QuillAudits

Audit Report
December, 2023

For

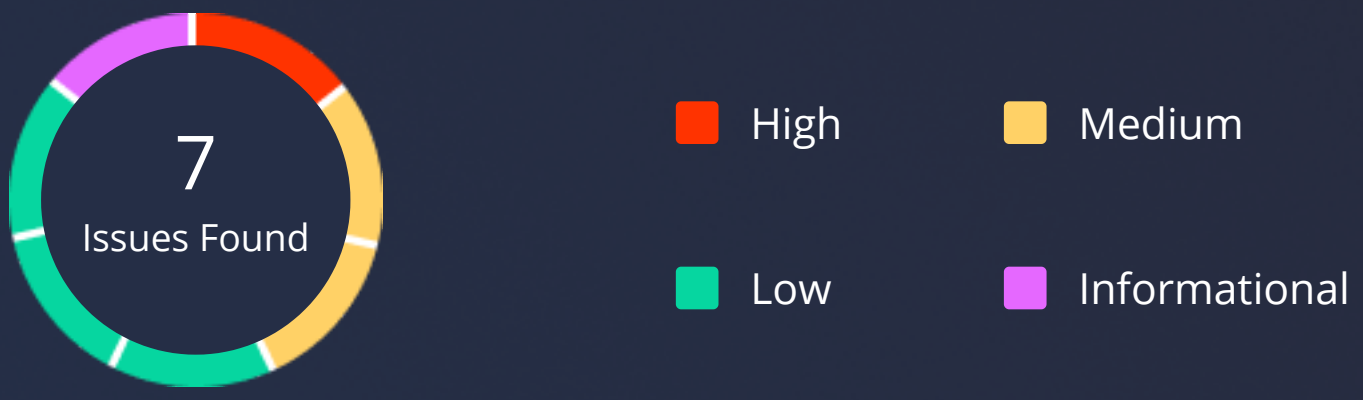
Table of Content

Table of Content

Executive Summary

Project Name	Greedy Art
Project URL	*https://greedy.art/*
Overview	GreedyArt - a decentralized NFT auction with full payment. Bids made in the auction are non-refundable and form a reward pool, which is shared between the seller and the auction winner.
Audit Scope	Manual and Automated review, Functional testing
Contracts in Scope	*https://testnet.bscscan.com/address/0x2FCD6984d80e8455676B8a34FC64C280D6Eac43f#code* *https://testnet.bscscan.com/address/0x3A7B9e7f0db834E4972305734C3a492eCf51Fc48#code*
Language	Solidity
Blockchain	BNB
Method	Manual and Automated Review, Functional Testing
Review 1	16th November 2023 - 26th November 2023
Updated Code Received	4th December 2023
Review 2	5th December 2023 - 6th December 2023
Fixed In	*https://testnet.bscscan.com/address/0x912Ab5373eD35Db7E5D3377B3e5A21b1174d142B#code* *https://testnet.bscscan.com/address/0x80Cae2E077D6d2d31660B1152EFfB6a71A7f5E4A#code*

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	1	3	1

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Solhint, Mythril, Slither, Solidity Statistic Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

A. Contract - GreedyMarket.sol

High Severity Issues

A.1 Precision Loss in Fee Calculation

Line

83

Functions - buy()

```
uint256 fee = (amountIn_ * uint256(PROJECT_FEE_RATE)) / 10000;
uint256 partnerFee = 0;

// take partner fee
if (code_ != 0x0 && codes[code_] != address(0)) {
    // @audit precision loss here because of this equation. Here we are multiplying and then again doing the division.
    // @note partnerFee = (((amountIn_ * uint256(PROJECT_FEE_RATE)) / 10000 * uint256(partners[codes[code_]].rewardRate)) / 10000)
    partnerFee =      Emil Sergeev, 3 months ago • chore: restructure contracts dir
        (fee * uint256(partners[codes[code_]].rewardRate)) /
        10000;

    paymentToken.safeTransferFrom(msg.sender, codes[code_], partnerFee);
}
```

Description

We used a specific technique for analyzing these: manually expanding the function calls & variables in an equation to expose hidden division before multiplication. The code snippet shown above exhibits a potential precision loss issue in the calculation of the **partnerFee** and then dividing the result by 10000. This sequence of operations can lead to precision loss. Precision loss in financial calculations can result in inaccurate fee distributions, potentially leading to financial discrepancies and unintended outcomes.

