HORD

SMART CONTRACT AUDIT

zokyo

February 22nd 2023| v. 1.0

Security Audit Score

PASS

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

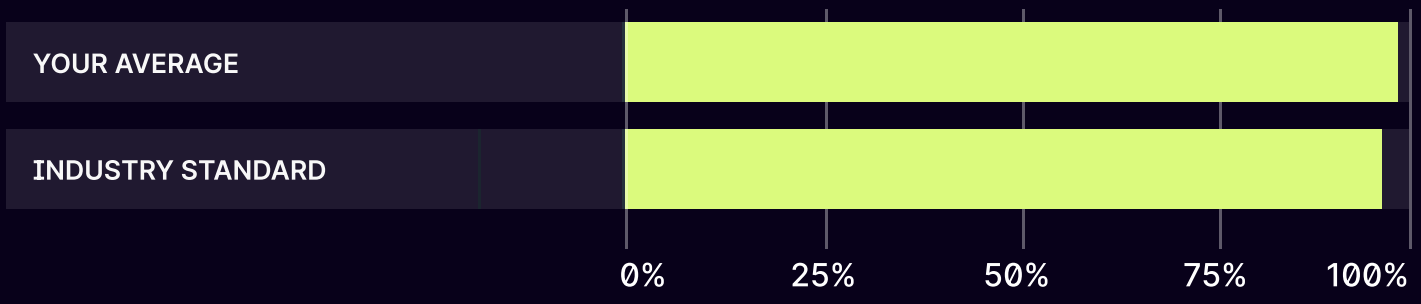
TECHNICAL SUMMARY

This document outlines the overall security of the 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



Contracts have sufficient test coverage, prepared by both Hord team and Zokyo Security team.

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

Intial commit: `0c7c529e44958df8a0ba94fd63040f8dcb58f952`
Final commit: `c3b0e02e6b42f2ffe524f3f36990d459e0555533`

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

- HETH.sol
- HordETHStakingManager.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 resent 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.	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

During the audit, the Zokyo Security team reviewed both smart contracts. Contracts represent the ETH staking part of the Hord protocol. HordETHStakingManager is an ERC20 token allowing users to wrap ETH into HordETH. The ratio between ETH and HETH depends on validators' stats taken from beacon-chain API. These stats include block_proposal_eth1_rewards, mev_eth1_rewards, and the amount of ETH total staked by the validator. The maintainer puts stats to the smart contract. However, it is essential that the provided information is valid and corresponds to the actual info about validators. Otherwise, the ETH and HETH ratio may not be current. The contract also interacts with TokensFarmSDK.sol. TokensFarmSDK is responsible for tracking users' balances in HETH tokens and distributing rewards based on balances and total supply. Info about stakes is updated during ETH deposit and HETH transfer.

The audit's goal was to analyze smart contracts against the list of common vulnerabilities, check that the implementation corresponds to the business logic of the protocol, and validate that the code corresponds to Solidity best practices in terms of code quality and gas optimizations.

