Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Sand Token
Website:	sandbox.game
Platform:	Ethereum
Language:	Solidity
Date:	May 8th, 2024

Table of contents

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the smart contracts of Sand Token from sandbox.game were audited. 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 May 8th, 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 SAND is a smart contract written in the Ethereum blockchain's Solidity programming language. Let's break down the functionality and components of the contract:
    - **BytesUtil Library:** This library contains functions for manipulating byte arrays, including copying memory, converting addresses to bytes, and converting uint256 to bytes.
    - **ERC20Events Contract:** This contract defines events for ERC20 token transfers and approvals.
    - **Admin Contract:** This contract defines an administrator role and allows for changing the administrator address.
    - **SuperOperators Contract:** This contract extends the Admin contract and adds super operator functionality. Super operators have special rights, such as the ability to transfer tokens of all users.
    - **ERC20BasicApproveExtension Contract:** This contract provides additional functionality for approving and calling contracts with data. It includes methods for approving transfers and executing calls with specific gas limits.
    - **ERC20ExecuteExtension Contract:** This contract extends the SuperOperators contract and adds execution administrator and operator functionality. It allows for executing calls on behalf of the contract with specific gas limits and charging tokens for gas used.

    - **ERC20BaseToken Contract:** This contract implements the basic functionality of an ERC20 token. It includes methods for transferring tokens, checking balances, and approving transfers.
    - **Sand Contract:** This contract implements the SAND ERC20 token. It inherits functionality from ERC20ExecuteExtension and ERC20BasicApproveExtension contracts and initializes the SAND token with an initial supply of 3 billion tokens.
- Overall, the provided code constitutes a comprehensive ERC20 token implementation with additional features such as super operators, execution administrators, and extended approval and execution functionalities. The SAND token is designed to be used within the Ethereum ecosystem for various purposes such as payments, governance, or utility within decentralized applications (dApps).

Audit scope

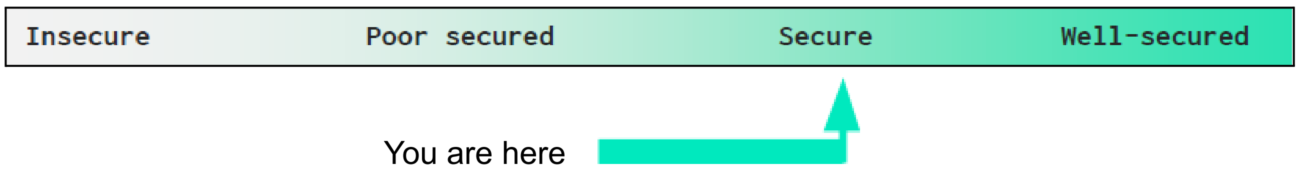
Name	**Code Review and Security Analysis Report for Sand Token Smart Contract**
Platform	**Ethereum**
File	Sand.sol
Smart Contract Code	0x3845badAde8e6dFF049820680d1F14bD3903a5d0
Audit Date	May 8th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: SAND • Symbol: SAND • Decimals: 18 • Total Supply: 3 billion	**YES, This is valid.**
Admin control: • The admin can update the new super operator address. • The current admin can update a new admin address. • The admin can update the new execution admin and operator address. • The owner can add an allowance.	**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 6 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 is not specified	Passed
	Solidity version is 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	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

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?	No
🟢 Max Tax?	No
🟢 Is it Anti-whale?	No
🟢 Is Anti-bot?	Not Detected
🟢 Is it a Blacklist?	Not Detected
🟢 Blacklist Check	No
🟢 Can Mint?	No
🟢 Is it a Proxy?	No
🟢 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 Sand 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 Sand 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 well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Sand Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are well commented on. and 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	Missing Zero Address Validation	Refer Audit Findings
2	name	read	Inappropriate State Mutability for the Function	Refer Audit Findings
3	symbol	read	Inappropriate State Mutability for the Function	Refer Audit Findings
4	totalSupply	read	Passed	No Issue
5	balanceOf	read	Passed	No Issue
6	allowance	read	Passed	No Issue
7	decimals	read	Inappropriate State Mutability for the Function	Refer Audit Findings
8	transfer	write	Passed	No Issue
9	transferFrom	write	Passed	No Issue
10	burn	external	Passed	No Issue
11	burnFor	external	Passed	No Issue
12	approve	write	Passed	No Issue
13	approveFor	write	Passed	No Issue
14	addAllowanceIfNeeded	write	Passed	No Issue
15	_addAllowanceIfNeeded	internal	Passed	No Issue
16	_approveFor	internal	Passed	No Issue
17	_transfer	internal	Passed	No Issue
18	_mint	internal	Passed	No Issue
19	_burn	internal	Passed	No Issue
20	getExecutionAdmin	external	Passed	No Issue
21	changeExecutionAdmin	external	Passed	No Issue
22	setExecutionOperator	external	Passed	No Issue
23	isExecutionOperator	read	Passed	No Issue
24	executeWithSpecificGas	external	Missing Zero Address Validation	Refer Audit Findings
25	approveAndExecuteWithSpecificGas	external	Passed	No Issue
26	approveAndExecuteWithSpecificGasAndChargeForIt	external	Passed	No Issue
27	transferAndChargeForGas	external	Passed	No Issue
28	_charge	internal	Passed	No Issue
29	_approveAndExecuteWithSpecificGas	internal	Passed	No Issue
30	_transfer	internal	Passed	No Issue
31	_addAllowanceIfNeeded	internal	Passed	No Issue
32	approveAndCall	external	Passed	No Issue

33	paidCall	external	Passed	No Issue
34	_approveFor	internal	Passed	No Issue
35	_addAllowanceIfNeeded	internal	Passed	No Issue
36	setSuperOperator	external	Passed	No Issue
37	isSuperOperator	read	Passed	No Issue
38	getAdmin	external	Passed	No Issue
39	changeAdmin	external	Missing Zero Address Validation	Refer Audit Findings

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 a 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 a 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

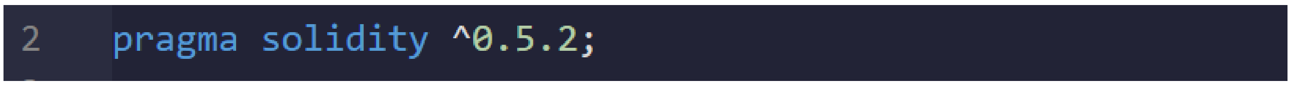
No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Use the latest solidity version: **Admin.sol, BytesUtil.sol**

