Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Streakk Chain
Platform:	Binance Smart Chain
Language:	Solidity
Date:	June 4th, 2023

Table of contents

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

EtherAuthority was contracted by Streakk Chain to perform the Security audit of the Streakk 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 4th, 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

- Streakk Chain (STKC) is a BEP20 standard token contract on the Binance Smart Chain blockchain.
- The Streakk Chain Contracts handle multiple contracts, and all contracts have different functions.
    - Streakk: Streak is used for mint token and burn token.
    - Freezable: This contract is used for freeze Account addresses.
    - TrustedContracts: This contract is used to add trusted Contracts addresses and notify trusted addresses.
    - BEP20: This contract is used to set the values for {name} and {symbol}.
- Streakk Chain is a smart contract which has functions mint, burn, unfreezed, freezeAccount, transferAnyBEP20Token, etc.
- There are 4 smart contracts, which were included in the audit scope. And there were some standard library codes, such as OpenZepelin, that were excluded. Because those standard library code is considered as time tested and community audited, so we can safely ignore them.

Audit scope

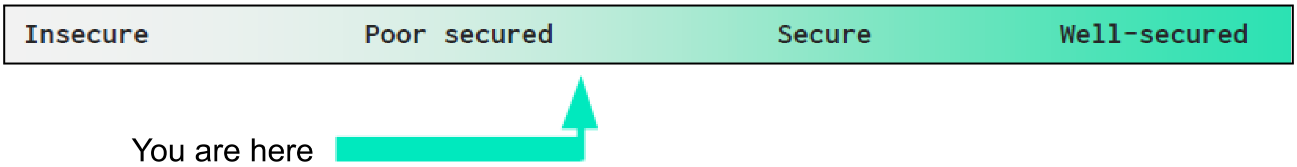
Name	**Code Review and Security Analysis Report for Streakk Smart Contracts**
Platform	**BSC / Solidity**
File 1	Streakk.sol
File 1 MD5 Hash	E015E7C429D135441E2D25E6AE6FEAF2
File 2	Freezable.sol
File 2 MD5 Hash	F6BB7C7AD4AB6C8D3020E8CCFC9CA4FB
File 3	TrustedContracts.sol
File 3 MD5 Hash	2FA04ABBA8234CCD3BA40B57D7095507
File 4	BEP20.sol
File 4 MD5 Hash	4DB97A4FE83068FD74C6375145E99E07
Audit Date	June 4th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: - Community Mining/Staking: 50% - Reserve: 10% - Treasury: 15% - Team: 5% - Web3 Foundation: 620% - Team Tokens are locked for 21 Months. - The tokens will be minted then the Circulating Supply will be: 5 Million.	**YES, This is valid. Owner wallet's private key must be handled very securely. Because if that is compromised, then it will create problems.**
File 1 Streakk.sol - Name: Streakk Chain - Symbol: STKC - decimals: 18 - Total Supply: 85 million	**YES, This is valid.**
File 2 Freezable.sol **Owner has control over following functions:** - Freeze account address can be set by the owner. - Emergency freeze all account addresses by the owner.	**YES, This is valid.**
File 3 TrustedContracts.sol - Trusted contract address can be set by the owner. - Owner of the contract can transfer any BEP20 compatible tokens sent to this contract.	**YES, This is valid.**
File 4 BEP20.sol - Total Supply: 85 million	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Un-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 1 critical, 0 high, 0 medium and 2 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	Passed
	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	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	Moderated
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: FAILED

Code Quality

This audit scope has 4 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 Streakk Chain Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Streakk Chain Protocol.

The Streakk Chain 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 a Streakk Protocol 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.

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

Streakk.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Can not mint for Owner address	Refer Audit findings
2	onlyOwnerOrBridge	modifier	Passed	No Issue
3	mint	write	Function input parameters lack of check	Refer Audit findings
4	_transfer	internal	Passed	No Issue
5	_approve	internal	Passed	No Issue
6	burn	write	Passed	No Issue
7	currentSupply	read	Passed	No Issue
8	startTimestamp	read	Passed	No Issue
9	bulkTransfer	write	Function input parameters lack of check	Refer Audit findings
10	bulkTransferFrom	write	Function input parameters lack of check	Refer Audit findings
11	signedTransfer	write	Function input parameters lack of check	Refer Audit findings
12	signedApprove	write	Passed	No Issue
13	signedIncreaseAllowance	write	Function input parameters lack of check	Refer Audit findings
14	signedDecreaseApproval	write	Function input parameters lack of check	Refer Audit findings
15	getSigner	write	Passed	No Issue
16	getChainID	read	Passed	No Issue
17	_setApproveWithAuthorization	internal	Passed	No Issue
18	_signedIncreaseAllowance	write	Passed	No Issue
19	_signedDecreaseAllowance	write	Function input parameters lack of check	Refer Audit findings
20	_transferWithAuthorization	internal	Passed	No Issue
21	_domainSeparatorV4	internal	Passed	No Issue
21	_buildDomainSeparator	read	Passed	No Issue
22	_hashTypedDataV4	internal	Passed	No Issue
23	eip712Domain	read	Passed	No Issue
24	onlyRole	modifier	Passed	No Issue
25	supportsInterface	read	Passed	No Issue

26	hasRole	read	Passed	No Issue
27	_checkRole	internal	Passed	No Issue
28	_checkRole	internal	Passed	No Issue
29	getRoleAdmin	read	Passed	No Issue
30	grantRole	write	access only Role	No Issue
31	revokeRole	write	access only Role	No Issue
32	renounceRole	write	Passed	No Issue
33	_setupRole	internal	Passed	No Issue
34	_setRoleAdmin	internal	Passed	No Issue
35	_grantRole	internal	Passed	No Issue
36	_revokeRole	internal	Passed	No Issue
37	onlyOwner	modifier	Passed	No Issue
38	owner	read	Passed	No Issue
39	_checkOwner	internal	Passed	No Issue
40	renounceOwnership	write	access only Owner	No Issue
41	transferOwnership	write	access only Owner	No Issue
42	_transferOwnership	internal	Passed	No Issue
43	unfreezed	modifier	Passed	No Issue
44	noEmergencyFreeze	modifier	Passed	No Issue
45	freezeAccount	write	access only Owner	No Issue
46	emergencyFreezeAllAccounts	write	access only Owner	No Issue
47	isContract	read	Passed	No Issue
48	addTrustedContracts	write	access only Owner	No Issue
49	notifyTrustedContract	internal	Passed	No Issue
50	transferAnyBEP20Token	write	access only Owner	No Issue
51	totalSupply	read	Passed	No Issue
52	balanceOf	read	Passed	No Issue
53	transfer	write	Passed	No Issue
54	allowance	read	Passed	No Issue
55	approve	write	Passed	No Issue
56	transferFrom	write	Passed	No Issue
57	increaseAllowance	write	Passed	No Issue
58	decreaseAllowance	write	Passed	No Issue
59	_transfer	internal	Passed	No Issue
60	_mint	internal	Passed	No Issue
61	_burn	internal	Passed	No Issue
62	_approve	internal	Passed	No Issue
63	_spendAllowance	internal	Passed	No Issue
64	_beforeTokenTransfer	internal	Passed	No Issue
65	_afterTokenTransfer	internal	Passed	No Issue

Freezable.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	_checkOwner	internal	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	unfreezed	modifier	Passed	No Issue
8	noEmergencyFreeze	modifier	Passed	No Issue
9	freezeAccount	write	access only Owner	No Issue
10	emergencyFreezeAllAccounts	write	access only Owner	No Issue

TrustedContracts.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	_checkOwner	internal	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	isContract	read	Passed	No Issue
8	addTrustedContracts	write	access only Owner	No Issue
9	notifyTrustedContract	internal	Passed	No Issue
10	transferAnyBEP20Token	write	access only Owner	No Issue

BEP20.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Can not mint for Owner address	Refer Audit findings
2	totalSupply	read	Passed	No Issue

3	balanceOf	read	Passed	No Issue
4	transfer	write	Passed	No Issue
5	allowance	read	Passed	No Issue
6	approve	write	Passed	No Issue
7	transferFrom	write	Passed	No Issue
8	increaseAllowance	write	Passed	No Issue
9	decreaseAllowance	write	Passed	No Issue
10	_transfer	internal	Passed	No Issue
11	_mint	internal	Passed	No Issue
12	_burn	internal	Passed	No Issue
13	_approve	internal	Passed	No Issue
14	_spendAllowance	internal	Passed	No Issue
15	_beforeTokenTransfer	internal	Passed	No Issue
16	_afterTokenTransfer	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

(1) Can not mint for Owner address: **Streakk.sol, BEP20.sol**

In the constructor, tokens are not alloted to the owner and can not mint tokens for the owner address.

Resolution: We suggest checking token allocation logic and allot tokens to the owner too.

High Severity

No high severity vulnerabilities were found in the contract code.

Medium

No medium severity vulnerabilities were found in the contract code.

Low

(1) Function input parameters lack of check: **Streakk.sol**

Variable validation is not performed in below functions:

- bulkTransferFrom = sender ,to_addresses
- signedTransfer = from , to
- bulkTransfer = to_addresses
- mint = to
- _signedDecreaseAllowance = from , to
- signedDecreaseApproval = from , to
- signedIncreaseAllowance = from , to

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).

(2) Other programming issues: **Streakk.sol**

The signedIncreaseAllowance is a public function; inside this function, the _signedIncreaseAllowance function is called, which is also a public function.

Resolution: We suggest writing the logic of the _signedIncreaseAllowance function logic into the signedIncreaseAllowance function and removing the _signedIncreaseAllowance function.

Very Low / Informational / Best practices:

No Informational severity vulnerabilities were found in the contract code.

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:

Streakk.sol

- mint: The owner can mint a token.

TrustedContracts.sol

- addTrustedContracts: Trusted contract address can be set by the owner.
- transferAnyBEP20Token: Owner of contract can transfer any BEP20 compatible tokens sent to this contract.

Freezable.sol

- freezeAccount: Freeze account address can be set by the owner.
- emergencyFreezeAllAccounts: Emergency freeze all account addresses by the owner.

AccessControl.sol

- grantRole: Grants `role` to `account` can be set by the owner.
- revokeRole: Revokes `role` from `account` by the owner.
- renounceRole: Renounce Role from `account` 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 file. And we have used all possible tests based on given objects as files. We had observed 1 critical severity issue, 2 low severity issues in the smart contracts. **So, the smart contracts are ready for the mainnet deployment after resolving those issues.**

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 **"Un-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.

