Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Wrapped BTC Token
Website:	wbtc.network
Platform:	Ethereum
Language:	Solidity
Date:	March 16th, 2024

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

As part of EtherAuthority’s community smart contracts audit initiatives, the Wrapped BTC token smart contract from wbtc.network was audited extensively. 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 March 16th, 2024.

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 BTC smart contract, when wrapped, inherits multiple contracts with varying functions:
    - **CanReclaimToken:** The contract is expected to be capable of recovering tokens.
    - **Claimable:** This enables the new owner to accept the transfer.
    - **BurnableToken:** A token that can be permanently destroyed.
    - **MintableToken:** The ERC20 token is a simple and mintable option for generating digital currency.
    - **PausableToken:** The StandardToken has been modified with the ability to make pausable transfers.
    - **StandardToken:** The implementation of the basic standard token.
    - **BasicToken:** The StandardToken is a basic version of the cryptocurrency, without any allowances.
- The Wrapped BTC Token is a standard smart contract that offers functions such as Minting, burning, finish minting, renouncing ownership, reclaim token, claim ownership, pause-unpause.

Audit scope

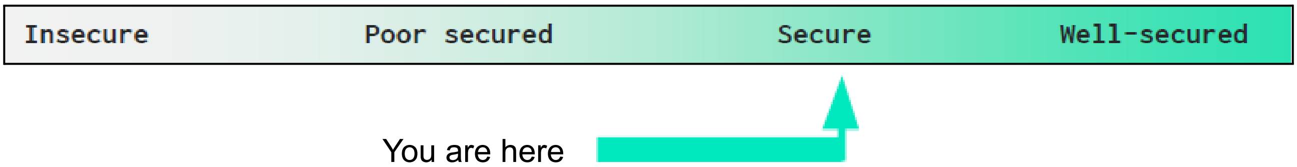
Name	**Code Review and Security Analysis Report for Wrapped BTC Token Smart Contract**
Platform	**Ethereum**
File	WBTC.sol
Smart Contract Code	0x2260fac5e5542a773aa44fbcfedf7c193bc2c599
Audit Date	March 16th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Wrapped BTC • Symbol: WBTC • Decimals: 8	**YES, This is valid.**
Owner control: • Burn tokens. • Finish minting. • Renounce ownership. • Reclaim all ERC20 Basic compatible tokens. • The current owner can transfer to a new owner's address. • Allows the pending Owner address to finalize the transfer. • Pause / Unpause contract state. • Mint tokens. • Stop minting new tokens.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**. Also, these contracts 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 a 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. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low and 4 very low level issues.

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

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	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	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	Moderated
	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

Business Risk Analysis

Category	Result
🟢 Buy Tax	0%
🟢 Sell Tax	0%
🟢 Cannot Buy	No
🟢 Cannot Sell	No
🟢 Max Tax	0%
🟢 Modify Tax	Not Detected
🟢 Fee Check	No
🟢 Is Honeypot	Not Detected
🟢 Trading Cooldown	Not Detected
🟢 Can Pause Trade?	No
🟢 Pause Transfer?	Yes
🟢 Max Tax?	No
🟢 Is it Anti-whale?	Not Detected
🟢 Is Anti-bot?	Not Detected
🟢 Is it a Blacklist?	Not Detected
🟢 Blacklist Check	No
🟢 Can Mint?	Yes
🟢 Is it a Proxy?	Not Detected
🟢 Can Take Ownership?	No
🟢 Hidden Owner?	Not Detected
🟢 Self Destruction?	Not Detected
🟢 Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

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

The libraries in Wrapped BTC 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 Wrapped BTC Token.

The EtherAuthority team has no 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 Wrapped BTC Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. 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 used in this smart contract infrastructure that is based on well-known industry-standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	burn	write	access only Owner	No Issue
3	finishMinting	write	Gas Optimization	Refer Audit Findings
4	renounceOwnership	write	Gas Optimization	Refer Audit Findings
5	reclaimToken	external	access only Owner	No Issue
6	onlyPendingOwner	modifier	Error message for require condition	Refer Audit Findings
7	transferOwnership	write	access only Owner	No Issue
8	claimOwnership	write	access only Pending Owner	No Issue
9	transfer	write	Passed	No Issue
10	transferFrom	write	Passed	No Issue
11	approve	write	Passed	No Issue
12	increaseApproval	write	Passed	No Issue
13	decreaseApproval	write	Passed	No Issue
14	whenNotPaused	modifier	Error message for require condition	Refer Audit Findings
15	whenPaused	modifier	Error message for require condition	Refer Audit Findings
16	pause	write	access only Owner	No Issue
17	unpause	write	access only Owner	No Issue
18	burn	write	Passed	No Issue
19	_burn	internal	Error message for require condition	Refer Audit Findings
20	canMint	modifier	Error message for require condition	Refer Audit Findings
21	hasMintPermission	modifier	Error message for require condition	Refer Audit Findings
22	mint	write	has Mint Permission	No Issue
23	finishMinting	write	access only Owner	No Issue
24	onlyOwner	modifier	Error message for require condition	Refer Audit Findings
25	renounceOwnership	write	access only Owner	No Issue
26	transferOwnership	write	Missing events access control	Refer Audit Findings
27	_transferOwnership	internal	Error message for require condition	Refer Audit Findings
28	transferFrom	write	Passed	No Issue
29	allowance	read	Passed	No Issue
30	transferFrom	write	Passed	No Issue
31	approve	write	Passed	No Issue

32	totalSupply	read	Passed	No Issue
33	transfer	write	Passed	No Issue
34	balanceOf	read	Passed	No Issue
35	totalSupply	read	Passed	No Issue
36	balanceOf	read	Passed	No Issue
37	transfer	write	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) Use the latest solidity version:

```
pragma solidity 0.4.24;
```

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use versions greater than 0.8.7.