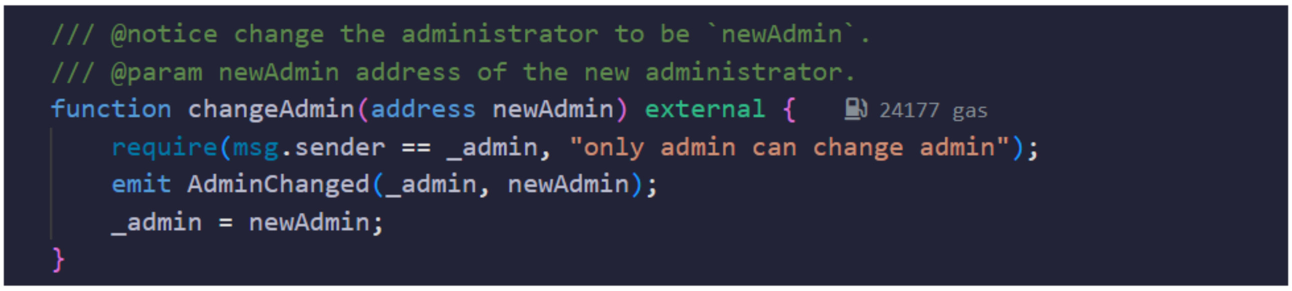


```
pragma solidity ^0.5.2;
```

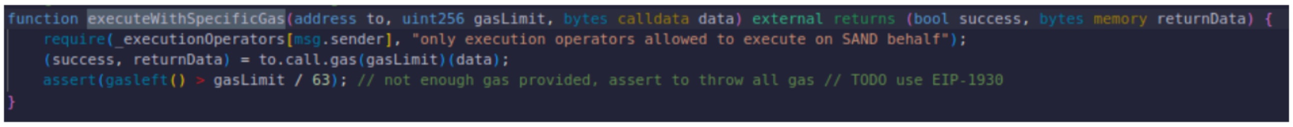
Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use 0.8.22 which is the latest version.

(2) Missing Zero Address Validation: **Admin.sol, ERC20ExecuteExtension.sol**

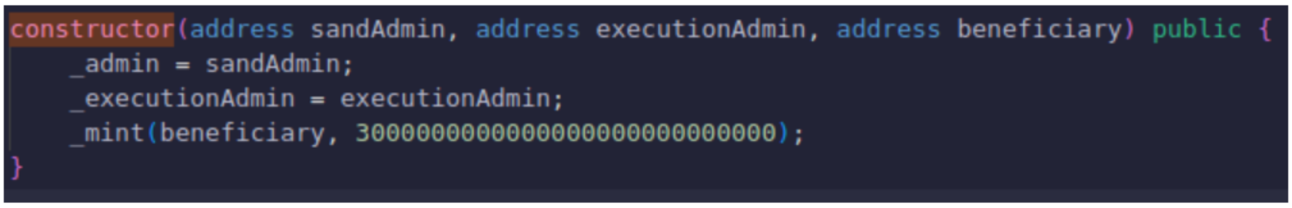


```
/// @notice change the administrator to be `newAdmin`.
/// @param newAdmin address of the new administrator.
function changeAdmin(address newAdmin) external {    24177 gas
    require(msg.sender == _admin, "only admin can change admin");
    emit AdminChanged(_admin, newAdmin);
    _admin = newAdmin;
}
```



```
function executeWithSpecificGas(address to, uint256 gasLimit, bytes calldata data) external returns (bool success, bytes memory returnData) {
    require(_executionOperators[msg.sender], "only execution operators allowed to execute on SAND behalf");
    (success, returnData) = to.call.gas(gasLimit)(data);
    assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw all gas // TODO use EIP-1930
}
```

Sand.sol

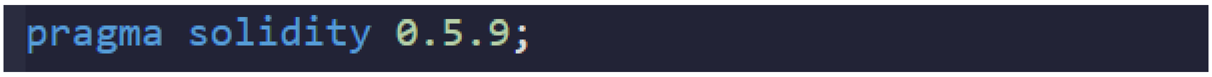


```
constructor(address sandAdmin, address executionAdmin, address beneficiary) public {
    _admin = sandAdmin;
    _executionAdmin = executionAdmin;
    _mint(beneficiary, 3000000000000000000000000000);
}
```

Some functions in this contract may not appropriately check for zero addresses being used.

Resolution: Check that newAdmin is not zero.

(3) Floating Pragma: **ERC20BaseToken.sol, ERC20BasicApproveExtension.sol, ERC20Events.sol, ERC20ExecuteExtension.sol, Sand.sol, SuperOperators.sol**



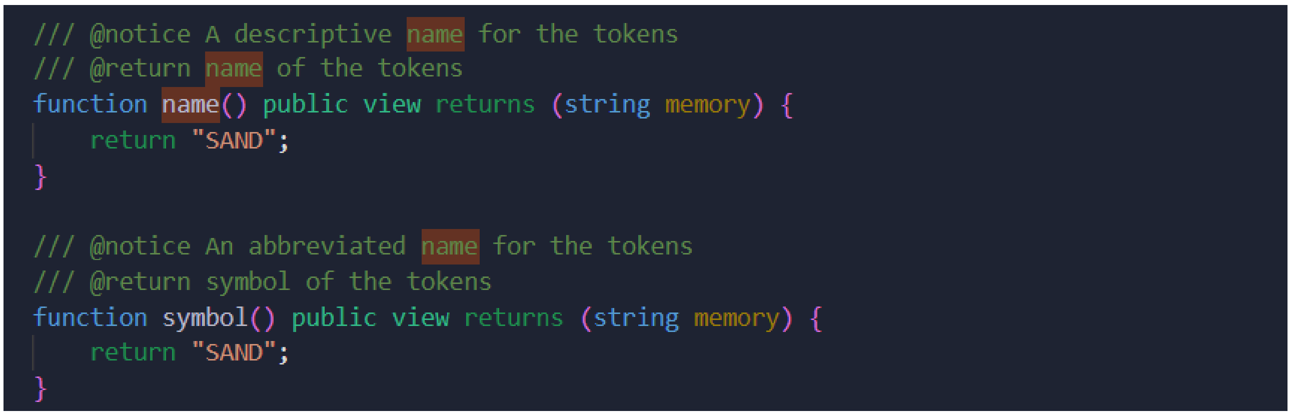
```
pragma solidity 0.5.9;
```

This contract may not function as expected due to inconsistent solidity compiler versions being specified.