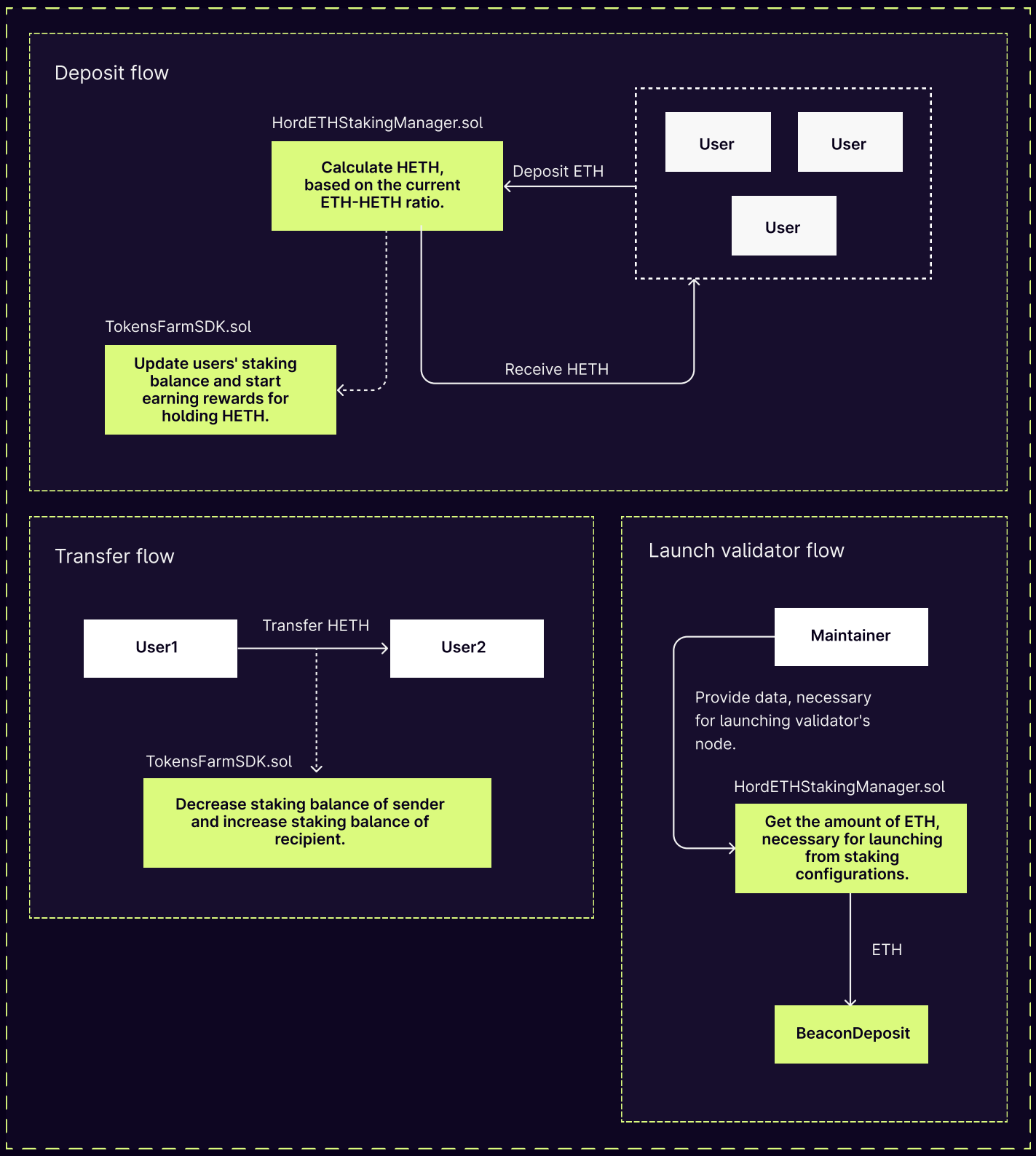
The manual audit found two high, one low, and several informational issues. One of the high issues showed the inability of the Staking Manager to support a flat fee on TokensFarmSDK. As a result, if it is considered to enable a flat fee on TokensFarmSDK, basic operations of Staking Manager would revert. The Hord team has verified that the flat fee won’t be enabled; thus, it is unnecessary to support it. Another high-severity issue showed the possible frontrun attacks. Though, it was verified by the Hord team and additionally tested by the Zokyo Security team that frontrun attacks can’t affect the protocol. Therefore, both issues were classified as false-positive and decreased to Info notes (since both have valuable info to be present in the report).

Other issues were connected to the absence of share variables validation, lack of events in setters, unused functions, validation of ERC20 interface in HETH.sol, and absence of HETH redemption mechanism. All of these issues were successfully fixed or verified by the Hord team. It should also be noted, that both HETH and HordETHStakingManager are upgradable ERC20 smart contracts, which means that the owner of the protocol can upgrade their implementation at any time. However, thanks to it, Hord team will be able to implement new features, such as a redemption mechanism, in the future.