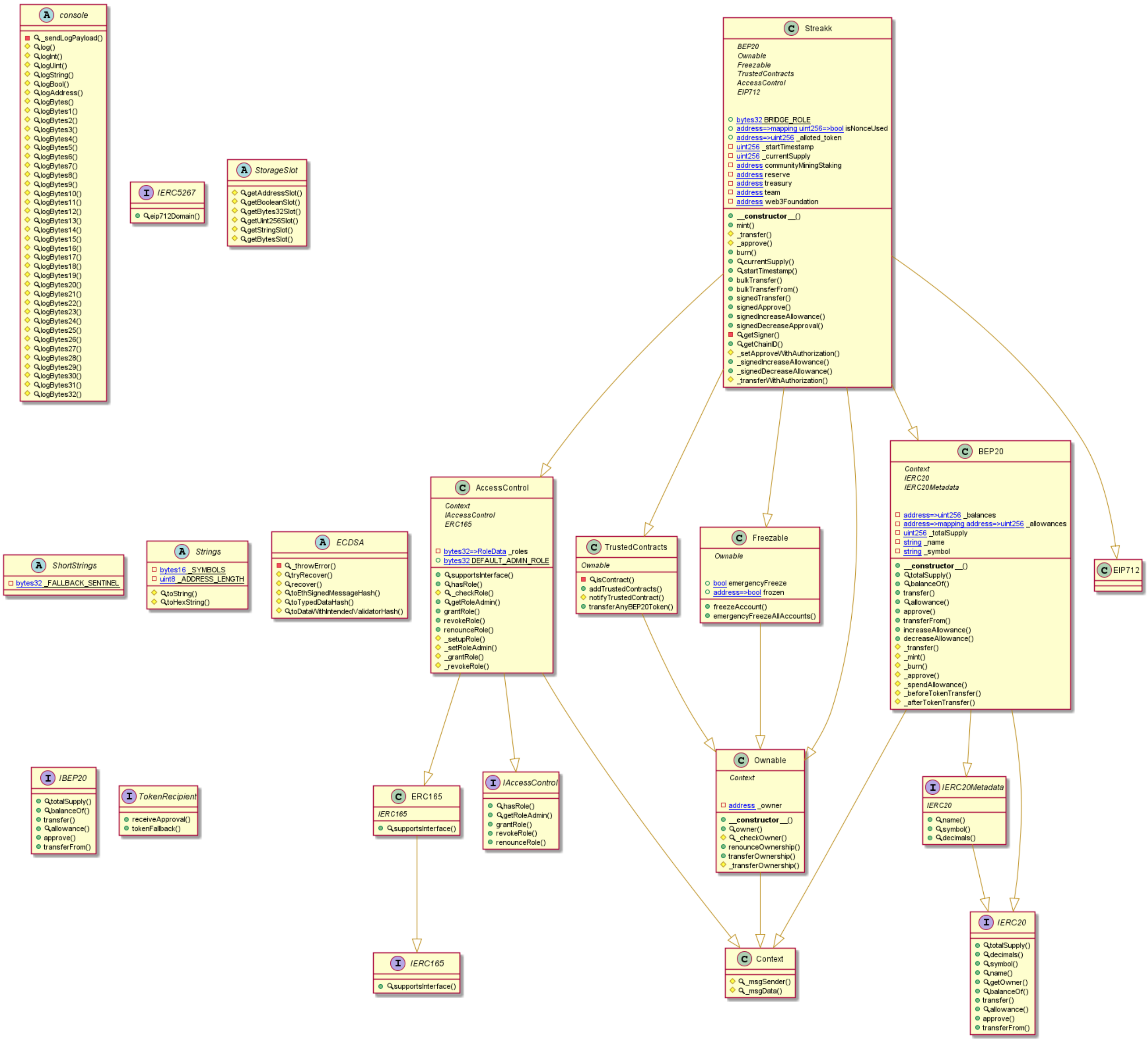
Technical Disclaimer

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.

