QuillAudits

Audit Report
September, 2024

For

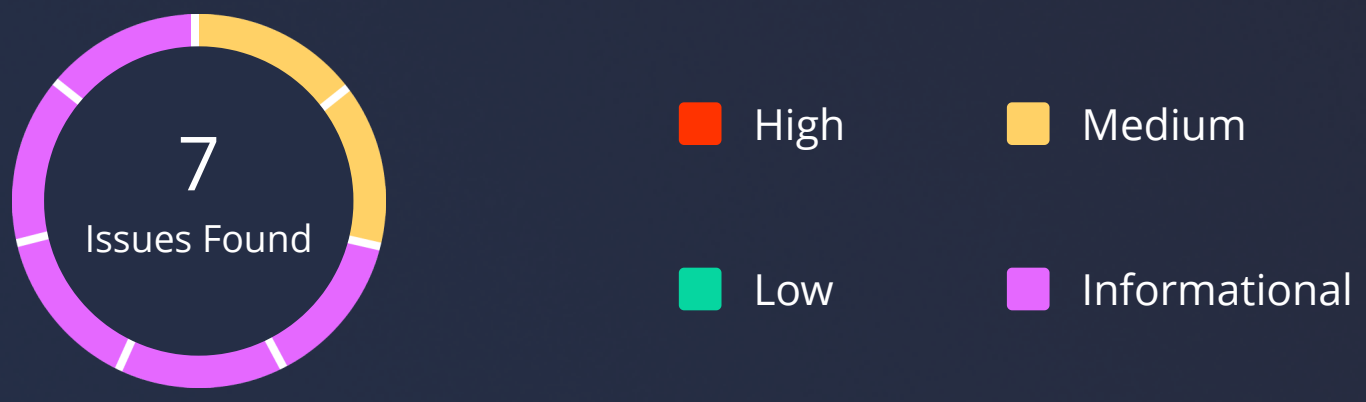
HUDDLE 01

Table of Content

Executive Summary

Project Name	Huddle01
Project URL	https://huddle01.com/
Overview	Huddle01 sales contracts receive payment for the purchase of sales tokens and tracks users information which is used to airdrop NFT that represents the sales token. It is important to note that the airdrop of NFT is not done immediately. The contract is basically a tracking system and receives payment for interested users. The sale contract inherits the Openzeppelin library; it uses the MerkleProof contract for whitelist purposes, Ownable for permissioned functions, and SafeERC20 for safe transfer of the payment token. The contract tracks promo codes used by users during purchase.
Audit Scope	Cashable FixedSaleV12 Purchaseable Sale Whitelistable
Contracts In-Scope	https://github.com/Huddle01/sale-contract
Commit Hash	77b2bbcdb2130ec9f77ce2bd5ebbf04f65c55a94
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	11th September 2024 - 25th September 2024
Updated Code Received	30th September 2024
Second Review	30th September 2024
Fixed In	a95838c0ee29dcef60465fb803a41c36dffd6bbd

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	2	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	5

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
- Unsafe type inference
- Style guide violation
- 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 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.

Medium Severity Issues

1. Centralization Issue

Description

The sale contract is not only limited to track the number of sale tokens purchased. Users have to make payment into the contract without corresponding tokens immediately sent to the users. The protocol promises to airdrop an NFT in accordance with their lifecycle. However, the sale contract does not define or clarify how the NFT representation will work but it is dependent on the protocol to airdrop any amount of NFT to any amount of payment tokens made into the contract.

Recommendation

protocol should redesign the contract to immediately send these sales tokens to users or an NFT representation. This helps get the users trust since there is a form of receipt sent to their addresses on every purchase.

Status

Acknowledged

Client's comment:

The decision to delay NFT minting was intentional. We plan to run specific campaigns around the minting event. Additionally, while the sale will take place on Arbitrium, the NFTs will be minted on our Layer 3 blockchain, making immediate minting both complex and error-prone.

