Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	LibraX Finance
Website:	https://librax.finance
Platform:	Astar Network
Language:	Solidity
Date:	April 26th, 2022

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 the LibraX Finance team to perform the Security audit of the LibraX protocol smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on April 26th, 2022.

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

LibraX is an automated market-making (AMM) decentralized exchange (DEX) for the Astar network. LibraX Finance smart Contract has functions like mint, transfer, permit, createPair, setFeeToSetter, setFeeTo, burn, swap, skim, getBlockHash, etc.

Audit scope

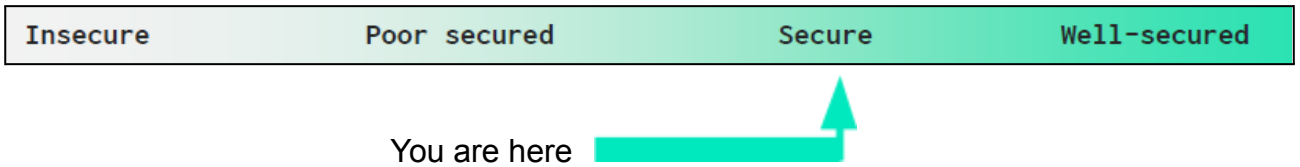
Name	**Code Review and Security Analysis Report for LibraX Protocol Smart Contracts**
Platform	**Astar Network / Solidity**
File 1	UniswapV2ERC20.sol
File 1 Github Commit	037f07f46d9e921f5fdf28a07e2ce7885ad0e20f
File 2	UniswapV2Factory.sol
File 2 Github Commit	19eed777eab19602d011f27b39e4bb1c499c6042
File 3	UniswapV2Pair.sol
File 3 Github Commit	19eed777eab19602d011f27b39e4bb1c499c6042
File 4	UniswapV2Router02.sol
File 4 Github Commit	19eed777eab19602d011f27b39e4bb1c499c6042
File 5	Multicall.sol
File 5 Github Commit	19eed777eab19602d011f27b39e4bb1c499c6042
Audit Date	April 26th, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 UniswapV2ERC20.sol • Name: LibraX LP • Symbol: LXLP • Decimals: 18	**YES, This is valid.**
File 2 UniswapV2Factory.sol • Generates trading pairs	**YES, This is valid.**
File 3 UniswapV2Pair.sol • It serves as an automated market maker and keeps track of pool token balances. • Minimum Liquidity: 1000	**YES, This is valid.**
File 4 UniswapV2Router02.sol • It supports all the trading functions for many pairs. • It has functions like: receive, addLiquidity, addLiquidityETH, etc.	**YES, This is valid.**
File 5 Multicall.sol • It Aggregates results from multiple read-only function calls • It has functions like: aggregate, getEthBalance, getBlockHash, etc.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts do not contain owner control, which does make them fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 1 low and some very low level issues.
These issues are fixed / acknowledged in the revised code.

Investors Advice: 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.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	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 not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 5 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the LibraX Protocol 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 LibraX Protocol.

The LibraX team has provided unit test scripts, which have helped to determine the integrity of the code in an automated way.

Some code parts are not well commented on smart contracts. We suggest using Ethereum's NatSpec style for the commenting.

Documentation

We were given a LibraX Protocol smart contract code in the form of a Github web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are not well commented. But the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

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

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

UniswapV2ERC20.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	_mint	internal	Passed	No Issue
3	_burn	internal	Passed	No Issue
4	_approve	write	Passed	No Issue
5	_transfer	write	Passed	No Issue
6	approve	external	Passed	No Issue
7	transfer	external	Passed	No Issue
8	transferFrom	external	Passed	No Issue
9	permit	external	Passed	No Issue

UniswapV2Factory.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	allPairsLength	external	Passed	No Issue
3	pairCodeHash	external	Passed	No Issue
4	createPair	external	Passed	No Issue
5	setFeeTo	external	Passed	No Issue
6	setFeeToSetter	external	Passed	No Issue

UniswapV2Pair.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	_mint	internal	Passed	No Issue
3	_burn	internal	Passed	No Issue
4	_approve	write	Passed	No Issue
5	_transfer	write	Passed	No Issue
6	approve	external	Passed	No Issue
7	transfer	external	Passed	No Issue
8	transferFrom	external	Passed	No Issue
9	permit	external	Passed	No Issue
10	lock	modifier	Passed	No Issue
11	getReserves	read	Passed	No Issue
12	_safeTransfer	write	Passed	No Issue
13	initialize	external	Passed	No Issue

14	_update	write	Passed	No Issue
15	_mintFee	write	Passed	No Issue
16	mint	external	Passed	No Issue
17	burn	external	Passed	No Issue
18	swap	external	Passed	No Issue
19	skim	external	Passed	No Issue
20	sync	external	Passed	No Issue

UniswapV2Router02.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	ensure	modifier	Passed	No Issue
3	receive	external	Passed	No Issue
4	_addLiquidity	internal	Passed	No Issue
5	addLiquidity	external	Passed	No Issue
6	addLiquidityETH	external	Passed	No Issue
7	removeLiquidity	write	Passed	No Issue
8	removeLiquidityETH	write	Passed	No Issue
9	removeLiquidityWithPermit	external	Passed	No Issue
10	removeLiquidityETHWithPermit	external	Passed	No Issue
11	removeLiquidityETHSupportingFeeOnTransferTokens	write	Passed	No Issue
12	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	external	Passed	No Issue
13	_swap	internal	Passed	No Issue
14	swapExactTokensForTokens	external	Passed	No Issue
15	swapTokensForExactTokens	external	Passed	No Issue
16	swapExactETHForTokens	external	Passed	No Issue
17	swapTokensForExactETH	external	Passed	No Issue
18	swapExactTokensForETH	external	Passed	No Issue
19	swapETHForExactTokens	external	Passed	No Issue
20	_swapSupportingFeeOnTransferTokens	internal	Passed	No Issue
21	swapExactTokensForTokensSupportingFeeOnTransferTokens	external	Passed	No Issue
22	swapExactETHForTokensSupportingFeeOnTransferTokens	external	Passed	No Issue
23	swapExactTokensForETHSupportingFeeOnTransferTokens	external	Passed	No Issue
24	quote	write	Passed	No Issue
25	getAmountOut	write	Passed	No Issue
26	getAmountIn	write	Passed	No Issue
27	getAmountsOut	read	Passed	No Issue
28	getAmountsIn	read	Passed	No Issue