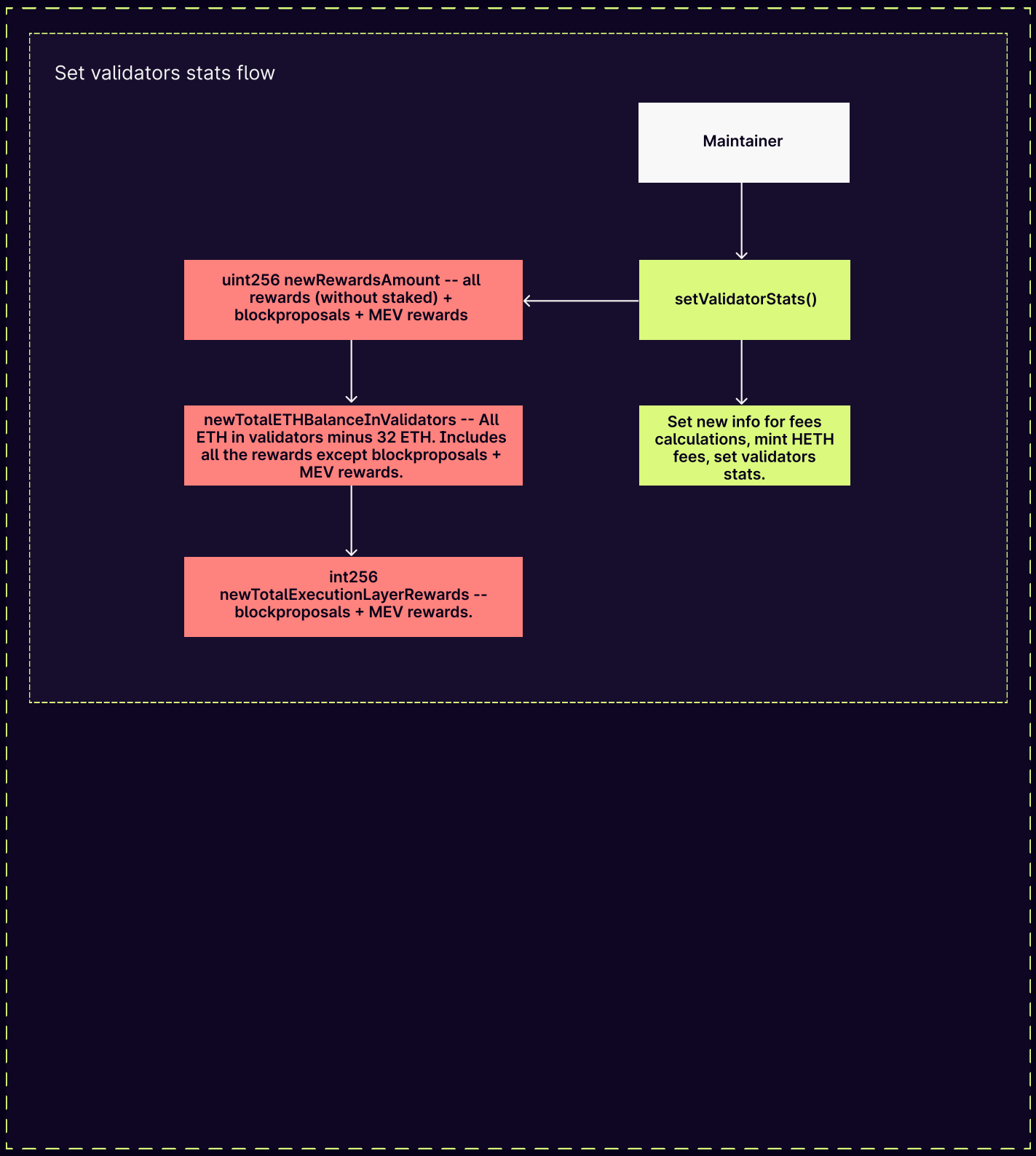
The overall security of ETH staking protocol is high enough. Contracts are well-written and have good natspec documentation. Also, the auditors note that the smart contracts are well-tested by the Hord and Zokyo Security teams. Zokyo Security team has verified smart contracts against the list of common vulnerabilities, analyzed smart contracts against the internal checklist of issues, checked the implementation of ERC20, and validated the business logic of the smart contracts. During the discussion of the issues with the Hord team, two high-severity issues were marked as info, as they were verified and tested.

PROTOCOL OVERVIEW

Hord ETH staking scheme



Hord ETH staking scheme



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

LOW-1 | RESOLVED

Percentage variables are not validated.

