DeFi Liquidity Zapper

8/28/2025 • 1 files • 12 issues found

After a professional audit?

⚠️ **Disclaimer:** This AI-generated scan may contain inaccuracies and false positives. For professional audits, consult Hashlock security audit experts.

Contract Overview

Description

LiquidityZapETH is a smart contract that facilitates one-click liquidity provision to Uniswap V2 pools. It accepts ETH deposits, allocates 10% to a treasury address, and uses the remaining 90% to create liquidity positions in OEC/ETH pairs. The contract optimally splits the ETH between swapping for tokens and providing liquidity to minimize slippage and maximize LP token output.

Access Control

Ownable

Privileged Roles

owner

External Calls

IUniswapV2Router02 IUniswapV2Factory IUniswapV2Pair IERC20 (OEC Token) WETH

External Systems

Uniswap V2 DEX Treasury Address

View Call Graph

Critical	0
High	1
Medium	3
Low	5

Gas 2

Informational 1

High Severity 1

Treasury ETH Transfer Can Cause Complete Transaction Rever... HIGH

The zapETH function uses a low-level call to send ETH to the treasury address without proper error handling. If the treasury is a contract that reverts on receive (e.g., due to gas limit, logic failure, or malicious behavior), the entire zap transaction will revert, causing users to lose gas and preventing them from providing liquidity. This creates a denial-of-service vector where a compromised or malfunctioning treasury can brick the entire protocol.

View Details

Medium Severity 3

Optimal Swap Calculation Can Return Zero Leading to Ineffic... MEDIUM

The _optimalSwapAmt function can return 0 when the mathematical formula produces a negative or zero result (when b <= c). In this case, the contract falls back to a naive 50/50 split which is suboptimal and results in worse price impact and less LP tokens minted. This particularly affects small deposits or edge cases in pool reserves.

View Details

Timelock Can Be Bypassed Through Ownership Transfer MEDIUM

While treasury updates have a 12-hour timelock, the owner can instantly transfer ownership to a new address that could then queue another treasury change. This allows circumventing the timelock's protective purpose through a shell game of ownership transfers.

View Details

Slippage Calculation Based on Stale Reserves MEDIUM

The contract calculates expected output and slippage protection based on reserves queried at the beginning of the transaction. However, these reserves can change between the query and actual swap execution due to other transactions in the same block, leading to unexpected slippage or failed transactions.

View Details

Low Severity 5

Incorrect Error Message on Treasury Transfer Failure

LOW

When the treasury ETH transfer fails, the contract reverts with 'BadRouterRefund()' error instead of a more appropriate error. This misleading error message could confuse users and developers debugging failed transactions, as the failure is related to treasury transfer, not router refunds.

View Details

MIN_ETH_FOR_ZAP Too Low Allows Dust Griefing

LOW

The MIN_ETH_FOR_ZAP constant is set to only 2 wei, which is effectively no minimum. This allows griefing attacks where attackers can spam the contract with dust amounts that still emit events and consume gas but provide negligible liquidity. Additionally, such small amounts can cause precision loss in calculations and potentially revert due to Uniswap's minimum liquidity requirements.

View Details

Receive Function Allows Unrestricted ETH From WETH Contract

LOW

The receive function accepts ETH from both the router and WETH contracts. However, WETH can be called by anyone to unwrap ETH to any address. An attacker could unwrap WETH to this contract, potentially interfering with accounting or causing unexpected behavior when the contract calculates leftover balances.

View Details

No Validation of Router Liquidity Return Values Beyond Minimu...

LOW

The contract checks that returned amounts meet minimum thresholds but doesn't validate that the router actually used all provided tokens/ETH efficiently. If the router has a bug or is upgraded maliciously, it could return valid but suboptimal amounts while keeping the difference.

View Details

No Mechanism to Recover Accidentally Sent Tokens

LOW

If users accidentally send ERC20 tokens directly to the contract (not through zapETH), there's no way to recover them. While the contract is designed to handle

ETH and specific token interactions, it lacks a recovery mechanism for mistakenly sent tokens.

View Details

Gas Severity 2

Gas Optimization: Redundant Balance Check for Treasury Paym... GAS

The contract reads address(this).balance to calculate leftover ETH after all operations, but this value is always known to be 0 or a refund amount since the contract doesn't hold ETH otherwise. The balance check adds unnecessary gas cost.

View Details

Sqrt Function Can Be Optimized Using Solidity 0.8 Features GAS

The custom _sqrt implementation uses a Newton-Raphson method that could be optimized using more modern Solidity patterns or replaced with a more gas-efficient implementation.

View Details

Informational Severity 1

Professional Benefits

Manual code review by security experts

Detailed remediation guidance

Compliance-ready reports

Ongoing security partnership

Get Professional Audit

Learn More

Developed by

Built in Collaboration with