Multicall.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	aggregate	write	Passed	No Issue
3	getEthBalance	read	Passed	No Issue
4	getBlockHash	read	Passed	No Issue
5	getLastBlockHash	read	Passed	No Issue
6	getCurrentBlockTimestamp	read	Passed	No Issue
7	getCurrentBlockDifficulty	read	Passed	No Issue
8	getCurrentBlockGasLimit	read	Passed	No Issue
9	getCurrentBlockCoinbase	read	Passed	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 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 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

(1) Missing event logs in UniswapV2Factory.sol

It is best practice to fire an event when a significant state change is happening. It helps clients interact with the blockchain. We suggest to add events in following functions:

- setFeeTo
- setFeeToSetter

Resolution: Add appropriate events in above functions.

Status: Acknowledged

Very Low / Informational / Best practices:

(1) Consider using the latest solidity version while contract deployment to prevent any compiler version level bugs. There are many features introduced and many security bugs are fixed so it is a good practice to use the latest solidity version.

Resolution: Please use the latest solidity version.

Status: Acknowledged

Conclusion

We were given a contract code in the form of github repositories. And we have used all possible tests based on given objects as files. We had observed some issues in the smart contracts, but those issues are not critical ones. **So, the smart contracts are ready for the 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 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.

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

Security state of the reviewed 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 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 whitebox penetration testing. We look at the project's web site 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, 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 are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing 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.

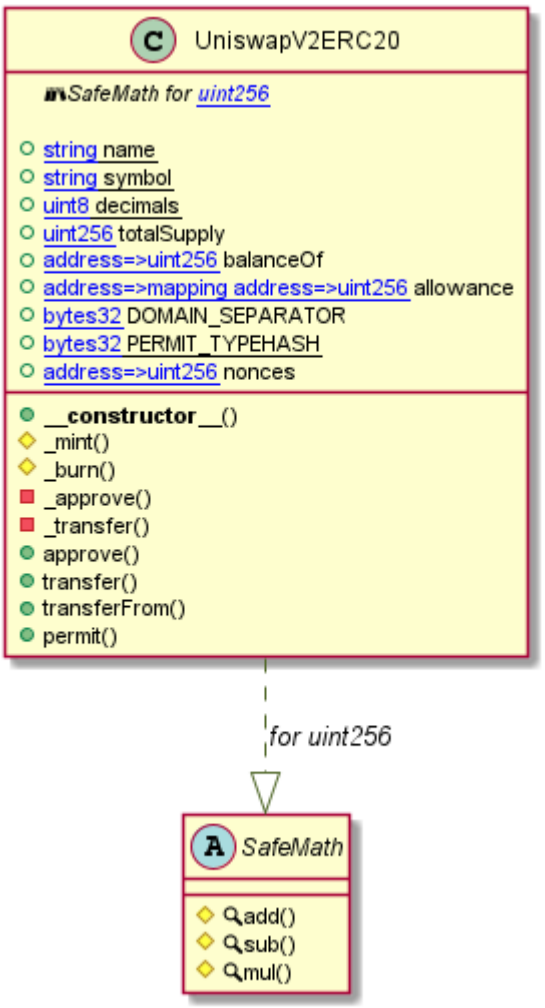
Technical Disclaimer

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 explicit security of the audited smart contracts.

