RD AUDITORS

Orbs, Smart Contract Code Review and Security Analysis Report

Customer: Orbs
Prepared on: 28th August 2022
Platform: Ethereum and Polygon
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of Orbs
Platform	ETH / Solidity
File 1	OrderLib.sol
MD5 hash	1af9ef8b77ca972a8e45bb5500523d51
SHA256 hash	dfd7554ccbaa9f2715dc19fd48bd1ebb58673ec6f79a9e27d2fb731bf28a644e
File 2	TWAP.sol
MD5 hash	ea2fd373f9ec71aba6e015201e810be5
SHA256 hash	b2f7bfb5ed2fde067b728394060a7454d83e9f17d94ea2a63aa3a7a8d471b629
File 3	UniswapV2Exchange.sol
MD5 hash	3abd6dc1528ca03153aebfd8040a585e
SHA256 hash	9241c0a50f8025b1671de562871be4295132a9c823f6720b445273b73549f176
Date	28/08/2022

Introduction

RD Auditors (Consultant) were contacted by Orbs (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contract and its code review conducted between 15th - 28th August 2022.

This contract consists of three files.

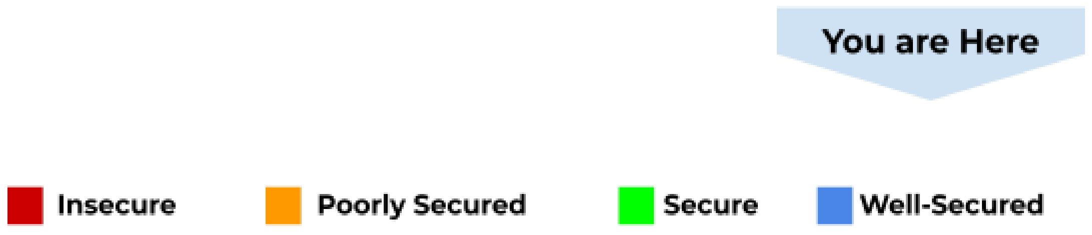
Project Scope

The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer's solidity smart contract is now **Well-Secured.**



Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit 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 issues found are located in the audit overview section.

We found the following;

Total Issues	0
🟥 Critical	0
🟧 High	0
🟨 Medium	0
🟩 Low	0
🟦 Very Low	0

Code Quality

The libraries within this smart contract are part of a 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 to a specific address and its properties/methods can be reused many times by other contracts.

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

Overall, the code is well commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

OrderLib.sol

File and Function Level Report

File:	OrderLib.sol
Contrac/library:	OrderLib
Observation:	Passed
Test Report:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	newOrder	read	Passed	All Passed	No Issue	Passed
2	newBid	read	Passed	All Passed	No Issue	Passed
3	srcBidAmountNext	read	Passed	All Passed	No Issue	Passed
4	dstMinAmountNext	read	Passed	All Passed	No Issue	Passed
5	filled	read	Passed	All Passed	No Issue	Passed

TWAP.sol

File and Function Level Report

File :	TWAP.sol
Contract:	TWAP
Import:	ERC20, SafeERC20, Address, ReentrancyGuard, OrderLib, IExchange
Inherit:	ReentrancyGuard
Observation:	Passed
Test Report:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	order	read	Passed	All Passed	No Issue	Passed
2	length	read	Passed	All Passed	No Issue	Passed
3	ask	write	Passed	All Passed	No Issue	Passed
4	bid	write	Passed	All Passed	No Issue	Passed
5	fill	write	Passed	All Passed	No Issue	Passed
6	cancel	write	Passed	All Passed	No Issue	Passed
7	verifyBid	write	Passed	All Passed	No Issue	Passed
8	verifyMakerBalance	write	Passed	All Passed	No Issue	Passed
9	performFill	write	Passed	All Passed	No Issue	Passed
10	performFillSwap	write	Passed	All Passed	No Issue	Passed
11	prune	write	Passed	All Passed	No Issue	Passed

