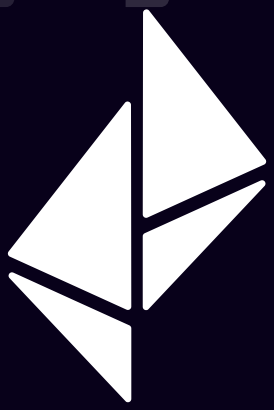


HORD

SMART CONTRACT AUDIT

zokyo

May 24th 2024 | 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
- 2 Medium issue: 2 resolved = 0 points deducted
- 3 Low issues: 2 resolved, 1 verified = 0 points deducted
- 3 Informational issues: 3 verified = 0 points deducted
- 2 points are deducted as for the contracts upgradeability (controllable backdoor)

Thus, 100 - 2 = 98

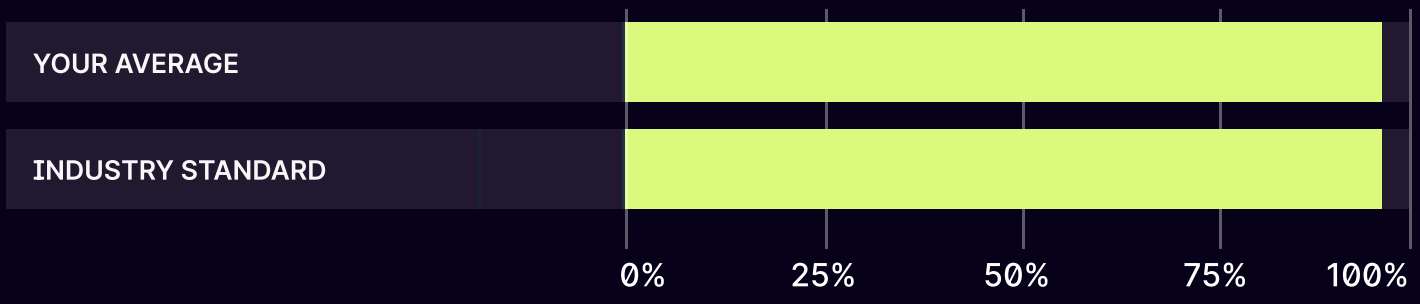
TECHNICAL SUMMARY

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: 1c1e86543d85bed739194b5d71f63d8cdccbef28
Final commit: a2b67a2faaf126ca53d273f59bda4934f9c7d56e

The scope includes the 4th audit iteration (3rd upgrade re-audit) and is based on the previously audited codebase. For the previous audit iterations refer to the reports from Dec 27th (2023), Sep 19th (2023) and Feb 22nd (2023):

Previous version of code: cee0dd705f7b04d1eb4700ebefdea311a335e11c

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

- configurations/StakingConfiguration.sol (14 lines changed)
- ethStaking/HordETHStakingManager.sol (406 lines changed)
- ethStaking/HordETHStakingWithdrawalManager.sol (9 lines changed)

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.

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

Executive Summary

Zokyo Security conducted another consecutive audit iteration over the Hord protocol. The current audit iteration included a new setting in the StakingConfiguration, a set of changes over the staking and withdrawal managers. A detailed overview of changes is available in the Protocol Overview section. However, as a summary - the protocol is enhanced with the tracking of ETH coming in/out of the contract and changes in the tracking of LSD tokens (now based on the contract balance). Also, the Hord team integrated the Chainlink price feed for the stETH price tracking and the flow for registering validators in the SSV network. Other changes are utilitary.

The audit's goal was to verify the safety of user deposits, confirm the integrity of the protocol after the patching, and verify the correctness of the calculations during the LSD tokens deposit and fee minting. Also, auditors checked the contract's code against the internal checklist of vulnerabilities, validated the updated business logic of the contract, and ensured the integrity of all operations (including registering the SSV validator as the newest feature). The report contains two medium issues connected to the validator registering flow and missed access control check. Other issues refer to missed validations (including the consistency check during the ETH movement tracking) and the required feedback regarding the ambiguity of some of the newest changes. All issues were successfully fixed or verified.