Resolution: Specify and lock the Solidity pragma version to a fixed and known version in your smart contract to avoid potential issues caused by unexpected compiler behavior in future versions.

(4) Inappropriate State Mutability for the Function:

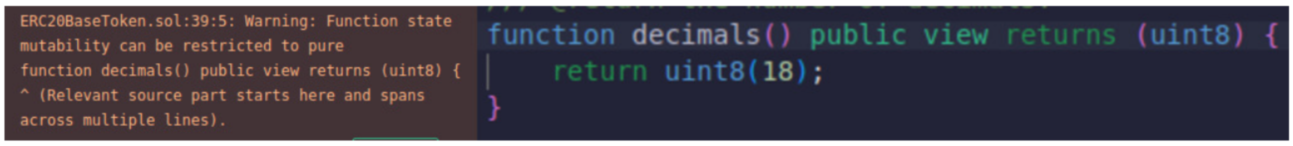
Sand.sol



```
/// @notice A descriptive name for the tokens
/// @return name of the tokens
function name() public view returns (string memory) {
    return "SAND";
}

/// @notice An abbreviated name for the tokens
/// @return symbol of the tokens
function symbol() public view returns (string memory) {
    return "SAND";
}
```

ERC20BaseToken.sol



```
ERC20BaseToken.sol:39:5: Warning: Function state
mutability can be restricted to pure
function decimals() public view returns (uint8) {
^ (Relevant source part starts here and spans
across multiple lines).
```

```
function decimals() public view returns (uint8) {
    return uint8(18);
}
```

The decimals function is erroneously declared as public view when it doesn't modify any state variables. This misleading declaration can lead to misunderstandings about the function's behavior.

Resolution: Revise the decimals, name, and symbol functions state mutability to accurately represent their non-state-modifying nature, thereby enhancing code clarity and optimizing gas consumption.

(5) Low-Level Calls: **ERC20BasicApproveExtension.sol, ERC20ExecuteExtension.sol**
This contract uses low-level calls, which may be unsafe.

Resolution: Enhance safety by reviewing low-level calls, considering high-level alternatives, and consulting security experts. Prioritize code security and integrity.

(6) Public Functions Should be Declared External: **Sand.sol**
Some functions in this contract should be declared as external in order to save gas.

Resolution: Update public functions in the smart contract code to explicitly declare them as "external" for clarity and to adhere to best practices for state mutability.

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. The following are Admin functions:

ERC20BaseToken.sol

- approveFor: The user'spender' is authorized to transfer 'amount' tokens from 'owner'.
- addAllowanceIfNeeded: The owner can add allowance.

ERC20ExecuteExtension.sol

- changeExecutionAdmin: The admin can update the new execution admin address.
- setExecutionOperator: The admin can update the new execution operator address.

SuperOperators.sol

- setSuperOperator: The admin can update the new super operator address.

Admin.sol

- changeAdmin: The current admin can update a new admin address.

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 6 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, and 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).

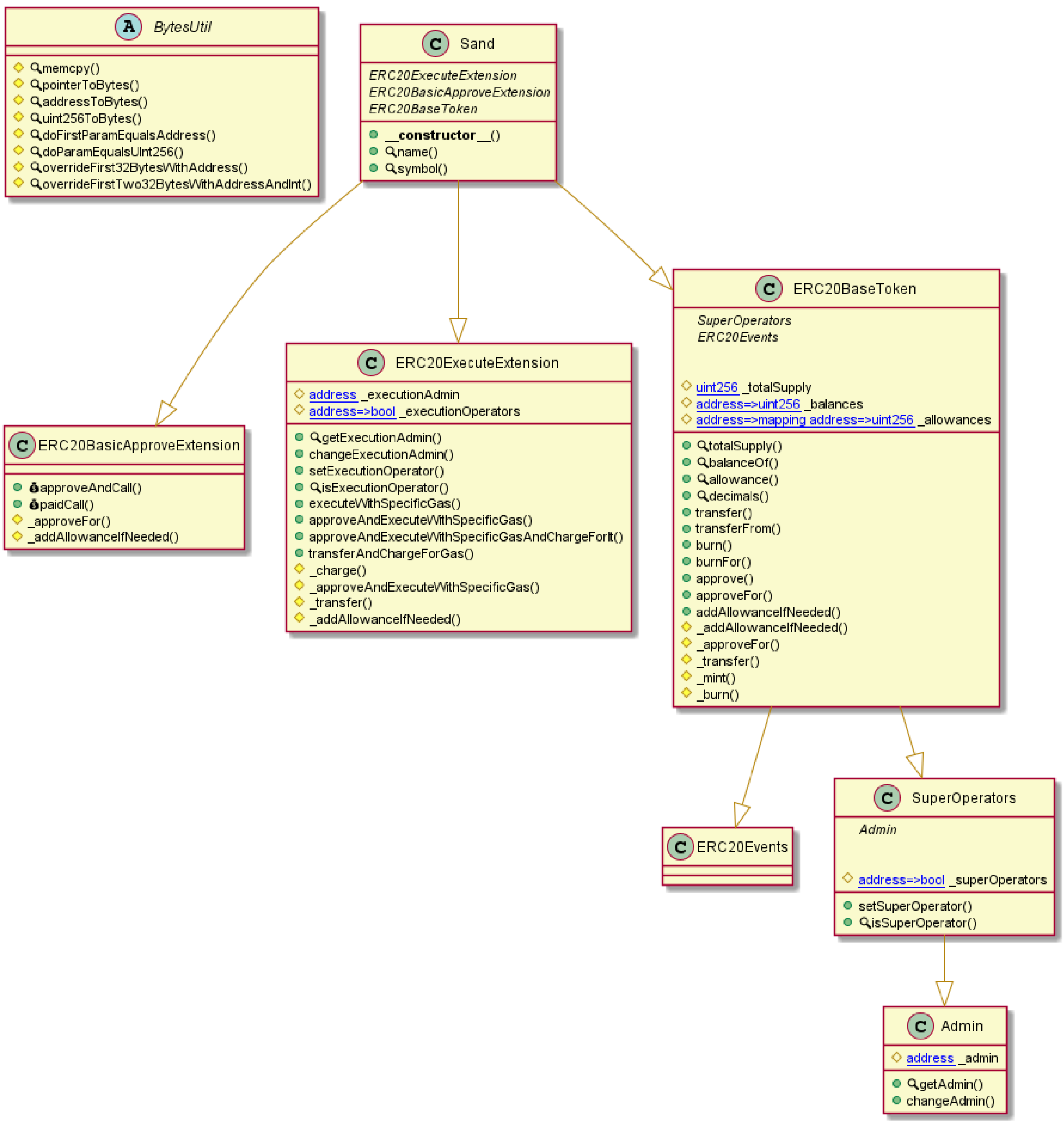
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While 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 the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Sand Token

