CYBER UNIT
Cyber Security Strategic Partner
www.cyberunit.tech

Smart Contract Audit

Date: May 24, 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

Contract	https://github.com/hord/smart-contracts/tree/develop/contracts
Commit	cbfddcb3bb06cb69266c9c2e0c23d20c4b276576
Files	HordTicketFactory.sol HordTicketManager.sol
Date	24.05.2021

Table of contents

Introduction

This report presents the findings of the security assessment of Customer` s smart contract and its code review conducted between May 17th 2021 – May 24th 2021.

Scope

The scope of the project is Hord smart contracts, which can be found on github:
https://github.com/hord/smart-contracts/tree/master/contracts

Files in scope:

HordTicketFactory.sol
HordTicketManager.sol

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 NFT smart contracts security risk is high; these risks should be reviewed and mitigated by Customer.

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

HordTicketFactory.sol

HordTicketFactory is a smart contract for ERC1155 token with minting and adding token supply functionality.

HordTicketFactory has following parameters and structs:

- uint256 public lastMintedTokenId;
- uint256 public maxFungibleTicketsPerPool;
- mapping (uint256 => uint256) tokenIdToMintedSupply;
- IHordTicketManager public hordTicketManager;

HordTicketFactory contract has following functions:

- initialize – public function that initializes contract state. Has initializer modifier.
- pause – public function that pauses some functionality. Has onlyHordCongress modifier.
- unpause – public function that unpauses some functionality. Has onlyHordCongress modifier.
- setMaxFungibleTicketsPerPool – external function that sets new value to maxFungibleTicketsPerPool. Has onlyHordCongress modifier.
- mintNewHPoolNFT – public function that mints new pool NFT. Has onlyMaintainer modifier.
- addTokenSupply – public function that adds supply to pool. Has onlyMaintainer modifier.
- getTokenSupply – external view function that returns supply minted for a token.

HordTicketManager.sol

HordTicketManager is a smart contract for .

HordTicketManager has following parameters and structs:

- uint256 public minTimeToStake;
- uint256 public minAmountToStake;
- IERC20 public stakingToken;
- IHordTicketFactory public hordTicketFactory;
- mapping (uint256 => uint256[]) internal championIdToMintedTokensIds;
- struct UserStake that stores uint256 amountStaked; uint256 amountOfTicketsGetting; uint256 unlockingTime; bool isWithdrawn;
- mapping(address => mapping(uint => UserStake[])) public addressToTokenIdToStakes;
- mapping(uint256 => uint256) internal tokenIdToNumberOfTicketsReserved;

HordTicketManager contract has following functions:

- initialize – public function that initializes contract state. Has initializer modifier.
- setHordTicketFactory – public function that sets new Hord ticket factory address.
- setMinTimeToStake – external function that sets new min time to stake. Has onlyHordCongress modifier.
- setMinAmountToStake – external function that sets new min amount to stake. Has onlyHordCongress modifier.
- addNewTokenIdForChampion – external function that sets new tokenId for champion.
- stakeAndReserveNFTs – public function that stakes and reserves NFTs.
- claimNFTs – public function that claims NFTs and returns staked tokens.
- getAmountOfTokensClaimed – external view function that returns number of tokens claimed by users.
- getAmountOfTicketsReserved – external view function that returns number of tokens reserved for specific ticket.
- getUserStakesForTokenId – external view function that returns user stakes for token.
- getCurrentAmountStakedForTokenId – external view function that returns current stake amount for user.
- getChampionTokenIds – external view function that returns all tokens for a champion.

Audit overview

Critical

No critical vulnerabilities found.

High

1. DoS via gas limit in claimNFTs function. If the user creates too many stakes, the length of addressToTokenIdToStakes[msg.sender][tokenId] will be big and the loop in claimNFTs function will fail. Users won't be able to claim NFT and withdraw funds.

Medium

2. setMaxFungibleTicketsPerPool allows setting maxFungibleTicketsPerPool to be less than the actual ticket amount for some pool. It means that some pools may have more tokens than maxFungibleTicketsPerPool.

Low

3. Solidity pragma is not locked for both contracts. It's recommended to lock the Solidity version to a specific stable one.
4. It's possible to calculate tokenId in the mintNewHPoolNFT function and not to pass it to it. All math operations and checks will require extra gas. Moreover, there is only one value that can be a parameter so the function won't fail.

Lowest / Code style / Best Practice

5. There is wrong description of contract on line 9 of HordTicketFactory – it says that it's HordTicketManager contract.

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 issues; one of them may cause tokens and NFT loss.

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.