(2) Error message for the required condition:

Detect missing error message in require statement in:

- onlyOwner
- _transferOwnership
- hasMintPermission
- canMint
- _burn
- onlyPendingOwner

- whenNotPaused
- whenPaused

Resolution: We suggest writing a proper message in the required condition, Otherwise user can't identify the actual problem of why the transaction was not successful.

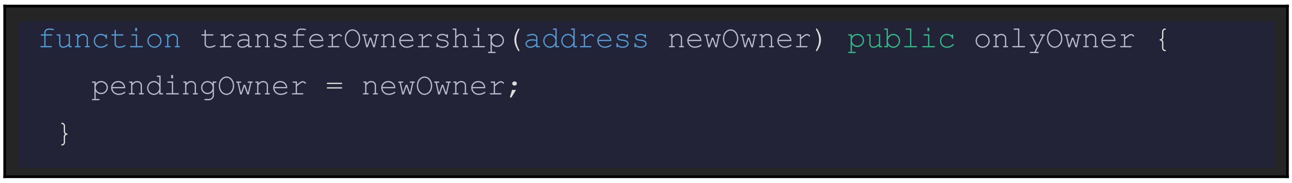
(3) Gas Optimization:

- renounceOwnership
- finishMinting

These are public functions without any code logic written and accessible only by the Owner.

Resolution: We suggest either removing both functions or writing an appropriate code logic to save the Gas.

(4) Missing events access control:



```
function transferOwnership(address newOwner) public onlyOwner {
   pendingOwner = newOwner;
 }
```

Detects missing events for critical access control parameters. transferownership() has no event, so it is difficult to track off-chain owner changes.

Resolution: We suggest emitting an event for critical parameter changes.

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:

WBTC.sol

- burn: burn tokens by the owner.
- finishMinting: Finish minting by the owner.
- renounceOwnership: Renounce ownership by the owner.

CanReclaimToken.sol

- reclaimToken: Reclaim all ERC20 Basic compatible tokens by the owner.

Claimable.sol

- transferOwnership: The current owner can transfer to a new owner's address.
- claimOwnership: Allows the pending Owner address to finalize the transfer.

Pausable.sol

- pause: A stopped state can be triggered by the owner.
- unpause: normal state can be returned by the owner.

MintableToken.sol

- mint: Mint tokens by the owner.
- finishMinting: Stop minting new tokens 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.

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

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 4 informational issues in the smart contracts. And those issues are not critical. 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 the 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 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 the 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 white box penetration testing. We look at the project's website 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).

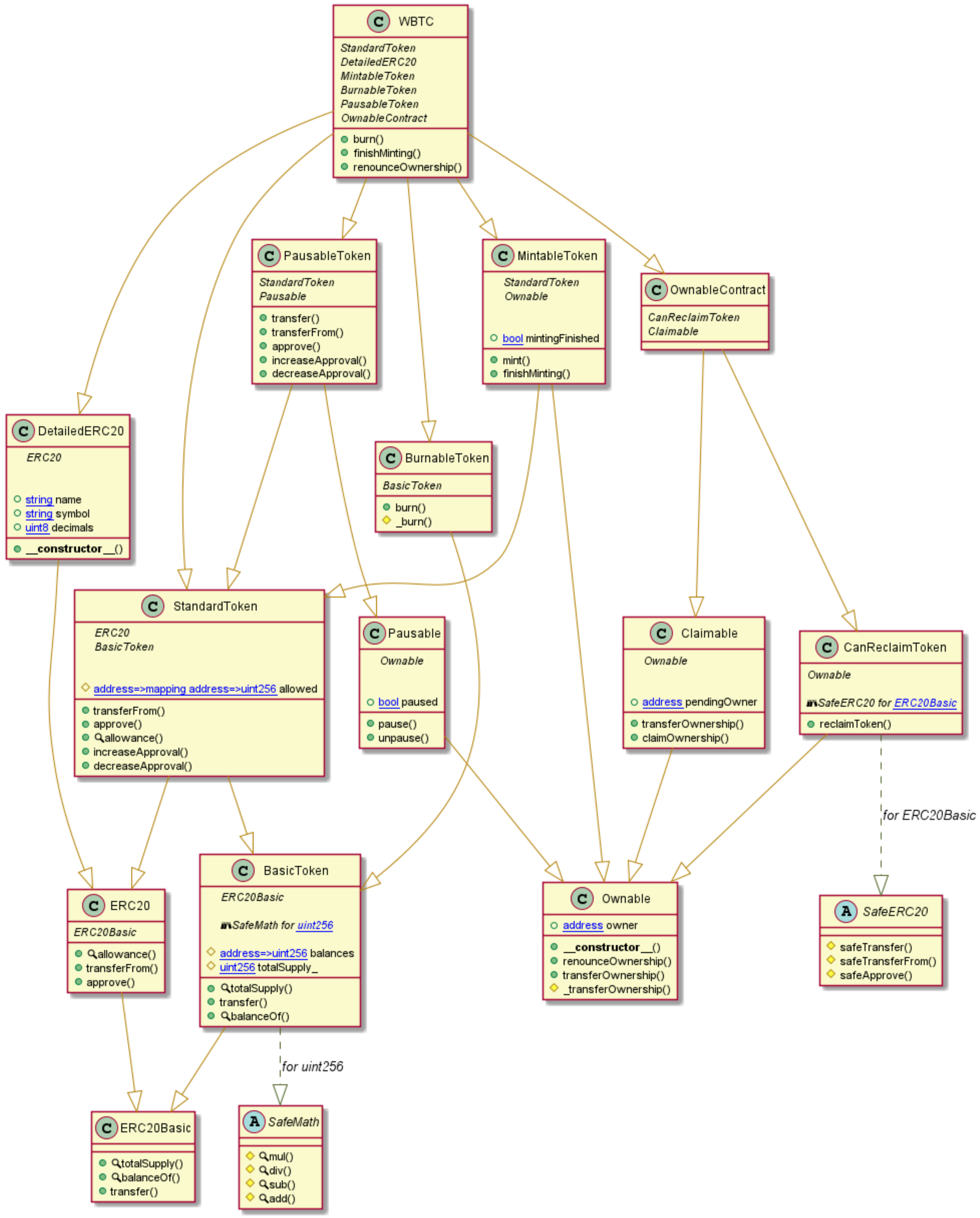
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

