5th APRIL 2021

SMART CONTRACT AUDIT REPORT

-

version v2.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

1 Issues (0 Critical, 1 Major, 0 Minor) Found

About HAECHI AUDIT

HAECHI AUDIT is a global leading smart contract security audit and development firm operated by HAECHI LABS. HAECHI AUDIT consists of professionals with years of experience in blockchain R&D and provides the most reliable smart contract security audit and development services.

So far, based on the HAECHI AUDIT's security audit report, our clients have been successfully listed on the global cryptocurrency exchanges such as Huobi, Upbit, OKEX, and others.

Our notable portfolios include SK Telecom, Ground X by Kakao, and Carry Protocol while HAECHI AUDIT has conducted security audits for the world's top projects and enterprises.

Trusted by the industry leaders, we have been incubated by Samsung Electronics and awarded the Ethereum Foundation Grants and Ethereum Community Fund.

Contact : audit@haechi.io
Website : audit.haechi.io

01. Introduction

This report was written to provide a security audit for the WanSwap smart contract. HAECHI AUDIT conducted the audit focusing on whether WanSwap smart contract is designed and implemented in accordance with publicly released information and whether it has any security vulnerabilities.

The issues found are classified as CRITICAL, MAJOR, MINOR or TIPS according to their severity.

CRITICAL	Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage.
MAJOR	Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed.
MINOR	Minor issues are some potential risks that require some degree of modification.
TIPS	Tips could help improve the code's usability and efficiency

HAECHI AUDIT advises addressing all the issues found in this report.

02. Summary

The code used for the audit can be found at GitHub

- https://github.com/wanswap/wanswap-contracts/tree/master/contracts
  - Commit Hash : 17bcf096277e6763ff3c61f8fd6f0535dabf7918
  - v2.0 Hash: aed10509507259825429eb83736990689742b278

Issues

HAECHI AUDIT has 0 Critical Issues, 1 Major Issues, and 0 Minor Issue; also, we included 0 Tip category that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
MAJOR	UniswapV2Router01#getAmountIn() returns inappropriate value.	(Found - v1.0) (Resolved - v2.0)
Notice	UniswapV2Pair#burn() is likely to consume unnecessary gas.	(Found - v1.0) (Acknowledged - v2.0)

Update

[v2.0] - WanSwap has confirmed that 1 notice Acknowledged, and 1 issue is resolved.

03. Overview

Contracts Subject to Audit

- Multicall.sol
- UniswapV2ERC20.sol
- UniswapV2Factory.sol
- UniswapV2Migrator.sol
- UniswapV2Pair.sol
- UniswapV2Router01.sol
- UniswapV2Router02.sol
- libraries
    - Babylonian.sol
    - FixedPoint.sol
    - Math.sol
    - SafeMath.sol
    - TransferHelper.sol
    - UQ112x112.sol
    - UniswapV2Library.sol
    - UniswapV2OracleLibrary.sol

Notice

- **UniswapV2Pair#burn() is likely to consume unnecessary gas.**

  When a user calls the burn function in pairs that the totalSupply is zero, the operation of divide by zero is performed in the calculation process of amount0, amount1.

  When divided by zero, the whole transaction will be reverted and uses the gas left for the transaction. This can easily be prevented by using SafeMath. But, when using SafeMath, additional opcodes are added as a result, and more gas is consumed in normal operation.

  So this is a trade off between gas cost of normal behavior and abnormal behavior. Which is why HAECHI AUDIT does not recommend to apply the SafeMath for contract. But users should be noticed that this issue exists while interacting with the contract, especially when interacting through other contracts.

Update

WanSwap team has confirmed that this issue is acknowledged.

