HORD

SMART CONTRACT AUDIT

zokyo

December 27th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passed a security audit.

ZOKYO AUDIT SCORING DEFACTOR

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 1 Medium issue: 1 resolved = 0 points deducted
- 3 Low issues: 3 resolved = 0 points deducted
- 5 Informational issues: 4 resolved, 1 verified = -2 points deducted due to missing settings for IDubUniswap contract deployment (which makes it a 3rd party dependency) and for existing concerns for possible token de-peg.
- 1 point was deducted for the absence of the native tests for the functionality

Thus, 100 - 2 - 1 = 97

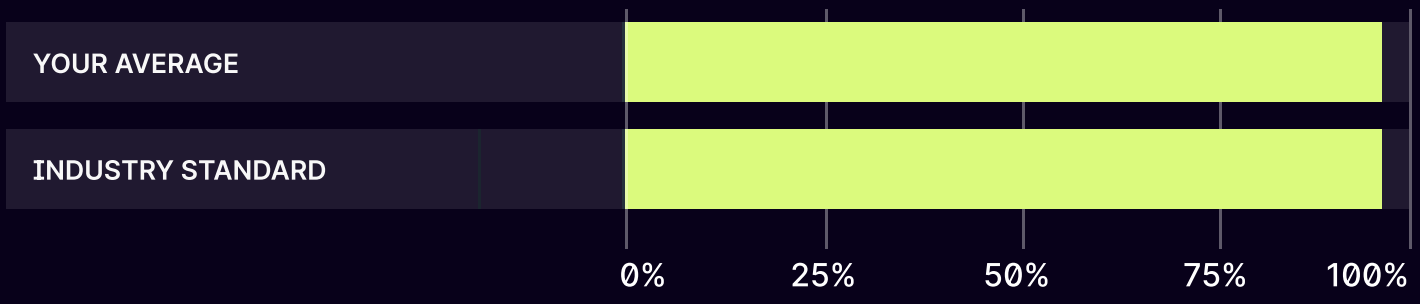
TECHNICAL SUMMARY

This document outlines the overall security of the Hord smart contracts evaluated by the Zokyo Security team.

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

Contract Status

Testable Code



Corresponds to industry standards.

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

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Hord repository:
https://github.com/hord/smart-contracts

Branch: staging
Initial commit: 0d6967dbcf25d59375098fe59b47b8571cb96223
Final commit: cee0dd705f7b04d1eb4700ebefdea311a335e11c

Check for changes, re-audit after 1 iteration of audit, use first version code to check contract upgrade during testing:

Previous/First version of code commit: efb71eb817291972822e05a66251a795c3c43663

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- HordETHStakingManager.sol (291 lines changed)
- DcentraLabTokensUtil.sol*

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

*Contract was needed to correct work of main contract. DcentraLabTokensUtil was taken from https://github.com/DcentraLab/dub-contracts/blob/master/contracts/DcentraLabTokensUtil.sol on commit 87141e500734978a830cfac993bde5563f6a88c3

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Hord smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

