Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Otcom Token
Platform: Ethereum
Language: Solidity
Date: September 28th, 2024

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by Otcom Token to perform the Security audit of the Otcom Token smart contract code. The audit was performed using manual analysis and automated software tools. This report presents all the findings regarding the audit performed on September 28th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

The contract provided implements a custom ERC20 token called "Otcom" (OTOM) with additional features such as liquidity provision, tax mechanisms, and trading restrictions. Here's a summary of its key functionalities:

Key Features:

- **Tax Mechanism:**
  - A configurable tax mechanism with separate percentages for liquidity tax and development tax.
  - The taxes are deducted during buy/sell transactions, collected in the contract, and used for adding liquidity and funding the development wallet.
- **Trading Restrictions:**
  - Blacklist Protection: The contract has a feature to blacklist addresses for a certain number of blocks after trading is enabled, primarily to combat bots.
  - Maximum Transaction Amount: A configurable limit on the maximum number of tokens that can be transferred in a single transaction.
  - Trade Control: Trading can only be enabled by the owner of the contract.
- **Liquidity Management:**
  - The contract automatically manages liquidity by adding liquidity to a Uniswap pair when a threshold is reached.
  - The `swapAndLiquify` function splits tokens, converts half to ETH, and adds liquidity on Uniswap. It also transfers part of the collected ETH to the development wallet.

- **Token Transfers:**
  - Standard ERC20 functionality with added mechanisms for tax deductions during buys/sells.
  - Internal transfer function: Implements additional checks, including restrictions on blacklisted addresses and trading status.
- **Ownership & Configuration:**
  - Only the owner can modify certain contract parameters, like the dev wallet, tax percentages, maximum transaction amount, and blacklist settings.

The contract is designed to be used as a token with tax and liquidity management for decentralized exchanges like Uniswap.
It can be suitable for projects aiming to ensure liquidity and development funding through every transaction.

Audit scope

Name	**Code Review and Security Analysis Report for Otcom Token Smart Contract**
Platform	**Ethereum / Solidity**
File	OTCOMToken.sol
GitHub Commit Hash	b2cbfd317b6b2182d817bf6b6b8bc8f0c05c0e4e
Updated GitHub Commit Hash	6eea7827826c9ff104e7ac0c5c63ee70f948686a
Audit Date	September 28th, 2024
Revised Audit Date	October 17th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Token Details: • Name: Otcom • Symbol: OTOM • Decimals: 18 • Total Supply: 10 million tokens	**YES, This is valid.**
Tax and Limits: • **Liquidity Tax:** 1% (1000 basis points) for liquidity purposes. • **Dev Tax:** 1% (1000 basis points) for development purposes. • **Tax Threshold:** Minimum token amount (10,000) required to trigger tax functions. • **Maximum Amount:** Maximum buy/sell limit set to 20,000 tokens. • **Number of Blocks For Blacklist:** Restriction on trades within a set number of blocks.	**YES, This is valid.**
The owner has several administrative functions: • Enable trading. • Set blacklist duration for sniper bots. • Adjust maximum transaction limits. • Update development wallet. • Change tax thresholds and percentages. • The current owner can transfer ownership of the contract to a new account. • Deleting ownership will leave the contract without an owner, removing any owner-only functionality.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, Customer`s solidity-based smart contracts are **“secured”**. This token contract does contain owner control, which does not make it fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 3 very low-level issues.

We confirm that all the issues are fixed.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Email: audit@EtherAuthority.io

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
Buy Tax	2%
Sell Tax	2%
Cannot Buy	No
Cannot Sell	No
Max Tax	2%
Modify Tax	Yes
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	Not Detected
Max Transaction amount?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	No
Can Mint?	No
Is it a Proxy?	No
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in the Otcom Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Otcom Token.

The Otcom Token team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

Documentation

We were given an Otcom Token smart contract code in the form of a GitHub weblink.

