QuillAudits

Audit Report
September, 2022

For

CRYPTOUNITY

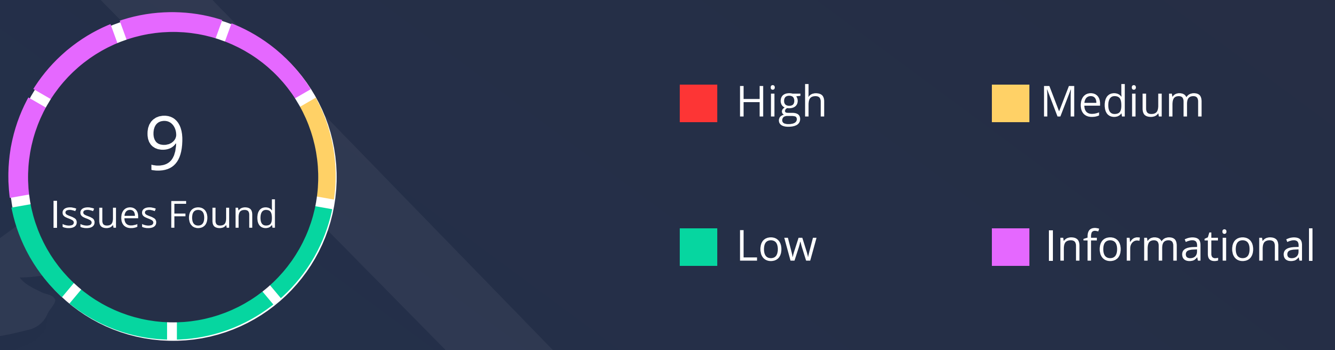
START SIMPLE, STAY SECURE

Table of Content

Table of Content

Executive Summary

Project Name	CryptoUnity
Timeline	12 July, 2022 to 22 September, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse CryptoUnity's smart contract for quality, security, and correctness. *https://github.com/srajca/CryptoUnity-SmartContractAudit/commit/001944ca04ab7860cc6534bbad283911fdbdc321* Commit hash: 001944ca04ab7860cc6534bbad283911fdbdc321
Fixed In	*https://github.com/srajca/CU-SmartContract/commit/309f115d85be18d362b8e75cbe08235cdc9f6596* **Commit hash:** 309f115d85be18d362b8e75cbe08235cdc9f6596



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	1	1
Resolved Issues	0	1	2	3

Types of Severities

High

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

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

Low-level severity issues can cause minor impact and or 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 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.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token 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 analysed 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

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 analysed, 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 behaviour 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, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

