ChainAudits

Smart Contract Audit

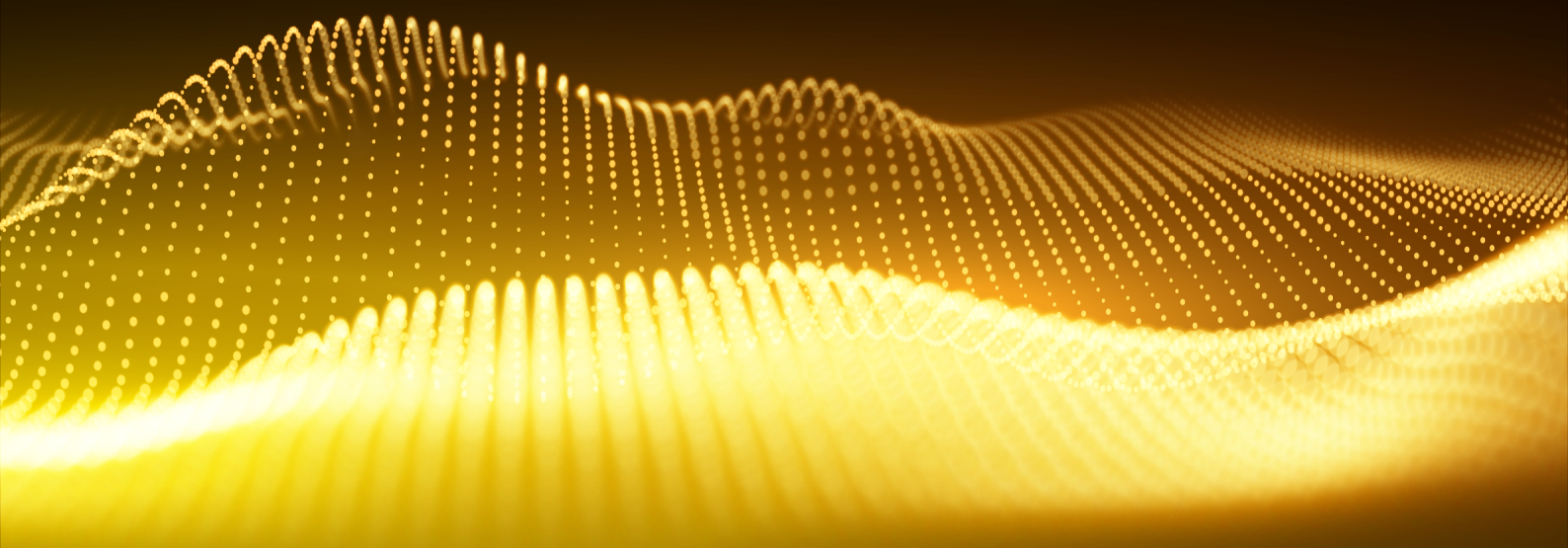
Security Assessment

Audit Dashboard Link

https://app.chainaudits.io/project/octoswap-1?cid=111

OctoSwap

Universal Router



ChainAudits

Table of Contents

ChainAudits

Project Details

CODEBASE
Upload

NETWORK
N/A

LANGUAGE
Solidity

NAME
OctoSwap

TOTAL SUPPLY
N/A

OWNER
N/A

WEBSITE
https://octo.exchange/

UNIT TESTS
N/A

FORK
N/A

ABOUT THE PROJECT
OctoSwap is a decentralized exchange on Monad that combines traditional automated market maker pools with concentrated liquidity pools in a single trading system. Its smart routing engine sources liquidity across both models to provide efficient execution for users. The platform includes swap functionality, liquidity management, and ecosystem features such as gasless approvals, MEV protections, incentives, and a points system.

Protocol Inheritance Analysis:

OctoSwap's contracts are verified to be directly forked from Uniswap contracts. All functional components were reviewed to confirm that the implemented logic matches Uniswap's original design. The differences identified are cosmetic in nature and do not alter the core functionality or expected behavior of the protocol.

Social Media Links

Linktree	N/A
Telegram	N/A
Facebook	N/A
Medium	N/A
X	• https://x.com/OctoSwapDex
Instagram	N/A
Youtube	N/A
Github	N/A
Reddit	N/A
Discord	N/A
Tiktok	N/A
LinkedIn	N/A
Flooz	N/A
Coingecko	N/A
Coinmarketcap	N/A

ChainAudits

Vulnerability Summary

Critical	Critical risks are those that affect the platform's safe operation and must be resolved before launch. Users should avoid investing in any project with unresolved critical risks.
High	High risks include centralization issues and logical errors. These risks can potentially result in the loss of funds or control over the project under certain conditions.
Medium	Medium risks might not directly threaten users' funds, but they can impact the platform's overall functionality.
Low	Low risks are similar to the above categories but on a smaller scale. They typically do not compromise the project's overall integrity but may lead to less efficient solutions.
Informational	Informational errors are recommendations aimed at improving code style or aligning operations with industry best practices. These usually do not affect the code's overall functionality.
Optimization	Optimization issues refer to inefficiencies in the code or smart contract logic that could be improved for better performance, reduced gas costs, or enhanced scalability. These issues do not impact security but can make the system more cost-effective and efficient.