However, auditors still need to notice that the Hord team relies on manual operations during the upgrade. Thus it creates a risk of human error - an appropriate warning is placed in the Analysis section. Also, auditors leave a note for the maintainers to monitor the price changes of the stETH and wstETH assets as they may affect the debt/profit relation for the protocol.

During the audit, the security team upgraded the accumulated test suite for the protocol and conducted several rounds of additional testing (including manual tests and tests for the SSV validator registration). The complete set of unit tests can be seen in the Code Coverage and Test Result sections.

In conclusion, the Hord contracts have a high security level and function correctly with a known level of security. The contracts worked as expected, and the auditors prepared tests to cover all the core logic, ensuring the contract's functionality and security. Nevertheless, the project still fails the check against the backdoors, as the contract set is upgradeable. Despite being a common approach, it still creates a controllable backdoor, which should be noted in the funds-bearing contract. From all other aspects, the protocol is reliable and has a high security level.

PROTOCOL OVERVIEW

Description section includes changes discovered during the current audit iteration, refer to the previous audit report for more details on the protocol mechanics.

StakingConfiguration.sol:

- new parameter timeBetweenSetStatsCalls (with appropriate setter available for the congress only).

HordETHStakingWithdrawalManager.sol:

- **t**he amount of ETH to withdraw (via requestWithdraw()) is now redeemed via the new getAmountOfHETHforETH() function which returns HETH/ETH ratio, and then scaled appropriately based on 1e18 accuracy.

HordETHStakingManager.sol

1) State

- totalHETHMinted variable became obsolete and its storage is marked as deprecated

New variables:

- token addresses: stETH. wstETH
- ssvNetwork contract instance
- ssvToken token instance
- lastTimeUpdatedStats - time for the validator stats latest update. It works in together with the new setting from the StakingConfiguration
- totalETHOnContractRecorded variable - it represents contract's balance at the moment of the stats last update
- amountETHDepositedInMeantime - it tracks the amount of ETH deposited to the contract in-between validator stats updates
- amountETHUsedToPayDebtInMeantime - it tracks the amount of ETH transferred from the contract in-between the validator stats updates

2) error messages

- error messages have become concise
- a NatSpec documentation for errors was added

PROTOCOL OVERVIEW

HordETHStakingManager.sol

3) Track of funds

- appropriateReservesToPayDebt(), moveETHToAssuredBalance() - now track the total amount of funds transferred out
- swapLSDToExactAmountOfETH(). depositWithLSDToken() now track the total amount of funds transferred in (either via LSD to ETH conversion or via deposit)
- both tracks are refreshed during the validator stats update

4) New setters:

- initialSetters() - temporary function for testing purposes (to be removed before the mainnet)
- setSSVSetAddresses()

5) LSD token deposit

- is now tracked based on ERC20 balance on the contract instead of the on-token storage change (depositWithLSDToken, removeLSDToken)
- depositWithLSDToken() supports direct deposit in stETH (based on epy current price from Chainlink)

6) setValidatorStats function

- integrates the newly added parameter from the StakingConfiguration
- checks the tracked ETH movements against the moment of L2 to L1 balance movement and the moment of the validator liquidation
- refreshes the tracked ETH movements in the end of the operation
- fee minting is moved into the separate function

7) Fees minting

- mintProtocolFees handles fees functionality separately from the stats update
- fees minting is now independent from the validator stats update process and can be performed at any moment, though it still matches the previous fee minting process
- fees minting now ignores tolerance percentage

PROTOCOL OVERVIEW

HordETHStakingManager.sol

8) SSV validator
- new functions for interaction with SSV Network: registering validators and native staking performing
- SSV validator should be registered only after the 32 ETH deposit to the beacon contract (regular validator first - SSV validator after that)

9) wrapStETH function
- utility wrapper of stETH into wstETH

10) Changes in getters
- getAmountOfHETHforETH() is removed (alternative getter is now used by the withdrawal manager)
- getLSDTokenBalanceInETH() is modified to include stETH direct deposits and handling of LSD from the contract balances

11) Chainlink price feed integrated
- getLatestTokenPrice function uses the chainLinkAggregator for the stETH price in ETH

ROLES AND RESPONSIBILITIES

Section includes changes discovered during the current audit iteration, refer to the previous audit report for more details on the protocol mechanics.

