hexens x Valantis

Security Review Report for Valantis

May 2025

Table of Contents

1. About Hexens

Hexens is a pioneering cybersecurity firm dedicated to establishing robust security standards for Web3 infrastructure, driving secure mass adoption through innovative protection technology and frameworks. As an industry elite experts in blockchain security, we deliver comprehensive audit solutions across specialized domains, including infrastructure security, Zero Knowledge Proof, novel cryptography, DeFi protocols, and NFTs.

Our methodology combines industry-standard security practices combined with unique methodology of two teams per audit, continuously advancing the field of Web3 security. This innovative approach has earned us recognition from industry leaders.

Since our founding in 2021, we have built an exceptional portfolio of enterprise clients, including major blockchain ecosystems and Web3 platforms.

2. Executive Summary

This report covered a new withdrawal module for Valantis' Stake Exchange AMM (STEX-AMM), namely for the kHYPE token. The module is similar to the stHYPE module, but instead works with the redemption mechanism specific to kHYPE.

Our security assessment was a full review of the new smart contracts in scope, spanning a total of 1 week.

During our audit, we did not identify any major severity vulnerabilities.

We also identified several minor severity vulnerabilities and code optimisations.

All of our reported issues were fixed by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

3. Security Review Details

- **Review Led by**

SoonChan Hwang, Lead Security Researcher
Jahyun Koo, Lead Security Researcher

- **Scope**

The analyzed resources are located on:

🔗 https://github.com/ValantisLabs/valantis-stex-khype

📌 **Commit:** `092823a6c16d2e2ffb5614db9753450de8fa4e40`

The issues described in this report were fixed in the following commit:

🔗 https://github.com/ValantisLabs/valantis-stex-khype

📌 **Commit:** `1165090ad44a42de5aa486a37cd712d7b3293a9b`

- **Changelog**

26 May 2025	Audit start
02 June 2025	Initial report
03 June 2025	Revision received
04 June 2025	Final report

4. Severity Structure

The vulnerability severity is calculated based on two components:

1. Impact of the vulnerability
2. Probability of the vulnerability

Impact	Probability			
	Rare	Unlikely	Likely	Very likely
Low	Low	Low	Medium	Medium
Medium	Low	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

- **Severity Characteristics**

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical	Vulnerabilities that are highly likely to be exploited and can lead to catastrophic outcomes, such as total loss of protocol funds, unauthorized governance control, or permanent disruption of contract functionality.
High	Vulnerabilities that are likely to be exploited and can cause significant financial losses or severe operational disruptions, such as partial fund theft or temporary asset freezing.

Medium	Vulnerabilities that may be exploited under specific conditions and result in moderate harm, such as operational disruptions or limited financial impact without direct profit to the attacker.
Low	Vulnerabilities with low exploitation likelihood or minimal impact, affecting usability or efficiency but posing no significant security risk.
Informational	Issues that do not pose an immediate security risk but are relevant to best practices, code quality, or potential optimizations.

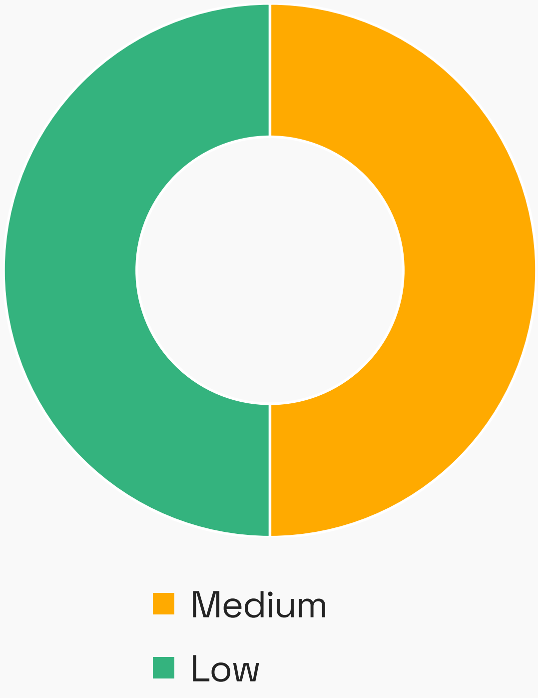
- **Issue Symbolic Codes**

Each identified and validated issue is assigned a unique symbolic code during the security research stage.

Due to the structure of the vulnerability reporting flow, some rejected issues may be missing.

5. Findings Summary

Severity	Number of findings
Critical	0
High	0
Medium	1
Low	1
Informational	0
Total:	**2**



6. Weaknesses

This section contains the list of discovered weaknesses.

VLTS4-2 | Delayed State Updates After Withdrawal Confirmation

Fixed

Severity:	Medium	Probability:	Unlikely	Impact:	Medium

Path:

src/kHYPEWithdrawalModule.sol
.../kinetiq/src/StakingManager.sol
.../kinetiq/src/StakingAccountant.sol

Description:

The **_confirmWithdrawal** function in **kHYPEWithdrawalModule** processes pending withdrawal requests and receive native tokens from the staking manager. However, after a successful confirmation, the contract doesn't immediately call **_update(false)** to reconcile its internal accounting state.

```
    function _confirmWithdrawal(uint256 id) private returns (bool isConfirmed)
{
        IStakingManager.WithdrawalRequest memory request =
            IStakingManager(stakingManager).withdrawalRequests(address(this),
id);

        // Request does not exist, has been cancelled,
        // or has already been confirmed
        if (request.hypeAmount == 0) return false;

        // Request is not yet ready to claim
        if (block.timestamp < request.timestamp +
IStakingManager(stakingManager).withdrawalDelay()) {
            return false;
        }

        uint256 preBalance = address(this).balance;
```

```
        IStakingManager(stakingManager).confirmWithdrawal(id);

        isConfirmed = address(this).balance >= preBalance + request.hypeAmount;

        emit WithdrawalRequestConfirmed(id, request.hypeAmount, isConfirmed);
    }
```