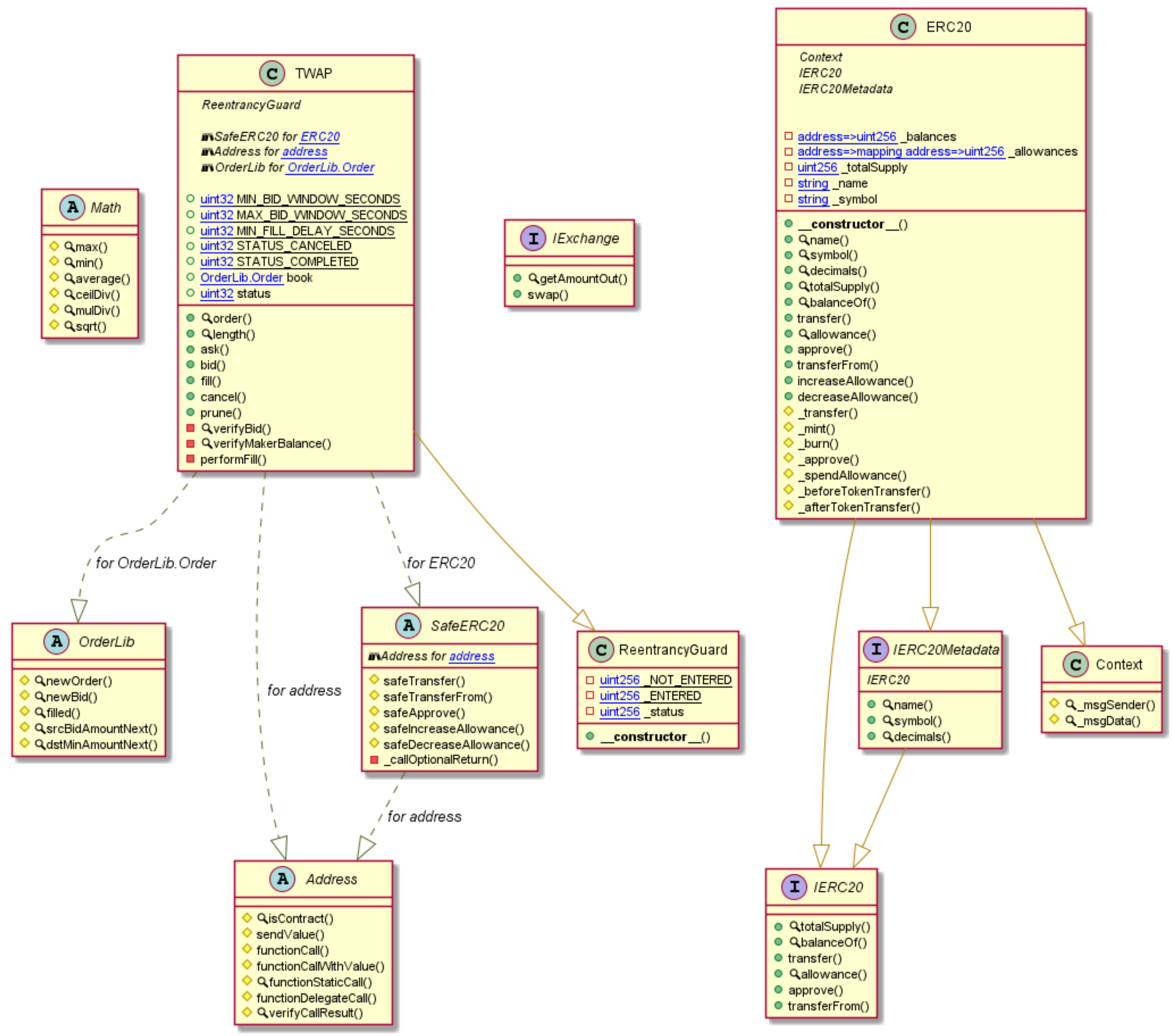
UniswapV2Exchange.sol

File and Function Level Report

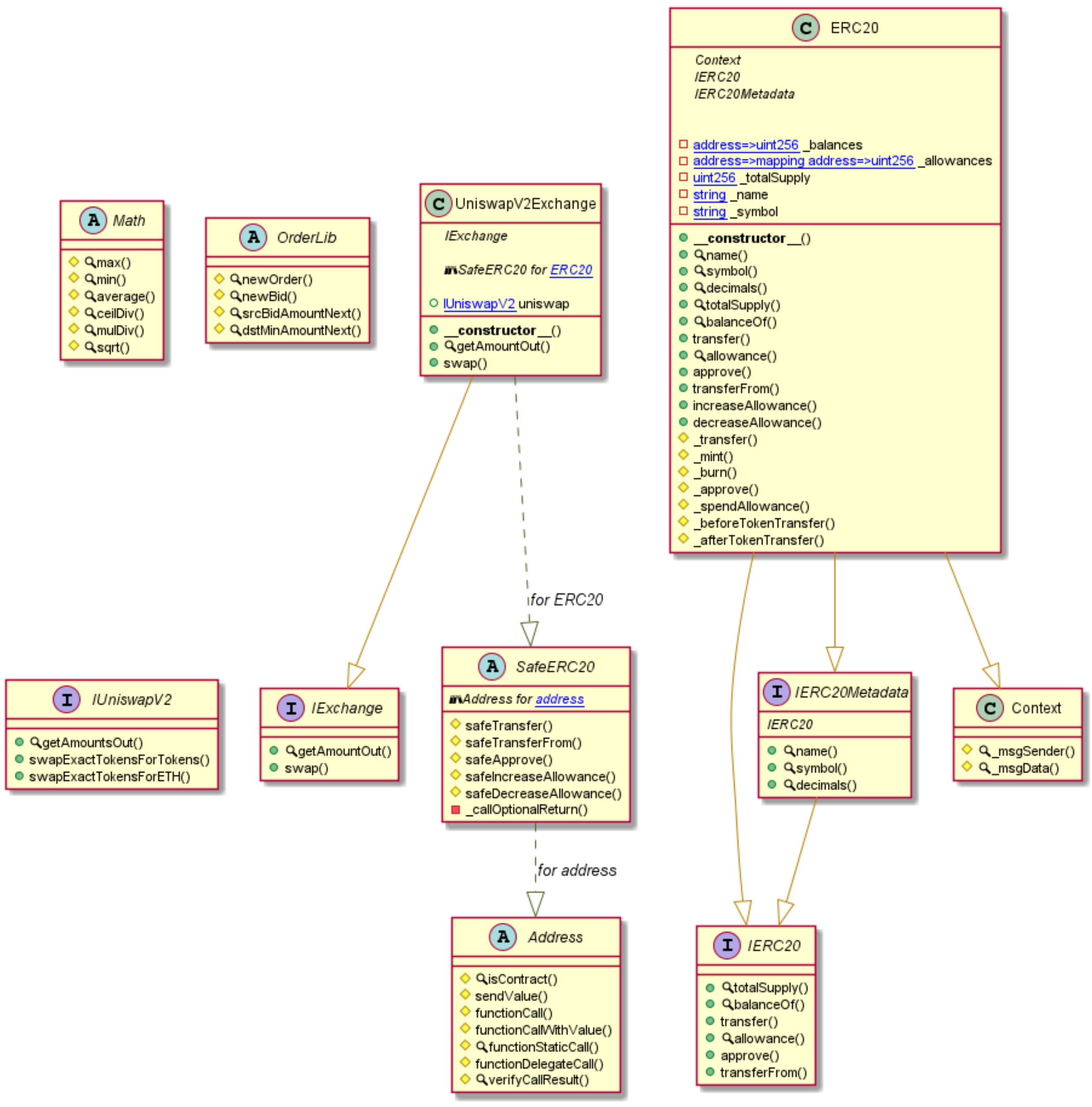
File:	UniswapV2Exchange.sol
Contract:	UniswapV2Exchange
Import:	ERC20, SafeERC20, IExchange, IUniswapV2
Inherit:	IExchange
Observation:	Passed
Test Report:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getAmountOut	read	Passed	All Passed	No Issue	Passed
2	swap	write	Passed	All Passed	No Issue	Passed

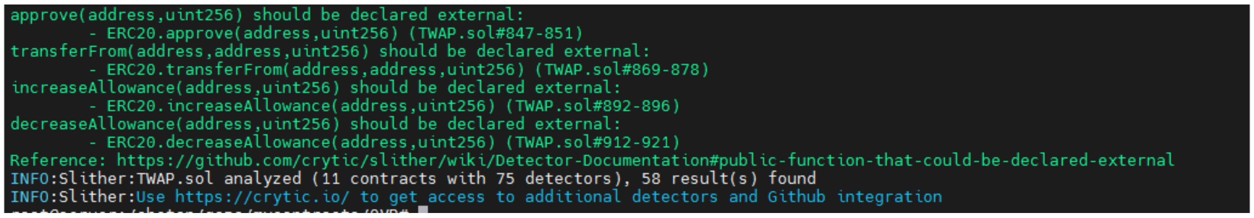
Code Flow Diagram - Twap



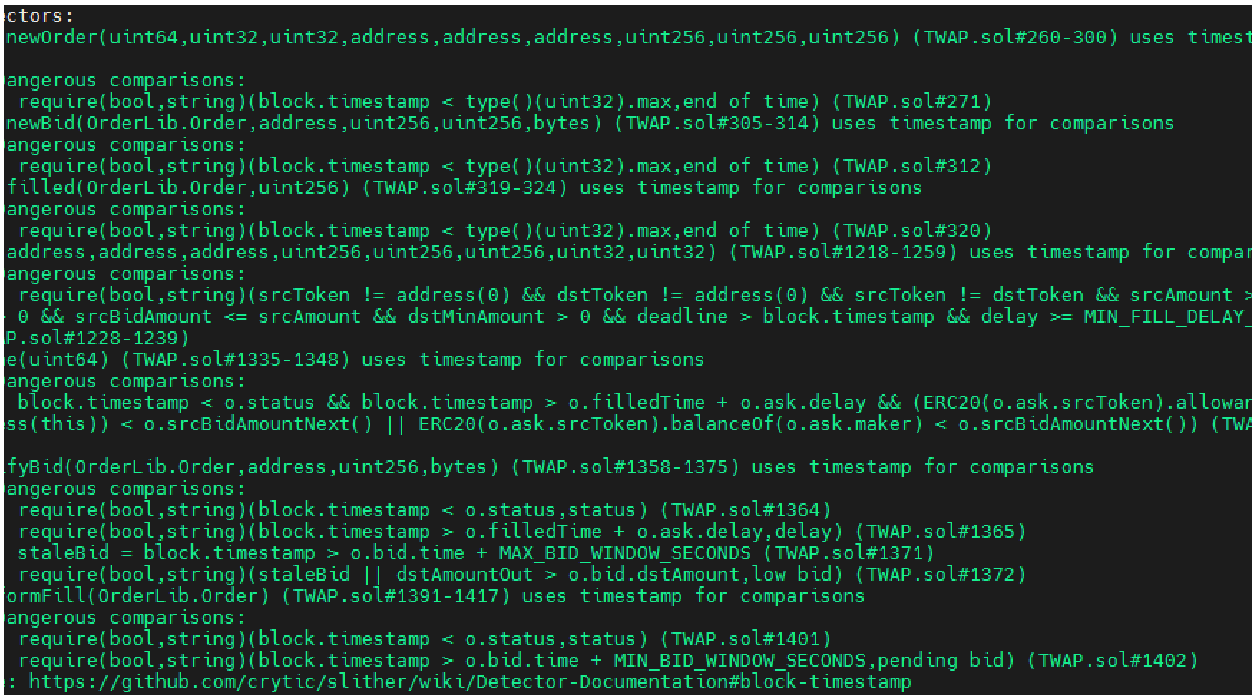
Code Flow Diagram - UniswapV2Exchange



Slither Results Log - Twap

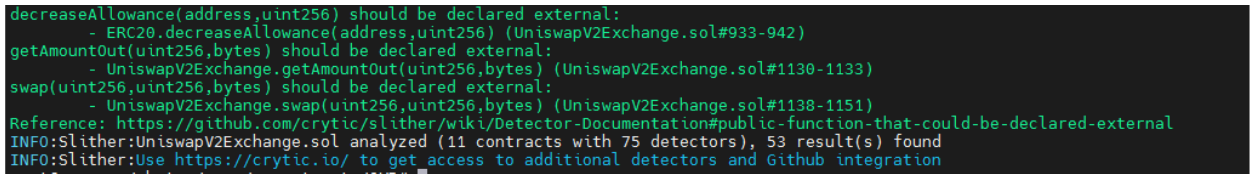


```
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (TWAP.sol#847-851)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (TWAP.sol#869-878)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (TWAP.sol#892-896)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (TWAP.sol#912-921)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:TWAP.sol analyzed (11 contracts with 75 detectors), 58 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

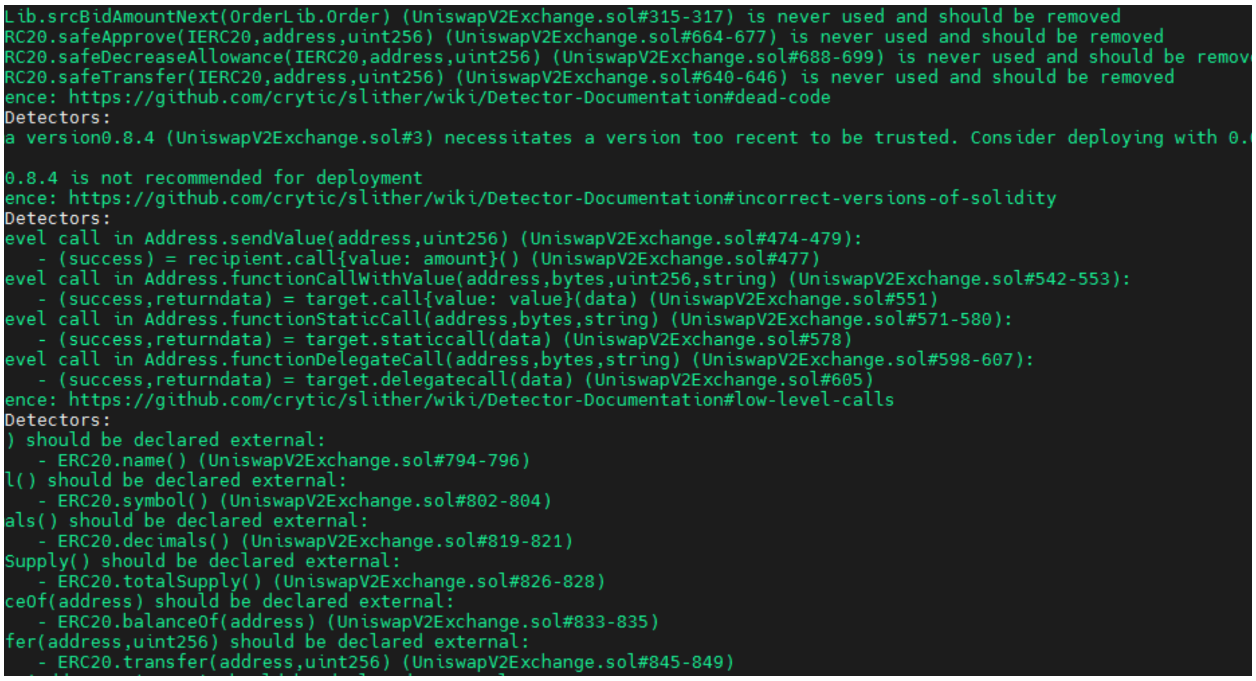


```
ctors:
newOrder(uint64,uint32,uint32,address,address,address,uint256,uint256,uint256) (TWAP.sol#260-300) uses timest

angerous comparisons:
 require(bool,string)(block.timestamp < type()(uint32).max,end of time) (TWAP.sol#271)
newBid(OrderLib.Order,address,uint256,uint256,bytes) (TWAP.sol#305-314) uses timestamp for comparisons
angerous comparisons:
 require(bool,string)(block.timestamp < type()(uint32).max,end of time) (TWAP.sol#312)
filled(OrderLib.Order,uint256) (TWAP.sol#319-324) uses timestamp for comparisons
angerous comparisons:
 require(bool,string)(block.timestamp < type()(uint32).max,end of time) (TWAP.sol#320)
address,address,address,uint256,uint256,uint256,uint32,uint32) (TWAP.sol#1218-1259) uses timestamp for compar
angerous comparisons:
 require(bool,string)(srcToken != address(0) && dstToken != address(0) && srcToken != dstToken && srcAmount >
 0 && srcBidAmount <= srcAmount && dstMinAmount > 0 && deadline > block.timestamp && delay >= MIN_FILL_DELAY_
P.sol#1228-1239)
e(uint64) (TWAP.sol#1335-1348) uses timestamp for comparisons
angerous comparisons:
 block.timestamp < o.status && block.timestamp > o.filledTime + o.ask.delay && (ERC20(o.ask.srcToken).allowan
ss(this)) < o.srcBidAmountNext() || ERC20(o.ask.srcToken).balanceOf(o.ask.maker) < o.srcBidAmountNext()) (TWA

fyBid(OrderLib.Order,address,uint256,bytes) (TWAP.sol#1358-1375) uses timestamp for comparisons
angerous comparisons:
 require(bool,string)(block.timestamp < o.status,status) (TWAP.sol#1364)
 require(bool,string)(block.timestamp > o.filledTime + o.ask.delay,delay) (TWAP.sol#1365)
 staleBid = block.timestamp > o.bid.time + MAX_BID_WINDOW_SECONDS (TWAP.sol#1371)
 require(bool,string)(staleBid || dstAmountOut > o.bid.dstAmount,low bid) (TWAP.sol#1372)
ormFill(OrderLib.Order) (TWAP.sol#1391-1417) uses timestamp for comparisons
angerous comparisons:
 require(bool,string)(block.timestamp < o.status,status) (TWAP.sol#1401)
 require(bool,string)(block.timestamp > o.bid.time + MIN_BID_WINDOW_SECONDS,pending bid) (TWAP.sol#1402)
: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