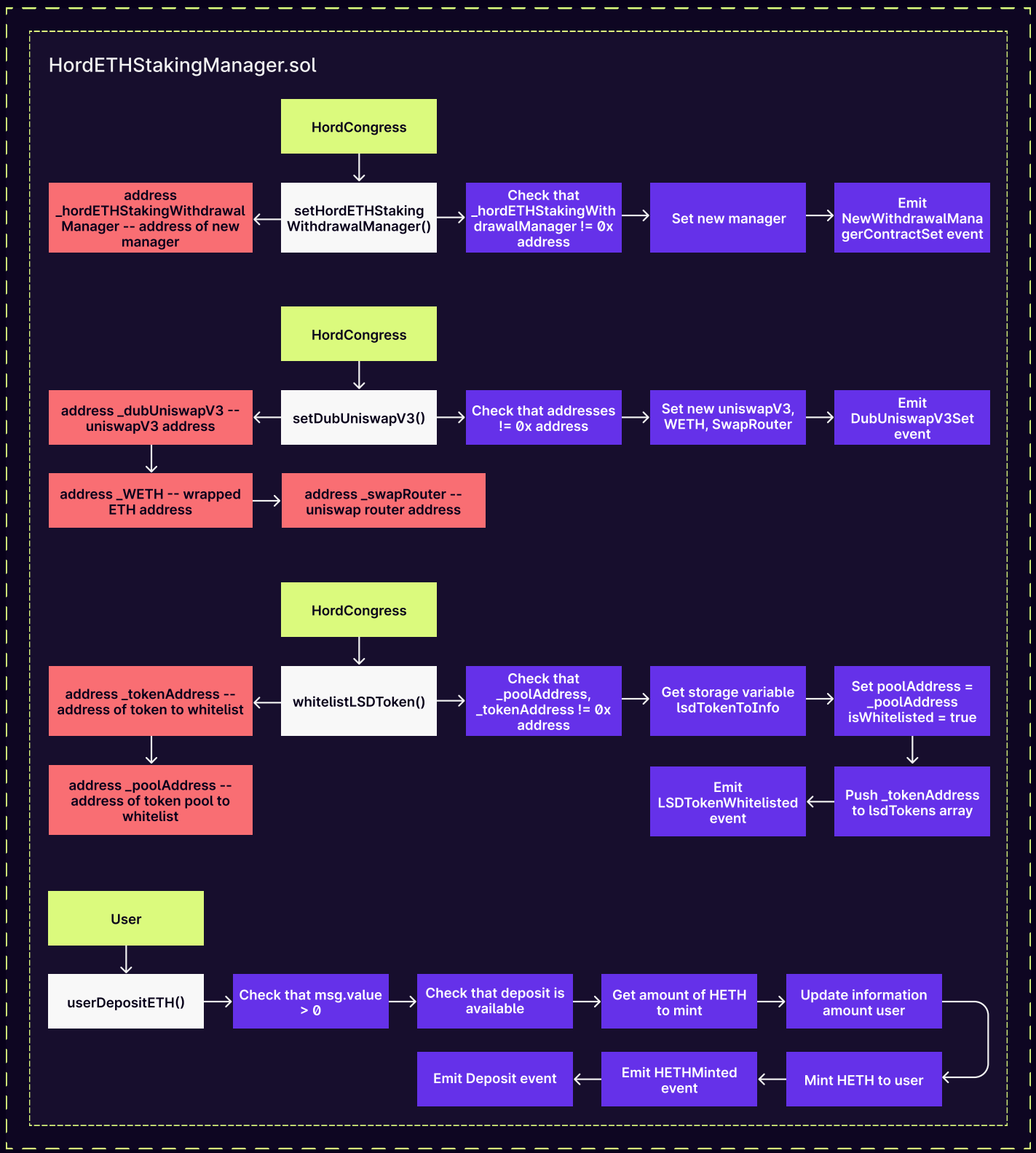
Zokyo Security received the Hord project contract for a security audit. The protocol represents a staking and withdrawal system for Hord Ethereum (HETH) tokens, including fee calculations and debt handling implementation. In this iteration of the audit, the team worked with the staking part of the protocol.

The audit's goal was to verify the safety of user deposits, confirm the correctness of the calculation during the SD tokens deposit, and check the correctness of the LSD tokens swap via the UniswapV3. Our auditors also checked the contract's code against the internal checklist of vulnerabilities, validated the business logic of the contract, and ensured that best practices in terms of gas spending were applied. The report contains one medium issue connected to the unchecked transfer. Other issues refer to adding the same LSD token to the allowlist, user deposits track, usage of safeApprove, absent functionality for removing LSD token from the allowlist, missing contract, misleading function name, and potential loss of LSD token price attachment. All issues were successfully fixed or verified. However, auditors still need to notice that the missing contract issue (Info-3) was transformed into an out-of-scope dependency: the contract was added into the scope and verified by the team, though it belongs to a separate package and is not present in the current protocol settings.