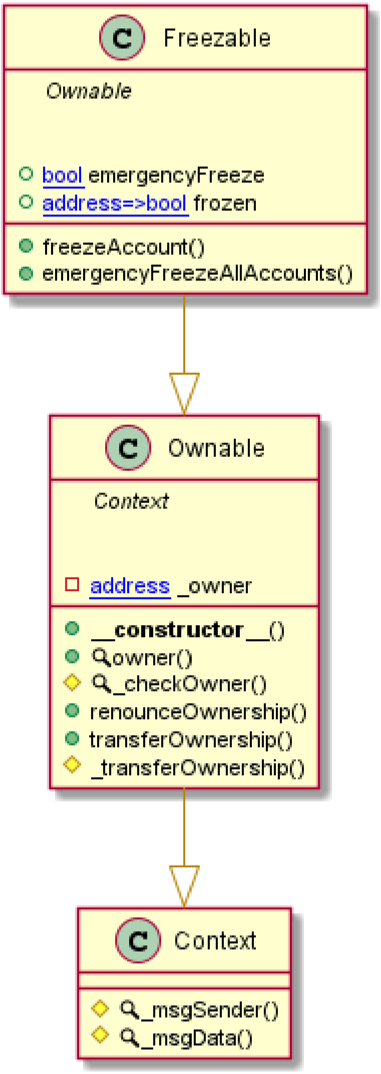
Appendix

Code Flow Diagram - Streakk Chain

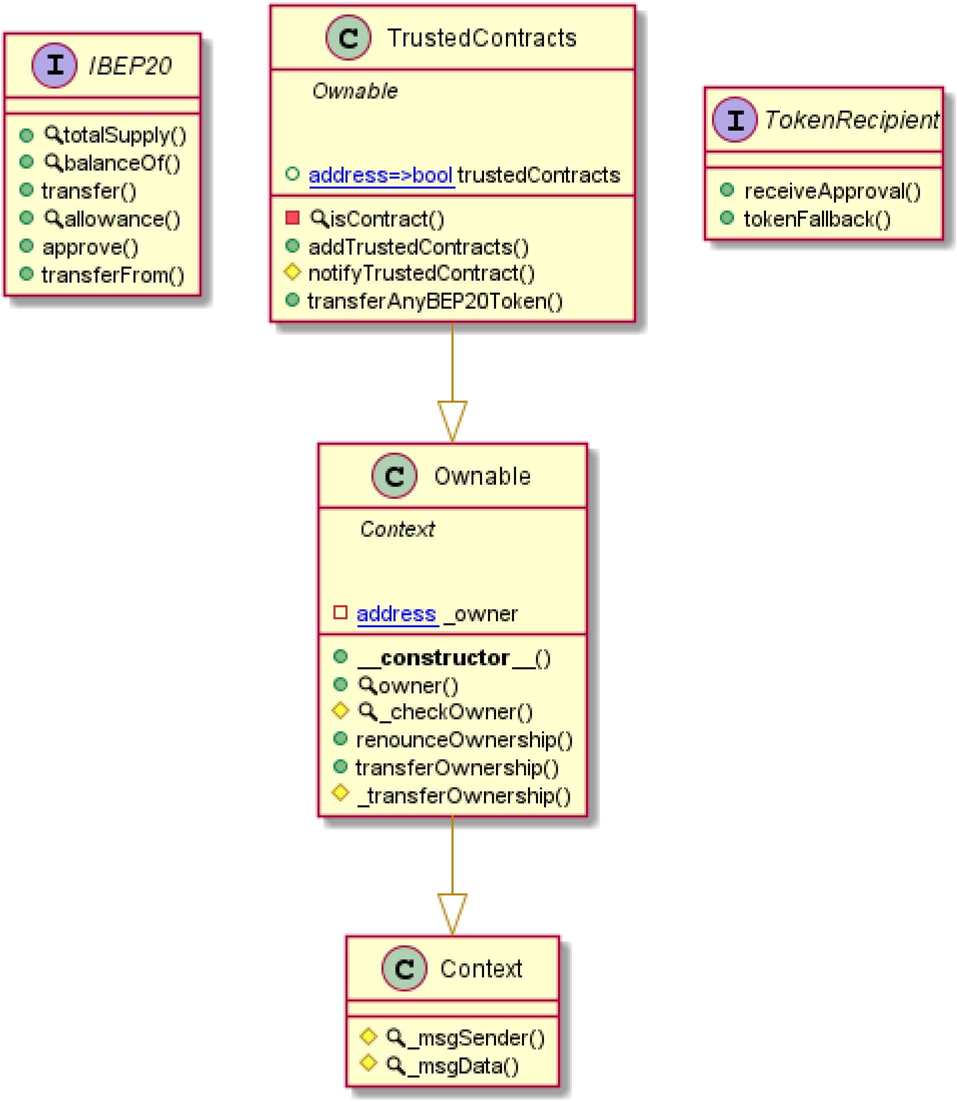
Streakk Diagram



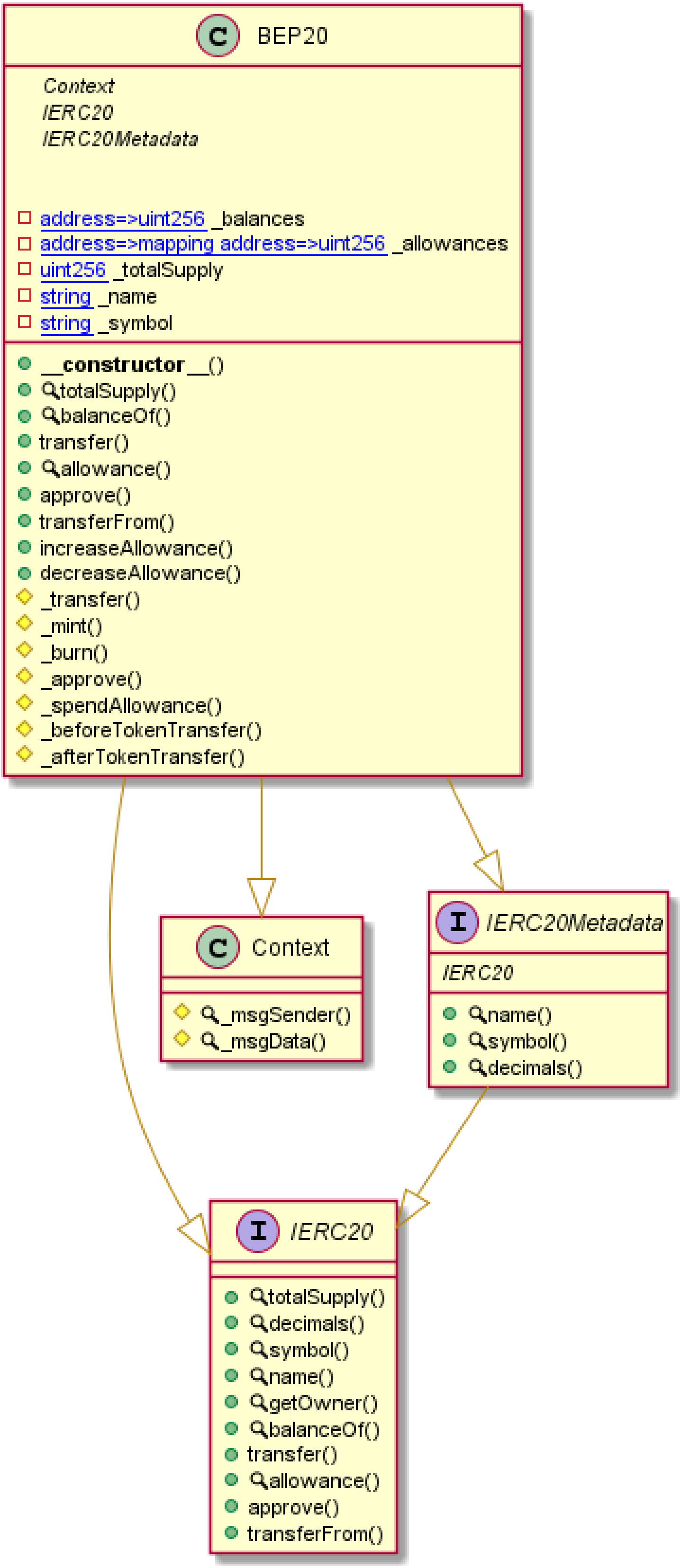
Freezable Diagram



TrustedContracts Diagram

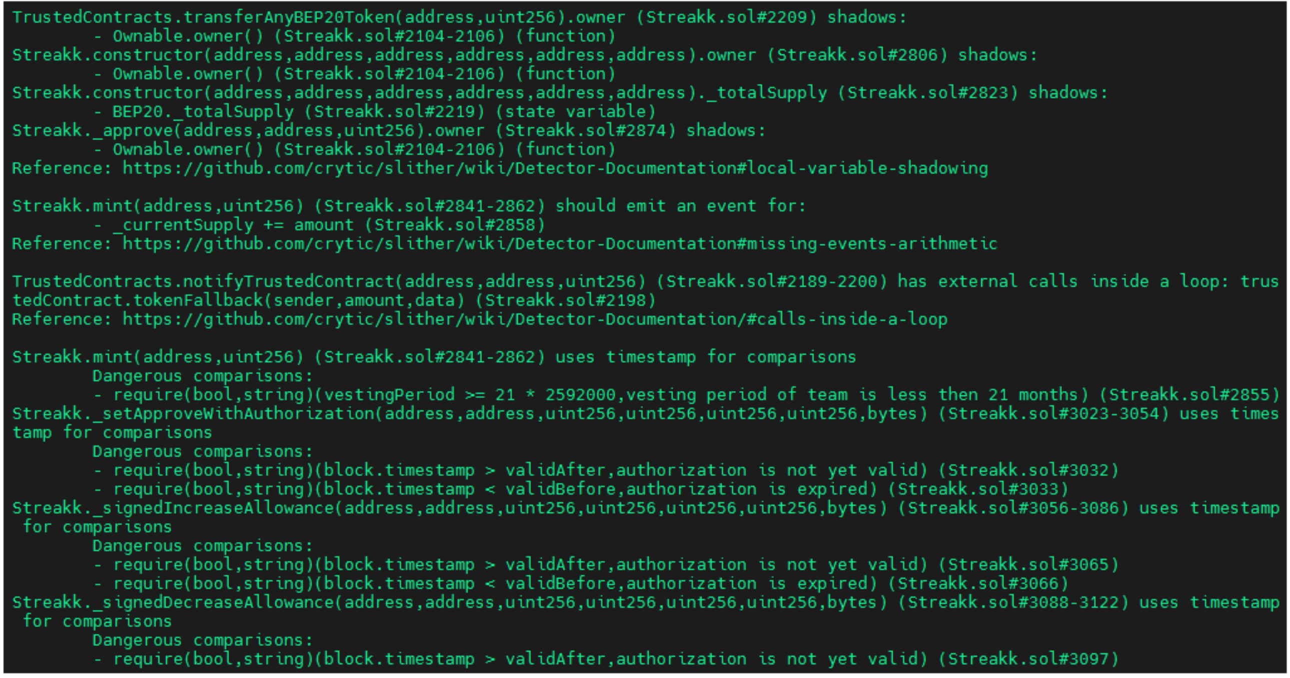


BEP20 Diagram

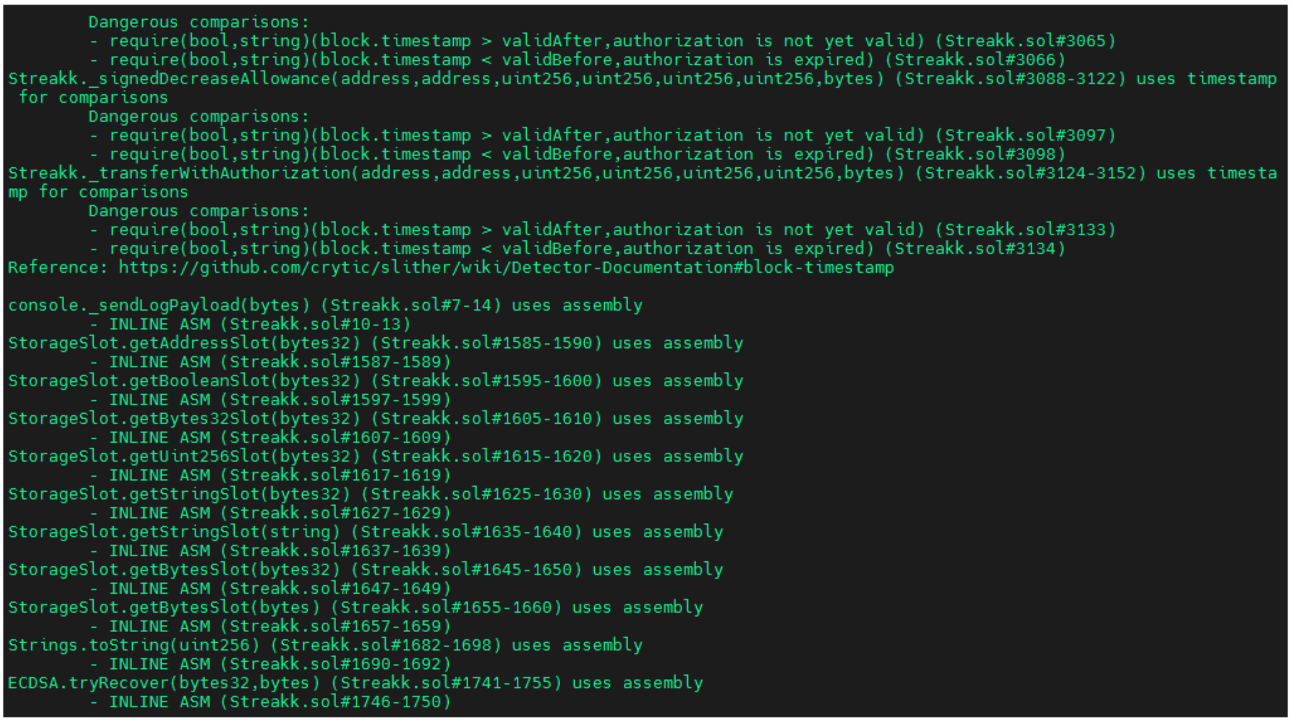


Slither Results Log

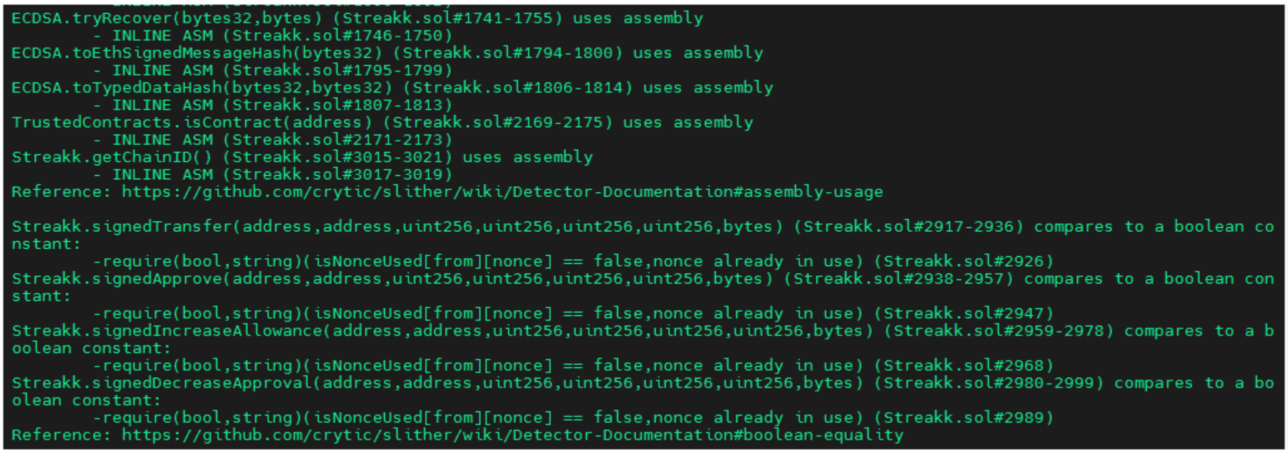
Slither log >> Streakk.sol



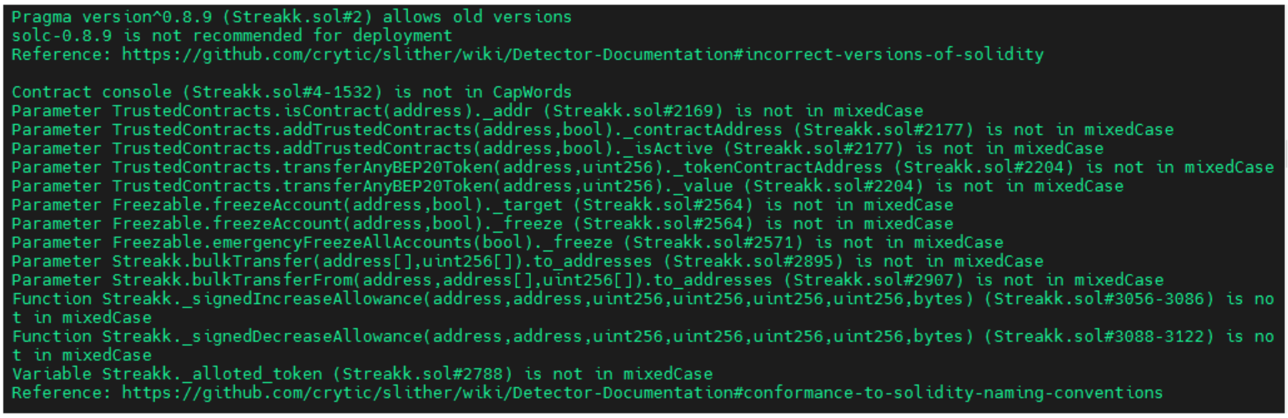
```
TrustedContracts.transferAnyBEP20Token(address,uint256).owner (Streakk.sol#2209) shadows:
        - Ownable.owner() (Streakk.sol#2104-2106) (function)
Streakk.constructor(address,address,address,address,address,address).owner (Streakk.sol#2806) shadows:
        - Ownable.owner() (Streakk.sol#2104-2106) (function)
Streakk.constructor(address,address,address,address,address,address)._totalSupply (Streakk.sol#2823) shadows:
        - BEP20._totalSupply (Streakk.sol#2219) (state variable)
Streakk._approve(address,address,uint256).owner (Streakk.sol#2874) shadows:
        - Ownable.owner() (Streakk.sol#2104-2106) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Streakk.mint(address,uint256) (Streakk.sol#2841-2862) should emit an event for:
        - _currentSupply += amount (Streakk.sol#2858)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

TrustedContracts.notifyTrustedContract(address,address,uint256) (Streakk.sol#2189-2200) has external calls inside a loop: trus
tedContract.tokenFallback(sender,amount,data) (Streakk.sol#2198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Streakk.mint(address,uint256) (Streakk.sol#2841-2862) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(vestingPeriod >= 21 * 2592000,vesting period of team is less then 21 months) (Streakk.sol#2855)
Streakk._setApproveWithAuthorization(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#3023-3054) uses times
tamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > validAfter,authorization is not yet valid) (Streakk.sol#3032)
        - require(bool,string)(block.timestamp < validBefore,authorization is expired) (Streakk.sol#3033)
Streakk._signedIncreaseAllowance(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#3056-3086) uses timestamp
 for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > validAfter,authorization is not yet valid) (Streakk.sol#3065)
        - require(bool,string)(block.timestamp < validBefore,authorization is expired) (Streakk.sol#3066)
Streakk._signedDecreaseAllowance(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#3088-3122) uses timestamp
 for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > validAfter,authorization is not yet valid) (Streakk.sol#3097)
```