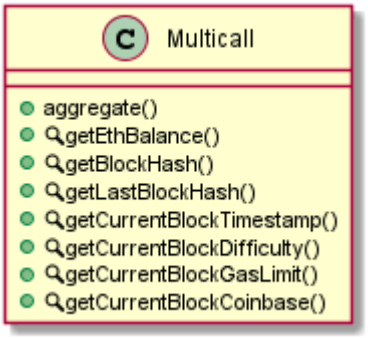
Appendix

Code Flow Diagram - LibraX Protocol

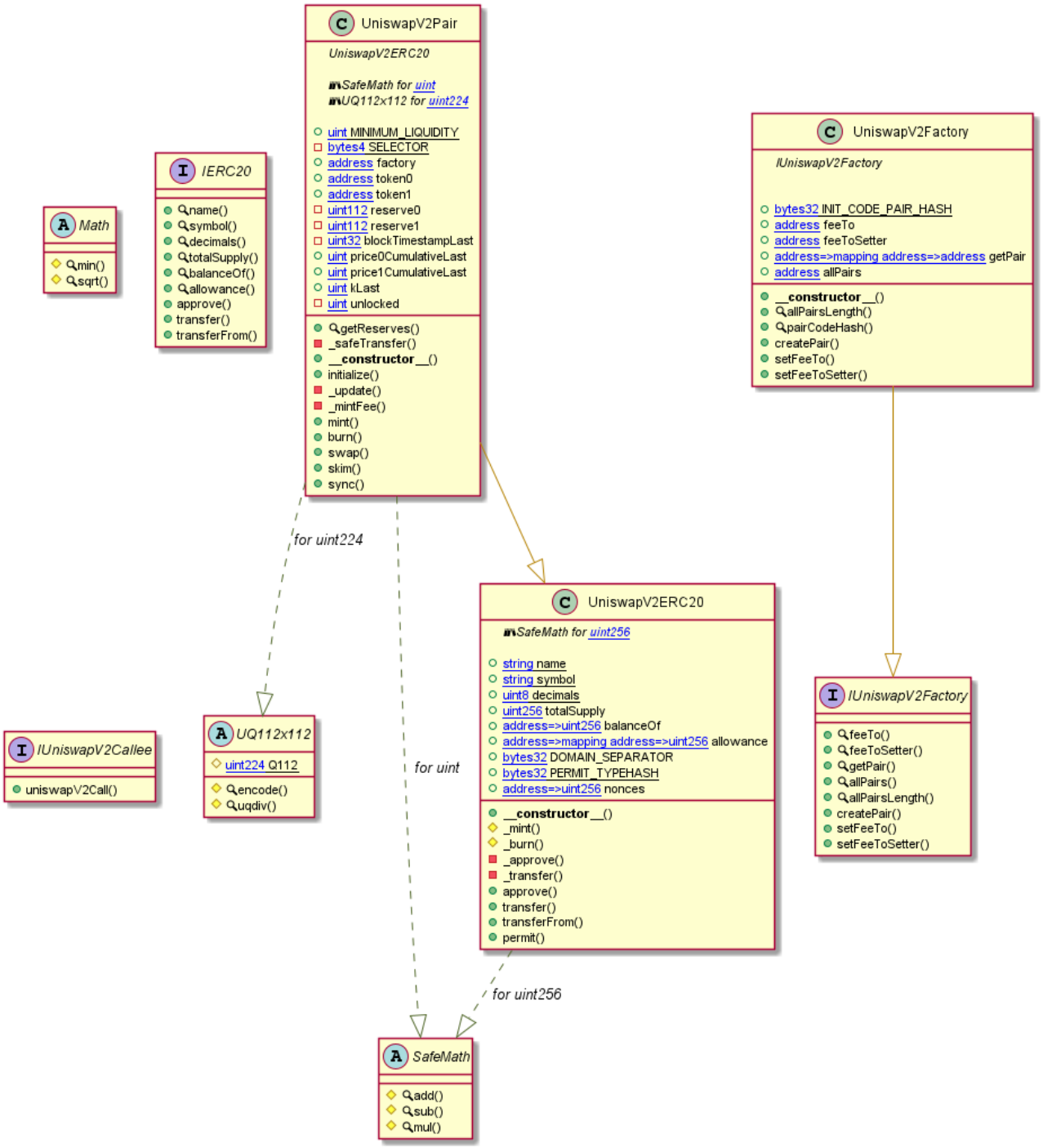
UniswapV2ERC20 Diagram



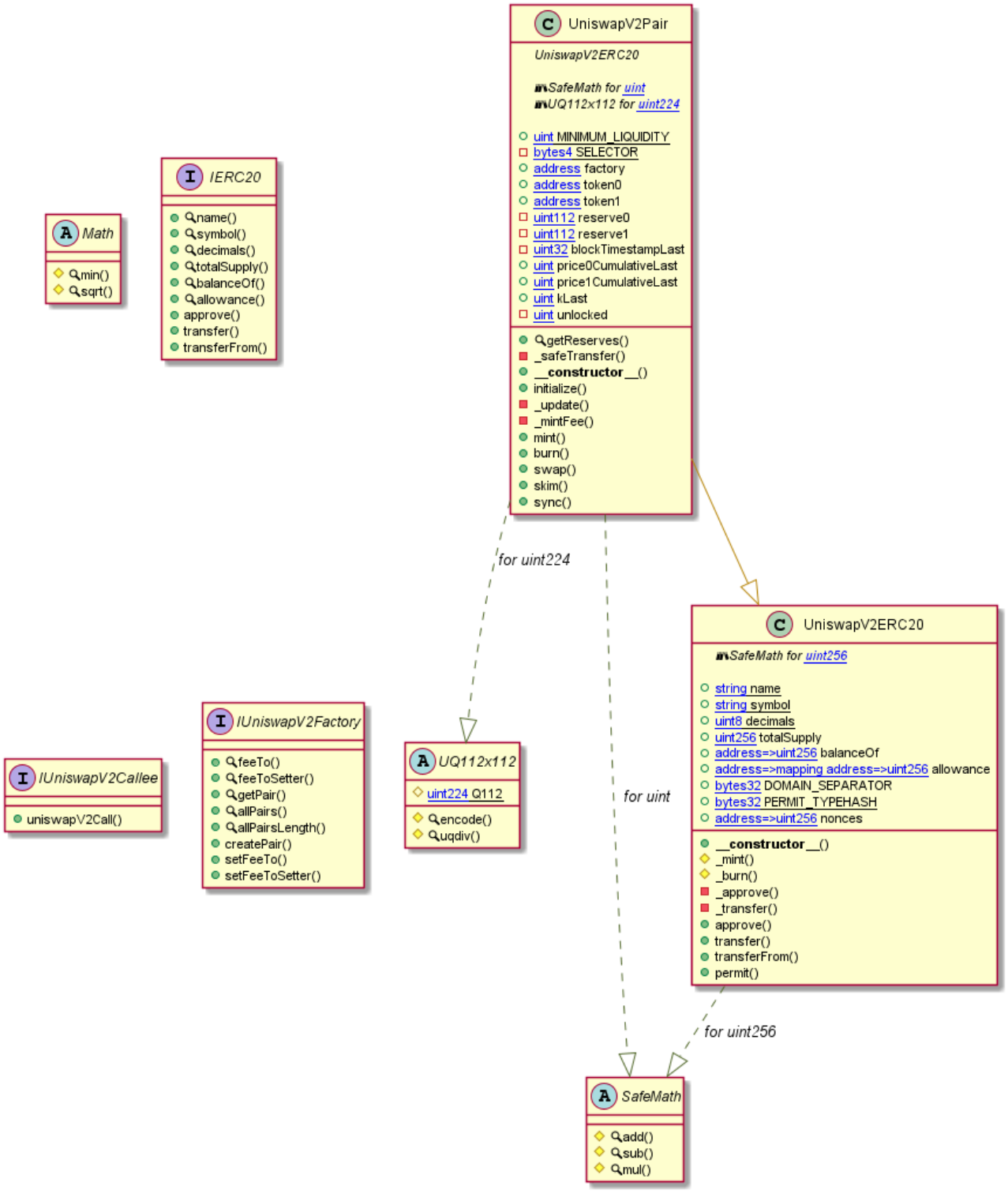
Multicall Diagram



UniswapV2Factory Diagram

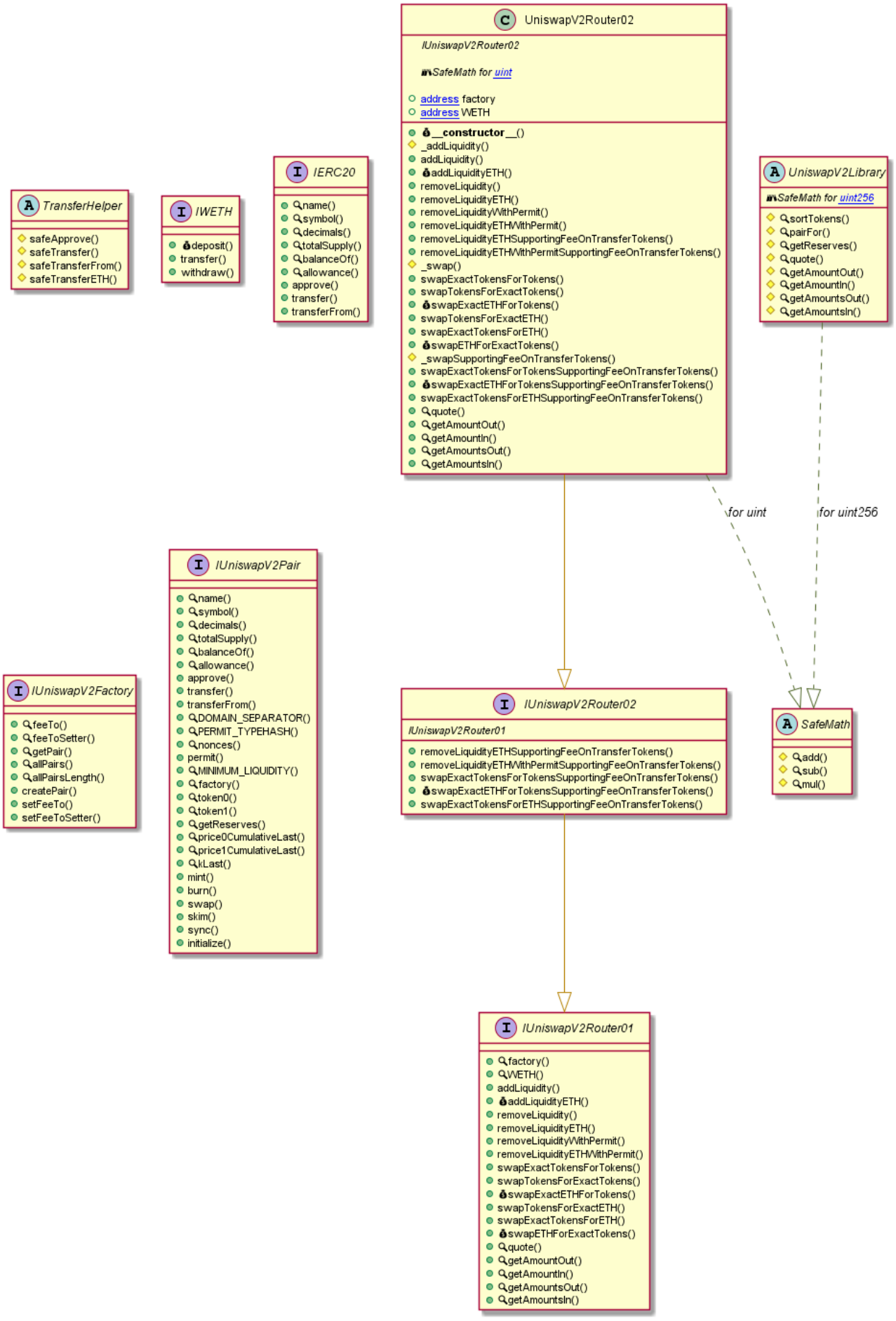


UniswapV2Pair Diagram



UniswapV2Router02 Diagram

Email: audit@EtherAuthority.io



Slither Results Log

Slither log >> UniswapV2ERC20.sol

```
INFO:Detectors:
UniswapV2ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (UniswapV2ERC20.sol#106-126) uses timestamp for co
parisons
        Dangerous comparisons:
        - require(bool,string)(deadline >= block.timestamp,UniswapV2: EXPIRED) (UniswapV2ERC20.sol#115)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
UniswapV2ERC20.constructor() (UniswapV2ERC20.sol#37-51) uses assembly
        - INLINE ASM (UniswapV2ERC20.sol#39-41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
SafeMath.mul(uint256,uint256) (UniswapV2ERC20.sol#14-16) is never used and should be removed
UniswapV2ERC20._burn(address,uint256) (UniswapV2ERC20.sol#59-63) is never used and should be removed
UniswapV2ERC20._mint(address,uint256) (UniswapV2ERC20.sol#53-57) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Variable UniswapV2ERC20.DOMAIN_SEPARATOR (UniswapV2ERC20.sol#29) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:UniswapV2ERC20.sol analyzed (2 contracts with 75 detectors), 6 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> UniswapV2Factory.sol

