EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	WishFinance
Website:	www.wishfinance.io
Platform:	Binance Smart Chain
Language:	Solidity
Date:	August 12th, 2021

Table of contents

Introduction

EtherAuthority was contracted by the WishFinance team to perform the Security audit of the WishFinance smart contract 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 August 12th, 2021.

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 Wish Finance - A next evolution DeFi exchange on Binance Smart Chain (BSC). Wish tokens bear the WISH ticket on the Binance Smart Chain (BSC). It complies with the BEP20 standard. WISH has a fixed circulation, in order to ensure that there will not be a case of hyperinflation caused by an ever growing supply. On that same note, WISH tokens will still benefit from deflationary measures which will be outlined in a later section.

Audit scope

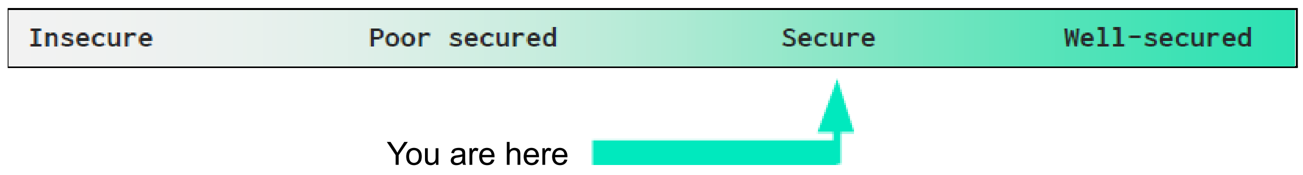
Name	**Code Review and Security Analysis Report for WishFinance Smart Contract**
Platform	**BSC / Solidity**
File	WishFinance.sol
Smart Contract Online Code	https://testnet.bscscan.com/address/0xE0Ddfec03C665252fAc7893B9359821DAFDEac25#code
File MD5 Hash	AD59821727C529DF139174BEAD811EB3
Audit Date	August 12th, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: WishFinance	**YES, This is valid.**
Symbol: WISH	**YES, This is valid.**
Decimals: 18	**YES, This is valid.**
Project Tax: 0.5%	**YES, This is valid.**
LP Tax: 0.5%	**YES, This is valid.**
Total Supply: 10000	**YES, This is valid.**
Number of Tokens Sell To Add To Liquidity: 10000	**YES, This is valid.**
The Operator can handle function like: changeProjectWallet, setOperator, setLiquification, enableDisableFee, setNumTokensSellToAddToLiquidity, addOrRemoveFromWhitelist, etc.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.



We used various tools like MythX, Slither 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.

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	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	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	Passed
	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
	Other code specification issues	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 1 smart contract. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in WishFinance 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 WishFinance Token token.

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

Some code parts are **not well** commented on smart contracts.

Documentation

We were given a WishFinance smart contracts code in the form of a BscScn web link. The hash of that code is mentioned above in the table.

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

Another source of information was its official website https://www.wishfinance.io/ which provided rich information about the project architecture and tokenomics.

Use of Dependencies

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

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

AS-IS overview

WishFinance.sol

(1) Interface

(a) IBEP20

(b) IUniswapV2Factory

(c) IUniswapV2Pair

(d) IUniswapV2Router01

(e) IUniswapV2Router02

(2) Inherited contracts

(a) Context

(b) Ownable

(3) Usages

(a) using SafeMath for uint256;

(4) Events

(a) event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);

(b) event SwapAndLiquifyEnabledUpdated(bool enabled);

(c) event SwapAndLiquify(uint256 tokensSwapped,uint256 ethReceived,uint256 tokensIntoLiqudity);

(5) Functions

Sl.	Functions	Type	Observation	Conclusion
1	onlyOperator	modifier	Passed	No Issue
2	receive	external	Passed	No Issue
3	changeProjectWallet	write	Critical operation lacks event log	Refer audit finding section
4	setNumTokensSellToAddToLiquidity	write	access only Operator	No Issue
5	addOrRemoveFromWhitelist	write	access only Operator	No Issue
6	setOperator	write	access only Operator	No Issue
7	enableDisableFee	write	access only Operator	No Issue
8	setLiquification	write	access only Operator	No Issue
9	mint	write	access only Operator	No Issue
10	name	read	Passed	No Issue
11	symbol	read	Passed	No Issue

12	decimals	read	Passed	No Issue
13	totalSupply	read	Passed	No Issue
14	balanceOf	read	Passed	No Issue
15	transfer	write	Passed	No Issue
16	allowance	read	Passed	No Issue
17	approve	write	Passed	No Issue
18	transferFrom	write	Passed	No Issue
19	increaseAllowance	write	Passed	No Issue
20	decreaseAllowance	write	Passed	No Issue
21	mint	internal	access only Owner	No Issue
22	_transfer	internal	Passed	No Issue
23	_mint	internal	Passed	No Issue
24	_burn	internal	Passed	No Issue
25	_approve	internal	Passed	No Issue
26	_burnFrom	internal	Passed	No Issue
27	swapAndLiquify	write	Passed	No Issue
28	swapTokensForEth	write	Passed	No Issue
29	addLiquidity	write	Centralized risk in addLiquidity	Refer audit finding section
30	owner	read	Passed	No Issue
31	onlyOwner	modifier	Passed	No Issue
32	renounceOwnership	write	access only Owner	No Issue
33	transferOwnership	write	access only Owner	No Issue
34	_msgSender	internal	Passed	No Issue
35	_msgData	internal	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.

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Function input parameters lack of check:

Variable validation is not performed in below functions :

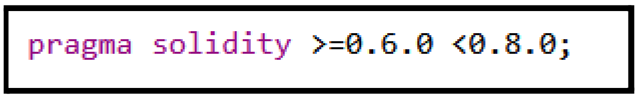
transfer, mint, burn.

Resolution: There should be some validations to check the variable is not empty or greater than 0.

Status: Open

Very Low / Discussion / Best practices:

(1) Use latest solidity version:



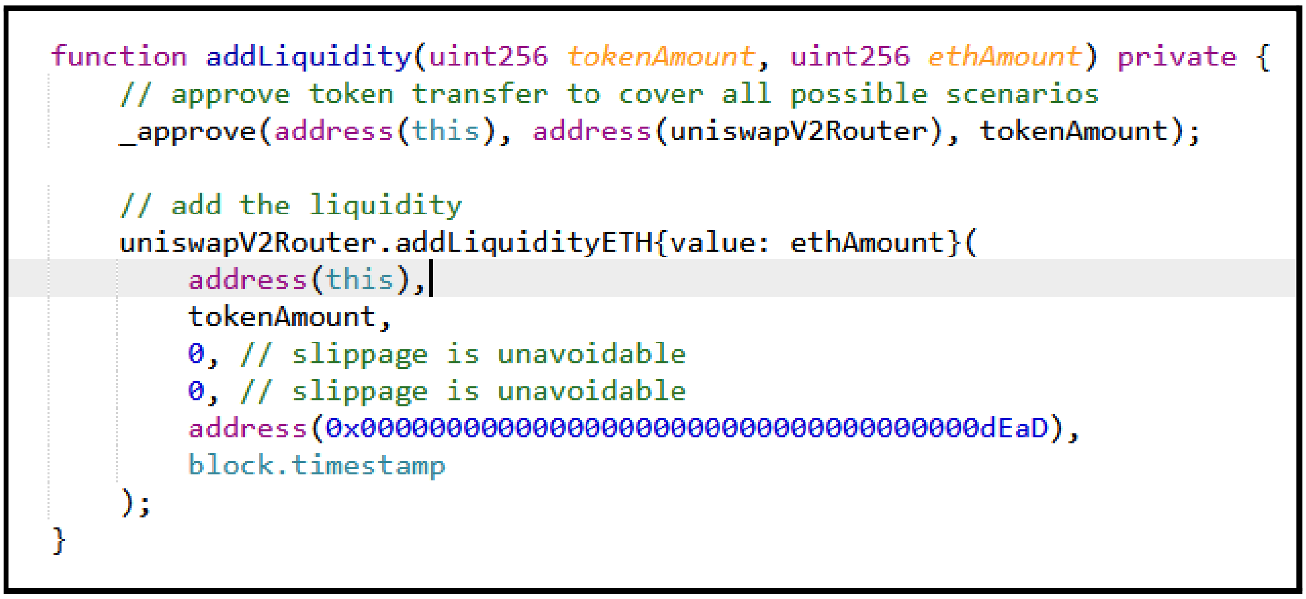
```
pragma solidity >=0.6.0 <0.8.0;
```

Using the latest solidity will prevent any compiler level bugs.

Resolution: Please use 0.8.6 which is the latest version.

Status: Open

(2) Risk in addLiquidity:



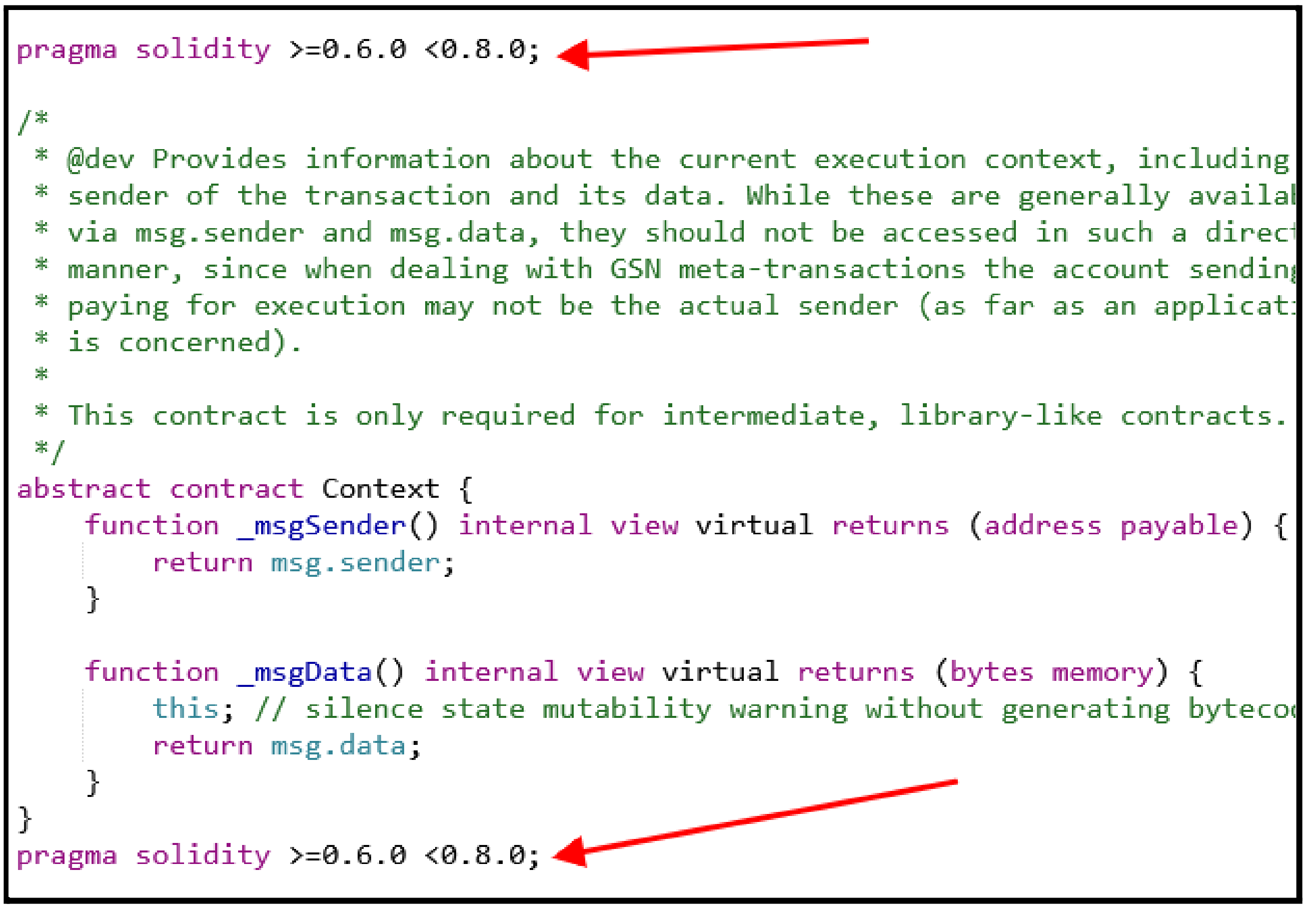
```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        address(0x000000000000000000000000000000000000dEaD),
        block.timestamp
    );
}
```

In addLiquidity function, tokens are sent to black hole.

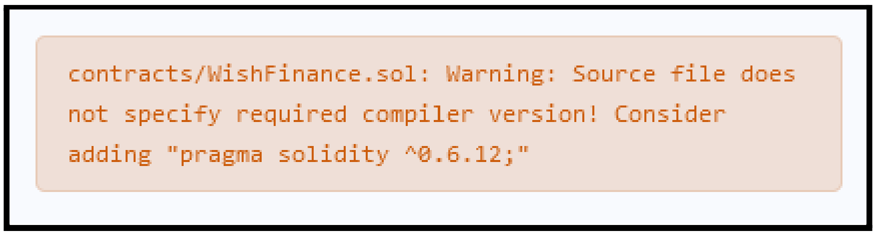
Resolution: If it is a part of the plan then disregard this point.

Status: Open

(3) Multiple Pragma added:



```
pragma solidity >=0.6.0 <0.8.0;

/*
 * @dev Provides information about the current execution context, including
 * sender of the transaction and its data. While these are generally availal
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sendin
 * paying for execution may not be the actual sender (as far as an applicat
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating byteco
        return msg.data;
    }
}
pragma solidity >=0.6.0 <0.8.0;
```



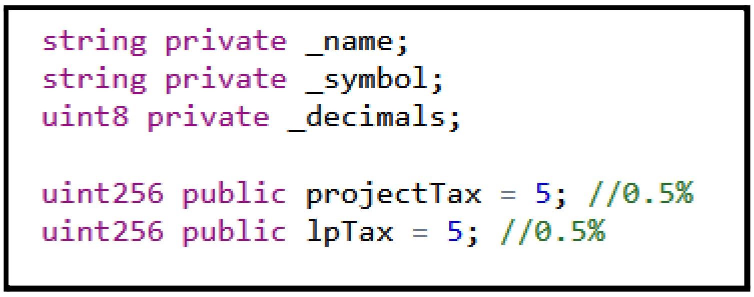
```
contracts/WishFinance.sol: Warning: Source file does
not specify required compiler version! Consider
adding "pragma solidity ^0.6.12;"
```

we should keep only one pragma.

Resolution: Please remove multiple pragmas and keep one at the top of the code.

Status: Open

(4) Make variables constant:



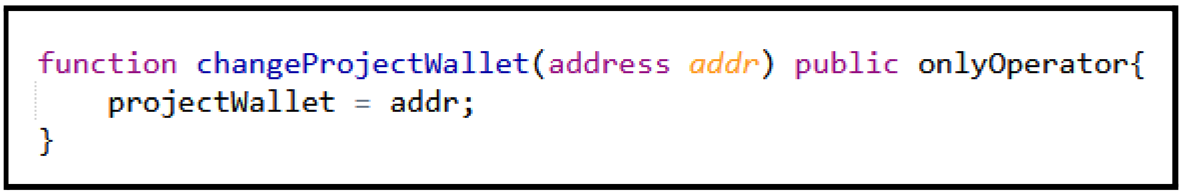
```
string private _name;
string private _symbol;
uint8 private _decimals;

uint256 public projectTax = 5; //0.5%
uint256 public lpTax = 5; //0.5%
```

name, symbol, decimals,lpTax, projectTax. These variables will not be changed. So, please make it constant. It will save some gas.

Resolution: Declare those variables as constant.

Status: Open

(5) Critical operation lacks event log:



```
function changeProjectWallet(address addr) public onlyOperator{
    projectWallet = addr;
}
```

Missing event log for : changeProjectWallet.

Resolution: Please write an event log for listed events.

Status: Open

Centralization

These smart contracts have some functions which can be executed by Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- mint: The Owner can set the user wallet address and amount.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those issues are fixed in revised code. So, **it's good to go to production**.

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 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).

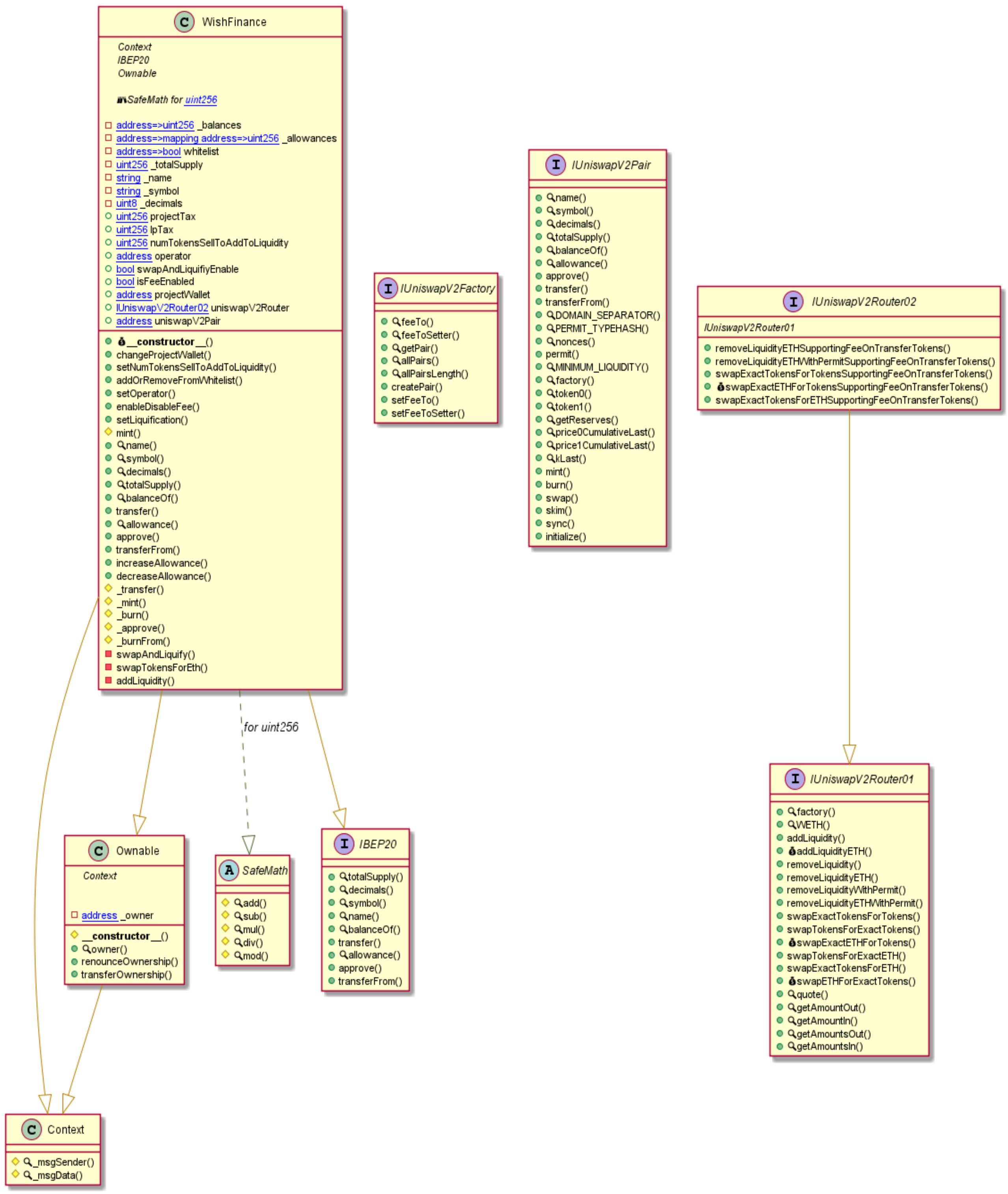
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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.

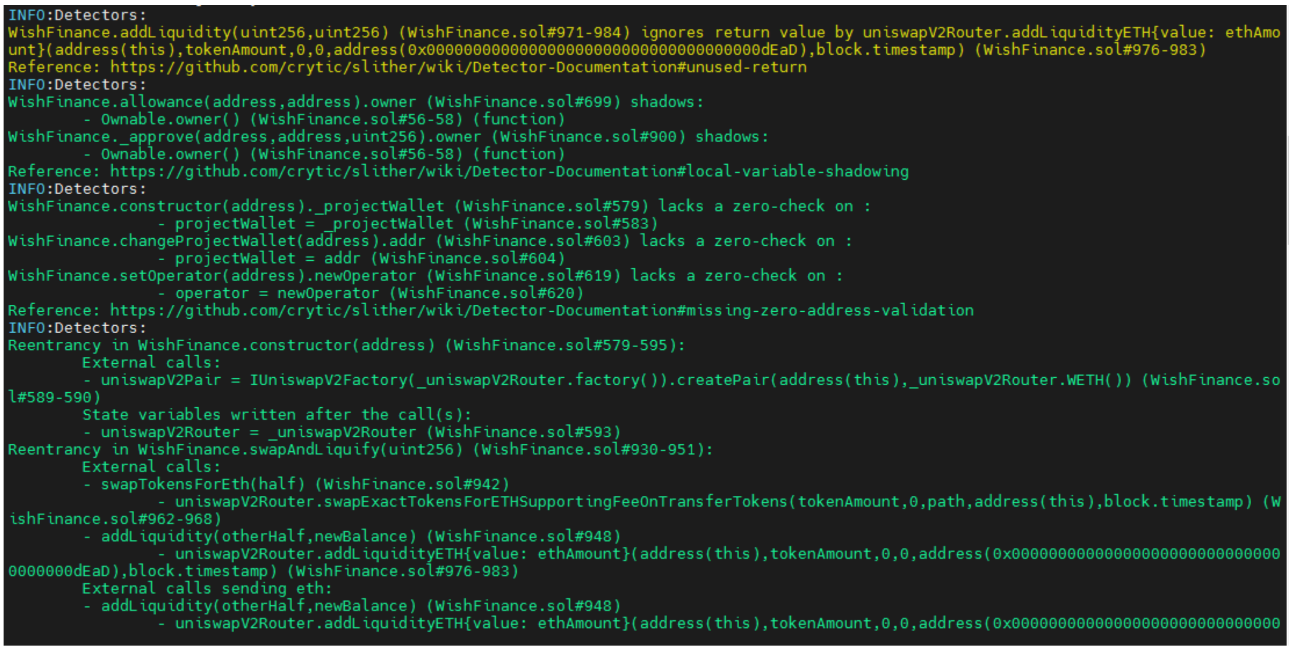
Appendix

Code Flow Diagram - WishFinance

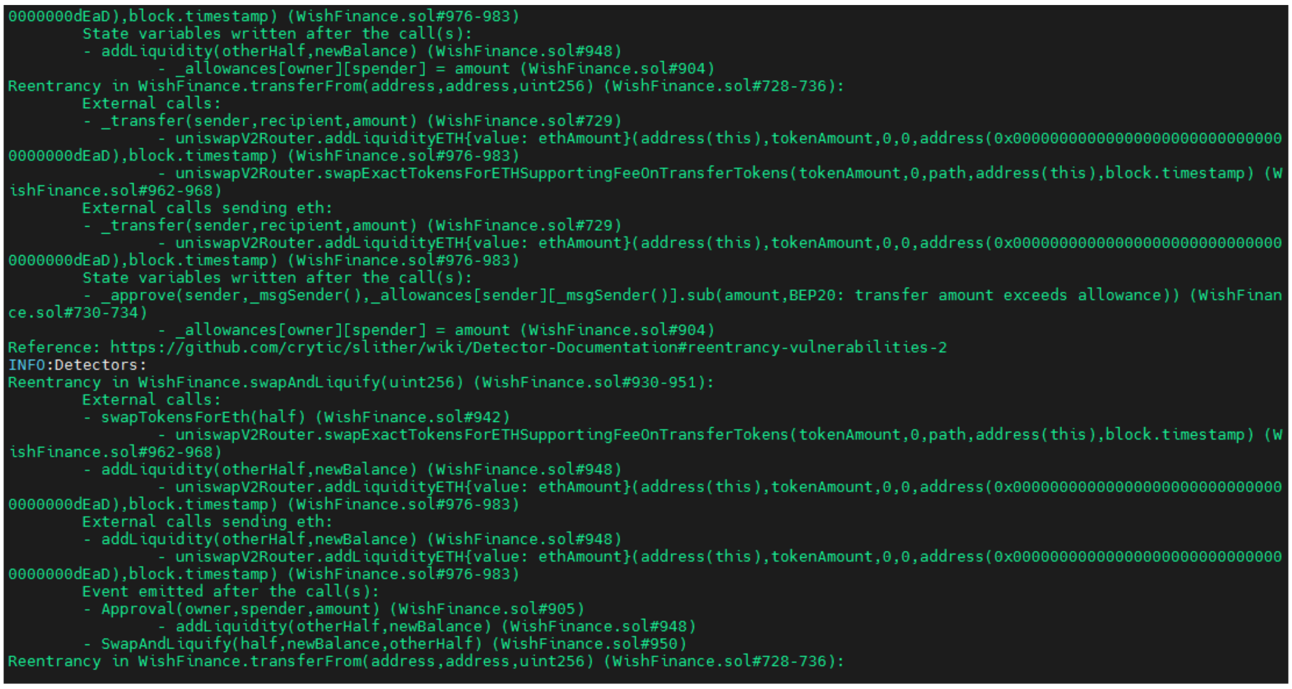


Slither Results Log

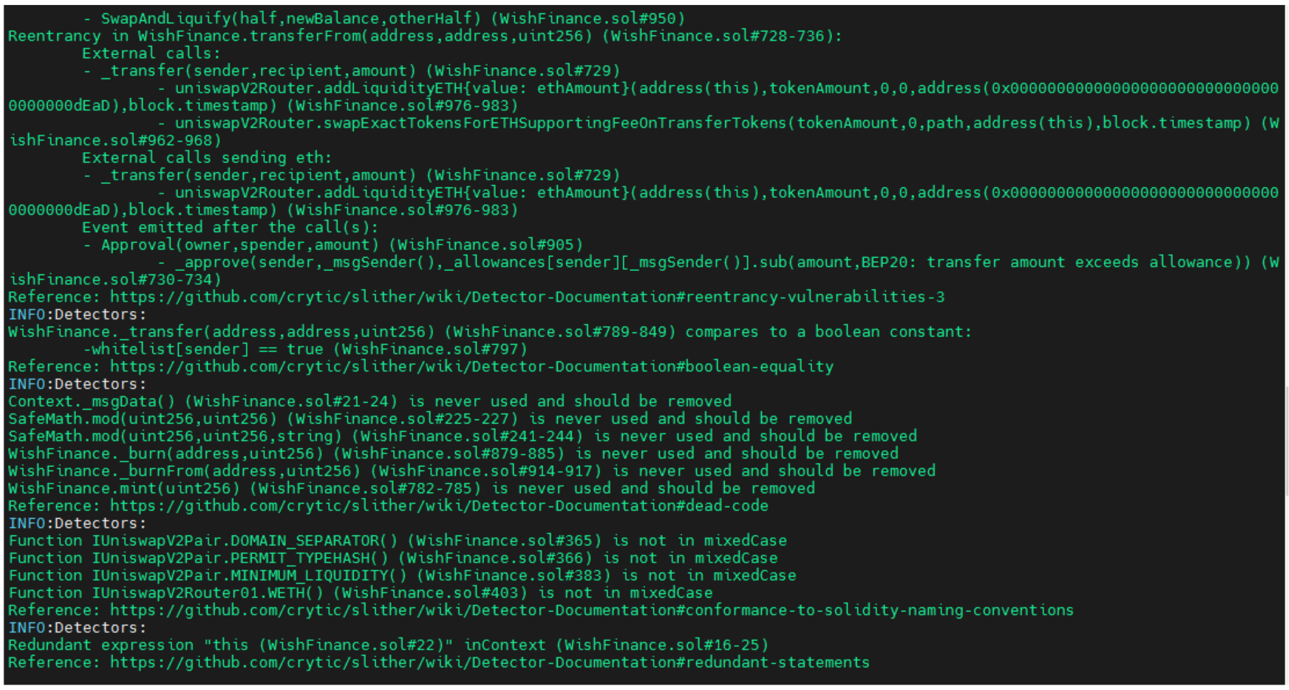
Slither log >> WishFinance.sol



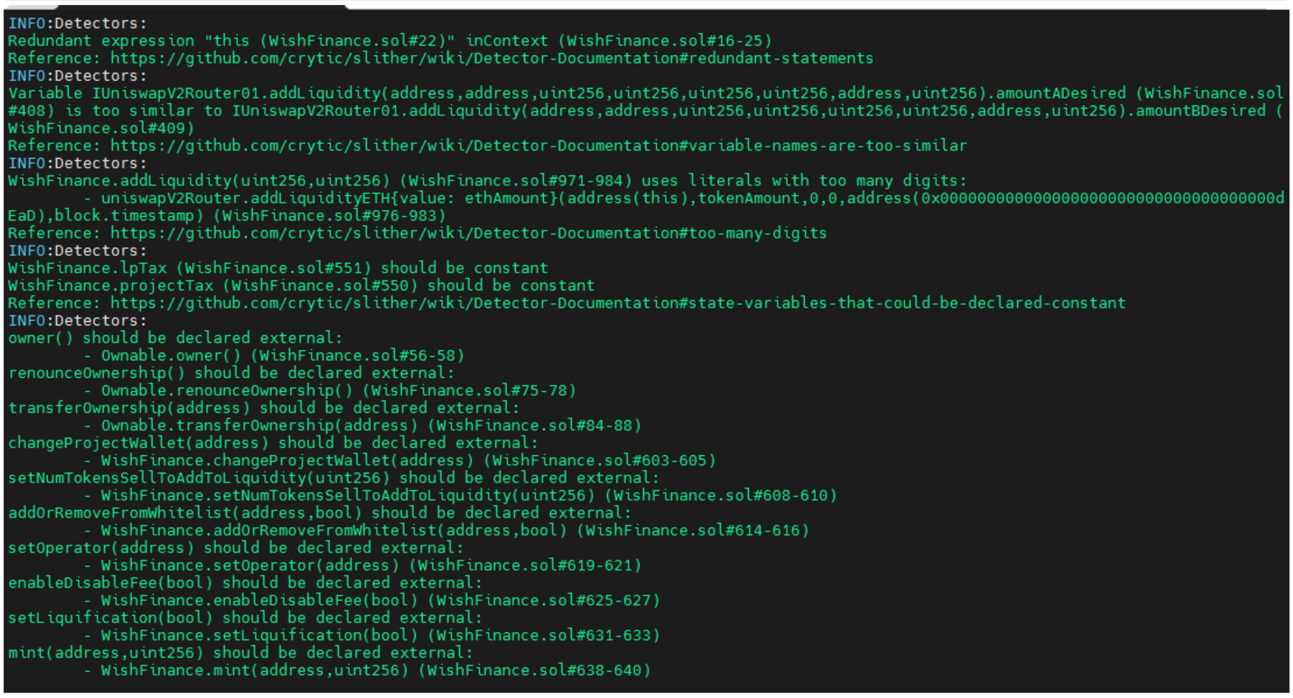
```
INFO:Detectors:
WishFinance.addLiquidity(uint256,uint256) (WishFinance.sol#971-984) ignores return value by uniswapV2Router.addLiquidityETH{value: ethAmo
unt}(address(this),tokenAmount,0,0,address(0x000000000000000000000000000000000000dEaD),block.timestamp) (WishFinance.sol#976-983)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
WishFinance.allowance(address,address).owner (WishFinance.sol#699) shadows:
        - Ownable.owner() (WishFinance.sol#56-58) (function)
WishFinance._approve(address,address,uint256).owner (WishFinance.sol#900) shadows:
        - Ownable.owner() (WishFinance.sol#56-58) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
WishFinance.constructor(address)._projectWallet (WishFinance.sol#579) lacks a zero-check on :
                - projectWallet = _projectWallet (WishFinance.sol#583)
WishFinance.changeProjectWallet(address).addr (WishFinance.sol#603) lacks a zero-check on :
                - projectWallet = addr (WishFinance.sol#604)
WishFinance.setOperator(address).newOperator (WishFinance.sol#619) lacks a zero-check on :
                - operator = newOperator (WishFinance.sol#620)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in WishFinance.constructor(address) (WishFinance.sol#579-595):
        External calls:
        - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (WishFinance.so
l#589-590)
        State variables written after the call(s):
        - uniswapV2Router = _uniswapV2Router (WishFinance.sol#593)
Reentrancy in WishFinance.swapAndLiquify(uint256) (WishFinance.sol#930-951):
        External calls:
        - swapTokensForEth(half) (WishFinance.sol#942)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
ishFinance.sol#962-968)
        - addLiquidity(otherHalf,newBalance) (WishFinance.sol#948)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x00000000000000000000000000000
0000000dEaD),block.timestamp) (WishFinance.sol#976-983)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (WishFinance.sol#948)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x00000000000000000000000000000
```



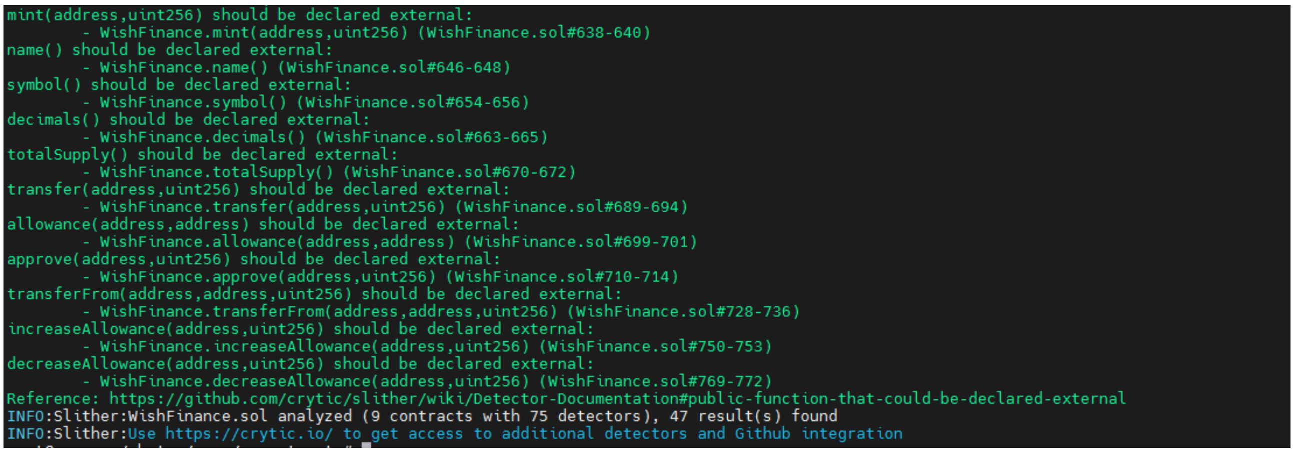
```
0000000dEaD),block.timestamp) (WishFinance.sol#976-983)
        State variables written after the call(s):
        - addLiquidity(otherHalf,newBalance) (WishFinance.sol#948)
                - _allowances[owner][spender] = amount (WishFinance.sol#904)
Reentrancy in WishFinance.transferFrom(address,address,uint256) (WishFinance.sol#728-736):
        External calls:
        - _transfer(sender,recipient,amount) (WishFinance.sol#729)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x00000000000000000000000000000
0000000dEaD),block.timestamp) (WishFinance.sol#976-983)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
ishFinance.sol#962-968)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (WishFinance.sol#729)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x00000000000000000000000000000
0000000dEaD),block.timestamp) (WishFinance.sol#976-983)
        State variables written after the call(s):
        - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,BEP20: transfer amount exceeds allowance)) (WishFinan
ce.sol#730-734)
                - _allowances[owner][spender] = amount (WishFinance.sol#904)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in WishFinance.swapAndLiquify(uint256) (WishFinance.sol#930-951):
        External calls:
        - swapTokensForEth(half) (WishFinance.sol#942)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
ishFinance.sol#962-968)
        - addLiquidity(otherHalf,newBalance) (WishFinance.sol#948)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x00000000000000000000000000000
0000000dEaD),block.timestamp) (WishFinance.sol#976-983)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (WishFinance.sol#948)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x00000000000000000000000000000
0000000dEaD),block.timestamp) (WishFinance.sol#976-983)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WishFinance.sol#905)
                - addLiquidity(otherHalf,newBalance) (WishFinance.sol#948)
        - SwapAndLiquify(half,newBalance,otherHalf) (WishFinance.sol#950)
Reentrancy in WishFinance.transferFrom(address,address,uint256) (WishFinance.sol#728-736):
```



```
        - SwapAndLiquify(half,newBalance,otherHalf) (WishFinance.sol#950)
Reentrancy in WishFinance.transferFrom(address,address,uint256) (WishFinance.sol#728-736):
        External calls:
        - _transfer(sender,recipient,amount) (WishFinance.sol#729)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x0000000000000000000000000000
0000000dEaD),block.timestamp) (WishFinance.sol#976-983)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
ishFinance.sol#962-968)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (WishFinance.sol#729)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x0000000000000000000000000000
0000000dEaD),block.timestamp) (WishFinance.sol#976-983)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WishFinance.sol#905)
                - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,BEP20: transfer amount exceeds allowance)) (W
ishFinance.sol#730-734)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
WishFinance._transfer(address,address,uint256) (WishFinance.sol#789-849) compares to a boolean constant:
        -whitelist[sender] == true (WishFinance.sol#797)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Context._msgData() (WishFinance.sol#21-24) is never used and should be removed
SafeMath.mod(uint256,uint256) (WishFinance.sol#225-227) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (WishFinance.sol#241-244) is never used and should be removed
WishFinance._burn(address,uint256) (WishFinance.sol#879-885) is never used and should be removed
WishFinance._burnFrom(address,uint256) (WishFinance.sol#914-917) is never used and should be removed
WishFinance.mint(uint256) (WishFinance.sol#782-785) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (WishFinance.sol#365) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (WishFinance.sol#366) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (WishFinance.sol#383) is not in mixedCase
Function IUniswapV2Router01.WETH() (WishFinance.sol#403) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (WishFinance.sol#22)" inContext (WishFinance.sol#16-25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```



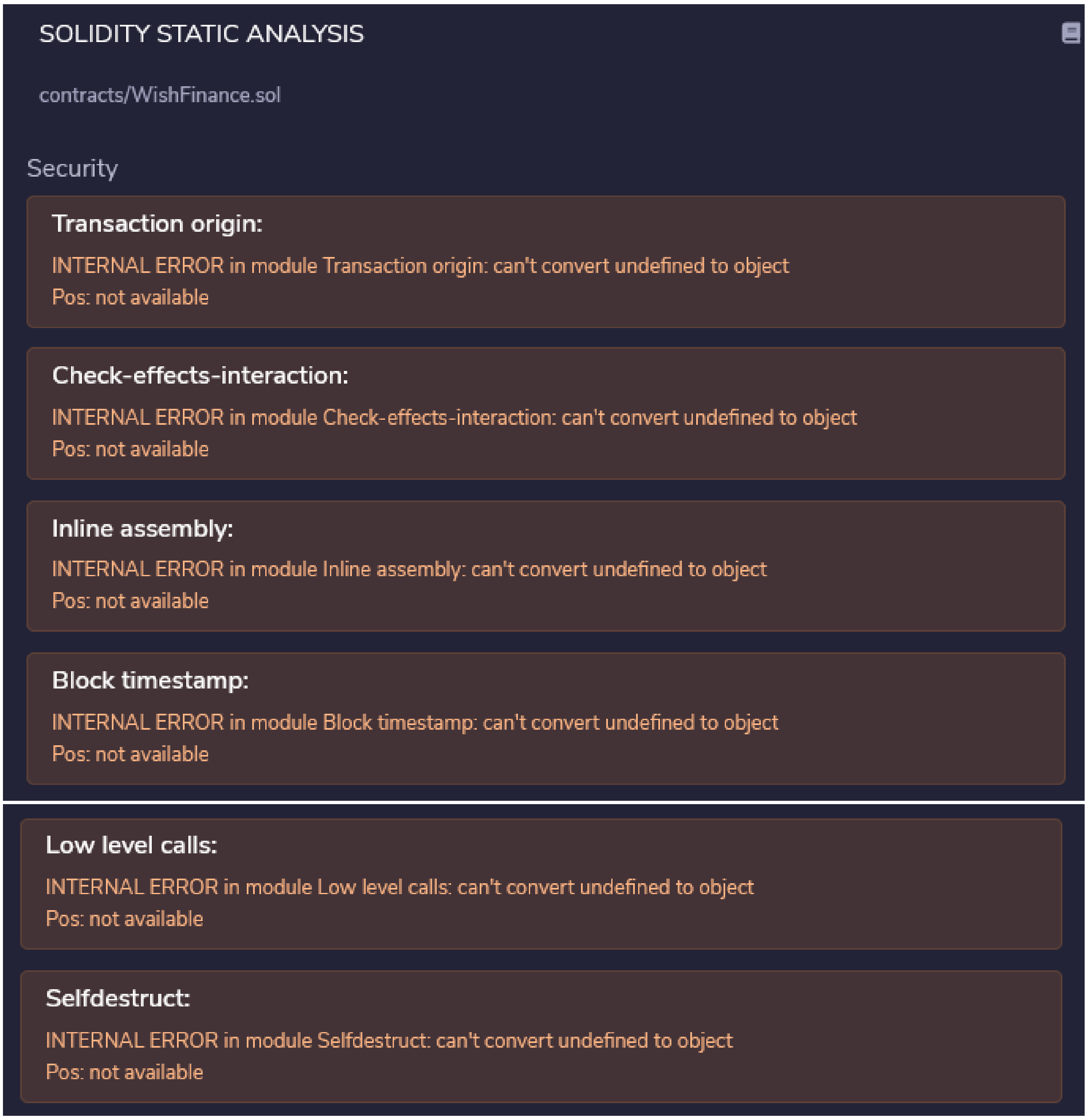
```
INFO:Detectors:
Redundant expression "this (WishFinance.sol#22)" inContext (WishFinance.sol#16-25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (WishFinance.sol
#408) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (
WishFinance.sol#409)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
WishFinance.addLiquidity(uint256,uint256) (WishFinance.sol#971-984) uses literals with too many digits:
        - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0x000000000000000000000000000000000000d
EaD),block.timestamp) (WishFinance.sol#976-983)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
WishFinance.lpTax (WishFinance.sol#551) should be constant
WishFinance.projectTax (WishFinance.sol#550) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner() (WishFinance.sol#56-58)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (WishFinance.sol#75-78)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (WishFinance.sol#84-88)
changeProjectWallet(address) should be declared external:
        - WishFinance.changeProjectWallet(address) (WishFinance.sol#603-605)
setNumTokensSellToAddToLiquidity(uint256) should be declared external:
        - WishFinance.setNumTokensSellToAddToLiquidity(uint256) (WishFinance.sol#608-610)
addOrRemoveFromWhitelist(address,bool) should be declared external:
        - WishFinance.addOrRemoveFromWhitelist(address,bool) (WishFinance.sol#614-616)
setOperator(address) should be declared external:
        - WishFinance.setOperator(address) (WishFinance.sol#619-621)
enableDisableFee(bool) should be declared external:
        - WishFinance.enableDisableFee(bool) (WishFinance.sol#625-627)
setLiquification(bool) should be declared external:
        - WishFinance.setLiquification(bool) (WishFinance.sol#631-633)
mint(address,uint256) should be declared external:
        - WishFinance.mint(address,uint256) (WishFinance.sol#638-640)
```



```
mint(address,uint256) should be declared external:
        - WishFinance.mint(address,uint256) (WishFinance.sol#638-640)
name() should be declared external:
        - WishFinance.name() (WishFinance.sol#646-648)
symbol() should be declared external:
        - WishFinance.symbol() (WishFinance.sol#654-656)
decimals() should be declared external:
        - WishFinance.decimals() (WishFinance.sol#663-665)
totalSupply() should be declared external:
        - WishFinance.totalSupply() (WishFinance.sol#670-672)
transfer(address,uint256) should be declared external:
        - WishFinance.transfer(address,uint256) (WishFinance.sol#689-694)
allowance(address,address) should be declared external:
        - WishFinance.allowance(address,address) (WishFinance.sol#699-701)
approve(address,uint256) should be declared external:
        - WishFinance.approve(address,uint256) (WishFinance.sol#710-714)
transferFrom(address,address,uint256) should be declared external:
        - WishFinance.transferFrom(address,address,uint256) (WishFinance.sol#728-736)
increaseAllowance(address,uint256) should be declared external:
        - WishFinance.increaseAllowance(address,uint256) (WishFinance.sol#750-753)
decreaseAllowance(address,uint256) should be declared external:
        - WishFinance.decreaseAllowance(address,uint256) (WishFinance.sol#769-772)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:WishFinance.sol analyzed (9 contracts with 75 detectors), 47 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity static analysis

WishFinance.sol



SOLIDITY STATIC ANALYSIS

contracts/WishFinance.sol

Security

Transaction origin:

INTERNAL ERROR in module Transaction origin: can't convert undefined to object
Pos: not available

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: can't convert undefined to object
Pos: not available

Inline assembly:

INTERNAL ERROR in module Inline assembly: can't convert undefined to object
Pos: not available

Block timestamp:

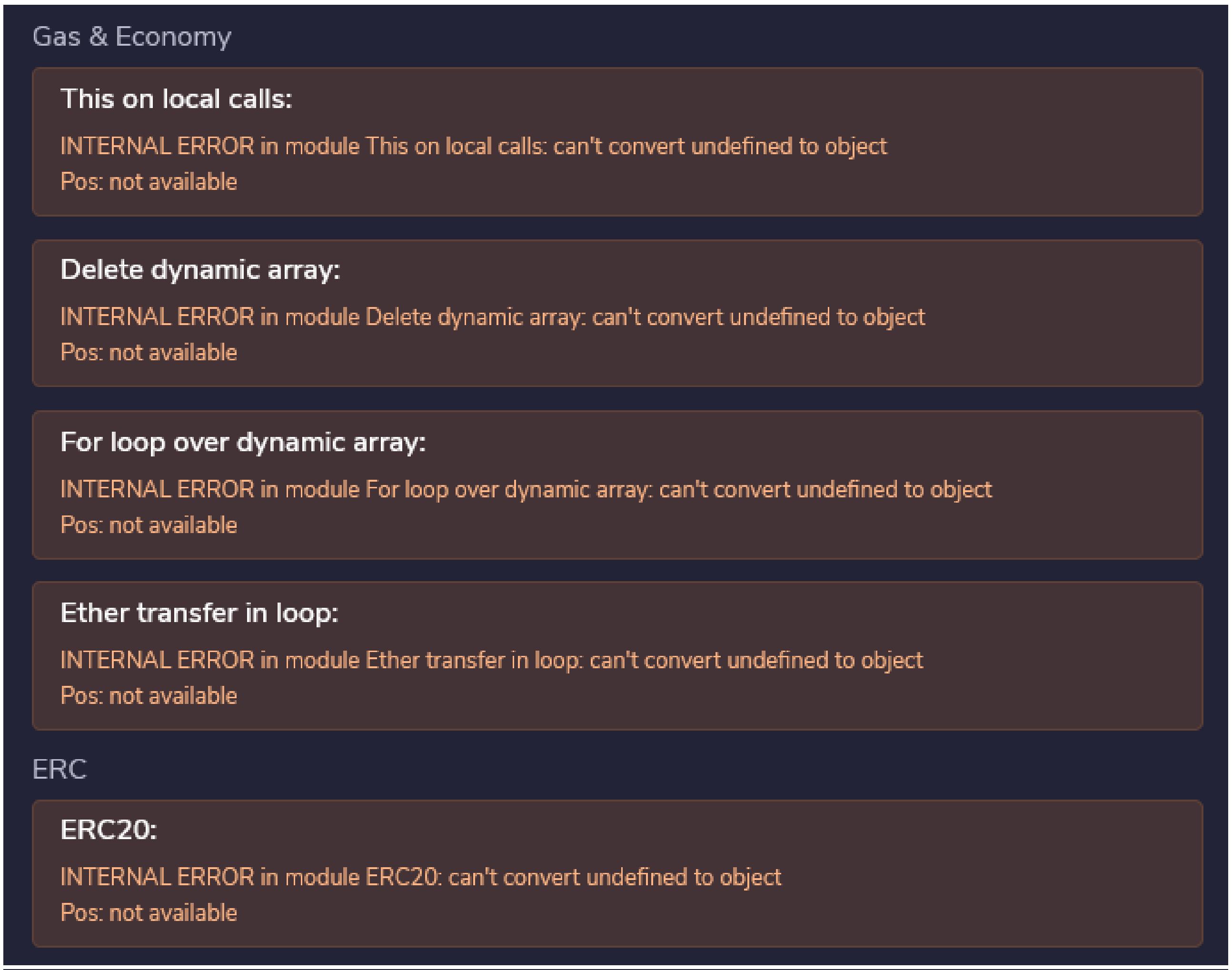
INTERNAL ERROR in module Block timestamp: can't convert undefined to object
Pos: not available

Low level calls:

INTERNAL ERROR in module Low level calls: can't convert undefined to object
Pos: not available

Selfdestruct:

INTERNAL ERROR in module Selfdestruct: can't convert undefined to object
Pos: not available



Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: can't convert undefined to object
Pos: not available

Delete dynamic array:

INTERNAL ERROR in module Delete dynamic array: can't convert undefined to object
Pos: not available

For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: can't convert undefined to object
Pos: not available

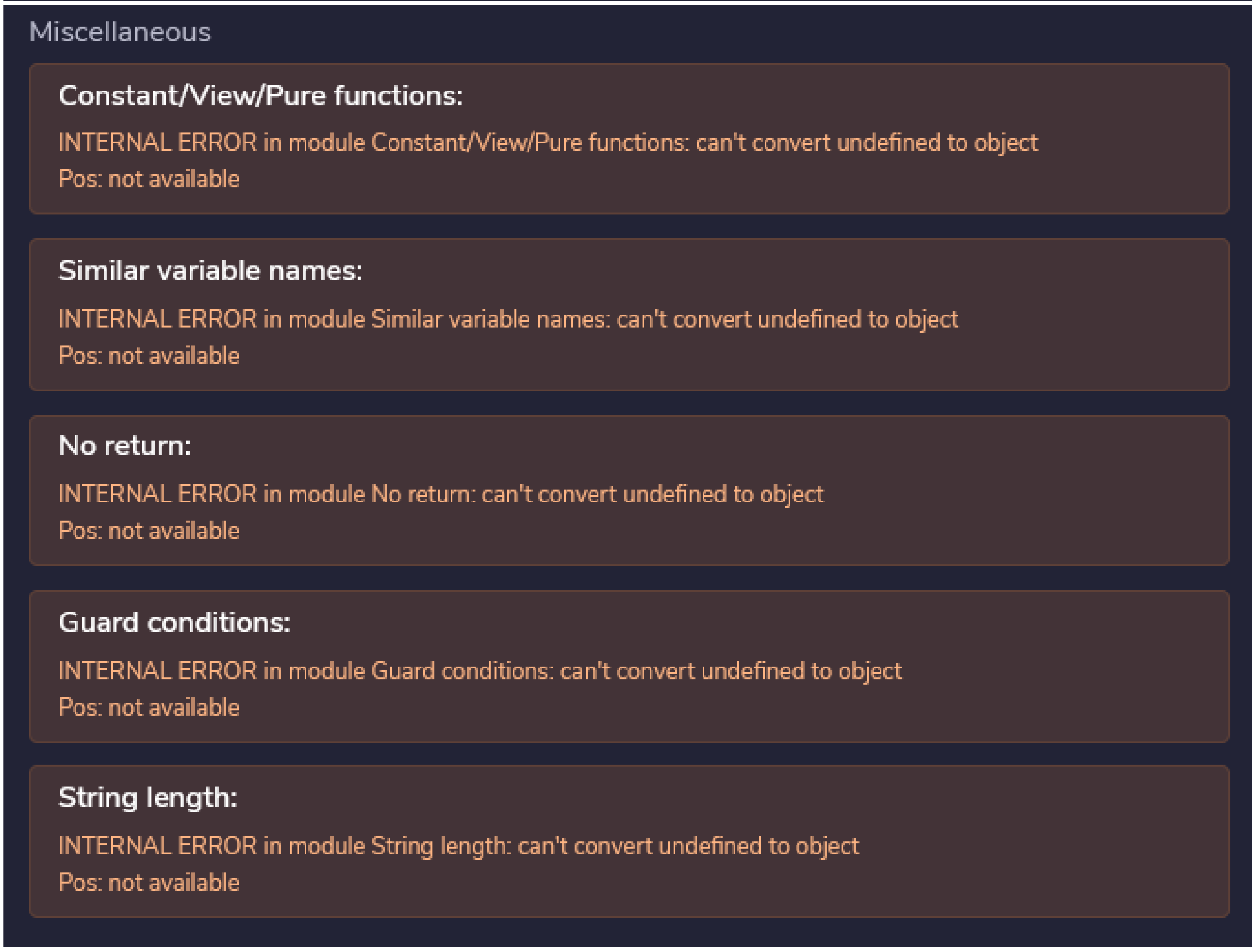
Ether transfer in loop:

INTERNAL ERROR in module Ether transfer in loop: can't convert undefined to object
Pos: not available

ERC

ERC20:

INTERNAL ERROR in module ERC20: can't convert undefined to object
Pos: not available



Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: can't convert undefined to object
Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: can't convert undefined to object
Pos: not available

No return:

INTERNAL ERROR in module No return: can't convert undefined to object
Pos: not available

Guard conditions:

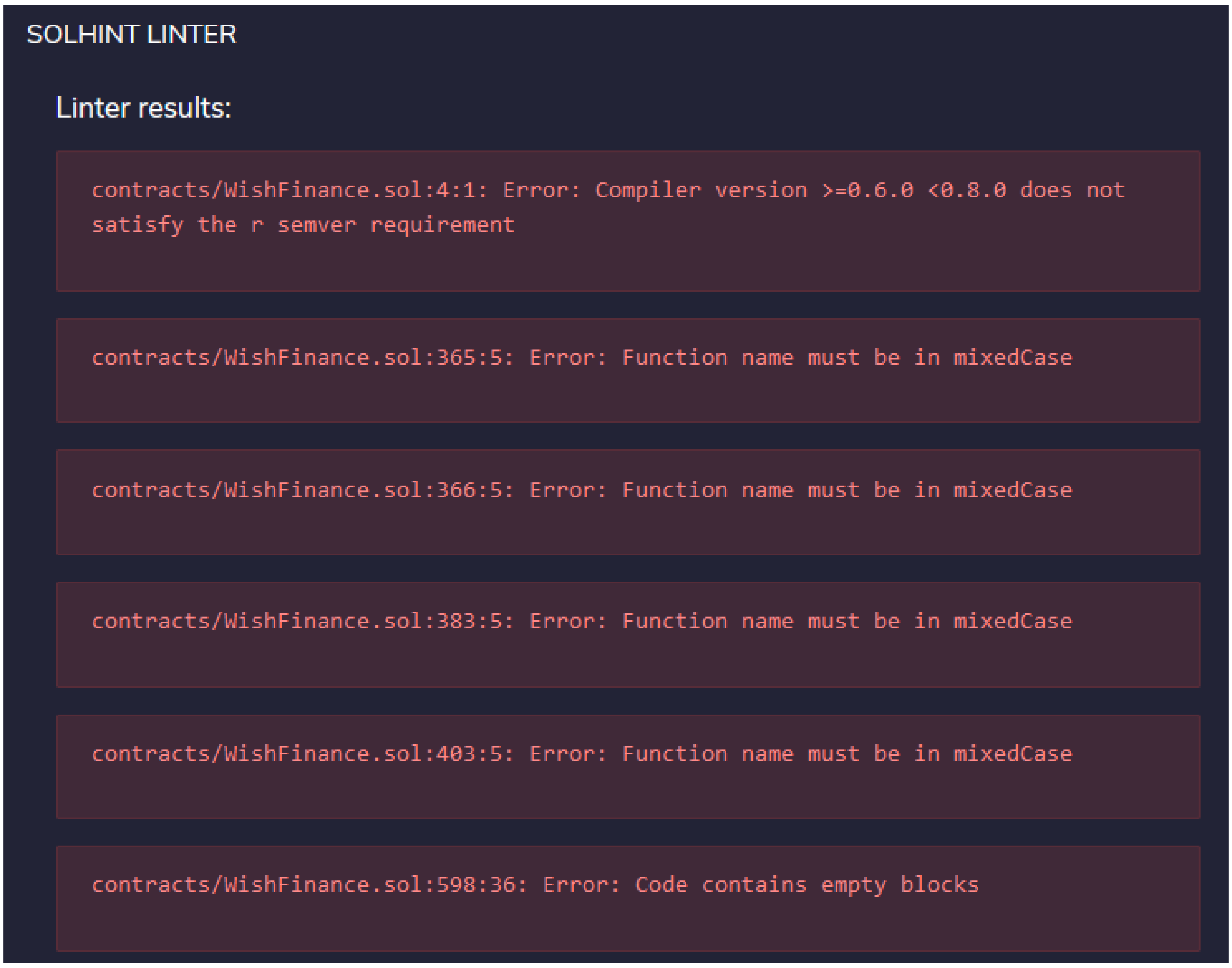
INTERNAL ERROR in module Guard conditions: can't convert undefined to object
Pos: not available

String length:

INTERNAL ERROR in module String length: can't convert undefined to object
Pos: not available

Solhint Linter

WishFinance.sol



SOLHINT LINTER

Linter results:

```
contracts/WishFinance.sol:4:1: Error: Compiler version >=0.6.0 <0.8.0 does not
satisfy the r semver requirement
```

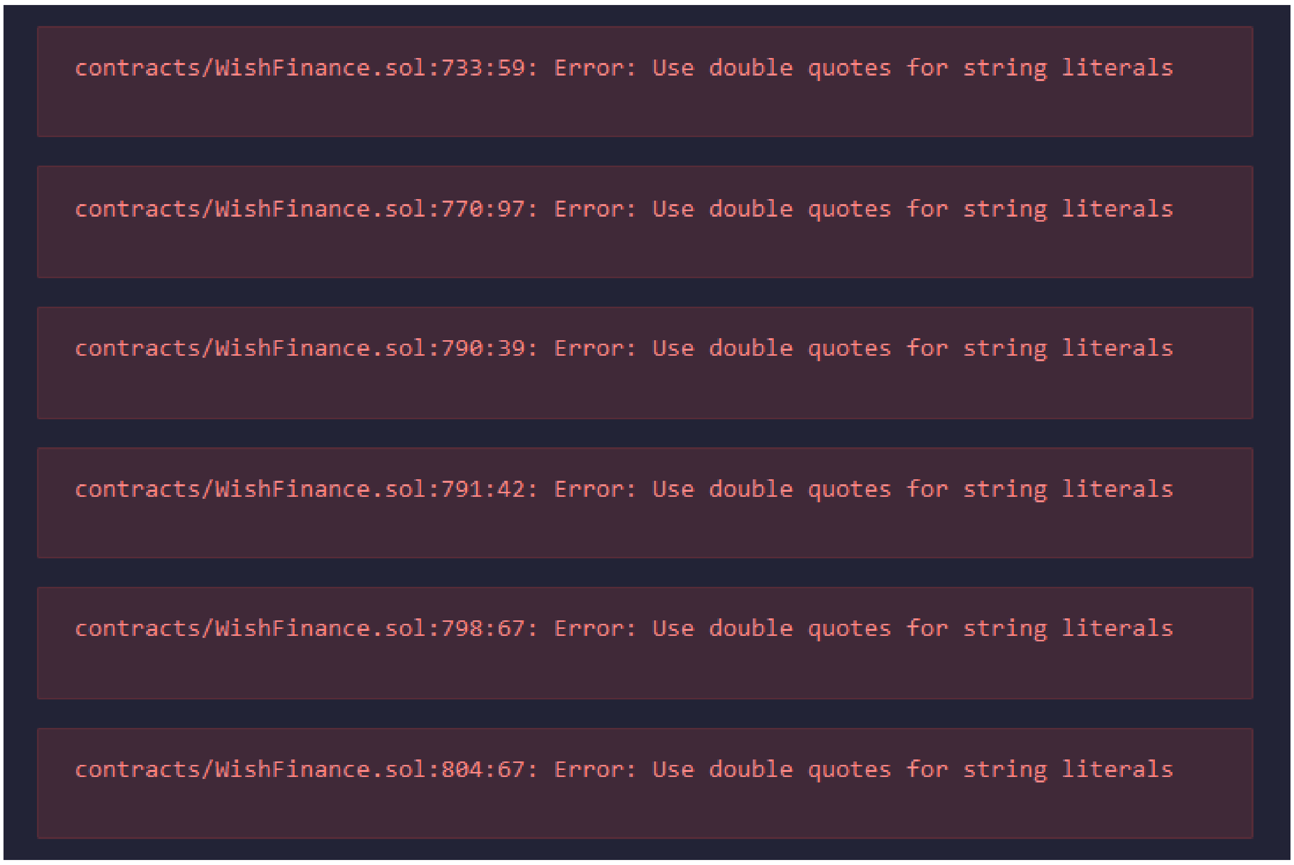
```
contracts/WishFinance.sol:365:5: Error: Function name must be in mixedCase
```

```
contracts/WishFinance.sol:366:5: Error: Function name must be in mixedCase
```

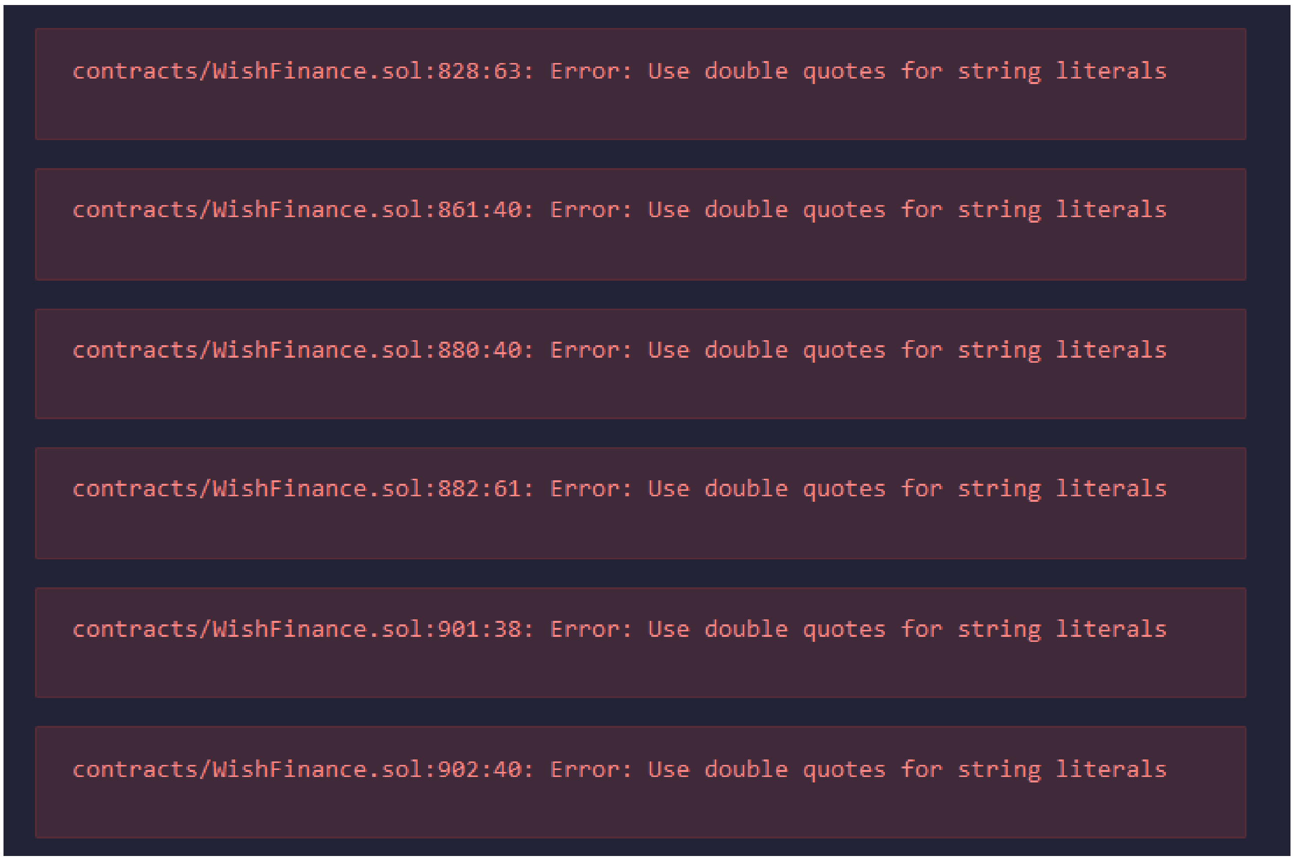
```
contracts/WishFinance.sol:383:5: Error: Function name must be in mixedCase
```

```
contracts/WishFinance.sol:403:5: Error: Function name must be in mixedCase
```

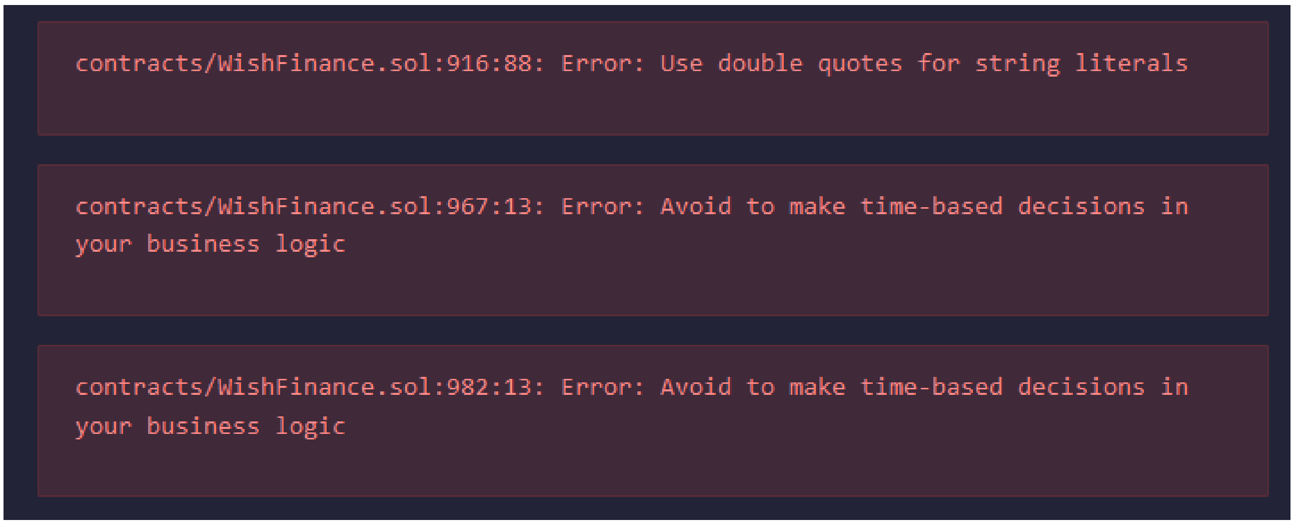
```
contracts/WishFinance.sol:598:36: Error: Code contains empty blocks
```



```
contracts/WishFinance.sol:733:59: Error: Use double quotes for string literals

contracts/WishFinance.sol:770:97: Error: Use double quotes for string literals

contracts/WishFinance.sol:790:39: Error: Use double quotes for string literals

contracts/WishFinance.sol:791:42: Error: Use double quotes for string literals

contracts/WishFinance.sol:798:67: Error: Use double quotes for string literals

contracts/WishFinance.sol:804:67: Error: Use double quotes for string literals
```



```
contracts/WishFinance.sol:828:63: Error: Use double quotes for string literals

contracts/WishFinance.sol:861:40: Error: Use double quotes for string literals

contracts/WishFinance.sol:880:40: Error: Use double quotes for string literals

contracts/WishFinance.sol:882:61: Error: Use double quotes for string literals

contracts/WishFinance.sol:901:38: Error: Use double quotes for string literals

contracts/WishFinance.sol:902:40: Error: Use double quotes for string literals
```



```
contracts/WishFinance.sol:916:88: Error: Use double quotes for string literals

contracts/WishFinance.sol:967:13: Error: Avoid to make time-based decisions in
your business logic

contracts/WishFinance.sol:982:13: Error: Avoid to make time-based decisions in
your business logic
```