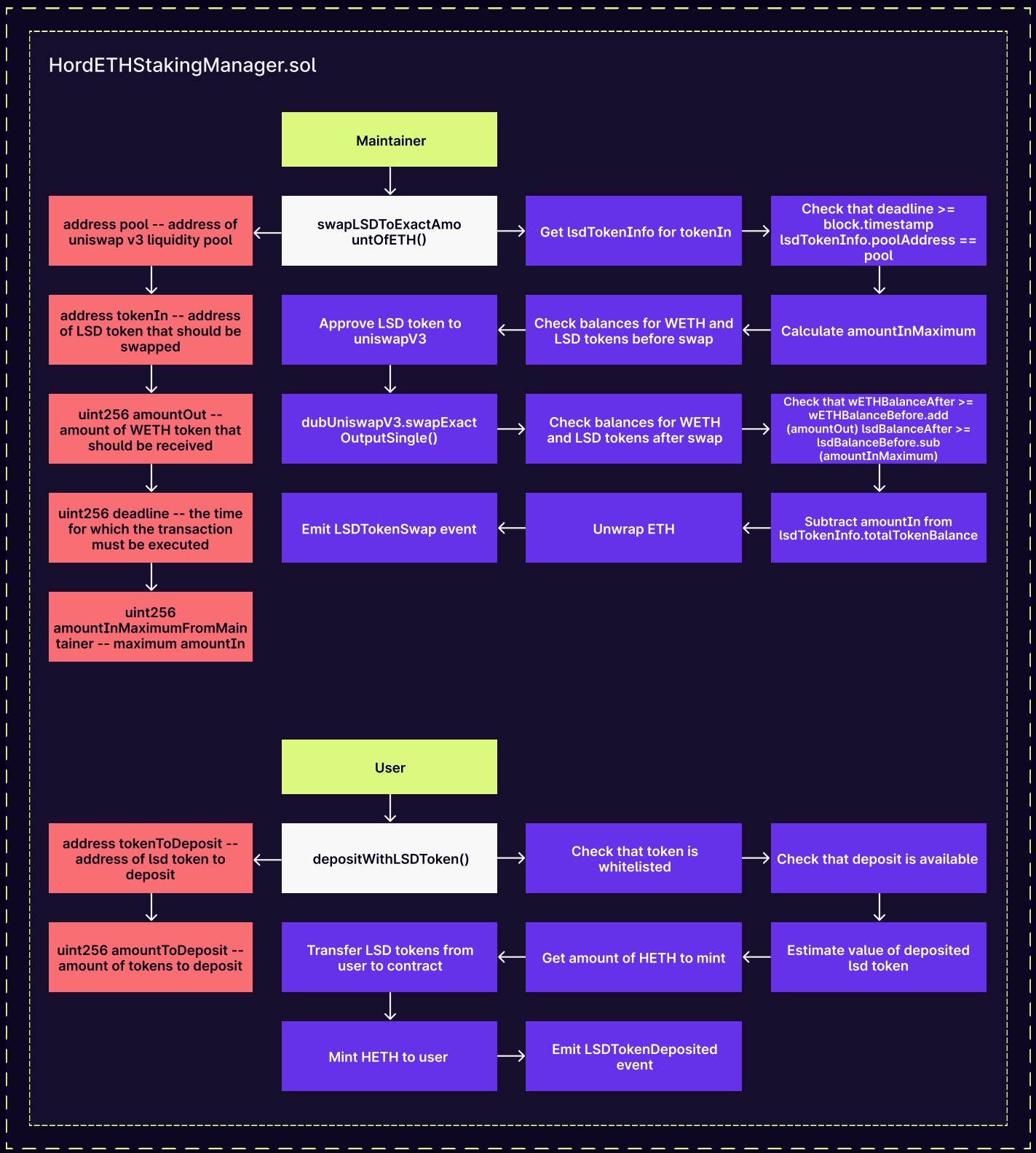
Another part of the audit process was to prepare tests to check contract functionality as well as check main hypotheses: potential issues with safeApprove usage, risk of front-run attack because of price manipulation, and WETH implementation potential transaction revert. The risk of front-run was moved to the informational issue, as it is connected to the usage of LSD tokens only, and the Hord team accepted the potential risk of the token price de-peg. Potential issues with the safeApprove moved were confirmed (as it could break subsequent implementations), and the Hord team fixed the issue. WETH issue was marked as false-positive and removed from the report. The complete set of unit tests can be seen in the Code Coverage and Test Result sections.