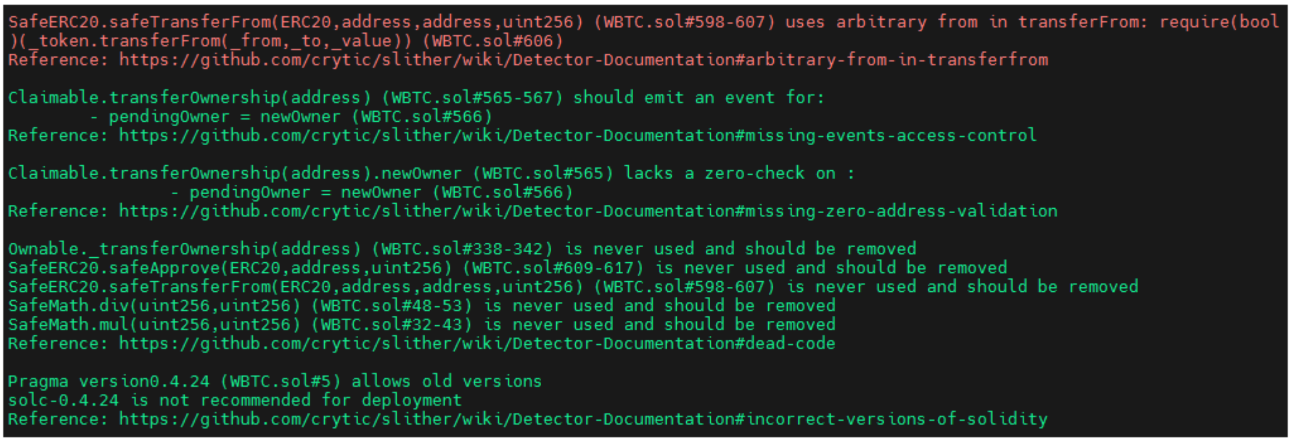


Slither Results Log

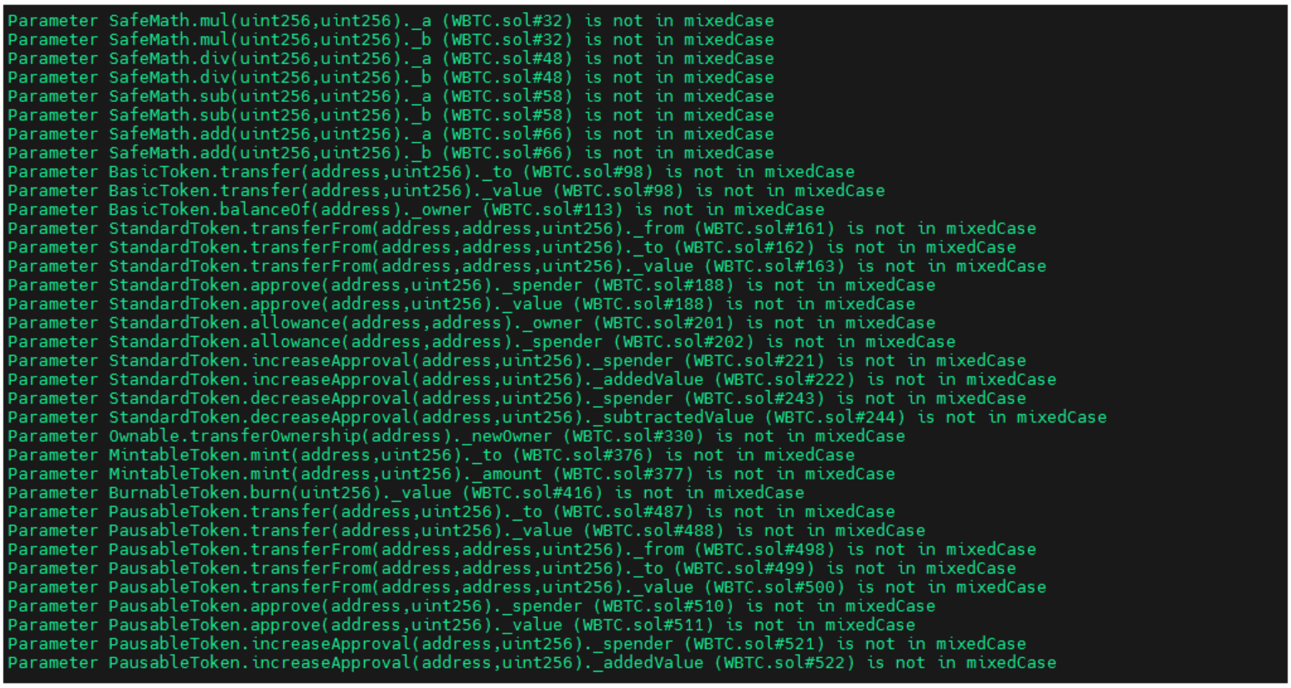
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report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