Note - The provided audit report thoroughly examines the security aspects of the smart contract employed in the project, encompassing potential malicious manipulation of the contract's functions from external sources. However, it's important to note that this analysis does not incorporate functional or unit testing of the contract's logic. Therefore, we cannot ensure the absolute correctness of the contract's logic, including internal calculations within the formulae utilised in the contract.

ChainAudits

Overview

Score

> The score is solely based on audited code findings and is not to be used as financial advice.

In-Scope Files

The team provided us with files to review during security audits. This audit covered the following files listed below with their respective SHA-1 Hash.

File Name	SHA-1 Hash
BytesLib.sol	64a946be6e6d27940d3c4f66700cc806e042b682
Commands.sol	1d0cf8099484bf1f88c01a389e7c0792297def84
Constants.sol	fc94e4ca540c94b4332616fa642f8cc74451ce19
Dispatcher.sol	368551e379f339dc3626f587eda03d1de4ad3776
IUniversalRouter.sol	88daa61d529c2538258cc3fd383e59055e7f4b65
Lock.sol	b1c3f26ac2913495a411615383a11002256d0300
Locker.sol	e2ec05e23727f084f87fa3341104b9d2328bf31d
MaxInputAmount.sol	34ac9c0fa871413563c998e13dc6990a155b0d92
MigratorImmutables.sol	37cbff547eda3969ad4c19ca68267dfe54fe274a
Payments.sol	03ba5626fab5484bba5355b05d5c66d68ba8c8e8
PaymentsImmutables.sol	b0c612dab4455701350c2daa33d3283249118436
Permit2Payments.sol	3c1575b8fe0e2b00fd4bf884295b1006ae1ccdd8
UniswapImmutables.sol	6f549af186e3e10e91a0da5e7b3a1c9819fe0acd
UniswapV2Library.sol	5a9878482cf8179102216c1f425dbe1d36f6f610
UniversalRouter.sol	ad7880fdd71e9449465af63e1588f041dd8e8438
V2SwapRouter.sol	83fdda3ec2488652a5158d7c86719cf0e1846d2f
V3Path.sol	6bc107ce6158b8802c45a6cad6601ac6d0b4eefe
V3SwapRouter.sol	ce48cf08edde6171ec790949f2782eae605c9100
V3ToV4Migrator.sol	0f0bdaf74094278fdd31f358f1910123151d870d
V4SwapRouter.sol	bd3ce362d7d0f88506543291538ae797aab1f472

Please note that files with hash values different from those listed in this table have been changed after security checks, intentionally or unintentionally, as a particular hash value may indicate a changed state or potential vulnerabilities not checked in this scan.

> Note for Investors: We only examined agreements indicated in the indicated ratings. No contracts associated with the project beyond this range have been audited, therefore, we cannot provide insight or assume responsibility for their security

External/Public functions

External/public functions can be invoked outside of the contract, i.e., accessed by other contracts or external accounts on the blockchain. These functions are identified using external or public visibility modifiers in the function declaration.

State variables

State variables are stored on the blockchain as part of the contract conditions. They are declared at the contract level and can be accessed and changed by any action in the contract (except with modifiers like onlyOwner, etc.). State transitions can be described using a visibility modifier, such as public, private, or internal, which refers to the access to the transition.

Components

Contracts	5
Libraries	7
Interfaces	1
Abstract	7

Exposed Functions

Public	7
Payable	5
External	5
internal	65
Private	7
Pure	19
View	12

State Variables

Total	41
Public	2

Capabilities

Solidity Versions observed	^0.8.24, >=0.6.0, ^0.8.0, >=0.8.0
Transfers ETH	`yes`
Can Receive Funds	yes
Uses Assembly	yes (27 asm blocks)
Has Destroyable Contracts	No

Audit Information

Strategies

While our projects undergo rigorous testing to address all known vulnerabilities, it's important to provide further clarification and understanding by outlining specific vulnerabilities that have been identified. This additional information enhances transparency and helps stakeholders comprehend the security measures taken and any potential risks mitigated.