HordETHStakingManager

1) Maintainer (new responsibilities)
- can mint protocol fees
- can launch new validator and register it via the ssv contract
- can remove validator from ssv contract
- can wrap stETH to wstETH

StakingConfiguration

1) HordCongress
- new responsibility on setting the time between set stats calls

LIST OF VALUABLE ASSETS

HordETHStakingManager
1. stETH after the direct deposit
2. wstETH tokens (after the stETH wrapping)

Note: Currently wstETH can be received after the conversion of the stETH deposited by the user. Therefore, the user gets hETH tokens for the stETH deposit, which is later converted into the wstETH. Therefore, it creates a risk of encountering impact of the price fluctuation of the wstETH - negative for the protocol in case of the wstETH inflation (as debt for users will be increased). This note should serve as a general warning for the maintainers.

3. ssvToken (during the transit into the SSV validator)

SETTINGS

As after the upgrade
1) timeBetweenSetStatsCalls
New parameter is introduced in the StakingConfiguration contract. It is responsible for the delay between the next possible update of the validator stats.

2) ssvNetwork and ssvToken addresses.

3) stETH, wstETH, hordETHStakingWithdrawalManager, dubUniswapV3, WETH, swapRouter addresses are now set via the initialSetters() function, that will be removed before the mainnet upgrade since it's only required values for testing purposes. Values for these variables are already set on prod.

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.

COMPLETE ANALYSIS

MEDIUM-1 | RESOLVED

Missing validator's registering

HordStakingManager.sol, launchNewValidatorWithSSV()
The `removeValidatorFromSSV()` function contains the check on validator to be initialized: `require(isValidatorInitialized[pubkey], "VNI")`. However, this pubkeys mapping is not set during the SSV validator registration. Therefore, it is not possible to remove a validator, unless it was already registered as a regular validator (via the launchNewValidator())

Recommendation:
Based on the launchNewValidator() function, consider adding another pubkeys mapping isSSVValidatorInitialized SSV related functions. Note, that it is important to make this pubkeys mapping different from the existing one, to avoid intersecting of SSV and regular validators. OR verify that validators' lists will not intersect, and thus a single mapping can be used - in that case add validators to the mapping in launchNewValidatorWithSSV().

Post audit.
Validators are now tracked within the same registry (isValidatorInitialized). The Hord team verified that there will be no intersection between SSV validators and regular validators.

MEDIUM-2 | RESOLVED

Missing access modifier

StakingConfigurator.sol, setTolerancePercentageForRewards()
The `setTolerancePercentageForRewards()` function can be called by anyone, so tolerance percentage for rewards can be changed without HordCongress proposal mechanism.

The issue is marked as Medium, as despite the missing authorization check, the setting itself has low impact on the funds processing.

Recommendation:
Add `onlyHordCongress` access modifier.

LOW-1 | RESOLVED

Missing parameter's boundaries validation

StakingConfiguration.sol, setTimeBetweenSetStatsCalls()
New parameter (timeBetweenSetStatsCalls) was introduced, though no validations were performed. It is recommended to have several validations for each parameter:

- secure default value
- min and max boundaries
- extreme values (in case if boundaries are not introduced)

Though, no default value is set (including deployment/tuning/upgrade scripts) and there is no min/max values to validate against. Therefore the parameter is exposed to the risk of misconfiguration and/or human error.

The issue is marked as Low, as despite the risk of misconfiguration, the parameter can be changed at any time by the congress. However, the change of the incorrectly set parameter may take some time (based on the congress procedure), thus creating a window of instability.

Recommendation:
Provide the default value for the parameter, introduce reasonable min/max boundaries for the parameter.