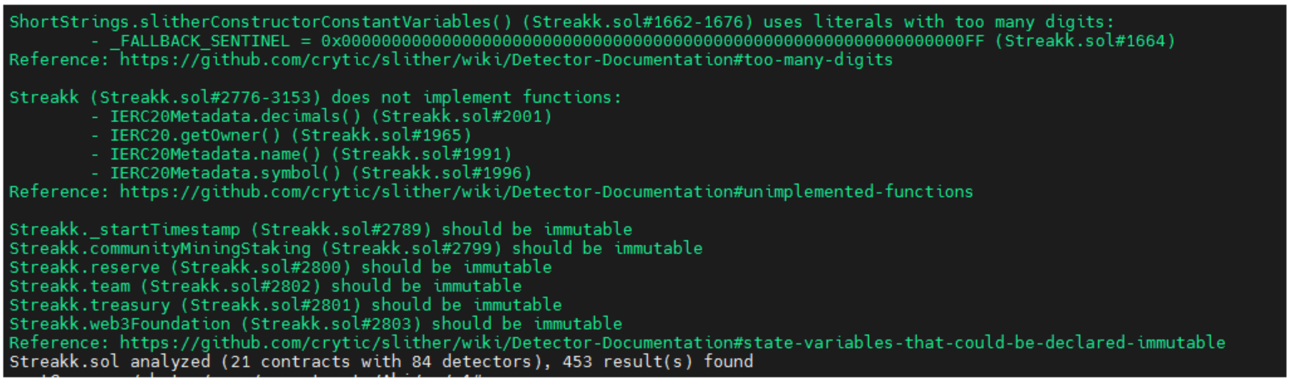
```
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > validAfter,authorization is not yet valid) (Streakk.sol#3065)
        - require(bool,string)(block.timestamp < validBefore,authorization is expired) (Streakk.sol#3066)
Streakk._signedDecreaseAllowance(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#3088-3122) uses timestamp
 for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > validAfter,authorization is not yet valid) (Streakk.sol#3097)
        - require(bool,string)(block.timestamp < validBefore,authorization is expired) (Streakk.sol#3098)
Streakk._transferWithAuthorization(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#3124-3152) uses timesta
mp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > validAfter,authorization is not yet valid) (Streakk.sol#3133)
        - require(bool,string)(block.timestamp < validBefore,authorization is expired) (Streakk.sol#3134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

console._sendLogPayload(bytes) (Streakk.sol#7-14) uses assembly
        - INLINE ASM (Streakk.sol#10-13)
StorageSlot.getAddressSlot(bytes32) (Streakk.sol#1585-1590) uses assembly
        - INLINE ASM (Streakk.sol#1587-1589)
StorageSlot.getBooleanSlot(bytes32) (Streakk.sol#1595-1600) uses assembly
        - INLINE ASM (Streakk.sol#1597-1599)
StorageSlot.getBytes32Slot(bytes32) (Streakk.sol#1605-1610) uses assembly
        - INLINE ASM (Streakk.sol#1607-1609)
StorageSlot.getUint256Slot(bytes32) (Streakk.sol#1615-1620) uses assembly
        - INLINE ASM (Streakk.sol#1617-1619)
StorageSlot.getStringSlot(bytes32) (Streakk.sol#1625-1630) uses assembly
        - INLINE ASM (Streakk.sol#1627-1629)
StorageSlot.getStringSlot(string) (Streakk.sol#1635-1640) uses assembly
        - INLINE ASM (Streakk.sol#1637-1639)
StorageSlot.getBytesSlot(bytes32) (Streakk.sol#1645-1650) uses assembly
        - INLINE ASM (Streakk.sol#1647-1649)
StorageSlot.getBytesSlot(bytes) (Streakk.sol#1655-1660) uses assembly
        - INLINE ASM (Streakk.sol#1657-1659)
Strings.toString(uint256) (Streakk.sol#1682-1698) uses assembly
        - INLINE ASM (Streakk.sol#1690-1692)
ECDSA.tryRecover(bytes32,bytes) (Streakk.sol#1741-1755) uses assembly
        - INLINE ASM (Streakk.sol#1746-1750)
```



```
ECDSA.tryRecover(bytes32,bytes) (Streakk.sol#1741-1755) uses assembly
        - INLINE ASM (Streakk.sol#1746-1750)
ECDSA.toEthSignedMessageHash(bytes32) (Streakk.sol#1794-1800) uses assembly
        - INLINE ASM (Streakk.sol#1795-1799)
ECDSA.toTypedDataHash(bytes32,bytes32) (Streakk.sol#1806-1814) uses assembly
        - INLINE ASM (Streakk.sol#1807-1813)
TrustedContracts.isContract(address) (Streakk.sol#2169-2175) uses assembly
        - INLINE ASM (Streakk.sol#2171-2173)
Streakk.getChainID() (Streakk.sol#3015-3021) uses assembly
        - INLINE ASM (Streakk.sol#3017-3019)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Streakk.signedTransfer(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#2917-2936) compares to a boolean co
nstant:
        -require(bool,string)(isNonceUsed[from][nonce] == false,nonce already in use) (Streakk.sol#2926)
Streakk.signedApprove(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#2938-2957) compares to a boolean con
stant:
        -require(bool,string)(isNonceUsed[from][nonce] == false,nonce already in use) (Streakk.sol#2947)
Streakk.signedIncreaseAllowance(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#2959-2978) compares to a b
oolean constant:
        -require(bool,string)(isNonceUsed[from][nonce] == false,nonce already in use) (Streakk.sol#2968)
Streakk.signedDecreaseApproval(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#2980-2999) compares to a bo
olean constant:
        -require(bool,string)(isNonceUsed[from][nonce] == false,nonce already in use) (Streakk.sol#2989)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

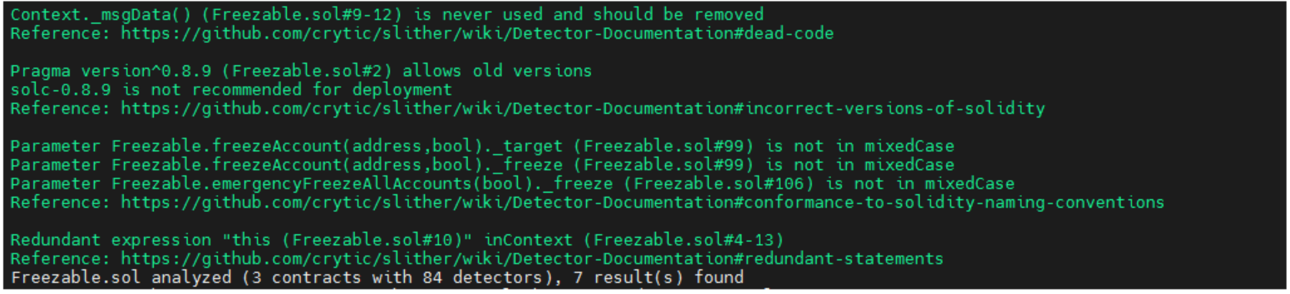


```
Pragma version^0.8.9 (Streakk.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Contract console (Streakk.sol#4-1532) is not in CapWords
Parameter TrustedContracts.isContract(address)._addr (Streakk.sol#2169) is not in mixedCase
Parameter TrustedContracts.addTrustedContracts(address,bool)._contractAddress (Streakk.sol#2177) is not in mixedCase
Parameter TrustedContracts.addTrustedContracts(address,bool)._isActive (Streakk.sol#2177) is not in mixedCase
Parameter TrustedContracts.transferAnyBEP20Token(address,uint256)._tokenContractAddress (Streakk.sol#2204) is not in mixedCase
Parameter TrustedContracts.transferAnyBEP20Token(address,uint256)._value (Streakk.sol#2204) is not in mixedCase
Parameter Freezable.freezeAccount(address,bool)._target (Streakk.sol#2564) is not in mixedCase
Parameter Freezable.freezeAccount(address,bool)._freeze (Streakk.sol#2564) is not in mixedCase
Parameter Freezable.emergencyFreezeAllAccounts(bool)._freeze (Streakk.sol#2571) is not in mixedCase
Parameter Streakk.bulkTransfer(address[],uint256[]).to_addresses (Streakk.sol#2895) is not in mixedCase
Parameter Streakk.bulkTransferFrom(address,address[],uint256[]).to_addresses (Streakk.sol#2907) is not in mixedCase
Function Streakk._signedIncreaseAllowance(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#3056-3086) is no
t in mixedCase
Function Streakk._signedDecreaseAllowance(address,address,uint256,uint256,uint256,uint256,bytes) (Streakk.sol#3088-3122) is no
t in mixedCase
Variable Streakk._alloted_token (Streakk.sol#2788) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```