Title	issue
Reentrancy	Improper Enforcement of Behavioral workflow
Unexpected Ether Balance	Improper Locking
Code with no effects	Irrelevant Code
Flash Loan Attacks	External Oracle Manipulation
Sandwich Attacks	A Form of Front Running
Write to arbitrary storage location	Improper Write to Arbitrary Storage Location
Variable Shadowing	Improper Coding Standards
Unprotected SELF-DESTRUCT	Improper Access Control
Potential Honeypot	Improper Function
Unprotected ETH withdrawal	Improper Access Control
Outdated Compiler Version	Using components with Known vulnerabilities
Weak Source of Randomness from Chain Attributes	Use of Insufficiently Random values
Unsafe use of libraries	Improper Implementation
Wrong Implementation of Token Standards	Improper Coding Standards

Auditing Strategy and Techniques Applied

All Projects at ChainAudits undergo our in-house developed “4-Eye” method. This process ensures that the code is analyzed not by a single auditor but by our entire technical team. Moreover, this is accomplished most efficiently, leaving no stone unturned.

We manually check every file, line by line. Automated tools are used solely to help us achieve faster and better results.

Code Analysis Methodology

The auditing process follows a routine series of steps:

- Manual Code Review:
  - Evaluate the overall structure and organization of the smart contract code line-by-line
  - Review the implementation of access control mechanisms to ensure that sensitive functions are appropriately restricted to authorized users.
  - Ensure that critical vulnerabilities are adequately tested and that edge cases and boundary conditions are covered.
- Static Code Analysis:
  - Utilise static analysis tools to scan the codebase for vulnerabilities and security issue
  - Identify common vulnerabilities like reentrancy, integer overflow/underflow, and unchecked external calls.
  - Evaluate compliance with coding standards and best practices, ensuring adherence t security guidelines.
- Code Structure and Architecture Review:
  - Analyze the overall structure and architecture of the smart contract code.
  - Assess the modularity, readability, and maintainability of the code.
  - Review the separation of concerns and adherence to design patterns for robustness and security.
- Security Best Practices Evaluation:
  - Evaluate the implementation of security best practices, including access control mech anisms and input validation.
  - Verify proper error handling to mitigate potential vulnerabilities and ensure contract robustness.
  - Check for gas optimization techniques to enhance efficiency and reduce transaction costs.
- External Dependency Assessment:
  - Assess the integration with external contracts, libraries, or services.
  - Review the security of external dependencies and their impact on the overall security of the smart contract codebase.
  - Ensure secure interactions with external components to prevent vulnerabilities and attack vectors.
- Post-Analysis Support:
  - Offer support and guidance to assist the development team in addressing identified vulnerabilities.
  - Collaborate with stakeholders to ensure effective implementation of recommended security enhancements.
  - Provide ongoing assistance and consultation to maintain the security of the smart contract codebase.

ChainAudits

General Findings - Type: ERC-20

Can the owner mint tokens arbitrarily?

The owner may have the ability to mint new tokens at any time, potentially diluting the token supply.

No

Can the owner burn tokens from any address without approval?

If the owner can burn tokens from user addresses without the holder's approval, it could lead to unjust token loss.

No

Can the owner pause or freeze token transfers?

The owner may have the authority to pause all transfers, which could prevent users from accessing their funds when needed.

No

Can the owner blacklist or whitelist addresses?

The owner may be able to control which addresses can hold or transfer tokens, potentially leading to censorship or arbitrary exclusion of participants.

No

Are there any integer overflow or underflow vulnerabilities?

The contract should ensure that all arithmetic operations are safe from overflow or underflow issues, especially if using older Solidity versions where these checks are not automatic.

No

Is there improper handling of Approvals?

The approve function should properly mitigate risks of double-spending when allowance is modified and then used in separate transactions.

No

Disclaimer

ChainAudits reports should not be construed as an endorsement or disapproval of any particular business or group. These reports do not reflect the economic value of any of the products or assets developed by the group. Also, ChainAudits does not consider integration with external contracts or services (e.g., Unicrypt, Uniswap, PancakeSwap).

ChainAudits reports aim to identify successful audit processes to help our clients improve the quality of the code and manage the risks associated with cryptographic tokens and blockchain technology. It is important to understand that blockchain technology and cryptographic assets pose significant ongoing risks. Each company and individual should conduct its own due diligence to maintain consistent safety measures. ChainAudits makes no representations about the security or performance of the technologies we audit.

ChainAudits does not provide any warranty or guarantee that the analyzed technology is completely defect-free, nor does it imply approval by the technology's owners. These audits should not be used to make input or output decisions; they will be involved in any project. They are not giving financial advice and should not be construed as such.