DeHive

SMART CONTRACT AUDIT

ZOKYO.

September 6th, 2021 | v. 2.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges

TECHNICAL SUMMARY

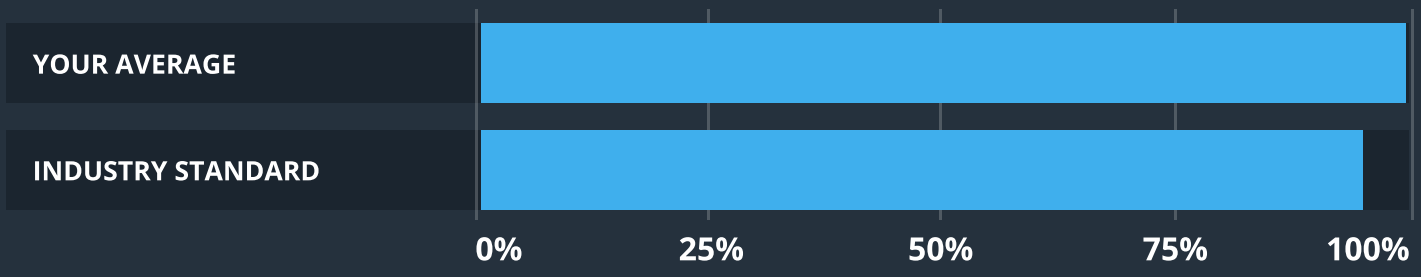
This document outlines the overall security of the DeHive smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the DeHive smart contract codebase for quality, security, and correctness.

Contract Status

There were 3 critical issues found during the audit.

Testable Code



The testable code is 99.66%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the DeHive team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the DeHive repository.

V1 Keccak256 – ce97ba268d227a9078ccf33fa7be152fd3f025368800d80fd3ed91ff554b4415
V2 Keccak256 – f0775749485bf87c1cf8d177a567d5cd10a41c2e14da8c19e5e50312f62d3bb8
V3 Keccak256 – 21b1587bf72002cf79b2a5b117ce3a9d2886b6eb

Requirements:
https://drive.google.com/drive/folders/1_XTtKmGFnWKs86zro_sq7njb0_GUPf1w?usp=sharing

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of DeHive smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

SUMMARY

Zokyo team has conducted a security audit. All the issues found and the recommendations are described in the “Complete Analysis” section. Among the findings, there are 3 critical issues, three of which are successfully resolved. Apart from critical issues, we found 3 issues with high severity status and one issue with low severity status.

During the auditing process we came up with a couple of suggestions about the potential improvements and marked all the findings with an “Informational” tag. DeHive team took into consideration all the points and implemented the necessary fixes and improvements.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the ability of the contract to compile or operate in a significant way.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

Farmer has the ability to collect all tokens from the cluster

CRITICAL | RESOLVED

In the smart contract ClusterToken the role FARMER_ROLE is granted to the deployer, which makes it possible to collect all tokens from the cluster. Also anytime the owner can grant Farmer role to another account and this account will be able to collect underlying tokens.

Recommendation:
Consider using a modifier for this purpose, initialize in the constructor, if it needs to be changeable add ownable functions with delay for that.

From client:
It is impossible to set Farmer contract in the constructor since it is still in development and there will probably be several Farmers (for different clusters or their modifications in further development). Thus specific timeframes or a single farmer contract are not applicable in such cases.

Also, as for the Farmer role: by design it is initially assigned to the contract ClusterLock (which is still in development). Also, the Admin over the contract will be moved to the multisig wallet (DAO in the future), thus the roles distribution will be decentralized.

Also, all underlyings movements are documented on-chain in public variables with proper reflection in the Farmer contract itself.

It's possible to transfer tokens even if the cooldown period is not finished

CRITICAL | RESOLVED

In the smart contract ClusterToken you override transferFrom by adding modifier checkCooldownPeriod but this modifier checks the cooldown period for msg.sender not for the actual token holder, it makes possible to transfer tokens even if the cooldown period is not finished.

Step to reproduce:

- User A deposits 1 ETH and gets 100 cluster tokens, cooldownTimestamps is written;
- User approve 100 tokens to address B;
- User B invokes transferFrom, modifier checkCooldownPeriod use cooldownTimestamps of address B for checks, since cooldownTimestamps for B equal 0 the transaction will pass and tokens will be transferred.

Recommendation:
Modify checkCooldownPeriod adding an address parameter for which cooldown period will be checked.