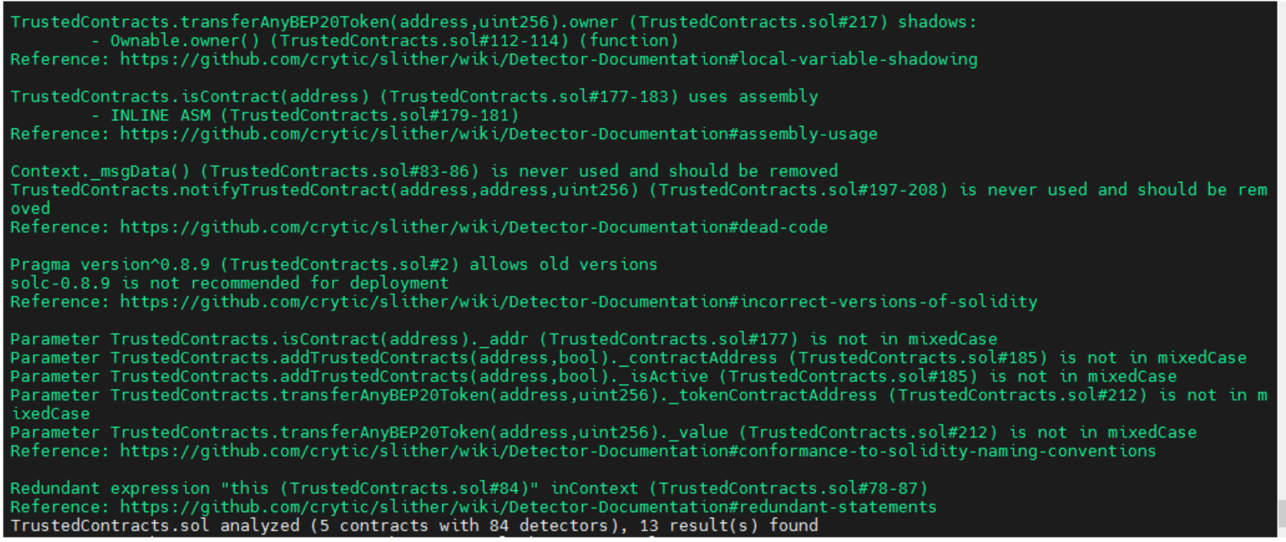
```
ShortStrings.slitherConstructorConstantVariables() (Streakk.sol#1662-1676) uses literals with too many digits:
        - _FALLBACK_SENTINEL = 0x00000000000000000000000000000000000000000000000000000000000000FF (Streakk.sol#1664)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Streakk (Streakk.sol#2776-3153) does not implement functions:
        - IERC20Metadata.decimals() (Streakk.sol#2001)
        - IERC20.getOwner() (Streakk.sol#1965)
        - IERC20Metadata.name() (Streakk.sol#1991)
        - IERC20Metadata.symbol() (Streakk.sol#1996)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

Streakk._startTimestamp (Streakk.sol#2789) should be immutable
Streakk.communityMiningStaking (Streakk.sol#2799) should be immutable
Streakk.reserve (Streakk.sol#2800) should be immutable
Streakk.team (Streakk.sol#2802) should be immutable
Streakk.treasury (Streakk.sol#2801) should be immutable
Streakk.web3Foundation (Streakk.sol#2803) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
Streakk.sol analyzed (21 contracts with 84 detectors), 453 result(s) found
```

Slither log >> Freezable.sol



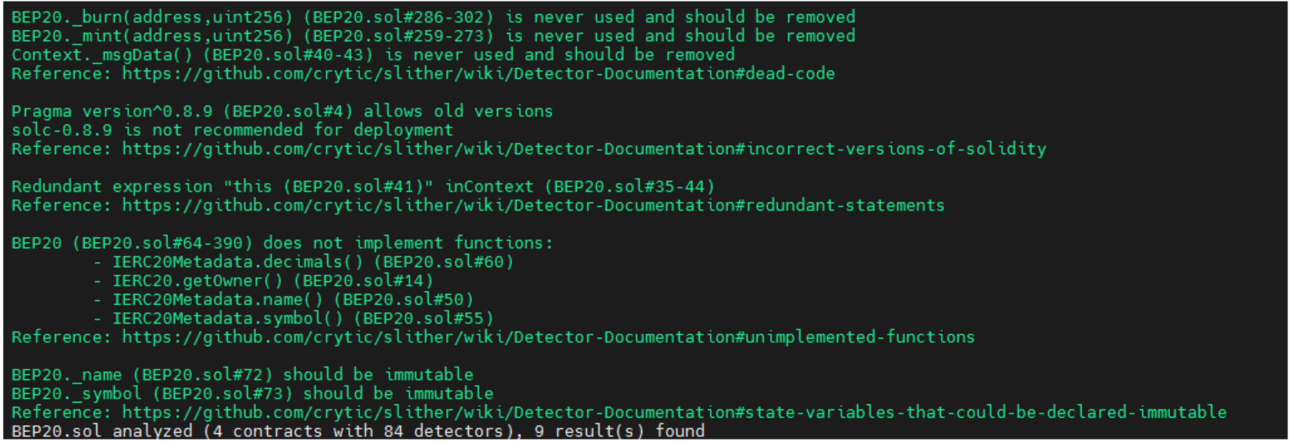
```
Context._msgData() (Freezable.sol#9-12) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.9 (Freezable.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter Freezable.freezeAccount(address,bool)._target (Freezable.sol#99) is not in mixedCase
Parameter Freezable.freezeAccount(address,bool)._freeze (Freezable.sol#99) is not in mixedCase
Parameter Freezable.emergencyFreezeAllAccounts(bool)._freeze (Freezable.sol#106) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (Freezable.sol#10)" inContext (Freezable.sol#4-13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
Freezable.sol analyzed (3 contracts with 84 detectors), 7 result(s) found
```

Slither log >> TrustedContracts.sol



```
TrustedContracts.transferAnyBEP20Token(address,uint256).owner (TrustedContracts.sol#217) shadows:
        - Ownable.owner() (TrustedContracts.sol#112-114) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

TrustedContracts.isContract(address) (TrustedContracts.sol#177-183) uses assembly
        - INLINE ASM (TrustedContracts.sol#179-181)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Context._msgData() (TrustedContracts.sol#83-86) is never used and should be removed
TrustedContracts.notifyTrustedContract(address,address,uint256) (TrustedContracts.sol#197-208) is never used and should be rem
oved
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.9 (TrustedContracts.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter TrustedContracts.isContract(address)._addr (TrustedContracts.sol#177) is not in mixedCase
Parameter TrustedContracts.addTrustedContracts(address,bool)._contractAddress (TrustedContracts.sol#185) is not in mixedCase
Parameter TrustedContracts.addTrustedContracts(address,bool)._isActive (TrustedContracts.sol#185) is not in mixedCase
Parameter TrustedContracts.transferAnyBEP20Token(address,uint256)._tokenContractAddress (TrustedContracts.sol#212) is not in m
ixedCase
Parameter TrustedContracts.transferAnyBEP20Token(address,uint256)._value (TrustedContracts.sol#212) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (TrustedContracts.sol#84)" inContext (TrustedContracts.sol#78-87)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
TrustedContracts.sol analyzed (5 contracts with 84 detectors), 13 result(s) found
```

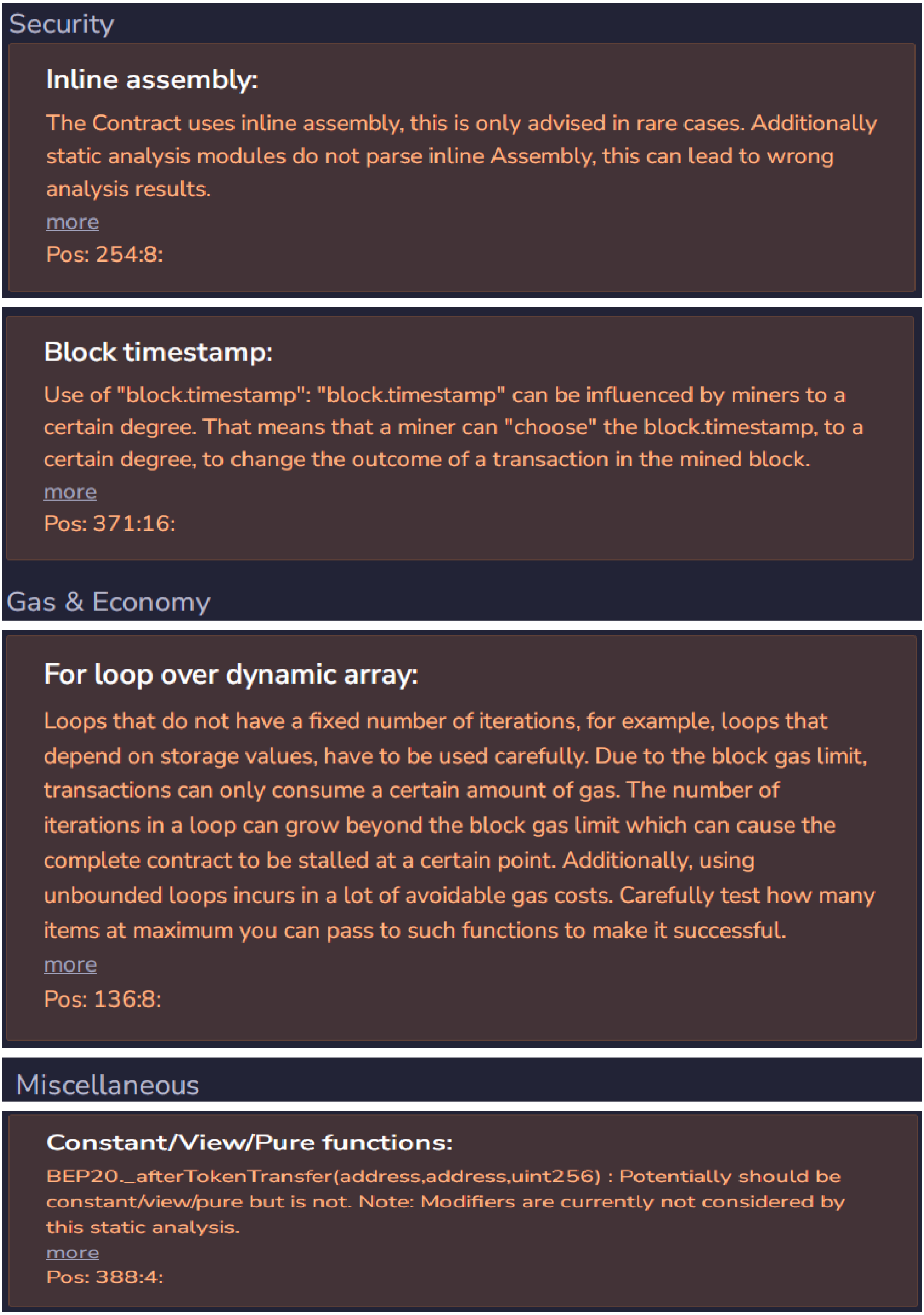
Slither log >> BEP20.sol



```
BEP20._burn(address,uint256) (BEP20.sol#286-302) is never used and should be removed
BEP20._mint(address,uint256) (BEP20.sol#259-273) is never used and should be removed
Context._msgData() (BEP20.sol#40-43) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.9 (BEP20.sol#4) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Redundant expression "this (BEP20.sol#41)" inContext (BEP20.sol#35-44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

BEP20 (BEP20.sol#64-390) does not implement functions:
        - IERC20Metadata.decimals() (BEP20.sol#60)
        - IERC20.getOwner() (BEP20.sol#14)
        - IERC20Metadata.name() (BEP20.sol#50)
        - IERC20Metadata.symbol() (BEP20.sol#55)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

BEP20._name (BEP20.sol#72) should be immutable
BEP20._symbol (BEP20.sol#73) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
BEP20.sol analyzed (4 contracts with 84 detectors), 9 result(s) found
```