```
INFO:Detectors:
UniswapV2Pair.initialize(address,address)._token0 (UniswapV2Factory.sol#253) lacks a zero-check on :
                - token0 = _token0 (UniswapV2Factory.sol#255)
UniswapV2Pair.initialize(address,address)._token1 (UniswapV2Factory.sol#253) lacks a zero-check on :
                - token1 = _token1 (UniswapV2Factory.sol#256)
UniswapV2Factory.constructor(address)._feeToSetter (UniswapV2Factory.sol#399) lacks a zero-check on :
                - feeToSetter = _feeToSetter (UniswapV2Factory.sol#400)
UniswapV2Factory.setFeeTo(address)._feeTo (UniswapV2Factory.sol#428) lacks a zero-check on :
                - feeTo = _feeTo (UniswapV2Factory.sol#430)
UniswapV2Factory.setFeeToSetter(address)._feeToSetter (UniswapV2Factory.sol#433) lacks a zero-check on :
                - feeToSetter = _feeToSetter (UniswapV2Factory.sol#435)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in UniswapV2Pair.burn(address) (UniswapV2Factory.sol#321-343):
        External calls:
        - _safeTransfer(_token0,to,amount0) (UniswapV2Factory.sol#335)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (UniswapV2Factory.sol#232)
        - _safeTransfer(_token1,to,amount1) (UniswapV2Factory.sol#336)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (UniswapV2Factory.sol#232)
        State variables written after the call(s):
        - _update(balance0,balance1,_reserve0,_reserve1) (UniswapV2Factory.sol#340)
                - price0CumulativeLast += uint256(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed (UniswapV2Factory
sol#266)
        - _update(balance0,balance1,_reserve0,_reserve1) (UniswapV2Factory.sol#340)
                - price1CumulativeLast += uint256(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed (UniswapV2Factory
sol#267)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
UniswapV2ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (UniswapV2Factory.sol#175-195) uses timestamp for
omparisons
        Dangerous comparisons:
        - require(bool,string)(deadline >= block.timestamp,UniswapV2: EXPIRED) (UniswapV2Factory.sol#184)
UniswapV2Pair._update(uint256,uint256,uint112,uint112) (UniswapV2Factory.sol#260-273) uses timestamp for comparisons
        Dangerous comparisons:
        - timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0 (UniswapV2Factory.sol#264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
UniswapV2ERC20.constructor() (UniswapV2Factory.sol#106-120) uses assembly
        - INLINE ASM (UniswapV2Factory.sol#108-110)
UniswapV2Factory.createPair(address,address) (UniswapV2Factory.sol#411-426) uses assembly
        - INLINE ASM (UniswapV2Factory.sol#418-420)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Low level call in UniswapV2Pair._safeTransfer(address,address,uint256) (UniswapV2Factory.sol#231-234):
        - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (UniswapV2Factory.sol#232)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Variable UniswapV2ERC20.DOMAIN_SEPARATOR (UniswapV2Factory.sol#98) is not in mixedCase
Parameter UniswapV2Pair.initialize(address,address)._token0 (UniswapV2Factory.sol#253) is not in mixedCase
Parameter UniswapV2Pair.initialize(address,address)._token1 (UniswapV2Factory.sol#253) is not in mixedCase
Parameter UniswapV2Factory.setFeeTo(address)._feeTo (UniswapV2Factory.sol#428) is not in mixedCase
Parameter UniswapV2Factory.setFeeToSetter(address)._feeToSetter (UniswapV2Factory.sol#433) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable UniswapV2Pair.swap(uint256,uint256,address,bytes).balance0Adjusted (UniswapV2Factory.sol#367) is too similar to Uniswa
V2Pair.swap(uint256,uint256,address,bytes).balance1Adjusted (UniswapV2Factory.sol#368)
Variable UniswapV2Pair.price0CumulativeLast (UniswapV2Factory.sol#213) is too similar to UniswapV2Pair.price1CumulativeLast (Un
swapV2Factory.sol#214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
UniswapV2Factory.pairCodeHash() (UniswapV2Factory.sol#407-409) uses literals with too many digits:
        - keccak256(bytes)(type()(UniswapV2Pair).creationCode) (UniswapV2Factory.sol#408)
UniswapV2Factory.createPair(address,address) (UniswapV2Factory.sol#411-426) uses literals with too many digits:
        - bytecode = type()(UniswapV2Pair).creationCode (UniswapV2Factory.sol#416)
UniswapV2Factory.slitherConstructorConstantVariables() (UniswapV2Factory.sol#389-439) uses literals with too many digits:
        - INIT_CODE_PAIR_HASH = keccak256(bytes)(abi.encodePacked(type()(UniswapV2Pair).creationCode)) (UniswapV2Factory.sol#39
)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:UniswapV2Factory.sol analyzed (9 contracts with 75 detectors), 32 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> UniswapV2Pair.sol

```
INFO:Detectors:
UniswapV2Pair.initialize(address,address)._token0 (UniswapV2Pair.sol#254) lacks a zero-check on :
                - token0 = _token0 (UniswapV2Pair.sol#256)
UniswapV2Pair.initialize(address,address)._token1 (UniswapV2Pair.sol#254) lacks a zero-check on :
                - token1 = _token1 (UniswapV2Pair.sol#257)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in UniswapV2Pair.burn(address) (UniswapV2Pair.sol#322-344):
        External calls:
        - _safeTransfer(_token0,to,amount0) (UniswapV2Pair.sol#336)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (UniswapV2Pair.sol#233)
        - _safeTransfer(_token1,to,amount1) (UniswapV2Pair.sol#337)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (UniswapV2Pair.sol#233)
        State variables written after the call(s):
        - _update(balance0,balance1,_reserve0,_reserve1) (UniswapV2Pair.sol#341)
                - price0CumulativeLast += uint256(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed (UniswapV2Pair.so
#267)
```

```
        - IUniswapV2Callee(to).uniswapV2Call(msg.sender,amount0Out,amount1Out,data) (UniswapV2Pair.sol#360)
        Event emitted after the call(s):
        - Swap(msg.sender,amount0In,amount1In,amount0Out,amount1Out,to) (UniswapV2Pair.sol#374)
        - Sync(reserve0,reserve1) (UniswapV2Pair.sol#273)
                - _update(balance0,balance1,_reserve0,_reserve1) (UniswapV2Pair.sol#373)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
UniswapV2ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (UniswapV2Pair.sol#176-196) uses timestamp for com
arisons
        Dangerous comparisons:
        - require(bool,string)(deadline >= block.timestamp,UniswapV2: EXPIRED) (UniswapV2Pair.sol#185)
UniswapV2Pair._update(uint256,uint256,uint112,uint112) (UniswapV2Pair.sol#261-274) uses timestamp for comparisons
        Dangerous comparisons:
        - timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0 (UniswapV2Pair.sol#265)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
UniswapV2ERC20.constructor() (UniswapV2Pair.sol#107-121) uses assembly
        - INLINE ASM (UniswapV2Pair.sol#109-111)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Low level call in UniswapV2Pair._safeTransfer(address,address,uint256) (UniswapV2Pair.sol#232-235):
        - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (UniswapV2Pair.sol#233)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Variable UniswapV2ERC20.DOMAIN_SEPARATOR (UniswapV2Pair.sol#99) is not in mixedCase
Parameter UniswapV2Pair.initialize(address,address)._token0 (UniswapV2Pair.sol#254) is not in mixedCase
Parameter UniswapV2Pair.initialize(address,address)._token1 (UniswapV2Pair.sol#254) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable UniswapV2Pair.swap(uint256,uint256,address,bytes).balance0Adjusted (UniswapV2Pair.sol#368) is too similar to UniswapV2
air.swap(uint256,uint256,address,bytes).balance1Adjusted (UniswapV2Pair.sol#369)
Variable UniswapV2Pair.price0CumulativeLast (UniswapV2Pair.sol#214) is too similar to UniswapV2Pair.price1CumulativeLast (Unisw
pV2Pair.sol#215)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Slither:UniswapV2Pair.sol analyzed (8 contracts with 75 detectors), 20 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> UniswapV2Router02.sol