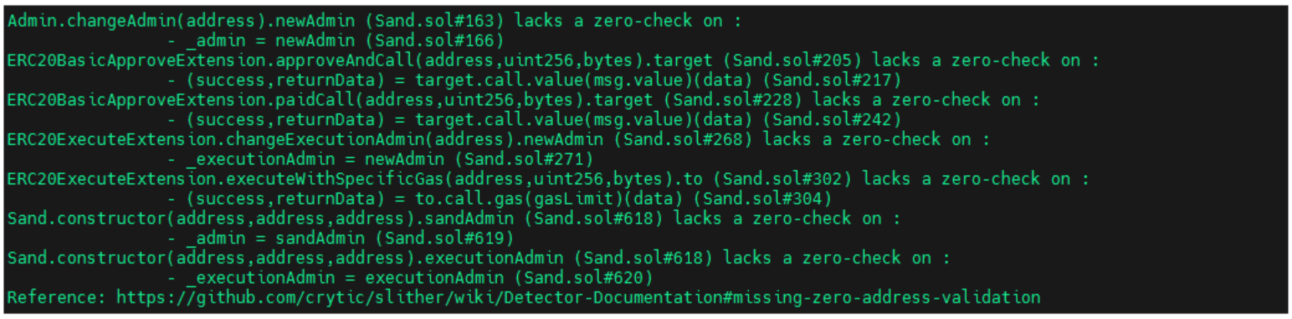


Slither Results Log

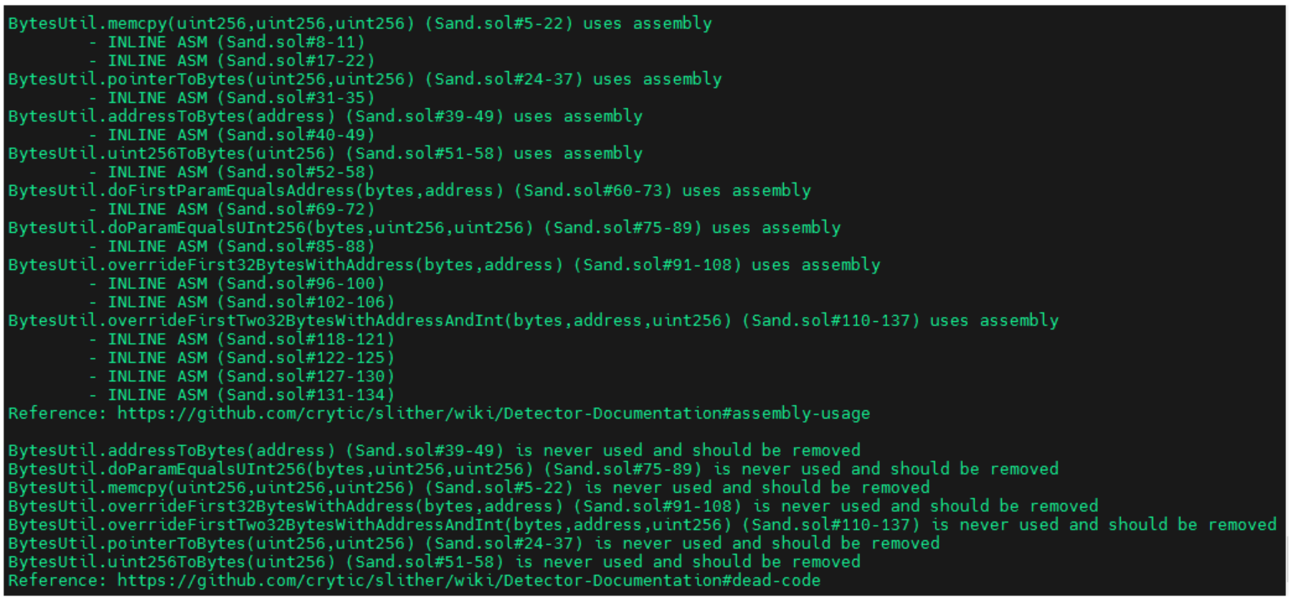
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report 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 >> Sand.sol