The contract owner has too many permissions

CRITICAL | RESOLVED

In the smart contract ClusterToken the owner can block transferring of tokens, which means that users lose access to deposited funds.

Recommendation:
Consider rejecting pausable functionality of ERC20.

Cooldown period not limited

HIGH | RESOLVED

In the smart contract ClusterToken the contract owner can update the cooldown period on cluster redeeming and transferring as much as he wants. The cooldown period should be limited by the max value so that users will know that it can't be changed to huge durations and blocked token transfers in this way.

Recommendation:
Add require which will limit the max cooldown period.

Each new deposit will lock all tokens of the user

HIGH | RESOLVED

Since you override transfer and transferFrom in the smart contract ClusterToken by adding checkCooldownPeriod it lock transfer all cluster tokens for the user not only of the last deposited amount.

Recommendation:
Consider adding functionality to lock a specific amount of tokens.

No check for duplicates in function addClusterToRegister

HIGH | RESOLVED

In the smart contract Controller owner can add duplicates that can cause incorrect behaviour.

Recommendation:
Consider adding a require for checking duplicates.

Redundant operation for burning tokens

LOW | RESOLVED

In the smart contract ClusterToken no need to transfer tokens from user to contract and then burn it, tokens can be burned directly from the user balance, it will save gas during the function call. Furthermore, in this case, users don't need to approve tokens anymore.

Recommendation:
Consider burning tokens directly from user balance.

The owner of the contract can improperly assign a roles

INFORMATIONAL | UNRESOLVED

The DeHive protocol access management is built based on specific roles, one of the roles is used to allocate tokens for the contract Farming. In case of misuse, the assignment of the role by the owner of the cluster token will be possible to carry out an attack that will allow you to extract a small number of funds from the cluster token.

There are specific cases when the attack is possible, in particular when the owner will be in cahoots with the attacker or owner will be the attacker:

- in the beginning when the locked token equals zero:

    1) Owner grant Farmer role to account A;
    2) Account A deposits 5 ETH to get 100 ClusterToken;
    3) Account A invoke withdrawToAccumulation(100) and got underlying tokens that equal share for 100 ClusterToken;
    4) Since 100 ClusterToken has not moved anywhere Account A will be able to withdraw deposited funds after invoking withdrawToAccumulation(100).

    So that Account A gets back the deposited 5 ETH and an equivalent amount of ETH but in underlying tokens;

- when the attacker has more cluster tokens than locked:

    1) The case when on the contract locked 100 cluster tokens;
    2) Owner grant Farmer role to account A;
    3) Account A deposits 10 ETH to get 200 ClusterToken;
    4) Account A invoke withdrawToAccumulation(100) with amount 100 because on contract already locked 100 and got underlying tokens that equal share for 100 ClusterToken;
    5) Since 100 ClusterToken has not moved anywhere Account A will be able to withdraw deposited funds after invoking withdrawToAccumulation(100).

    So that Account A gets back the deposited 10 ETH and an equivalent of the half deposited amount of ETH but in underlying tokens.

Recommendation:
Consider using a multisig for managing the DeHive protocol.

From client:
The Admin over the contract will be moved to the multi-sig wallet (DAO in the future), thus the distribution of the roles will be decentralized. Also, all underlying movements are documented on-chain in public variables with proper reflection in the Farmer contract itself.

IERC20Extend duplicate functionality of IERC20Metadata

INFORMATIONAL | RESOLVED

The interface IERC20Extend duplicate functionality of IERC20Metadata from OpenZeppelin library.

Recommendation:
Use existing interface from OpenZeppelin library.

Missed keyword indexed in events

INFORMATIONAL | RESOLVED

Address parameters in all events of ClusterToken are not indexed.

Recommendation:
Consider using indexed parameters in events. The indexed parameters for logged events will allow you to search for these events using the indexed parameters as filters.

Initializing of DUST_AMOUNT

INFORMATIONAL | RESOLVED

Since DUST_AMOUNT in smart contract ClusterToken is not a constant but named in uppercase and initializing after define it should be initialized in the constructor. It prevents misunderstanding during code reading.

Recommendation:
Initialize DUST_AMOUNT in the constructor and consider renaming this variable since it's not constant.

Useless payable identifier

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken no need for a payable identifier in function disassemble(), since this function is intended for withdrawing funds.

Recommendation:
Consider removing the payable identifier for disassemble().

Multiple overrides for check lock period

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken no need to override transfer and transferFrom for the check cooldown period, just override _beforeTokenTransfer by adding this requirement.