```
INFO:Detectors:
UniswapV2Router02.constructor(address,address)._factory (UniswapV2Router02.sol#391) lacks a zero-check on :
                - factory = _factory (UniswapV2Router02.sol#392)
UniswapV2Router02.constructor(address,address)._WETH (UniswapV2Router02.sol#391) lacks a zero-check on :
                - WETH = _WETH (UniswapV2Router02.sol#393)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
UniswapV2Router02._swap(uint256[],address[],address) (UniswapV2Router02.sol#580-591) has external calls inside a loop: IUniswap
2Pair(UniswapV2Library.pairFor(factory,input,output)).swap(amount0Out,amount1Out,to,new bytes(0)) (UniswapV2Router02.sol#587-58
)
UniswapV2Router02._swapSupportingFeeOnTransferTokens(address[],address) (UniswapV2Router02.sol#689-706) has external calls insi
e a loop: (reserve0,reserve1) = pair.getReserves() (UniswapV2Router02.sol#697)
UniswapV2Router02._swapSupportingFeeOnTransferTokens(address[],address) (UniswapV2Router02.sol#689-706) has external calls insi
e a loop: amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput) (UniswapV2Router02.sol#699)
UniswapV2Router02._swapSupportingFeeOnTransferTokens(address[],address) (UniswapV2Router02.sol#689-706) has external calls insi
e a loop: pair.swap(amount0Out,amount1Out,to,new bytes(0)) (UniswapV2Router02.sol#704)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
TransferHelper.safeApprove(address,address,uint256) (UniswapV2Router02.sol#6-10) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in TransferHelper.safeApprove(address,address,uint256) (UniswapV2Router02.sol#6-10):
        - (success,data) = token.call(abi.encodeWithSelector(0x095ea7b3,to,value)) (UniswapV2Router02.sol#8)
Low level call in TransferHelper.safeTransfer(address,address,uint256) (UniswapV2Router02.sol#12-16):
        - (success,data) = token.call(abi.encodeWithSelector(0xa9059cbb,to,value)) (UniswapV2Router02.sol#14)
Low level call in TransferHelper.safeTransferFrom(address,address,address,uint256) (UniswapV2Router02.sol#18-22):
        - (success,data) = token.call(abi.encodeWithSelector(0x23b872dd,from,to,value)) (UniswapV2Router02.sol#20)
Low level call in TransferHelper.safeTransferETH(address,uint256) (UniswapV2Router02.sol#24-27):
        - (success) = to.call{value: value}(new bytes(0)) (UniswapV2Router02.sol#25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
Function IUniswapV2Router01.WETH() (UniswapV2Router02.sol#31) is not in mixedCase
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (UniswapV2Router02.sol#231) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (UniswapV2Router02.sol#232) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (UniswapV2Router02.sol#249) is not in mixedCase
Variable UniswapV2Router02.WETH (UniswapV2Router02.sol#384) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (Unisw
pV2Router02.sol#36) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,u
nt256).amountBDesired (UniswapV2Router02.sol#37)
Variable UniswapV2Router02.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (Uniswa
V2Router02.sol#432) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,u
nt256).amountBDesired (UniswapV2Router02.sol#37)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
quote(uint256,uint256,uint256) should be declared external:
        - UniswapV2Router02.quote(uint256,uint256,uint256) (UniswapV2Router02.sol#771-773)
getAmountOut(uint256,uint256,uint256) should be declared external:
        - UniswapV2Router02.getAmountOut(uint256,uint256,uint256) (UniswapV2Router02.sol#775-783)
getAmountIn(uint256,uint256,uint256) should be declared external:
        - UniswapV2Router02.getAmountIn(uint256,uint256,uint256) (UniswapV2Router02.sol#785-793)
getAmountsOut(uint256,address[]) should be declared external:
        - UniswapV2Router02.getAmountsOut(uint256,address[]) (UniswapV2Router02.sol#795-803)
getAmountsIn(uint256,address[]) should be declared external:
        - UniswapV2Router02.getAmountsIn(uint256,address[]) (UniswapV2Router02.sol#805-813)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:UniswapV2Router02.sol analyzed (10 contracts with 75 detectors), 36 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> Multicall.sol

```
INFO:Detectors:
Multicall.aggregate(Multicall.Call[]) (Multicall.sol#15-23) has external calls inside a loop: (success,ret) = calls[i].target.c
ll(calls[i].callData) (Multicall.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Pragma version>=0.5.0 (Multicall.sol#2) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Multicall.aggregate(Multicall.Call[]) (Multicall.sol#15-23):
        - (success,ret) = calls[i].target.call(calls[i].callData) (Multicall.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
aggregate(Multicall.Call[]) should be declared external:
        - Multicall.aggregate(Multicall.Call[]) (Multicall.sol#15-23)
getEthBalance(address) should be declared external:
        - Multicall.getEthBalance(address) (Multicall.sol#25-27)
getBlockHash(uint256) should be declared external:
        - Multicall.getBlockHash(uint256) (Multicall.sol#28-30)
getLastBlockHash() should be declared external:
        - Multicall.getLastBlockHash() (Multicall.sol#31-33)
getCurrentBlockTimestamp() should be declared external:
        - Multicall.getCurrentBlockTimestamp() (Multicall.sol#34-36)
getCurrentBlockDifficulty() should be declared external:
        - Multicall.getCurrentBlockDifficulty() (Multicall.sol#37-39)
getCurrentBlockGasLimit() should be declared external:
        - Multicall.getCurrentBlockGasLimit() (Multicall.sol#40-42)
getCurrentBlockCoinbase() should be declared external:
        - Multicall.getCurrentBlockCoinbase() (Multicall.sol#43-45)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Multicall.sol analyzed (1 contracts with 75 detectors), 11 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

UniswapV2ERC20.sol

Security

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 39:8:

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.

more

Pos: 115:28:

Gas & Economy

Gas costs:

Gas requirement of function UniswapV2ERC20.permit 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: 106:4:

Miscellaneous

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.

more

Pos: 124:8:

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: 15:26:

Email: audit@EtherAuthority.io

UniswapV2Factory.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in UniswapV2Pair._mintFee(uint112,uint112): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 276:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in UniswapV2Factory.createPair(address,address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 411:4:

Gas & Economy

Gas costs:

Gas requirement of function UniswapV2Factory.createPair 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: 411:4:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 45:4:

Miscellaneous

Constant/View/Pure functions:

UniswapV2Factory.pairCodeHash() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 407:4:

Similar variable names:

UniswapV2Factory.createPair(address,address) : Variables have very similar names "token0" and "tokenB". Note: Modifiers are currently not considered by this static analysis.
Pos: 423:24:

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.
more
Pos: 434:8:

UniswapV2Pair.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in UniswapV2Pair._mintFee(uint112,uint112): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 277:4:

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.
more
Pos: 263:39:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 233:44:

Gas & Economy

Gas costs:

Gas requirement of function UniswapV2Pair.sync 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: 386:4:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 46:4:

Miscellaneous

Similar variable names:

UniswapV2Pair.getReserves() : Variables have very similar names "reserve0" and "_reserve1".
Note: Modifiers are currently not considered by this static analysis.
Pos: 227:20:

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.
more
Pos: 348:8:

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: 333:18:

UniswapV2Router02.sol

Security

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.
more
Pos: 387:28:

Gas & Economy

Gas costs:

Gas requirement of function UniswapV2Router02.getAmountsOut 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: 795:4:

Gas costs:

Gas requirement of function UniswapV2Router02.getAmountsIn 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: 805:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 690:8:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

more

Pos: 190:4:

Miscellaneous

Similar variable names:

UniswapV2Router02.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256) : Variables have very similar names "amountAMin" and "amountBMin". Note: Modifiers are currently not considered by this static analysis.

Pos: 439:103:

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.

more

Pos: 486:8:

Multicall.sol

Security

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.

more

Pos: 35:20:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 19:47:

Block hash:

Use of "blockhash": "blockhash(uint blockNumber)" is used to access the last 256 block hashes. A miner computes the block hash by "summing up" the information in the current block mined. By "summing up" the information cleverly, a miner can try to influence the outcome of a transaction in the current block. This is especially easy if there are only a small number of equally likely outcomes.
Pos: 32:20:

Gas & Economy

Gas costs:

Gas requirement of function Multicall.aggregate 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: 15:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 18:8:

Miscellaneous

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.
more
Pos: 20:12:

Solhint Linter

UniswapV2ERC20.sol