As mentioned above, the code parts are well commented on. And the logic is straightforward. So, it is easy to understand the programming flow and complex code logic quickly. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	name	read	Passed	No Issue
3	symbol	read	Passed	No Issue
4	decimals	read	Passed	No Issue
5	totalSupply	read	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	burn	write	access only owner	No Issue
8	_burn	internal	Passed	No Issue
9	transfer	write	Passed	No Issue
10	transferFrom	write	Passed	No Issue
11	allowance	read	Passed	No Issue
12	approve	write	Passed	No Issue
13	_approve	internal	Passed	No Issue
14	_spendAllowance	internal	Passed	No Issue
15	_transferTokens	internal	Passed	No Issue
16	setDevWallet	external	access only owner	No Issue
17	setTaxPercentage	external	access only owner	No Issue
18	setTaxThreshold	external	access only owner	No Issue
19	recoverETHfromContract	external	-	Removed
20	swapTokensForEth	write	Passed	No Issue
21	swapAndLiquify	internal	Passed	No Issue
22	addLiquidity	write	Passed	No Issue
23	_transfer	internal	Passed	No Issue
24	_calculateTax	internal	Passed	No Issue
25	fallback	external	-	Removed
26	receive	external	Passed	No Issue
27	onlyOwner	modifier	Passed	No Issue
28	owner	read	Passed	No Issue
29	_checkOwner	internal	Passed	No Issue
30	renounceOwnership	write	access only owner	No Issue
31	transferOwnership	write	access only owner	No Issue
32	_transferOwnership	internal	Passed	No Issue
33	enableTrade	write	access only owner	No Issue
34	setNumberOfBlocksForBlacklist	external	access only owner	No Issue
35	setMaxAmount	external	access only owner	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No critical severity vulnerabilities were found.

High Severity

No high-severity vulnerabilities were found.

Medium

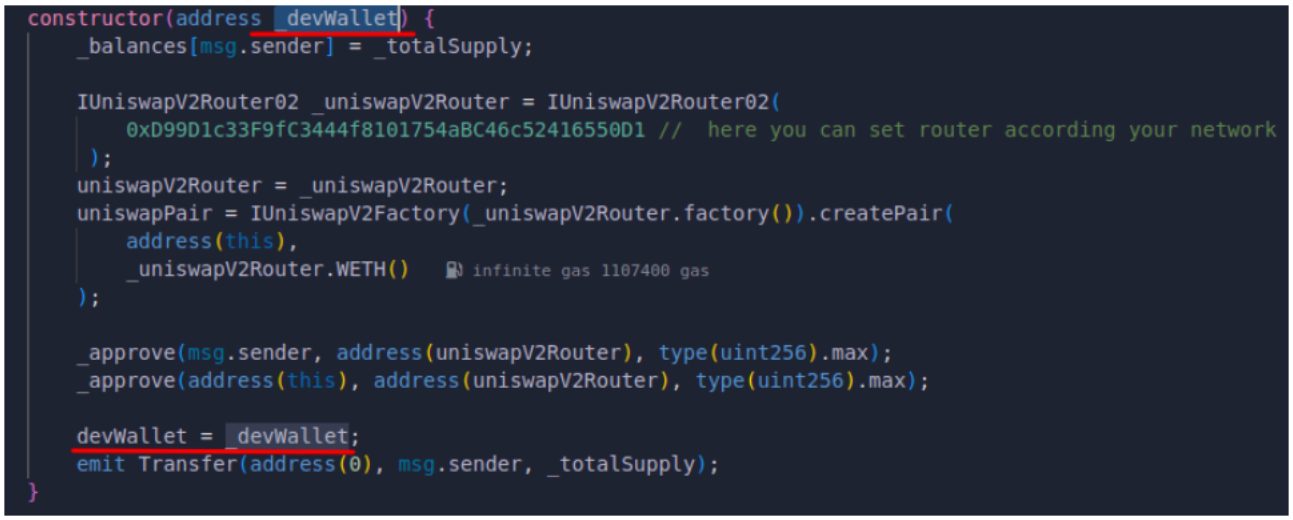
No medium-severity vulnerabilities were found.