04. Issues Found

MAJOR : UniswapV2Router01#getAmountIn() returns inappropriate value. (Found - v.1.0) (Resolved - v2.0)

MAJOR

```
285.    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) public pure override returns
        (uint amountIn) {
286.        return UniswapV2Library.getAmountOut(amountOut, reserveIn, reserveOut);
287.    }
288.
```

Problem Statement

UniswapV2Router01#getAmountIn() is pure function which should return UniswapV2Library#getAmountIn() value. But according the code, UniswapV2Router01#getAmountIn() returns UniswapV2Library#getAmountOut() value.

The UniswapV2 contract contains the same issue[1]. So they updated the code and now use router02 instead of router01.

Recommendation

Like Uniswap, using router02 instead of router01 is recommended.

Update

WanSwap team has fixed this issue by deleting UniswapV2Router01 contract from wanswap project.

[1] https://uniswap.org/docs/v2/smart-contracts/router01/#getamountin

05. Disclaimer

This report is not an advice on investment, nor does it guarantee adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. The code may include problems on Ethereum that are not included in this report. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.

Appendix A. Test Results

The following are the results of a unit test that covers the major logics of the smart contract under audit. The parts in red contain issues and therefore have failed the test.

```
Babylonian
 sqrt()
  ✓ should return 0 when input is zero
  ✓ should return 1 when input < 3 (45ms)
  ✓ should return appropriate value when input > 3 (47ms)

FixedPoint
 encode()
  ✓ should return 0 when input is zero
  ✓ will not take >uint112(-1)
 #encode144()
  ✓ shifts left by 112
  ✓ will not take >uint144(-1)
 #decode()
  ✓ shifts right by 112
  ✓ will not take >uint224(-1)
 #decode144()
  ✓ shifts right by 144
  ✓ will not take >uint256(-1)
 #mul()
  ✓ works for 0
  ✓ correct multiplication
  ✓ overflow
  ✓ max without overflow, largest fixed point
 #div()
  ✓ works for 0 numerator
  ✓ throws for 0 denominator
  ✓ equality 30/30
  ✓ divides 30/10
 #fraction
  ✓ correct computation less than 1
  ✓ correct computation greater than 1
  ✓ fails with 0 denominator

Math
 min()
```

```
    ✓ should return min value
  sqrt()
    ✓ should return appropriate value

UniswapV2OracleLibrary
  currentBlockTimestamp()
    ✓ should return appropriate value
  currentCumulativePrices()
    ✓ should update price0Cumulative
    ✓ should update price1Cumulative
    ✓ should update blockTimestamp

Multicall
  getEthBalance()
    ✓ should return address's ether balance
  getLastBlockHash()
    ✓ should return blockHash
  getCurrentBlockTimestamp()
    ✓ should return block.timestamp
  getCurrentBlockDifficulty()
    ✓ should return block.difficulty
  getCurrentBlockGasLimit()
    ✓ should return block.gaslimit
  getCurrentBlockCoinbase()
    ✓ should return block.coinbase
  aggregate()
    ✓ should success calling

UniswapV2ERC20
  mint()
    ✓ should increase total supply
    ✓ should increase user balance
  burn()
    ✓ should decrease total supply
    ✓ should increase user balance
  approve()
    ✓ should increase allowance
  transfer()
    ✓ should decrease sender balance
    ✓ should increase recipient balance
  transferFrom()
    ✓ should fail when not approved sufficient amount
```

```
    valid case
      ✓ should decrease sender balance
      ✓ should increase recipient balance
      ✓ should decrease allowance
      ✓ should not decrease allowance when max approve
permitERC20
  ✓ permit (72ms)

UniswapV2Factory
  constructor()
