EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	PitCoin Token
Website:	https://pitcoin.fun
Platform:	Ethereum
Language:	Solidity
Date:	August 29th, 2023

Table of contents

`

Introduction

EtherAuthority was contracted by the PitCoin team to perform the Security audit of the PitCoin Token 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 29th, 2023.

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

- PitCoin Token smart contract inherits ERC20 and Ownable standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community audited and time tested, and hence are not part of the audit scope.
- The token follows ERC20 standard, which will make it compatible with all the platforms who support ERC20 standard.
- The smart contracts have functions like Auto Liquidity, swapping, withdrawing ether and tokens , etc.

Audit scope

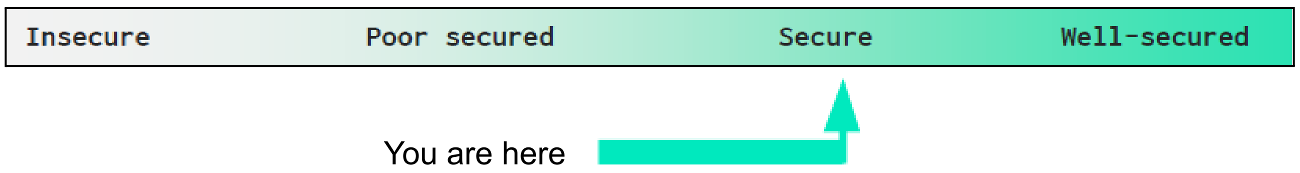
Name	Code Review and Security Analysis Report for PitCoin Token Smart Contract
Platform	**Ethereum / Solidity**
File	PitCoin.sol
File MD5 Hash	9F52C402BB2F79654856DD85526E9022
Audit Date	August 29th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: PitCoin ● Symbol: PITCOIN ● Decimals: 18 ● Total Supply: 21 billion	**YES, This is valid.**
Other Customization: ● 0% tax on buy and sell. ● 95% token sent to LP. ● 5% for exchange listings kept in multi signature wallet Stealth launch. ● 5% tax on transfer, to be swapped for Eth and sent to marketing wallet, for non-whitelisted users.	**YES, This is valid.**
Ownership Control: ● Set the marketing wallet address. ● Set the swap tokens amount value. ● Toggled swapping. ● Set the whitelist address with status. ● Owner can renounce ownership. ● Current owner can transfer the ownership.	**YES, This is valid.**

Audit Summary

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



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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 0 low and 6 very low level issues.

These issues are fixed / acknowledged in the revised contract 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	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Passed
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

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

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

The PitCoin 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.

Code parts are not **well** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a PitCoin Token smart contract code in the form of a file. 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.

Another source of information was its official project URL: https://pitcoin.fun 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.

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

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	setMarketingWallet	external	access only Owner	No Issue
3	setSwapTokensAtAmount	external	access only Owner	No Issue
4	toggleSwapping	external	access only Owner	No Issue
5	setWhitelistStatus	external	access only Owner	No Issue
6	checkWhitelist	external	Passed	No Issue
7	_takeTax	internal	Passed	No Issue
8	_transfer	internal	Passed	No Issue
9	internalSwap	internal	Passed	No Issue
10	swapToETH	internal	Passed	No Issue
11	withdrawStuckETH	external	access only Owner	No Issue
12	withdrawStuckTokens	external	access only Owner	No Issue
13	receive	external	Passed	No Issue
14	name	read	Passed	No Issue
15	symbol	read	Passed	No Issue
16	decimals	read	Passed	No Issue
17	totalSupply	read	Passed	No Issue
18	balanceOf	read	Passed	No Issue
19	transfer	write	Passed	No Issue
20	allowance	read	Passed	No Issue
21	approve	write	Passed	No Issue
22	transferFrom	write	Passed	No Issue
23	increaseAllowance	write	Passed	No Issue
24	decreaseAllowance	write	Passed	No Issue
25	_transfer	internal	Passed	No Issue
26	_update	internal	Passed	No Issue
27	_mint	internal	Passed	No Issue
28	_burn	internal	Passed	No Issue
29	_approve	internal	Passed	No Issue
30	_approve	internal	Passed	No Issue
31	_spendAllowance	internal	Passed	No Issue
32	onlyOwner	modifier	Passed	No Issue
33	owner	read	Passed	No Issue
34	_checkOwner	internal	Passed	No Issue
35	renounceOwnership	write	access only Owner	No Issue
36	transferOwnership	write	access only Owner	No Issue
37	_transferOwnership	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 Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

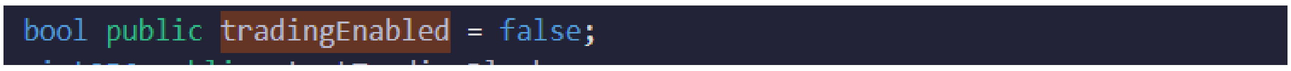
No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Unused Variable:



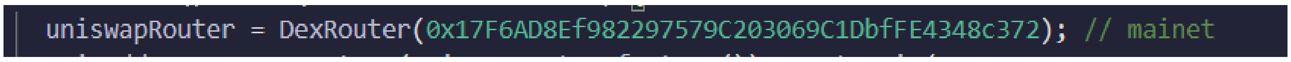
```
bool public tradingEnabled = false;
```

The tradingEnabled variable is not used in contract.

Resolution: We suggest removing unused variables.

Status: Fixed

(2) Hard Coded variable values:



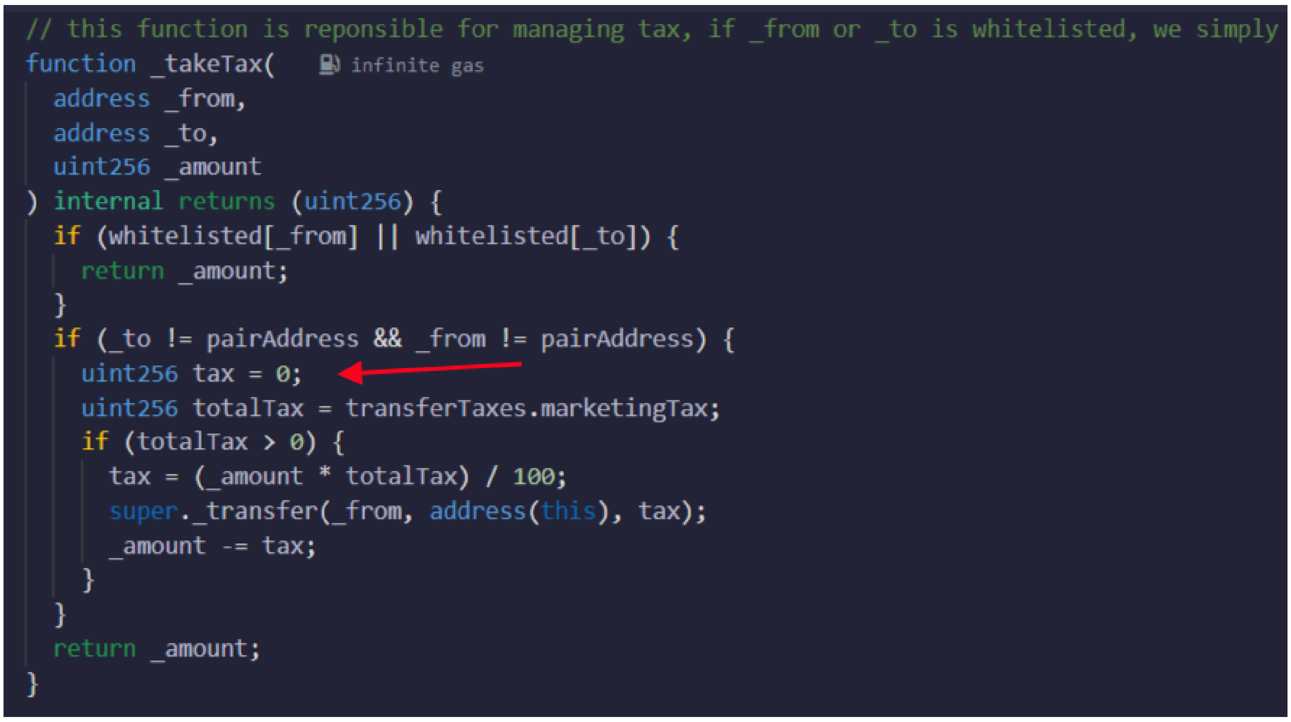
```
uniswapRouter = DexRouter(0x17F6AD8Ef982297579C203069C1DbfFE4348c372); // mainet
```

The uniswapRouter is set with hardcoded values.

Resolution: We suggest always ensuring the set hardcoded values.

Status: Fixed

(3) Variable initialization:



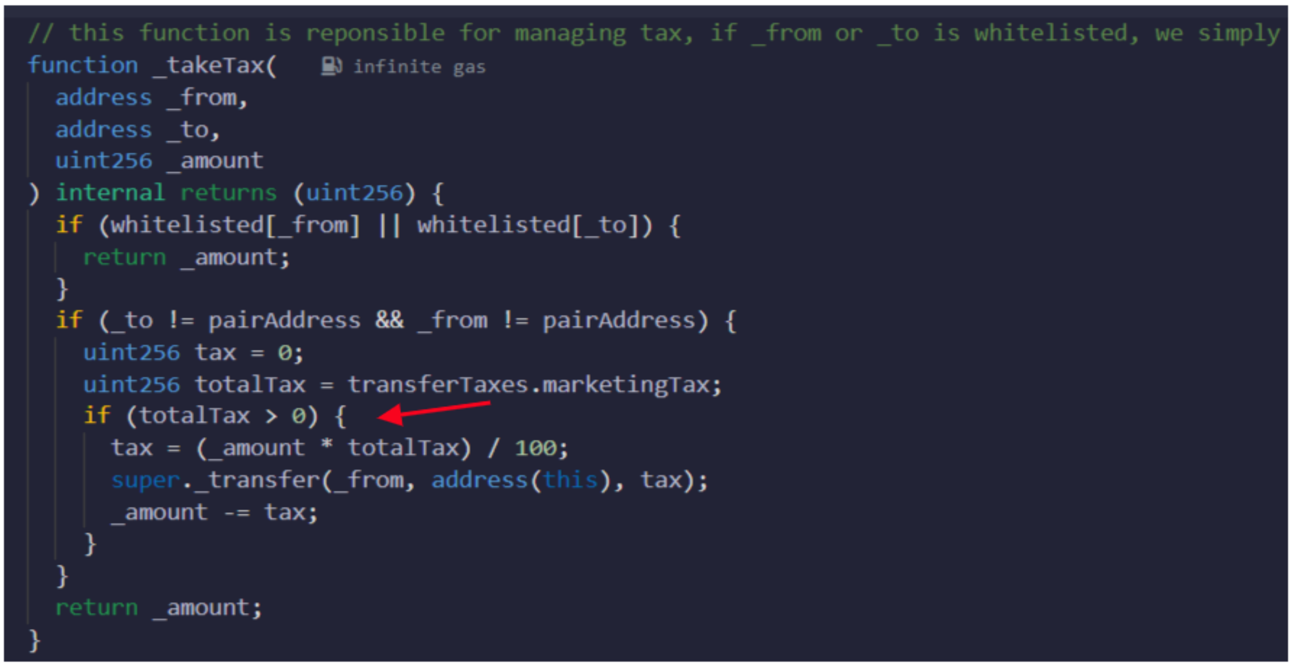
```
// this function is reponsible for managing tax, if _from or _to is whitelisted, we simply
function _takeTax(    infinite gas
  address _from,
  address _to,
  uint256 _amount
) internal returns (uint256) {
  if (whitelisted[_from] || whitelisted[_to]) {
    return _amount;
  }
  if (_to != pairAddress && _from != pairAddress) {
    uint256 tax = 0;
    uint256 totalTax = transferTaxes.marketingTax;
    if (totalTax > 0) {
      tax = (_amount * totalTax) / 100;
      super._transfer(_from, address(this), tax);
      _amount -= tax;
    }
  }
  return _amount;
}
```

In the_takeTax function, a tax variable is declared and initialized with zero value uint256 tax = 0; Here, uint256 considers variable default value =0.

Resolution: We suggest removing tax variable value initialization with zero to save gas fee.

Status: Fixed

(4) Other Programming Issue:



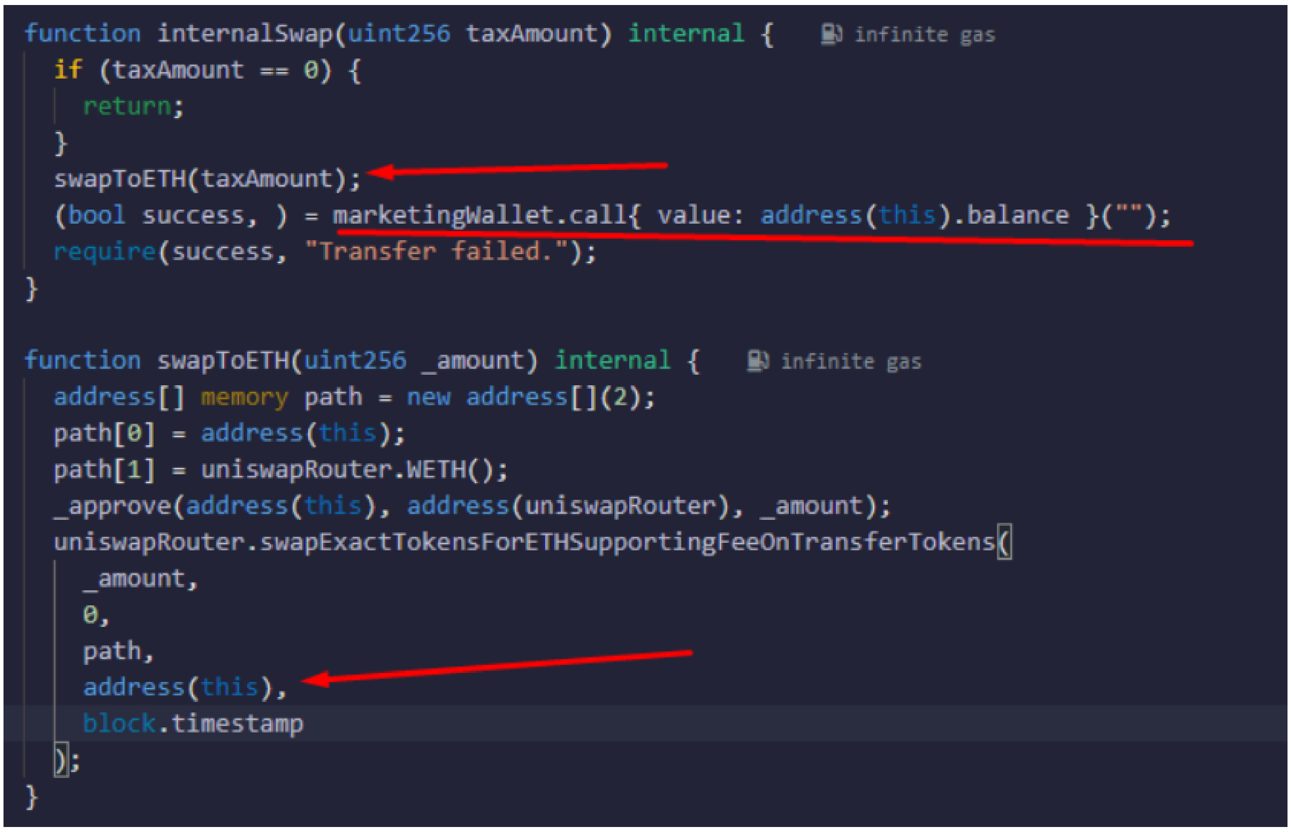
```
// this function is reponsible for managing tax, if _from or _to is whitelisted, we simply
function _takeTax(    infinite gas
    address _from,
    address _to,
    uint256 _amount
) internal returns (uint256) {
    if (whitelisted[_from] || whitelisted[_to]) {
        return _amount;
    }
    if (_to != pairAddress && _from != pairAddress) {
        uint256 tax = 0;
        uint256 totalTax = transferTaxes.marketingTax;
        if (totalTax > 0) {
            tax = (_amount * totalTax) / 100;
            super._transfer(_from, address(this), tax);
            _amount -= tax;
        }
    }
    return _amount;
}
```

The marketingTax is set to 5% and further not updated anywhere in the contract. In _takeTax function no need to declare totalTax variable as well validation to check totalTax is greater than 0.

Resolution: We suggest removing the totalTax variable and use marketingTax and also validation to check totalTax is greater than 0 to save some gas.

Status: Fixed

(5) Eth can be transferred directly to marketing wallet:



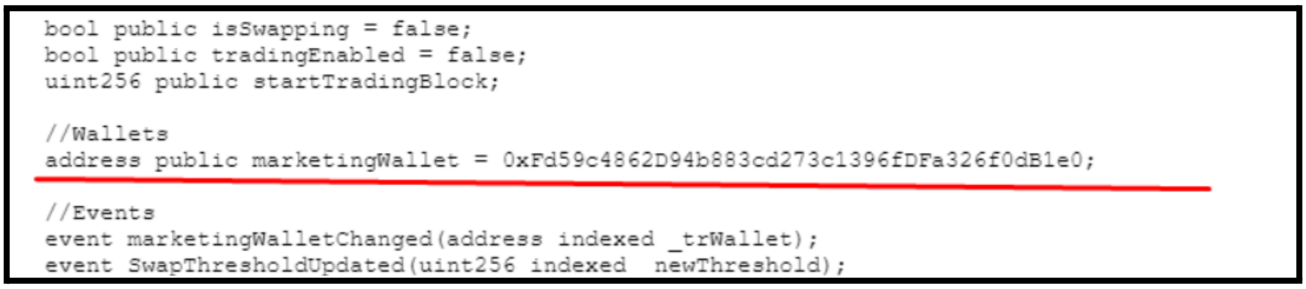
```
function internalSwap(uint256 taxAmount) internal {    infinite gas
    if (taxAmount == 0) {
        return;
    }
    swapToETH(taxAmount);
    (bool success, ) = marketingWallet.call{ value: address(this).balance }("");
    require(success, "Transfer failed.");
}

function swapToETH(uint256 _amount) internal {    infinite gas
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapRouter.WETH();
    _approve(address(this), address(uniswapRouter), _amount);
    uniswapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
        _amount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

In the internalSwap function after swapToETH all contract balance is transferred to the marketing wallet, instead we can directly set “to address” to the marketing wallet in swapExactTokensForETHSupportingFeeOnTransferTokens function. By this only swapped tokens’ coins will be sent to the marketing wallet.

Resolution: We suggest removing redundant code to save the gas.
Remove this line (bool success,) = marketingWallet.call{ value: address(this).balance }(""); and set “to address” to the marketing wallet in the swap function. This will save the gas.

Status: Fixed

(6) Wrong marketing wallet address:



```
bool public isSwapping = false;
bool public tradingEnabled = false;
uint256 public startTradingBlock;

//Wallets
address public marketingWallet = 0xFd59c4862D94b883cd273c1396fDFa326f0dB1e0;

//Events
event marketingWalletChanged(address indexed _trWallet);
event SwapThresholdUpdated(uint256 indexed  newThreshold);
```

The marketing wallet address set in the contract is different from the requirements given. Contract requirements has this marketing address: 0xd1394c663AaF3CD87136fa046135967e0a8501B7. The owner can update the marketing wallet.

Resolution: We advise to set the correct marketing wallet address. The Owner can update this using setMarketingWallet, but need to confirm before coins get transferred by swapping.

Status: Fixed

Centralization

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

PitCoin.sol

- setMarketingWallet: Marketing Wallet address can be set by the owner.
- setSwapTokensAtAmount: Swap tokens amount value can be set by the owner.
- toggleSwapping: Swapping status can be toggled by the owner.
- setWhitelistStatus: Whitelist address with status can be set by the owner.
- withdrawStuckETH: Owner can withdraw ether value.
- withdrawStuckTokens: Owner can withdraw token value.

Ownable.sol

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

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

Conclusion

We were given a contract code in the form of a file. And we have used all possible tests based on given objects as files. We had observed 6 informational issues in the smart contracts. And those issues are fixed in the revised contract code. So, **it's good to go for the 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 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 smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of 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).

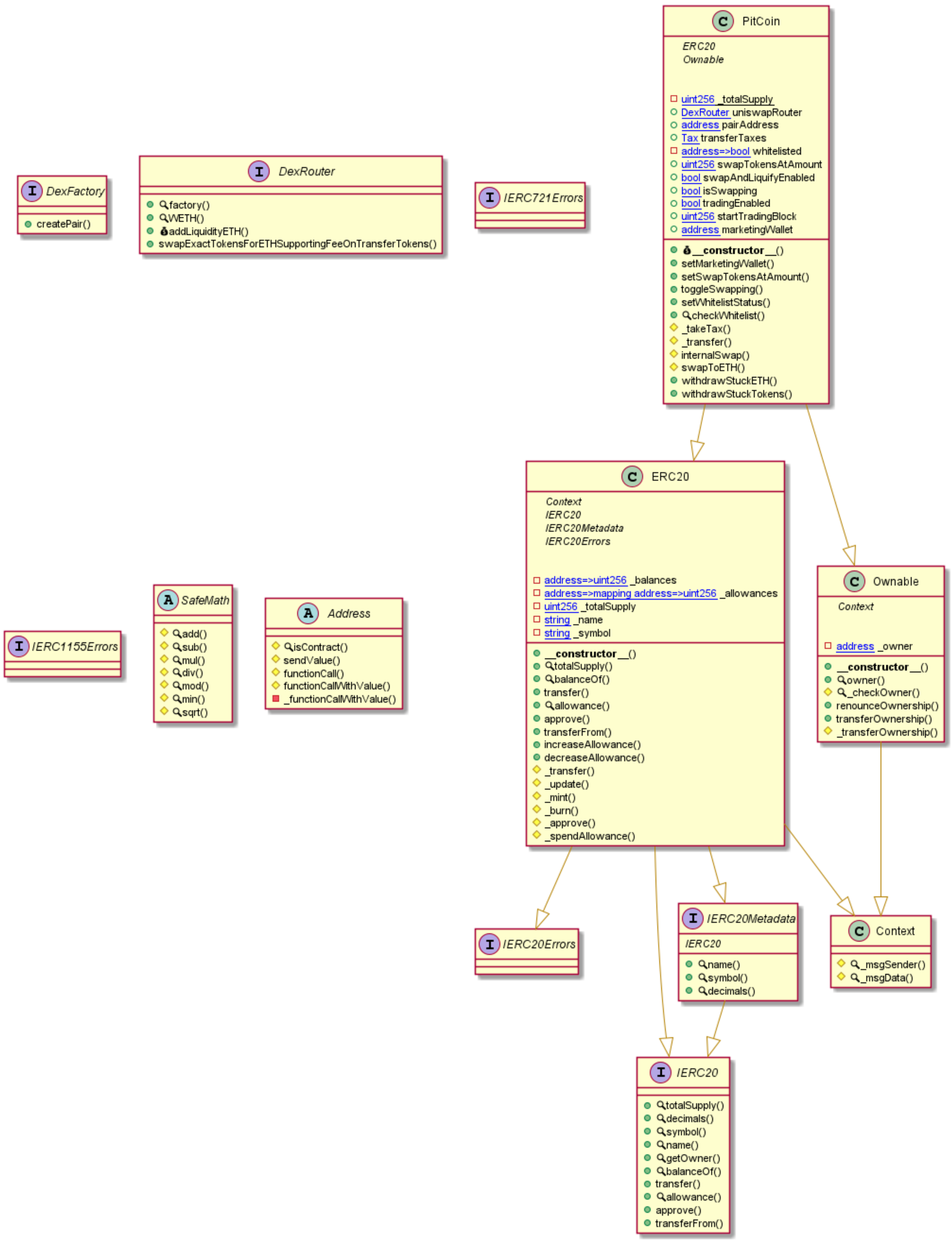
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

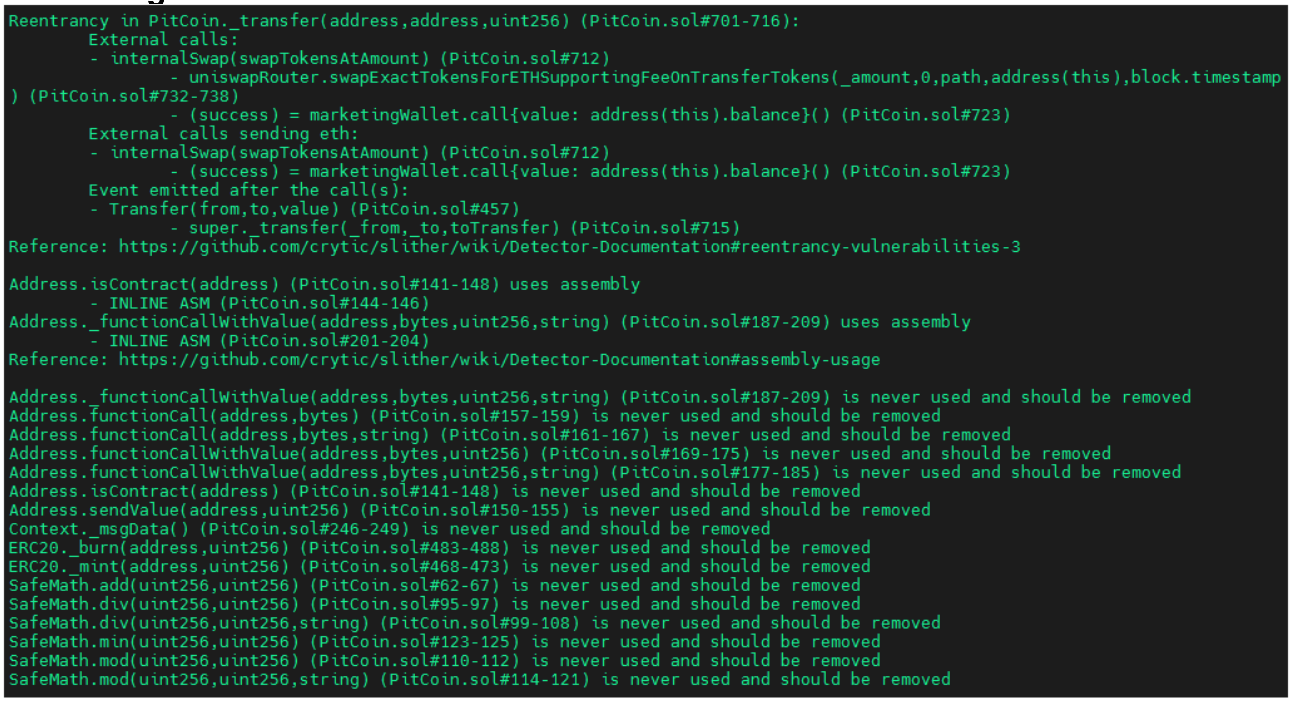


Slither Results Log

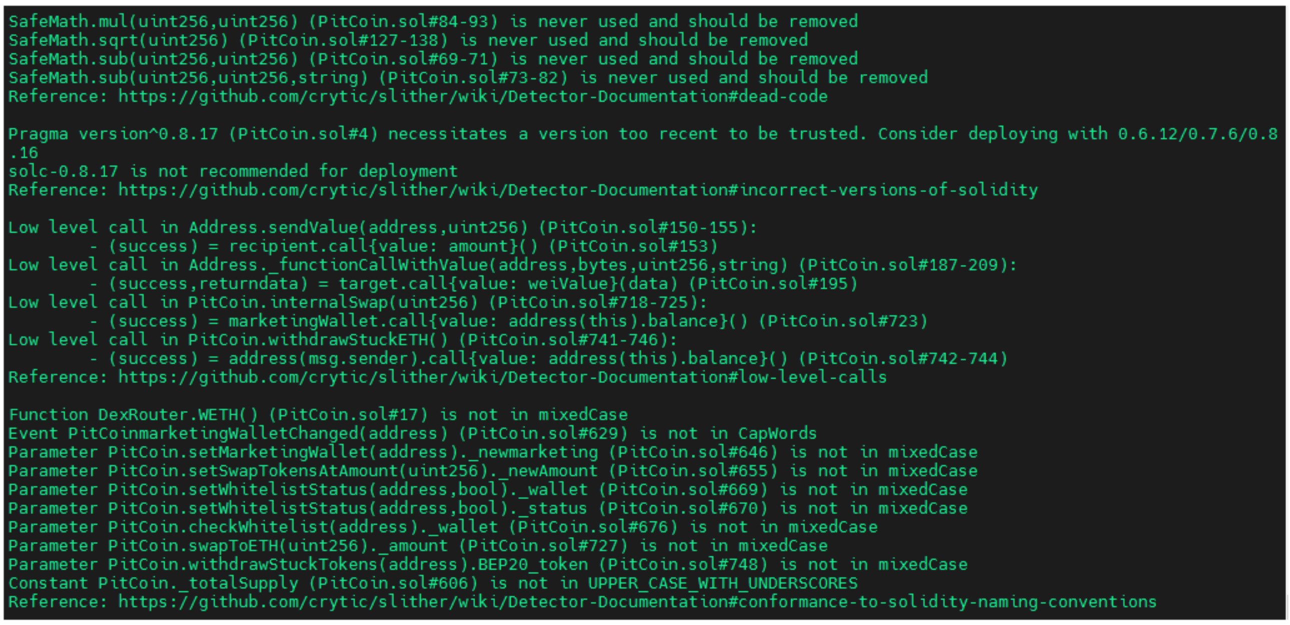
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

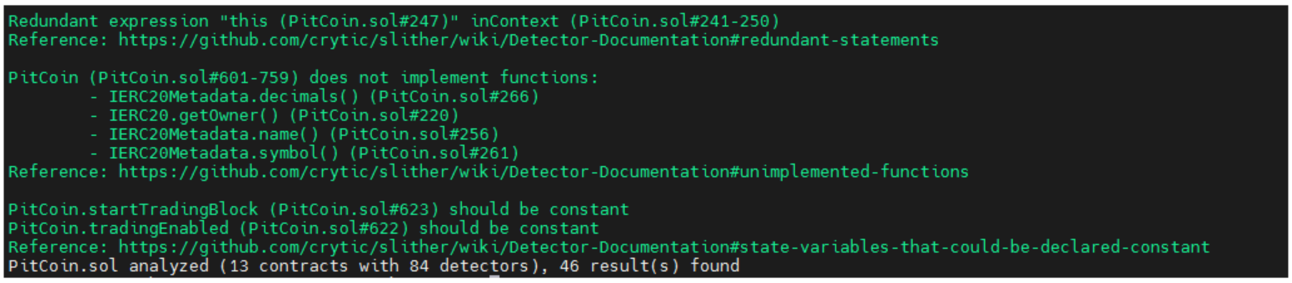
Slither Log >> PitCoin.sol



```
Reentrancy in PitCoin._transfer(address,address,uint256) (PitCoin.sol#701-716):
        External calls:
        - internalSwap(swapTokensAtAmount) (PitCoin.sol#712)
                - uniswapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(_amount,0,path,address(this),block.timestamp
) (PitCoin.sol#732-738)
                - (success) = marketingWallet.call{value: address(this).balance}() (PitCoin.sol#723)
        External calls sending eth:
        - internalSwap(swapTokensAtAmount) (PitCoin.sol#712)
                - (success) = marketingWallet.call{value: address(this).balance}() (PitCoin.sol#723)
        Event emitted after the call(s):
        - Transfer(from,to,value) (PitCoin.sol#457)
                - super._transfer(_from,_to,toTransfer) (PitCoin.sol#715)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Address.isContract(address) (PitCoin.sol#141-148) uses assembly
        - INLINE ASM (PitCoin.sol#144-146)
Address._functionCallWithValue(address,bytes,uint256,string) (PitCoin.sol#187-209) uses assembly
        - INLINE ASM (PitCoin.sol#201-204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address._functionCallWithValue(address,bytes,uint256,string) (PitCoin.sol#187-209) is never used and should be removed
Address.functionCall(address,bytes) (PitCoin.sol#157-159) is never used and should be removed
Address.functionCall(address,bytes,string) (PitCoin.sol#161-167) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (PitCoin.sol#169-175) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (PitCoin.sol#177-185) is never used and should be removed
Address.isContract(address) (PitCoin.sol#141-148) is never used and should be removed
Address.sendValue(address,uint256) (PitCoin.sol#150-155) is never used and should be removed
Context._msgData() (PitCoin.sol#246-249) is never used and should be removed
ERC20._burn(address,uint256) (PitCoin.sol#483-488) is never used and should be removed
ERC20._mint(address,uint256) (PitCoin.sol#468-473) is never used and should be removed
SafeMath.add(uint256,uint256) (PitCoin.sol#62-67) is never used and should be removed
SafeMath.div(uint256,uint256) (PitCoin.sol#95-97) is never used and should be removed
SafeMath.div(uint256,uint256,string) (PitCoin.sol#99-108) is never used and should be removed
SafeMath.min(uint256,uint256) (PitCoin.sol#123-125) is never used and should be removed
SafeMath.mod(uint256,uint256) (PitCoin.sol#110-112) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (PitCoin.sol#114-121) is never used and should be removed
```



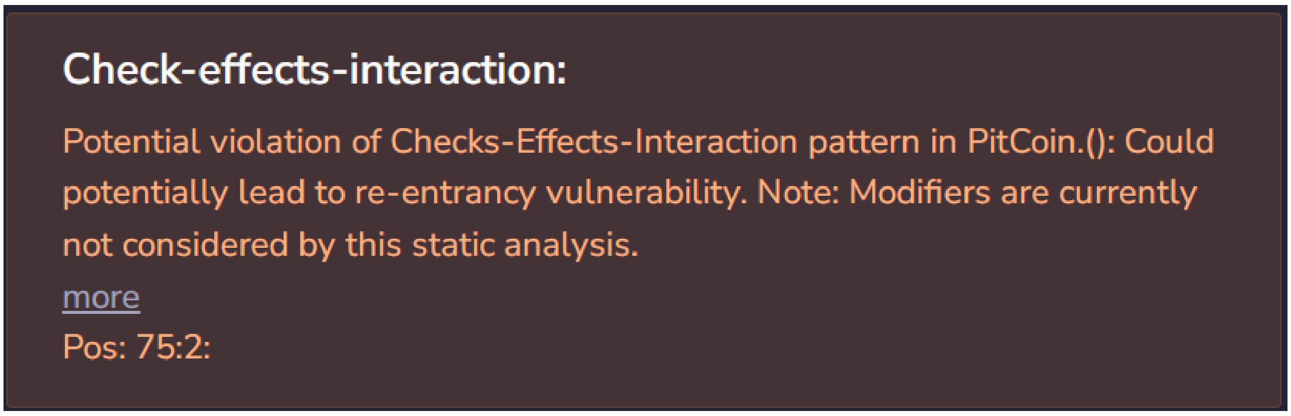
```
SafeMath.mul(uint256,uint256) (PitCoin.sol#84-93) is never used and should be removed
SafeMath.sqrt(uint256) (PitCoin.sol#127-138) is never used and should be removed
SafeMath.sub(uint256,uint256) (PitCoin.sol#69-71) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (PitCoin.sol#73-82) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.17 (PitCoin.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8
.16
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (PitCoin.sol#150-155):
        - (success) = recipient.call{value: amount}() (PitCoin.sol#153)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (PitCoin.sol#187-209):
        - (success,returndata) = target.call{value: weiValue}(data) (PitCoin.sol#195)
Low level call in PitCoin.internalSwap(uint256) (PitCoin.sol#718-725):
        - (success) = marketingWallet.call{value: address(this).balance}() (PitCoin.sol#723)
Low level call in PitCoin.withdrawStuckETH() (PitCoin.sol#741-746):
        - (success) = address(msg.sender).call{value: address(this).balance}() (PitCoin.sol#742-744)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function DexRouter.WETH() (PitCoin.sol#17) is not in mixedCase
Event PitCoinmarketingWalletChanged(address) (PitCoin.sol#629) is not in CapWords
Parameter PitCoin.setMarketingWallet(address)._newmarketing (PitCoin.sol#646) is not in mixedCase
Parameter PitCoin.setSwapTokensAtAmount(uint256)._newAmount (PitCoin.sol#655) is not in mixedCase
Parameter PitCoin.setWhitelistStatus(address,bool)._wallet (PitCoin.sol#669) is not in mixedCase
Parameter PitCoin.setWhitelistStatus(address,bool)._status (PitCoin.sol#670) is not in mixedCase
Parameter PitCoin.checkWhitelist(address)._wallet (PitCoin.sol#676) is not in mixedCase
Parameter PitCoin.swapToETH(uint256)._amount (PitCoin.sol#727) is not in mixedCase
Parameter PitCoin.withdrawStuckTokens(address).BEP20_token (PitCoin.sol#748) is not in mixedCase
Constant PitCoin._totalSupply (PitCoin.sol#606) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```



```
Redundant expression "this (PitCoin.sol#247)" inContext (PitCoin.sol#241-250)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

PitCoin (PitCoin.sol#601-759) does not implement functions:
        - IERC20Metadata.decimals() (PitCoin.sol#266)
        - IERC20.getOwner() (PitCoin.sol#220)
        - IERC20Metadata.name() (PitCoin.sol#256)
        - IERC20Metadata.symbol() (PitCoin.sol#261)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

PitCoin.startTradingBlock (PitCoin.sol#623) should be constant
PitCoin.tradingEnabled (PitCoin.sol#622) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
PitCoin.sol analyzed (13 contracts with 84 detectors), 46 result(s) found
```

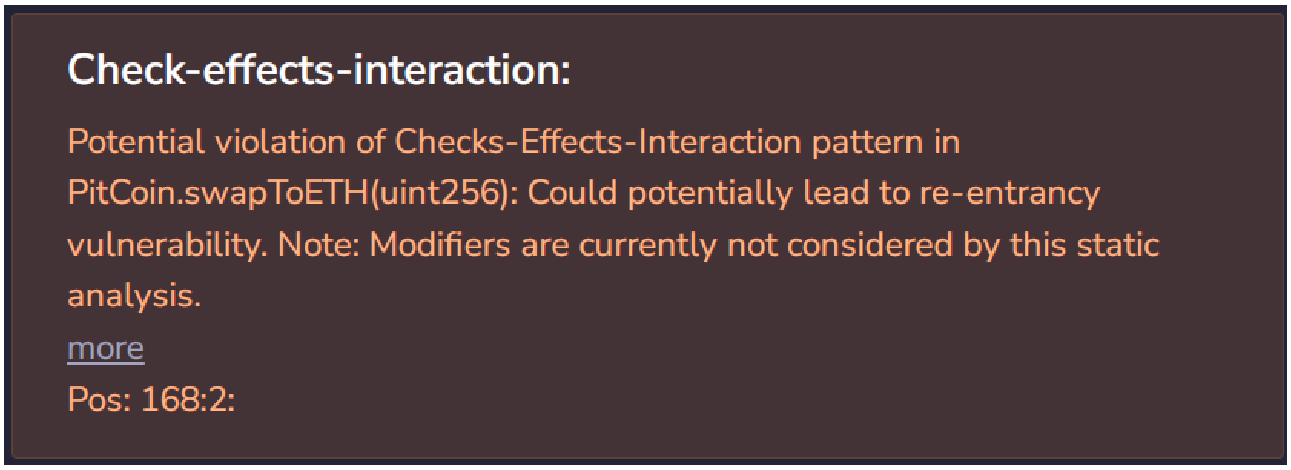
Solidity Static Analysis

PitCoin.sol



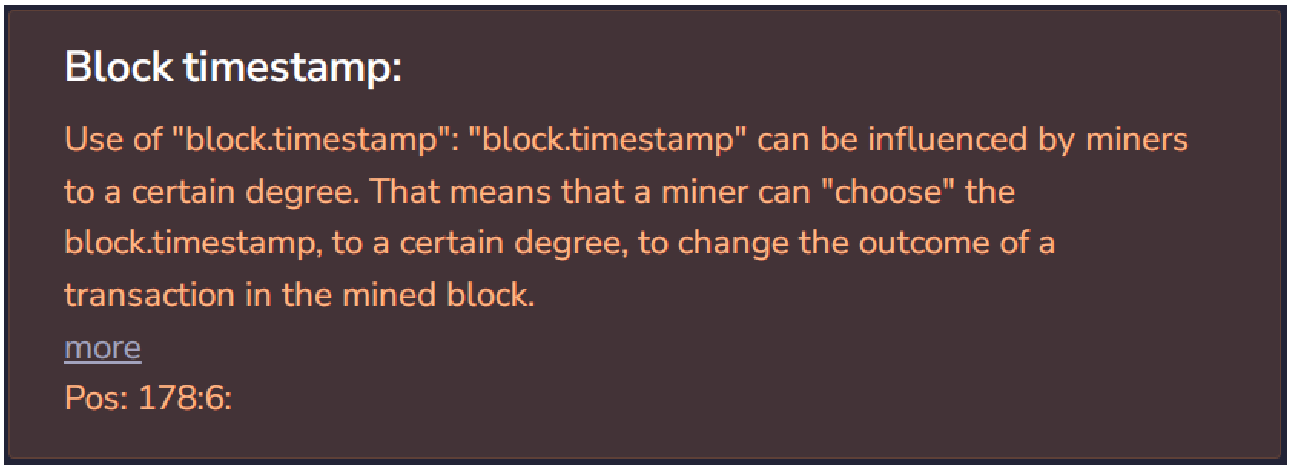
Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in PitCoin.(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 75:2:



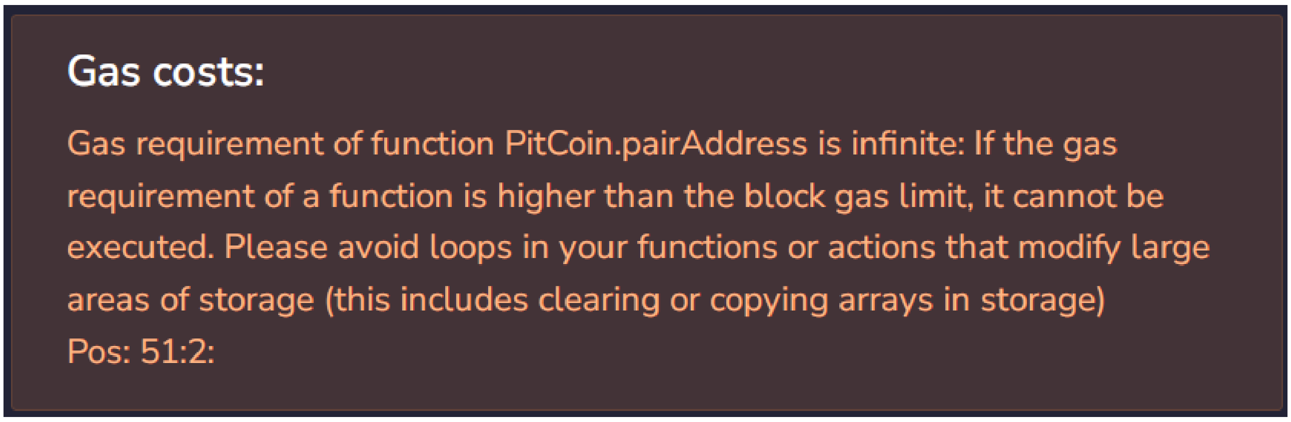
Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in PitCoin.swapToETH(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 168:2:



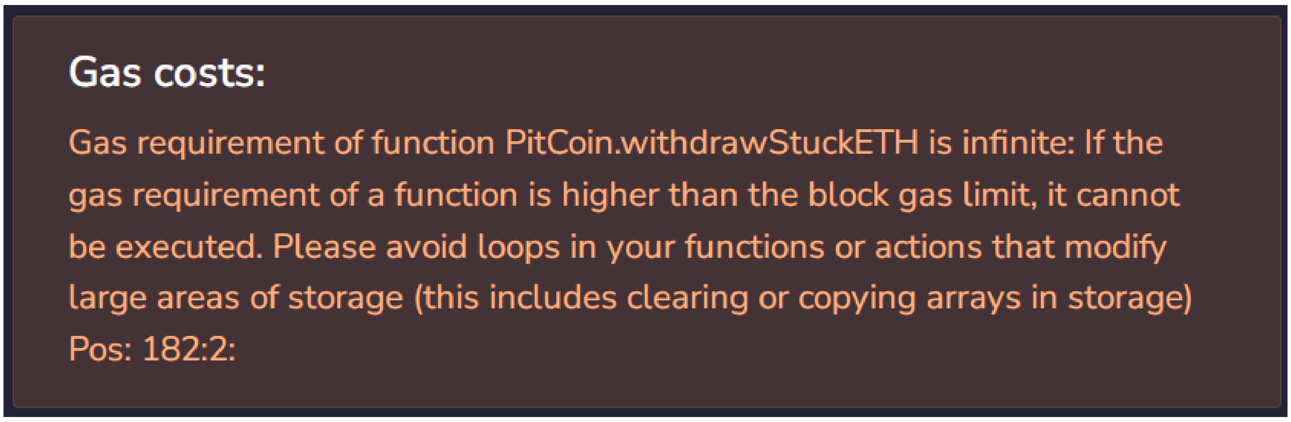
Block timestamp:

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: 178:6:



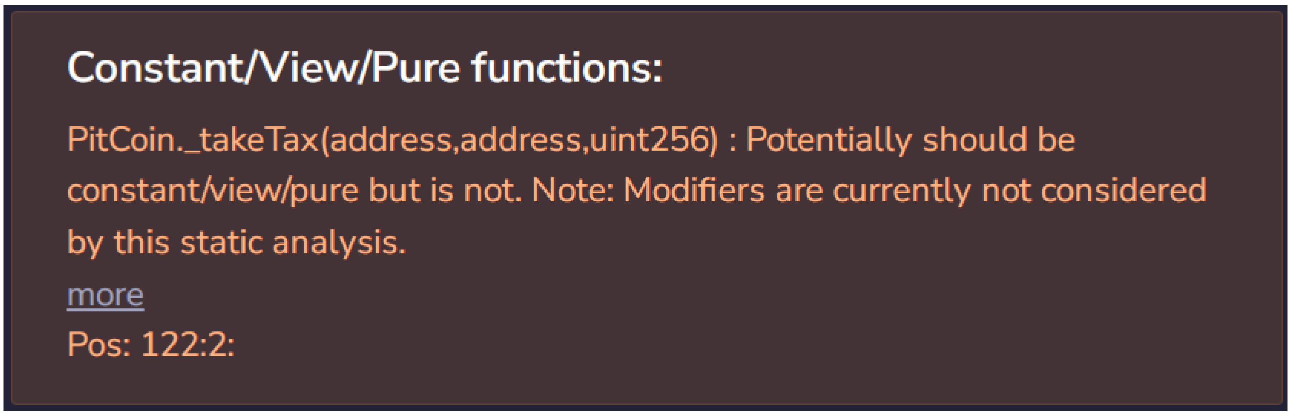
Gas costs:

Gas requirement of function PitCoin.pairAddress 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: 51:2:



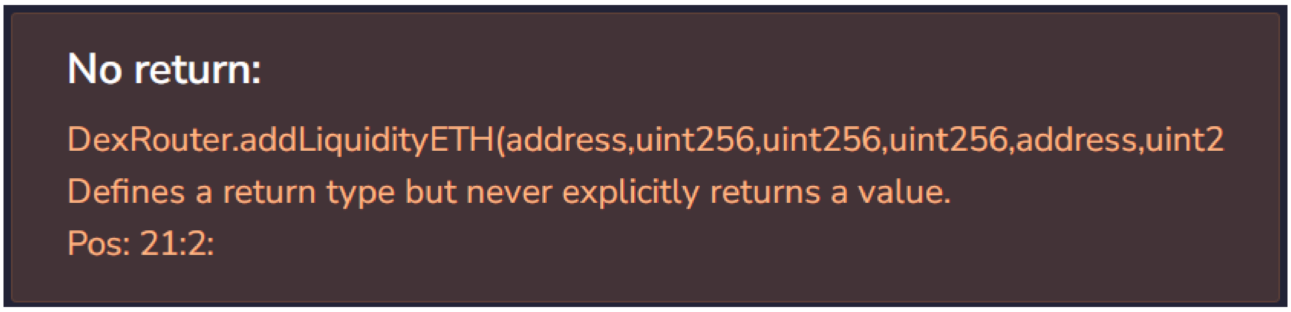
Gas costs:

Gas requirement of function PitCoin.withdrawStuckETH 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: 182:2:



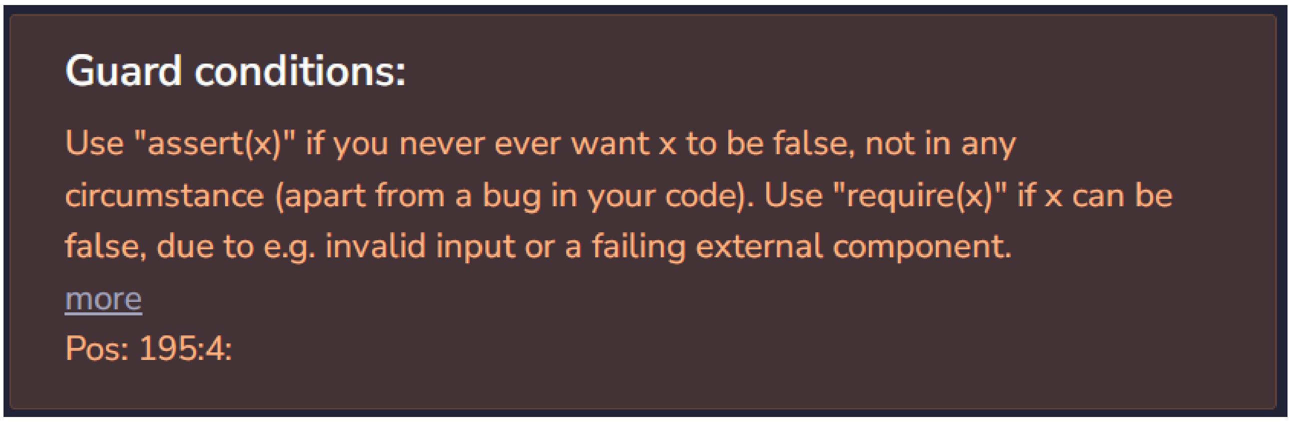
Constant/View/Pure functions:

PitCoin._takeTax(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 122:2:



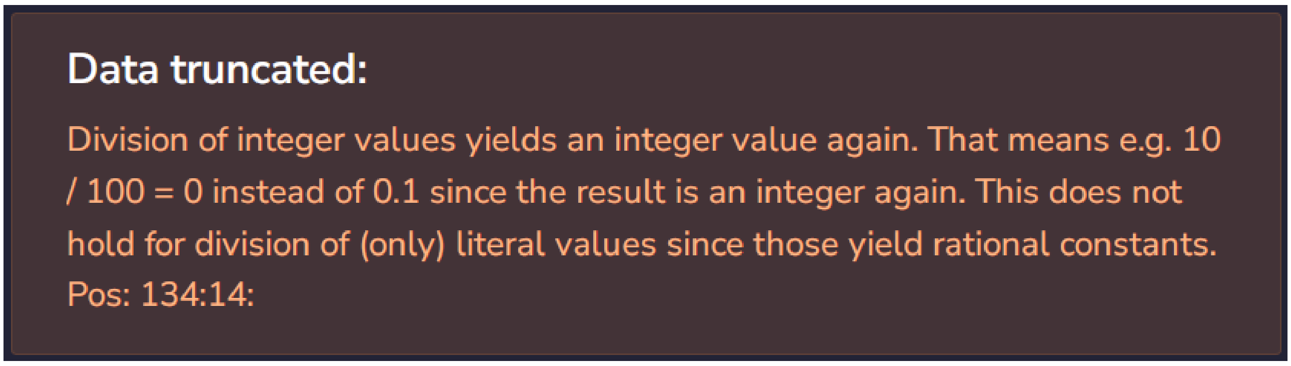
No return:

DexRouter.addLiquidityETH(address,uint256,uint256,uint256,address,uint2
Defines a return type but never explicitly returns a value.
Pos: 21:2:



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: 195:4:

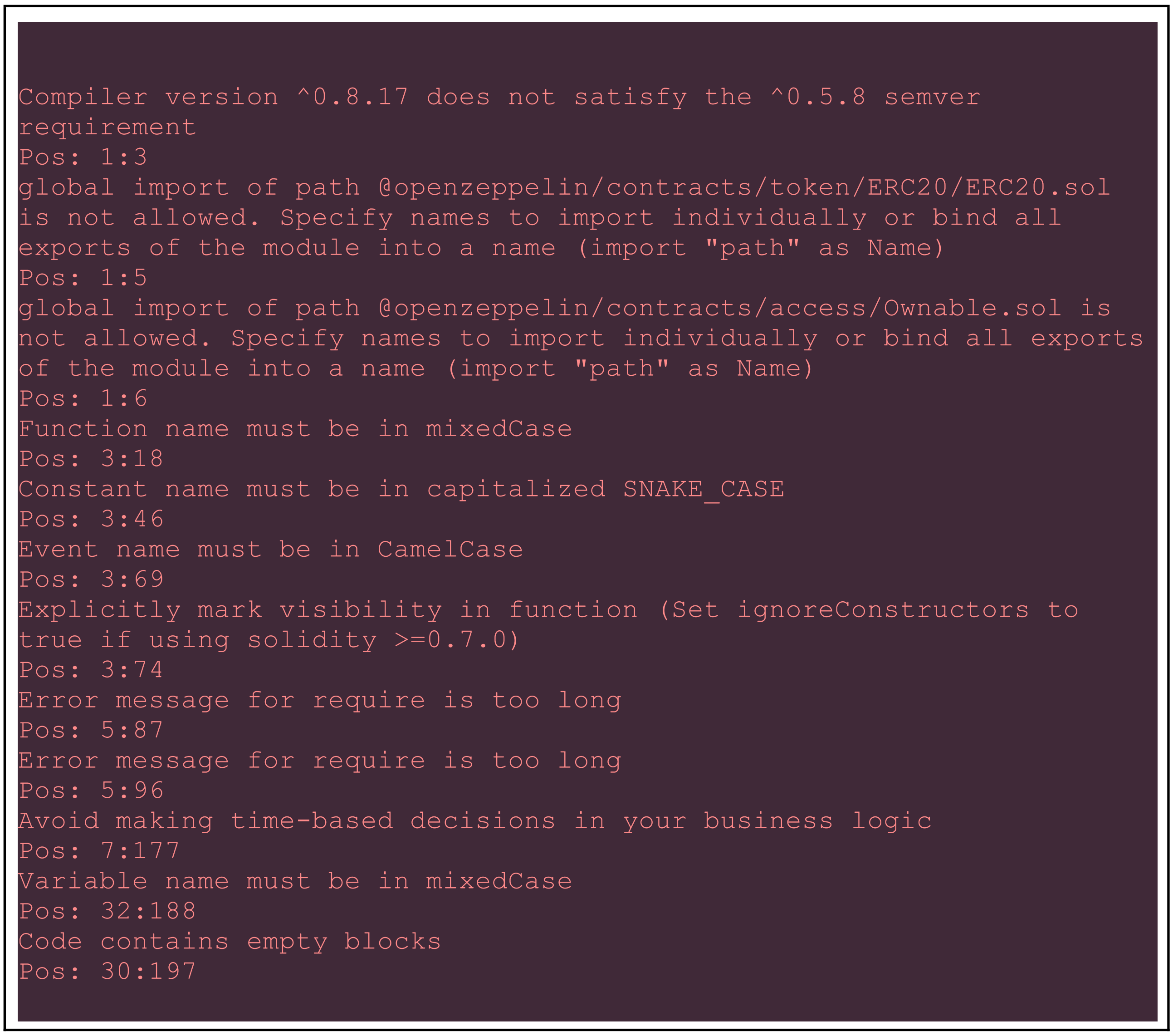


Data truncated:

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: 134:14:

Solhint Linter

PitCoin.sol



```
Compiler version ^0.8.17 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:3
global import of path @openzeppelin/contracts/token/ERC20/ERC20.sol
is not allowed. Specify names to import individually or bind all
exports of the module into a name (import "path" as Name)
Pos: 1:5
global import of path @openzeppelin/contracts/access/Ownable.sol is
not allowed. Specify names to import individually or bind all exports
of the module into a name (import "path" as Name)
Pos: 1:6
Function name must be in mixedCase
Pos: 3:18
Constant name must be in capitalized SNAKE_CASE
Pos: 3:46
Event name must be in CamelCase
Pos: 3:69
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 3:74
Error message for require is too long
Pos: 5:87
Error message for require is too long
Pos: 5:96
Avoid making time-based decisions in your business logic
Pos: 7:177
Variable name must be in mixedCase
Pos: 32:188
Code contains empty blocks
Pos: 30:197
```

Software analysis result:

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

EtherAuthority