Smart Contract Security Assessment

September 12, 2022

Project Name:

Dancing Seahorse - Legendary

Prepared for:

Jose Soeiro, Cradleblock

Prepared by:

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 analysing 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 summarise 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 of Findings

Executive Summary

HYDN audited the scoped contracts listed below and discovered 12 findings. Four findings of critical severity and one of high severity was found. Of the further 7 findings, 5 were of medium impact, and 2 were of low impact. HYDN has also validated that funds can only be transferred out of the contract to the wallet address that is set in the contract.

Project Summary

Platform	Ethereum
Contract type	721
Language	Solidity
Scope	legendary.sol Deployment and Utility Scripts
Repository	https://github.com/Cradleblock/dch
Initial Commits	4da10152985b5f1a5aed75ee4543fa82769a67b5
Latest Commits	665d01be1deb6e61cee8d6dcf34eef27bb55cedc

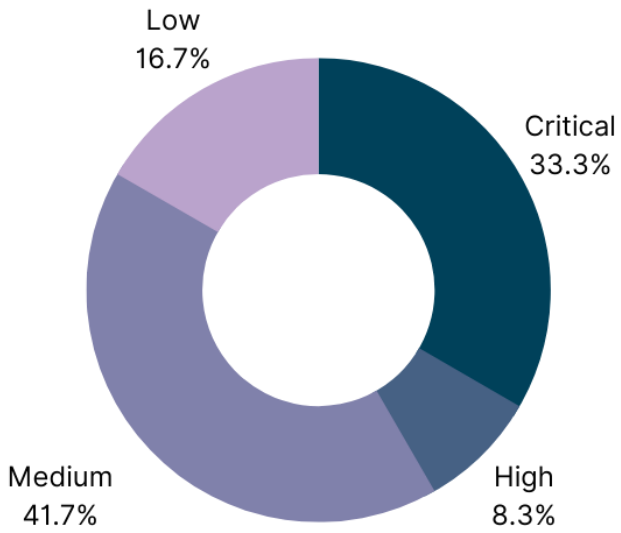
Audit Summary

Delivery Date	13-06-2022
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



Total Issues - 12

- Critical - 4
- High - 1
- Medium 5
- Low - 2

ID	Title	Severity	Status
1	Incorrect withdraw logic	Critical	Resolved
2	Re-entrancy vulnerability	Critical	Resolved
3	Max supply over pass	Critical	Resolved
4	Limits supply over pass	Critical	Resolved
5	Unexpected logic	High	Acknowledged
6	Unused logic	Medium	Resolved
7	Unnecessary math control	Medium	Resolved
8	Uninitialized storage	Medium	Resolved
9	Redundant duplicate functions	Medium	Resolved
10	Floating mint cost control	Medium	Resolved

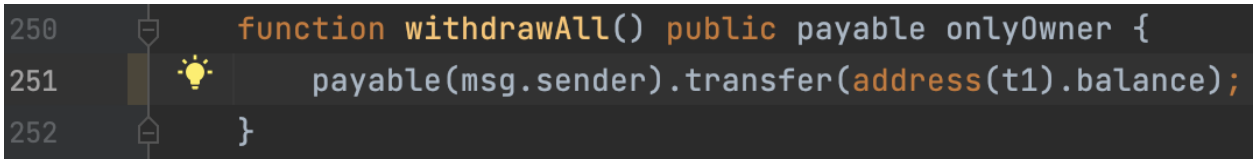
11	Gas optimization - incorrect function visibility	Low	Acknowledged
12	Outdated Solidity compiler version used	Low	Resolved

Detailed Findings

1 Incorrect withdraw logic [Critical]

1.1 Description

The logic under the withdrawal of the funds collected is incorrect: it transfers to the owner an amount equal to the balance hold by an arbitrary address (named as t1).



1.2 Impact

This incorrect logic may lead to loss of funds.

1.3 Recommendations

The withdrawal amount must be equal to the current contract balance holding.

1.4 Remediation Completed

Successfully remediated. Changes have been made to the withdrawAll()function and a function to specify the amount to be withdrawn as per withdrawFixed() has been included. HYDN has also validated that funds can only be transferred out of the contract to the wallet address that is set in the contract.

2 Re-entrancy vulnerability [Critical]

2.1 Description

A re-entrancy vulnerability can occur as the call to _safeMint allows the receiver to perform a re-entrancy attack by re-calling the function.

```
70      for (uint256 i = 1; i <= _mintAmount; i++) {
71          _safeMint(msg.sender, supply + i);
72      }
73      addPrimintCount(_mintAmount);
```

```
92      for (uint256 i = 1; i <= _mintAmount; i++) {
93          _safeMint(msg.sender, supply + i);
94      }
95
96      addnftCount(_mintAmount);
```

```
146     addBrandCounts(_mintAmount);
147     for (uint256 i = 1; i <= _mintAmount; i++) {
148         _safeMint(msg.sender, supply + i);
149     }
150
151     addBrandCount(count.add(_mintAmount));
```

2.2 Impact

This issue can lead to state manipulation and execution order manipulation.

2.3 Recommendations

Reorder mutable operations such as:

- premintNFT: addPrimintCount before calling _safeMint
- makeNft: addnftCount before calling _safeMint

- brandNft: addBrandCount before calling _safeMint

2.4 Remediation Completed

Successfully remediated to reorder mutable operations.

3 Max supply over pass [Critical]

3.1 Description

With the re-entrancy attack detailed in issue 2 the quantity of tokens can then exceed the max supply control.

```
60    function premintNFT(uint256 _mintAmount) public payable {
61        uint256 supply = totalSupply();
62        IERC1155 token = IERC1155(tokenAddress);
63        require(token.balanceOf(msg.sender, 3) >= 1, "No Mintpass found");
64        require(salePremActive, "Sale is not active");
65        require(supply <= maxSupply, "Sold out");
```

```
77    function makeNft(uint256 _mintAmount) public payable {
78        uint256 supply = totalSupply();
79        IERC1155 token = IERC1155(tokenAddress);
80        uint256 brand = token.balanceOf(msg.sender, 1);
81        uint256 artist = token.balanceOf(msg.sender, 2);
82        uint256 legendaryplus = token.balanceOf(msg.sender,
83        uint256 legendary = token.balanceOf(msg.sender, 4);
84
85        require(saleActive, "Sale is not active");
86        require(supply <= maxSupply, "Sold out");
```

```
99    function giftNft(uint256 _mintAmount, address _address) public onlyOwner {
100       uint256 supply = totalSupply();
101       require(supply <= maxSupply, "Sold out");
```

```
109   function OTC(uint256 _mintAmount, address _address) public onlyOwner {
110       uint256 supply = totalSupply();
111       require(supply <= maxSupply, "Sold out");
```

```
119   function vault(uint256 _mintAmount, address _address) public onlyOwner {
120       uint256 supply = totalSupply();
121       require(supply <= maxSupply, "Sold out");
```

```
132     function brandNft(uint256 _mintAmount) public payable {
133         uint256 supply = totalSupply();
134         IERC1155 token = IERC1155(tokenAddress);
135         require(token.balanceOf(msg.sender, 1) >= 1, "No Mintpass found");
136         require(salePremActive, "Sale is not active");
137         require(supply <= maxSupply, "Sold out");
```

3.2 Impact

This issue would result in more tokens than expected being minted.
Function impacted:

- premintNFT
- makeNft
- giftNft
- OTC
- vault
- brandNft

3.3 Recommendations

Max supply control must include current supply + the current mintAmount. Additionally because of the _safeMint hook to strictly avoid abuse of the supply use a non-re-entrance pattern.

3.4 Remediation Completed

Successfully remediated. Based on the feedback provided, Dancing Seahorse has added OpenZeppelin ReentrancyGuard library to all sale functions to avoid execution interruption. Note that the issue still persists but the global lock covers it.

4 Limits supply over pass [Critical]

4.1 Description

Similar to the max supply control, the quantity of tokens can exceed the expected control.

```
67          require(premintSaleCounter < premintlimit,"Premint Sale Sold out");
```

```
90          require(nftSaleCounter < makenftlimit,"Premint Sale Sold out");
```

```
139          require(brandSaleCounter < brandlimit,"Brand Sale Sold out");
```

4.2 Impact

This will result in more tokens than expected being minted.

Functions impacted:

- premintNFT - premintlimit over pass
- makeNft - makenftlimit over pass
- brandNft - brandlimit over pass

4.3 Recommendations

premintlimit and makenftlimit control must include current supply + the current mintAmount

4.4 Remediation Completed

Successfully remediated to include control on the current supply and mintAmount.

5 Unexpected logic [High]

5.1 Description

In this issue, the updateMeta function holds unexpected logic. This allows the owner to push infinitely new items in the array and arbitrary bytecode.

```
178    function updateMeta(uint256 _tokenid,string calldata  data) public {
179
180            require(ownerOf(_tokenid) == msg.sender, "You are not the owner of this NFT");
181            Metadata.push(
182                metaData({
183                    tokenid: _tokenid,
184                    data: data,
185                    addr: msg.sender
186                })
187            );
```

5.2 Impact

This can lead to extra gas costs.

5.3 Recommendations

Remove unused function and associate storage.

5.4 Remediation Completed

Issue acknowledged by Dancing Seahorse, but taking into account the business logic of the project, no further action has been taken. **Disclaimer: HYDN accept no responsibility associated with any nefarious or undesirable activity associated with this vulnerability.**

Dancing Seahorse states that, "*The main requirement is that we can update the metadata stored on the AWS S3 bucket for a specific NFT. We can go directly to AWS to update the files as it isn't an immutable database, this fact has been included in the Terms of Use agreement on the website and directly with the partners like Artists and Brands.*

On our site the process will be that the holder of a NFT will give consent for the metadata to be updated at which time we will push new metadata to AWS and call the updateMeta function to record this action."

6 Unused logic [Medium]

6.1 Description

Here the pause and unpause logic are not used.

```
234     function pause(bool _state) public onlyOwner {
235         paused = _state;
236     }
```

6.2 Impact

This will lead to extra gas costs.

6.3 Recommendations

Remove unused function and storage variable associate.

6.4 Remediation Completed

Successfully remediated and unused functions have been removed.

7 Unnecessary math control [Medium]

7.1 Description

Since 0.8 solc version overflow and underflow throws errors and the libraries such as SafeMath are not necessary.

7.2 Impact

This will lead to extra gas costs.

7.3 Recommendations

Remove SafeMath library and usage.

7.4 Remediation Completed

Successfully remediated and SafeMath has been removed.

8 Uninitialized storage [Medium]

8.1 Description

Here this essential variable is not initialised.

```
12        address tokenAddress;
```

8.2 Impact

This issue leads to a state issue if the contract is called before these variables are properly defined.

8.3 Recommendations

Initialise the constructor value for tokenAddress.

8.4 Remediation Completed

Successfully remediated. Dancing Seahorse has added a check to validate whether the token address is zero.

9 Redundant duplicate functions [Medium]

9.1 Description

As you can see below, multiple functions contain the exact same logic.

```
99      function giftNft(uint256 _mintAmount, address _address) public onlyOwner {
100         uint256 supply = totalSupply();
101         require(supply <= maxSupply, "Sold out");
102         require(_mintAmount > 0, "need to mint at least 1 NFT");
103
104         for (uint256 i = 1; i <= _mintAmount; i++) {
105             _safeMint(_address, supply + i);
106         }
107     }
108
109     function OTC(uint256 _mintAmount, address _address) public onlyOwner {
110         uint256 supply = totalSupply();
111         require(supply <= maxSupply, "Sold out");
112         require(_mintAmount > 0, "need to mint at least 1 NFT");
113
114         for (uint256 i = 1; i <= _mintAmount; i++) {
115             _safeMint(_address, supply + i);
116         }
117     }
118
119     function vault(uint256 _mintAmount, address _address) public onlyOwner {
120         uint256 supply = totalSupply();
121         require(supply <= maxSupply, "Sold out");
122         require(_mintAmount > 0, "need to mint at least 1 NFT");
123
124         for (uint256 i = 1; i <= _mintAmount; i++) {
125             _safeMint(_address, supply + i);
126         }
```

9.2 Impact

This will lead to extra gas costs.

Duplicate functions:

- giftNft
- OTC

- vault

9.3 Recommendations

Remove duplicate functions giftNft, OTC, and vault as they embark on the exact same body function they must factorise or simply reduce to one copy.

9.4 Remediation Completed

Successfully remediated. Dancing Seahorse has refactored the logic to keep one unique function to handle this logic.

10 Floating mint cost control [Medium]

10.1 Description

In this issue, you can see that the mint price is not fixed.

```
68          require(msg.value >= cost * _mintAmount, "insufficient funds");
```

```
89          require(msg.value >= cost * _mintAmount, "insufficient funds");
```

10.2 Impact

This means that the mint cost may exceed the expected mint price for the minter.

Functions impacted:

- premintNFT
- makeNft

10.3 Recommendations

Mint price must be fixed.

10.4 Remediation Completed

Successfully remediated and has now been updated so that it is equal to the cost * _mintAmount.

11 Gas optimization - incorrect function visibility [Low]

11.1 Description

Functions that are not called by the contract itself must be flagged as external.

11.2 Impact

This will lead to extra gas costs.

11.3 Recommendations

Change public to external visibility for non-internally called functions.

11.4 Remediation Completed

Issue has been acknowledged by Dancing Seahorse, but no further action has been deemed necessary by the Dancing Seahorse team.

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

12 Outdated Solidity compiler version used and not fixed version allowed [Low]

12.1 Description

An outdated Solidity version has been used on contracts.
No fixed solidity compiler version is allowed.

12.2 Impact

Compiler issues can be caused by using outdated versions of Solidity.
Using different compiler versions will produce arbitrary outputs and can lead to unexpected issues.

12.3 Recommendations

To remediate this issue: Replace Solidity compiler version with 0.8.16.

12.4 Remediation Completed

Successfully remediated and Solidity compiler version has been replaced with 0.8.16.

Remediations Summary

Successfully remediated

- 1 Incorrect withdraw logic [Critical]
- 2 Re-entrancy vulnerability [Critical]
- 3 Max supply over pass [Critical]
- 4 Limits supply over pass [Critical]
- 6 Unused logic [Medium]
- 7 Unnecessary math control [Medium]
- 8 Uninitialized storage[Medium]
- 9 Redundant duplicate functions [Medium]
- 10 Floating mint cost control [Medium]
- 12 Outdated Solidity compiler version used and not fixed version allowed [Low]

Acknowledged but no further action taken

- 5 Unexpected logic [High]
- 11 Gas optimization - incorrect function visibility [Low]

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:
10000002

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