    ✓ should set feeToSetter
  setFeeTo()
    ✓ should fail when msg.sender is not feeToSetter
    ✓ should set feeTo
  setFeeToSetter()
    ✓ should fail when msg.sender is not feeToSetter
    ✓ should set feeTo
  createPair()
    ✓ should fail when tokenA = tokenB
    ✓ should fail when tokenA is address(0)
    ✓ should fail when tokenB is address(0)
    ✓ should fail when pair already exist
    valid case
      ✓ should make pair
      ✓ should emit PairCreated event

UniswapV2Migrator
  migrate()
    ✓ should transfer exchangeToken to contract (54ms)
    ✓ should transfer token to msg.sender when amountTokenV1 > V2 (64ms)

UniswapV2Pair
  constructor()
    ✓ should set factory
  initialize()
    ✓ should fail when msg.sender is not factory
    valid case
      ✓ should set token0
      ✓ should set token1
  update()
    ✓ should fail when balance is over max uint112
    valid case
```

```
    ✓ should emit Sync event
  mintFee()
   CASE : feeTo is address(0)
    ✓ should set kLast zero
   CASE : feeTo is not address(0)
    ✓ should mint fee token to feeTo
  mint()
   CASE : totalSupply is zero
    ✓ should mint token to address(to) (54ms)
    ✓ should update reserve (50ms)
    ✓ should update kLast (54ms)
    ✓ should emit Mint event (49ms)
   CASE : totalSupply is not zero
    ✓ should mint token to address(to) (47ms)
    ✓ should update reserve (45ms)
    ✓ should update kLast (46ms)
    ✓ should emit Mint event (43ms)
  burn()
   ✓ should fail when amount0 and 1 is zero (65ms)
   ✓ should burn token (73ms)
   ✓ should transfer token0 (76ms)
   ✓ should transfer token1 (70ms)
   ✓ should update reserve (58ms)
   ✓ should update kLast (86ms)
   ✓ should emit Burn event (61ms)
  swap()
   ✓ should fail when amount0 and 1 is zero
   ✓ should fail when amount > reserve
   ✓ should fail when address(to) is token 0
   valid case
    ✓ should transfer token0 (69ms)
    ✓ should transfer token1 (79ms)
    ✓ should call uinswapV2Call (65ms)
    ✓ should update reserve (56ms)
    ✓ should emit Swap event (57ms)
  skim()
   ✓ should transfer token0 (50ms)
   ✓ should transfer token1 (44ms)
  sync()
   ✓ should update reserve

UniswapV2Router01
```

```
constructor()
  ✓ should set factory
  ✓ should set WETH
addLiquidity()
  CASE : 1st add -> no reserve
    ✓ should transfer tokenA to pair contract
    ✓ should transfer tokenB to pair contract
  CASE : 2nd add -> optimal <= desired
    ✓ should transfer tokenA to pair contract
    ✓ should transfer tokenB to pair contract
  CASE : 2nd add -> optimalB > desiredB
    ✓ should transfer tokenA to pair contract
    ✓ should transfer tokenB to pair contract
addLiquidityETH()
  ✓ should transfer tokenA to pair contract
  ✓ should transfer weth to pair contract
  ✓ should transfer weth back to msg.sender when rest (100ms)
removeLiquidity()
  ✓ should transfer liquidity to pair (66ms)
  ✓ should fail when amountA < min (64ms)
  ✓ should fail when amountB < min (70ms)
removeLiquidityETH()
  ✓ should withdraw weth
  ✓ should transfer tokenA
swapExactTokensForTokens()
  ✓ shoud transfer tokenA (82ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapTokensForExactTokens()
  ✓ shoud transfer tokenA (85ms)
  ✓ should fail when amounts[0] > amountInMax
swapExactETHForTokens()
  ✓ shoud transfer weth (83ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapTokensForExactETH()
  ✓ shoud transfer tokenA (87ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapExactTokensForETH()
  ✓ shoud transfer tokenA (96ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapETHForExactTokens()