In conclusion, the Hord contracts have high security level and function correctly with a known level of security. The contracts work as expected, and the auditors prepared tests to cover all the core logic, ensuring the contract's functionality and security. Nevertheless, the project fails the check against the backdoors, as the contracts set is upgradeable. Despite it is a common approach, it still creates a controllable backdoor, which should be noted for the funds bearing contract. Also, auditors should note, that the protocol does not contain own tests for the added functionality - all tests within the audit were prepared by the auditors.

Hord scheme


HordETHStakingManager.sol
HordCongress
setHordETHStakingWithdrawalManager()
address _hordETHStakingWithdrawalManager -- address of new manager
Check that _hordETHStakingWithdrawalManager != 0x address
Set new manager
Emit NewWithdrawalManagerContractSet event
HordCongress
setDubUniswapV3()
address _dubUniswapV3 -- uniswapV3 address
Check that addresses != 0x address
Set new uniswapV3, WETH, SwapRouter
Emit DubUniswapV3Set event
address _WETH -- wrapped ETH address
address _swapRouter -- uniswap router address
HordCongress
whitelistLSDToken()
address _tokenAddress -- address of token to whitelist
Check that _poolAddress, _tokenAddress != 0x address
Get storage variable lsdTokenToInfo
Set poolAddress = _poolAddress isWhitelisted = true
address _poolAddress -- address of token pool to whitelist
Emit LSDTokenWhitelisted event
Push _tokenAddress to lsdTokens array
User
userDepositETH()
Check that msg.value > 0
Check that deposit is available
Get amount of HETH to mint
Update information amount user
Emit Deposit event
Emit HETHMinted event
Mint HETH to user

Hord scheme


HordETHStakingManager.sol
Maintainer
swapLSDToExactAmo untOfETH()
address pool -- address of uniswap v3 liquidity pool
Get lsdTokenInfo for tokenIn
Check that deadline >= block.timestamp lsdTokenInfo.poolAddress == pool
address tokenIn -- address of LSD token that should be swapped
Approve LSD token to uniswapV3
Check balances for WETH and LSD tokens before swap
Calculate amountInMaximum
uint256 amountOut -- amount of WETH token that should be received
dubUniswapV3.swapExact OutputSingle()
Check balances for WETH and LSD tokens after swap
Check that wETHBalanceAfter >= wETHBalanceBefore.add (amountOut) lsdBalanceAfter >= lsdBalanceBefore.sub (amountInMaximum)
uint256 deadline -- the time for which the transaction must be executed
Emit LSDTokenSwap event
Unwrap ETH
Subtract amountIn from lsdTokenInfo.totalTokenBalance
uint256 amountInMaximumFromMain tainer -- maximum amountIn
User
address tokenToDeposit -- address of lsd token to deposit
depositWithLSDToken()
Check that token is whitelisted
Check that deposit is available
uint256 amountToDeposit -- amount of tokens to deposit
Transfer LSD tokens from user to contract
Get amount of HETH to mint
Estimate value of deposited lsd token
Mint HETH to user
Emit LSDTokenDeposited event

DESCRIPTION

The `HordETHStakingManager` contract is a part of the Hord project, a decentralized platform for social trading. This contract manages the staking of HordETH (HETH) by users. Users can deposit ETH or LSD tokens in exchange for HETH tokens.
The contract also interacts with other contracts, such as `HordETHStakingWithdrawalManager` to pay debts and move ETH to assured balance and with `UniswapV3` to swap LSD tokens to ETH. It also emits several events to track the state of withdrawals and deposits.