HordETHStakingManager.sol: setValidatorStats(), lines 157, 158, 169, 172.
During every calculation, the contract takes percentage variables from the Staking Configuration, and divides it by 100. However, there are no validations in setters of Staking Configuration that percentage variables are less than 100. Thus calculated values can be greater than 100%. The issue is marked as low since only Hord Congress can set percentage values in Staking Configuration OR it is valid for the protocol if percentage values are greater than 100%. In any case, the Hord team should verify the behavior.

Recommendation:
Validate that percentage values don't exceed 100 **OR** verify that percentage values can exceed 100.

Post-audit:
Validation was added in Staking Configuration smart contract.

INFO-1 | RESOLVED

Lack of event.

HordETHStakingManager.sol: setTokensFarmSDK().
In order to keep track of historical changes of storage variables, it is recommended to emit events on every change in the setter.

Recommendation:
Emit event in setter.

INFO-2 | VERIFIED

Unused functions.

1) HETH.sol: _setupDecimals().

Currently, this function is not used. According to the commentary section, this function should only be used only during initialization, and it is unlikely that it will be used in future smart contract upgrades.

2) HETH.sol: _burn().

The internal burning function is defined. However, both HETH and HordETHStakingManager have no public interface for it.

Recommendation:

Remove unused functions **OR** add usage of them **OR** verify they are necessary in future upgrades of smart contract.

From client:

All unused functions are planned for future development.

INFO-3 | VERIFIED

HETH doesn't inherit IERC20.

Since HETH.sol is a custom implementation of ERC20 token, it is recommended to explicitly show that contract inherits and implements interface of IERC20 token, to ensure that it can be supported as a standard ERC20 token.

Recommendation:

Inherit IERC20 interface from OpenZeppelin.

Post-audit:

According to the Hord team, all the functions are implemented in correspond of IERC20 interface, thus there is no need to inherit IERC20.

INFO-4 | VERIFIED

No redemption mechanism.

HordETHStakingManager.sol
Currently, users can only deposit ETH and receive HETH in exchange. However, there is no redemption mechanism implemented in the contract. Thus users can't receive ETH back. The issue is marked as info since this is a business logic validation rather than a security concern. Since HordETHStakingManager is an upgradable smart contract, it should be verified in case such functionality is planned for the future.

Recommendation:
Verify that HETH shouldn't be exchanged back into ETH **OR** that such functionality will be implemented later.

From client:
Redemption mechanism will be implemented in the future.

INFO-5 | VERIFIED

Contract doesn't support the flat fee of TokensFarmSDK.

HordETHStakingManager.sol: userDepositETH(), line 245.
HETH.sol: _beforeTokenTransfer(), lines 302, 307.
When the user performs deposit or stake reducing on TokensFarmSDK, a certain flat fee might be collected (if the flat fee is allowed). Thus in case a flat fee is enabled on TokensFarmSDK, depositing of ETH or transferring of HETH will be blocked. The issue is marked as high since it might break the business logic of the protocol in case a flat fee should be supported.
Note: the issue was re-classified as info after the discussion with the Hord team.

Recommendation:
Add a support of flat fee **OR** verify that TokensFarmSDK where HETH will be a staking token, flat fee won't be used.

From client:
According to the Hord team, flat fee will be disabled on TokensFarmSDK.

Slippage validation is absent.

HordETHStakingManager.sol: userDepositETH().
When a certain asset is exchanged for another one, it is necessary to validate the output value of another asset so that users receive an expected amount. Such validation is essential to protect protocol users against frontrun attacks and other unexpected ratio changes between HETH and ETH. Thus, auditors recommend passing a `minAmountOut` parameter and the validation that the minted amount of HETH corresponds to it.

Recommendation:
Pass a `minAmountOut` parameter to function and validate that `amountForMint` is greater or equal to `minAmountOut`..

From client:
Hord team has verified the mechanism behind HETH:ETH ratio calculation. According to the team, the only way to change the ratio is to add new rewards to the protocol, which is why frontrun attacks can't be applied to the protocol. Thus, users will receive the exact amount of HETH they deserve.

Post-audit:
Auditors provided additional testing of the issue and didn't confirm it. Frontrun attacks had no effect on the protocol and the only way to change the ratio between ETH and HETH is to add rewards with maintainer function setValidatorStats(). Thefore the issue was reclassified as info

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

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.

