Valkyrie
Security Audit Report

PREPARED FOR:

BlockSafe

ARCADIA CONTACT INFO
Email: audits@arcadiamgroup.com

Telegram: https://t.me/thearcadiagroup

Revision history

Date	Reason	Commit
12/16/2023	Removal of Misidentified Issues and Addition of Customer Responses	

Table of Contents

Executive Summary

Introduction

Blocksafe engaged Arcadia to perform a security audit of their main smart contracts.

Review Team

1. Tuan “Anhnt” Nguyen - Security Researcher and Engineer
2. Joel Farris - Project Manager

Coverage

For this audit, we performed research, test coverage, investigation, and review of Valkyrie's main contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report. The following code repositories, files, and/or libraries are considered in scope for the review.

Contracts	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/ValkyrieTransferAgent/ValkyrieTransferAgent.sol	50	34	27	1	37
contracts/ValkyrieTransferAgent/IValkyrieTransferAgent.sol	8	5	3	1	5
contracts/ValkyrieExecutionAgent/ValkyrieExecutionAgent.sol	387	353	263	54	159
contracts/ValkyrieExecutionAgent/ValkyrieExecutionAgentERC1155.ignore.sol	564	511	333	144	217
contracts/ValkyrieExecutionAgent/IValkyrieExecutionAgent.sol	26	26	23	1	1
contracts/Library/ValkyrieOrders.sol	44	44	34	6	1
contracts/Library/Whitelist.sol	21	21	15	1	11
contracts/ValkyrieNonceRegistry/IValkyrieNonceRegistry.sol	44	28	18	4	13
contracts/ValkyrieNonceRegistry/ValkyrieNonceRegistry.sol	116	107	89	4	104
contracts/ValkyrieFeeRegistry/IValkyri	6	5	3	1	3

eFeeRegistry.sol					
contracts/ValkyrieFeeRegistry/ValkyrieFeeRegistry.sol	30	30	22	3	29
contracts/ValkyrieRoyaltyRegistry/IValkyrieRoyaltyRegistry.sol	10	5	3	1	3
contracts/ValkyrieRoyaltyRegistry/ValkyrieRoyaltyRegistry.sol	58	54	33	11	25

Methodology

Arcadia completed this security review using various methods, primarily consisting of dynamic and static analysis. This process included a line-by-line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

The followings are the steps we have performed while auditing the smart contracts:

- Investigating the project and its technical architecture overview through its documentation
- Understanding the overview of the smart contracts, the functions of the contracts, the inheritance, and how the contracts interface with each others thanks to the graph created by Solidity Visual Developer
- Manual smart contract audit:
    - Review the code to find any issue that could be exploited by known attacks listed by Consensys
    - Identifying which existing projects the smart contracts are built upon and what are the known vulnerabilities and remediations to the existing projects
    - Line-by-line manual review of the code to find any algorithmic and arithmetic related vulnerabilities compared to what should be done based on the project's documentation
    - Find any potential code that could be refactored to save gas
    - Run through the unit-tests and test-coverage if exists
- Static Analysis:
    - Scanning for vulnerabilities in the smart contracts using Static Code Analysis Software
    - Making a static analysis of the smart contracts using Slither
- Additional review: a follow-up review is done when the smart contracts have any new update. The follow-up is done by reviewing all changes compared to the audited commit revision and its impact to the existing source code and found issues.

Summary

There were **7** issues found, **1** of which were deemed to be ‘critical’, and **4** of which were rated as ‘high’.

Severity Rating	Number of Original Occurrences
CRITICAL	0
HIGH	3
MEDIUM	1
LOW	1
INFORMATIONAL	1

Findings

(VA-1) Lack of authorization setFeeBPS.

Status

Addressed

Risk Level

Severity: High, likelihood: Low

Code Segment

```
  function setFeeBPS(uint256 _feeBPS) external {
       require(_feeBPS <= 10000, "ValkyrieFeeRegistry: feeBPS cannot be greater
than 100%");
       feeBPS = _feeBPS;
   }
```

Description

FeeBPS is used across the market to collect to fee but anyone can set it without any permission.

Code location

```
contracts/ValkyrieFeeRegistry/ValkyrieFeeRegistry.sol
```

Recommendation

Use **onlyOwner** modifier.