Main functionality:
1. User's deposit: A user initiates a deposit by calling the `userDepositETH` function. The user sets the msg.value, which represents the amount of ETH to deposit. The function checks that there is no launching/closing validator and that the deposit is available. The user gets HETH in return as a representation of the deposit to protocol. Users can also deposit to protocol with allowed LSD tokens using the `depositWithLSDToken` function. The function will check the estimated price of a given LSD token using uniswapV3. Users will exchange LSD tokens for HETH tokens in return.

2. Launch validator: Protocol can register new validators using `launchNewValidator`. After the protocol has 32 ETH deposited, it will add a new validator to the Ethereum chain.

3. Swap LSD: As one of the abilities to deposit is with LSD tokens, the function `swapExactAmountOfLSDToken` swaps the given LSD token to ETH so that the protocol will be able to register a new validator. Swaps are performed with uniswapV3.

ROLES AND RESPONSIBILITIES

1. Deployer

The deployer is responsible for the deployment of HordETHStakingManager.sol. The deployer must pass valid initial parameters such as the Hord Congress contract address, maintainers registry contract address, staking configuration contract address, beacon contract address, and Hord Congress member registry contract address.

2. HordCongress

The HordCongress is responsible for the unpause contract, setting the HordETHStakingWithdrawalManager contract, setting the DubUniswapV3 contract, allowlisting LSD tokens, and removing them.

3. Maintainer

The Maintainer is responsible for appropriate reserves to pay the debt, set validator stats, launch a new validator, and swap LSD to ETH.

4. HordETHStakingWithdrawalManager

The HordETHStakingWithdrawalManager contract is responsible for burning tokens from users when the withdrawal is requested and moving ETH to an assured balance.

DEPLOYMENT

The deployment scripts are located at scripts/deployment/deploy_07_ethStaking.js

The script first deploys `StakingConfiguration` and `HordETHStakingManager` contracts. The second script deploys `HordETHStakingWithdrawalManager` contract as a proxy, using OpenZeppelin's Upgrades plugin. This is a good practice as it allows you to upgrade the contract in the future without losing the state.

In addition, you should ensure that the `HordCongress,` `MaintainersRegistry,` and `BeaconDeposit` contract addresses are set correctly.

Make sure to test deployment scripts thoroughly on a test network before deploying to the mainnet. Note, that scripts **do not contain** the setting of the dub contract (see **Info-3**)

SETTINGS

1. **HordCongress Contract Address**: This is the address of the `HordCongress` contract, which was set during the initialization of the `HordETHStakingManager` contract. This contract provides important control of contracts like `unpause,` `setHordETHStakingWithdrawalManager,` and `setDubUniswapV3`. The swaps may not function as expected if this is not set correctly.

2. **MaintainersRegistry Contract Address**: This is the address of the `MaintainersRegistry` contract, which was set during the initialization of the `HordETHStakingManager` contract. This contract is used to check if a maintainer initiates a function call. If this is not set correctly, unauthorized addresses may be able to call functions restricted to maintainers.

3. **StakingConfiguration Contract Address**: This is the address of the `StakingConfiguration` contract, which was set during the initialization of the `HordETHStakingManager` contract. This contract provides important parameters like `percentPrecision,` `amountETHInValidator,` `rewardFeePercentage,` `stakeETHTokenSymbol,` `tolerancePercentageForFee,` `priceSlippageThreshold,` and `feeRecipient.` If this is not set correctly, the adding validators process may not function as expected.

4. **BeaconDeposit Contract Address**: This is the address of the `Beacon Deposit` contract, which was set during the initialization of the `HordETHStakingManager` contract. If this is not set correctly, the protocol cannot add validators to the Ethereum network.