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

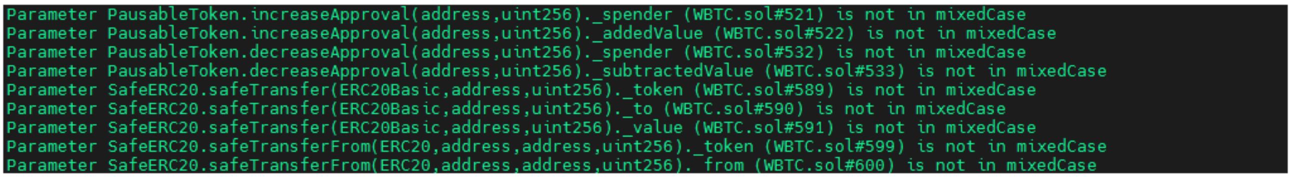
Slither Log >> WBTC.sol



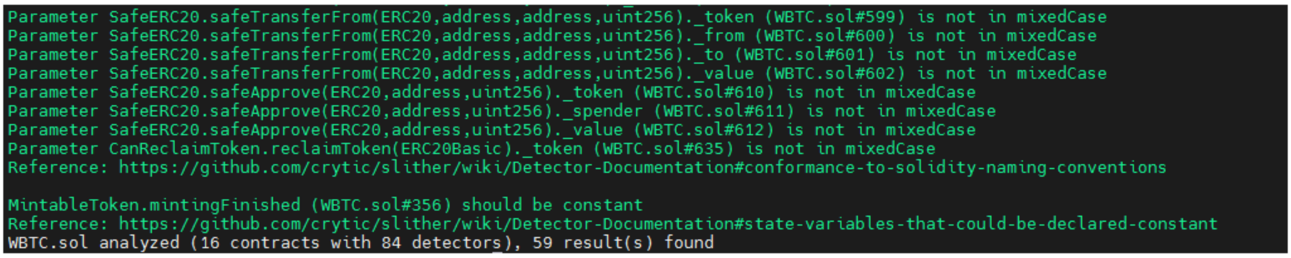
```
SafeERC20.safeTransferFrom(ERC20,address,address,uint256) (WBTC.sol#598-607) uses arbitrary from in transferFrom: require(bool
)(_token.transferFrom(_from,_to,_value)) (WBTC.sol#606)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom

Claimable.transferOwnership(address) (WBTC.sol#565-567) should emit an event for:
        - pendingOwner = newOwner (WBTC.sol#566)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

Claimable.transferOwnership(address).newOwner (WBTC.sol#565) lacks a zero-check on :
                - pendingOwner = newOwner (WBTC.sol#566)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Ownable._transferOwnership(address) (WBTC.sol#338-342) is never used and should be removed
SafeERC20.safeApprove(ERC20,address,uint256) (WBTC.sol#609-617) is never used and should be removed
SafeERC20.safeTransferFrom(ERC20,address,address,uint256) (WBTC.sol#598-607) is never used and should be removed
SafeMath.div(uint256,uint256) (WBTC.sol#48-53) is never used and should be removed
SafeMath.mul(uint256,uint256) (WBTC.sol#32-43) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.4.24 (WBTC.sol#5) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```



```
Parameter SafeMath.mul(uint256,uint256)._a (WBTC.sol#32) is not in mixedCase
Parameter SafeMath.mul(uint256,uint256)._b (WBTC.sol#32) is not in mixedCase
Parameter SafeMath.div(uint256,uint256)._a (WBTC.sol#48) is not in mixedCase
Parameter SafeMath.div(uint256,uint256)._b (WBTC.sol#48) is not in mixedCase
Parameter SafeMath.sub(uint256,uint256)._a (WBTC.sol#58) is not in mixedCase
Parameter SafeMath.sub(uint256,uint256)._b (WBTC.sol#58) is not in mixedCase
Parameter SafeMath.add(uint256,uint256)._a (WBTC.sol#66) is not in mixedCase
Parameter SafeMath.add(uint256,uint256)._b (WBTC.sol#66) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._to (WBTC.sol#98) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._value (WBTC.sol#98) is not in mixedCase
Parameter BasicToken.balanceOf(address)._owner (WBTC.sol#113) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (WBTC.sol#161) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (WBTC.sol#162) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (WBTC.sol#163) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._spender (WBTC.sol#188) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (WBTC.sol#188) is not in mixedCase
Parameter StandardToken.allowance(address,address)._owner (WBTC.sol#201) is not in mixedCase
Parameter StandardToken.allowance(address,address)._spender (WBTC.sol#202) is not in mixedCase
Parameter StandardToken.increaseApproval(address,uint256)._spender (WBTC.sol#221) is not in mixedCase
Parameter StandardToken.increaseApproval(address,uint256)._addedValue (WBTC.sol#222) is not in mixedCase
Parameter StandardToken.decreaseApproval(address,uint256)._spender (WBTC.sol#243) is not in mixedCase
Parameter StandardToken.decreaseApproval(address,uint256)._subtractedValue (WBTC.sol#244) is not in mixedCase
Parameter Ownable.transferOwnership(address)._newOwner (WBTC.sol#330) is not in mixedCase
Parameter MintableToken.mint(address,uint256)._to (WBTC.sol#376) is not in mixedCase
Parameter MintableToken.mint(address,uint256)._amount (WBTC.sol#377) is not in mixedCase
Parameter BurnableToken.burn(uint256)._value (WBTC.sol#416) is not in mixedCase
Parameter PausableToken.transfer(address,uint256)._to (WBTC.sol#487) is not in mixedCase
Parameter PausableToken.transfer(address,uint256)._value (WBTC.sol#488) is not in mixedCase
Parameter PausableToken.transferFrom(address,address,uint256)._from (WBTC.sol#498) is not in mixedCase
Parameter PausableToken.transferFrom(address,address,uint256)._to (WBTC.sol#499) is not in mixedCase
Parameter PausableToken.transferFrom(address,address,uint256)._value (WBTC.sol#500) is not in mixedCase
Parameter PausableToken.approve(address,uint256)._spender (WBTC.sol#510) is not in mixedCase
Parameter PausableToken.approve(address,uint256)._value (WBTC.sol#511) is not in mixedCase
Parameter PausableToken.increaseApproval(address,uint256)._spender (WBTC.sol#521) is not in mixedCase
Parameter PausableToken.increaseApproval(address,uint256)._addedValue (WBTC.sol#522) is not in mixedCase
```