```
UniswapV2ERC20.sol:3:1: Error: Compiler version =0.6.12 does not
satisfy the r semver requirement
UniswapV2ERC20.sol:7:35: Error: Use double quotes for string literals
UniswapV2ERC20.sol:11:35: Error: Use double quotes for string
literals
UniswapV2ERC20.sol:15:49: Error: Use double quotes for string
literals
UniswapV2ERC20.sol:22:28: Error: Constant name must be in capitalized
SNAKE_CASE
UniswapV2ERC20.sol:22:35: Error: Use double quotes for string
literals
UniswapV2ERC20.sol:23:28: Error: Constant name must be in capitalized
SNAKE_CASE
UniswapV2ERC20.sol:23:37: Error: Use double quotes for string
literals
UniswapV2ERC20.sol:24:27: Error: Constant name must be in capitalized
SNAKE_CASE
UniswapV2ERC20.sol:29:20: Error: Variable name must be in mixedCase
UniswapV2ERC20.sol:39:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
UniswapV2ERC20.sol:44:27: Error: Use double quotes for string
literals
UniswapV2ERC20.sol:46:33: Error: Use double quotes for string
literals
UniswapV2ERC20.sol:115:29: Error: Avoid to make time-based decisions
in your business logic
UniswapV2ERC20.sol:115:46: Error: Use double quotes for string
literals
UniswapV2ERC20.sol:118:17: Error: Use double quotes for string
literals
UniswapV2ERC20.sol:124:78: Error: Use double quotes for string
literals
```

UniswapV2Factory.sol

```
UniswapV2Factory.sol:3:1: Error: Compiler version =0.6.12 does not
satisfy the r semver requirement
UniswapV2Factory.sol:26:5: Error: Explicitly mark visibility of state
UniswapV2Factory.sol:76:35: Error: Use double quotes for string
literals
UniswapV2Factory.sol:80:35: Error: Use double quotes for string
literals
UniswapV2Factory.sol:84:49: Error: Use double quotes for string
literals
UniswapV2Factory.sol:91:28: Error: Constant name must be in
capitalized SNAKE_CASE
UniswapV2Factory.sol:91:35: Error: Use double quotes for string
```

```
literals
UniswapV2Factory.sol:92:28: Error: Constant name must be in
capitalized SNAKE_CASE
UniswapV2Factory.sol:92:37: Error: Use double quotes for string
literals
UniswapV2Factory.sol:93:27: Error: Constant name must be in
capitalized SNAKE_CASE
UniswapV2Factory.sol:98:20: Error: Variable name must be in mixedCase
UniswapV2Factory.sol:108:9: Error: Avoid using inline assembly. It is
```

UniswapV2Pair.sol

```
UniswapV2Pair.sol:3:1: Error: Compiler version =0.6.12 does not
satisfy the r semver requirement
UniswapV2Pair.sol:27:5: Error: Explicitly mark visibility of state
UniswapV2Pair.sol:77:35: Error: Use double quotes for string literals
UniswapV2Pair.sol:81:35: Error: Use double quotes for string literals
UniswapV2Pair.sol:85:49: Error: Use double quotes for string literals
UniswapV2Pair.sol:92:28: Error: Constant name must be in capitalized
SNAKE_CASE
UniswapV2Pair.sol:92:35: Error: Use double quotes for string literals
UniswapV2Pair.sol:93:28: Error: Constant name must be in capitalized
SNAKE_CASE
UniswapV2Pair.sol:93:37: Error: Use double quotes for string literals
UniswapV2Pair.sol:94:27: Error: Constant name must be in capitalized
SNAKE_CASE
UniswapV2Pair.sol:99:20: Error: Variable name must be in mixedCase
UniswapV2Pair.sol:109:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
UniswapV2Pair.sol:114:27: Error: Use double quotes for string
literals
UniswapV2Pair.sol:116:33: Error: Use double quotes for string
literals
UniswapV2Pair.sol:185:29: Error: Avoid to make time-based decisions
in your business logic
UniswapV2Pair.sol:185:46: Error: Use double quotes for string
literals
UniswapV2Pair.sol:188:17: Error: Use double quotes for string
literals
UniswapV2Pair.sol:194:78: Error: Use double quotes for string
literals
UniswapV2Pair.sol:204:63: Error: Use double quotes for string
literals
UniswapV2Pair.sol:220:32: Error: Use double quotes for string
literals
UniswapV2Pair.sol:233:45: Error: Avoid using low level calls.
UniswapV2Pair.sol:234:76: Error: Use double quotes for string
literals
UniswapV2Pair.sol:255:40: Error: Use double quotes for string
literals
UniswapV2Pair.sol:262:69: Error: Use double quotes for string
literals
UniswapV2Pair.sol:263:40: Error: Avoid to make time-based decisions
```

```
in your business logic
UniswapV2Pair.sol:313:32: Error: Use double quotes for string
literals
UniswapV2Pair.sol:334:45: Error: Use double quotes for string
literals
UniswapV2Pair.sol:348:51: Error: Use double quotes for string
literals
UniswapV2Pair.sol:350:67: Error: Use double quotes for string
literals
UniswapV2Pair.sol:357:49: Error: Use double quotes for string
literals
UniswapV2Pair.sol:366:49: Error: Use double quotes for string
literals
UniswapV2Pair.sol:370:104: Error: Use double quotes for string
literals
```

UniswapV2Router02.sol

```
UniswapV2Router02.sol:2:1: Error: Compiler version =0.6.12 does not
satisfy the r semver requirement
UniswapV2Router02.sol:8:45: Error: Avoid using low level calls.
UniswapV2Router02.sol:9:76: Error: Use double quotes for string
literals
UniswapV2Router02.sol:14:45: Error: Avoid using low level calls.
UniswapV2Router02.sol:15:76: Error: Use double quotes for string
literals
UniswapV2Router02.sol:31:5: Error: Function name must be in mixedCase
UniswapV2Router02.sol:173:35: Error: Use double quotes for string
literals
UniswapV2Router02.sol:177:35: Error: Use double quotes for string
literals
UniswapV2Router02.sol:181:49: Error: Use double quotes for string
literals
UniswapV2Router02.sol:231:5: Error: Function name must be in
mixedCase
UniswapV2Router02.sol:232:5: Error: Function name must be in
mixedCase
UniswapV2Router02.sol:249:5: Error: Function name must be in
mixedCase
UniswapV2Router02.sol:273:35: Error: Use double quotes for string
literals
UniswapV2Router02.sol:275:39: Error: Use double quotes for string
literals
UniswapV2Router02.sol:354:35: Error: Use double quotes for string
literals
UniswapV2Router02.sol:369:35: Error: Use double quotes for string
literals
UniswapV2Router02.sol:384:39: Error: Variable name must be in
mixedCase
UniswapV2Router02.sol:387:29: Error: Avoid to make time-based
decisions in your business logic
UniswapV2Router02.sol:387:46: Error: Use double quotes for string
literals
UniswapV2Router02.sol:391:35: Error: Variable name must be in
mixedCase
UniswapV2Router02.sol:419:55: Error: Use double quotes for string
literals
```

Multicall.sol

```
Multicall.sol:2:1: Error: Compiler version >=0.5.0 does not satisfy
the r semver requirement
Multicall.sol:19:48: Error: Avoid using low level calls.
Multicall.sol:35:21: Error: Avoid to make time-based decisions in
your business logic
```

Software analysis result:

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

Email: audit@EtherAuthority.io