Audit Report – kinetiq

Repository: [https://github.com/code-423n4/2025-04-kinetiq]

Date: [29.04.2025]

Auditor: [Harisuthan]

Summary:
Total issues found: [5 (high and medium included)]
Severity breakdown: [3 high, 2 medium]

Issue H-1: `receive()` re-stakes withdrawal funds, blocking user withdrawals

Summary

The contract's `receive()` function is intended to automatically stake HYPE when a user directly sends funds to the contract.
However, due to missing sender validation, the receive() function is also triggered when HYPE is returned from validators during undelegation.
Instead of keeping this HYPE available for user withdrawals, the contract re-stakes it again.

As a result, when users try to confirm their withdrawal, the contract has no HYPE balance, causing the withdrawal to revert.

Vulnerability Details

The `receive()` function does not distinguish who is sending HYPE:

✅ Intended sender: User

❌ Unintended sender: Validator / system undelegation

When validators send back HYPE during withdrawal processing, the following happens:

The HYPE enters the contract

receive() is triggered

receive() automatically calls stake()

Withdrawal funds are re-delegated instead of staying in the contract

This breaks the withdrawal flow.

Step-by-Step Attack / Failure Flow

User stakes 1 HYPE

Operator calls processL1Operations

HYPE is delegated to validators

User requests a withdrawal

Operator calls processL1Operations

Validators undelegate and send HYPE back to the contract

Returned HYPE triggers receive()

HYPE is staked again instead of held for withdrawal

User calls confirmWithdrawal

❌ Reverts because no HYPE is available in the contract

Impact

Users cannot withdraw their HYPE

Withdrawals are permanently blocked

This can result in a loss of user funds

Breaks a core protocol guarantee: users can always exit

Proof of Concept

The following test demonstrates that confirmWithdrawal() reverts even when the user has a valid withdrawal queued.

```
function test_WithdrawalConfirmationRevertsWhenNoHYPE() public {
    /**
     * Test goal:
     * ----------
     * Verify that `confirmWithdrawal` reverts when the
     * StakingManager does not have enough native HYPE,
     * even though the user has a valid withdrawal queued.
     */
```

```
        uint256 stakeAmount = 1 ether;

        // Fund user
        vm.deal(user, stakeAmount);

        // Setup validator
        vm.startPrank(manager);
        validatorManager.activateValidator(validator);
        validatorManager.setDelegation(address(stakingManager), validator);
        vm.stopPrank();

        // User stakes HYPE
        vm.prank(user);
        stakingManager.stake{value: stakeAmount}();

        // Delegate to validator
        vm.prank(operator);
        stakingManager.processL1Operations(0);

        // User queues withdrawal
        vm.startPrank(user);
        kHYPE.approve(address(stakingManager), stakeAmount);
        stakingManager.queueWithdrawal(stakeAmount);
        vm.stopPrank();

        // Undelegate from validator
        vm.prank(operator);
        stakingManager.processL1Operations(0);

        // Simulate unbonding period
        vm.warp(block.timestamp + 7 days);

        // Withdrawal confirmation fails
        vm.prank(user);
        vm.expectRevert();
        stakingManager.confirmWithdrawal(0);
    }
```

Run with:

```
forge test --mt test_WithdrawalConfirmationRevertsWhenNoHYPE -vvv
```

Root Cause

receive() automatically stakes all incoming HYPE

No check exists to verify:

Who sent the HYPE

Whether the HYPE is meant for staking or withdrawals

Recommendation

Add sender-based validation in receive().

Issue H-2 : `hypeBuffer` is not restored when `withdrawal` is cancelled, causing incorrect buffer accounting

Summary:

The `withdrawal` flow is intended to temporarily reduce the HYPE buffer when a user queues a withdrawal and restore it if the withdrawal is cancelled, but due to a missing buffer update on withdrawal cancellation, the hypeBuffer remains reduced. This causes the contract to think it has less available HYPE than it actually does.

Vulnerability Detail:

When a user requests a withdrawal, the contract first tries to fulfill it from the hypeBuffer. This buffer represents available HYPE that can be used immediately without touching validators.

How the bug occurs (step by step):

1. A user calls queueWithdrawal.
2. The function calculates the withdrawal amount and calls _withdrawFromValidator.
3. Inside _withdrawFromValidator, for user withdrawals:
4. The contract checks the current hypeBuffer.
5. If enough buffer exists, it reduces hypeBuffer and uses it to cover part or all of the withdrawal.