```
Parameter PausableToken.increaseApproval(address,uint256)._spender (WBTC.sol#521) is not in mixedCase
Parameter PausableToken.increaseApproval(address,uint256)._addedValue (WBTC.sol#522) is not in mixedCase
Parameter PausableToken.decreaseApproval(address,uint256)._spender (WBTC.sol#532) is not in mixedCase
Parameter PausableToken.decreaseApproval(address,uint256)._subtractedValue (WBTC.sol#533) is not in mixedCase
Parameter SafeERC20.safeTransfer(ERC20Basic,address,uint256)._token (WBTC.sol#589) is not in mixedCase
Parameter SafeERC20.safeTransfer(ERC20Basic,address,uint256)._to (WBTC.sol#590) is not in mixedCase
Parameter SafeERC20.safeTransfer(ERC20Basic,address,uint256)._value (WBTC.sol#591) is not in mixedCase
Parameter SafeERC20.safeTransferFrom(ERC20,address,address,uint256)._token (WBTC.sol#599) is not in mixedCase
Parameter SafeERC20.safeTransferFrom(ERC20,address,address,uint256)._from (WBTC.sol#600) is not in mixedCase
```

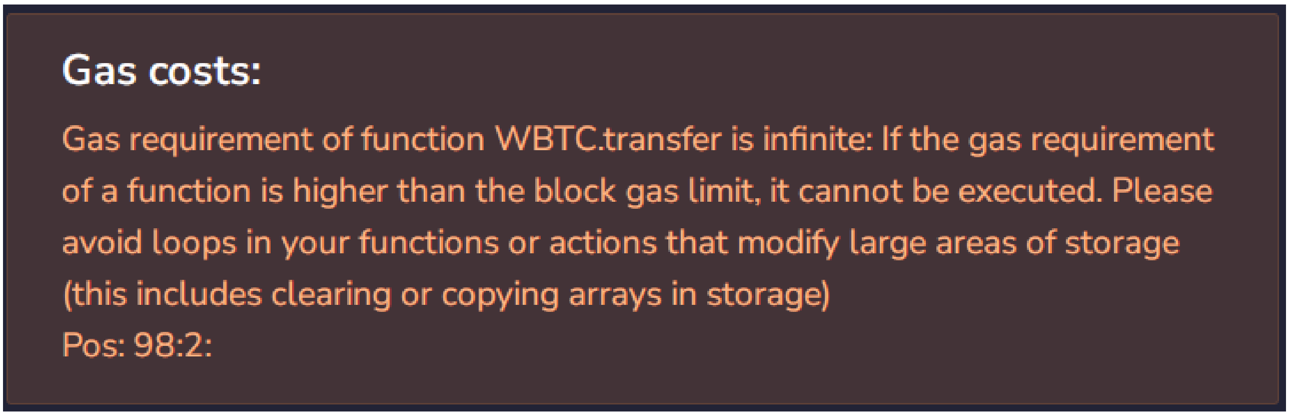


```
Parameter SafeERC20.safeTransferFrom(ERC20,address,address,uint256)._token (WBTC.sol#599) is not in mixedCase
Parameter SafeERC20.safeTransferFrom(ERC20,address,address,uint256)._from (WBTC.sol#600) is not in mixedCase
Parameter SafeERC20.safeTransferFrom(ERC20,address,address,uint256)._to (WBTC.sol#601) is not in mixedCase
Parameter SafeERC20.safeTransferFrom(ERC20,address,address,uint256)._value (WBTC.sol#602) is not in mixedCase
Parameter SafeERC20.safeApprove(ERC20,address,uint256)._token (WBTC.sol#610) is not in mixedCase
Parameter SafeERC20.safeApprove(ERC20,address,uint256)._spender (WBTC.sol#611) is not in mixedCase
Parameter SafeERC20.safeApprove(ERC20,address,uint256)._value (WBTC.sol#612) is not in mixedCase
Parameter CanReclaimToken.reclaimToken(ERC20Basic)._token (WBTC.sol#635) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