5. **HordCongressMembersRegistry Contract Address**: This is the address of the `HordCongressMembersRegistry` contract, which was set during the initialization of the `HordETHStakingManager` contract. If this is not set correctly, unauthorized addresses may be able to call the pause function.

6. **HordETHStakingWithdrawalManager Contract Address**: This is the address of the `HordETHStakingWithdrawalManager` contract, which is set by HordCongress using the `setHordETHStakingWithdrawalManager` function. If this is not set correctly, users cannot withdraw staked ETH.

7. **UniswapV3 Contract Address**: This is the address of `UniswapV3` contract which can be set by HordCongress using `setDubUniswapV3` function. If this is not set correctly, users will not be able to swap LSD tokens for ETH.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. 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 contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

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.

MEDIUM-1 | RESOLVED

Transfer is not validated.

HordETHStakingManager.sol: depositWithLSDToken(), line 566.
Transferring is performed with a regular transferFrom() method from the OpenZeppeling IERC20 interface without validation if a transfer is successful. To ensure the security of the transfer, it is recommended to validate the success of the transfer call. As IERC20 already uses SafeERC20, replacing transferFrom() with safeTransferFrom() is necessary.

Recommendation:
Replace transferFrom() with safeTransferFrom() function.

Post audit.
safeTransferFrom() is now used.

LOW-1 | RESOLVED

Ability to add the same LSD token to the whitelist.

HordETHStakingManager.sol: whitelistLSDToken().
Because in the allowlist, there is no check on the address of whether the LSD token was added, it makes it functional to add the same token again. Consider checking whenever a token is added to the list. As a result, the same token might repeat in the array `lsdTokens,` and it is possible to update the pool address for the token. The issue is marked as low since only the manager can execute the function.

Recommendation:
Check if LSD tokens are already in the allowlist.

Post audit.
Added check if token is already in allowlist.

LOW-2 | RESOLVED

User information is not tracked when depositing with LSD token.

HordETHStakingManager.sol: userDepositETH() updates the total number of participants in the case of the first deposit. It also updates the user's total data about how much was deposited from him. The function HordETHStakingManager.sol: depositWithLSDToken() does not update both the number of participants and the amount of LSD tokens deposited from him.

Recommendation:
Confirm whether it is necessary to track the user's information about his deposits via LSD token or even tokens.

Post audit.
The Hord team verified that according to the technical specification, they do not need any information regarding the user who deposited the lsd token. Deposit tracking was removed from the contract.

LOW-3 | RESOLVED

SafeApprove deprecated

HordETHStakingManager.sol: swapExactAmountOfLSDToken(), line 464.
Because safeApprove checks for already approved amounts, the function will revert later if not all allowance is spent during the previous swap, making swap functionality unusable. This has no effect in the current implementation, but in later versions, it could be an issue. Check here: https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#SafeERC20-safeApprove-contract-ERC20-address-uint256-

Recommendation:
Use safeIncreaseAllowance in favor of safeApprove.

Post audit.
safeApprove changed to safeIncreaseAllowance()

INFORMATIONAL-1 | RESOLVED

Absent functionality to remove LSD token from whitelist.

HordETHStakingManager.sol: whitelistLSDToken().
Once an LSD token is allowed, it cannot be removed from the allowlist. This will prevent unwanted LSD tokens from being canceled in the future. Consider adding functionality to remove unwanted tokens from the allowlist.

Recommendation:
Make functionality to remove LSD tokens from the allowlist.

Post audit.
Added ability to remove LSD tokens.

INFORMATIONAL-2 | RESOLVED

The function modifier and natspec do not match.

HordETHStakingManager.sol: whitelistLSDToken() has modifier onlyHordCongress while in natspec mentioned that this function is called only by the maintainer.

Recommendation:
Check which role should be used in the function and make sure that the natspec is appropriate.

Post audit.
Fixed modifier name in natspec.

INFORMATIONAL-3 | RESOLVED

Unknown IDubUniswapV3 3rd party dependency (which is out of scope) + missed setting.

