# BLOCKCHAIN IN AUDIT

PRESENTED BY: ANSELMO SANCHEZ

Prepared by: Anselmo Ramon Sanchez Titla

# Table of Contents



# Protocol Summary

This decentralized staking protocol is built around HYPE tokens and a modular smart contract architecture. Users can stake HYPE to receive kHYPE, which represents their staked position. The StakingManager handles deposits, withdrawals, and buffer management to ensure liquidity. ValidatorManager oversees validators, managing activation, rewards, slashing, and fund delegation. Performance data is sourced and verified through oracles managed by the OracleManager, ensuring validator reliability. The StakingAccountant tracks total staked assets and maintains the exchange rate between HYPE and kHYPE. A buffer is used to fulfill withdrawals instantly, but only up to a certain limit. The system balances decentralization with practical mechanisms for performance, accountability, and fund management.

# Disclaimer

The Anselmo Ramon Sanchez Titla team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

# Risk Classification

| | **Impact** | | |
|---|---|---|---|
| | High | Medium | Low |

| | | Impact | | |
|---|---|---|---|---|
| | High | H | H/M | M |
| Likelihood | Medium | H/M | M | M/L |
| | Low | M | M/L | L |

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

# Audit Details

**The findings described in this document correspond to the following commit hash:**

- Commit Hash: xxxxx

## Scope

```
*See [scope.txt](https://github.com/code-423n4/2025-04-
kinetiq/blob/main/scope.txt)*

| File                               | Logic Contracts | Interfaces | nSLOC    |
|------------------------------------|-----------------|------------|----------|
| /src/oracles/DefaultAdapter.sol | 1               | 1          | 24       |
| /src/oracles/DefaultOracle.sol  | 1               | ****       | 77       |
| /src/oracles/IOracleAdapter.sol | ****            | 1          | 4        |
| /src/KHYPE.sol                  | 1               | ****       | 43       |
| /src/OracleManager.sol          | 1               | ****       | 212      |
| /src/PauserRegistry.sol         | 1               | ****       | 75       |
| /src/StakingAccountant.sol      | 1               | ****       | 120      |
| /src/StakingManager.sol         | 1               | ****       | 543      |
| /src/ValidatorManager.sol       | 1               | ****       | 234      |
| **Totals**                         | **8**           | **2**      | **1332** |
```

## Roles

Roles hierarchy:

```
DEFAULT_ADMIN_ROLE
├── StakingManager
│   ├── OPERATOR_ROLE
│   ├── MANAGER_ROLE
│   ├── TREASURY_ROLE
│   └── SENTINEL_ROLE
├── ValidatorManager
```

```
│   ├── MANAGER_ROLE
│   └── ORACLE_MANAGER_ROLE(contract)
├── OracleManager
│   ├── MANAGER_ROLE
│   └── OPERATOR_ROLE
├── StakingAccountant
│   └── MANAGER_ROLE
├── KHYPE Token
│   ├── MINTER_ROLE(contract)
│   └── BURNER_ROLE(contract)
└── PauserRegistry
    ├── PAUSER_ROLE
    ├── UNPAUSER_ROLE
    └── PAUSE_ALL_ROLE
```

1. Admin Trust: The DEFAULT_ADMIN_ROLE has full control over the protocol and must be a highly trusted entity, ideally a multi-sig or governance contract.
2. Operational Trust:
    - OPERATOR_ROLE: Must reliably execute L1 operations and manage the operation queue
    - MANAGER_ROLE: Has significant control over protocol parameters across multiple contracts
    - ORACLE_MANAGER_ROLE: Controls the flow of external data into the protocol
3. Emergency Controls:
    - SENTINEL_ROLE: Likely has emergency powers to protect the protocol
    - PAUSER_ROLE/PAUSE_ALL_ROLE: Can halt protocol operations in emergencies
4. Financial Trust:
    - TREASURY_ROLE: Receives protocol revenues
    - MINTER_ROLE/BURNER_ROLE: Controls token supply

# Executive Summary

We spent 50 hours with 3 auditors using manual review, finding the below vulnerabilities classified from high to informational

## Issues found

| Severity | Number of issues found |
|---|---|
| High | 0 |
| Medium | 1 |
| Low | 0 |
| Infomational | 0 |
| Gas | 0 |

| Severity | Number of issues found |
| --- | --- |
| total | 1 |

# Findings

## Medium

### [M-1] Not all staked ETH is held in the `StakingManager` contract, which means some users may be unable to withdraw their ETH.

**Severity:**

- Impact: Medium
- Likelihood: Medium

**Description:**

When users stake their tokens, the funds are first stored in the buffer up to a certain limit. Any excess is transferred to another contract. As a result, if the total withdrawal requests exceed the ETH available in the buffer, not all users will be able to withdraw their funds.

**Impact:**

A large portion of the staked tokens will be locked in another smart contract, separate from the buffer, unless a centralized authority transfers the required tokens back to the buffer. This means that not all users will be able to withdraw their ETH when they request it.

**Proof of Concept:**

▶ Proof of code

Let's assume that 4,000 users each deposit 1 ETH. The target buffer is set to 1% of the limit (10,000 ETH), which equals 100 ETH. If suddenly around 150 users try to withdraw their ETH, only the first 100 users will be able to withdraw from the buffer. From user 101 onward, withdrawals will be blocked due to insufficient funds in the buffer, which was only intended to hold 100 ETH (1% of the total).

Please paste the following code into `StakingManager.t.sol` to run the test and verify that not all users are able to withdraw their staked ETH.

```
    uint256 constant NUM_DEPOSITS = 150;
    function testNotAllUsersCanWithdraw() public {

        // Set up delegation first
        vm.startPrank(manager);
        validatorManager.activateValidator(validator);
```

```
        validatorManager.setDelegation(address(stakingManager), validator);
        vm.stopPrank();

        // Set target buffer
        vm.prank(manager);
        uint target = 100 ether;
        stakingManager.setTargetBuffer(target);

        // Users deposit or stake 1 ether each one
        address[NUM_DEPOSITS] memory users;
        for(uint i = 0; i<NUM_DEPOSITS; i++) {
            users[i] = makeAddr(vm.toString(i+1));
            vm.deal(users[i], 1 ether);
            vm.prank(users[i]);
            stakingManager.stake{value: 1 ether}();
        }

        // Approve and queue withdrawal
        for ( uint256 j = 0; j<NUM_DEPOSITS; j++){
            vm.startPrank(users[j]);
            kHYPE.approve(address(stakingManager), 1 ether);
            stakingManager.queueWithdrawal(1 ether);
            vm.stopPrank();
        }

        // Set withdrawal delay to 0 to allow immediate confirmation
        vm.prank(manager);
        stakingManager.setWithdrawalDelay(0);

        // Only the first 100 users will be able to withdraw their ETH
        for (uint i=0; i<100; i++){
            vm.prank(users[i]);
            stakingManager.confirmWithdrawal(0);
        }

        // From user 101 onwards, withdrawals will be blocked due to
insufficient funds in the buffer
        for (uint i=100; i<150; i++){
            vm.prank(users[i]);
            vm.expectRevert("Insufficient contract balance");
            stakingManager.confirmWithdrawal(0);
        }
    }
```

**Recommended Mitigation**

Implement a mechanism to transfer staked tokens to the buffer, ensuring that all users who request withdrawals are able to receive their tokens.