
            ++i;
        }
    }
}
```

Kinetiq: Fixed in commit @7d808fb034...

Zenith: Verified.

[L-3] It's impossible to set a non-zero maxStakeAmount when the stakingLimit is 0

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- StakingManager.sol

Description:

In the `StakingManager`, a `stakingLimit` of 0 means that there is no limit on the total `staked amount`. The `maxStakeAmount` is used to limit the individual `staking` amount. However, the `setMaxStakeAmount` function reverts when the current `stakingLimit` is 0 and the `newMaxStakeAmount` is non-zero.

- StakingManager.sol#L420

```
function setMaxStakeAmount(uint256 newMaxStakeAmount)
    external onlyRole(MANAGER_ROLE) {
    if (newMaxStakeAmount > 0) {
        require(newMaxStakeAmount > minStakeAmount, "Max stake must be
    greater than min");
        require(newMaxStakeAmount < stakingLimit, "Max stake must be less
    than limit");
    }
    maxStakeAmount = newMaxStakeAmount;
    emit MaxStakeAmountUpdated(newMaxStakeAmount);
}
```

Recommendations:

```
function setMaxStakeAmount(uint256 newMaxStakeAmount)
    external onlyRole(MANAGER_ROLE) {
    if (newMaxStakeAmount > 0) {
        require(newMaxStakeAmount > minStakeAmount, "Max stake must be
    greater than min");
```

```
+^^I^^Iif (stakingLimit > 0) {
            require(newMaxStakeAmount < stakingLimit, "Max stake must be less
    than limit");
+^^I^^I}
    }
    maxStakeAmount = newMaxStakeAmount;
    emit MaxStakeAmountUpdated(newMaxStakeAmount);
}
```

Kinetiq: Fixed in commit @eb25a3eb6...

Zenith: Verified.

4.4 Informational

A total of 4 informational findings were identified.

[I-1] The check for stakingLimit is insufficient when maxStakeAmount is 0

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- StakingManager.sol

Description:

Update the check for `stakingLimit` as follows:

```
function initialize(
    address admin,
    address operator,
    address manager,
    address _pauserRegistry,
    address _kHYPE,
    address _validatorManager,
    uint256 _minStakeAmount,
    uint256 _maxStakeAmount,
    uint256 _stakingLimit
) public initializer {
    if (_stakingLimit > 0) {
        require(_stakingLimit > _maxStakeAmount, "Invalid staking limit");
        if (_maxStakeAmount > 0) {

    require(_stakingLimit > _maxStakeAmount, "Invalid staking limit");
        } else {

    require(_stakingLimit > _minStakeAmount, "Invalid staking limit");
        }
```

```
    }
}

function setStakingLimit(uint256 newStakingLimit)
    external onlyRole(MANAGER_ROLE) {
    if (newStakingLimit > 0) {
        require(newStakingLimit > maxStakeAmount, "Invalid staking limit");
        if (maxStakeAmount > 0) {

    require(newStakingLimit > maxStakeAmount, "Invalid staking limit");
        } else {

    require(newStakingLimit > minStakeAmount, "Invalid staking limit");
        }
    }
    stakingLimit = newStakingLimit;
    emit StakingLimitUpdated(newStakingLimit);
}
```

Recommendations:

Kinetiq: Fixed with @00b166614c...

Zenith: Verified.

[I-2] `GeneratePerformance` consider large loop in OracleManager.sol

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- OracleManager.sol

Description:

```
function generatePerformance()
    external whenNotPaused onlyRole(OPERATOR_ROLE) returns (bool) {
```

The function `generateReport` contains long loop and iterate over all validator and oracle several times.

Recommendations:

Add a function to update individual validator score performance report

Kinetiq: Fixed in commit @b19d899149a...

Zenith: Verified. The fix ensures the operator can update the individual operator's performance.

[I-3] Unused code in StakingManager.sol

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- StakingManager.sol

Description:

1. isAutoDelegation is not used and always set to true
2. ReentrancyGuard is imported and initialized but not used.
3. MAX_HISTORY_LENGTH is unused in `ValidationManager`.

```
uint256 public constant MAX_HISTORY_LENGTH = 30;
```

Recommendations:

Unused code in StakingManager.sol should be handled or removed.

Kinetiq: Fixed in commit c66f1ac10dc

Zenith: Verified

[I-4] There is no need to use both validatorExists and validatorActive modifiers simultaneously

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: High

Target

- ValidatorManager.sol

Description:

In the `validatorActive` modifier, the `validatorExists` modifier is already applied.

- ValidatorManager.sol#L441

```
modifier validatorActive(address validator) {
    require(validatorActiveState(validator), "Validator not active");
    _;
}

function validatorActiveState(address validator)
    public view validatorExists(validator) returns (bool) {
    return _validators[_validatorIndexes.get(validator)].active;
}
```

Therefore, there is no need to use both modifiers in the same function.

- ValidatorManager.sol#L384

```
function updateValidatorPerformance(
    address validator,
    uint256 balance,
    uint256 uptimeScore,
    uint256 speedScore,
    uint256 integrityScore,
    uint256 selfStakeScore
) external whenNotPaused onlyRole(ORACLE_ROLE) validatorExists(validator)
    validatorActive(validator) {
}
```

Recommendations:

Remove the `validatorExists` modifier in the `updateValidatorPerformance`, `reportRewardEvent`, `reportSlashingEvent`, and `setDelegation` functions.

Kinetiq: Fixed with @75753123a3...

Zenith: Verified.