Recommendation:
Consider override _beforeTokenTransfer() instead of transfer and transferFrom.

The function disassembleByAdapter not follow Checks Effects Interactions pattern

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken function disassembleByAdapter not follow Checks Effects Interactions pattern.

Recommendation:
Consider firstly burning cluster tokens and then transfer underlying tokens.

Inconsistency code style

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken function disassembleByAdapter burns tokens but as an account uses msg.sender instead of use _msgSender() from contract Context.

Recommendation:
Consider using _msgSender().

The function _payInEth duplicate functionality of sendValue from library Address

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken the function _payInEth duplicate functionality of sendValue from the library Address, which you use in your contract.

Recommendation:
Consider using sendValue from library Address.

Redundant modifiers

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken are redundant modifiers onlyAdmin, onlyFarmer, onlyDelegate, onlyAdapter. The contract AccesssControl has modifier onlyRole which checks permissions for the specific role.

Recommendation:
Consider using onlyRole from the contract AccesssControl.

Sending ETH not through the relevant function

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken function _swapEthToUnderlyings send ether in a common way instead of using function _payInEth or sendValue of library Address.

Recommendation:
Consider using library Address for sending ether.

Unsuitable parameter return in docs

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken wrongly described return in docs for function getUnderlyingBalance.

Recommendation:
Update docs for function getUnderlyingBalance.

Order of Layout

INFORMATIONAL | RESOLVED

Layout contract elements in Controller and StakingDHV contracts are not logically grouped.

The contract elements should be grouped and ordered in the following way:

- Pragma statements;
- Import statements;
- Interfaces;
- Libraries;
- Contract.

Inside each contract, library or interface, use the following order:

- Library declarations (using statements);
- Constant variables;
- Type declarations;
- State variables;
- Events;
- Modifiers;
- Functions.

Ordering helps readers to navigate the code and find the elements more quickly.

Recommendation:
Consider changing order of layout according to solidity documentations: Order of Layout.

Re-audit:
From client: Since there is no impact on the functionality and since the code is already deployed (so, changing storage layout will have an impact) – no need to fix.

Order of Functions

INFORMATIONAL | RESOLVED

The functions in contract ClusterToken are not grouped according to their visibility and order.

Functions should be grouped according to their visibility and ordered in the following way:

- Constructor;
- Receive function (if exists);
- Fallback function (if exists);
- External;
- Public;
- Internal;
- Private.

Ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier.

Recommendation:
Consider changing functions order according to solidity documentations: Order of Functions.

Re-audit:
From client: All functions are grouped into meaningful groups with appropriate comments. Thus, changing the order of layout by the recommendation will DECREASE the readability and code quality. Thus we insist that this finding is not relevant to the code and will have a more harmful impact.

Redundant modifiers

INFORMATIONAL | RESOLVED

In the smart contract StakingDHV are redundant modifiers onlyAdmin, onlyClusterLock. The contract AccesssControl has modifier onlyRole which checks permissions for the specific role.

Recommendation:
Consider using onlyRole from the contract AccesssControl.

Duplicates functionality of ownable

INFORMATIONAL | RESOLVED

The smart contract StakingDHV inherits AccessControlUpgradeable and StakingPools which inherit OwnableUpgradeable.

Recommendation:
Consider using AccessControlUpgradeable in StakingPools.

Function updatePoolSettings not using modifier hasPool for check pool existence

INFORMATIONAL | RESOLVED

In the smart contract StakingPools function updatePoolSettings use require to check pool existence instead of using modifier hasPool.

Recommendation:
Consider using hasPool.

Redundant converting

INFORMATIONAL | RESOLVED

In the smart contract StakingPools functions deposit and withdraw has the redundant conversion of address to address (lines 181, 203).

Recommendation:
Consider removing redundant convertingthe.

Redundant converting

INFORMATIONAL | RESOLVED

In the smart contract StakingDHV modifier hasDHV has the redundant conversion of address to address (lines 31).

Recommendation:
Consider removing redundant convertingthe.

Redundant modifiers

INFORMATIONAL | RESOLVED

In the smart contract ClusterToken the function disassemble() two times check checkCooldownPeriod, first when call function second when burn tokens. So modifier checkCooldownPeriod can be deleted from function disassemble() and since checkCooldownPeriod will be used only in _beforeTokenTransfer() all conditions can be moved to the _beforeTokenTransfer() and modifier checkCooldownPeriod can be deleted.