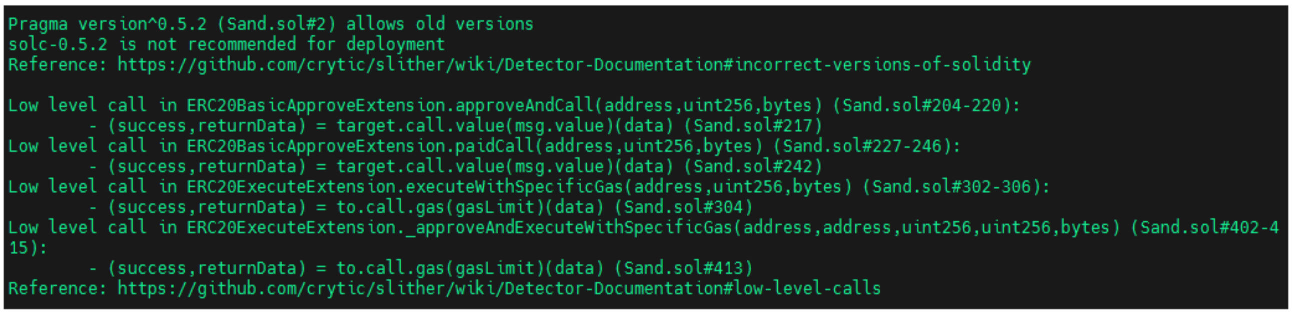


```
Admin.changeAdmin(address).newAdmin (Sand.sol#163) lacks a zero-check on :
                - _admin = newAdmin (Sand.sol#166)
ERC20BasicApproveExtension.approveAndCall(address,uint256,bytes).target (Sand.sol#205) lacks a zero-check on :
                - (success,returnData) = target.call.value(msg.value)(data) (Sand.sol#217)
ERC20BasicApproveExtension.paidCall(address,uint256,bytes).target (Sand.sol#228) lacks a zero-check on :
                - (success,returnData) = target.call.value(msg.value)(data) (Sand.sol#242)
ERC20ExecuteExtension.changeExecutionAdmin(address).newAdmin (Sand.sol#268) lacks a zero-check on :
                - _executionAdmin = newAdmin (Sand.sol#271)
ERC20ExecuteExtension.executeWithSpecificGas(address,uint256,bytes).to (Sand.sol#302) lacks a zero-check on :
                - (success,returnData) = to.call.gas(gasLimit)(data) (Sand.sol#304)
Sand.constructor(address,address,address).sandAdmin (Sand.sol#618) lacks a zero-check on :
                - _admin = sandAdmin (Sand.sol#619)
Sand.constructor(address,address,address).executionAdmin (Sand.sol#618) lacks a zero-check on :
                - _executionAdmin = executionAdmin (Sand.sol#620)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```



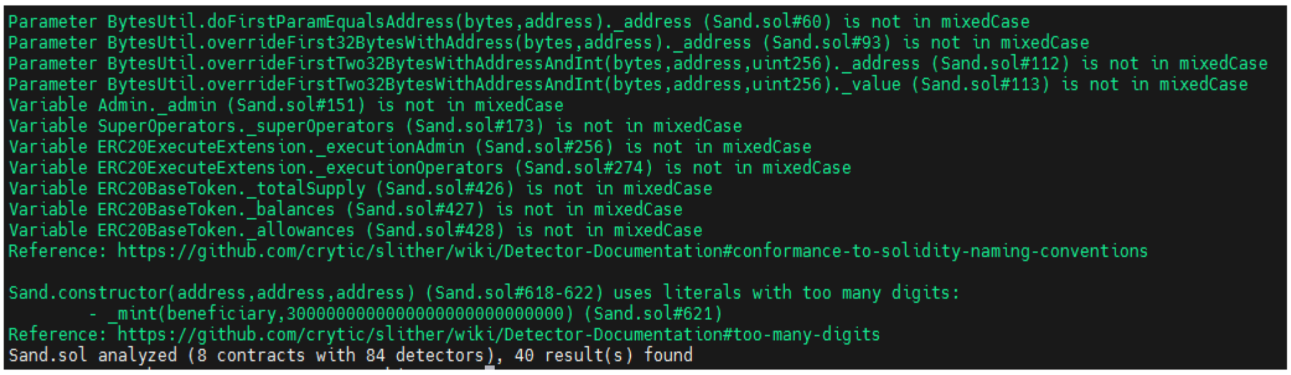
```
BytesUtil.memcpy(uint256,uint256,uint256) (Sand.sol#5-22) uses assembly
        - INLINE ASM (Sand.sol#8-11)
        - INLINE ASM (Sand.sol#17-22)
BytesUtil.pointerToBytes(uint256,uint256) (Sand.sol#24-37) uses assembly
        - INLINE ASM (Sand.sol#31-35)
BytesUtil.addressToBytes(address) (Sand.sol#39-49) uses assembly
        - INLINE ASM (Sand.sol#40-49)
BytesUtil.uint256ToBytes(uint256) (Sand.sol#51-58) uses assembly
        - INLINE ASM (Sand.sol#52-58)
BytesUtil.doFirstParamEqualsAddress(bytes,address) (Sand.sol#60-73) uses assembly
        - INLINE ASM (Sand.sol#69-72)
BytesUtil.doParamEqualsUInt256(bytes,uint256,uint256) (Sand.sol#75-89) uses assembly
        - INLINE ASM (Sand.sol#85-88)
BytesUtil.overrideFirst32BytesWithAddress(bytes,address) (Sand.sol#91-108) uses assembly
        - INLINE ASM (Sand.sol#96-100)
        - INLINE ASM (Sand.sol#102-106)
BytesUtil.overrideFirstTwo32BytesWithAddressAndInt(bytes,address,uint256) (Sand.sol#110-137) uses assembly
        - INLINE ASM (Sand.sol#118-121)
        - INLINE ASM (Sand.sol#122-125)
        - INLINE ASM (Sand.sol#127-130)
        - INLINE ASM (Sand.sol#131-134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