  ✓ shoud transfer tokenA (91ms)
  ✓ should fail when amounts[-1] < amountOutMin
```

```
  quote()
    ✓ should return uniswap.quote
  getAmountOut()
    ✓ should return uniswap.getAmountOut
  getAmountIn()
    1) should return uniswap.getAmountIn
  getAmountsOut()
    ✓ should return uniswap.getAmountsOut
  getAmountsIn()
    ✓ should return uniswap.getAmountsIn

permitRouter01
  ✓ removeLiquidityWithPermit (205ms)
  ✓ removeLiquidityETHWithPermit (227ms)

permitRouter02
  ✓ removeLiquidityWithPermit (268ms)
  ✓ removeLiquidityETHWithPermit (242ms)
  ✓ removeLiquidityETHWithPermitSupportingFeeOnTransferTokens (229ms)

UniswapV2Router02
  constructor()
    ✓ should set factory
    ✓ should set WETH
  addLiquidity()
    CASE : 1st add -> no reserve
      ✓ should transfer tokenA to pair contract
      ✓ should transfer tokenB to pair contract
    CASE : 2nd add -> optimal <= desired
      ✓ should transfer tokenA to pair contract
      ✓ should transfer tokenB to pair contract
    CASE : 2nd add -> optimalB > desiredB
      ✓ should transfer tokenA to pair contract
      ✓ should transfer tokenB to pair contract
  addLiquidityETH()
    ✓ should transfer tokenA to pair contract
    ✓ should transfer weth to pair contract
    ✓ should transfer weth back to msg.sender when rest (96ms)
  removeLiquidity()
    ✓ should transfer liquidity to pair (81ms)
    ✓ should fail when amountA < min (70ms)
    ✓ should fail when amountB < min (67ms)
```

```
removeLiquidityETH()
  ✓ should withdraw weth
  ✓ should transfer tokenA
removeLiquidityETHSupportingFeeOnTransferTokens()
  ✓ should withdraw weth
  ✓ should transfer tokenA
swapExactTokensForTokens()
  ✓ shoud transfer tokenA (78ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapTokensForExactTokens()
  ✓ shoud transfer tokenA (84ms)
  ✓ should fail when amounts[0] > amountInMax (40ms)
swapExactETHForTokens()
  ✓ shoud transfer weth (81ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapTokensForExactETH()
  ✓ shoud transfer tokenA (95ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapExactTokensForETH()
  ✓ shoud transfer tokenA (99ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapETHForExactTokens()
  ✓ shoud transfer tokenA (81ms)
  ✓ should fail when amounts[-1] < amountOutMin
swapExactTokensForTokensSupportingFeeOnTransferTokens()
  ✓ should transfer tokenA (79ms)
  ✓ should fail when tokenB balance < amountOutMin (107ms)
swapExactETHForTokensSupportingFeeOnTransferTokens()
  ✓ should transfer weth (79ms)
  ✓ should fail when tokenA balance < amountOutMin (76ms)
swapExactTokensForETHSupportingFeeOnTransferTokens()
  ✓ should transfer tokenA (89ms)
  ✓ should fail when tokenA balance < amountOutMin (89ms)
quote()
  ✓ should return uniswap.quote
getAmountOut()
  ✓ should return uniswap.getAmountOut
getAmountIn()
  ✓ should return uniswap.getAmountIn
getAmountsOut()
  ✓ should return uniswap.getAmountsOut
getAmountsIn()
```

✓ should return uniswap.getAmountsIn

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					
Multicall.sol	100	100	100	100	
UniswapV2ERC20.sol	100	100	100	100	
UniswapV2Factory.sol	100	100	100	100	
UniswapV2Migrator.sol	100	100	100	100	
UniswapV2Pair.sol	100	100	100	100	
UniswapV2Router01.sol	100	100	100	100	
UniswapV2Router02.sol	100	100	100	100	
contracts/libraries/					
Babylonian.sol	100	100	100	100	
FixedPoint.sol	100	100	100	100	
Math.sol	100	100	100	100	
SafeMath.sol	100	100	100	100	
TransferHelper.sol	100	100	100	100	
UQ112x112.sol	100	100	100	100	
UniswapV2Library.sol	100	100	100	100	
UniswapV2OracleLibrary.sol	100	100	100	100	

[Table 1] Test Case Coverage