BlockApex

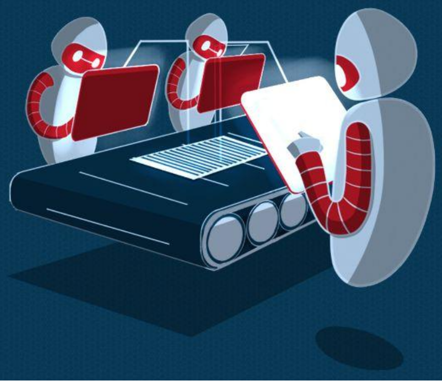
SMART CONTRACT SECURITY ANALYSIS REPORT

```
pragma solidity 0.7.0;
contract Contract {

    function hello() public returns (string) {
        return "Hello World!";
    }

    function findVulnerability() public returns (string) {
        return "Finding Vulnerability";
    }

    function solveVulnerability() public returns (string) {
        return "Solve Vulnerability";
    }
}
```



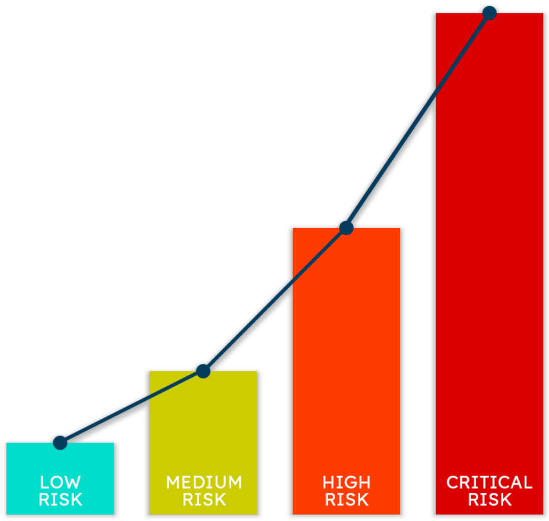
PREFACE

Objectives

The purpose of this document is to highlight the identified bugs/issues in the provided codebase. This audit has been conducted in a closed and secure environment, free from influence or bias of any sort. This document may contain confidential information about IT systems/architecture and intellectual property of the client. It also contains information about potential risks and the processes involved in mitigating/exploiting the risks mentioned below.

The usage of information provided in this report is limited, internally, to the client. However, this report can be disclosed publicly with the intention to aid our growing blockchain community; under the discretion of the client.

Key understandings



CRITICAL RISK xxxx	Critical vulnerabilities are too easy to exploit and can lead to damages/loss in assets or manipulations.
HIGH RISK xxx	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution.
MEDIUM RISK xx	Medium-level vulnerabilities are equally imperative to fix but they tend to have minimal impact on asset loss or data manipulations.
LOW RISK x	Lowest-level vulnerabilities, informational errors, violating code styles/practices usually can't affect smart contract execution; hence they can be ignored.

TABLE OF CONTENTS

INTRODUCTION

BlockApex (Auditor) was contracted by Flower Fam (Client) for the purpose of conducting a Smart Contract Audit/ Code Review. This document presents the findings of our analysis which started from 19 may 2022.

Name
Flower Fam
Auditor
Kaif Ahmed | Muhammad Jarir Uddin
Platform
Ethereum/Solidity
Type of review
Manual Code Review | Automated Code Review
Methods
Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git repository/SHA256 Checksum
3c6fec450c5aa25e6dec9a0378d99dcbf44b4f113d096e581e01d86720b9cd1d
White paper/ Documentation
https://docs.flowerfam.earth/welcome.
Document log
Initial Audit: 19th May 2022 Final Audit: 23rd May 2022

Scope

The git-repository shared was checked for common code violations along with vulnerability-specific probing to detect **major issues/vulnerabilities**. Some specific checks are as follows:

Code review		**Functional review**
Reentrancy	Unchecked external call	Business Logics Review
Ownership Takeover	ERC20 API violation	Functionality Checks
Timestamp Dependence	Unchecked math	Access Control & Authorization
Gas Limit and Loops	Unsafe type inference	Escrow manipulation
DoS with (Unexpected) Throw	Implicit visibility level	Token Supply manipulation
DoS with Block Gas Limit	Deployment Consistency	Asset's integrity
Transaction-Ordering Dependence	Repository Consistency	User Balances manipulation
Style guide violation	Data Consistency	Kill-Switch Mechanism
Costly Loop		Operation Trails & Event Generation

Project Overview

FlowerFam is an NFT based project, after you mint your NFT you can “harvest” them on weekly bases to get 60% royalties. It's quite simple: every flower has a 10% chance to win. The rarer the species of a flower. Besides the weekly harvest, flowers can make $honeycoin through a lot of other fun activities in the Oasis. You can earn $honeycoin by staking your Flower, catching bees, and buying seeds that grow into beautiful new Flowers.