Slither Results Log - UniswapV2Exchange

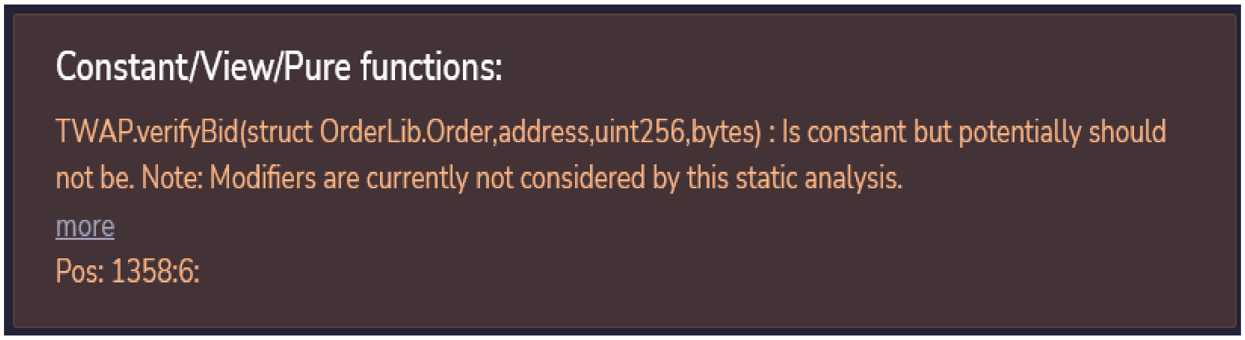


```
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (UniswapV2Exchange.sol#933-942)
getAmountOut(uint256,bytes) should be declared external:
        - UniswapV2Exchange.getAmountOut(uint256,bytes) (UniswapV2Exchange.sol#1130-1133)
swap(uint256,uint256,bytes) should be declared external:
        - UniswapV2Exchange.swap(uint256,uint256,bytes) (UniswapV2Exchange.sol#1138-1151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:UniswapV2Exchange.sol analyzed (11 contracts with 75 detectors), 53 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```



```
Lib.srcBidAmountNext(OrderLib.Order) (UniswapV2Exchange.sol#315-317) is never used and should be removed
RC20.safeApprove(IERC20,address,uint256) (UniswapV2Exchange.sol#664-677) is never used and should be removed
RC20.safeDecreaseAllowance(IERC20,address,uint256) (UniswapV2Exchange.sol#688-699) is never used and should be remov
RC20.safeTransfer(IERC20,address,uint256) (UniswapV2Exchange.sol#640-646) is never used and should be removed
ence: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
Detectors:
a version0.8.4 (UniswapV2Exchange.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.