Low

No low-severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Function input parameters lack of check:



```
constructor(address _devWallet) {
    _balances[msg.sender] = _totalSupply;

    IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
        0xD99D1c33F9fC3444f8101754aBC46c52416550D1 //  here you can set router according your network
    );
    uniswapV2Router = _uniswapV2Router;
    uniswapPair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(
        address(this),
        _uniswapV2Router.WETH()    infinite gas 1107400 gas
    );

    _approve(msg.sender, address(uniswapV2Router), type(uint256).max);
    _approve(address(this), address(uniswapV2Router), type(uint256).max);

    devWallet = _devWallet;
    emit Transfer(address(0), msg.sender, _totalSupply);
}
```

Constructor parameter _devWallet requires validation before execution.

Resolution: We suggest using validation, like for numerical variables that should be greater than 0, and for address-type check variables that are not addressed (0). For percentage-type variables, values should have some range, like a minimum of 0 and a maximum of 100.

Status: **Fixed**

(2) Duplicate events define:

```
//events
event Transfer(address indexed from, address indexed to, uint256 value);
event Approval(address indexed owner, address indexed spender, uint256 value);
```

```
event Transfer(address indexed from, address indexed to, uint256 value);

/**
 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
 * a call to {approve}. `value` is the new allowance.
 */
event Approval(
    address indexed owner,
    address indexed spender,
    uint256 value
);
```

Transfer and Approval events are defined twice.

Resolution: Please remove duplicate events defined.

Status: **Fixed**

(3) Set tax Percentage Limit:

```
function setTaxPercentage(uint256 _taxPercentage) external onlyOwner {
    require(_taxPercentage <= 100000, "Tax percentage cannot exceed 100%");

    // Split the total tax percentage equally between liquidityTax and devTax
    liquidityTaxPercentage = _taxPercentage / 2;
    devTaxPercentage = _taxPercentage / 2;

    totalTaxPercentage = _taxPercentage;
    emit UpdatedTaxPercentage(totalTaxPercentage,liquidityTaxPercentage,devTaxPercentage);
}
```

The logic in the code is valid, as it correctly calculates the liquidityTaxPercentage and devTaxPercentage based on the _taxPercentage. If the total tax is 100%, then the entire amount will be distributed between these two taxes, leaving nothing for the buyer.

Resolution: It's important to note that setting the tax percentage to 100% might not be desirable in most scenarios, as it effectively prevents buyers from acquiring tokens. It's generally recommended to set a reasonable tax percentage that allows for both token distribution and project funding.

Status: **Fixed**

Centralization

This smart contract has some functions that can only be executed by the Admin (Owner). If the admin wallet's private key is compromised, then it usually creates trouble. The following are Admin functions:

OTCOMToken.sol

- burn: Allows the contract owner to burn a specific amount of tokens from the caller's address.
- enableTrade: Allows the contract owner to enable trading by setting the `tradeOpen` flag to true.
- setNumberOfBlocksForBlacklist: Allows the contract owner to set the number of blocks during which sniper bot protection is active.
- setMaxAmount: Allows the contract owner to set the maximum transaction amount.
- setDevWallet: Allows the contract owner to set a new development wallet address.
- setTaxPercentage: Allows the contract owner to set the total tax percentage for the contract.
- setTaxThreshold: Allows the contract owner to set the minimum token threshold for tax collection.

Ownable.sol

- renounceOwnership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code as a GitHub weblink, and we used all possible tests based on the given objects. We have observed 3 very low-severity issues. We confirm that all smart contract issues are fixed. **So, the smart contract is ready for mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