BytesUtil.addressToBytes(address) (Sand.sol#39-49) is never used and should be removed
BytesUtil.doParamEqualsUInt256(bytes,uint256,uint256) (Sand.sol#75-89) is never used and should be removed
BytesUtil.memcpy(uint256,uint256,uint256) (Sand.sol#5-22) is never used and should be removed
BytesUtil.overrideFirst32BytesWithAddress(bytes,address) (Sand.sol#91-108) is never used and should be removed
BytesUtil.overrideFirstTwo32BytesWithAddressAndInt(bytes,address,uint256) (Sand.sol#110-137) is never used and should be removed
BytesUtil.pointerToBytes(uint256,uint256) (Sand.sol#24-37) is never used and should be removed
BytesUtil.uint256ToBytes(uint256) (Sand.sol#51-58) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```



```
Pragma version^0.5.2 (Sand.sol#2) allows old versions
solc-0.5.2 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in ERC20BasicApproveExtension.approveAndCall(address,uint256,bytes) (Sand.sol#204-220):
        - (success,returnData) = target.call.value(msg.value)(data) (Sand.sol#217)
Low level call in ERC20BasicApproveExtension.paidCall(address,uint256,bytes) (Sand.sol#227-246):
        - (success,returnData) = target.call.value(msg.value)(data) (Sand.sol#242)
Low level call in ERC20ExecuteExtension.executeWithSpecificGas(address,uint256,bytes) (Sand.sol#302-306):
        - (success,returnData) = to.call.gas(gasLimit)(data) (Sand.sol#304)
Low level call in ERC20ExecuteExtension._approveAndExecuteWithSpecificGas(address,address,uint256,uint256,bytes) (Sand.sol#402-4
15):
        - (success,returnData) = to.call.gas(gasLimit)(data) (Sand.sol#413)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```



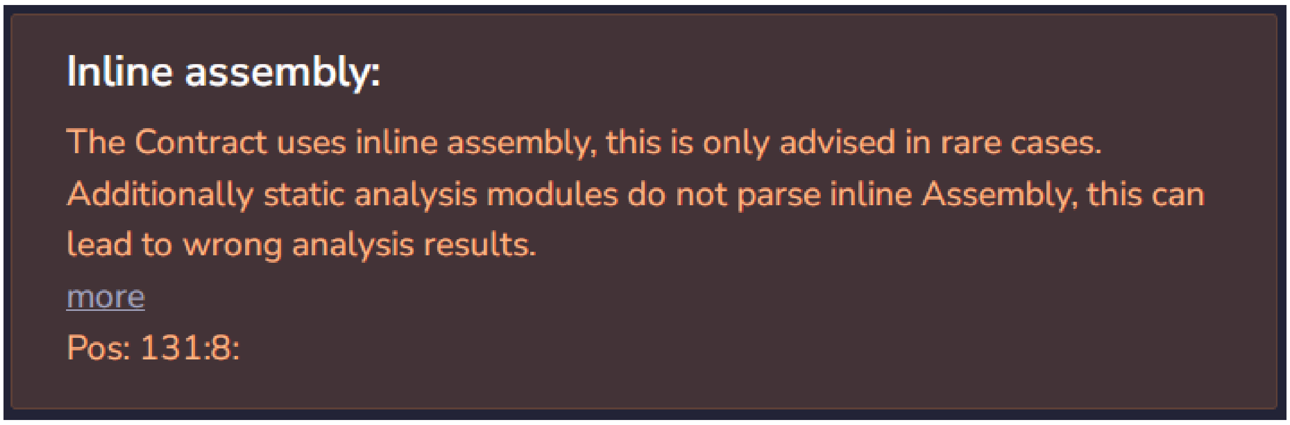
```
Parameter BytesUtil.doFirstParamEqualsAddress(bytes,address)._address (Sand.sol#60) is not in mixedCase
Parameter BytesUtil.overrideFirst32BytesWithAddress(bytes,address)._address (Sand.sol#93) is not in mixedCase
Parameter BytesUtil.overrideFirstTwo32BytesWithAddressAndInt(bytes,address,uint256)._address (Sand.sol#112) is not in mixedCase
Parameter BytesUtil.overrideFirstTwo32BytesWithAddressAndInt(bytes,address,uint256)._value (Sand.sol#113) is not in mixedCase
Variable Admin._admin (Sand.sol#151) is not in mixedCase
Variable SuperOperators._superOperators (Sand.sol#173) is not in mixedCase
Variable ERC20ExecuteExtension._executionAdmin (Sand.sol#256) is not in mixedCase
Variable ERC20ExecuteExtension._executionOperators (Sand.sol#274) is not in mixedCase
Variable ERC20BaseToken._totalSupply (Sand.sol#426) is not in mixedCase
Variable ERC20BaseToken._balances (Sand.sol#427) is not in mixedCase
Variable ERC20BaseToken._allowances (Sand.sol#428) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Sand.constructor(address,address,address) (Sand.sol#618-622) uses literals with too many digits:
        - _mint(beneficiary,3000000000000000000000000000) (Sand.sol#621)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
Sand.sol analyzed (8 contracts with 84 detectors), 40 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.

Sand.sol

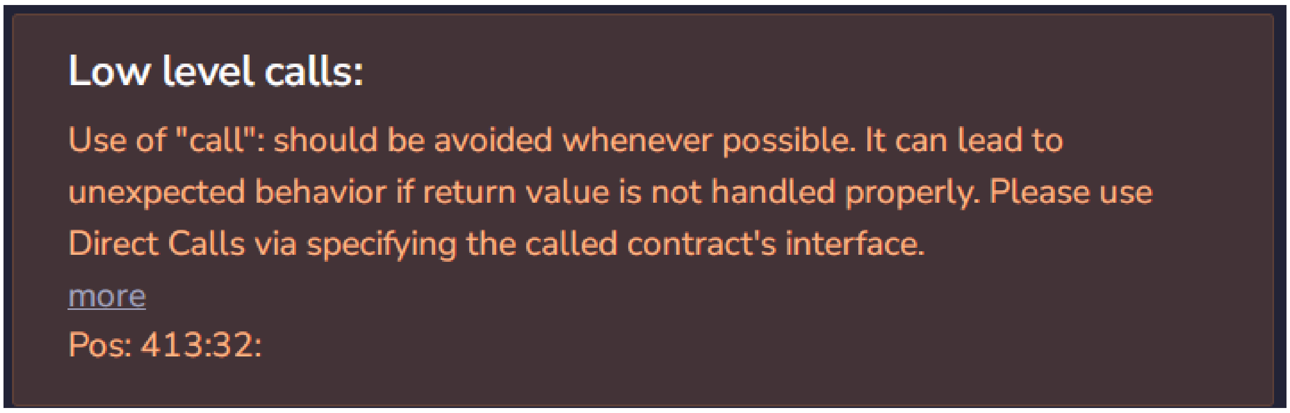


Inline assembly:

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: 131:8:

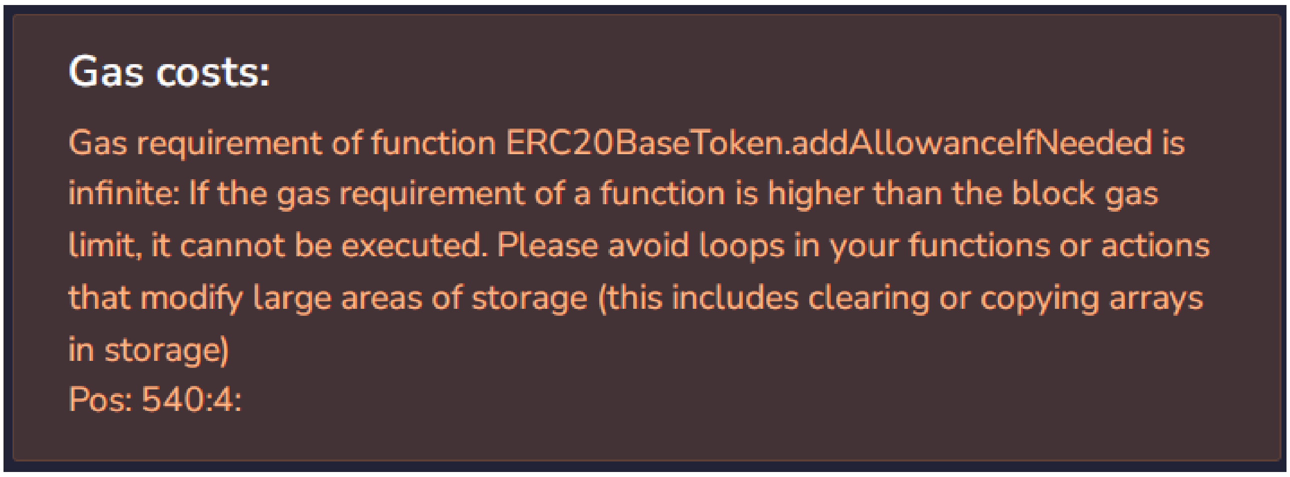


Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

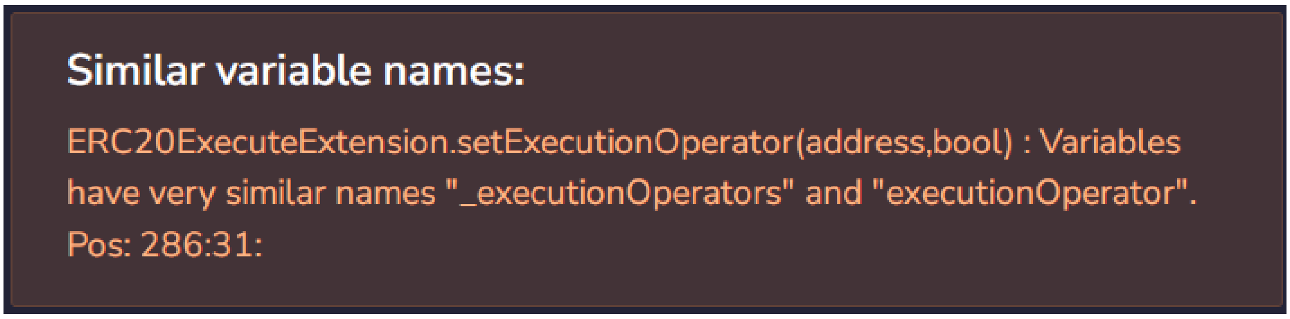
Pos: 413:32:



Gas costs:

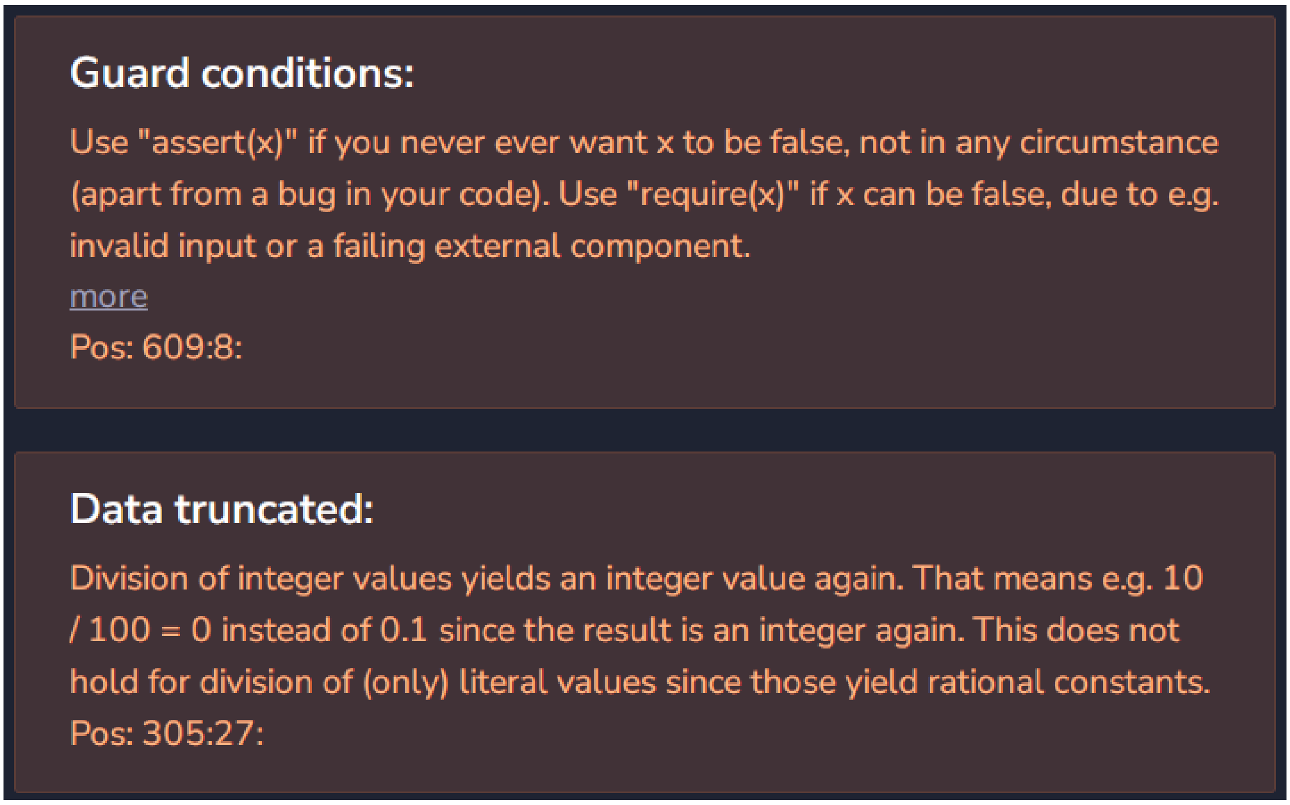
Gas requirement of function ERC20BaseToken.addAllowanceIfNeeded is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 540:4:



Similar variable names:

ERC20ExecuteExtension.setExecutionOperator(address,bool) : Variables have very similar names "_executionOperators" and "executionOperator". Pos: 286:31:



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: 609:8:

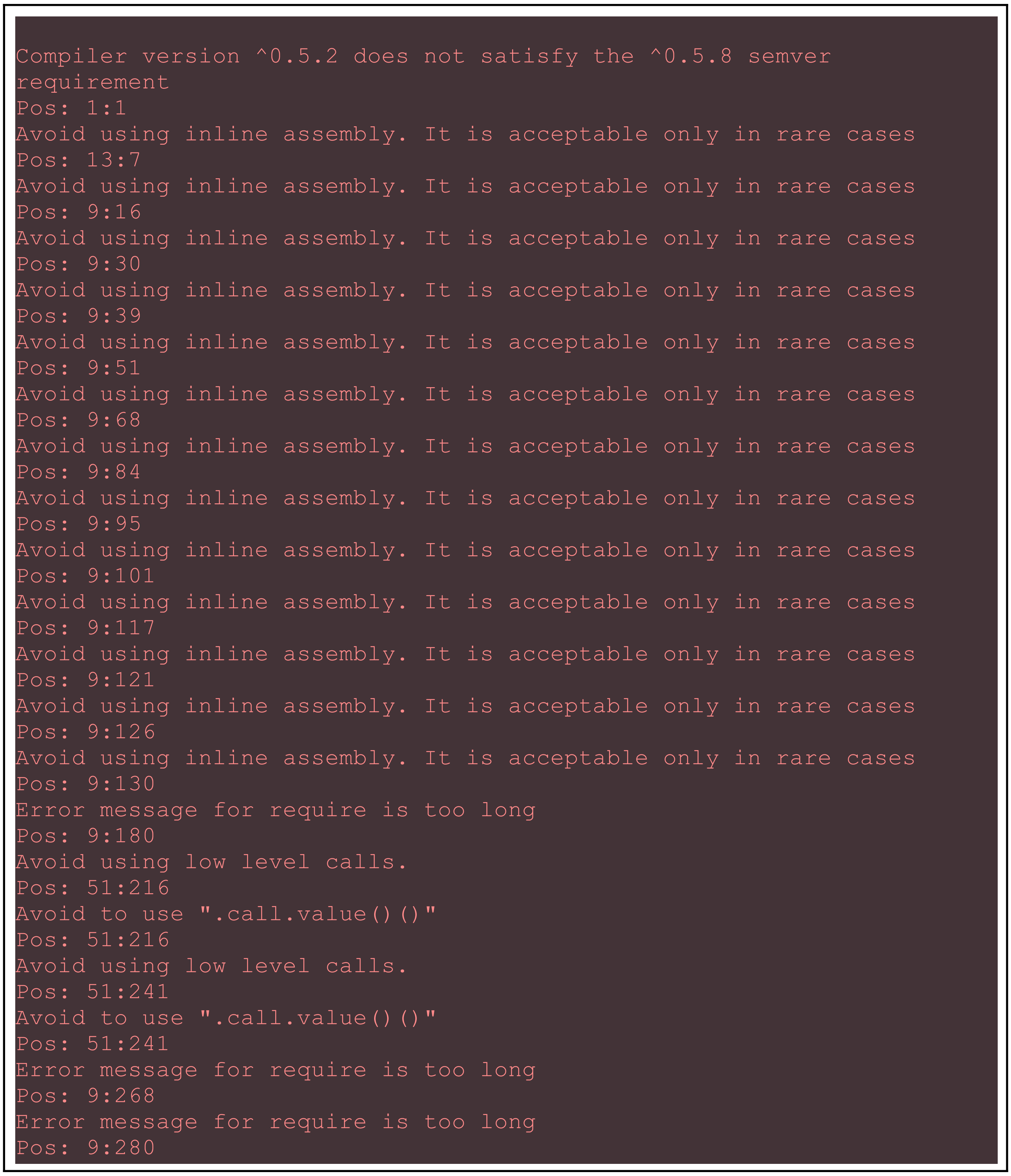
Data truncated:

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: 305:27:

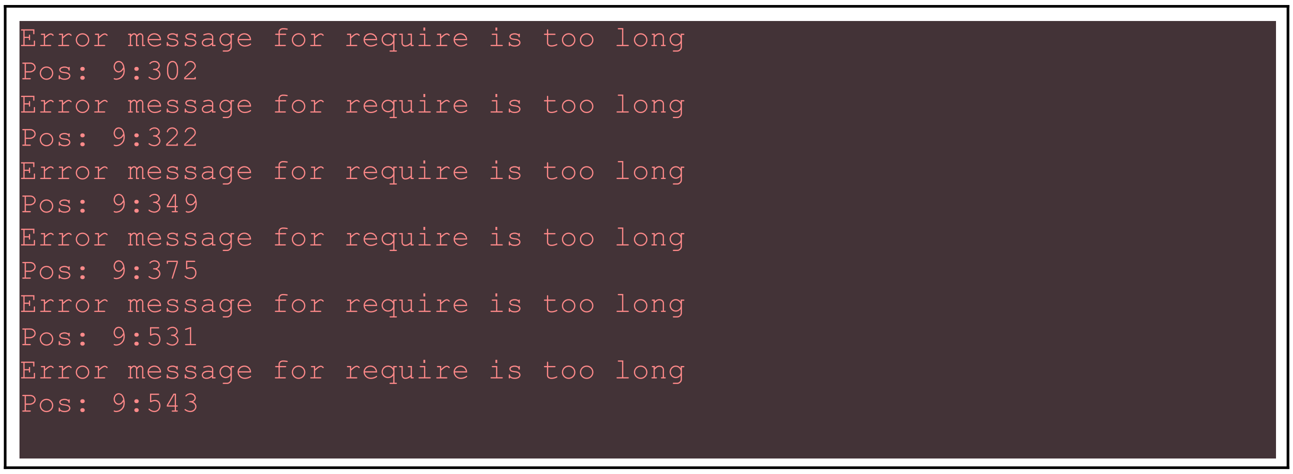
Solhint Linter

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.

Sand.sol



```
Compiler version ^0.5.2 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:7
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:16
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:30
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:39
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:51
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:68
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:84
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:95
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:101
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:117
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:121
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:126
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:130
Error message for require is too long
Pos: 9:180
Avoid using low level calls.
Pos: 51:216
Avoid to use ".call.value()()"
Pos: 51:216
Avoid using low level calls.
Pos: 51:241
Avoid to use ".call.value()()"
Pos: 51:241
Error message for require is too long
Pos: 9:268
Error message for require is too long
Pos: 9:280
```



```
Error message for require is too long
Pos: 9:302
Error message for require is too long
Pos: 9:322
Error message for require is too long
Pos: 9:349
Error message for require is too long
Pos: 9:375
Error message for require is too long
Pos: 9:531
Error message for require is too long
Pos: 9:543
```

Software analysis result:

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

EtherAuthority