CYBER UNIT
Cyber Security Strategic Partner
www.cyberunit.tech

Hord Smart Contract Audit

Date: April 7, 2021
Report for: Hord
By: CyberUnit.Tech

This document may contain confidential information about IT systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.
The report containing confidential information can be used internally by the customer, or it can disclose publicly after all vulnerabilities are fixed – upon the decision of the customer.

Scope and Code Revision Date

Repository	https://github.com/hord/smart-contracts
Files	contracts/token folder contracts/governance folder
Initial Audit Commit	9c89a001409601b7821b72fa94d9f5db50355b50
Initial Audit Date	31.03.2021
Secondary Audit Commit	9151ecb05dcb236d3075800cbff9e6cda584c0c0
Secondary Audit Date	07.04.2021

Table of contents

Document 2
Introduction 4
Scope 4
Executive Summary 5
Severity Definitions 5
AS-IS overview 6
Audit overview 8
Conclusion 10
Disclaimers 11

Introduction

This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between March 24th 2021 – March 31st 2021; Secondary audit conducted between April 5th 2021 – April 7th 2021

Scope

The scope of the project is Hord smart contracts, which can be found in repo:
https://github.com/hord/smart-contracts
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the widely known vulnerabilities that considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call – Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Hord smart contracts security risk is low.

Several critical, high, medium issues were found for the smart contract during initial audit, however, were fixed or accepted for the secondary audit.

Our team performed an analysis of code functionality, manual audit and automated checks with Slither and remixed IDE. All issues found during automated investigation manually reviewed and application vulnerabilities presented in the Audit overview section. A general overview presented in the AS-IS section and all encountered matters can be found in the Audit overview section.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss.
High	High-level vulnerabilities are difficult to exploit. However, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium-level vulnerabilities are essential to fix; however, they can't lead to tokens loss.
Low	Low-level vulnerabilities are mostly related to outdated or unused code snippets.
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can generally be ignored.

AS-IS overview

Context.sol

Context is a standard contract for providing execution context.

HordToken.sol

HordToken is a smart contract for custom ERC20 token.

Contract HordToken is Context, IERC20, IERC20Metadata.

HordToken has following parameters and structs:

- mapping (address => uint256) private _balances;
- mapping (address => mapping (address => uint256)) private _allowances;
- uint256 private _totalSupply;
- string private _name;
- string private _symbol;

HordToken contract has following functions:

- constructor – public function that mints total supply to specified address, sets token parameters
- name – public view function that returns token name
- symbol – public view function that returns token symbol
- decimals – public view function that returns 18
- totalSupply – public view function that returns token total supply
- balanceOf – public view function that returns user balance
- transfer – public function that calls internal transfer function
- allowance – public view function that returns spender allowance for account
- approve – public function that calls internal approve function
- transferFrom – public function that calls internal transfer for specified spender; changes approved amount
- increaseAllowance – public function that calls internal approve to increase spender allowance
- decreaseAllowance – public function that calls internal approve to decrease spender allowance
- _transfer – internal function that performs token transfer
- _mint – internal function that mints new tokens
- _burn – internal function to burn tokens
- _approve – internal function to change allowance
- _beforeTokenTransfer – internal function without any logic

HordCongress.sol

HordCongress is a smart contract for governance.

HordCongress has following parameters and structs:

- string public constant name = "HordCongress";
- IHordCongressMembersRegistry membersRegistry;
- uint public proposalCount;
- mapping (uint => Proposal) public proposals;
- struct Proposal
    - uint id;
    - address proposer;
    - address[] targets;
    - uint[] values;
    - string[] signatures;
    - bytes[] calldatas;
    - uint forVotes;
    - uint againstVotes;
    - bool canceled;
    - bool executed;
    - mapping (address => Receipt) receipts;
- struct Receipt
    - bool hasVoted;
    - bool support;

HordCongress contract has following functions and modifiers:

- onlyMember modifier – checks whether address is in members registry
- setMembersRegistry – public function that sets address for members registry
- propose – public function that creates new proposals. Has onlyMember modifier.
- castVote – public function that calls internal cote cast
- execute – public function that executes proposal. Has onlyMember modifier.
- cancel – public function that cancels proposal. Has onlyMember modifier.
- _castVote – internal function to cast the vote
- get actions – public view function that returns proposal data
- getMemberRegistry – public view function that returns member registry address
- add256 – internal view function for preventing addition overflows
- sub256 – internal view function for preventing subtraction overflows
- receive – external payable function to receive Ether

HordCongressMembersRegistry.sol

HordCongressMembersRegistry is a smart contract for managing Hord congress members.