High Severity Issues

No issues found

Medium Severity Issues

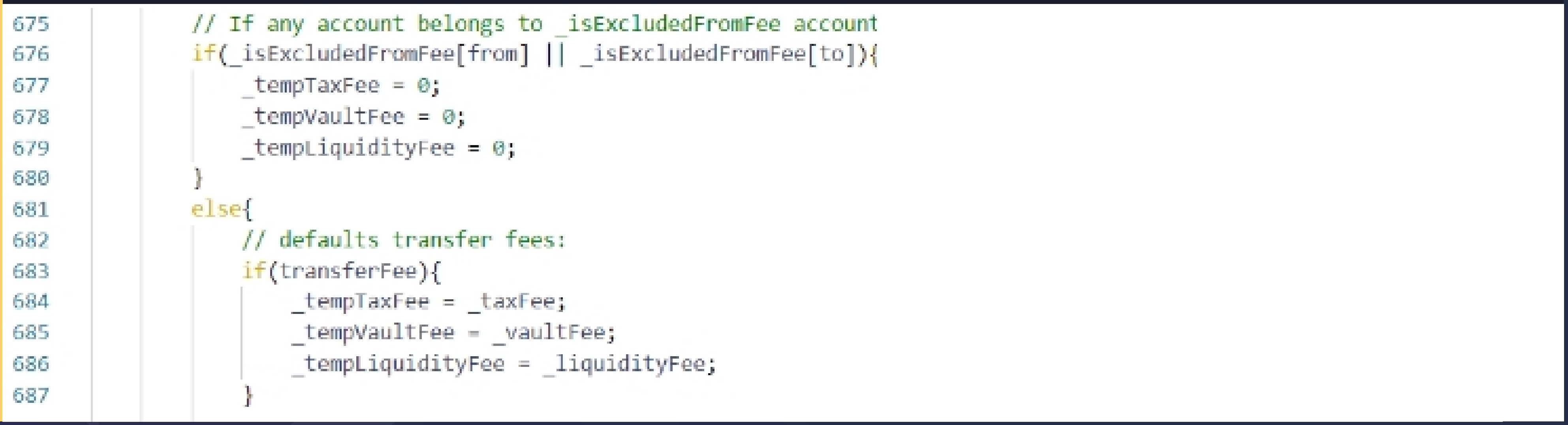
A.1 Re-initialize fee variables

Description

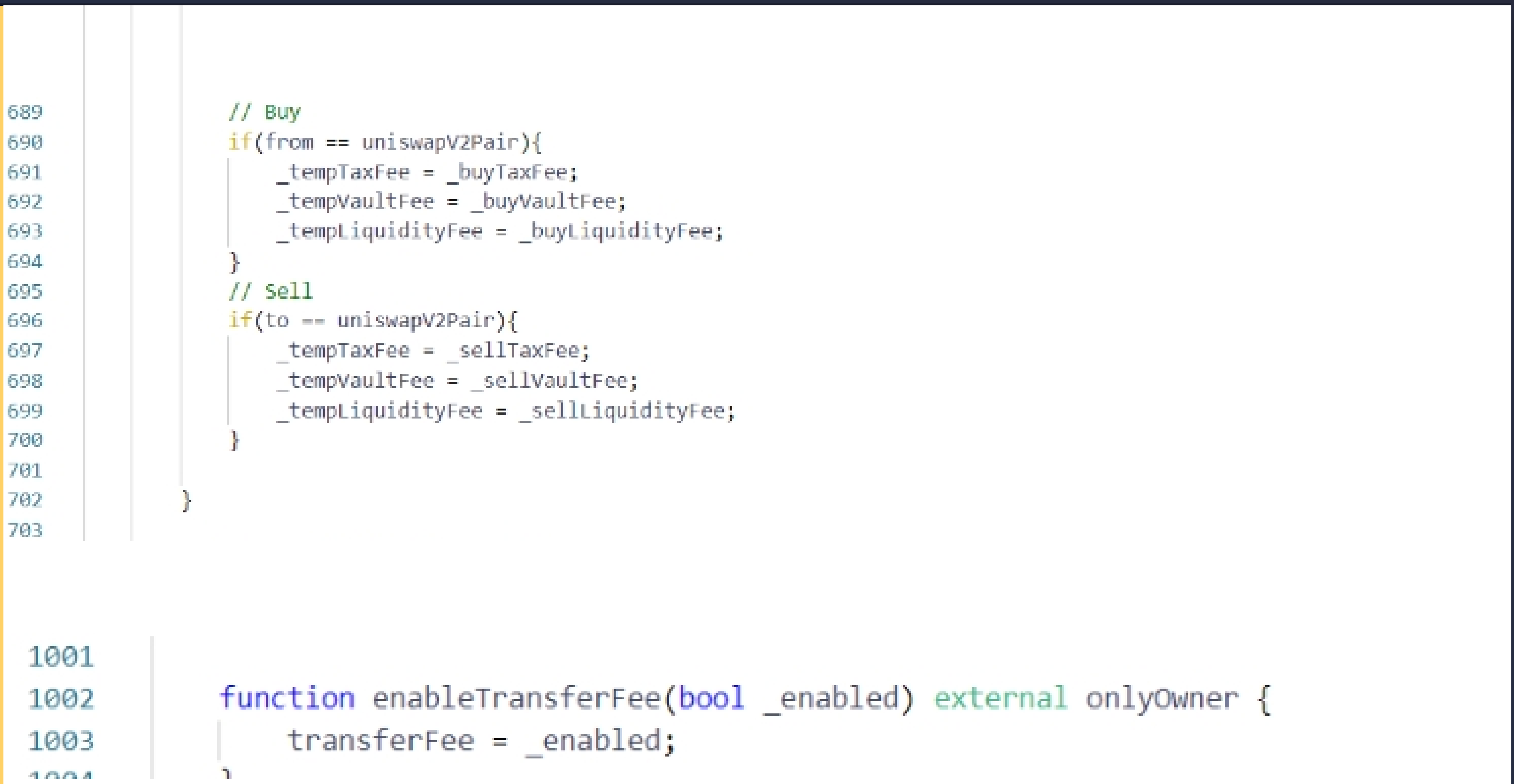
_tempTaxFee, _tempVaultFee, _tempLiquidityFee are the variables that are getting initialized in _transfer function on certain conditions. For some condition if owner sets transferFee to false then in some scenarios it will result in unwanted fee deductions for transfers, as described below:

Example

1) Transfer happens for accounts that are not excluded from fees.
2) In this transfer condition on the line 683 gets executed which assigns amount to _tempTaxFee, _tempVaultFee, _tempLiquidityFee variables.
3) Now the admin/owner sets transferFee to false using enableTransferFee(). Once transferFee is set to false then no fees should get deducted for next token transfers.
Now the next token transfer happens (for not excluded accounts) and it will still deduct the fees.
4) This happens because the enableTransferFee() is not setting these three state variables (_tempTaxFee, _tempVaultFee, _tempLiquidityFee) to 0 When setting transferFee to false.



```
675            // If any account belongs to _isExcludedFromFee account
676            if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
677                _tempTaxFee = 0;
678                _tempVaultFee = 0;
679                _tempLiquidityFee = 0;
680            }
681            else{
682                // defaults transfer fees:
683                if(transferFee){
684                    _tempTaxFee = _taxFee;
685                    _tempVaultFee = _vaultFee;
686                    _tempLiquidityFee = _liquidityFee;
687                }
```



```
        // Buy
        if(from == uniswapV2Pair){
            _tempTaxFee = _buyTaxFee;
            _tempVaultFee = _buyVaultFee;
            _tempLiquidityFee = _buyLiquidityFee;
        }
        // Sell
        if(to == uniswapV2Pair){
            _tempTaxFee = _sellTaxFee;
            _tempVaultFee = _sellVaultFee;
            _tempLiquidityFee = _sellLiquidityFee;
        }

    }
```

```

    function enableTransferFee(bool _enabled) external onlyOwner {
        transferFee = _enabled;
    }
```

Remediation

When setting transferFee to false using enableTransferFee() set these fee variables _tempTaxFee, _tempVaultFee, _tempLiquidityFee to 0.

Status

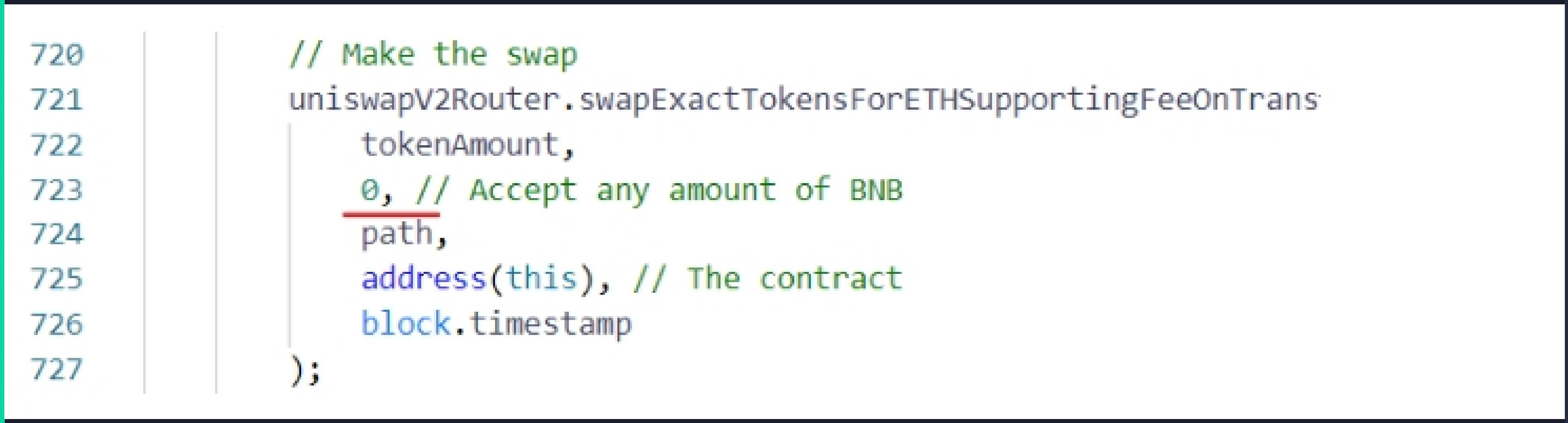
Resolved

Low Severity Issues

A2. Minimum amount to receive is 0:

Description

Minimum amount of output tokens that must be received is 0, which allows trade to execute even when the output amount is very low. This type of transaction can be sandwiched by a malicious user/attacker. Which results in pool amount manipulation which can cause the amount of output tokens that must be received to be low.



```
// Make the swap
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTrans
    tokenAmount,
    0, // Accept any amount of BNB
    path,
    address(this), // The contract
    block.timestamp
);
```

Remediation

Consider adding a minimum amount to receive greater than zero, The minimum amount to receive may vary according to The token amount passed in while swapping for ETH.

CryptoUnity team's comment: we dont want any user to complain on the failure of transaction during buy sell, it will in general create a panic in community, so setting the min swap to wei or even 0 will make sense and should not affect the security of the smart contract

Auditor's comment: In commit (309f115d85be18d362b8e75cbe08235cdc9f6596) CryptoUnity team has added changeMinTokenLiqSwap() function from which amount to received can be changed accordingly.

Status

Resolved

A.3 Centralization risk:

Description

addLiquidity() function calls addLiquidityETH() function in router contract with the parameter of lp tokens recipient set to owner address. With time the owner address may accumulate a significant amount of LP tokens which may be dangerous for token economics if an owner acts maliciously or its account gets compromised. This issue can be fixed by changing the recipient address to a smart contract account with enhanced security like Multi Signature wallets.

Remediation

Consider replacing to address the argument of the addLiquidityETH to multisignature wallet.

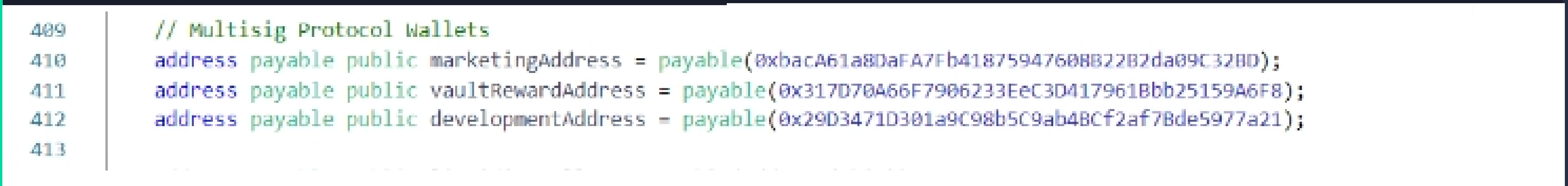
Status

Partially Resolved

A.4: Check hardcoded addresses

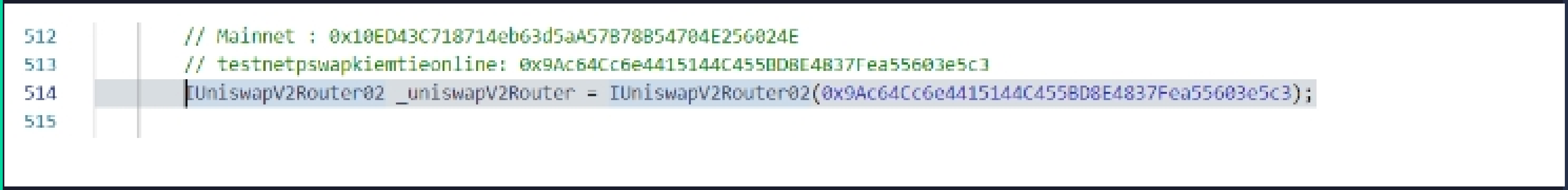
Description

L 410,411,412 shows hardcoded marketingAddress, vaultRewardAddress, developmentAddress respectively. Care needs to be taken while hardcoding addresses.



```
409    // Multisig Protocol Wallets
410    address payable public marketingAddress = payable(0xbacA61a8DaFA7Fb41875947608B22B2da09C328D);
411    address payable public vaultRewardAddress = payable(0x317D70A66F7906233EeC3D417961Bbb25159A6F8);
412    address payable public developmentAddress = payable(0x29D3471D301a9C98b5C9ab4BCf2af7Bde5977a21);
413
```