Solidity Static Analysis

Streakk.sol



Security

Inline assembly:

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

more

Pos: 254:8:

Block timestamp:

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

more

Pos: 371:16:

Gas & Economy

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: 136:8:

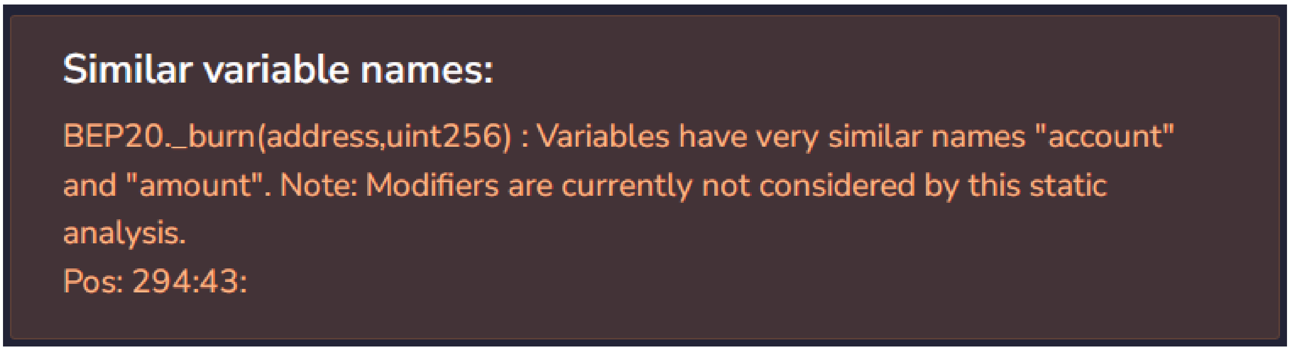
Miscellaneous

Constant/View/Pure functions:

BEP20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

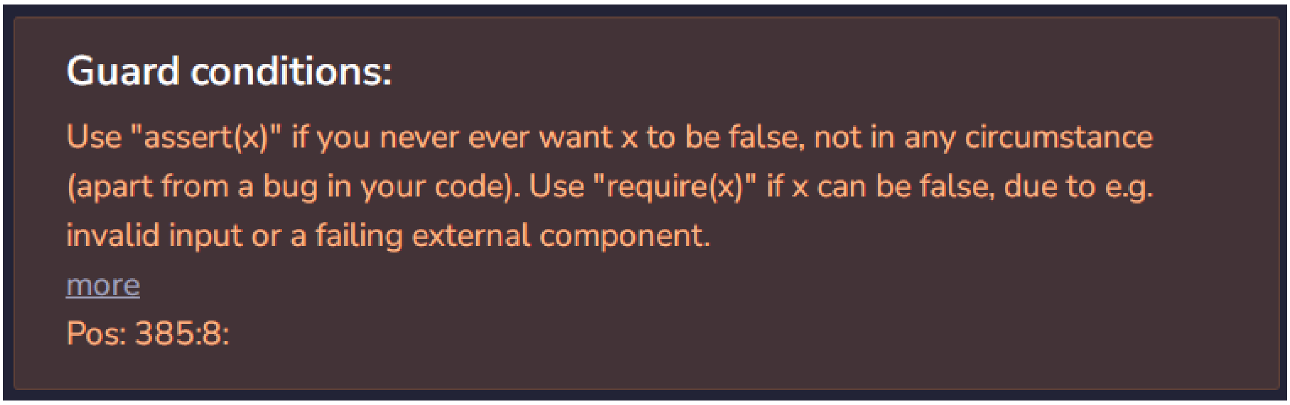
more

Pos: 388:4:



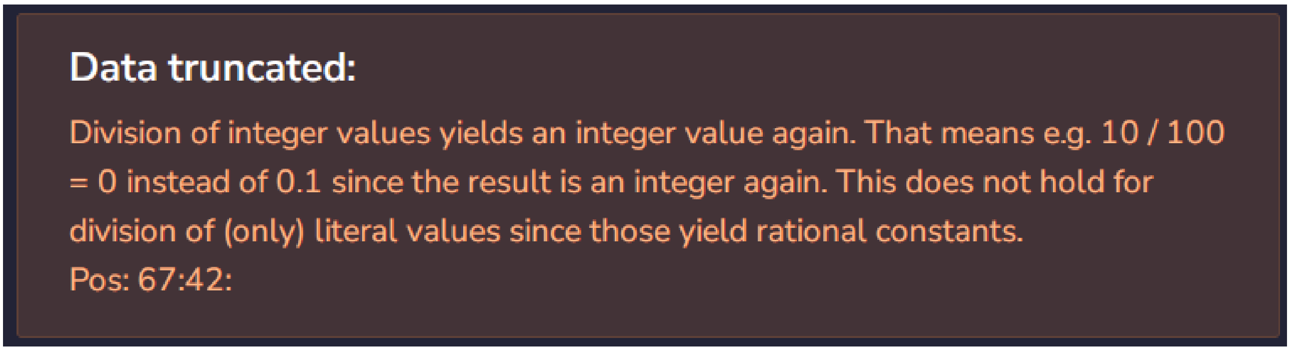
Similar variable names:

BEP20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 294:43:



Guard conditions:

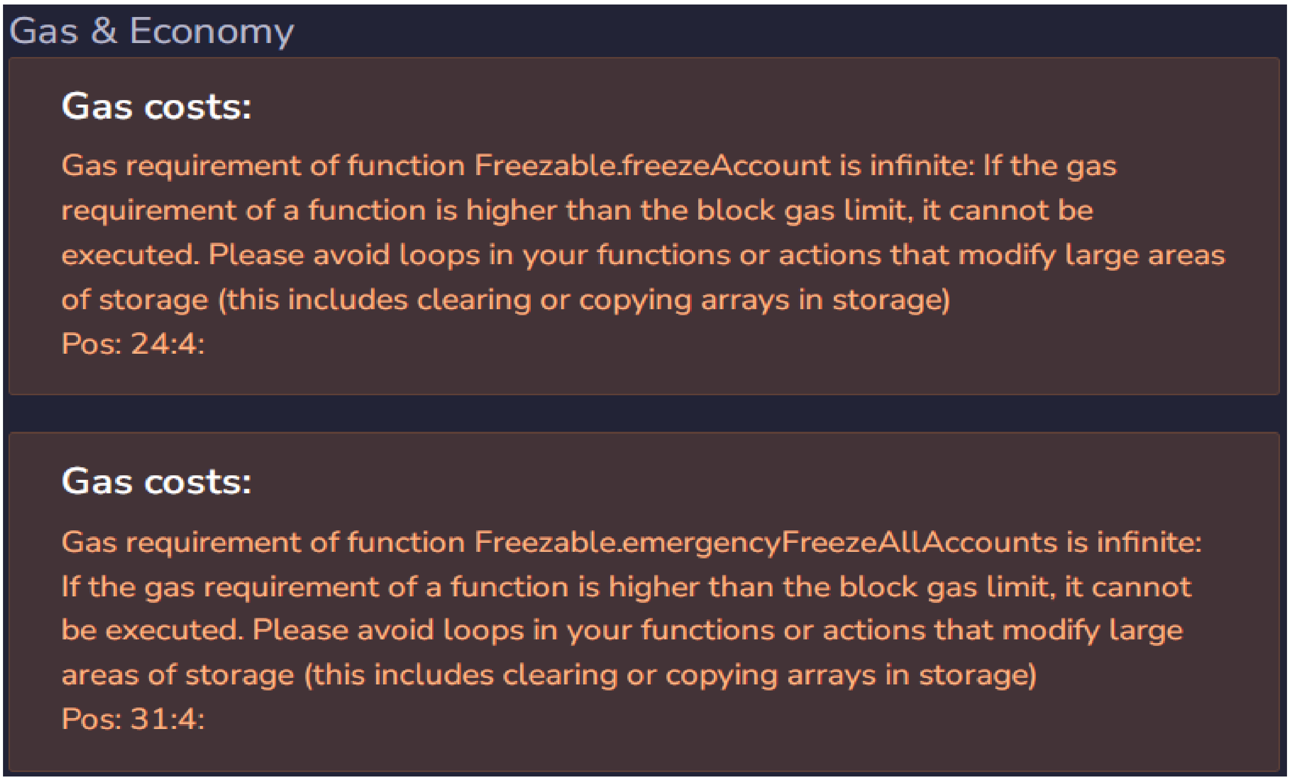
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 385:8:



Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 67:42:

Freezable.sol



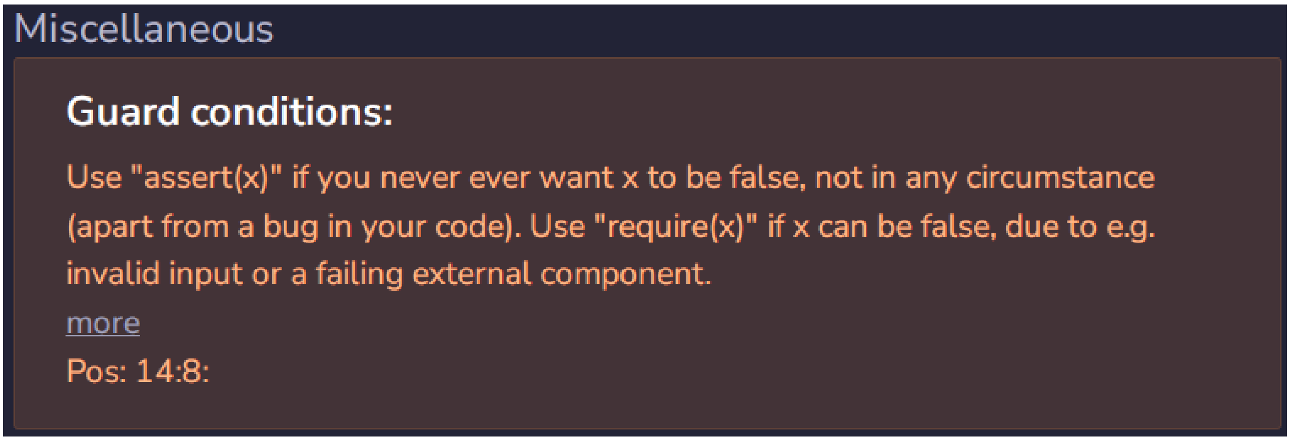
Gas & Economy

Gas costs:

Gas requirement of function Freezable.freezeAccount 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: 24:4:

Gas costs:

Gas requirement of function Freezable.emergencyFreezeAllAccounts 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: 31:4:



Miscellaneous

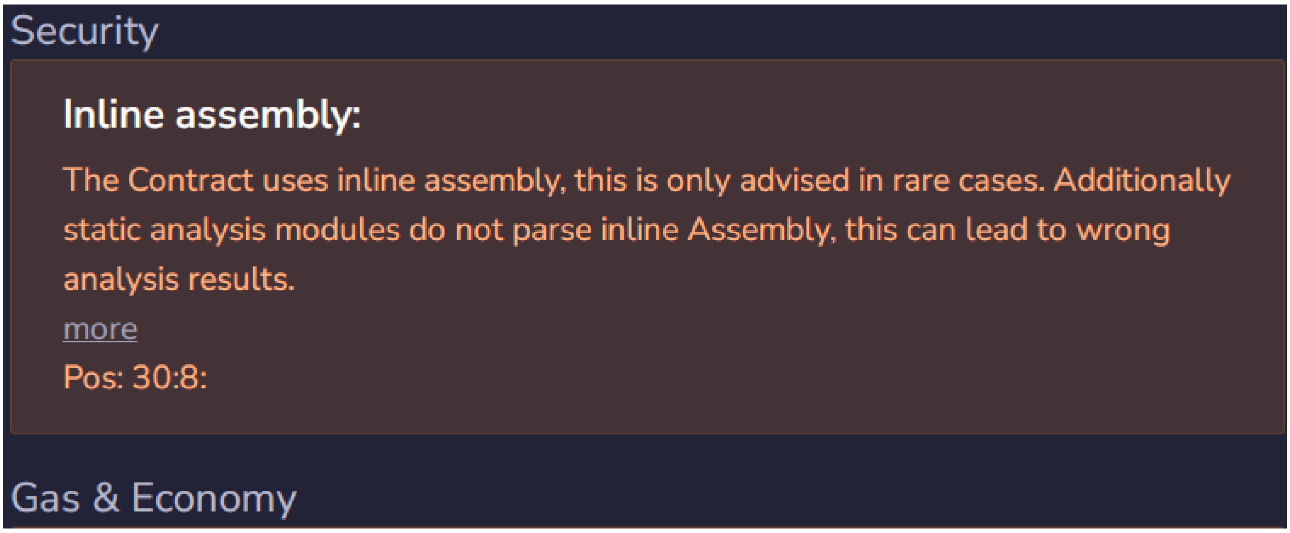
Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 14:8:

TrustedContracts.sol



Security

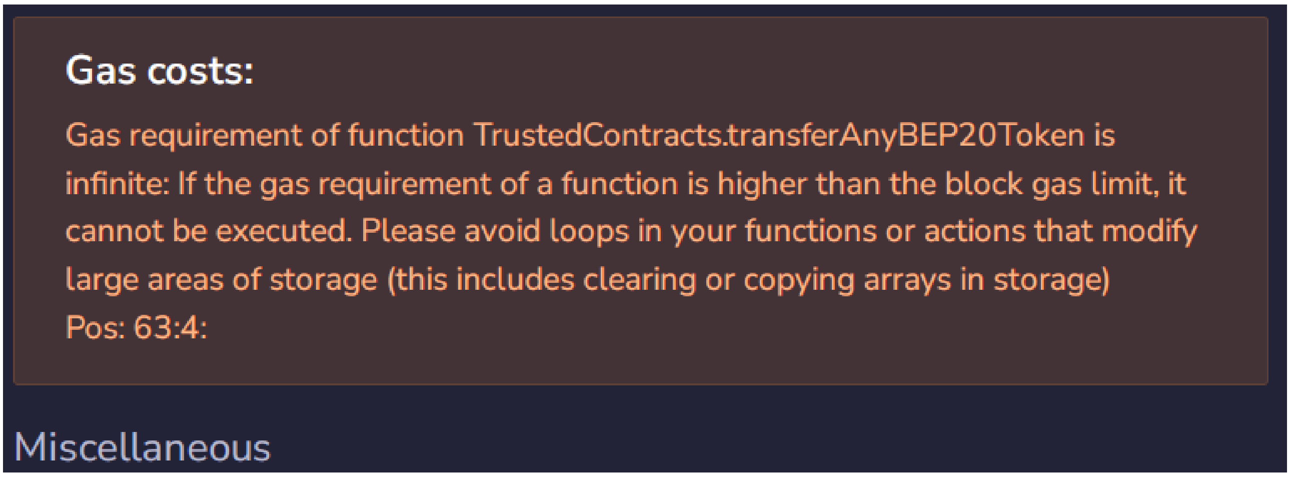
Inline assembly:

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

more

Pos: 30:8:

Gas & Economy

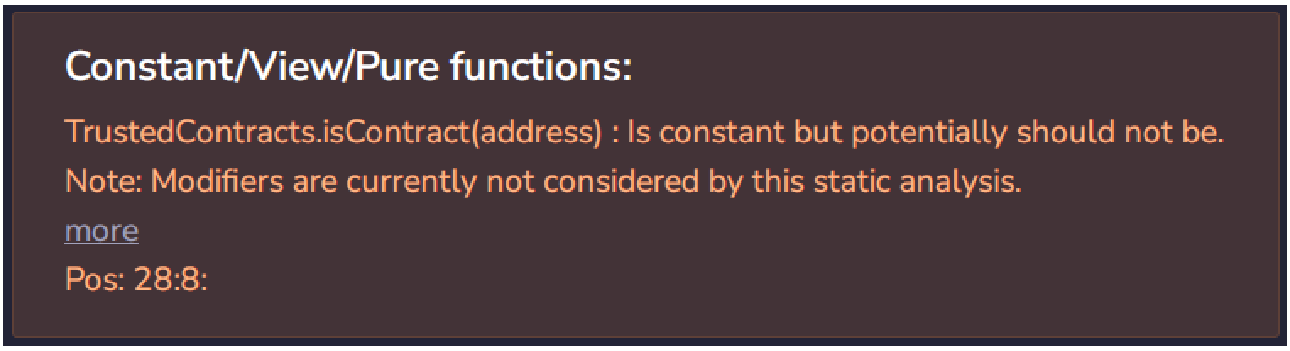


Gas costs:

Gas requirement of function TrustedContracts.transferAnyBEP20Token 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: 63:4:

Miscellaneous



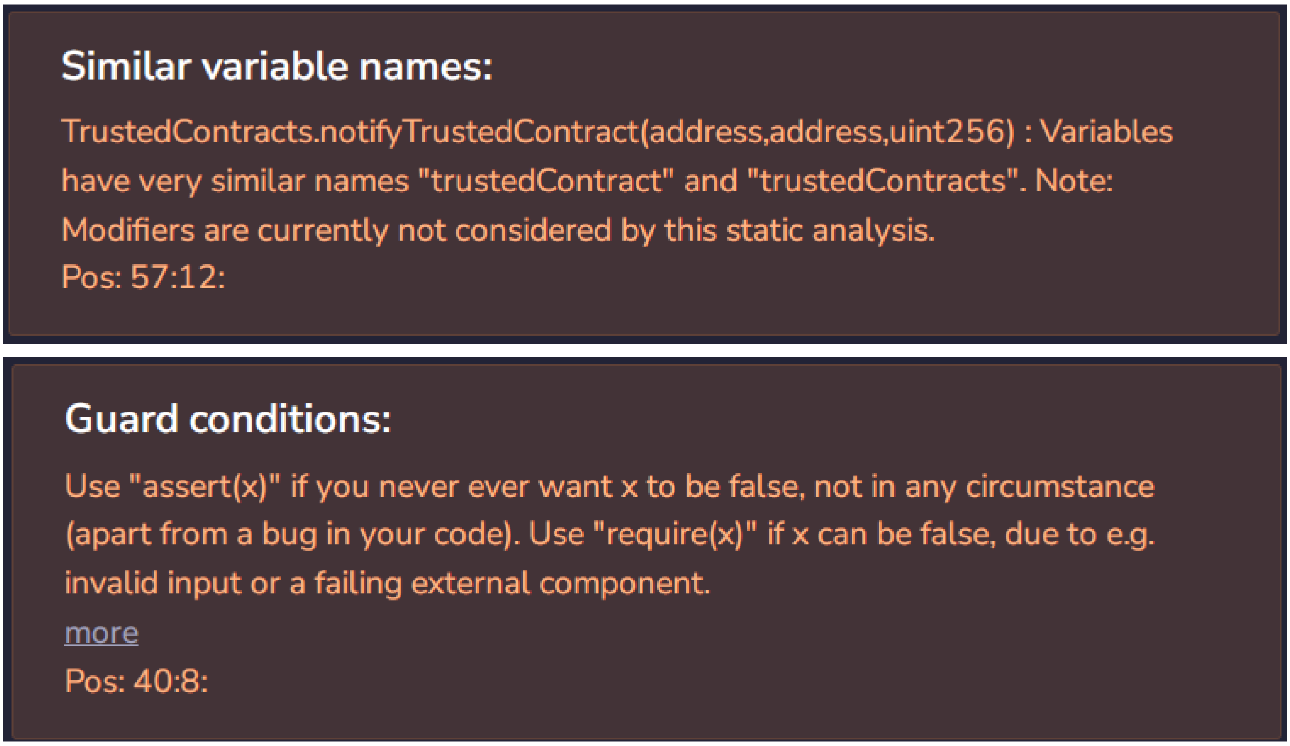
Constant/View/Pure functions:

TrustedContracts.isContract(address) : Is constant but potentially should not be.

Note: Modifiers are currently not considered by this static analysis.

more

Pos: 28:8:



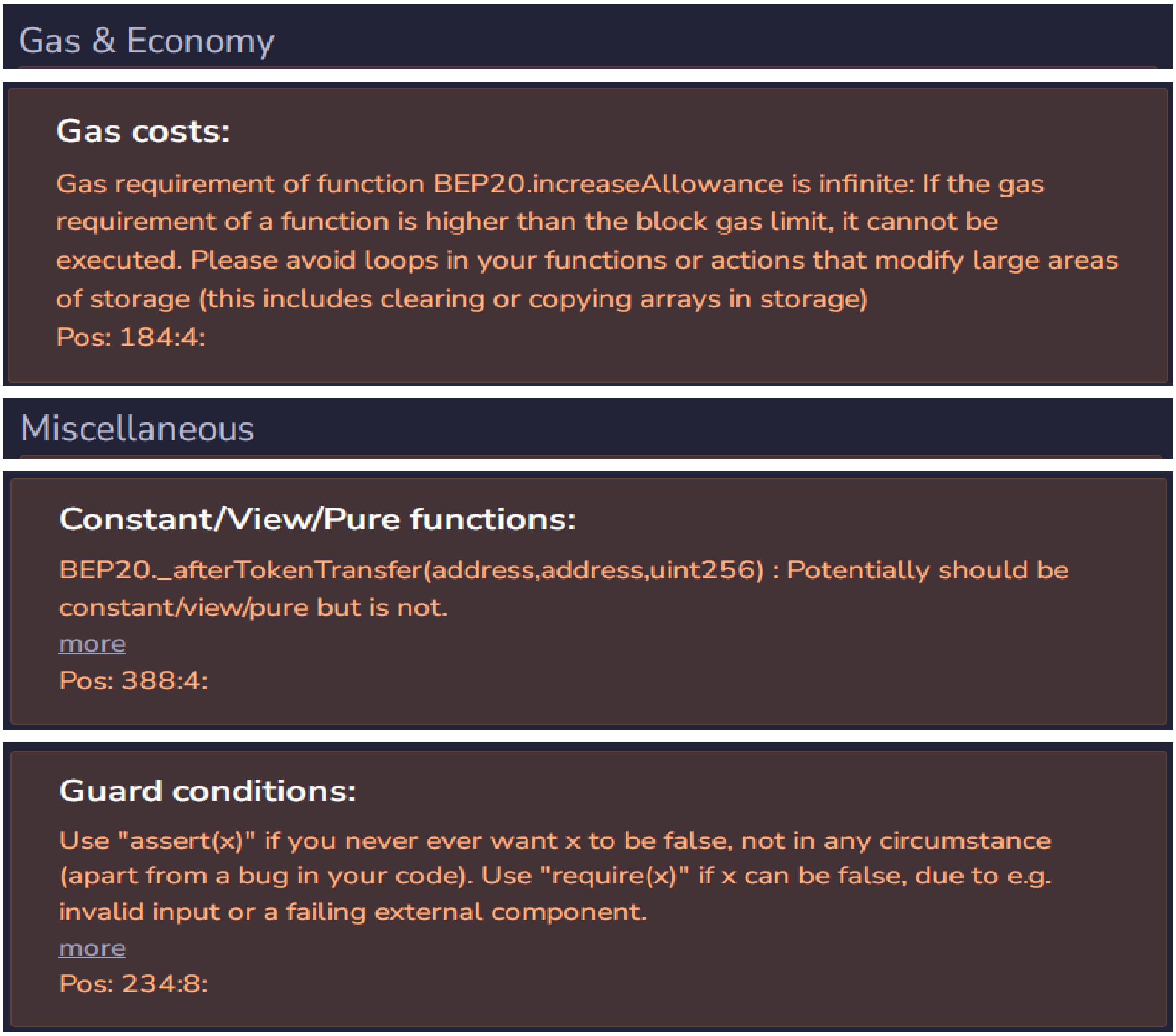
Similar variable names:

TrustedContracts.notifyTrustedContract(address,address,uint256) : Variables have very similar names "trustedContract" and "trustedContracts". Note: Modifiers are currently not considered by this static analysis.
Pos: 57:12:

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: 40:8:

BEP20.sol



Gas & Economy

Gas costs:

Gas requirement of function BEP20.increaseAllowance 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: 184:4:

Miscellaneous

Constant/View/Pure functions:

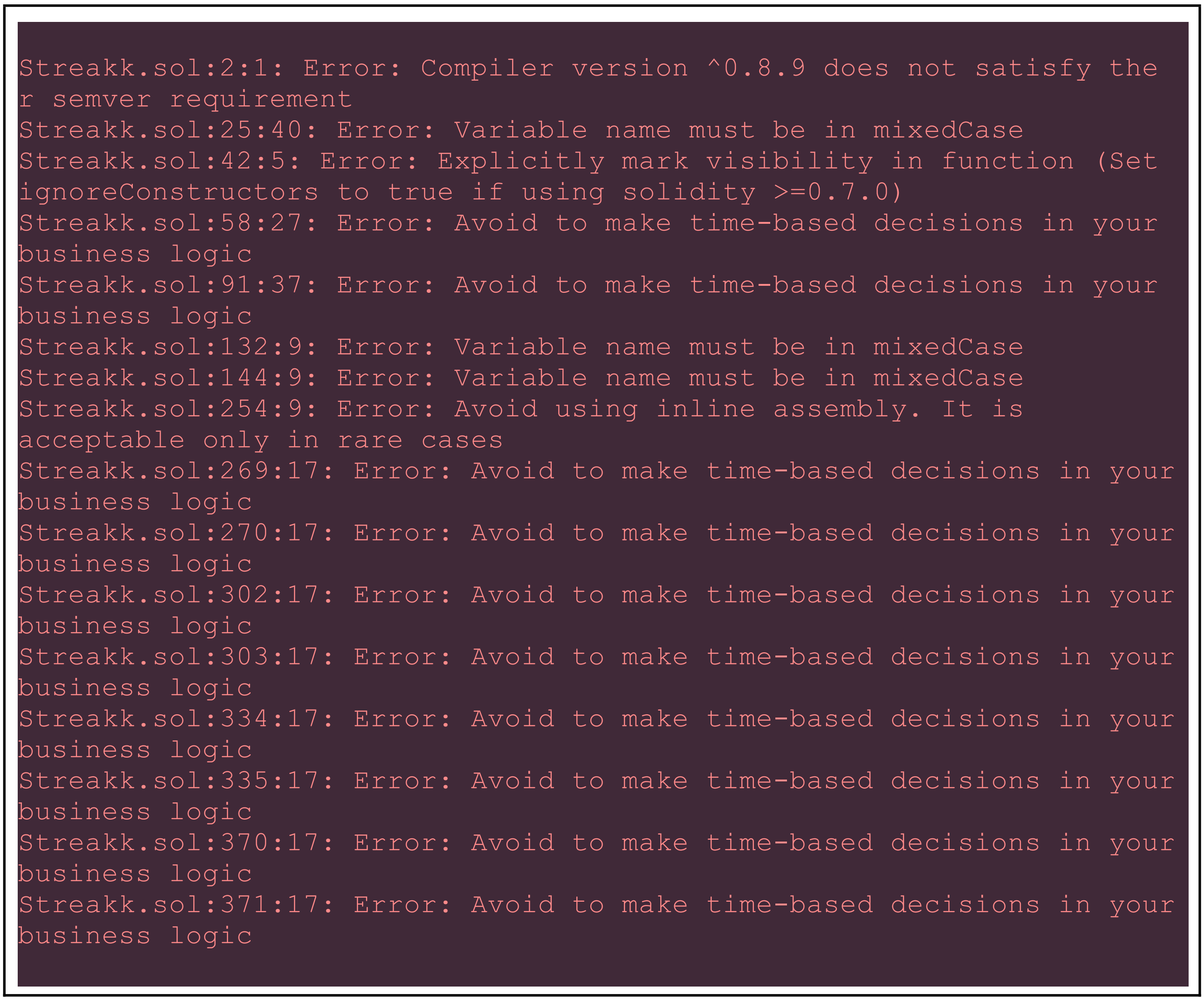
BEP20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not.
more
Pos: 388:4:

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: 234:8:

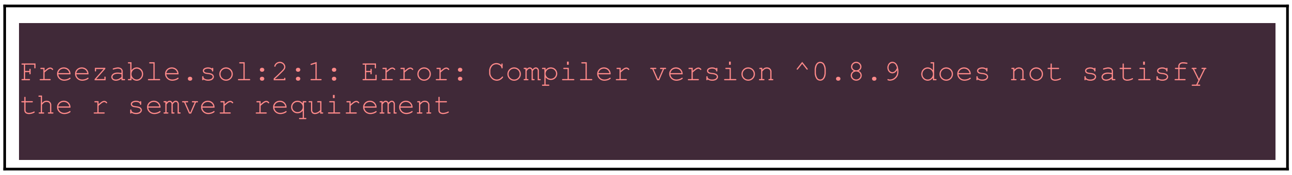
Solhint Linter

Streakk.sol



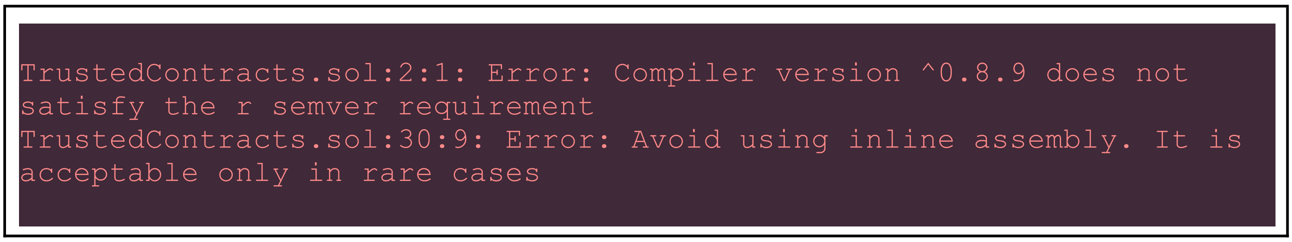
```
Streakk.sol:2:1: Error: Compiler version ^0.8.9 does not satisfy the
r semver requirement
Streakk.sol:25:40: Error: Variable name must be in mixedCase
Streakk.sol:42:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Streakk.sol:58:27: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:91:37: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:132:9: Error: Variable name must be in mixedCase
Streakk.sol:144:9: Error: Variable name must be in mixedCase
Streakk.sol:254:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
Streakk.sol:269:17: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:270:17: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:302:17: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:303:17: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:334:17: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:335:17: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:370:17: Error: Avoid to make time-based decisions in your
business logic
Streakk.sol:371:17: Error: Avoid to make time-based decisions in your
business logic
```

Freezable.sol



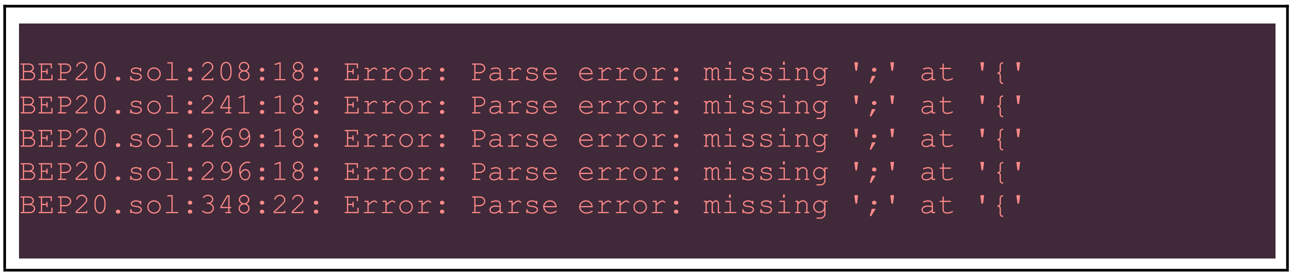
```
Freezable.sol:2:1: Error: Compiler version ^0.8.9 does not satisfy
the r semver requirement
```

TrustedContracts.sol



```
TrustedContracts.sol:2:1: Error: Compiler version ^0.8.9 does not
satisfy the r semver requirement
TrustedContracts.sol:30:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
```

BEP20.sol



```
BEP20.sol:208:18: Error: Parse error: missing ';' at '{'
BEP20.sol:241:18: Error: Parse error: missing ';' at '{'
BEP20.sol:269:18: Error: Parse error: missing ';' at '{'
BEP20.sol:296:18: Error: Parse error: missing ';' at '{'
BEP20.sol:348:22: Error: Parse error: missing ';' at '{'
```

Software analysis result:

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