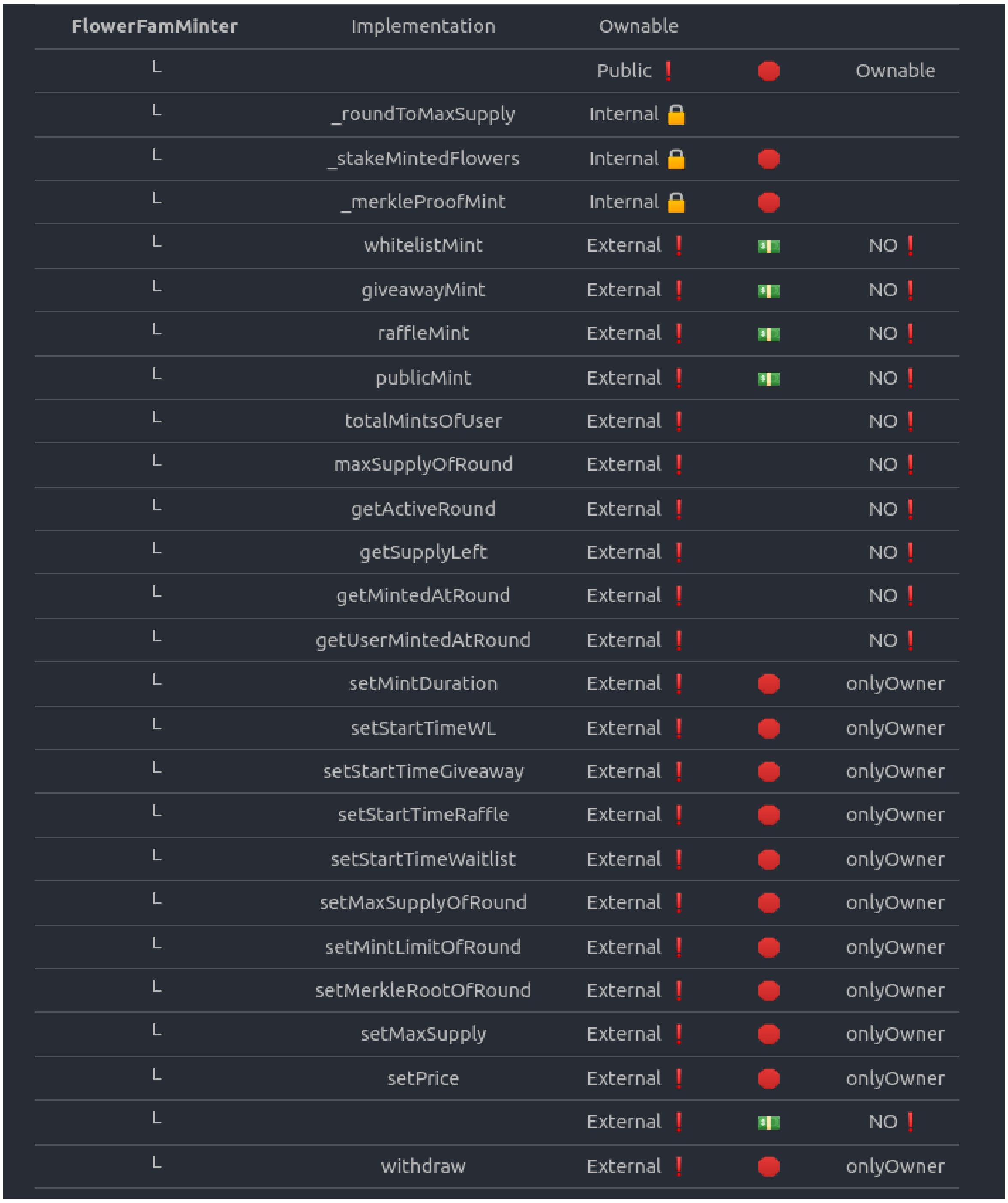
System Architecture

FlowerFam is a single NFT minter contract which is composed of three other contracts, FloweFam.sol , FlowerFamMintPass.sol and FlowerFamEcosystem.sol. This contract is used to make users whitelist so that only whitelisted addresses would be able to mint NFTs.

