HYDN

Dancing Seahorse

Smart Contract Audit Report

August 17, 2022

Prepared for:

Jose Soeiro, Cradleblock

Prepared by:

Warren Mercer, HYDN
James Williams, HYDN
David Goulbourn, HYDN

Contents

Introduction

About HYDN

HYDN is an industry leader in blockchain security and smart contract audits. Founded by Warren Mercer, a world renowned cybersecurity and blockchain expert, who has previously held senior roles at NYSE, Cisco, and Alert Logic. Warren has been a guest speaker at cybersecurity conferences all over the globe, including giving talks on Bitcoin at Microsoft DCC, NorthSec, Kaspersky SAS, VB, and many more.

Having been involved in cryptocurrency for over 10 years, Warren is dedicated to making the blockchain ecosystem as secure as it can be for everyone. Warren serves as the CEO for HYDN and heads up the delivery team to ensure that work is carried out to the highest standard. The HYDN delivery team has over 10 years combined experience in blockchain, having performed smart contract audits and built security systems for a large range of protocols and companies.

HYDN works closely with clients to consider their unique business needs, objectives, and concerns. Our mission is to ensure the blockchain supports secure business operations for all and he has built the team at HYDN from the ground up to meet this very personal objective.

To keep up to date with our latest news and announcements, check out our website https://hydnsec.com/ or follow @hydnsecurity on Twitter.

About Dancing Seahorse

Dancing Seahorse is a Web3 business disrupting the music industry, uniting fans, artists and musicians through unforgettable VIP experiences, with benefits and rewards for those that hold the Legendary Dancing Seahorse NFT. The Legendary Dancing Seahorse NFT is the access point for members to become a founding member of an exclusive Web 3.0 company. Holding a LDS NFT along with a valid DS Mint Pass gives the member rights to claim benefits on an ongoing basis from the Dancing Seahorse Company.

Nobody will be able to own a Legendary Dancing Seahorse NFT without holding a Mint Pass in their wallet. Legendary Seahorse NFT will be minted in ETH only, there will then be a reveal on the 20th September when traits and associated seahorse image will be applied . NFT will have some trait values which can be updated on the metadata stored in AWS S3 and the image associated with the Seahorse can also be

visually updated on the AWS S3 storage location. No changes to the traits and images will affect the rarity of the seahorse against standard traits added at time of reveal.

Methodology

When tasked with conducting a security assessment, HYDN works through multiple phases of security auditing to ensure smart contracts are audited thoroughly. To begin the process automated tests are carried out, before HYDN then moves onto carrying out a detailed manual review of the smart contracts.

HYDN uses a variety of open-source tools and analyzers as and when they are required and alongside this, HYDN primarily focuses on the following classes of security and reliability issues:

Basic Coding Mistakes

One of the most common causes of critical vulnerabilities is basic coding mistakes. Countless projects have experienced hacks and exploits due to simple, surface level mistakes that could have been flagged and addressed by a simple code review. The HYDN automated audit process which includes model checkers, fuzzers, and theorem provers analyses the smart contract for many of these basic coding mistakes. Once the automated audit has taken place, HYDN then performs a manual review of the code to gain familiarity with the contracts.

Business Logic Errors

HYDN reviews the platform or projects design documents before analyzing the code to ensure that the team has a deep understanding of the business logic and goals. Following this, HYDN reviews the smart contracts to ensure that the contract logic is in Line with the expected functionality.

HYDN also analyses the code for inconsistencies, flaws, or vulnerabilities which could impact business logic such as Tokenomics errors, arbitrage opportunities, and share pricing.

Complex Integration Risks

Several high-profile exploits have been the result of not any bug within the

contract itself, but rather an unintended consequence of its interaction with the broader DeFi ecosystem.

We perform a meticulous review of all of the contract's possible external interactions, and summarize the associated risks; for example: flash loan attacks, oracle price manipulation, MEV/sandwich attacks, etc.