HordETHStakingManager.sol
Project does not have an address or mock for this contract. It is unknown how this contract should behave. Verify the usage of this contract. Also, deployment scripts contain no setting for this contract.

Recommendation:
Provide information about the IDubUniswapV3 contract.

Post audit.
Hord team shared the requested contract and it was added to the scope. Auditors provided a successful testing of the protocol with the added contract. Those, auditors need to note, that the contract is located in the separate package and is not added into the deployment/ settings flow (and can be changed by the admin). Despite being added into the scope, it is still treated as a 3rd-party dependency.

INFORMATIONAL-4 | RESOLVED

Incorrect function name for swap

HordETHStakingManager.sol: swapExactAmountOfLSDToken()
The function is named swapExactAmountOfLSDToken, corresponding to `swapExactInput.` However, the protocol uses `swapExactOutput.` Thus, the confusion makes the calculation of the amountInMaximum seem incorrect. Swap with the expected by the naming parameters produces the 'STF' error (which corresponds to the safeTransferFrom error). The change of the safeApprove amount produces the error 'Too much requested' happens, which explains that amountIn does not correspond to amountOut. Thus, it effectively blocks the operations unless the tweak is applied to the parameters. We recommend changing the usage of `swapExactOutput` to `swapExactInput` as it removes complexity, and it is more understandable to swap 10 LSD tokens than get 10 ETH when swapping. An example of the function is given in the screenshot.

```
function swapExactAmountOfLSDToken(
    address pool↑,
    address tokenIn↑,
    uint256 amountIn↑,
    uint256 deadline↑,
    uint256 amountOutMinimumByMaintainer↑
) external nonReentrant whenNotPaused onlyMaintainer returns (uint256) {
    LSDTokenInfo storage lsdTokenInfo = lsdTokenToInfo[tokenIn↑];

    require(deadline↑ >= block.timestamp, "wrong deadline");
    require(lsdTokenInfo.poolAddress == pool↑, "wrong liquidity pool address");

    uint256 wETHBalanceBefore = IWrappedToken(WETH).balanceOf(address(this));
    uint256 lsdBalanceBefore = IERC20(tokenIn↑).balanceOf(address(this));

    require(amountIn↑ <= lsdBalanceBefore, "dont have enough lsd tokens");
    IERC20(tokenIn↑).safeApprove(address(dubUniswapV3), amountIn↑);

    uint256 amountOut = dubUniswapV3.swapExactInputSingle(
        swapRouter,
        pool↑,
        tokenIn↑,
        amountIn↑,
        deadline↑,
        amountOutMinimumByMaintainer↑
    );

    uint256 wETHBalanceAfter = IWrappedToken(WETH).balanceOf(address(this));
    uint256 lsdBalanceAfter = IERC20(tokenIn↑).balanceOf(address(this));

    require(wETHBalanceAfter >= wETHBalanceBefore.add(amountOut), "didnt receive total amountOut");
    require(lsdBalanceAfter >= lsdBalanceBefore.sub(amountIn↑), "spent more than amountInMaximum");

    lsdTokenInfo.totalTokenBalance = lsdBalanceBefore.sub(amountIn↑);

    // Uniswap V3 contracts work only with WETH, so at the end of swap we will get WETH and in the next line we have
    // to swap that back into  native ETH in order to work with that (launch validators/pay out debts)
    IWrappedToken(WETH).withdraw(amountOut);

    emit LSDTokenSwap(tokenIn↑, amountIn↑, amountOut);

    return amountIn↑;
}
```

Recommendation:
Change function logic.

Post audit.
The function name changed to `swapLSDToExactAmountOfETH().` Hord team added the need to function to return the exact amount of ETH; that's the spec as they swap intending to obtain a pre-ascertained amount of ETH required as the remainder for launching the validator or repaying debt to users. Still, the auditors raise the concern, that a lot depends from the correct calculation of the parameters by the outer services which trigger the function.