L 514 Shows hardcoded address PancakeRouter contract. This address is of PancakeRouter from testnet and not from mainnet.



```
512        // Mainnet : 0x10ED43C718714eb63d5aA57B78B54704E256024E
513        // testnetpswapkiemtieonline: 0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3
514        IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3);
515
```

Remediation

Check hardcoded addresses are correct.

Status

Acknowledged

A.5: Missing Testcases

Description

Test cases for the code and functions have not been provided.

Remediation

It is recommended to write test cases of all the functions. Any existing tests that fail must be resolved. Tests will help in determining if the code is working in the expected way. Unit tests should have been performed to achieve good test coverage across the entire codebase.

Status

Resolved

Informational Issues

A.6: Redundant Code

Description

Some unused functions, events and variables are given below:

Unused functions:
• L882 transferToAddressBNB() and L707 swapTokens() are private functions and not getting called by any of the function present in the contract. Hence remained unused.
• changeRouterVersion() and setRouterAddressAndCreatePair() are getting used for similar operations i.e to set new router address. (There's a slight difference in changeRouterVersion() which checks for the pair already exists or not and if not it creates it) , Our suggestion is to review business logic and to remove any one redundant function.

Unused functions:
•Events L478 RewardLiquidityProviders , L480 SwapAndLiquifyBNB , L491 SwapBNBForTokens are unused.

Unused Variables:
•_sellLiqFee and _buyLiqFee variables are getting assigned in setReawardMarketingDevFee() on L1041 and L1047 but never getting used.
• _liqFee is getting assigned in setTransferFee() on L1056 but never getting used.

Remediation

Review logic and remove unused code statements/functions/events.

Status

Partially Resolved

CryptoUnity team's comment for "unused variables": These are user information variables for reading from the contract and using it in future applications UI display.

A.7: Using SafeMath with SOL 0.8

Description

The contract is using SafeMath for uint256 however the sol version that is getting used is 0.8 where it gives protection against underflow and overflow conditions. Using double checks can increase execution cost for transaction.

Remediation

Consider Removing SafeMath in the current case. Care needs to be taken while removing use of SafeMath as many lines of code are relying on it.

Status

Resolved

Auditor's comment: safeMath implementation for sol version 0.8 or later is getting used.

A.7: Using SafeMath with SOL 0.8

Description

using directive is used to attach functions from Address library to address type on the L408 but functions from this lirbary are never used.

Remediation

Consider removing import statement from the contract if library is not getting used

Status

Resolved

A.9: Floating pragma

Description

The contract is using floating pragma ^0.8.16 , Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version get selected while deploying contract which has higher chances of having bugs in it.

Remediation

Lock the compiler version that is chosen.

Status

Resolved

Functional Testing

- ✓ Should be able to revert on transfers to dead address
- ✓ Should be able to set a new pair address
- ✓ Should be able to set new router address
- ✓ Should be able to exclude address from reward
- ✓ Should be able to include address in reward
- ✓ Should be able to exclude address from fee
- ✓ Should be able to include address in fee
- ✓ Should be able to set buyMaxTxAmount
- ✓ Should be able to set sellMaxTxAmount
- ✓ Should be able to set marketing address
- ✓ Should be able to set development address
- ✓ Should be able to set reward address
- ✓ Should be able to set RewardMarketingDevFee
- ✓ Should be able to enable TransferFee
- ✓ Should be able to increase and decrease allowances
- ✓ Should be able to addLiquidity for from accumulated tokens

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

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the CryptoUnity Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the CryptoUnity Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

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.

600+
Audits Completed

$15B
Secured

600K
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2022

For

CRYPTOUNITY

START SIMPLE, STAY SECURE

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com