Code Maturity

HYDN reviews the smart contracts for potential improvements in the codebase. These improvements ensure that the code follows industry best practises and guideLines, or code quality standards. Alongside this, HYDN makes suggestions for code optimization items such as gas optimization, upgradeability weaknesses, centralization risks, and more.

Impact Rankings

Once HYDN has completed the security assessment, each issue is assigned an impact rating. This impact rating is based upon both the severity and likelihood of the issue occurring. Each security assessment is different and the business needs and goals, such as project timeLines, and client threat modelling. are taken into account when the impact rankings are created.

HYDN assigns the following impact rankings: **Critical, High, Medium, Low** (listed by severity).

Disclaimer

This HYDN security assessment does not provide any warranties on finding all possible issues within the scope and the evaluation results do not guarantee the absence of any issues. **HYDN cannot make any guarantees on any further code which is added or altered after the security assessment has taken place.** As a single security assessment can never be considered fully comprehensive, HYDN always recommends multiple independent assessments paired with a bug bounty program. This security assessment report should not be considered as financial or investment advice.

Summary

Executive Summary

HYDN audited the scoped contracts listed below and discovered 6 findings. One finding of critical severity and one of high severity was found. Of the further 4 findings, one was of medium impact, and three were of low impact.

Project Summary

Dancing Seahorse	Dancing Seahorse
Platform	Ethereum
Deployment Address	0×8F86C6fa0F2Aa138D256ed7Ab5C64181e60a6f89
Language	Solidity
Scope	Mint Pass 1155 Smart Contract DHSH.sol Deployment and Utility Scripts
Repository	https://github.com/Cradleblock/dch
Initial Commits	97d0c01280e0ab3ca49dcf8fb67051425b4945d8
Latest Commits	f545424e96d5fb1595d066340f70c037ccf4a185

Audit Summary

Delivery Date	03-08-2022
Audit Method	Static Automated Analysis + Manual Review

List of Checks

- Overflows & Underflows
- Reentrancy Vulnerability
- Replay Vulnerability
- Short Address Vulnerability
- Kill-Switch / Self-Destruct
- Denial of Service
- Arithmetic Accuracy
- Uninitialized Storage Pointers
- Time Based Vulnerability
- Loop Vulnerability
- Ownership Takeover
- Access Control
- Oracle Security
- Deployment process
- Unsafe Type Inference
- Visibility Level Explicity
- Centralization Risks
- Deprecated Issuage
- Standard Compliance
- Code Stability
- Best Practices

Breakdown of Findings

Impact Level	Found	Description
Critical Severity Issues	**1**	**Whitelisted user can mint any token ID**
High Severity Issues	**2**	**Media files may become corrupt and unexpected behaviours may occur**
Medium Severity Issues	**2**	**Unnecessary gas costs may be incurred**
Low Severity Issues	**3**	**Centralization risks and potential compiler issues**
Total Issues	**8**	

Detailed Findings

1.1 Weak Whitelist Minting Validation [Critical]

- Target: DHSH.sol
- Severity: Critical
- Impact: Critical

1.1.1 Description

On Line 44: The Merkle Tree Leaf is only based on the caller address and other parameters are not verified.

```
44      function makeNft(
45          address _account,
46          uint256 _id,
47          uint256 _amount,
48          bytes32[] memory _proof
49
50      ) public {
51          uint256 nftbalance = balanceOf(_account, _id);
52          require(nftbalance == 0, "You have exceeding number of mints");
53          require(
54              isWhitelisted(_proof,keccak256(abi.encodePacked(msg.sender))),
55              "Not on Whitelist"
56          );
57           require(_amount == maxMintAmount, "You only have one pass allocated to you");
58          _mint(_account, _id, _amount, "");
59          emit Claimedmintpass(_id, _amount, msg.sender);
60      }
```

1.1.2 Impact

The impact of this issue is that a whitelisted user can mint any token ID he wants and there is no validation required.