MintableToken.mintingFinished (WBTC.sol#356) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
WBTC.sol analyzed (16 contracts with 84 detectors), 59 result(s) found
```

Solidity Static Analysis

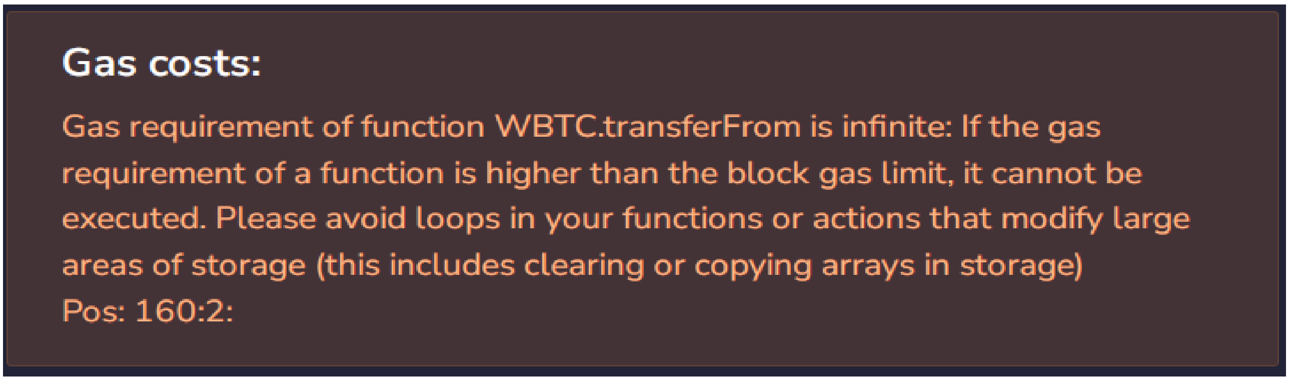
Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

WBTC.sol



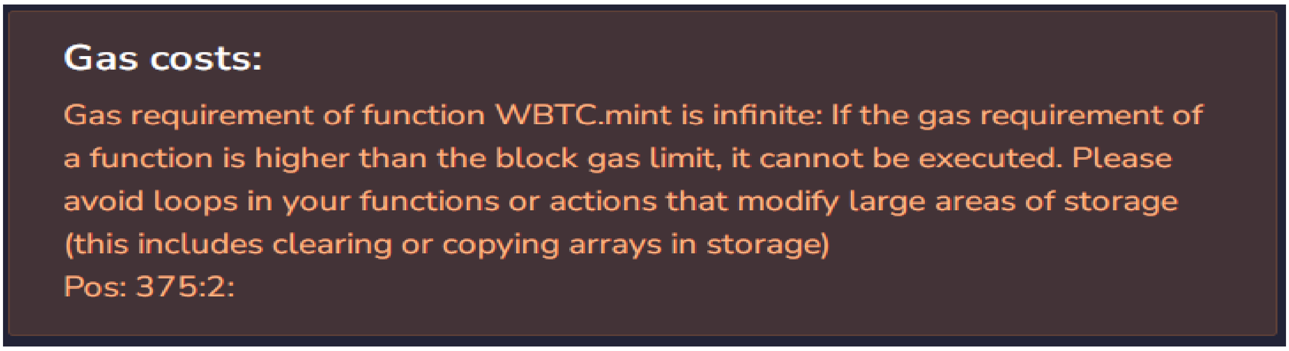
Gas costs:

Gas requirement of function WBTC.transfer 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: 98:2:



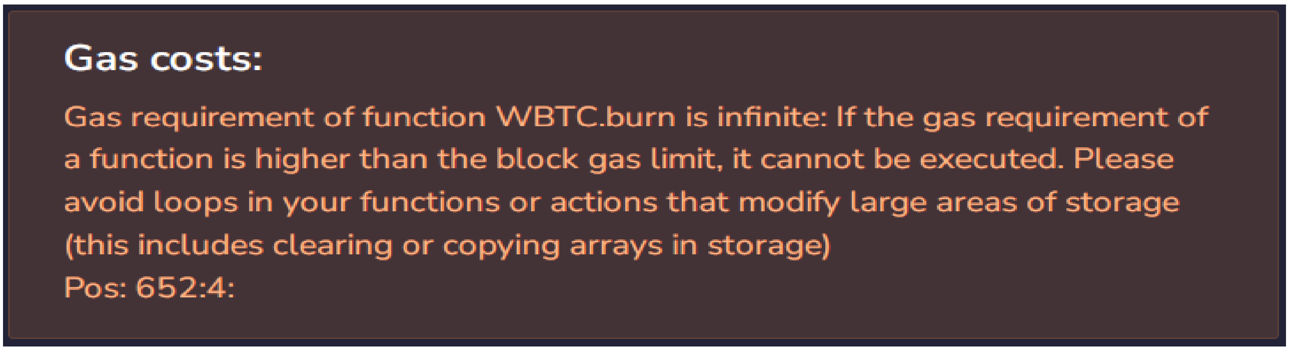
Gas costs:

Gas requirement of function WBTC.transferFrom 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: 160:2:



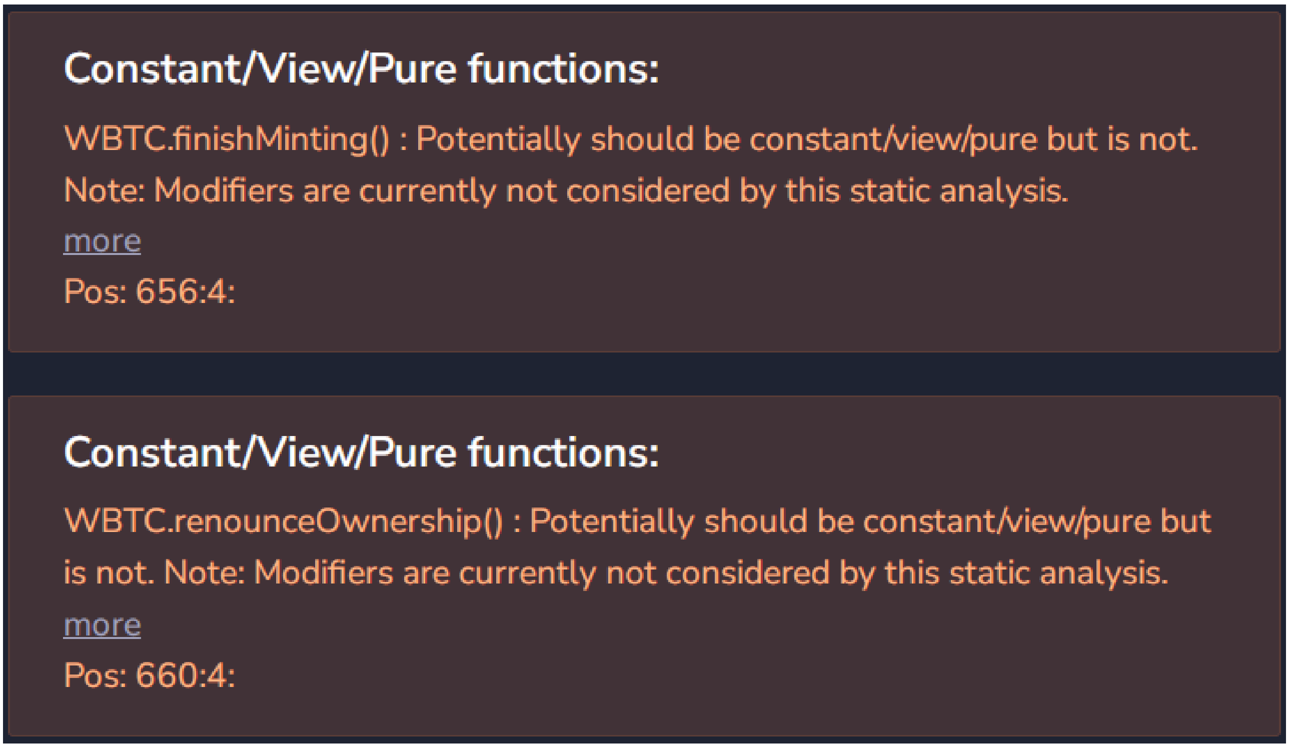
Gas costs:

Gas requirement of function WBTC.mint 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: 375:2:



Gas costs:

Gas requirement of function WBTC.burn 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: 652:4:



Constant/View/Pure functions:

WBTC.finishMinting() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

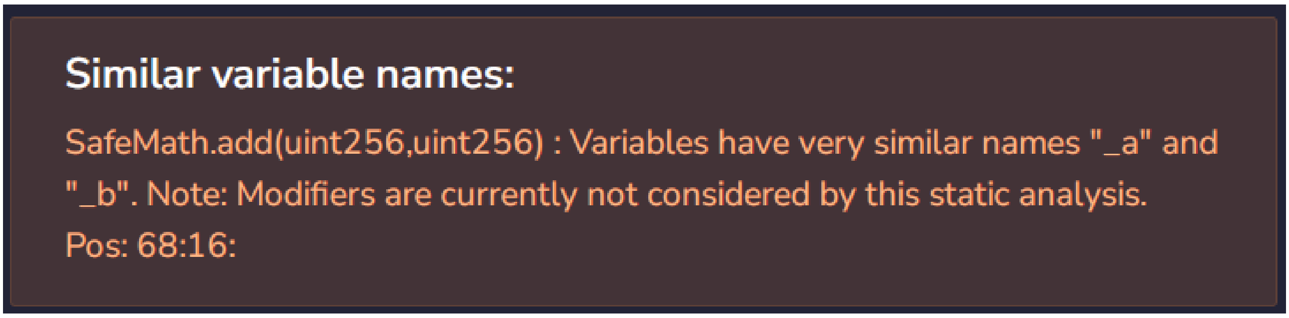
Pos: 656:4:

Constant/View/Pure functions:

WBTC.renounceOwnership() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

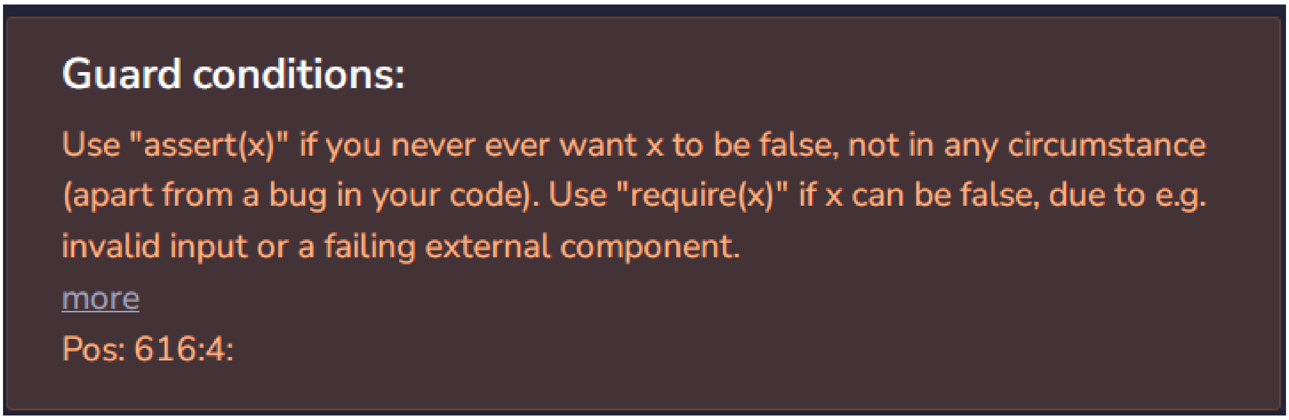
Pos: 660:4:



Similar variable names:

SafeMath.add(uint256,uint256) : Variables have very similar names "_a" and "_b". Note: Modifiers are currently not considered by this static analysis.

Pos: 68:16:

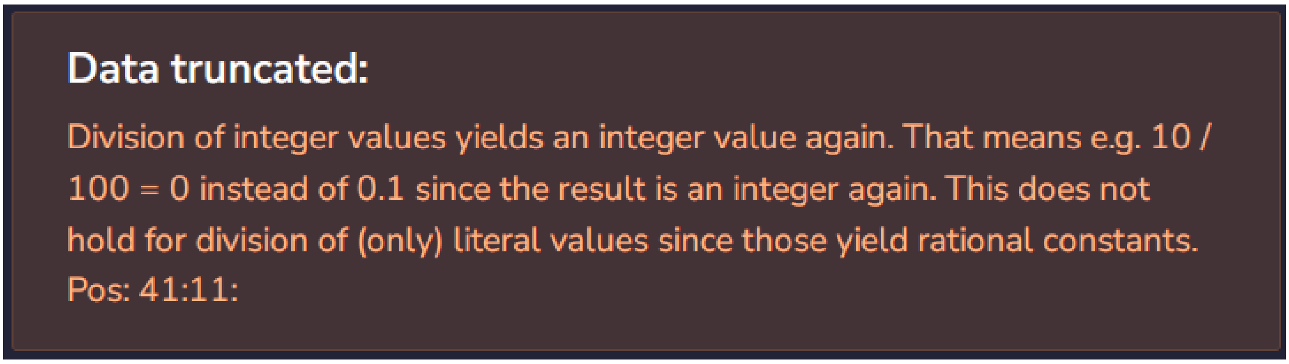


Guard conditions:

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: 616:4:



Data truncated:

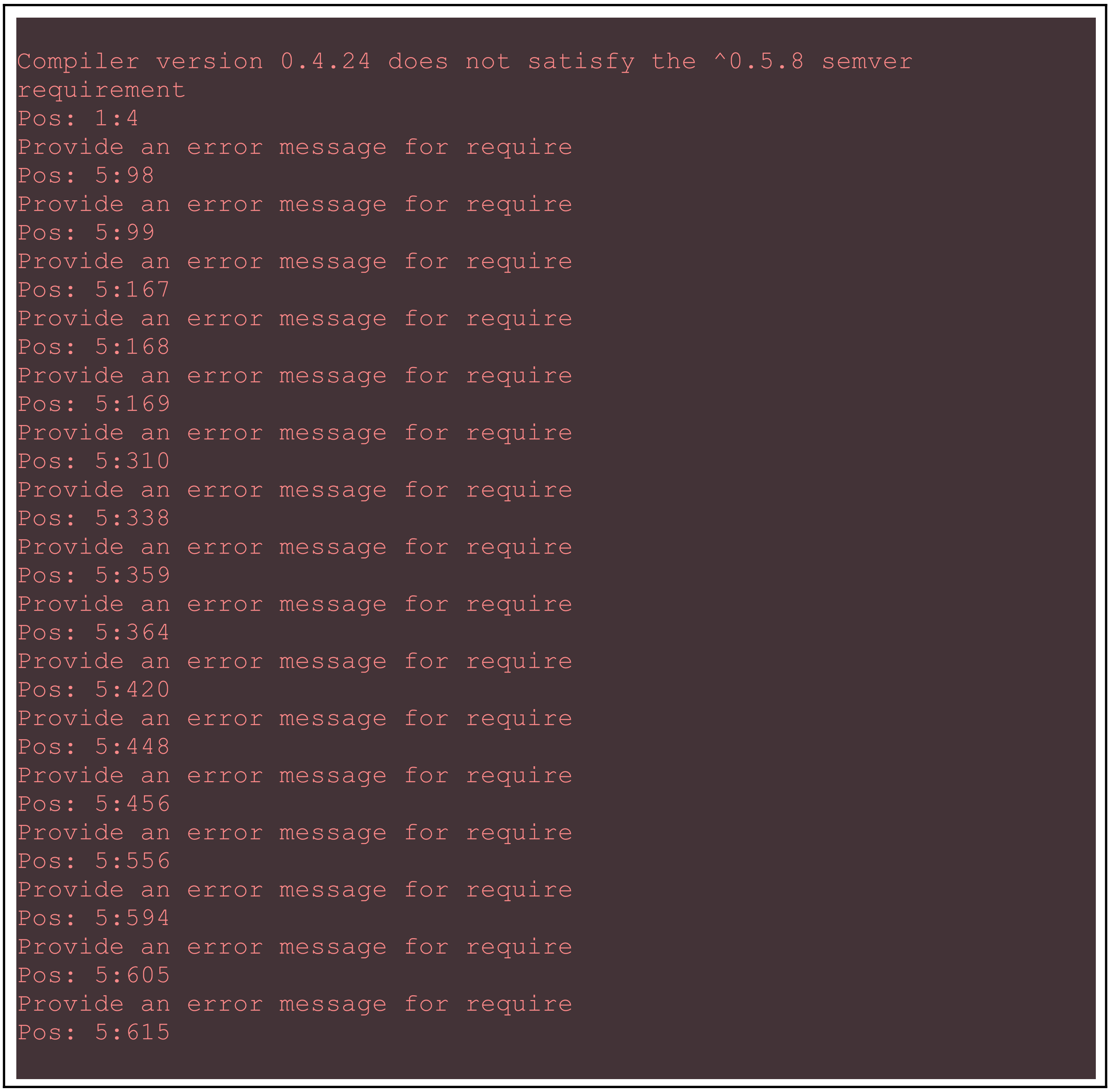
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 41:11:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

WBTC.sol



```
Compiler version 0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:4
Provide an error message for require
Pos: 5:98
Provide an error message for require
Pos: 5:99
Provide an error message for require
Pos: 5:167
Provide an error message for require
Pos: 5:168
Provide an error message for require
Pos: 5:169
Provide an error message for require
Pos: 5:310
Provide an error message for require
Pos: 5:338
Provide an error message for require
Pos: 5:359
Provide an error message for require
Pos: 5:364
Provide an error message for require
Pos: 5:420
Provide an error message for require
Pos: 5:448
Provide an error message for require
Pos: 5:456
Provide an error message for require
Pos: 5:556
Provide an error message for require
Pos: 5:594
Provide an error message for require
Pos: 5:605
Provide an error message for require
Pos: 5:615
```

Software analysis result:

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

EtherAuthority

Email: audit@EtherAuthority.io