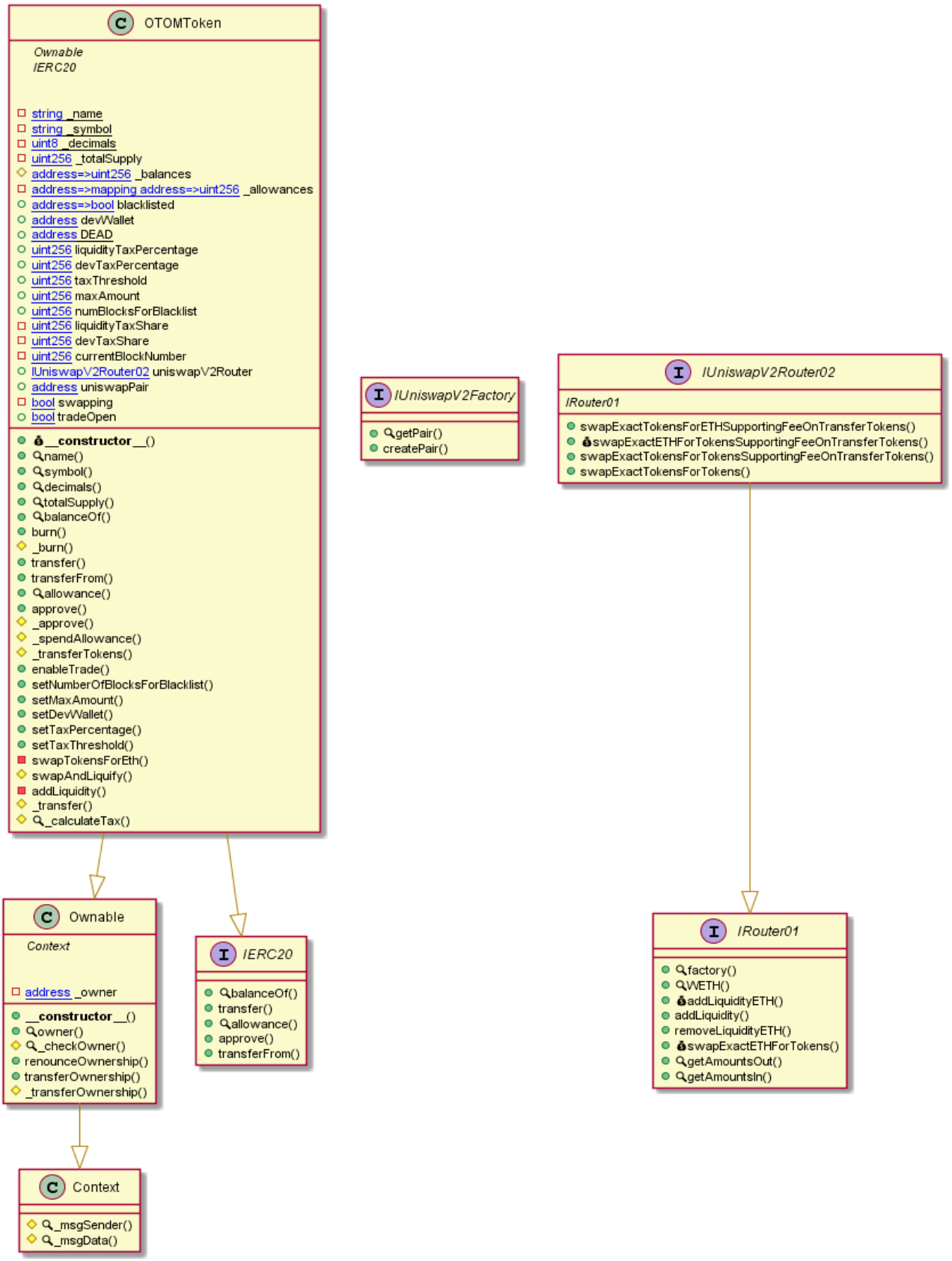
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best to conduct the analysis and produce this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Otcom Token