0.8.4 is not recommended for deployment
ence: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
Detectors:
evel call in Address.sendValue(address,uint256) (UniswapV2Exchange.sol#474-479):
   - (success) = recipient.call{value: amount}() (UniswapV2Exchange.sol#477)
evel call in Address.functionCallWithValue(address,bytes,uint256,string) (UniswapV2Exchange.sol#542-553):
   - (success,returndata) = target.call{value: value}(data) (UniswapV2Exchange.sol#551)
evel call in Address.functionStaticCall(address,bytes,string) (UniswapV2Exchange.sol#571-580):
   - (success,returndata) = target.staticcall(data) (UniswapV2Exchange.sol#578)
evel call in Address.functionDelegateCall(address,bytes,string) (UniswapV2Exchange.sol#598-607):
   - (success,returndata) = target.delegatecall(data) (UniswapV2Exchange.sol#605)
ence: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
Detectors:
) should be declared external:
   - ERC20.name() (UniswapV2Exchange.sol#794-796)
l() should be declared external:
   - ERC20.symbol() (UniswapV2Exchange.sol#802-804)
als() should be declared external:
   - ERC20.decimals() (UniswapV2Exchange.sol#819-821)
Supply() should be declared external:
   - ERC20.totalSupply() (UniswapV2Exchange.sol#826-828)
ceOf(address) should be declared external:
   - ERC20.balanceOf(address) (UniswapV2Exchange.sol#833-835)
fer(address,uint256) should be declared external:
   - ERC20.transfer(address,uint256) (UniswapV2Exchange.sol#845-849)
```

Solidity Static Analysis - Twap

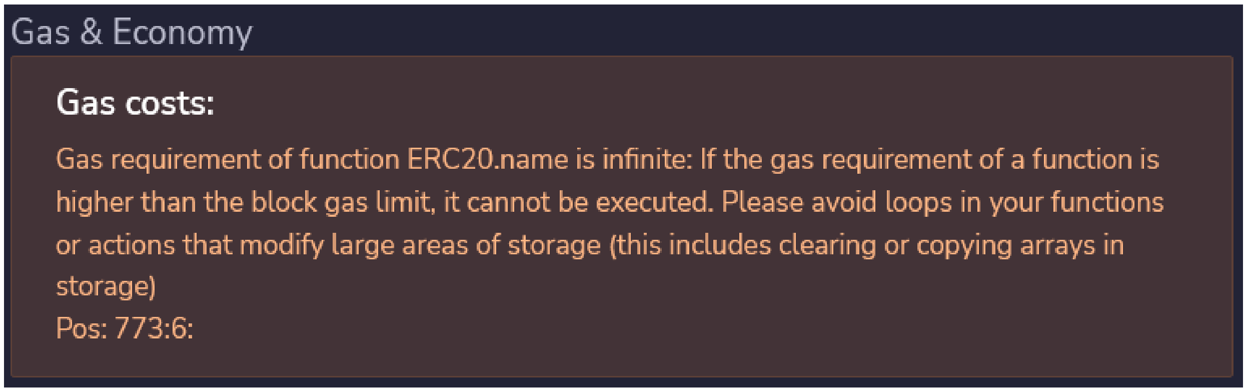


Constant/View/Pure functions:

TWAP.verifyBid(struct OrderLib.Order,address,uint256,bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1358:6:



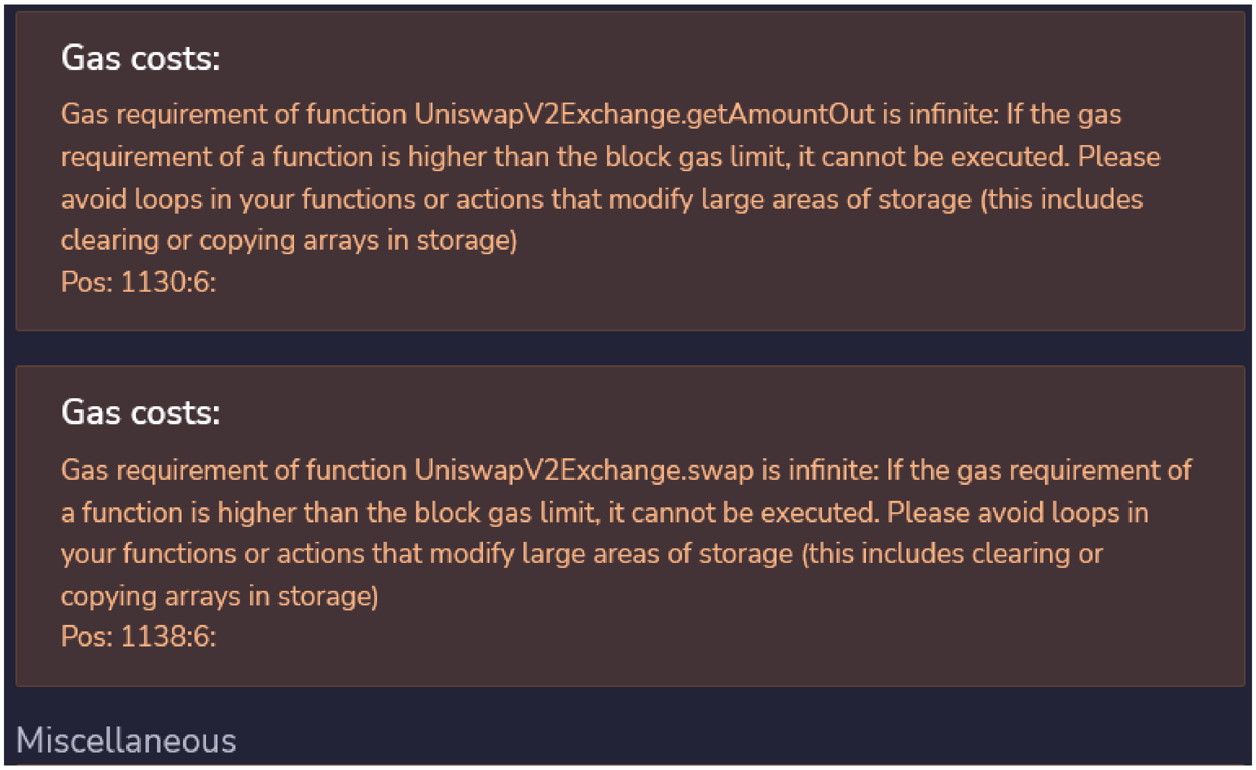
Gas & Economy

Gas costs:

Gas requirement of function ERC20.name 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: 773:6:

Solidity Static Analysis - UniswapV2Exchange



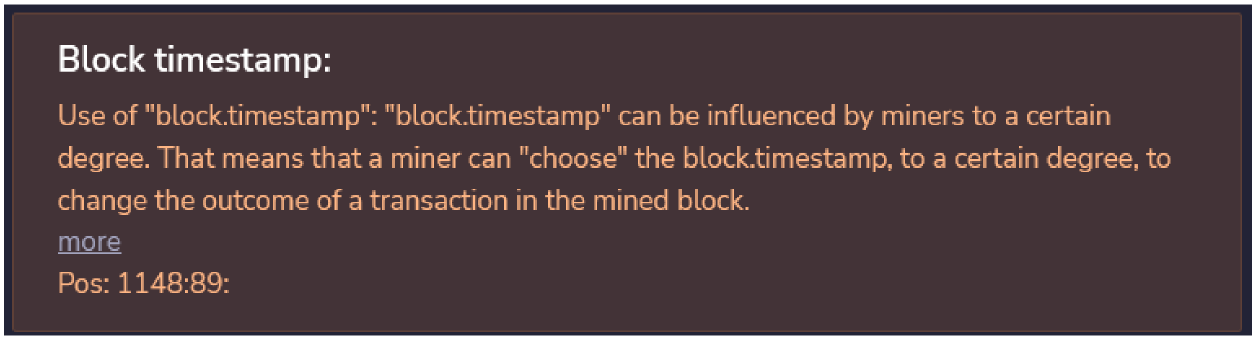
Gas costs:

Gas requirement of function UniswapV2Exchange.getAmountOut 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: 1130:6:

Gas costs:

Gas requirement of function UniswapV2Exchange.swap 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: 1138:6:

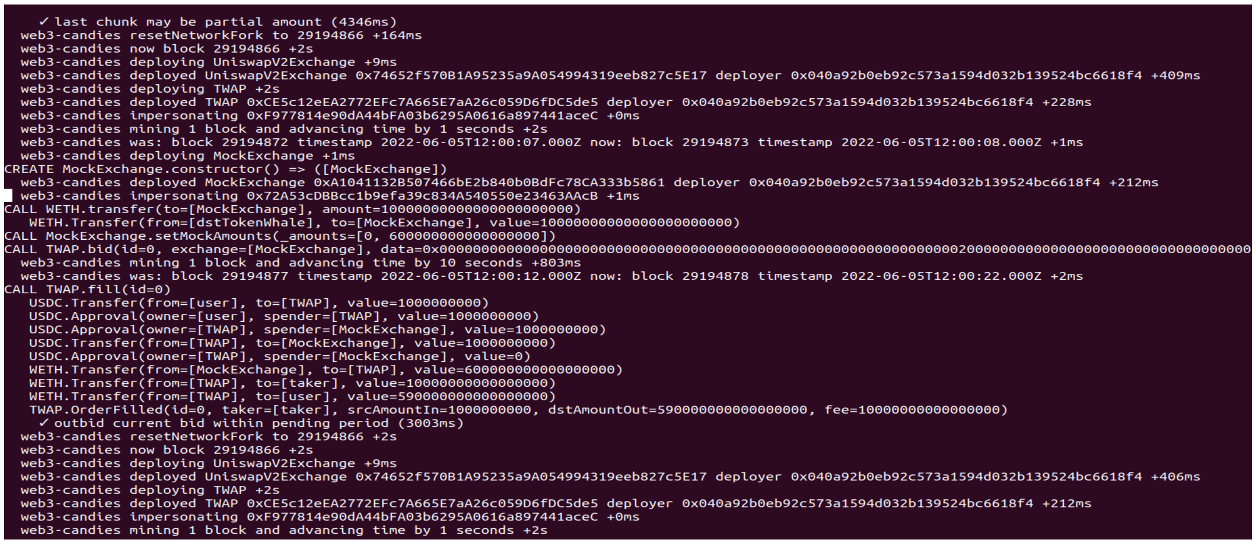
Miscellaneous



Block timestamp:

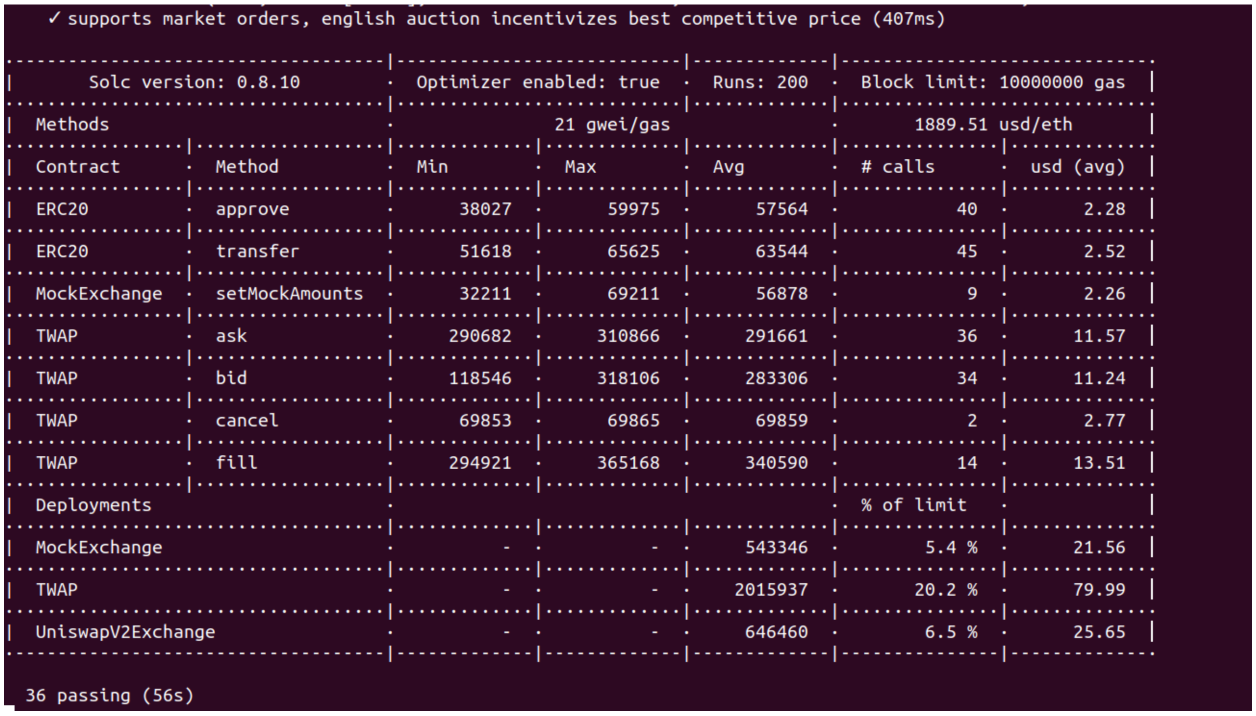
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 1148:89:

Unit Tests