Recommendation:
Consider deleting modifier checkCooldownPeriod from disassemble().

	ClusterToken	Controller	StakingDHV
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	DexAdapterCore	UniswapAdapter	QickswapAdapter
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting DeHive in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the DeHive contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
contracts/	100.00	94.29	100.00	100.00
ClusterToken.sol	100.00	96.00	100.00	100.00
Controller.sol	100.00	90.00	100.00	100.00
contracts/dex-adapter/	100.00	96.15	100.00	100.00
DexAdapterCore.sol	100.00	96.15	100.00	100.00
contracts/dex-adapter/mainnet/	100.00	100.00	100.00	100.00
UniswapAdapter.sol	100.00	100.00	100.00	100.00
contracts/dex-adapter/polygon/	80.00	75.00	100.00	80.00
QuickSwapAdapter.sol	80.00	75.00	100.00	80.00
contracts/staking/	100.00	100.00	100.00	100.00
StakingDHV.sol	100.00	100.00	100.00	100.00
All files	**99.66**	**94.74**	**100.00**	**99.51**

Test Results

Contract: StakingDHV

StakingDHV Test Cases

- ✓ should initialize with correct admin (154ms)
- ✓ shouldn't cover locking amount of DHV if DHV token not setted (626ms)
- ✓ should set DHV token correctly (348ms)
- ✓ should set DHV token by not the current admin (492ms)
- ✓ shouldn't cover locking amount of DHV with wrong pool id (220ms)
- ✓ shouldn't cover locking amount of DHV if cluster rate not setted (322ms)
- ✓ should set rate for cluster token correctly (273ms)
- ✓ should cover locking amount of DHV for staking cluster token correctly (754ms)
- ✓ shouldn't cover locking amount of DHV for staking cluster token with to small amount (226ms)
- ✓ shouldn't cover locking amount of DHV for staking cluster token with insufficient balance of DHV (489ms)
- ✓ should unlock dhv tokens partially (394ms)
- ✓ should unlock dhv tokens partially with small cluster rate (849ms)
- ✓ shouldn't unlock dhv tokens partially with amount equal 0 (198ms)
- ✓ should unlock total dhv tokens for the user correctly (706ms)

Contract: QuickSwapAdapter

QuickSwapAdapter Test Cases

- ✓ should deploy with correct address router (146ms)
- ✓ should deploy with correct address of WMATIC (116ms)

QuickSwapAdapter Controller Test Cases

- ✓ should calculate amount of eth, necessary to buy cluster correctly (9925ms)
- ✓ should calculate amount of cluster user will receive from depositing certain amount of eth correctly (546ms)

Contract: Controller

Controller Initialize Test Cases

- ✓ should deploy with correct shares decimals (550ms)
- ✓ should deploy with correct cluster token decimals (89ms)
- ✓ shouldn't initialize with zero address (1010ms)
- ✓ shouldn't initialize twice (369ms)
- ✓ should initialize with correct owner (96ms)

Controller Admin Interface Test Cases

- ✓ should set deposit commission for cluster assemble correctly (286ms)
- ✓ shouldn't set deposit commission more than shares decimals (252ms)
- ✓ should set redeem commission for cluster disassemble correctly (467ms)
- ✓ shouldn't set redeem commission more than shares decimals (164ms)
- ✓ should set factory correctly (490ms)
- ✓ shouldn't set factory to the zero address (224ms)
- ✓ should set a swap router for cluster correctly (291ms)
- ✓ shouldn't set a swap router to the zero address (237ms)
- ✓ should add new cluster address to the list of all clusters correctly (651ms)
- ✓ shouldn't add new cluster to register if caller is not a factory (251ms)
- ✓ shouldn't add cluster with the zero address to register (234ms)
- ✓ shouldn't add one cluster to register twice (610ms)
- ✓ should change controller for cluster correctly (1363ms)

Controller View Interface Test Cases

- ✓ should return correct list of indices currently registered (547ms)
- ✓ should calculate amount of cluster user will receive from depositing ether correctly (299ms)
- ✓ should calculate amounts of underlyings from cluster correctly (678ms)
- ✓ should calculate amount of ether from underlyings correctly (468ms)
- ✓ should calculate amount of ether from underlyings correctly (373ms)
- ✓ should return correct dhv price (190ms)
- ✓ loreshould return correct cluster pricemipsumdolor (242ms)
- ✓ should return correct commission for the deposit for the chosen cluster (147ms)
- ✓ should return correct commission for the cluster redemption (125ms)