6. The withdrawal request is stored, but funds are not yet finalized.
7. If the user later cancels the withdrawal:
8. The withdrawal request is removed or invalidated.

However, hypeBuffer is NOT increased back by the amount previously taken.

As a result, the contract permanently thinks the buffer is lower than it really is.

Impact:

As a result of this vulnerability, the HYPE buffer becomes permanently undercounted.

Proof of Code:

```
function test_ConfirmBufferBug() public {
    /**
     * Test goal:
     * ----------
     * Verify that cancelling a queued withdrawal does NOT correctly
     * update the `hypeBuffer`, leaving the buffer in an inconsistent s
     */

    uint256 stakeAmount = 1 ether;
    uint256 withdrawalId = 0;

    /* ------------------------------------------------------------ */
    /*                         Test Setup                           */
    /* ------------------------------------------------------------ */

    // Activate validator and set delegation
    vm.startPrank(manager);
    validatorManager.activateValidator(validator);
    validatorManager.setDelegation(address(stakingManager), validator);
    vm.stopPrank();

    // Fund user and stake HYPE
    vm.deal(user, stakeAmount);
    vm.prank(user);
    stakingManager.stake{value: stakeAmount}();
```

```
        /* ------------------------------------------------------------ */
        /*                    Queue Withdrawal                          */
        /* ------------------------------------------------------------ */

        vm.startPrank(user);
        kHYPE.approve(address(stakingManager), stakeAmount);
        stakingManager.queueWithdrawal(stakeAmount);
        vm.stopPrank();

        // Buffer after withdrawal request
        uint256 hypeBufferAfterQueue =
            stakingManager.hypeBuffer();

        /* ------------------------------------------------------------ */
        /*                   Cancel Withdrawal                          */
        /* ------------------------------------------------------------ */

        // Manager cancels the queued withdrawal
        vm.prank(manager);
        stakingManager.cancelWithdrawal(user, withdrawalId);

        // Buffer after withdrawal cancellation
        uint256 hypeBufferAfterCancel =
            stakingManager.hypeBuffer();

        /* ------------------------------------------------------------ */
        /*                        Assertion                             */
        /* ------------------------------------------------------------ */

        // Buffer remains unchanged even after cancellation,
        // indicating improper buffer accounting
        assertEq(
            hypeBufferAfterQueue,
            hypeBufferAfterCancel
        );
    }
```

Run this code : `forge test --mt test_ConfirmBufferBug`

There is no corresponding buffer restoration when a withdrawal is cancelled.

Recommendation:

When a withdrawal is cancelled:

Track how much hypeBuffer was used for that withdrawal

Restore the same amount back to hypeBuffer

Issue H-3 : Users Who Queue `Withdrawal` Before A Slashing Event Can Receive More HYPE Than Intended, Causing Loss For Later Users

Summary:

The `confirmWithdrawal` function is intended to pay users their current HYPE based on queued kHYPE, but due to using the stored hypeAmount from queue time, users who queued withdrawals before a slashing event can claim more HYPE than the updated ratio allows. This causes later users to receive less or zero HYPE, resulting in loss for them.

Vulnerability Detail:

When a user calls queueWithdrawal, their HYPE amount is calculated from kHYPE using the current exchange ratio:

```
uint256 hypeAmount = stakingAccountant.kHYPEToHYPE(postFeeKHYPE);
_withdrawalRequests[msg.sender][withdrawalId] = WithdrawalRequest({
    hypeAmount: hypeAmount,
    kHYPEAmount: postFeeKHYPE,
    kHYPEFee: kHYPEFee,
    timestamp: block.timestamp
});
```

This hypeAmount is stored in the withdrawal request and represents the promised HYPE at the time of queueing.

If a slashing event occurs before the user confirms the withdrawal, the total HYPE balance decreases, changing the actual kHYPE-to-HYPE ratio.

When confirmWithdrawal is called:

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

`_processConfirmation` simply returns the stored hypeAmount.

It does not account for the updated ratio after slashing.

Early queued withdrawals are confirmed using the old, higher amount.

Later queued users get the remaining HYPE, which may be zero or much less than expected, even though their kHYPE is valid.

Impact:

Users who queue later can receive less or zero HYPE, despite having valid kHYPE.

Losses occur even if sufficient total HYPE exists.

