HYDN

Octane

Smart Contract Security Assessment

July 18, 2022

Prepared for:

Pedro J Rodriguez

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 Octane

Octane is an NFT collection designed by the world renowned automobile artist, Manu Campa. Manu Campa's work can be found worldwide, having worked with the likes of Coca Cola, PlayStation, Kia, Dunlop and many more.

The Octane community is a step forward for the artist, bringing Manu's art to the digital world, where motor enthusiasts and digital art collectors can get access to Manu’s work in digital form. Launching with an initial Octane - Genesis collection of 80 handpainted NFTs, the Octane collection aims to build a blockchain art project and community with multiple collections, perks and exciting surprises for owners and collectors.

The audit covers the following scope:

- NFT Factory
- Sales Contract
- Royalty Payments Splitter

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

Octane	Octane
Platform	Ethereum
Language	Solidity
Scope	OctaneGenesis.sol OctaneGenesisSale.sol RoyaltyWallet.sol Deployment and Utility Scripts
Repository	https://gitlab.com/block360/manu-campa/smartcontracts
Initial Commits	bcf2f0d4f2db59c1732b811c3262b041c4014345
Latest Commits	5d4fbf7e680122ec5f039e8c5613904f1566bc93

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

Impact Level	Severity	Description
Critical Severity Issues	**1**	**Funds can be lost or locked**
High Severity Issues	**1**	**Discontinuation of service**
Medium Severity Issues	**1**	**Service outage and unexpected behaviours**
Low Severity Issues	**3**	**Minor outages and theoretical outage**
Total Issues	**6**	

Detailed Findings

1.1 Royalty Wallet Contract Only Supports Native ETH Currency [Critical]

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

1.1.1 Description

The RoyaltyWallet.sol contract only supports ETH. Therefore, if royalties are sent in the form of any other ERC-20 tokens to the contract then there is no way to withdraw them and these funds will be lost.

```
338     function release(address payable account) public virtual {
339         require(_shares[account] > 0, "RoyaltyWallet: account has no shares");
340
341         uint256 totalReceived = address(this).balance + totalReleased();
342         uint256 payment = _pendingPayment(account, totalReceived, released(account));
343
344         require(payment != 0, "RoyaltyWallet: account is not due payment");
345
346         _released[account] += payment;
347         _totalReleased += payment;
348
349         Address.sendValue(account, payment);
350         emit PaymentReleased(account, payment);
351     }
```

1.1.2 Impact

If any ERC-20 tokens other than ETH are sent to this contract then there will be no way to withdraw them and funds will be lost.

For example, if an NFT is purchased using USDC on OpenSea, then the royalties will also be paid in USDC. With the current contract, these USDC royalty

payments would be sent to the royalties contract following the EIP-2981 standard, these funds will be inaccessible and lost.

1.1.3 Recommendations

To remediate this issue: Add support for further ERC-20 tokens to ensure that royalty payments can be accessed.

1.1.4 Remediation Completed

Issue remediated and validated in commit 5d4fbf.

1.2 Weak Quantity Peer Phase Control [High]

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

1.2.1 Description

OctaneGenesisSale.sol l86 tokens #1 and #2 are restricted by quantity (maximum of 3 which matches the mint quantity).

```
76      function _preValidatePurchase(uint256 NFTType, bytes32[] memory proof)
77          internal
78      {
79          if (applyMerkle()) {
80              require(isWhitelisted(proof), "Not whitelisted For presale");
81          }
82
83          require(alreadyMinted[msg.sender] < maxMint, "Max buy limit reached");
84          require(msg.value == getNFTPrice(NFTType), "Amount is not correct");
85
86          if ((NFTType == 0 || NFTType == 1) && sold[NFTType] == 2) {
87              revert("You can buy this NFT after pre sale is over");
88          }
89      }
```

1.2.2 Impact

All NFTs could be sold during Phase 1, and none will remain available for Phase 2.

1.2.3 Recommendations

To remediate this issue: Build new logic where depending on the current phase the contract allows different maximum numbers to be sold.

1.2.4 Remediation Completed

Following discussions with the Octane team, business logic has been clarified and Phase 2 items will be sold on an external marketplace.

1.3 Maximum Mint Per Wallet Weak Constraints [Medium]

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

1.3.1 Description

OctaneGenesisSale.sol l83 maxMint per wallet can easily be bypassed by a bad actor using multiple wallets.

```
76      function _preValidatePurchase(uint256 NFTType, bytes32[] memory proof)
77          internal
78      {
79          if (applyMerkle()) {
80              require(isWhitelisted(proof), "Not whitelisted For presale");
81          }
82
83          require(alreadyMinted[msg.sender] < maxMint, "Max buy limit reached");
84          require(msg.value == getNFTPrice(NFTType), "Amount is not correct");
85
86          if ((NFTType == 0 || NFTType == 1) && sold[NFTType] == 2) {
87              revert("You can buy this NFT after pre sale is over");
88          }
89      }
```

1.3.2 Impact

A bad actor using multiple wallets can easily bypass the maxMint and mint as many NFTs as they want.

1.3.3 Recommendations

To remediate this issue: Remove easy-to-by-pass check.

1.3.4 Remediation Completed

Following discussions with the Octane team, this issue has been acknowledged and the risks have been accepted with no further action needing to be taken.

1.4 Centralization Risks [Low]

At any time the contract owner is allowed to do the following:

1.4.1 OctaneGenesis.sol

- Pause and unpause the contract, which will stop all functions (transfer, allowance, etc...) from working
- setURI, which will change the media asset attached to the token
- setDefaultRoyalty, which will change the royalties default percentage
- setTokenRoyalty, which will change the royalty for a specific token

1.4.2 OctaneGenesisSale.sol

- Pause and unpause contract, which will stop the sale workflow
- setMerkleRoot, which will change the whitelisted addresses
- setOpeningTime, which will change the pre-sale deadline

1.4.3 Recommendations

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

1.4.4 Remediation Completed

Following discussions with the Octane team, this issue has been acknowledged and the risks have been accepted with no further action needing to be taken.

1.5 Unnecessary nonReentrant [Low]

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

1.5.1 Description

OctaneGenesisSale.sol l54 nonReentrant mutex lock is unnecessary if the payment splitter is trusted (as it should be), and does not serve a purpose.

```
54      function buy(uint256 NFTType, bytes32[] memory proof)
55          external
56          payable
57          nonReentrant
58          whenNotPaused
59          onlyWhileOpen
60      {
61          _preValidatePurchase(NFTType, proof);
62
63          paymentSplitter.sendValue(msg.value);
64
65          sold[NFTType] += 1;
66          alreadyMinted[msg.sender] += 1;
67          octaneGenesis.safeTransferFrom(
68              tokenWallet,
69              msg.sender,
70              NFTType,
71              1,
72              "0x0"
73          );
74      }
```

1.5.2 Impact

Using nonReentrant mutex lock will result in unnecessary extra gas costs.

1.5.3 Recommendations

To remediate this issue: Remove nonReentrant modifier.

1.5.4 Remediation Completed

Following discussions with the Octane team, this issue has been acknowledged and the risks have been accepted with no further action needing to be taken.

1.6 Royalty Wallet Contract Uses Hardcoded Libraries [Low]

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

1.6.1 Description

RoyaltyWallet.sol does not rely properly on OpenZeppelin libraries. Always use audited, battle-tested third party libraries whenever possible as they are less likely to contain bugs.

1.6.2 Impact

When using custom code rather than audited libraries, potential bugs are more likely to occur and cause issues.

1.6.3 Recommendations

To remediate this issue: Properly import OpenZeppelin dependencies.

1.6.4 Remediation Completed

Issue remediated and validated in commit 5d4fbf.

1.7 Outdated Solidity Versions Used [Low]

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

1.7.1 Description

An outdated Solidity version has been used on contracts.

1.7.2 Impact

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

1.7.3 Recommendations

To remediate this issue: Replace Solidity compiler version with 0.8.14.

1.7.4 Remediation Completed

Issue remediated and validated in commit 5d4fbf.

Recommendations Summary

Critical and High Importance

- Add support for further ERC-20 tokens to ensure that funds sent in anything other than ETH are not lost [1.1]
- Build new logic for sale phases to ensure that enough items remain for Phase 2 of sales process [1.2]Aud
- Remove the easy-to-bypass-checks to prevent bad actors with multiple wallets minting more than the allowed amount [1.3]

Medium and Low Importance

- Review all admin controls to ensure they are completely necessary [1.4]
- Remove nonReentrant modifier as payment splitter contract is trusted [1.5]
- Always use audited, battle-tested third party libraries whenever possible as they are less likely to contain bugs [1.6]
- Ensure that the latest version of Solidity is always being used [1.7]