2. Impossible to update undesirable public allocation value after sales has began

Path

FixedSaleV12

Function

setPublicAllocation

Description

The setPublicAllocation is expected to be called by the contract owner or the assigned whitelist setter to set the value of public allocation. This is only before the start time period for the sale. This makes the setPublicAllocation impossible to call afterwards. If the protocol sets an undesirable value before start time, this leaves the protocol with no choice but to operate on that value. Another possible instance is deploying the sale contract with a start time set to the block.timestamp or to a closer time, and if setPublicAllocation was not invoked immediately, then publicAllocation will be 0. This leaves the contract nonfunctional and users won't be able to interact with it.

Recommendation

Consider setting the public allocation value at the level of the constructor since the setPublicAllocation function will not be possible to invoke when sale begins.

Status

Acknowledged

Client's comment:

The allocation values are intentionally fixed and predefined based on crypto-economic models, and aren't designed to be altered post-sale. More details can be found here:

https://docs.google.com/spreadsheets/d/1hWrheJ5AGpTBChy7-K1lpOUlvy09xT1ErtGr8AUgxlQ/edit?gid=0#gid=0

Informational Issues

3. State variable should be marked as constant

Path

Purchasable

Function

maxPromoCodePerUser

Description

This state variable was initialized but there are no functions to further update this variable.

Recommendation

Making the variable constant will prevent it from taking up a storage slot and save gas.

Status

Resolved

4. Remove unused state variable and modifier

Path

Cashable.sol

Function

hasCashed

Modifier

onlyAfterSale

Description

A variable and modifier were declared but were never used within a function.

Recommendation

remove unused data type.

Status

Resolved

5. Cash event emits zero constantly for saleTokenBalance

Path

Cashable.sol

Function

Cash

Description

The Cash event which gets emitted when the cashier or the contract owner invokes the cashPaymentToken function for withdrawal of the payment tokens. However, this will constantly emit 0 for the saleTokenBalance parameter in the event. There is a need to also rename the paymentTokenBalance parameter in the event to a clarified variable name since it emits the amount to be withdrawn and not the balance of payment token in the contract.

Recommendation

Remove the saleTokenBalance parameter and rename paymentTokenBalance to amount.

Status

Resolved

6. getSaleTokensSold internal function was declared but unused

Path

Cashable.sol | Sale.sol

Function

getSaleTokensSold

Description

The purpose of the getSaleTokenSold function was to get information of how much sales token has been acquired. However, this function which has an internal visibility and made the Cashable an abstract contract, was clearly defined in the Sale contract yet it was not invoked anywhere. This was declared but left unused.

Recommendation

remove the function or define the function visibility to be public and possible to invoke by any user.

Status

Resolved

7. Variable only set at the constructor should be made immutable

Path

Purchasable

Function

salePrice

Description

This variable only gets set at the constructor and there is no function to update the salePrice value after deployment.

Recommendation

to save gas, this variable should be marked as immutable.

Status

Resolved

Functional Tests Cases

Some of the tests performed are mentioned below:

- Should revert sales if startTime has not reached.
- Should revert when setwhitelistSetter called by any account other than owner.
- Should accept when setWhitelist called by owner.
- Should revert when setWhiteList called by any account other than owner and whtelistSettter.
- Should revert when IntegerSale is set after sale has begun.
- Should revert when purchase before the sale has begun.
- Should revert when try to purchase when the sale is halted.
- Should revert when cashing more amount than the contract has.
- Should revert when cashPaymentToken is called by any account other than the cashier or owner.
- Should revert when setCasher is called by anyone other than the owner.
- Should revert when publicAllocation is set after sale has begun.

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 Huddle01. We performed our audit according to the procedure described above.

Some issues of medium and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture. In The End, Huddle01 Team Resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Huddle01. 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 Huddle01. 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 Huddle01 Team to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2024

For

HUDDLE 01

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com