Protocol's withdrawal mechanism becomes unfair.

Proof of Code:

queueWithdrawal stores hypeAmount at queue time:

```
_withdrawalRequests[msg.sender][withdrawalId] = WithdrawalRequest({
    hypeAmount: hypeAmount,
    ...
});
```

`confirmWithdrawal` pays the stored amount:

```
uint256 amount = _processConfirmation(msg.sender, withdrawalId);
require(address(this).balance >= amount, "Insufficient contract balance
```

No recalculation of HYPE based on current total after slashing.

Recommendation:

Do not use the stored hypeAmount directly when confirming withdrawals.

Recalculate HYPE at confirmation time based on the current kHYPE-to-HYPE ratio.

Issue M-1 : Missing `whenWithdrawalNotPaused` check in `confirmWithdrawal` allows withdrawal when paused

Summary:

The `confirmWithdrawal` function is intended to allow users to complete withdrawals only when withdrawals are not paused, but due to a missing whenWithdrawalNotPaused check, users can still call confirmWithdrawal even when withdrawals are paused. This breaks the contract's withdrawal rules.

Vulnerability Detail:

The contract includes a withdrawal pause mechanism to temporarily stop withdrawals when needed (for safety, upgrades, or emergencies).

However, the confirmWithdrawal function does not enforce this rule.

How the bug occurs (step by step):

1. The admin pauses withdrawals using the withdrawal pause mechanism.
2. While withdrawals are paused, users are not supposed to complete withdrawals.
3. A user calls confirmWithdrawal.

Since confirmWithdrawal does not check whenWithdrawalNotPaused, the function executes successfully.

The withdrawal is confirmed even though withdrawals are paused.

Impact:

As a result of this vulnerability, the withdrawal pause mechanism can be bypassed.
Users may withdraw funds even when withdrawals are intentionally paused.

This can lead to:

Loss of protocol control during emergencies

Inability to stop fund outflows

Violation of protocol safety assumptions

Proof of Code:

`confirmWithdrawal` function

Missing `whenWithdrawalNotPaused` modifier/check

Other withdrawal-related functions correctly enforce the pause, but confirmWithdrawal does not

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

          stakingAccountant.recordClaim(amount);

          // Process withdrawal using call instead of transfer
          (bool success, ) = payable(msg.sender).call{value: amount}("");
          require(success, "Transfer failed");
      }
```

Recommendation:

Add the `whenWithdrawalNotPaused` modifier (or equivalent pause check) to the confirmWithdrawal function.

This ensures that no part of the withdrawal process can be completed while withdrawals are paused, maintaining consistent and secure behavior.

Issue M-2 : Unnecessary balance check in `processValidatorRedelegation` can

cause unwanted reverts

Summary:

The `processValidatorRedelegation` function is intended to rebalance already-held stake by redistributing it, but due to an unnecessary contract balance check, it may revert even when the rebalance logic itself is valid. This check does not add safety and can break normal rebalance operations.

Vulnerability Detail:

The `processValidatorRedelegation` function is used during validator rebalance operations. Its purpose is to redistribute stake internally, not to accept new ETH or perform an external transfer.

However, the function includes this check:

```
require(address(this).balance >= amount, "Insufficient balance");
```

How the issue occurs (step by step):

1. The ValidatorManager calls processValidatorRedelegation during a rebalance.
2. The function is supposed to distribute stake using _distributeStake.
3. `_distributeStake` handles internal accounting and delegation logic.
4. Before reaching that logic, the function checks address(this).balance.
5. Even if the rebalance logic does not require this balance check, the function can revert.

This causes the entire rebalance operation to fail unnecessarily.

Impact:

As a result of this issue, validator rebalance operations may fail unexpectedly.

This can lead to:

Rebalance transactions reverting without real risk

Validators not being properly rebalanced

Disrupted staking operations and protocol inefficiency

Proof of Code:

```
function processValidatorRedelegation(uint256 amount)
    external
    nonReentrant
    whenNotPaused
{
    require(msg.sender == address(validatorManager), "Only ValidatorMar
    require(amount > 0, "Invalid amount");
    require(address(this).balance >= amount, "Insufficient balance"); /

    _distributeStake(amount, OperationType.RebalanceDeposit);
}
```

Recommendation:

Remove the `require(address(this).balance >= amount)` check from `processValidatorRedelegation`.