```
    ✓ last chunk may be partial amount (4346ms)
  web3-candies resetNetworkFork to 29194866 +164ms
  web3-candies now block 29194866 +2s
  web3-candies deploying UniswapV2Exchange +9ms
  web3-candies deployed UniswapV2Exchange 0x74652f570B1A95235a9A054994319eeb827c5E17 deployer 0x040a92b0eb92c573a1594d032b139524bc6618f4 +409ms
  web3-candies deploying TWAP +2s
  web3-candies deployed TWAP 0xCE5c12eEA2772EFc7A665E7aA26c059D6fDC5de5 deployer 0x040a92b0eb92c573a1594d032b139524bc6618f4 +228ms
  web3-candies impersonating 0xF977814e90dA44bFA03b6295A0616a897441aceC +0ms
  web3-candies mining 1 block and advancing time by 1 seconds +2s
  web3-candies was: block 29194872 timestamp 2022-06-05T12:00:07.000Z now: block 29194873 timestamp 2022-06-05T12:00:08.000Z +1ms
  web3-candies deploying MockExchange +1ms
CREATE MockExchange.constructor() => ([MockExchange])
  web3-candies deployed MockExchange 0xA1041132B507466bE2b840b0BdFc78CA333b5861 deployer 0x040a92b0eb92c573a1594d032b139524bc6618f4 +212ms
  web3-candies impersonating 0x72A53cDBBcc1b9efa39c834A540550e23463AAcB +1ms
CALL WETH.transfer(to=[MockExchange], amount=1000000000000000000000000)
   WETH.Transfer(from=[dstTokenWhale], to=[MockExchange], value=1000000000000000000000000)
CALL MockExchange.setMockAmounts(_amounts=[0, 600000000000000000])
CALL TWAP.bid(id=0, exchange=[MockExchange], data=0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000
  web3-candies mining 1 block and advancing time by 10 seconds +803ms
  web3-candies was: block 29194877 timestamp 2022-06-05T12:00:12.000Z now: block 29194878 timestamp 2022-06-05T12:00:22.000Z +2ms
CALL TWAP.fill(id=0)
   USDC.Transfer(from=[user], to=[TWAP], value=1000000000)
   USDC.Approval(owner=[user], spender=[TWAP], value=1000000000)
   USDC.Approval(owner=[TWAP], spender=[MockExchange], value=1000000000)
   USDC.Transfer(from=[TWAP], to=[MockExchange], value=1000000000)
   USDC.Approval(owner=[TWAP], spender=[MockExchange], value=0)
   WETH.Transfer(from=[MockExchange], to=[TWAP], value=600000000000000000)
   WETH.Transfer(from=[TWAP], to=[taker], value=10000000000000000)
   WETH.Transfer(from=[TWAP], to=[user], value=590000000000000000)
   TWAP.OrderFilled(id=0, taker=[taker], srcAmountIn=1000000000, dstAmountOut=590000000000000000, fee=10000000000000000)
    ✓ outbid current bid within pending period (3003ms)
  web3-candies resetNetworkFork to 29194866 +2s
  web3-candies now block 29194866 +2s
  web3-candies deploying UniswapV2Exchange +9ms
  web3-candies deployed UniswapV2Exchange 0x74652f570B1A95235a9A054994319eeb827c5E17 deployer 0x040a92b0eb92c573a1594d032b139524bc6618f4 +406ms
  web3-candies deploying TWAP +2s
  web3-candies deployed TWAP 0xCE5c12eEA2772EFc7A665E7aA26c059D6fDC5de5 deployer 0x040a92b0eb92c573a1594d032b139524bc6618f4 +212ms
  web3-candies impersonating 0xF977814e90dA44bFA03b6295A0616a897441aceC +0ms
  web3-candies mining 1 block and advancing time by 1 seconds +2s
```

`