Response

Checks will be added in a later version of the codebase

(VA-2) Using non-standard interface to check collection owner.

Status

Addressed

Risk Level

Severity: High, likelihood: High.

Code Segment

```
 function setCollectionRoyaltyBPS(address _collection, uint256 _collectionRoyaltyBPS)
external {
        // check if msg.sender is collection owner
        require(
            Ownable(_collection).owner() == msg.sender,
            "ValkyrieRoyaltyRegistry: msg.sender is not collection owner"
        );

        // set collectionRoyaltyBPS
        collectionRoyaltyBPS[_collection] = _collectionRoyaltyBPS;
    }
```

Description

Ownable(_collection).owner() does not adhere to the ERC721/ERC1155 standard. This method does not provide the appropriate means to validate the owner of an NFT collection.

Code location

```
contracts/ValkyrieRoyaltyRegistry/ValkyrieRoyaltyRegistry.sol
```

Recommendation

Request the owner of the collection to join the market by completing the whitelisting process.

Response

The style and failure case of this check is intended by the client.

(VA-3) Lack of validation for token amount received during token transfers into the contract.

Status

Addressed

Risk Level

Severity: Low, likelihood: Low

Code Segment

```
function _transferFund(address _token, address _from, address _to, uint256
_amount) internal {
      if (_token == address(0)) {
          payable(_to).transfer(_amount);
          return;
      }

      valkyrieTransferAgent.transferERC20(_token, _from, _to, _amount);
  }
```

Description

Many tokens apply taxes to user transfers, resulting in the received amount not matching the transfer parameter. This emphasizes the need to validate the received token amounts.

Code location

```
contracts/ValkyrieExecutionAgent/ValkyrieExecutionAgent.sol
```

Recommendation

Perform a secondary verification on the received token amount following token transfers.

Response

WETH and ETH are the only intended currencies to be supported, and as such this issue is not relevant to the current codebase.

(VA-4) Bypassing the royalty fee.

Status

Addressed

Risk Level

Severity: High, likelihood: High

Description

All execution code uses **royaltyAmount** as input parameter, so the order maker/taker can easily bypass the royalty fee by setting **royaltyAmount** zero.

Code location

```
contracts/ValkyrieExecutionAgent/ValkyrieExecutionAgent.sol
```

Recommendation

If **maxRoyaltyAmount** is zero, the code should disregard the **_order.royaltyAmoun** parameter. However, if **maxRoyaltyAmount** is larger than zero, maintain the current implementation code.

Response

This is intentional functionality as fees are intended to be optional.

(VA-5) Lacking onlyWhitelisted modifier.

Status

Addressed

Risk Level

Severity: Medium, likelihood: High

Code Segment

```
function batchTransferToken(
       address to,
       address[] calldata _collections,
       uint256[] calldata _tokenIds,
       uint256[] calldata _amount
   ) external nonReentrant {
       for (uint256 i = 0; i < _tokenIds.length; i++) {
           if (IERC721(_collections[i]).supportsInterface(0x80ac58cd)) {
               IERC721(_collections[i]).transferFrom(msg.sender, to, _tokenIds[i]);
           } else {
               IERC1155(_collections[i]).safeTransferFrom(msg.sender, to,
_tokenIds[i], _amount[i], "");
           }
       }
   }
```

Description

All transfer-related functions within ValkyrieTransferAgent are adorned with the **onlyWhitelisted** modifier, except for **batchTransferToken**.

Code location

```
contracts/ValkyrieTransferAgent/ValkyrieTransferAgent.sol
```

Recommendation

Apply the modifier to **batchTransferToken.**

Response

Noted and will be added

(VA-6) Code quality and suggestion.

Status

Addressed

Risk Level

Severity: Informational, likelihood: Medium

Description

NFT marketplace generally accommodate both ERC721 and ERC1155, which are widely adopted standards. Presently, the code lacks implementation for ERC1155.

Moreover, in bulk order scenarios, where the order details and maker's signature are already available, the utilization of a Merkle tree for validation might be deemed unnecessary.

Response

This is outside the scope of the initial intended functionality

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, Arcadia and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.