Simplified equation:
partnerFee = (((amountIn_ * uint256(PROJECT_FEE_RATE)) / 10000 * uint256(partners[codes[code_]].rewardRate)) / 10000)

Remediation

To address the precision loss issue, consider using a more robust approach, such as performing the multiplication before dividing to minimize the impact of rounding errors.

Status

Resolved

Medium Severity Issues

A.2 Funds can be lost if any **partner, projectFeeReceiver addresses** is blacklisted

Line

86,90

Function - buy

```
        partnerFee =
            (fee * uint256(partners[codes[code_]].rewardRate)) /
            10000;

        paymentToken.safeTransferFrom(msg.sender, codes[code_], partnerFee);
    }       Emil Sergeev, 3 months ago • chore: restructure contracts dir

    // take project fee // @audit what if the partner address is blacklisted?
    paymentToken.safeTransferFrom(
        msg.sender,
        projectFeeReceiver,
        fee - partnerFee
    );
```

Description

Popular tokens such as **USDT** may implement a blocklist of addresses that are not allowed to send or receive that token. This will cause a function to revert when attempting to transfer to one of these addresses.

If any of the **partner, projectFeeReceiver addresses** is blacklisted (due to illegal activities, using Tornado Cash for privacy reasons, etc.), depending on the implementation this could lock funds in a contract indefinitely.

Remediation

1. **Individual Withdrawals:**

   Instead of directly transferring funds to each address, maintain a state variable that tracks the amount each address is entitled to. Once the escrow concludes, introduce a withdrawal function. This allows participants to individually claim their funds. If one participant is blacklisted, it won't hinder others from retrieving their share.

2. Address Redirection:

To address the challenge of blacklisted participants being unable to withdraw, introduce a function that lets participants designate an alternate withdrawal address. This way, even if their primary address is blacklisted, they can redirect and access their funds through another address.

Status

Resolved

B. Contract - GreedyToken.sol

Medium Severity Issues

B.1 Inability to Set **deflationFeeReceiver** State Variable

Line

N/A

Function - N/A

```
ftrace | funcSig
function removeMember(address address_) external onlyOwner {
    require(deflationFeeReceiver != address_, "GT: FEE RECEIVER");
    require(address_ != address(0), "GT: ADDR");
    require(isMember[address_], "GT: NOT MEMBER");
    isMember[address_] = false;
    emit WhitelistChanged(address_, false);
}
```

Description

The contract lacks a mechanism to set or update the deflationFeeReceiver state variable. This state variable is responsible for collecting fees during token transfers. In the absence of a function or process to change this address, any funds sent to the deflation fee address will be irretrievable and effectively lost. This poses a risk of financial loss and could lead to discrepancies in fee management.

Remediation

To mitigate this issue, implement a function that allows the contract owner or authorized administrators to update the deflationFeeReceiver address. Ensure that only trusted entities have the ability to modify this critical address.

Status

Acknowledged

Low Severity Issues

B.2 Lack of Check for Existing Whitelisting Status

Line

207, 216

Functions - addToWhitelist, removeFromWhitelist

```
/**...
ftrace | funcSig
function addToWhitelist(address account_) external onlyOwner {
    require(account_ != address(0), "GM: ADDR"); // @audit checks are not there if account is already whitelisted or not
    whitelisted[account_] = true;
}

/**...
ftrace | funcSig
function removeFromWhitelist(address account_) external onlyOwner {
    require(account_ != address(0), "GM: ADDR"); // @audit checks are not there if account is even whitelisted or not
    whitelisted[account_] = false;
}
```

Description

The functions **addToWhitelist** and **removeFromWhitelist** lack checks to verify whether an address is already whitelisted or not before attempting to add or remove it. Without this verification, redundant operations may occur, potentially leading to increased gas costs or unintended changes in the whitelist status. The absence of a check for existing whitelisting status could result in inefficiencies and unnecessary state modifications.