Post audit.
Parameter is validated against the max value of 10 days

LOW-2 | RESOLVED

Function never called

HordETHStakingManager.sol, _launchNewValidator()

Function is never called, and it seems to be an internal clone of the launchValidator() function.

Recommendation:
Delete the unused function or finalize the functionality.

Missing consistency checks for the recorded ETH movements

HordETHStakingManager.sol, setValidatorStats()
After the upgrade, the contract utilizes new funds tracking mechanism:

- totalETHOnContractRecorded to track the balance after the last stats update
- amountETHDepositedInMeantime and amountETHUsedToPayDebtInMeantime to track in/out movements

However, there is no validation for the consistency after the upgrade. Consider case when values are not validated after the upgrade and the next scenario:

- after the upgrade, either ETH move or debt payment (for the withdrawal manager) can be performed (as it is a running contract)
- In that case, the setValidatorStats() will fail, as the recorded balance is `0`, though debt is already set.

The issue will be resolved either after the appropriate number of deposits or with another contract upgrade.

Recommendation:
Add validations to restrict debt increasing operations before the syncing of the total ETH stored OR verify that the setValidatorStats() will be called immediately after the upgrade before other maintenance operations.

Post audit.
The Hord team confirmed that the protocol will be paused during the upgrade and that the team will set the initial values via the setValidatorStats() call. The function call is available for the backend only (maintainer role), thus it will be performed in manual mode instead of including into the deployment/upgrade scripts. Although the security team still have a concern regarding the risks of the manual operation processing, the Hord team assured that the operation will be monitored by the team to be executed accordingly to make sure the state is synced.

INFORMATIONAL-1 | VERIFIED

Event parameter changed the meaning
HordETHStakingWithdrawalManager.sol, requestWithdraw()
NewUserWithdrawalRequest event emission
Before the change, the event represented the current value of reserves (totalETHReservesForPayingDebt()) as for the end of the operations. After the change the event represents the value before the operation - as the value of unAllocatedBuffer (used in the totalETHReservesForPayingDebt() is changed during the function execution and so the event emits the cached value from the beginning of the function.

The issue is marked as Info, as the change does not affect the onchain logic, however can affect the further offchain processing of the event data, thus the feedback is required regarding what value should be emitted (for the start of the operation, or for the end).
Recommendation:
Verify the change of the emitted data and its impact on the offchain processing.
Post audit.
The Hord team verified it as the desired behavior - parameter now represents the initial state of reserves for the further backend processing. The parameter replacement was performed based on the backend spec, and the value is used only by the backend.

INFORMATIONAL-2 | VERIFIED

Wrapped stETH is never used
HordETHStakingManager.sol, wrapStETH()
wstETH received after wrapping of stETH is never used or approved, and so - effectively locked on the contract.
The issue is marked as Info, as it refers to the business logic decision and requires the feedback from the Hord team before classification.
Recommendation:

Verify the usage of the wstETH and that it is supposed to be processed via the swapLSDToExactAmountOfETH(). Consider adding validation that the wStETH will stay as one of the supported LSD tokens.

Post audit.
The Hord team confirmed, that the wstEth should be locked on the contract until the maintainter will decide to process the swap to Eth (to launch the new validator or to pay the debt to the users)

INFORMATIONAL-3 | VERIFIED

Precision value has double meaning

HordETHStakingManager.sol, getLSDTokenBalanceInETH()
HordETHStakingManager.sol, depositWithLSDToken()

Looks like stakingConfiguration.percentPrecision() has double meaning in terms of application. It is used as an intermediate accuracy multiplier for the price calculation and as an price lining up to token decimals (as for the accuracy correction after the chainlink price extraction). Though, in a general case, the intermediate accuracy for HETH/ETH price calculation does not equal the accuracy for the oracle price lining up for stETH

Recommendation:
Verify that indeed the same value is required for both cases (1e18) and consider using a separate value for chainlink price lining up

Post-audit:
The Hord team verified the value to be the same for both cases.

	HordETHStakingManager.sol HordETHStakingWithdrawalManager.sol StakingConfiguration.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 Hord 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 Hord contract requirements for details about issuance amounts and how the system handles these.