Slither Results Log

Slither Log >> OTCOMToken.sol

```
INFO:Detectors:
OTCOMToken.constructor(address)._devWallet (OTCOMToken.sol#311) lacks a zero-check on :
        - devWallet = _devWallet (OTCOMToken.sol#326)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in OTCOMToken.swapAndLiquify() (OTCOMToken.sol#628-654):
    External calls:
    - swapTokensForEth(tokensToSwap) (OTCOMToken.sol#638)
        -
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,pat
h,address(this),block.timestamp) (OTCOMToken.sol#609-615)
    - (success1,None) = devWallet.call{gas: 35000,value: devAmount}()
(OTCOMToken.sol#647)
    - addLiquidity(otherLiqHalf,newBalance) (OTCOMToken.sol#650)
        - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp)
(OTCOMToken.sol#670-677)
    External calls sending eth:
    - (success1,None) = devWallet.call{gas: 35000,value: devAmount}()
(OTCOMToken.sol#647)
    - addLiquidity(otherLiqHalf,newBalance) (OTCOMToken.sol#650)
        - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp)
(OTCOMToken.sol#670-677)
    State variables written after the call(s):
    - addLiquidity(otherLiqHalf,newBalance) (OTCOMToken.sol#650)
        - _allowances[sender][spender] = amount (OTCOMToken.sol#477)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
OTCOMToken.swapAndLiquify() (OTCOMToken.sol#628-654) tries to limit the gas of an external
call that controls implicit decoding
    (success1,None) = devWallet.call{gas: 35000,value: devAmount}() (OTCOMToken.sol#647)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#return-bomb
INFO:Detectors:
Pragma version0.8.26 (OTCOMToken.sol#11) necessitates a version too recent to be trusted.
Consider deploying with 0.8.18.
solc-0.8.26 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in OTCOMToken.swapAndLiquify() (OTCOMToken.sol#628-654):
```

```
        - (success1,None) = devWallet.call{gas: 35000,value: devAmount}()
(OTCOMToken.sol#647)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
OTCOMToken (OTCOMToken.sol#267-751) should inherit from IERC20
(OTCOMToken.sol#116-191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
INFO:Detectors:
Parameter OTCOMToken.setTaxPercentage(uint256)._taxPercentage (OTCOMToken.sol#557) is
not in mixedCase
Parameter OTCOMToken.setTaxThreshold(uint256)._threshold (OTCOMToken.sol#576) is not in
mixedCase
Constant OTCOMToken._name (OTCOMToken.sol#269) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant OTCOMToken._symbol (OTCOMToken.sol#270) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant OTCOMToken._decimals (OTCOMToken.sol#271) is not in
UPPER_CASE_WITH_UNDERSCORES
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
OTCOMToken.setTaxPercentage(uint256) (OTCOMToken.sol#557-566) uses literals with too
many digits:
        - require(bool,string)(_taxPercentage <= 100000,Tax percentage cannot exceed 100%)
(OTCOMToken.sol#558)
OTCOMToken._calculateTax(uint256,uint256) (OTCOMToken.sol#736-738) uses literals with too
many digits:
        - amount * (_taxPercentage) / (100000) (OTCOMToken.sol#737)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
OTCOMToken.devTaxShare (OTCOMToken.sol#284) should be constant
OTCOMToken.liquidityTaxShare (OTCOMToken.sol#283) should be constant
OTCOMToken.swapEnabled (OTCOMToken.sol#291) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-decl
ared-constant
INFO:Slither:OTCOMToken.sol analyzed (7 contracts with 93 detectors), 29 result(s) found
```

Solidity Static Analysis

OTCOMToken.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 721:25:

Gas costs:
Gas requirement of function OTOMToken.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 407:37:

Gas costs:
Gas requirement of function OTOMToken.setNumberOfBlocksForBlacklist is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 579:25:

Gas costs:
Gas requirement of function OTOMToken.setDevWallet is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 607:38:

Similar variable names:
OTOMToken._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 427:12:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 636:2:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 793:512:

Solhint Linter

OTCOMToken.sol

```
Compiler version 0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:12
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:57
Error message for require is too long
Pos: 9:99
Function name must be in mixedCase
Pos: 5:202
Constant name must be in capitalized SNAKE_CASE
Pos: 5:272
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:315
Error message for require is too long
Pos: 9:316
Error message for require is too long
Pos: 9:592
Avoid making time-based decisions in your business logic
Pos: 13:649
Error message for require is too long
Pos: 9:726
Error message for require is too long
Pos: 17:766
Code contains empty blocks
Pos: 32:791
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io