QuillAudits

AUDIT REPORT

May, 2025

For

Hand of God

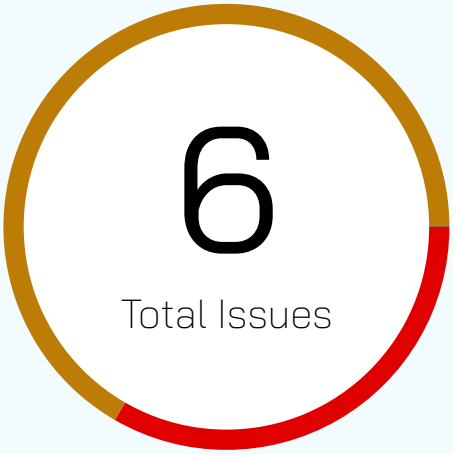
Table of Content

Executive Summary

Project name	Hands of God V2
Project URL	https://www.handofgod.app/
Overview	The MasonryV3 contract implements a staking mechanism where users can stake SHARE tokens to earn HOG rewards. It includes features like lockup periods, reward distribution with burn mechanisms, and governance functions. The veGHOG contract is an ERC721 implementation that allows users to lock GHOG tokens and receive NFTs with different fractions (Choir of Echoes, Ember Covenant, Silent Ledger) and rarity tiers based on lock amount. The veGHOGStaking contract connects veGHOG with MasonryV3, enabling GHOG token staking through NFT ownership. It handles reward distribution, compounding, and includes features to claim rewards by epoch with burn mechanisms and caller incentives. The system includes a feature to unlock GHOG after the reward pool end time by burning the veNFT.
Audit Scope	The scope of this Audit was to analyze the Hands of God V2 Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/chimpytuts/hand-of-god-contracts
Contracts in Scope	contracts under scope: contracts/venft/veGHOG.sol contracts/venft/veGHOGStaking.sol contracts/MasonryV3.sol
Branch	Main

Commit Hash	788aba0dfaa519fbc07216425f923554f87f412
Language	Solidity
Blockchain	Sonic
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	5th may 2025 - 13th may 2025
Updated Code Received	23rd April 2025
Review 2	23rd April 2025
Fixed In	f3ec54720f72d980caaa6ceb1f31c8f0a0c5da81

Number of Issues per Severity



High	2 (33.33%)
Medium	4 (66.67%)
Low	0 (0.00%)
Informational	0 (0.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**2**	**4**	0	0
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Computations 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

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] 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 behavior.
- Token distribution and calculations are as per the intended behavior 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, Foundry, Solhint, Mythril, Slither, Solidity statistical 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 Issues

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

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

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.

High Severity Issues

Rare NFTs Cannot Upgrade to Super-Rare

Resolved

Path

contracts/venft/veGHOG.sol

Function

increaseLock

Description

The increaseLock function prevents Rare (rarity 2) NFTs from being upgraded to Super-Rare (rarity 3) due to the condition currentRarity <= 1. This restricts legitimate upgrade for users who have already obtained Rare NFTs, even if they lock enough GHOG to qualify for Super-Rare status and the super-RareCount is below SUPER_RARE_CAP.

Missing Rarity Count Decrements in Burn Function

Resolved

Path

contracts/venft/veGHOG.sol

Function

burn

Description

When an NFT is burned via the burn function, the corresponding counts (superRareCount, rareCount) are not decremented. This leads to inaccurate tracking over time, which may unfairly prevent new users from minting higher rarity NFTs when caps appear to be reached but actual circulation is lower due to burns.

Medium Severity Issues

Ability to Recover Critical HOG Tokens

Resolved

Path

contracts/venft/veGHOGStaking.sol

Function

recoverTokens

Description

While the function prevents recovering ghog tokens, it allows the owner to recover any other token including hog (the reward token). This could allow the owner to extract hog tokens that were intended for future reward distributions.

Recommendation

Add a check to prevent recovering hog tokens.

Unclaimed Rewards Permanently Lost on NFT Unlock

Resolved

Path

contracts/venft/veGHOGStaking.sol

Function

unlockGHOG

Description

The unlockGHOG function doesn't check for or claim pending rewards before burning the NFT. Users can unknowingly lose accumulated rewards across multiple epochs when unlocking their GHOG. Once the NFT is burned, these rewards become permanently inaccessible as they're still recorded in state but can no longer be claimed without a valid token ID.

Recommendation

Add logic to check for and handle NFT-specific unclaimed rewards before burning.

Incorrect check in getUpgradeableRarityTiers function

Resolved

Path

contracts/venft/veGHOG.sol

Function

getUpgradeableRarityTiers

Description

The getUpgradeableRarityTiers function provides incorrect information to users about their NFT's upgrade eligibility. While the contract now allows Rare NFTs to upgrade to Super-Rare, this function still reports that such upgrades are impossible.

The function still uses the outdated condition nft.rarity <= 1 && superRareCount < SUPER_RARE_CAP.

Recommendation

:

```
function getUpgradeableRarityTiers(uint256 tokenId) external view returns (bool
canUpgradeToRare, bool canUpgradeToSuperRare) {
    require(_exists(tokenId), "Token does not exist");
    VeNFT memory nft = veNFTs[tokenId];

    // Check if can upgrade to Rare
    canUpgradeToRare = nft.rarity < 2 && rareCount < RARE_CAP;

    // Check if can upgrade to Super-rare (updated to match increaseLock logic)
    canUpgradeToSuperRare = nft.rarity < 3 && superRareCount < SUPER_RARE_CAP;
}
```

Missing Rarity Count Decrements on Upgrades

Resolved

Path

contracts/venft/veGHOG.solcontracts/venft/veGHOG.sol

Function

increaseLock

Description

When an NFT's rarity is upgraded, the count for the previous rarity tier is not decremented. This leads to artificial inflation of rarity counts, which could eventually prevent users from minting or upgrading NFTs when caps appear reached but actual circulation is lower.

Recommendation

decrement the count for the previous rarity tier

```
if (newRarity > currentRarity) {
    if (newRarity == 3) { // Upgrading to Super-rare
        if (superRareCount < SUPER_RARE_CAP && currentRarity < 3) {
            if (currentRarity == 2) rareCount--; // Decrement rareCount when upgrading from Rare
            superRareCount++;
            nft.rarity = newRarity;
        }
    } else if (newRarity == 2) { // Upgrading to Rare
        if (rareCount < RARE_CAP) {
            rareCount++;
            nft.rarity = newRarity;
        }
    } else {
        nft.rarity = newRarity;
    }
}
```

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 Hand of God. We performed our audit according to the procedure described above.

Issues of High and Medium severity were found. Hand of God team resolved them all

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

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

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

May, 2025

For

Hand of God

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com