When a withdrawal is initially queued via **_unstakeToken0**, the amount of HYPE to be received is calculated based on the current exchange ratio and stored in the withdrawal request.

```
    function queueWithdrawal(uint256 kHYPEAmount) external nonReentrant
whenNotPaused whenWithdrawalNotPaused {
        ...
        uint256 hypeAmount = stakingAccountant.kHYPEToHYPE(postFeeKHYPE);

        // Lock kHYPE tokens
        kHYPE.transferFrom(msg.sender, address(this), kHYPEAmount);

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

After **_confirmWithdrawal** executes successfully and the contract receives native tokens, the internal accounting state (particularly **_amountToken0PendingUnstaking** and excess native balance tracking) is not immediately updated. If the exchange ratio changes between confirmation and a subsequent **update** call, the calculations will use the new ratio to reconcile previously confirmed withdrawals.

```
    function _update(bool isPoolRebalance) private {
        ...
        if (!isPoolRebalance) {
            uint256 amountToken0PendingUnstakingCache =
_amountToken0PendingUnstaking;
            uint256 excessToken0Balance = convertToToken0(excessNativeBalance);
```

```
            if (amountToken0PendingUnstakingCache > excessToken0Balance) {
                _amountToken0PendingUnstaking =
amountToken0PendingUnstakingCache - excessToken0Balance;
            } else {
                _amountToken0PendingUnstaking = 0;
            }
        }
        ...
        emit Update();
    }
```

Unlike **stHYPEWithdrawalModule** which uses a rebasing token with 1:1 conversion, **kHYPEWithdrawalModule** relies on **StakingAccountant**'s dynamic exchange ratio calculations.

```
    function _getExchangeRatio() internal view returns (uint256) {
        // Calculate total kHYPE supply across all unique tokens
        uint256 totalKHYPESupply = 0;
        uint256 uniqueTokenCount = _uniqueTokens.length();

        // Sum up the supply of each unique token
        for (uint256 i = 0; i < uniqueTokenCount; i++) {
            address tokenAddress = _uniqueTokens.at(i);
            totalKHYPESupply += IERC20(tokenAddress).totalSupply();
        }

        // Return 1:1 ratio when no kHYPE has been minted yet
        if (totalKHYPESupply == 0) {
            return 1e18; // 1:1 ratio with 18 decimals precision
        }

        // Calculate total HYPE (in 8 decimals)
        uint256 rewardsAmount = validatorManager.totalRewards();
        uint256 slashingAmount = validatorManager.totalSlashing();
        uint256 totalHYPE = totalStaked + rewardsAmount - totalClaimed -
slashingAmount;

        // Calculate ratio with 18 decimals precision
        return Math.mulDiv(totalHYPE, 1e18, totalKHYPESupply);
    }
```

This discrepancy can lead to:

1. Incorrect updates to **_amountToken0PendingUnstaking**
2. Delays or incorrect ordering in LP withdrawal processing
3. Inefficient utilization of liquidity
4. Compounding accounting discrepancies

Remediation:

Call **_update(false)** immediately after a successful withdrawal confirmation.

```
function confirmWithdrawal(uint256 _id) external nonReentrant returns (bool
isConfirmed) {
    isConfirmed = _confirmWithdrawal(_id);

    // Update accounting state immediately after confirmation
    if (isConfirmed) {
        _update(false);
    }

    return isConfirmed;
}
```

VLTS4-1 | Multi-Entry Point Token risk in Sweep Function (stHYPEWithdrawalModule)

Fixed

Severity:	Low	Probability:	Rare	Impact:	Low

Path:

`src/stHYPEWithdrawalModule.sol#L341-L360`

Description:

The **sweep** function in **stHYPEWithdrawalModule** only checks for and prevents sweeping of the **token0** (**stHYPE**) but does not explicitly check for its dual-entry point token (**wstHYPE**). Since **stHYPE** and **wstHYPE** share the same underlying data as documented, the owner could sweep **wstHYPE** tokens, which would affect the same balances as **stHYPE**. While this issue requires owner action to exploit, it represents a potential risk to the protocol's token accounting if the owner mistakenly attempts to sweep these tokens.

```
function sweep(address _token, address _recipient) external onlyOwner {
    if (_token == address(0)) revert stHYPEWithdrawalModule__ZeroAddress();
    if (_recipient == address(0)) {
        revert stHYPEWithdrawalModule__ZeroAddress();
    }

    if (_token == ISTEXAMM(stex).token0()) {
        revert stHYPEWithdrawalModule__sweep_Token0CannotBeSweeped();
    }
    if (_token == ISTEXAMM(stex).token1()) {
        revert stHYPEWithdrawalModule__sweep_Token1CannotBeSweeped();
    }

    uint256 balance = ERC20(_token).balanceOf(address(this));
    if (balance > 0) {
        ERC20(_token).safeTransfer(_recipient, balance);

        emit Sweep(_token, _recipient, balance);
    }
}
```

Remediation:

Add an explicit check for wstHYPE in the **sweep** function:

```
if (_token == ISTEXAMM(stex).token0() || _token == WSTHYPE_ADDRESS) {
    revert stHYPEWithdrawalModule__sweep_Token0CannotBeSweeped();
}
```

hexens x Valantis