LSD token could potentially lose price attachment.

Because deposited LSD tokens are not swapped immediately, it is possible that the token could lose attachment to ETH price and drop, which may affect when swapping LSD to ETH. Consider swapping LSD tokens when deposited to prevent additional risks. This issue is info because it was tested on frontrun but does not affect price. However, the issue is still listed because of the risk of sudden de-peg of the asset.

Potential threads: Risk of Depeg LSD token

Recommendation:
Swap LSD tokens when deposited.

Post audit.
Hord team verified that they want minimize gas costs by holding selected LSD tokens, such as stETH and wstETH, within the Hord contract for extended periods to benefit from their value appreciation or APR, selling only when necessary for operational needs like deploying validators or covering debts in ETH. Thus the Hord team confirms that the risk is speced into the product.

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

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

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

Scope: HordETHStakingManager.sol

HordETHStakingManager

after initialization

✓ Should revert if call initialization (3809ms)

during initialization

✓ Should revert if stacking cfg addr is zero addr (216ms)

✓ Should revert if beacon addr is zero addr (59ms)

 # Set withdrawal manager

✓ Should allow to set withdrawal manager (56ms)

✓ Should revert if caller is not the hord congress

✓ Should revert if setting addr is zero addr (48ms)

 # Pause control

✓ Should allow to pause by maintainer

✓ Should allow to pause by hord congress member

✓ Should revert if caller is not the maintainer or hord congress member

✓ Should allow to unpause by hord congress (75ms)

✓ Should revert if caller is not hord congress member

 # Deposit with ETH

✓ Should allow user to deposit ETH (660ms)

✓ Should allow to deposit multiple times for 1 user (99ms)

✓ Should revert if paused

✓ Should revert if sended value is 0

✓ Should revert if stakes are 32 ETH or more

✓ Should revert if call is not direct

 # Set LSD Token

✓ Should allow to set LSD token (52ms)

✓ Should revert if token address is zero address (42ms)

✓ Should revert if pool address is zero address (40ms)

✓ Should revert if caller is not hord congress

 # Remove LSD token

✓ Should allow to remove LSD token (50ms)

✓ Should revert if caller is not hord congress
✓ Should revert if LSD token is zero address (44ms)
✓ Should revert if LSD token is not whitelisted (56ms)
 # DubUniswapV3
✓ Should allow to set dub uniswap v3 (60ms)
✓ Should revert if caller is not hord congress
✓ Should revert if dub uniswap v3 address is zero address (47ms)
✓ Should revert if WETH address is zero address (43ms)
✓ Should revert if swap router address is zero address (48ms)
 # Deposit with LSDToken
✓ Should allow to deposit with lsd token (901ms)
✓ Should revert if LSD token is not whitelisted
✓ Should revert if paused
✓ Should revert if balance above 32 ETH
✓ Should revert if call is not direct
 # Swap LSD to ETH
✓ Should allow to swap (1889ms)
✓ Should allow to swap with amount maximum from maintainer (148ms)
✓ Should revert if amountInMaximumFromMaintainer is wrong (82ms)
✓ Should revert if caller is not maintainer
✓ Should revert if deadline was reached
✓ Should revert if pool address is not matching with token info
✓ Should revert when paused (66ms)
✓ Should revert if LSD tokens are not enough (77ms)
 # Launch a new validator
✓ Should allow to launch new validator
✓ Should revert if validator is already initialized (39ms)
✓ Should revert if conditions are not met
✓ Should revert if contract is paused
✓ Should revert if caller is not maintainer
 # Burn
✓ Should revert when paused
✓ Should revert if caller is not hord withdrawal manager
 # Amount of ETH for HETH
✓ Should return equal amount when total supply is zero

51 passing (10s)

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

FILE	% STMTS	% BRANCH	% FUNCS
HordETHStakingManager.sol	94.5%	82.81%	100.00%

We are grateful for the opportunity to work with the Hord team.

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

Zokyo Security recommends the Hord team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo