Hiroshima Dragonfly

AUDIT REPORT

Version 1.0.0

Serial No. 20220516000012021

Presented by Fairyproof

May 16, 2022

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Hiroshima Dragonfly project, at the request of the Hiroshima Dragonfly team.

Audit Start Time:

May 13, 2022

Audit End Time:

May 16, 2022

Project Token's Name:

Hiroshima Dragonfly

Audited Code's Github Repository:

https://github.com/iqbalhanif313/hiroshima-dragonfly-nft-sc

Audited Code's Github Commit Number When Audit Started:

74948cef4408728ff455e97a9c072e61babbae57

Audited Code's Github Commit Number When Audit Ended:

5ed58f73a8e33248a8b172bad89eb9d868487632

Audited Source Files:

The calculated SHA-256 values for the audited files when the audit was done are as follows:

```
HDY721.sol     :
0x180d83b72338b93e318b81436024d10a62973c9a7bc9c36625ffc7c49ffdcf0c
Marketplace.sol:
0xbb475fc48c784da8516c1d9801e2330e6d1ac7b51a4b53de168792e8b1ec8252
```

The goal of this audit is to review Dragonfly's solidity implementation for its NFT issuance and sales functions, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Hiroshima Dragonfly team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.

3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

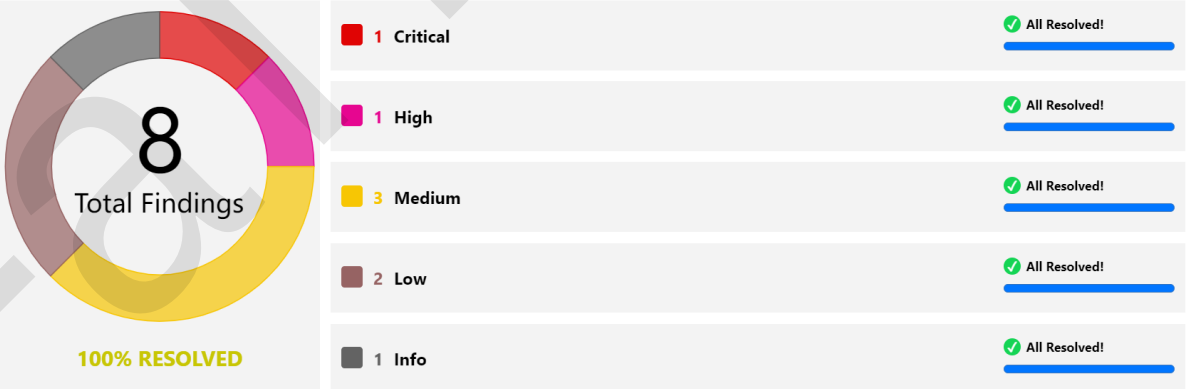
For this audit, we used the following sources of truth about how the NFT issuance and sales functions should work:

https://hiroshimadragonflies.com/

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Hiroshima Dragonfly team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2022051600012021	Fairyproof Security Team	May 13, 2022 - May 16, 2022	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, one issue of critical-severity, one issue of high-severity, three issues of medium-severity, two issues of low-severity and one issue of informational-severity were uncovered. All the issues have been fixed by the team.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Major functions of audited code

The audited code mainly implements NFT issuance and sales functions. Token issuance and sales are executed by the admin.

The Hiroshima Dragonfly team charges fees on NFT sales.

04. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- Replay Attack
- Reordering Attack
- Miner's Advantage
- Rollback Attack
- DDoS Attack
- Transaction Ordering Attack
- Race Condition
- Access Control
- Integer Overflow/Underflow
- Timestamp Attack
- Gas Consumption
- Inappropriate Callback Function
- Function Visibility
- Implementation Vulnerability
- Uninitialized Storage Pointer

- Arithmetic Precision
- Tx.origin
- Fake Deposit
- Shadow Variable
- Design Vulnerability
- Token Issuance
- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration
- Code Improvement
- Misc

05. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

06. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.

We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by "owner".

We found some issues, for more details please refer to `FP-2`, `FP-3` and `FP-7` in "08. Issue descriptions".

- Token Issuance

We checked whether or not the contract files could mint tokens at will.

We found one issue, for more details please refer to `FP-1` in "08. Issue description".

- State Update

We checked some key state variables which should only be set at initialization.

We found issues, for more details please refer to `FP-4`, `FP-5` and `FP-6` in "08. Issue description".

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.

We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We found one issue, for more details please refer to `FP-8` in "08. Issue description".

07. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Missing Check for Sales Price	Implementation Vulnerability	Critical	✓ Fixed
FP-2	Missing Access Control for Price Setting	Access Control	High	✓ Fixed
FP-3	Missing Access Control for Issuance	Access Control	Medium	✓ Fixed
FP-4	Repeatable Initialization	Implementation Vulnerability	Medium	✓ Fixed
FP-5	Missing Check for NFT Status	Implementation Vulnerability	Medium	✓ Fixed
FP-6	Missing Validation for Initialization	Implementation Vulnerability	Low	✓ Fixed
FP-7	Redundant Modifier	Access Control	Low	✓ Fixed
FP-8	Missing External Access to Variables	Misc	Info	✓ Fixed

08. Issue descriptions

[FP-1] Missing Check for Sales Price

Critical ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

The `Buy` function in `Marketplace.sol` didn't verify whether or not a user's payment amount was equal to the price. Therefore a user could buy an NFT by paying zero fees

Recommendation:

Consider adding a require(msg.value == price) directive

Update: the Hiroshima Dragonfly team added `require(msg.value == price, "msg.value is not matched with price!");`.

Status:

It has been fixed by the Hiroshima Dragonfly team.

[FP-2] Missing Access Control for Price Setting

High ✓ Fixed

Issue/Risk: Access Control

Description:

In `HDY721.sol`, any one could call the `setPriceNFT` function to set an NFT's price. Its desired behavior was that only the `Marketplace` contract could call the function.

Recommendation:

Consider adding a variable to record the address of the `Marketplace` contract. Only that variable can call the function.

Update:

The Hiroshima Dragonfly team has adopted the recommendation.

Status:

It has been fixed by the Hiroshima Dragonfly team.

[FP-3] Missing Access Control for Issuance

Medium ✓ Fixed

Issue/Risk: Access Control

Description:

In `HDY721.sol`, the `mint` function didn't validate the caller. Therefore users that had verified signatures could bypass the `Marketplace` contract and directly call the `mint` function. Its desired behavior was that only the `Marketplace` contract could call the function.

Recommendation:

Consider adding a variable to record the address of the `Marketplace` contract and adding a check to ensure the caller is `Marketplace`.

Update:

The Hiroshima Dragonfly team has adopted the recommendation.

Status:

It has been fixed by the Hiroshima Dragonfly team.

[FP-4] Repeatable Initialization

Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `Marketplace.sol` the `Init` function didn't verify whether or not some states were initialized. Therefore these states could be initialized repeatedly.

Recommendation:

Consider adding a `require` directive to ensure the states are only initialized once.

Update:

The Hiroshima Dragonfly team added a `require(!Initialized, "Contract already initialized!");` directive.

Status:

It has been fixed by the Hiroshima Dragonfly team.

[FP-5] Missing Check for NFT Status

Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `Marketplace.sol`, the `buy` function didn't verify whether or not an NFT was for sale. Therefore a user could buy an NFT that was not listed for sale

Recommendation:

Consider adding a require directive to ensure an NFT is for sale.

Update:

The Hiroshima Dragonfly team added a `require(statusNFT[tokenID], "NFT is not listed!");` directive.

Status:

It has been fixed by the Hiroshima Dragonfly team.

[FP-6] Missing Validation for Initialization

Low ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `Marketplace.sol` both `setRoyaltyFee` and `updatePlatform` should make sure they were called after initialization was done otherwise the parameters that were initialized might be reset.

Recommendation:

Consider adding a `initializer` modifier to ensure these functions are called after initialization is done.

Update:

The Hiroshima Dragonfly team added an `initializer` modifier.

Status:

It has been fixed by the Hiroshima Dragonfly team.

[FP-7] Redundant Modifier

Low ✓ Fixed

Issue/Risk: Access Control

Description:

`updatePlatform` had two modifiers `onlyOwner` and `onlyAdmin`. These two modifiers had different access control. They were confusing.

Is this a desired behavior? Consider removing one of them

Recommendation:

Consider removing either of them.

Update:

The Hiroshima Dragonfly team kept `onlyAdmin` and removed `onlyOwner`.

Status:

It has been fixed by the Hiroshima Dragonfly team.

[FP-8] Missing External Access to Variables

Low ✓ Fixed

Issue/Risk: Misc

Description:

Some state variable were private such as `platformAddress` and `Initialized` which users couldn't read.

Recommendation:

Consider changing them to `public`

Update:

The Hiroshima Dragonfly team changed them.

Status:

It has been fixed by the Hiroshima Dragonfly team.

09. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider inheriting `HDY721` from `ERC721Enumerable` to allow users to list their own NFTs.

Update: Hiroshima Dragonfly team replied that if they use ERC721Enumerable it will make the gas fee expensive, and they didn't use any function that were inherited from ERC721Enumerable.

- Consider adding an event in the `updatePlatform` function.

Update: Done.

- Consider adding a check in the `updatePlatform` function to ensure `newAddress` is a non-zero address.

Update: Done.

- Consider using a lower case for all the initial letters of the function names such as changing `Mint` to `mint`.

Update: Done.

- Consider adding a `require(statusNFT[tokenID],"not selled")` directive in the `CancelSell` function.

Update: Done.

- Both the `sell` function and the `sellWithMint` function used the same parameter signatures : `msg.sender, tokenID, price, transactionID`. This would cause confusion and unexpected issues. Consider adding an additional variable to differentiate them and verifying the signatures with care.

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech