Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	InfinityBee (IFB) Token
Website:	http://infinitybee.io
Platform:	Ethereum
Language:	Solidity
Date:	June 24th, 2023

Table of contents

Introduction

EtherAuthority was contracted by InfinityBee Token to perform the Security audit of the InfinityBee (IFB) Token smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 24th, 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

- The InfinityBee Token contracts handle multiple contracts, and all contracts have different functions.
    - INFINITYBEE: This contract is used to set transaction limits, wallet limits and toggle limits.
    - ICO: This contract is used for withdrawal tokens and Fiat.
- IFB Token is a smart contract that has functions like withdrawFiat, withdrawTokens, transferOwnership, claim, buyTokens, etc.

Audit scope

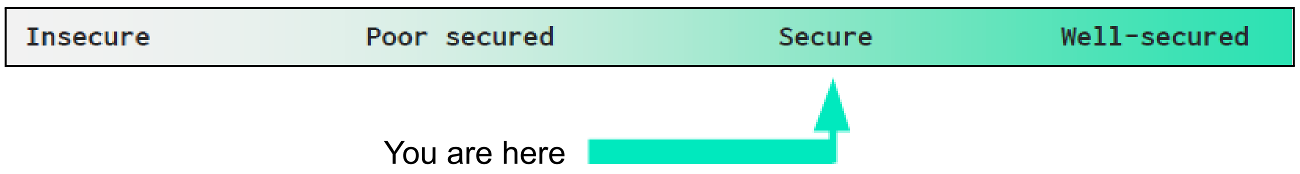
Name	**Code Review and Security Analysis Report for InfinityBee (IFB) Token Smart Contracts**
Platform	**Ethereum / Solidity**
File 1	INFINITYBEE.sol
File 1 Initial code link	0x67c81830e498428b03867D6Bd10Ad1749FbF9E99
File 2	ICO.sol
File 2 Initial code link	0x2e9B3F5F192b1e8666874Eea421d987b973055Ac
Audit Date	June 24th, 2023
Revised Audit Date	June 29th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 INFINITYBEE.sol ● Name: InfinityBee ● Symbol: IFB ● Decimals: 18 ● Maximum Transfers Per Day: 300 times ● Total Supply: 1800 Million ● Transaction Limit: 7 Million ● Wallet Limit: 45 Million **Owner has control over following functions:** ● Set the transaction limit. ● Set the wallet limit. ● Set the maximum transfer per day value. ● Set the toggle limit.	**YES, This is valid.**
File 2 ICO.sol ● Token Price: 0.008 ether ● Vesting Percentage: 50% ● Vesting periods: 18 months **Owner has control over following functions:** ● Set the next round time. ● Set the new vesting period time and percentage value. ● Set the new token time. ● Withdraw tokens. ● Withdraw fiat values. ● Current owner can transfer ownership of the contract to a new account.	**YES, This is valid.**

Audit Summary

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



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

We found 0 critical, 0 high, 1 medium and 2 low and 2 very low level issues.

We confirm that these issues have been fixed / acknowledged in the revised code.

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

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	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 2 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the InfinityBee 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 InfinityBee Token.

The IFB Token team has not provided 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 smart contracts.

Documentation

We were given an InfinityBee Token smart contract code in the form of a https://sepolia.etherscan.io web link. The hash of that code is mentioned above in the table.

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

Another source of information was its official website: http://infinitybee.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 contracts infrastructure that are based on well known industry standard open source projects.

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

AS-IS overview

INFINITYBEE.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOwner	modifier	Passed	No Issue
3	owner	read	Passed	No Issue
4	_checkOwner	internal	Passed	No Issue
5	renounceOwnership	write	access only Owner	No Issue
6	transferOwnership	write	access only Owner	No Issue
7	_transferOwnership	internal	Passed	No Issue
8	name	read	Passed	No Issue
9	symbol	read	Passed	No Issue
10	decimals	read	Passed	No Issue
11	totalSupply	read	Passed	No Issue
12	balanceOf	read	Passed	No Issue
13	transfer	write	Passed	No Issue
14	allowance	read	Passed	No Issue
15	approve	write	Passed	No Issue
16	transferFrom	write	Passed	No Issue
17	increaseAllowance	write	Passed	No Issue
18	decreaseAllowance	write	Passed	No Issue
19	_transfer	internal	Passed	No Issue
20	_mint	internal	Passed	No Issue
21	_burn	internal	Passed	No Issue
22	_approve	internal	Passed	No Issue
23	_spendAllowance	internal	Passed	No Issue
24	_beforeTokenTransfer	internal	Passed	No Issue
25	_afterTokenTransfer	internal	Passed	No Issue
26	decimals	write	Passed	No Issue
27	transfer	write	Passed	No Issue
28	canTransfer	internal	Passed	No Issue
29	isExcluded	external	Passed	No Issue
30	excludeAccount	external	access only Owner	No Issue
31	includeAccount	external	access only Owner	No Issue
32	setTxLimit	external	New limit is not set	Refer Audit Findings
33	setWalletLimit	external	New limit is not set	Refer Audit Findings
34	setMaxTransferPerDay	external	New limit is not set	Refer Audit Findings
35	toggleAllLimits	external	access only Owner	No Issue

ICO.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOwner	modifier	Passed	No Issue
3	random	read	Passed	No Issue
4	addReferralAddress	write	Passed	Fixed
5	getAddrByRefCode	read	Passed	No Issue
6	getRefByAddress	read	Passed	No Issue
7	getTotalRefRevenue	read	Passed	No Issue
8	getClaimPeriod	read	Passed	No Issue
9	buyTokens	external	Passed	No Issue
10	claim	external	Passed	No Issue
11	nextRound	external	access only Owner	No Issue
12	getDecimals	read	Passed	No Issue
13	setVesting	external	access only Owner	No Issue
14	setTokenPrice	external	Price limit is not set	Refer Audit Findings
15	transferOwnership	external	access only Owner	No Issue
16	withdrawTokens	external	access only Owner	No Issue
17	withdrawFiat	external	access only Owner	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have 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

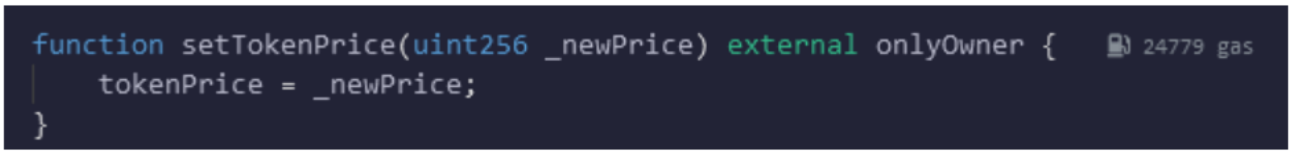
No critical severity vulnerabilities were found in the contract code.

High Severity

No high severity vulnerabilities were found in the contract code.

Medium

(1) Price limit is not set: **- ICO.sol**



```
function setTokenPrice(uint256 _newPrice) external onlyOwner {    24779 gas
    tokenPrice = _newPrice;
}
```

In the setTokenPrice function, Owner can set the price to any value. This might deter users that price might one day be set to 100% which will ultimately affect the token amount calculations.

Resolution: Consider adding an explicit limit to token price.

Status: Acknowledged

Low

(1) New tax, wallet address, transfer per day limit is not set: **- Infinitybee.sol**

In setTxlimit, setWalletLimit, setMaxTransferPerDay function, Owner can set the newlimit to any value. This might deter users that price might one day be set to 100% which will ultimately affect the token amount calculations.

Resolution: Consider adding an explicit limit to _newLimit.

Status: Acknowledged

(2) Function input parameters lack of check: **- ICO.sol**

Variable validation is not performed in below functions:

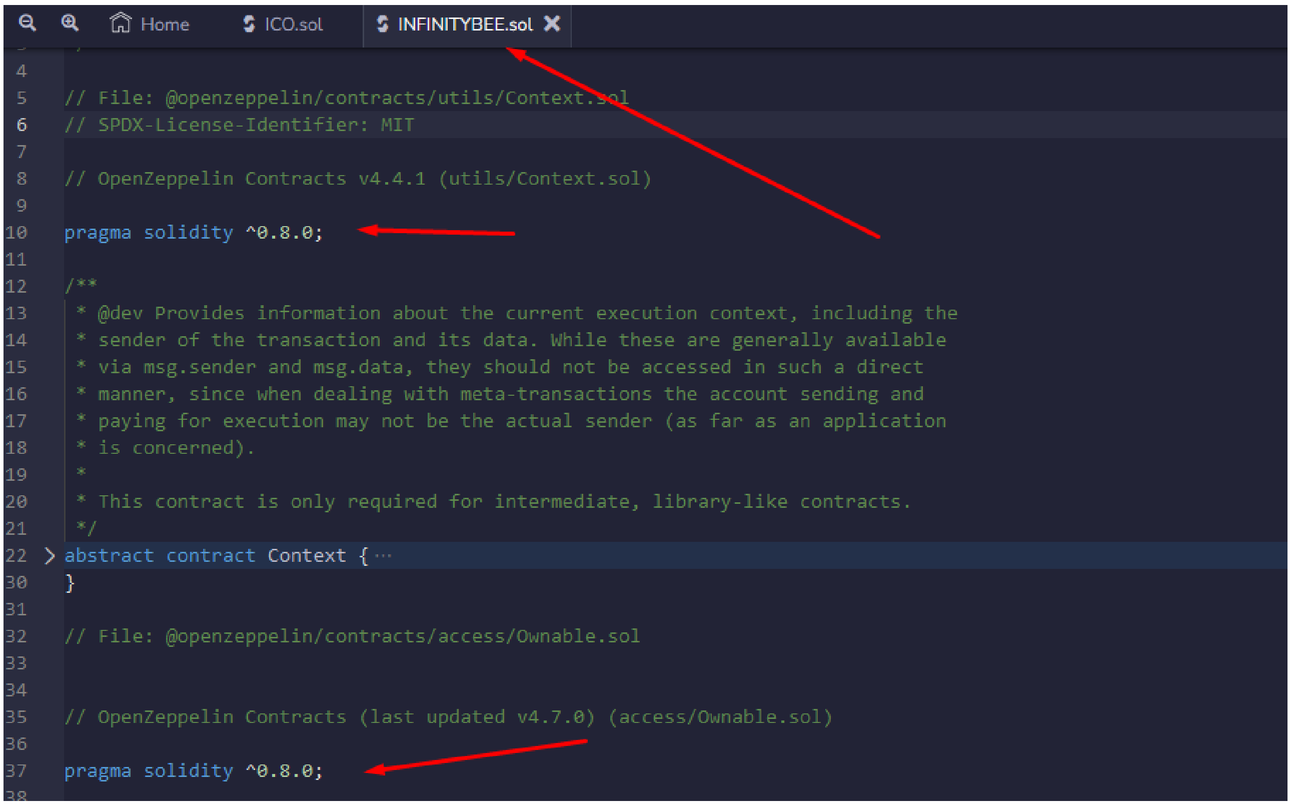
- addReferralAddress = _addr

Resolution: We advise to put validation: integer type variables should not be empty and greater than 0 & address type variables should not be address(0).

Status: Fixed

Very Low / Informational / Best practices:

(1) Multiple Pragma: **- Infinitybee.sol**



```
// File: @openzeppelin/contracts/utils/Context.sol
// SPDX-License-Identifier: MIT

// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)

pragma solidity ^0.8.0;

/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {⋯
}

// File: @openzeppelin/contracts/access/Ownable.sol


// OpenZeppelin Contracts (last updated v4.7.0) (access/Ownable.sol)

pragma solidity ^0.8.0;
```

There are multiple pragmas added to code with different solidity versions.

Resolution: We suggest keeping only one pragma on top of the contract code.

Status: Acknowledged

(2) SPDX license identifier is missing: **- Infinitybee.sol**

SPDX license identifier not provided in source file.

Resolution: We suggest adding an SPDX license identifier.

Status: Acknowledged

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:

INFINITYBEE.sol

- excludeAccount: Owner can check if the account is excluded or not then set status true.
- includeAccount: Owner can check if the account is excluded or not then set status false.
- setTxLimit: Transaction limit can be set by the owner.
- setWalletLimit: Wallet limit can be set by the owner.
- setMaxTransferPerDay: Maximum transfer limit per day can be set by the owner.
- toggleAllLimits: Toggle all limits can be set by the owner.

ICO.sol

- nextRound: Next round time can be set by the owner.
- setVesting: Vesting new period and percentage value can be set by the owner.
- setTokenPrice: New token Price can be set by the owner.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.
- withdrawTokens: Withdraw tokens by the owner.
- withdrawFiat: Withdraw fiat by the owner.

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.
- _checkOwner: Throws if the sender is not the owner.

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 https://sepolia.etherscan.io web link. And we have used all possible tests based on given objects as files. We had observed 1 medium severity issue, 2 low severity issues, 2 Informational severity issues in the smart contracts. We Confirm that these issues have been fixed / acknowledged in the revised code. **So, the smart contracts are ready for the mainnet deployment.**

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

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

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

The 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.

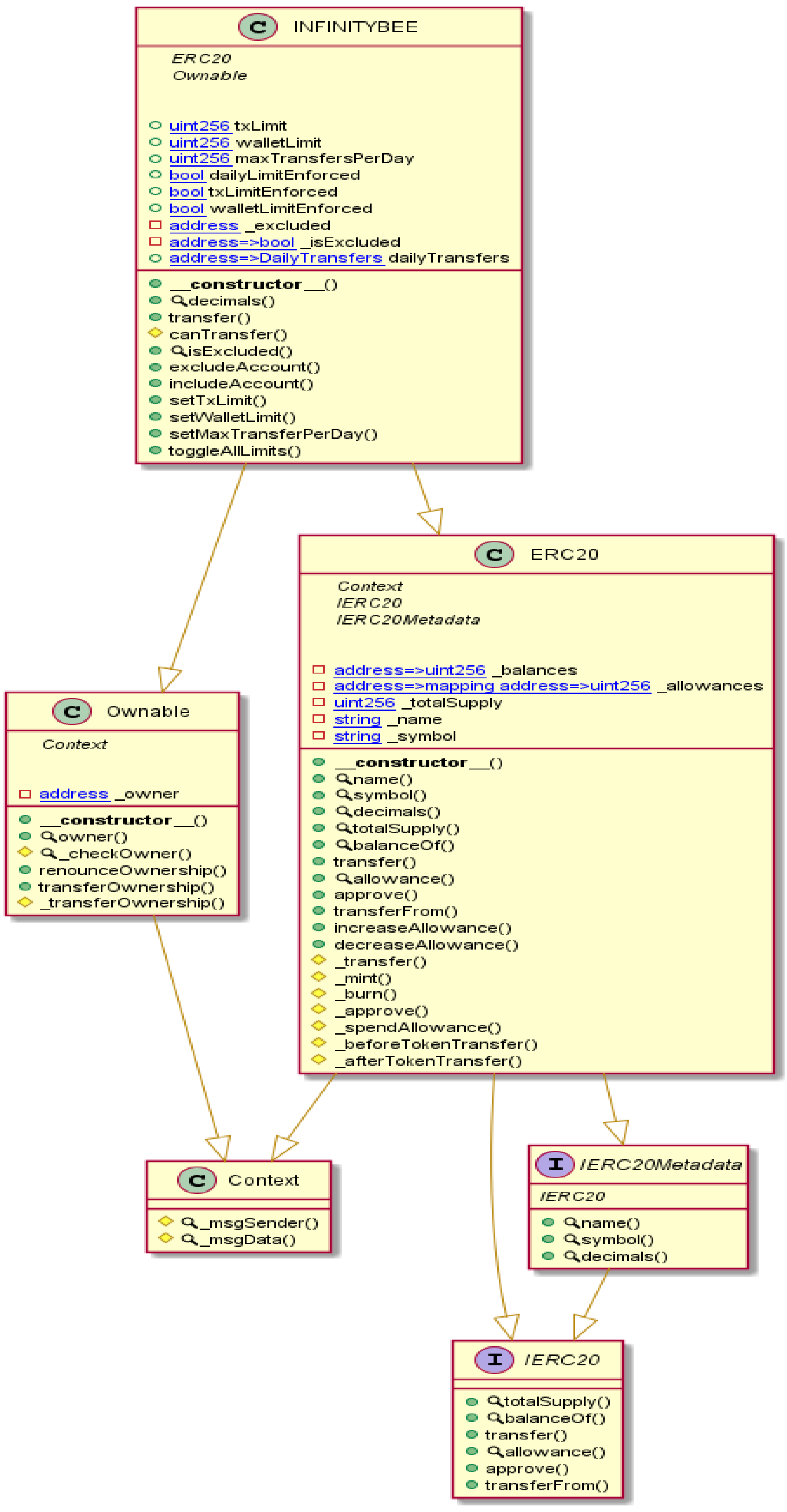
Technical Disclaimer

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.

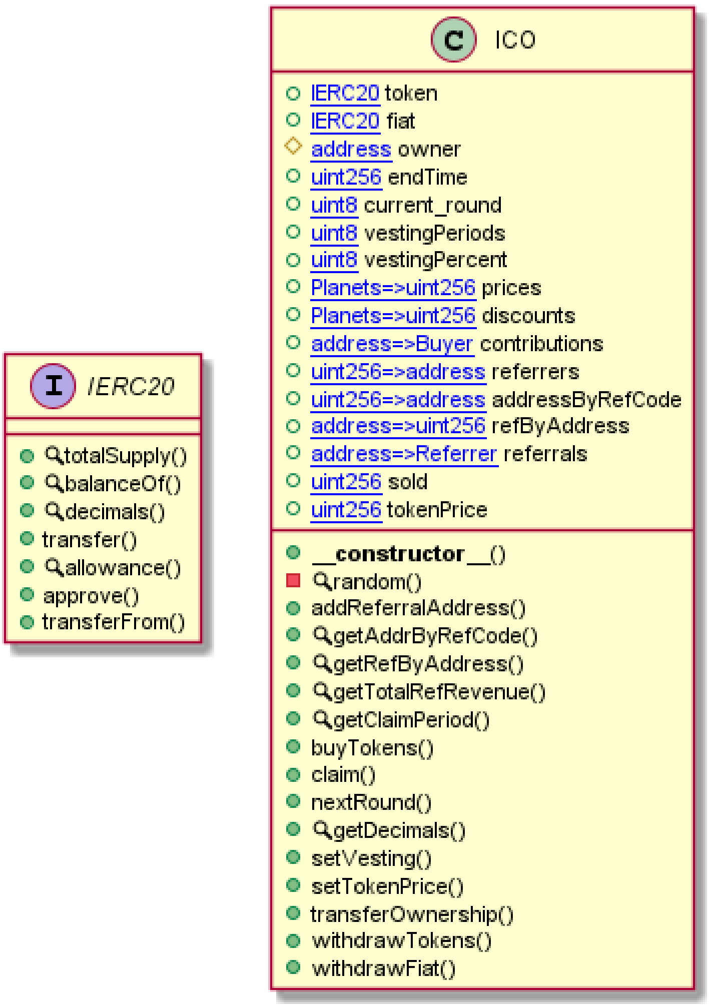
Appendix

Code Flow Diagram - InfinityBee (IFB) Token

INFINITYBEE Diagram

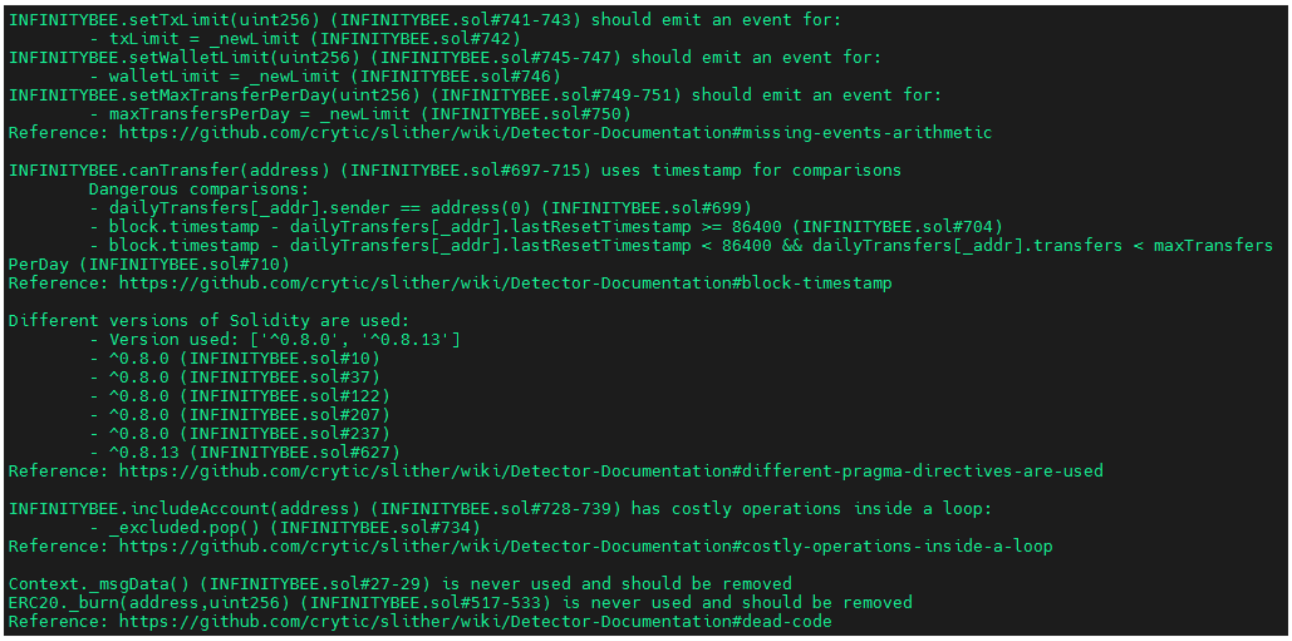


ICO Diagram


IERC20
totalSupply()
balanceOf()
decimals()
transfer()
allowance()
approve()
transferFrom()
ICO
IERC20 token
IERC20 fiat
address owner
uint256 endTime
uint8 current_round
uint8 vestingPeriods
uint8 vestingPercent
Planets=>uint256 prices
Planets=>uint256 discounts
address=>Buyer contributions
uint256=>address referrers
uint256=>address addressByRefCode
address=>uint256 refByAddress
address=>Referrer referrals
uint256 sold
uint256 tokenPrice
__constructor__()
random()
addReferralAddress()
getAddrByRefCode()
getRefByAddress()
getTotalRefRevenue()
getClaimPeriod()
buyTokens()
claim()
nextRound()
getDecimals()
setVesting()
setTokenPrice()
transferOwnership()
withdrawTokens()
withdrawFiat()

Slither Results Log

Slither log >> INFINITYBEE.sol



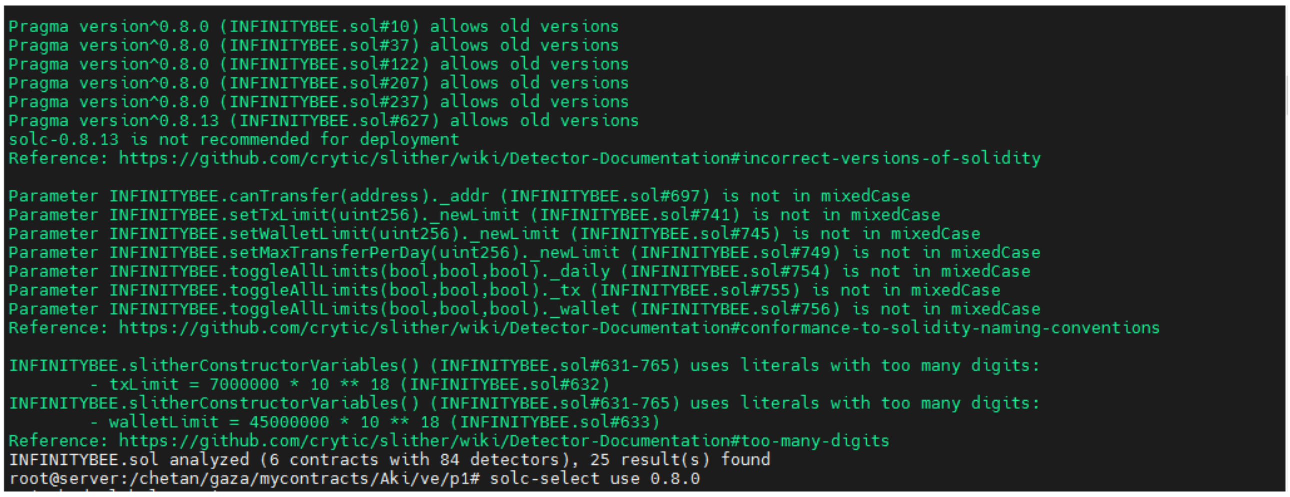
```
INFINITYBEE.setTxLimit(uint256) (INFINITYBEE.sol#741-743) should emit an event for:
        - txLimit = _newLimit (INFINITYBEE.sol#742)
INFINITYBEE.setWalletLimit(uint256) (INFINITYBEE.sol#745-747) should emit an event for:
        - walletLimit = _newLimit (INFINITYBEE.sol#746)
INFINITYBEE.setMaxTransferPerDay(uint256) (INFINITYBEE.sol#749-751) should emit an event for:
        - maxTransfersPerDay = _newLimit (INFINITYBEE.sol#750)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

INFINITYBEE.canTransfer(address) (INFINITYBEE.sol#697-715) uses timestamp for comparisons
        Dangerous comparisons:
        - dailyTransfers[_addr].sender == address(0) (INFINITYBEE.sol#699)
        - block.timestamp - dailyTransfers[_addr].lastResetTimestamp >= 86400 (INFINITYBEE.sol#704)
        - block.timestamp - dailyTransfers[_addr].lastResetTimestamp < 86400 && dailyTransfers[_addr].transfers < maxTransfers
PerDay (INFINITYBEE.sol#710)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.13']
        - ^0.8.0 (INFINITYBEE.sol#10)
        - ^0.8.0 (INFINITYBEE.sol#37)
        - ^0.8.0 (INFINITYBEE.sol#122)
        - ^0.8.0 (INFINITYBEE.sol#207)
        - ^0.8.0 (INFINITYBEE.sol#237)
        - ^0.8.13 (INFINITYBEE.sol#627)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

INFINITYBEE.includeAccount(address) (INFINITYBEE.sol#728-739) has costly operations inside a loop:
        - _excluded.pop() (INFINITYBEE.sol#734)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

Context._msgData() (INFINITYBEE.sol#27-29) is never used and should be removed
ERC20._burn(address,uint256) (INFINITYBEE.sol#517-533) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

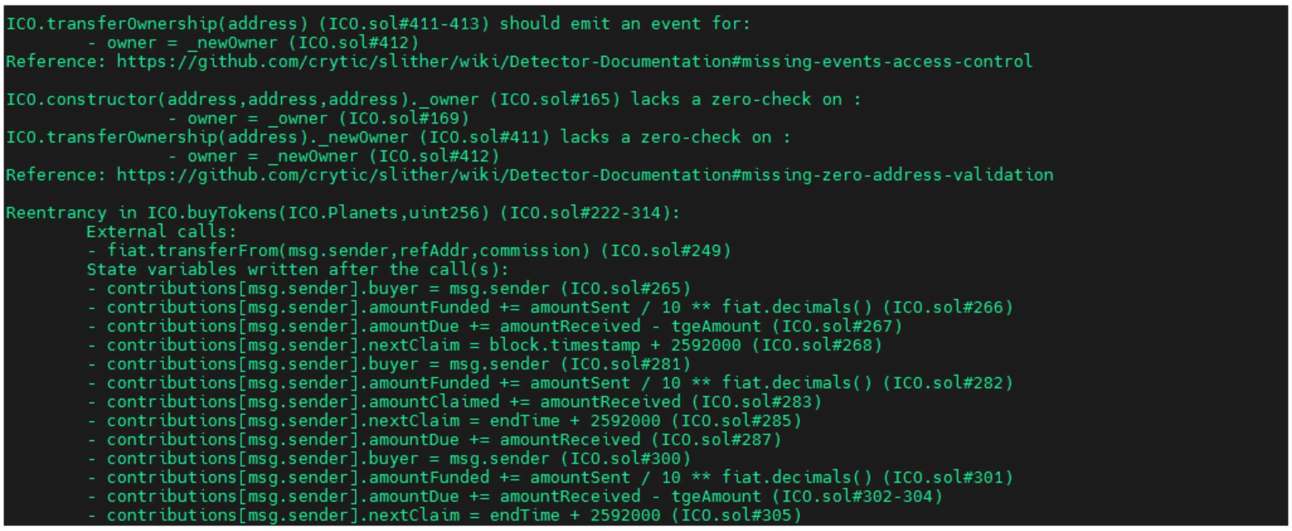


```
Pragma version^0.8.0 (INFINITYBEE.sol#10) allows old versions
Pragma version^0.8.0 (INFINITYBEE.sol#37) allows old versions
Pragma version^0.8.0 (INFINITYBEE.sol#122) allows old versions
Pragma version^0.8.0 (INFINITYBEE.sol#207) allows old versions
Pragma version^0.8.0 (INFINITYBEE.sol#237) allows old versions
Pragma version^0.8.13 (INFINITYBEE.sol#627) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter INFINITYBEE.canTransfer(address)._addr (INFINITYBEE.sol#697) is not in mixedCase
Parameter INFINITYBEE.setTxLimit(uint256)._newLimit (INFINITYBEE.sol#741) is not in mixedCase
Parameter INFINITYBEE.setWalletLimit(uint256)._newLimit (INFINITYBEE.sol#745) is not in mixedCase
Parameter INFINITYBEE.setMaxTransferPerDay(uint256)._newLimit (INFINITYBEE.sol#749) is not in mixedCase
Parameter INFINITYBEE.toggleAllLimits(bool,bool,bool)._daily (INFINITYBEE.sol#754) is not in mixedCase
Parameter INFINITYBEE.toggleAllLimits(bool,bool,bool)._tx (INFINITYBEE.sol#755) is not in mixedCase
Parameter INFINITYBEE.toggleAllLimits(bool,bool,bool)._wallet (INFINITYBEE.sol#756) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

INFINITYBEE.slitherConstructorVariables() (INFINITYBEE.sol#631-765) uses literals with too many digits:
        - txLimit = 7000000 * 10 ** 18 (INFINITYBEE.sol#632)
INFINITYBEE.slitherConstructorVariables() (INFINITYBEE.sol#631-765) uses literals with too many digits:
        - walletLimit = 45000000 * 10 ** 18 (INFINITYBEE.sol#633)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFINITYBEE.sol analyzed (6 contracts with 84 detectors), 25 result(s) found
root@server:/chetan/gaza/mycontracts/Aki/ve/p1# solc-select use 0.8.0
```

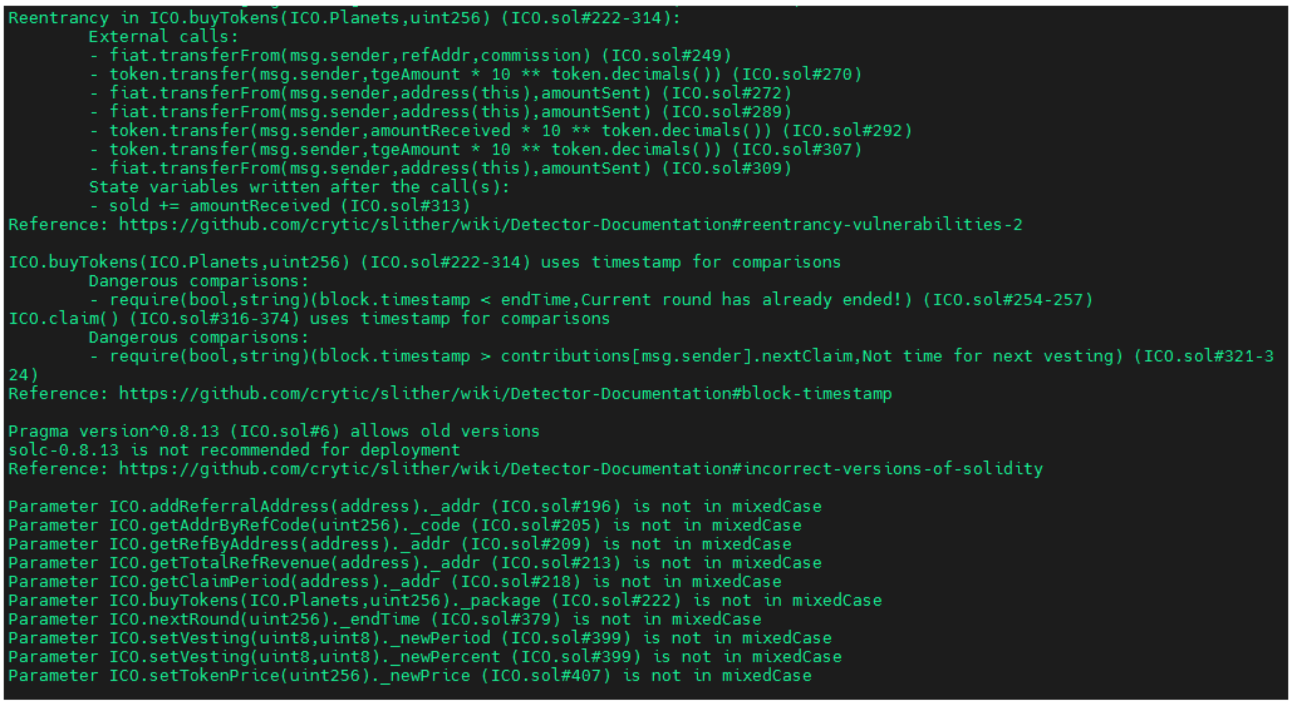
Slither log >> ICO.sol



```
ICO.transferOwnership(address) (ICO.sol#411-413) should emit an event for:
        - owner = _newOwner (ICO.sol#412)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

ICO.constructor(address,address,address)._owner (ICO.sol#165) lacks a zero-check on :
                - owner = _owner (ICO.sol#169)
ICO.transferOwnership(address)._newOwner (ICO.sol#411) lacks a zero-check on :
                - owner = _newOwner (ICO.sol#412)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in ICO.buyTokens(ICO.Planets,uint256) (ICO.sol#222-314):
        External calls:
        - fiat.transferFrom(msg.sender,refAddr,commission) (ICO.sol#249)
        State variables written after the call(s):
        - contributions[msg.sender].buyer = msg.sender (ICO.sol#265)
        - contributions[msg.sender].amountFunded += amountSent / 10 ** fiat.decimals() (ICO.sol#266)
        - contributions[msg.sender].amountDue += amountReceived - tgeAmount (ICO.sol#267)
        - contributions[msg.sender].nextClaim = block.timestamp + 2592000 (ICO.sol#268)
        - contributions[msg.sender].buyer = msg.sender (ICO.sol#281)
        - contributions[msg.sender].amountFunded += amountSent / 10 ** fiat.decimals() (ICO.sol#282)
        - contributions[msg.sender].amountClaimed += amountReceived (ICO.sol#283)
        - contributions[msg.sender].nextClaim = endTime + 2592000 (ICO.sol#285)
        - contributions[msg.sender].amountDue += amountReceived (ICO.sol#287)
        - contributions[msg.sender].buyer = msg.sender (ICO.sol#300)
        - contributions[msg.sender].amountFunded += amountSent / 10 ** fiat.decimals() (ICO.sol#301)
        - contributions[msg.sender].amountDue += amountReceived - tgeAmount (ICO.sol#302-304)
        - contributions[msg.sender].nextClaim = endTime + 2592000 (ICO.sol#305)
```

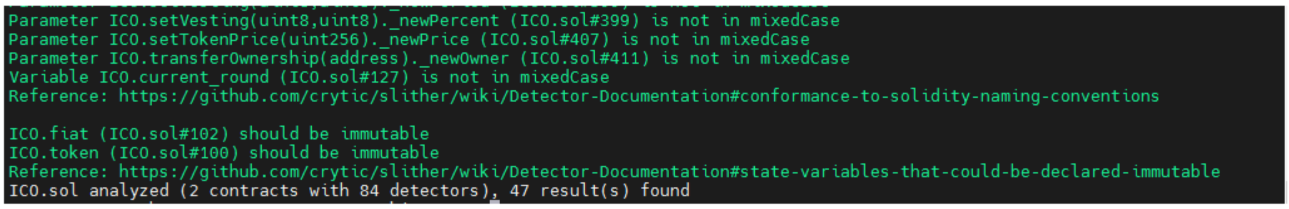


```
Reentrancy in ICO.buyTokens(ICO.Planets,uint256) (ICO.sol#222-314):
        External calls:
        - fiat.transferFrom(msg.sender,refAddr,commission) (ICO.sol#249)
        - token.transfer(msg.sender,tgeAmount * 10 ** token.decimals()) (ICO.sol#270)
        - fiat.transferFrom(msg.sender,address(this),amountSent) (ICO.sol#272)
        - fiat.transferFrom(msg.sender,address(this),amountSent) (ICO.sol#289)
        - token.transfer(msg.sender,amountReceived * 10 ** token.decimals()) (ICO.sol#292)
        - token.transfer(msg.sender,tgeAmount * 10 ** token.decimals()) (ICO.sol#307)
        - fiat.transferFrom(msg.sender,address(this),amountSent) (ICO.sol#309)
        State variables written after the call(s):
        - sold += amountReceived (ICO.sol#313)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

ICO.buyTokens(ICO.Planets,uint256) (ICO.sol#222-314) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < endTime,Current round has already ended!) (ICO.sol#254-257)
ICO.claim() (ICO.sol#316-374) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > contributions[msg.sender].nextClaim,Not time for next vesting) (ICO.sol#321-3
24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Pragma version^0.8.13 (ICO.sol#6) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter ICO.addReferralAddress(address)._addr (ICO.sol#196) is not in mixedCase
Parameter ICO.getAddrByRefCode(uint256)._code (ICO.sol#205) is not in mixedCase
Parameter ICO.getRefByAddress(address)._addr (ICO.sol#209) is not in mixedCase
Parameter ICO.getTotalRefRevenue(address)._addr (ICO.sol#213) is not in mixedCase
Parameter ICO.getClaimPeriod(address)._addr (ICO.sol#218) is not in mixedCase
Parameter ICO.buyTokens(ICO.Planets,uint256)._package (ICO.sol#222) is not in mixedCase
Parameter ICO.nextRound(uint256)._endTime (ICO.sol#379) is not in mixedCase
Parameter ICO.setVesting(uint8,uint8)._newPeriod (ICO.sol#399) is not in mixedCase
Parameter ICO.setVesting(uint8,uint8)._newPercent (ICO.sol#399) is not in mixedCase
Parameter ICO.setTokenPrice(uint256)._newPrice (ICO.sol#407) is not in mixedCase
```

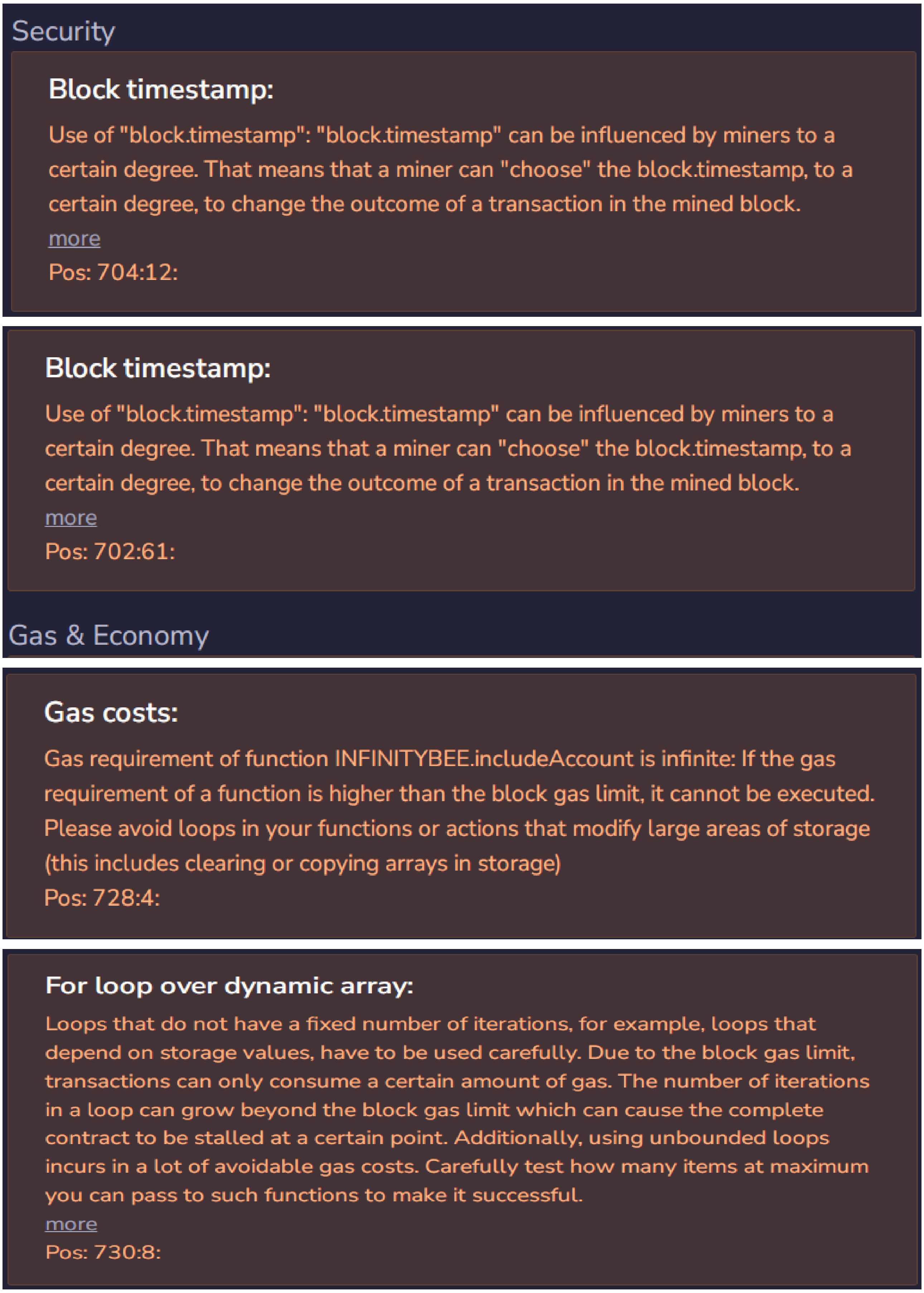


```
Parameter ICO.setVesting(uint8,uint8)._newPercent (ICO.sol#399) is not in mixedCase
Parameter ICO.setTokenPrice(uint256)._newPrice (ICO.sol#407) is not in mixedCase
Parameter ICO.transferOwnership(address)._newOwner (ICO.sol#411) is not in mixedCase
Variable ICO.current_round (ICO.sol#127) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

ICO.fiat (ICO.sol#102) should be immutable
ICO.token (ICO.sol#100) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
ICO.sol analyzed (2 contracts with 84 detectors), 47 result(s) found
```

Solidity Static Analysis

INFINITYBEE.sol



Security

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 704:12:

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: 702:61:

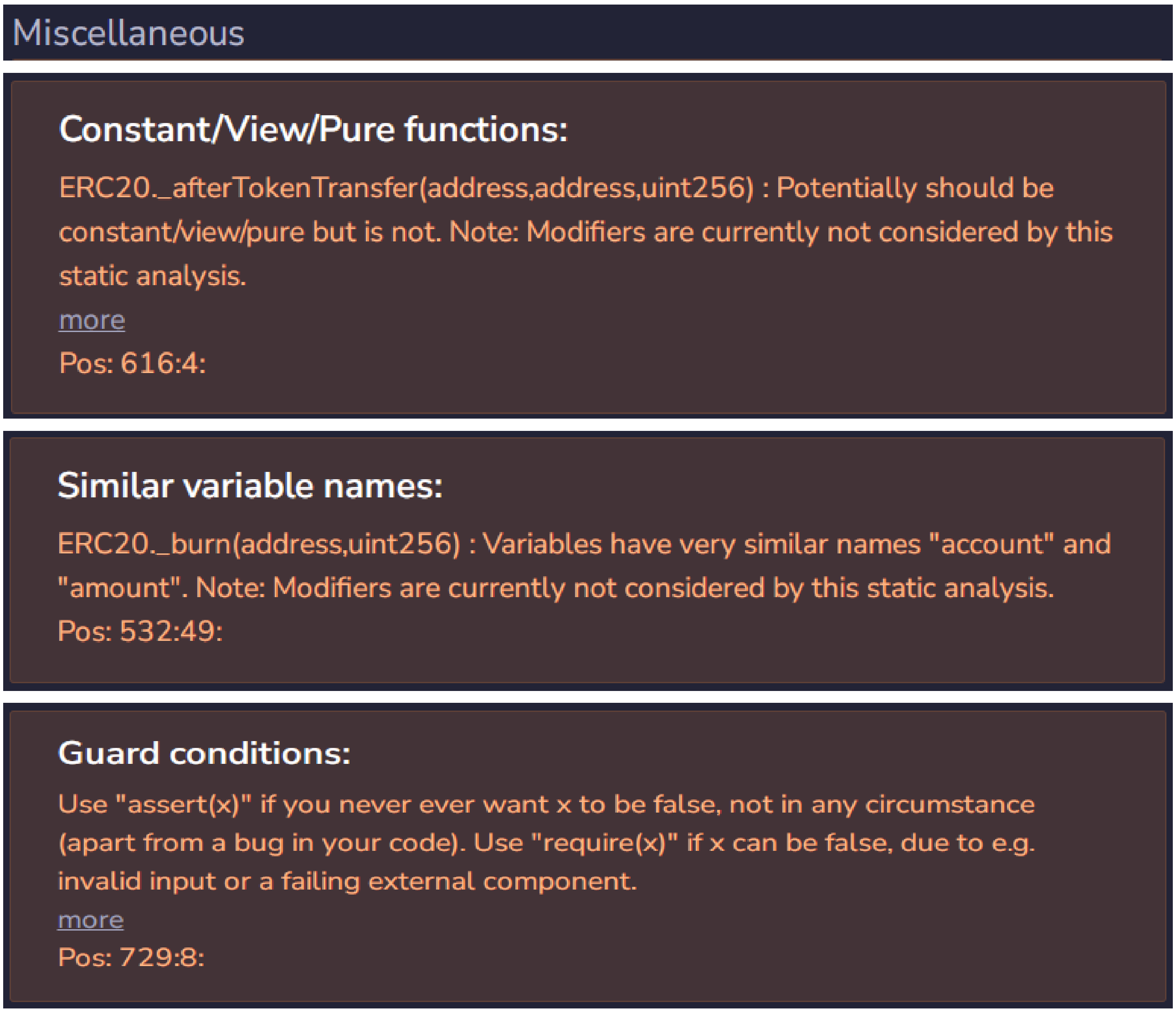
Gas & Economy

Gas costs:

Gas requirement of function INFINITYBEE.includeAccount 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: 728:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 730:8:


Miscellaneous

Constant/View/Pure functions:

ERC20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 616:4:

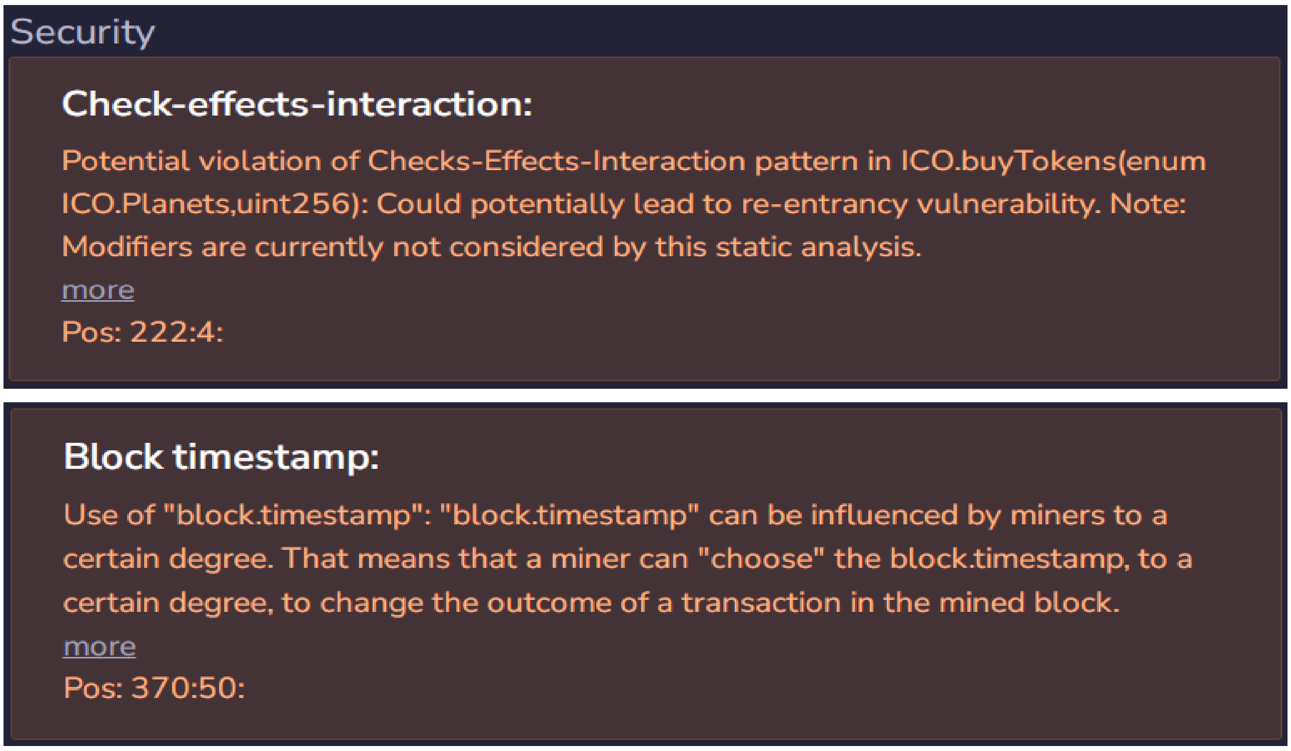
Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 532:49:

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: 729:8:

ICO.sol

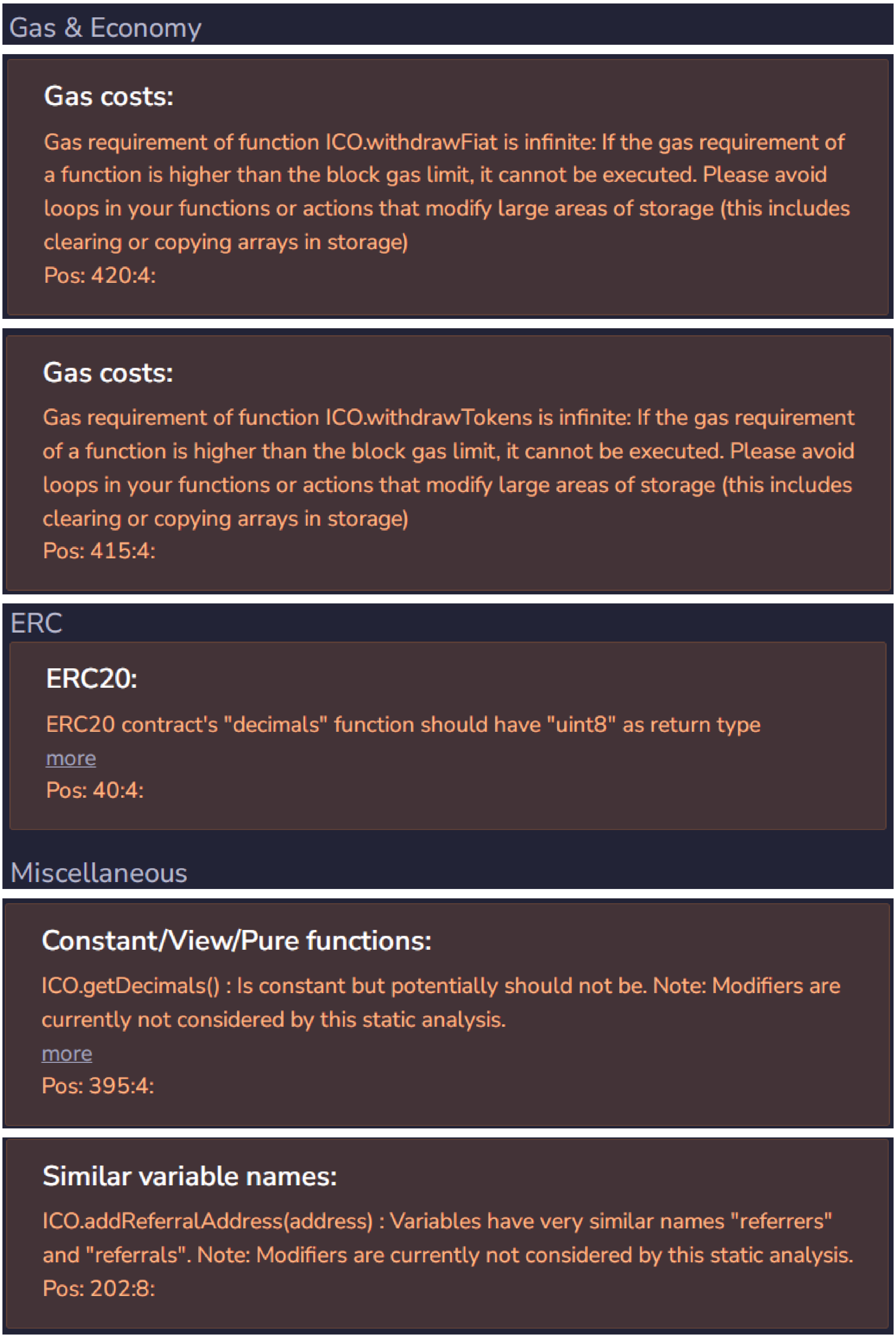

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in ICO.buyTokens(enum ICO.Planets,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 222:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 370:50:



Gas & Economy

Gas costs:

Gas requirement of function ICO.withdrawFiat 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: 420:4:

Gas costs:

Gas requirement of function ICO.withdrawTokens 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: 415:4:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 40:4:

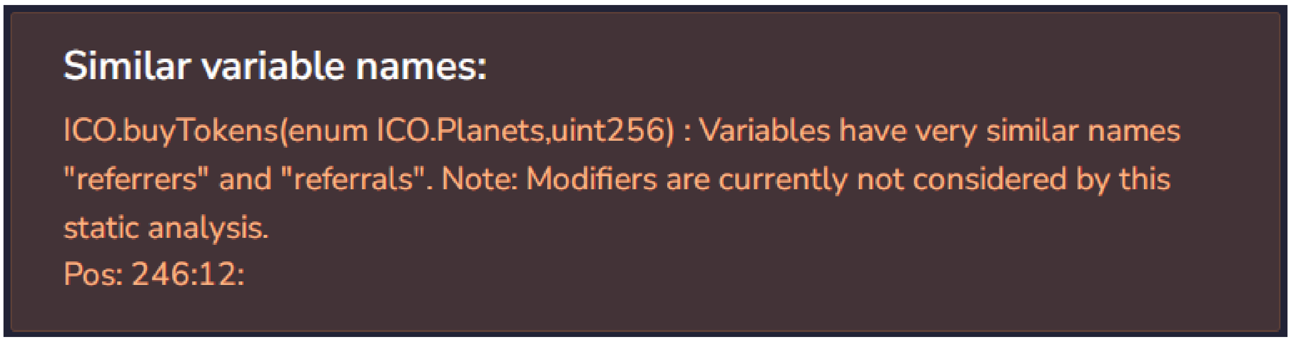
Miscellaneous

Constant/View/Pure functions:

ICO.getDecimals() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 395:4:

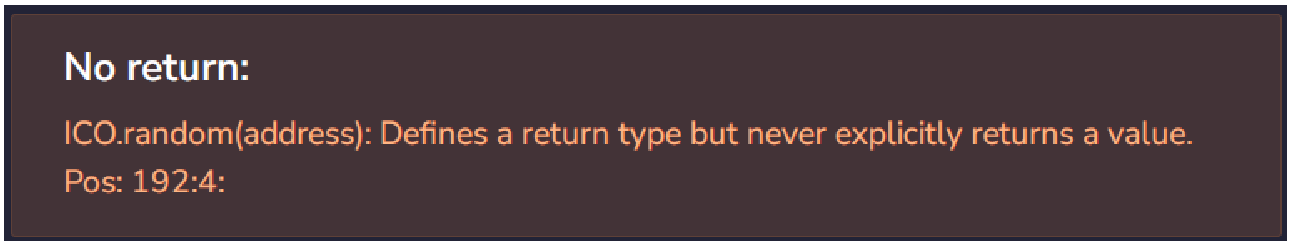
Similar variable names:

ICO.addReferralAddress(address) : Variables have very similar names "referrers" and "referrals". Note: Modifiers are currently not considered by this static analysis.
Pos: 202:8:



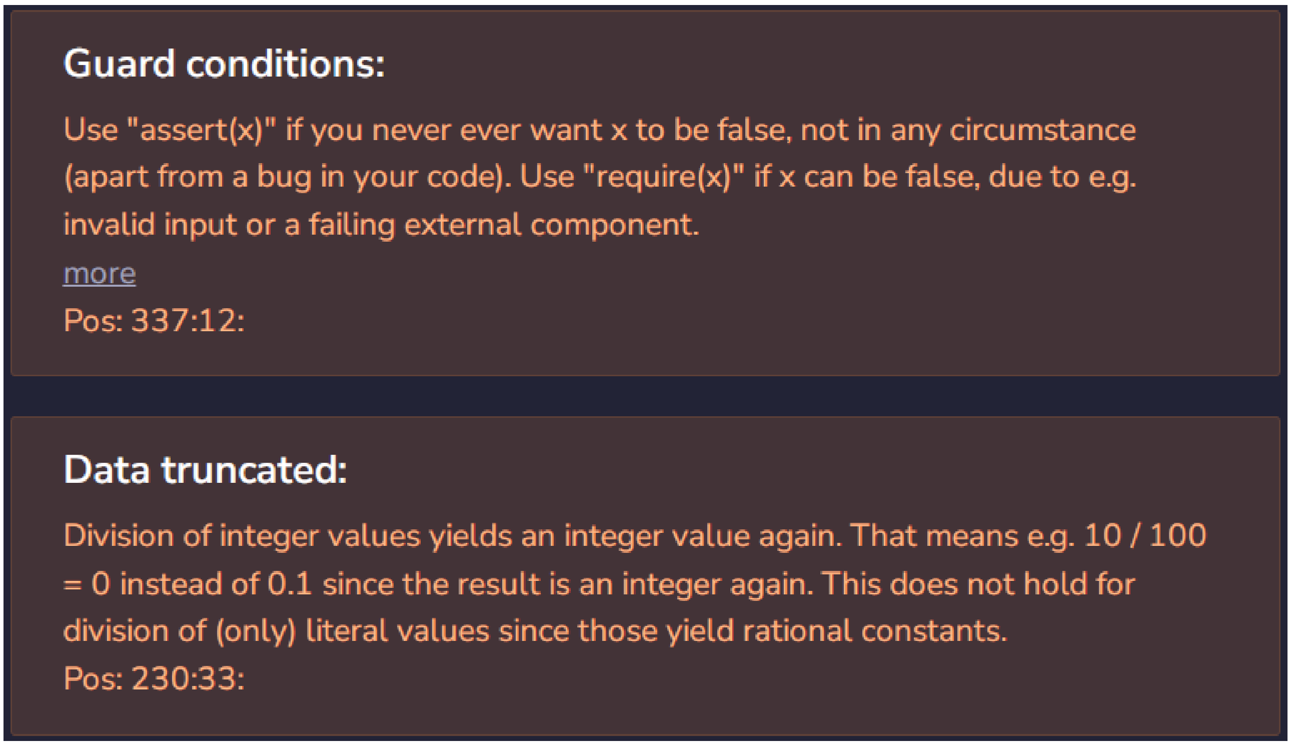
Similar variable names:

ICO.buyTokens(enum ICO.Planets,uint256) : Variables have very similar names "referrers" and "referrals". Note: Modifiers are currently not considered by this static analysis.
Pos: 246:12:



No return:

ICO.random(address): Defines a return type but never explicitly returns a value.
Pos: 192:4:

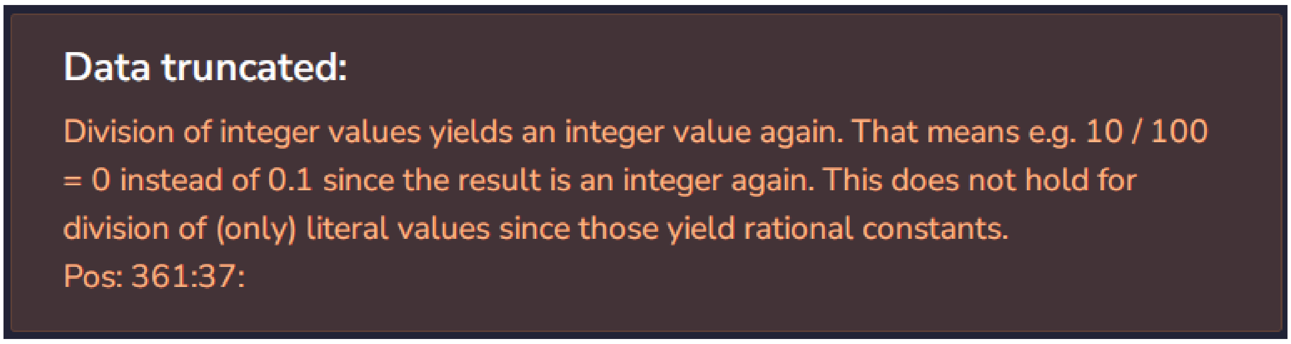


Guard conditions:

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: 337:12:

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: 230:33:

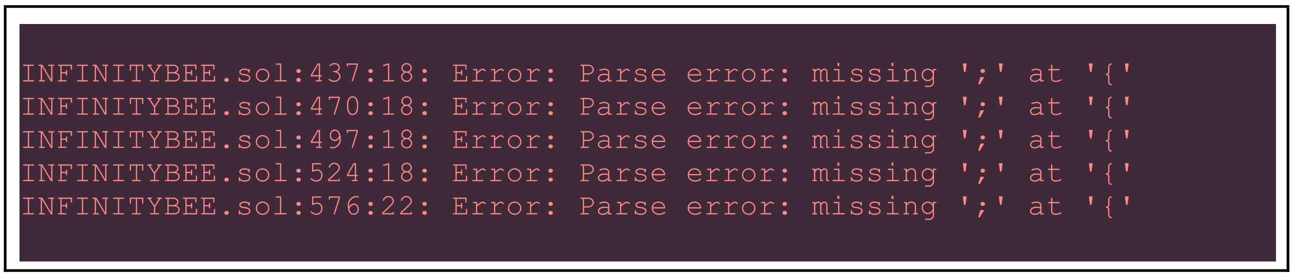


Data truncated:

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: 361:37:

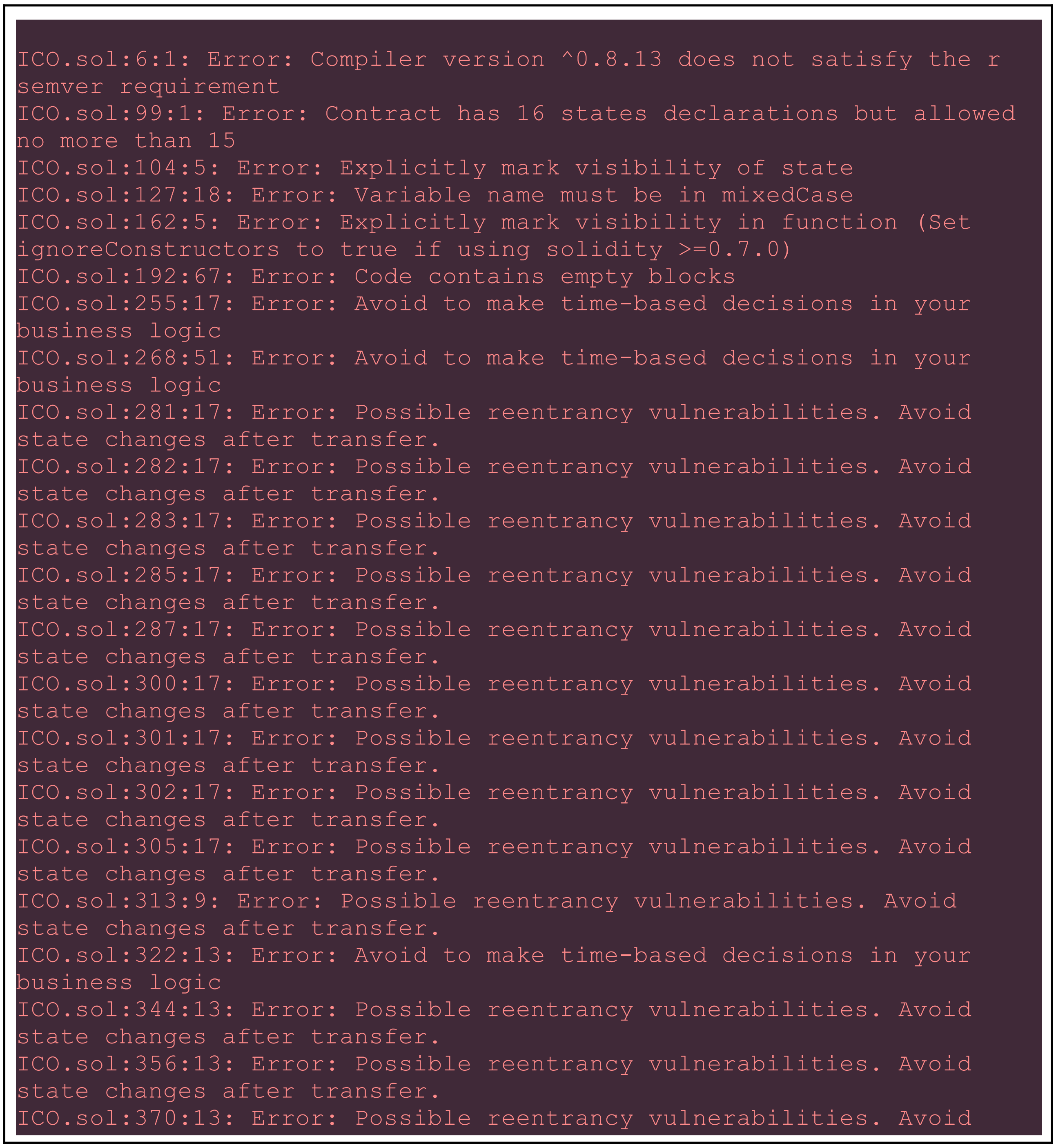
Solhint Linter

INFINITYBEE.sol

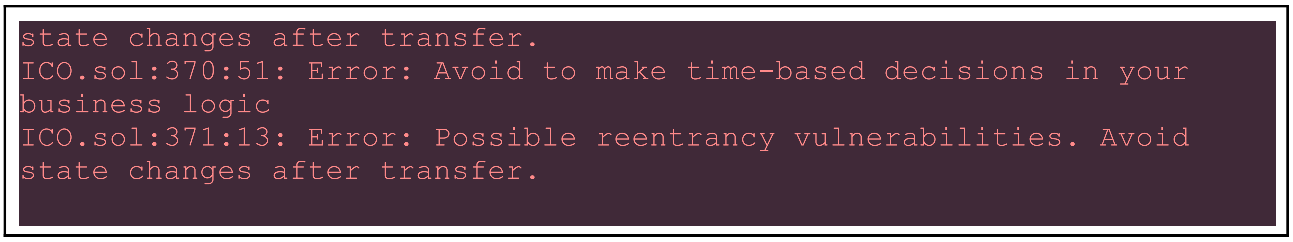


```
INFINITYBEE.sol:437:18: Error: Parse error: missing ';' at '{'
INFINITYBEE.sol:470:18: Error: Parse error: missing ';' at '{'
INFINITYBEE.sol:497:18: Error: Parse error: missing ';' at '{'
INFINITYBEE.sol:524:18: Error: Parse error: missing ';' at '{'
INFINITYBEE.sol:576:22: Error: Parse error: missing ';' at '{'
```

ICO.sol



```
ICO.sol:6:1: Error: Compiler version ^0.8.13 does not satisfy the r
semver requirement
ICO.sol:99:1: Error: Contract has 16 states declarations but allowed
no more than 15
ICO.sol:104:5: Error: Explicitly mark visibility of state
ICO.sol:127:18: Error: Variable name must be in mixedCase
ICO.sol:162:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
ICO.sol:192:67: Error: Code contains empty blocks
ICO.sol:255:17: Error: Avoid to make time-based decisions in your
business logic
ICO.sol:268:51: Error: Avoid to make time-based decisions in your
business logic
ICO.sol:281:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:282:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:283:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:285:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:287:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:300:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:301:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:302:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:305:17: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:313:9: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:322:13: Error: Avoid to make time-based decisions in your
business logic
ICO.sol:344:13: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:356:13: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
ICO.sol:370:13: Error: Possible reentrancy vulnerabilities. Avoid
```



```
state changes after transfer.
ICO.sol:370:51: Error: Avoid to make time-based decisions in your
business logic
ICO.sol:371:13: Error: Possible reentrancy vulnerabilities. Avoid
state changes after transfer.
```

Software analysis result:

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

Email: audit@EtherAuthority.io