Methodology & Scope

The codebase was audited in an iterative process. Fixes were applied on the way and updated contracts were examined for more bugs. We used a combination of static analysis tool (slither) and testing framework (Foundry) which indicated some of the critical bugs. We also did manual reviews of the code to find logical bugs, code optimizations, solidity design patterns, code style and the bugs/ issues detected by automated tools.

System Analysis:



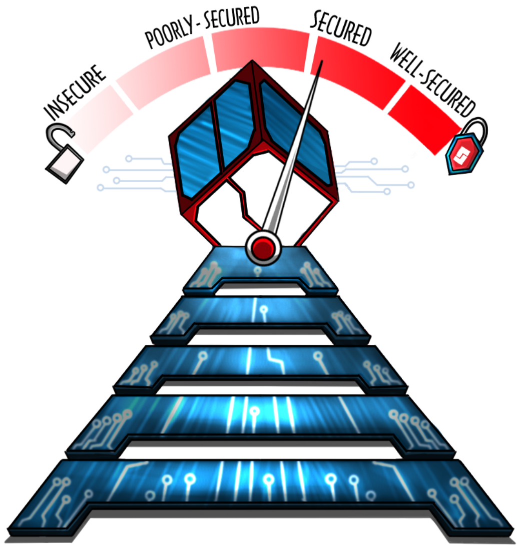
FlowerFamMinter	Implementation	Ownable		
└		Public ❗	🛑	Ownable
└	_roundToMaxSupply	Internal 🔒		
└	_stakeMintedFlowers	Internal 🔒	🛑	
└	_merkleProofMint	Internal 🔒	🛑	
└	whitelistMint	External ❗	💵	NO ❗
└	giveawayMint	External ❗	💵	NO ❗
└	raffleMint	External ❗	💵	NO ❗
└	publicMint	External ❗	💵	NO ❗
└	totalMintsOfUser	External ❗		NO ❗
└	maxSupplyOfRound	External ❗		NO ❗
└	getActiveRound	External ❗		NO ❗
└	getSupplyLeft	External ❗		NO ❗
└	getMintedAtRound	External ❗		NO ❗
└	getUserMintedAtRound	External ❗		NO ❗
└	setMintDuration	External ❗	🛑	onlyOwner
└	setStartTimeWL	External ❗	🛑	onlyOwner
└	setStartTimeGiveaway	External ❗	🛑	onlyOwner
└	setStartTimeRaffle	External ❗	🛑	onlyOwner
└	setStartTimeWaitlist	External ❗	🛑	onlyOwner
└	setMaxSupplyOfRound	External ❗	🛑	onlyOwner
└	setMintLimitOfRound	External ❗	🛑	onlyOwner
└	setMerkleRootOfRound	External ❗	🛑	onlyOwner
└	setMaxSupply	External ❗	🛑	onlyOwner
└	setPrice	External ❗	🛑	onlyOwner
└		External ❗	💵	NO ❗
└	withdraw	External ❗	🛑	onlyOwner

AUDIT REPORT

Executive Summary

The analysis indicates that some of the functionalities in the contracts audited are **working properly**.

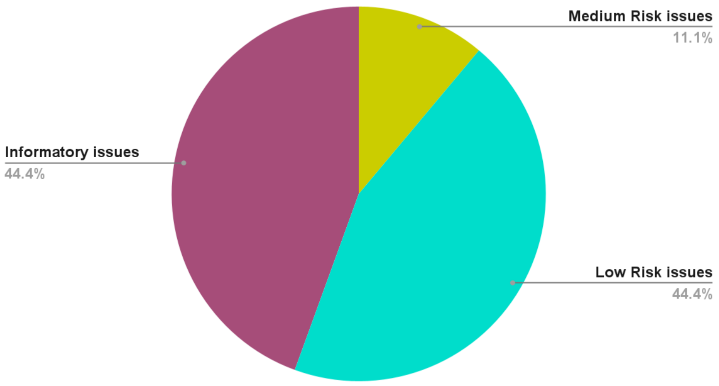
Our team performed a technique called "Filtered Audit", where the contract was separately audited by two individuals. After their thorough and rigorous process of manual testing, an automated review was carried out using Mythril, MythX, Surya and Slither. All the flags raised were manually reviewed and re-tested.



Our team found:

# of issues	Severity of the risk
0	Critical Risk issue(s)
0	High Risk issue(s)
1	Medium Risk issue(s)
4	Low Risk issue(s)
4	Informatory issue(s)

Proportion of Vulnerabilities



Findings

#	Findings	Risk	Status
1.	Centralization risk	Medium	Fixed
2.	setMerkleRootOfRound() can overwrite mapping	Low	Acknowledged
3.	Configuration inconsistency	Low	Acknowledged
4.	Withdraw function argument validation check	Low	Fixed
5.	Recommendation for missing function in the contract file as received by the client	Low	Acknowledged
6.	Netspac missing	Informatory	Acknowledged
7.	Order of Functions	Informatory	Fixed
8.	Unchecked setter values	Informatory	Acknowledged
9.	Interface Instead of Contract	Informatory	Fixed

Medium-risk issues

1. **Centralization risk.**

Description:
Heavy centralization risk using onlyowner check on *withdraw()* function, assuming owner address is never compromised else it might lead to lost funds.

```
function withdraw(address _to) external onlyOwner {
        uint256 balance = address(this).balance;
        require(balance > 0, "Balance zero");
        payable(_to).transfer(balance);
    }
```

Status:

Resolved

Low-risk issues

2. **setMerkleRootOfRound() can overwrite mapping**

   Description:

   The current implementation of the above function can override the mapping *roundToMerkleRoot* which can lead to loss of round minting for any of the four rounds.

   Status:

   Acknowledged

   Remedy:

   Place a check to ensure the root can only be set if the round has completed the preset timeline.

3. **Configuration inconsistency:**

   Description:

   Calling the functions *setMintLimitOfRound* or *setMaxSupplyOfRound* during a round can lead to inconsistencies of assumed configurations of the minting system.

   Status:

   Acknowledged

   Remedy:

   Owner can change configurations during the minting rounds and can lead to inconsistent management of user NFTs.

4. **Withdraw function argument validation check**

   Description:

   Withdraw() function is only callable by owner but still there is a chance of mistake. Function only checks for the amount it should also check for the value owner sends from the arguments.

   Status:

   Fixed

   Remedy:

   There should be a zero address check inside the function.

5. **Recommendation for missing function in the contract file as received by the client.**

   Description:

   The contract *IFlowerFamMintPass* contains a function named *validPasssesLeft* which is incompatible with the original files received later (out of scope) containing a similar function named as *userPassesLeft().*

   Status:

   Acknowledged

   Remedy:

   Ensure the functions are named properly and following the implemented interface architecture of the file system.

Informatory issues

6. No NatSpec Documentation

Description:
NatSpec documentation is an essential part of smart contract readability; it is therefore advised that contract and following files should contain proper explanatory commenting;

- *FlowerFamMinter.sol*

Status:
Acknowledged

7. Order of Functions

Description:
Move *receive()* function right below the constructor. Move most useable/callable (Public/External) functions right below the constructor and internal functions right below the public functions as suggested in the solidity docs:
“Ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier”.
Functions should be grouped according to their visibility and ordered:

constructor
receive function (if exists)
fallback function (if exists)
external
public
internal
private

Status:
Fixed

8. Unchecked setter values

Description:
All setter values are unchecked and can lead to redundant calling setter functions. There should be a zero value check in following functions:
setMintDuration()
setStartTimeWL()
setStartTimeGiveaway()
setStartTimeRaffle()
setStartTimeWaitlist()
setMaxSupplyOfRound()
setMintLimitOfRound()
setMerkleRootOfRound()

Status:
Acknowledged

9. Interface Instead of Contract

Description:

Contracts named *IFlowerFam* and *IFlowerFamMintPass* can be changed for interface type.

Status:
Fixed

Remedy:

Ensure that the contracts are retyped as interfaces and reflect all consequential changes e.g.

- In *IFlowerFamMintPass* contract, the function *balanceOf()* be marked with external accessibility
- In the *IFlowerFam* contract, the function named *ownerOf()* can be marked as external accessible.

DISCLAIMER

The smart contracts provided by the client for audit purposes have been thoroughly analyzed in compliance with the global best practices till date w.r.t cybersecurity vulnerabilities and issues in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token, its sale or any other aspect of the project.

Crypto assets/tokens are results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third-party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks.

This audit cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.