1.1.3 Recommendations

To remediate this issue: The Merkle Tree Leafs must include all of the MakeNft() method parameters to ensure them.

1.1.4 Remediation Information

The Dancing Seahorse developers have acknowledged this vulnerability and decided not to take actions to address it. The Dancing Seahorse development team is happy to assume the risks associated with it.

Disclaimer: HYDN accept no responsibility associated with any nefarious or undesirable activity associated with this vulnerability.

1.2 Inheritance Linearization Invalid [High]

- Target: DHSH.sol
- Severity: High
- Impact: High

1.2.1 Description

On Line 93: The override inheritance call order is not defined as expected.

```
86      function _beforeTokenTransfer(
87          address operator,
88          address from,
89          address to,
90          uint256[] memory ids,
91          uint256[] memory amounts,
92          bytes memory data
93      ) internal override(ERC1155, ERC1155Supply) whenNotPaused {
94          super._beforeTokenTransfer(operator, from, to, ids, amounts, data);
95      }
96  }
```

1.2.2 Impact

The impact of this is that the inheritance call order will not work as anticipated, and the execution order may lead to unexpected behaviors.

1.2.3 Recommendations

Inheritance must be ordered from the top level to the low level. For reference please see:
https://docs.soliditylang.org/en/v0.8.15/contracts.html#multiple-inheritance-and-Linearization

1.2.4 Remediation Completed

Issue has been successfully remediated in the f545424e commit version.

1.3 Token URI External Reference Immutable [High]

- Target: DHSH.sol
- Severity: High
- Impact: High

1.3.1 Description

On Line 19: The token URI base URL refers to a centralized http service, but the value of the URL is immutable.

```
18      constructor(bytes32 _root)
19          ERC1155(
20              "https://nftgen.s3.eu-west-2.amazonaws.com/testcheck/meta/{id}.json"
21          )
22      {
23          root = _root;
24      }
```

1.3.2 Impact

If the centralized http service becomes unavailable, the metadata will not be available anymore. This would result in the media asset files becoming corrupt.

1.3.3 Recommendations

There are two options to remediate this issue: if you wish to keep the value as immutable then you should use an immutable service such as IPFS.

The second option is to keep the centralized service, but the contract owner would need to have the ability to upgrade the value in case the service is no longer available.

1.3.4 Remediation Completed

Issue has been successfully remediated in the f545424e commit version.

1.4 Unnecessary Parameter [Medium]

- Target: DHSH.sol
- Severity: Medium
- Impact: Medium

1.4.1 Description

On Line 47: The makeNft (as minting function) requires an unless amount parameter. Following this, Line 57 ensures that the amount is always equal to the maxMintAmount, as maxMintAmount is a constant and is equal to 1.

```
44      function makeNft(
45          address _account,
46          uint256 _id,
47          uint256 _amount,
48          bytes32[] memory _proof
49
50      ) public {
51          uint256 nftbalance = balanceOf(_account, _id);
52          require(nftbalance == 0, "You have exceeding number of mints");
53          require(
54              isWhitelisted(_proof,keccak256(abi.encodePacked(msg.sender))),
55              "Not on Whitelist"
56          );
57           require(_amount == maxMintAmount, "You only have one pass allocated to you");
58          _mint(_account, _id, _amount, "");
59          emit Claimedmintpass(_id, _amount, msg.sender);
60      }
```

1.4.2 Impact

Unused and/or useless function parameters must be removed. If they are left in it leads to extra gas costs during both deployment and minting.

1.4.3 Recommendation

To remediate this issue: The amount parameter should be removed.

1.4.3 Remediation Completed

Issue has been successfully remediated in the f545424e commit version.

1.5 Unused Slot Storage [Medium]

- Target: DHSH.sol
- Severity: Medium
- Impact: Medium

1.5.1 Description

There are unused variables being used in the contract. Here on Line 15 the saleActive variable is not used.

```
11  contract Mintpass is ERC1155, Ownable, Pausable, ERC1155Supply, ERC1155Burnable {
12      bytes32 public root;
13      uint256 public maxMintAmount = 1;
14      string public name = "Dancing Seahorse Mint Pass";
15      bool public saleActive = false;
16      string public symbol = "DSCMP";
```

1.5.2 Impact

A smart contract must be designed efficiently. Unused storage reflects the development quality and weakness and leads to extra deployment gas costs.

1.5.3 Recommendations

To remediate this issue: Remove the unused variable.

1.5.4 Remediation Completed

Issue has been successfully remediated in the f545424e commit version.

1.6 Centralization Risks [Low]

- Target: DHSH.sol
- Severity: Low
- Impact: Low

1.6.1 Description

There are currently a number of centralization risks within the contract.

The contract owner can do the following:

- Line 32: Change the Merkel root at any time. This will result in different a whitelist
- Line 36 and Line 40: Pausing and unpausing the contract will stop any transfers
- Line 70: Burn any token owned by any user

1.6.2 Recommendations

To remediate this issue: Review all admin controls to ensure that they are completely necessary and remove any which are not.

1.6.3 Remediation Completed

The Dancing Seahorse team have acknowledged this point and decided that no further action needs to be taken.

Disclaimer: HYDN accept no responsibility associated with any nefarious or undesirable activity associated with this risk.

1.7 Gas Optimization Constant Variable Slot [Low]

- Target: DHSH.sol
- Severity: Low
- Impact: Low

1.7.1 Description

Immutable variables should be restricted to constant, as it will lead to extra getter and setter creation gas costs that are unnecessary.

```
11  contract Mintpass is ERC1155, Ownable, Pausable, ERC1155Supply, ERC1155Burnable {
12      bytes32 public root;
13      uint256 public maxMintAmount = 1;
14      string public name = "Dancing Seahorse Mint Pass";
15      bool public saleActive = false;
16      string public symbol = "DSCMP";
```

1.7.2 Impact

This will result in unnecessary gas costs for the getter and setter creation.

1.7.3 Recommendations

To remediate this issue: Flag unchanged variable as constant:

- Line 13 maxMintAmount
- Line 14 name
- Line 16 symbol

1.7.4 Remediation Completed

Issue has been successfully remediated in the f545424e commit version.

1.8 Outdated Solidity Versions Used [Low]

- Target: DHSH.sol
- Severity: Low
- Impact: Low

1.8.1 Description

An outdated Solidity version has been used on contracts.

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >=0.7.0 <0.9.0;
```

1.8.2 Impact

Compiler issues can be caused by using outdated versions of Solidity.

1.8.3 Recommendations

To remediate this issue: Replace Solidity compiler version with 0.8.15.

1.8.4 Remediation Completed

Issue has been successfully remediated in the f545424e commit version.

Remediations Summary

Remediated

- Inheritance must be from the top level to the low level 1.2
- Allow the value to be updated by the contract owner 1.3
- Unused and/or useless function parameters must be removed 1.4
- Remove unused saleActive variable to avoid unnecessary gas costs 1.5
- Flag unchanged variables as constant to avoid unnecessary gas costs 1.7
- Ensure that the latest version of Solidity is always being used 1.8

No Action Taken

- Merkle tree leafs must include all of the MakeNft() method parameters 1.1
- Review all admin controls to ensure they are completely necessary 1.6

HYDN Seal

Each HYDN Seal is an ERC-1155 token that contains key metadata about the completed audit, including the hash of the contract bytecode, date, and link to the report.

As each HYDN Seal is stored on-chain, it allows for real-time monitoring of a project's status. The image on the Seal is served dynamically. We will notify you if the contract changes so you can validate if it has been changed by nefarious activity.

The HYDN Seal token id assigned to the contracts covered by this report: *10000001*

The HYDN Guardian token factory contract can be found at: 0×5AFE007958D169abE98299bf645b4C5890901dCB on the same network (ethereum mainnet, chain id: 1)