HordETHStakingManager

after initialization
✓ Should revert if call initialization
during initialization
✓ Should revert if stacking cfg addr is zero addr
✓ Should revert if beacon addr is zero addr
 # 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
✓ Should revert if caller is not hord congress member
 # Deposit with ETH
✓ Should allow user to deposit ETH
✓ Should allow to deposit multiple times for 1 user
✓ Should revert if paused
✓ Should revert if sent 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
✓ Should revert if trying to whitelist token twice
✓ Should revert if token address is zero address
✓ Should revert if pool address is zero address
✓ Should revert if caller is not hord congress
 # Remove LSD token
✓ Should allow to remove LSD token

✓ Should revert if caller is not hord congress
✓ Should revert if LSD token is zero address
✓ Should revert if LSD token is not whitelisted
✓ Should allow to freeze LSD token
 # Deposit with LSDToken
✓ Should allow to deposit with lsd token
✓ 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
✓ Should revert if caller is not maintainer
✓ Should revert if pool address is not matching with token info
✓ Should revert when paused
✓ Should revert if LSD tokens are not enough
 # Launch a new validator
✓ Should allow to launch new validator
✓ Should revert if validator is already initialized
✓ Should revert if conditions are not met
✓ Should revert if contract is paused
✓ Should revert if caller is not maintainer
 # Move ETH to assured balance
✓ Should revert when paused
✓ Should revert if caller is not hord withdrawal manager
 # Set validator stats
✓ Should allow to set validator stats
✓ Should revert if caller is not maintainer
✓ Should revert if trying to set twice too soon
✓ Should revert if provided wrong new rewards amount
✓ Should revert if provided wrong new execution layer rewards
✓ Should allow to set if paused
✓ Should pause if new rewards amount is greater than collected + boundary
 # Mint protocol fees
✓ Should allow to set validator stats
✓ Should revert if caller is not maintainer
 # Set SSV addresses
✓ Should allow to set SSV addresses
✓ Should revert if caller is not hord congress
✓ Should revert if network address is zero address
 # Launch new validator with SSV
✓ Should allow to launch correct new validator with SSV
 # Remove validator from SSV
✓ Should allow to remove validator from SSV

 # Wrap StETH
✓ Should allow to wrap StETH
✓ Should revert if caller is not maintainer
✓ Should revert if not enough stETH tokens on the contract
 # Get LSD token balance in ETH
✓ Should return correct balance in ETH

HordETHStakingWithdrawalManager

 # Withdrawal
Two depositors scenario
✓ Withdrawal by the first depositor
✓ Withdrawals by the second depositor with closing out the debt to users because of closed validator
✓ Withdraw organization funds
✓ Withdrawal debt length
✓ Get withdrawal debts

Staking configuration

 # Initialize
✓ Should revert if call initialization
 # Set tolerance percentage for rewards
✓ Should revert if caller is not hord congress
✓ Should revert if upper limit reached
✓ Should allow to set tolerance percentage for rewards
 # Set time between set stats calls
✓ Should revert if caller is not hord congress
✓ Should revert if time is 10 days or more
✓ Should allow to set time between set stats calls

Withdrawal scenarios

 # Withdrawal scenarios
✓ Withdraws with protocol fee
✓ Withdraws and verification of shares calculation (with several depositors)
✓ Stakes and withdraws with a lot of users
✓ Withdraws with a lot of users (amount ETH to withdraw > total ETH reserves)
✓ Request 0 amount for withdrawal

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

FILE	% STMTS	% BRANCH	% FUNCS
HordETHStakingManager.sol	95.93%	84.81%	100.00%
HordETHStakingWithdrawalManager.sol	95.42%	63.49%	100.00%
StakingConfiguration.sol	100%	98.61%	100.00%

Note: Code coverage in the current audit iteration is based on the test suite accumulated from the previous iteration (including the native tests from the protocol). Also, the testing process included a around of manual tests and hypotheses testing, which concludes the coverage together with unit tests.

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