HordCongressMembersRegistry has following parameters and structs:

- string public constant name = "HordCongressMembersRegistry";
- address public hordCongress;
- uint256 minimalQuorum;
- mapping (address => bool) isMemberInCongress;
- apping(address => Member) public address2Member;
- address[] public allMembers;
- struct Member
    - address memberAddress;
    - bytes32 name;
    - uint memberSince;

HordCongressMembersRegistry contract has following functions and modifiers:

- onlyHordCongress modifier – checks whether address is Hord Congress
- constructor – public function that initializes starting members and sets Hord Congress address
- changeMinimumQuorum – public function that changes minimal quorum amount for proposal. Has onlyHordCongress modifier
- addMember – public function that calls addMemberInternal. Has onlyHordCongress modifier
- addMemberInternal – internal function for adding new members to registry
- removeMember – public function that removes Hord member. Has onlyHordCongress modifier
- isMember – public view function that returns true if address is member
- getAllMemberAddresses – public view function that returns a list of all member addresses
- getMemberInfo – public view function function that returns member info
- getMinimalQuorum – public view function function that returns minimal quorum value

Audit overview

Critical

1. [Fixed in 9151ecb] castVote and _castVote have no onlyMember modifier, thus, any addresses can vote on the proposals. One member will be enough to execute any proposal.
2. [Fixed in 9151ecb] All new proposals can be cancelled by rogue member. If a malicious member sees a new proposal, he calls cancel and it's cancelled. It will also be impossible to remove him from members. Proposal should be canceled only in case of reaching some threshold for negative votes.

High

3. [Fixed in 9151ecb] removeMember function has no event emitted.

Medium

4. [Fixed in 9151ecb] increaseAllowance contains integer overflow in the 3rd parameter of _approve function. It's possible to decrease allowance or set it to 0 via increaseAllowance function.
5. [Risk accepted by Customer] setMembersRegistry can be called by anyone before legit call. It's recommended that only specified address could set members registry address.
6. [Not an issue] When a new member is added, minimum quorum sets to all members, however, it should be increased by 1.

Low

7. [Fixed in 9151ecb] _beforeTokenTransfer function has an empty body, however, called for each transfer. The function should be removed.
8. [Fixed in 9151ecb] _mint and _burn functions are implemented and can't be ever called. It's recommended to use standard openZeppelin templates with required functionality.
9. [Risk accepted by Customer] Hord Congress contract has pragma experimental ABIEncoderV2; it's not recommended to use experimental pragma for production contracts.
10. [Fixed in 9151ecb] propose function checks address membership twice – in the modifier and in the function body. Double check wastes extra gas.
11. [Risk accepted by Customer] Solidity version is not locked, it's recommended to lock pragma to specific version.
12. [Fixed in 9151ecb] Member struct doesn't need to store member address because it's stored in address2member mapping. Storing it in struct will waste extra gas.

Lowest / Code style / Best Practice

13. [Risk accepted by Customer] SafeMath library is not explicitly used within contracts. For example, Hord Congress implements custom overflow protection functions, which, potentially, may increase gas usage.
14. [Fixed in 9151ecb] It's recommended for `getMemberInfo` to have an address as param so anyone could view info for any address.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality presented in As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found several critical, high, medium issues during the initial audit. All of the findings were fixed or accepted by Customer for the secondary audit. The overall risk is low.

Disclaimer

The smart contracts given for audit have analyzed following the best industry practices at the date of this report, concerning: cybersecurity vulnerabilities and issues in smart contract source code, the details of which disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit doesn't make warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the system, bug free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is essential to note that you should not rely on this report only. We recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee specific security of the audited smart contracts.