Contract: ClusterToken

Cluster Token Deploy Test Cases

- ✓ should deploy with correct name (178ms)
- ✓ should deploy with correct symbol (95ms)
- ✓ should deploy with correct cluster HashId (46ms)
- ✓ should deploy with correct cluster controller (407ms)
- ✓ should deploy with correct dhv token address (70ms)
- ✓ should deploy with correct admin (176ms)
- ✓ shouldn't deploy with controller zero address (1665ms)
- ✓ shouldn't deploy with treasury zero address (2315ms)
- ✓ shouldn't deploy with wrong arrays length (2536ms)
- ✓ shouldn't deploy with underlying zero address (2586ms)
- ✓ shouldn't deploy with underlying amount equal 0 (2718ms)

Cluster Token Admin Interface Test Cases

- ✓ should set cooldown period on cluster correctly (2498ms)
- ✓ shouldn't set cooldown period by not the admin (476ms)
- ✓ shouldn't set cooldown period more than two days (449ms)
- ✓ should set new treasury address correctly (343ms)
- ✓ shouldn't set new treasury address to the zero address (230ms)
- ✓ should pause correctly (229ms)
- ✓ should unpause correctly (271ms)
- ✓ should withdraw all ETH kept as dust correctly (589ms)
- ✓ should change controller correctly (581ms)
- ✓ shouldn't change controller by not the current controller (275ms)
- ✓ shouldn't change controller by not the current controller (239ms)

Cluster Token User's Interface Test Cases

- ✓ should deposit ETH and purchase cluster without cooldown period correctly (2907ms)
- ✓ should deposit ETH and purchase cluster with cooldown period correctly (2659ms)
- ✓ should deposit ETH and purchase cluster with commision in ETH correctly (2020ms)
- ✓ should deposit ETH and purchase cluster with commision in DHV correctly (2169ms)
- ✓ shouldn't buy cluster if not enough ether was sent (481ms)
- ✓ shouldn't buy cluster if caller is a contract (307ms)
- ✓ should exchange cluster to ETH correctly (2301ms)
- ✓ shouldn't exchange cluster to ETH with insufficient cluster balance (190ms)
- ✓ shouldn't exchange cluster to ETH if cooldown in progress (213ms)

Cluster Token Farmer And Delegate Interface Test Cases

- ✓ should assemble by adapter correctly (1197ms)
- ✓ should disassemble by adapter correctly (1080ms)
- ✓ should withdraw tokens of cluster for staking correctly (1964ms)
- ✓ should refund tokens of cluster from staking correctly (1979ms)
- ✓ should return debt from accumulation correctly (1424ms)
- ✓ should make correction of the underlyings proportion correctly (2198ms)
- ✓ shouldn't make correction with wrong shares array length (143ms)

Cluster Token View Interface Test Cases

- ✓ should return correct list of underlyings addresses for the cluster (97ms)
- ✓ should return correct list of amount of each underlying for 1 cluster (114ms)
- ✓ should calculate amounts of underlyings over the amount of cluster correctly (166ms)
- ✓ should return correct underlying balance (274ms)

Contract: UniswapAdapter

UniswapAdapter Deploy Test Cases

- ✓ should deploy with correct address of UniswapV2Router (81ms)

```
    ✓ should deploy with correct address of Wrapped ETH token (99ms)
    ✓ should deploy with correct address of USDT Token (70ms)
    ✓ shouldn't deploy with zero address (1251ms)
  Dex Adapter Core Test Cases
    ✓ should swap ETH to weth and send it to the sender (623ms)
    ✓ should swap ETH to underlying token and send it to the sender (3308ms)
    ✓ should swap underlying to ETH and send it to the sender correctly (2904ms)
    ✓ should swap one token to another one correctly (5664ms)
    ✓ should return amount of tokens, which can be bought for another token's
      amount correctly (99ms)
    ✓ should return prices for underlyings in ETH correctly (113ms)
    ✓ should return ETH price correctly (313ms)
    ✓ should return DHV price correctly (647ms)
    ✓ should return Eth amount with slippage correctly (152ms)
    ✓ should set path correctly (1430ms)
  UniswapAdapter Controller Test Cases
    ✓ should calculate amount of eth, necessary to buy cluster correctly (4895ms)
    ✓ should calculate amount of cluster user will receive from depositing certain amount
      of eth correctly (1189ms)

  103 passing (5m)
```

We are grateful to have been given the opportunity to work with the DeHive team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the DeHive team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.