```
    ✓ supports market orders, english auction incentivizes best competitive price (407ms)

·-----------------------------------|---------------------------|-------------|-----------------------------·
|        Solc version: 0.8.10       ·  Optimizer enabled: true  ·  Runs: 200  ·  Block limit: 10000000 gas  │
····································|···························|·············|······························
|  Methods                          ·               21 gwei/gas               ·       1889.51 usd/eth       │
·················|··················|·············|·············|·············|···············|··············
|  Contract      ·  Method          ·  Min        ·  Max        ·  Avg        ·  # calls      ·  usd (avg)  │
·················|··················|·············|·············|·············|···············|··············
|  ERC20         ·  approve         ·      38027  ·      59975  ·      57564  ·           40  ·       2.28  │
·················|··················|·············|·············|·············|···············|··············
|  ERC20         ·  transfer        ·      51618  ·      65625  ·      63544  ·           45  ·       2.52  │
·················|··················|·············|·············|·············|···············|··············
|  MockExchange  ·  setMockAmounts  ·      32211  ·      69211  ·      56878  ·            9  ·       2.26  │
·················|··················|·············|·············|·············|···············|··············
|  TWAP          ·  ask             ·     290682  ·     310866  ·     291661  ·           36  ·      11.57  │
·················|··················|·············|·············|·············|···············|··············
|  TWAP          ·  bid             ·     118546  ·     318106  ·     283306  ·           34  ·      11.24  │
·················|··················|·············|·············|·············|···············|··············
|  TWAP          ·  cancel          ·      69853  ·      69865  ·      69859  ·            2  ·       2.77  │
·················|··················|·············|·············|·············|···············|··············
|  TWAP          ·  fill            ·     294921  ·     365168  ·     340590  ·           14  ·      13.51  │
·················|··················|·············|·············|·············|···············|··············
|  Deployments                      ·                                         ·  % of limit   ·             │
····································|·············|·············|·············|···············|··············
|  MockExchange                     ·          -  ·          -  ·     543346  ·        5.4 %  ·      21.56  │
····································|·············|·············|·············|···············|··············
|  TWAP                             ·          -  ·          -  ·    2015937  ·       20.2 %  ·      79.99  │
····································|·············|·············|·············|···············|··············
|  UniswapV2Exchange                ·          -  ·          -  ·     646460  ·        6.5 %  ·      25.65  │
·-----------------------------------|-------------|-------------|-------------|---------------|-------------·

  36 passing (56s)
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens 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 functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

Conclusion

We were given a contract file and have used all possible tests based on the given object. So it is ready to go for production. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is “**Well-Secured**”.

Note For Contract Users

Please do your due diligence before investing. Our audit report is never an investment advice.

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 analyse 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 scrutinised 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

RD Auditors Disclaimer

The smart contracts given for audit have been analysed 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).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the 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 recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. 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.

RD
AUDITORS

Email: info@rdauditors.com
Website: www.rdauditors.com