Contract: HordETHStakingManager

- ✓ initializer: cannot be called twice
- ✓ initializer: StakingConfiguration can not be 0x0 address
- ✓ initializer: BeaconDeposit can not be 0x0 address
- ✓ deposit ETH (50ms)
- ✓ cannot deposit ETH while contract on pause
- ✓ cannot deposit ETH with zero eth sent
- ✓ Deposit ETH two times from the same user (71ms)
- ✓ setTokensFarmSDK
- ✓ tokens FarmSDK cannot be set as zero
- ✓ pause contract
- ✓ only HordCongressOrMaintainer can pause contract
- ✓ only congress member can unpause contract
- ✓ unpause contract
- ✓ getAmountOfHETHforETH contract call
- ✓ getAmountOfHETHforETH not contract call
- ✓ ValidatorStats can be set only by maintainer
- ✓ setValidatorStats wrong newRewardsAmoun
- ✓ setValidatorStats wrong newTotalExecutionLayerRewards
- ✓ setValidatorStats while on pause
- ✓ only maintainer can launchNewValidator
- ✓ cannot launchNewValidator while contract on pause
- ✓ launchNewValidator

Additional scenario

Depositing ETH #1
Depositing ETH #2
Launching new validator
Depositing ETH #3
Setting validator's stats
Depositing ETH #4
✓ Protocol flow (272ms)
1) Should deposit if flat fee is enabled on TokensFarmSDK
✓ Should update stake info during transfer (71ms)

FILE	% STMTS	% BRANCH	% FUNCS
HETH.sol	100	100	100
HordETHStakingManager.sol	97.83	76.79	100
All files	98.91	88.4	100

All the main functionality of HordETHStakingManager.sol was carefully tested. In particular, auditors have tested:

- ETH Depositing
- Interaction with TokensFarmSDK
- Validators stats setting up
- Launching of new validators.

As HETH.sol is the ERC20 token, auditors carefully reviewed all the functionality connected to ERC20 and checked the correspondence to the standard. Security team also tested all extra functionality, such as interaction with TokensFarmSDK in HETH.sol. Therefore, test coverage for HETH.sol includes the formal review and tests for additional functionality.

It should also be noted, that although the test “Should deposit if flat fee is enabled on TokensFarmSDK” is failing, the Hord team verified, that flat fee will be disabled on TokensFarmSDK.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the Hord team

As a part of our work assisting Hord in verifying the correctness of their contract code, our team has checked the complete set of tests prepared by the Minterest team.

We need to mention that the original code has a significant original coverage with testing scenarios provided by the Hord team. All of them were also carefully checked by the team of auditors.

```
ETHStakingManager
    ETHStakingManager::Pausable/Setters
    ✓ should let maintainer to pause contract (335ms)
    ✓ should not let maintainer to unpause contract (110ms)
    ✓ should let hordCongress to unpause contract (151ms)
    ✓ should setup proposal information (691ms)
    ETHStakingManager::UserDepositETH
    ✓ should not let user to deposit 0 ETH (112ms)
    ✓ should let users to deposit (491ms)
    ✓ should check states after deposit (94ms)
    ✓ should check values in Deposit event
    ✓ should check values in HETHMinted event
    ✓ should let user to deposit again (257ms)
    ETHStakingManager::LaunchNewValidator
    ✓ should not let maintainer to launch new validator if there is not enough ETH (109ms)
    ✓ should deposit more to reach conditions (852ms)
    ✓ should let maintainer to launch new validator (214ms)
    ✓ should check states after launchNewValidator() function (59ms)
    ETHStakingManager::rewards
    - should check return value in getRatioBetweenETHAndHETH() function
    ✓ should update total rewards value (236ms)
    HETH
    ✓ should check return values in name() function (53ms)
    ✓ should check return values in symbol() function (41ms)
    ✓ should check return values in decimals() function
    - should check return values in totalSupply() function

  18 passing (7s)
  2 pending
```

FILE	% STMTS	% BRANCH	% FUNCS
HETH.sol	27.91	8.33	31.58
HordETHStakingManager.sol	98.46	70	100
All files	**70.37**	**52.38**	**55.17**

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