Remediation

To address this issue, implement checks to verify whether an address is already whitelisted before attempting to add or remove it. This ensures that state modifications are performed only when necessary, reducing gas costs and improving overall contract efficiency.

Status

Resolved

B.3 State-changing methods are missing event emissions

Line

194, 198, 207, 216

Functions - setMinPurchaseAmount, setProjectFeeReceiver, addToWhitelist, removeFromWhitelist

```
/**...
ftrace | funcSig
function setMinPurchaseAmount(uint256 minSaleAmount_) external onlyOwner {
    minPurchaseAmount = minSaleAmount_;
}

ftrace | funcSig
function setProjectFeeReceiver(address newReceiver_) external onlyOwner {
    require(newReceiver_ != address(0), "GM: ADDR");
    projectFeeReceiver = newReceiver_;
}

/**...
ftrace | funcSig
function addToWhitelist(address account_) external onlyOwner {
    require(account_ != address(0), "GM: ADDR"); // @audit checks are not there if account is already whitelisted or not
    whitelisted[account_] = true;
}

/**...
ftrace | funcSig
function removeFromWhitelist(address account_) external onlyOwner {
    require(account_ != address(0), "GM: ADDR"); // @audit checks are not there if account is even whitelisted or not
    whitelisted[account_] = false;
}
```

Description

Critical functions within the contract lack the emission of events. Events play a vital role in providing transparency, enabling external systems to react to changes, and offering a way to track important contract activities. The absence of events can hinder monitoring and auditing efforts, making it difficult to detect and respond to critical contract actions.

Remediation

To address this issue and improve the transparency and auditability of the contract, you should add appropriate event emissions in critical functions. These events should capture essential information about the function's execution, including input parameters and outcomes. Additionally, consider emitting events both before and after critical state changes, where applicable.

Status

Resolved

B.4 State-changing methods are missing event emissions

Line

6

Contracts - GreedyToken and GreedyMarket

N/A

Description

Inheriting from OpenZeppelin's **Ownable** contract means you are using a single-step ownership transfer pattern. If an admin provides an incorrect address for the new owner this will result in none of the **onlyOwner** marked methods being callable again. The better way to do this is to use a two-step ownership transfer approach, where the new owner should first claim its new rights before they are transferred.

Remediation

Use OpenZeppelin's **Ownable2Step** instead of **Ownable**.

Status

Resolved

Informational Issues

B.5 General Recommendation

In the case of stable coins, the hierarchy of authority plays a crucial role and our team recommends that the roles with privileges must be given to the accounts with proven authority and functions like “Minting” and “Burning” must be used cautiously because once these functions are called on the mainnet then could be no minting or burning. Hence, it will be an irreversible change.

Note To Users:

1. According to the contract's algorithm, the token price invariably grows

2. The project team does not own the tokens if, immediately after deployment of the Greedy Token contract, all 100,000,000 GREEDY will be transferred by the Greedy Token deployer to the Greedy Market contract and the team has no methods to obtain them without buyback. Also, the team cannot issue additional tokens.

3. The project team cannot lock the tokens or prohibit their resale back to the smart contract.

Functional Tests

Some of the tests performed are mentioned below:

- Should be able to grant Minter, Burner and Asset Protection Roles to accounts
- Should be able to Mint and Burn tokens (from the owner's account as well as from any other account)
- Should be able to transfer tokens
- Should be able to transfer ownership and revert for a zero address
- Should revert if transfer amount exceeds balance or transfer amount is zero
- Should revert if Minter and Burners don't have desired roles
- Should be able to deduct the fee from the transfer amount and transfer it to the fee recipient
- Should not deduct fee if sender is owner, receiver is fee recipient, excluded from fee, or the fee percentage amount is not greater than zero
- be able to freeze and unfreeze accounts and revert if the caller does not have the asset protection role

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Greedy Art. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Greedy Art smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Greedy Art smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Greedy Art to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2023

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com